# Semi-supervised Ensemble Learning

| | |
|---|---|
| Name: | Jesper E. van Engelen |
| Date: | July 10, 2018 |
| Supervisor: | Holger H. Hoos |
| Second reader: | Matthijs van Leeuwen |

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

# Abstract

Semi-supervised learning is a branch of machine learning that combines concepts from supervised learning and unsupervised learning to construct better learners. As is the case in supervised learning, most research in this area is focused on the classification task. The spectrum of semi-supervised classification methods is extensive, and a multitude of algorithms exists. We provide an extensive literature review of the field of semi-supervised learning, analyzing the assumptions that form the foundation of semi-supervised learning. Additionally, we propose a new taxonomy of semi-supervised classification methods.

In recent years, significant advances have been made in semi-supervised ensemble methods, combining multiple learners to form better predictions. However, like many supervised methods, their performance is highly dependent on the base classifiers used and on the classification problem at hand. To mitigate this problem, we propose using semi-supervised learning in conjunction with automated machine learning (AutoML): we construct a strong classifier ensemble using an AutoML system, and we improve it using a novel semi-supervised ensembling method. Our method, *single-step co-ensembling*, consistently improves the performance of state-of-the-art ensembles in multiclass problems, while maintaining the computational complexity of the supervised ensemble.

Furthermore, we evaluate the performance of *co-forest*, a popular extension of random forests to the semi-supervised setting. We provide confirmatory evidence that this method outperforms random forests when the number of trees is limited, as reported in previous research. However, as the number of trees grows, co-forest is consistently outperformed by its supervised counterpart. Moreover, we show that supervised random forests with more trees almost always outperform co-forest with fewer trees while being computationally less expensive.

# Preface

The research that forms the foundation of this thesis was conducted while I was a student member of the ADA Research Group of the Leiden Institute of Advanced Computer Science at Leiden University, under the supervision of Holger H. Hoos. The research project was originally aimed at applying automated machine learning to semi-supervised learning, but over the course of the work, its focus has slightly shifted towards general semi-supervised learning and semi-supervised ensemble learning.

Chapters 2 and 3, which form the core of this thesis, are based on yet unpublished work co-authored with Holger H. Hoos. I took primary responsibility for the research conducted for these articles, as well as for preparing the manuscripts.

One piece of research conducted while I was at the ADA Research Group, regarding a collaborative project with the Leiden Academic Centre for Drug Research, is not part of this thesis. My contribution to the project was the application of automated machine learning methods to drug discovery.

# Table of Contents

**Abstract** ii

**Preface** iii

**Table of Contents** iv

**Acknowledgements** vi

**1 Introduction and Background** 1
- 1.1 Machine learning . . . 1
- 1.2 Semi-supervised learning . . . 2
- 1.3 Assumptions of semi-supervised learning . . . 3
- 1.4 Contributions and thesis outline . . . 7

**2 A Survey on Semi-supervised Learning** 8
- 2.1 Introduction . . . 8
- 2.2 Taxonomy of semi-supervised classification methods . . . 9
  - 2.2.1 Inductive methods . . . 11
  - 2.2.2 Transductive methods . . . 12
- 2.3 Wrapper methods . . . 12
  - 2.3.1 Self-training . . . 13
  - 2.3.2 Co-training . . . 15
  - 2.3.3 Boosting . . . 19
- 2.4 Unsupervised preprocessing . . . 21
  - 2.4.1 Feature extraction . . . 22
  - 2.4.2 Cluster-then-label . . . 23
  - 2.4.3 Pre-training . . . 23
- 2.5 Intrinsically semi-supervised methods . . . 25
  - 2.5.1 Maximum-margin methods . . . 25
  - 2.5.2 Perturbation-based methods . . . 30
  - 2.5.3 Manifolds . . . 36
  - 2.5.4 Generative models . . . 40
- 2.6 Transductive methods . . . 43
  - 2.6.1 A general framework for graph-based methods . . . 44

2.6.2 Inference in graphs . . . 45
2.6.3 Graph construction . . . 52
2.6.4 Scalable transductive learning . . . 57
2.6.5 From transduction to induction . . . 58
2.6.6 Classification in network data . . . 58
2.7 Related areas . . . 59
2.7.1 Semi-supervised regression . . . 59
2.7.2 Semi-supervised clustering . . . 60
2.8 Conclusions and future perspectives . . . 60

**3 Automated Single-step Co-ensembling 63**
3.1 Introduction . . . 63
3.2 AutoML: auto-sklearn . . . 65
3.3 Co-ensembling . . . 66
3.3.1 Co-training assumptions . . . 67
3.3.2 General co-ensembling framework . . . 67
3.3.3 Single-step co-ensembling . . . 68
3.4 Experiments and results . . . 70
3.4.1 Experimental setup . . . 70
3.4.2 Supervised ensemble weighting . . . 72
3.4.3 Multi-step co-ensembling . . . 73
3.4.4 Single-step co-ensembling . . . 74
3.5 Conclusions . . . 79

**4 Semi-supervised Decision Trees 80**
4.1 Introduction . . . 80
4.2 Co-forest . . . 82
4.2.1 The co-forest algorithm . . . 83
4.2.2 Subsampling and bootstrapping . . . 84
4.3 Semi-supervised node splitting . . . 84
4.3.1 Supervised decision tree construction . . . 84
4.3.2 Regularization by density . . . 87
4.4 Experiments and results . . . 88
4.4.1 Co-forest . . . 89
4.4.2 Semi-supervised node splitting . . . 95
4.5 Conclusions . . . 96

**5 Conclusions and Future Work 98**

**Bibliography 100**

**A Co-ensembling Results Per Data Set 117**

**B Self-training in auto-sklearn 120**

**C Data Sets 124**

# Acknowledgements

Holger, my thesis supervisor, for providing me with the opportunity to pursue my ever-changing research ideas, but keeping them in check; for his attention to detail, his emphasis on rigor, and his benevolent criticism.

Matthijs, the second reader of this thesis, who provided highly valuable feedback on my thesis and on the semi-supervised learning survey in particular. His perspective was refreshing and educational.

Chuan, Marie, and Can, fellow members of the ADA Research Group at LIACS, whose company has been highly rewarding, both personally and academically, and with whom I've shared many fruitful conversations and discussions over the past nine months.

# Chapter 1

# Introduction and Background

In this chapter, we introduce the field of semi-supervised learning and cover the background knowledge that forms the basis for the rest of this thesis. We elaborate on the so-called semi-supervised learning assumptions, which are fundamental to the semi-supervised learning methods that exist today. Lastly, we outline our contributions and the structure of the rest of this thesis.

## 1.1 Machine learning

Humans and other animals are capable of learning from observations of their environment. This ability is an essential ingredient for *intelligence*, which encompasses a broader collection of capacities such as reasoning and understanding. Crucially, it allows us to adapt our actions to our environment. Since the advent of computers and computer programming languages in the early 1950s, the idea of emulating this ability in computers has received considerable interest. In particular, it has given rise to the question of teaching a computer to perform certain actions or gain certain knowledge without explicitly programming it to do so, which forms the basis of the field of *machine learning*.

Machine learning techniques aim to enable a computer to learn from examples. In other words, the computer uses observations to learn how to attain a certain goal. Such techniques are already broadly applied in society, including in medicine, retail, logistics, robotics, and many other areas [106]. Over time, a variety of machine learning techniques has been developed to tackle complex problems, such as natural language processing [92], recommender systems [145], and computer vision [168]. Machine learning techniques use observations to infer some underlying structure of the data, which they then use to extrapolate to new data or to otherwise determine the best course of action. Consequently, all machine learning methods require *training data*, which can be explicitly provided to the computer or can be observed by the computer itself.

In machine learning, a distinction has traditionally been made between two major tasks: supervised learning and unsupervised learning [20]. In *supervised learning*, one is presented with a set of data points, each of which consists of some input object $x$

and a corresponding output object $y$. The goal is, then, to construct a classifier or regressor that can estimate the output value for previously unseen input objects. In *unsupervised learning*, on the other hand, no specific output value is provided. Instead, one tries to infer some underlying structure from the input objects. For instance, in the unsupervised clustering task, the goal is to infer a mapping from the input objects to groups such that similar objects are mapped to the same group. Traditionally, the fields of supervised and unsupervised learning have been studied rather independently. In many cases, however, substantial benefit can be gained by combining principles from both fields.

## 1.2 Semi-supervised learning

*Semi-supervised learning* is a branch of machine learning concerned with combining concepts and methods from supervised and unsupervised learning [33, 216]. Typically, semi-supervised learning algorithms attempt to improve performance in one of these two tasks by utilizing information generally associated with the other task. For instance, when tackling a classification problem, available data points for which the label is unknown could be of use in the classification process. For clustering methods, on the other hand, the learner might benefit from the knowledge that certain data points belong to the same class.

As is the case in general machine learning research, a large majority of the research conducted in semi-supervised learning is focused on classification: the task of assigning labels to data points based on training data. In supervised classification, this training data consists of pairs of data points and labels; in semi-supervised classification, data points whose labels are unknown are available as well. The set of possible labels is known beforehand. A plethora of methods exist, each with their own characteristics, advantages, and disadvantages. It has been shown empirically that semi-supervised learning can greatly enhance classification performance in the presence of unlabeled data [216].

In traditional supervised learning problems, we are presented with a set of $l$ labeled data points $D_L = ((x_i, y_i))_{i=1}^{l}$. Each data point $(x_i, y_i)$ consists of an input object $x_i \in \mathcal{X}$ from some input space $\mathcal{X}$, and has an associated label $y_i$, where $y_i$ is real-valued in regression problems and categorical in classification problems. Based on these samples, usually called the *training data*, supervised learning methods attempt to infer a function that can successfully determine the label $y^*$ of some previously unseen sample $x^*$.

In many real-world classification problems, however, we also have access to a set of $u$ data points $D_U = (x_i)_{i=l+1}^{l+u}$ whose labels are unknown. For instance, the data points for which we want to make predictions, usually called the *test data*, are unlabeled by definition. Semi-supervised classification methods attempt to utilize unlabeled data points to construct a learner whose performance exceeds the performance of learners obtained when using only the labeled data. In the remainder of this thesis, we denote with $X_L$ and $X_U$ the set of input objects for the labeled and unlabeled samples, respectively.

There are many cases where unlabeled data can help in constructing a classifier. Consider, for example, the problem of document classification, where we wish to assign topics to a set of text documents (such as news articles). Assuming our documents are represented by the set of words that appear in it, one could train a simple supervised classifier that, for example, learns to recognize that documents containing the word "neutron" are usually about physics. This classifier might work well on documents containing terms that it has seen in the training data, but will inherently fail when a document does not contain predictive words that also occurred in the training set. For example, if we encounter a physics document about particle accelerators that does not contain the word "neutron", the classifier is unable to recognize it as a document concerning physics. This is where semi-supervised learning comes in. If we consider the unlabeled data, there might be documents that connect the word "neutron" to the phrase "particle accelerator". For instance, the word "neutron" would often occur in a document that also contains the word "quark". In its turn, the word "quark" would regularly co-occur with the phrase "particle accelerator", which guides the classifiers towards classifying these documents as revolving around physics as well, despite having never seen the phrase "particle accelerator" in the labeled data.

Figure 1.1 provides some further intuition towards the use of unlabeled data for classification. We consider an artificial classification problem with two classes. For both classes, 100 samples are drawn from a 2-dimensional Gaussian distribution with identical covariance matrices. The labeled data set is then constructed by taking one sample from each class. Any supervised learning algorithm will most likely obtain as the decision boundary the solid line, which is perpendicular to the line segment connecting the two labeled data points and intersects it in the middle. However, this is clearly quite far from the optimal decision boundary, which corresponds to the dashed line. As is clear from this figure, the clusters we can infer from the unlabeled data can help us considerably in placing our decision boundary: assuming that the data stems from two Gaussian distributions, a simple semi-supervised learning algorithm can infer a decision boundary rather close to the optimal decision boundary.

Although semi-supervised learning methods have shown promising performance in a broad variety of classification problems, they are not foolproof. When the assumptions made by the learning algorithm regarding the underlying data distribution do not hold, the introduction of unlabeled data can deteriorate performance. Such behaviour has been discussed in the literature [113, 160], and is likely underreported due to publication bias [216].

## 1.3 Assumptions of semi-supervised learning

A necessary condition of semi-supervised learning is that the underlying, marginal data distribution $p(x)$ over the input space contains information about the posterior distribution $p(y|x)$. If this is the case, one might be able to use unlabeled data to gain information about $p(x)$ and thereby about $p(y|x)$. If $p(x)$ contains no information about $p(y|x)$, however, the additional unlabeled data is inherently unable to improve the predictive performance of the classifier [216].

Fortunately, the aforementioned condition appears to be satisfied in most learning problems encountered in the real world, as is suggested by the successful application of semi-supervised learning methods in practice. However, the way $p(x)$ and $p(y|x)$ inter-

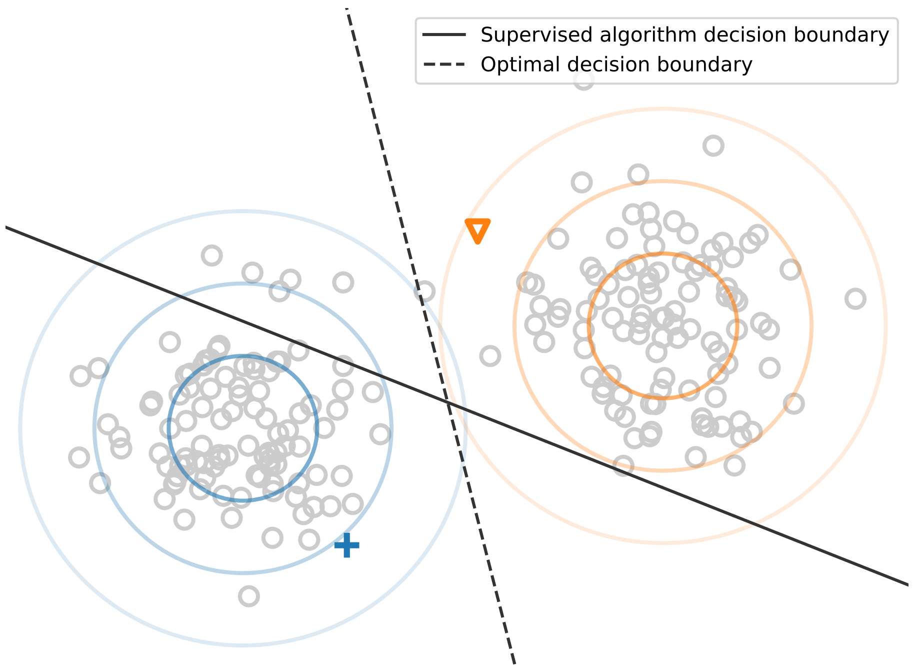


Figure 1.1: A basic example of binary classification in the presence of unlabeled data.

act is not always the same. This has given rise to the *semi-supervised learning assumptions*, which formalize the types of expected interaction [33]. The most widely recognized assumptions are the *smoothness assumption* (if two samples $x$ and $x'$ are close in input space, their labels $y$ and $y'$ should be the same), the *manifold assumption* (data points on the same low-dimensional manifold should have the same label), and the *low-density assumption* (the decision boundary should not pass through high-density areas in the input space). These assumptions are the foundation of most, if not all, semi-supervised learning algorithms, which generally depend on one or more of the assumptions either explicitly or implicitly. Throughout this thesis, we will elaborate on the underlying assumptions utilized by each specific learning algorithm. The assumptions are explained in more detail below.

### Smoothness assumption

The smoothness assumption states that, for two input points $x, x' \in \mathcal{X}$ that are close by in the input space, the corresponding labels $y, y'$ should be the same. This assumption is also commonly used in supervised learning, but has an extended benefit in the semi-supervised context: the smoothness assumption can be applied transitively to unlabeled data. For example, assume that a labeled data point $x_1 \in X_L$ and two unlabeled data points $x_2, x_3 \in X_U$ exist such that $x_1$ is close to $x_2$ and $x_2$ is close to $x_3$, but $x_1$ is not close to $x_3$. Then, through the smoothness assumption, we can still expect $x_3$ to have the same label as $x_1$, since the smoothness is transitively propagated through $x_2$.

### Low-density assumption

The low-density assumption implies that the decision boundary of a classifier should preferably pass through low-density regions in the input space. In other words, the decision boundary should not pass through high-density regions. Clearly, the low-density assumption is closely related to the smoothness assumption when a finite set of data points is available. Suppose a low-density area exists, i.e., an area $R \subset \mathcal{X}$ where $p(x)$ is low. Then very few observations are expected to be contained in $R$, and it is thus unlikely that any pair of similar data points in $R$ is observed. Thus, in placing the decision boundary in the low-density area, the smoothness assumption is not violated: it only concerns pairs of similar data points. For high-density areas, on the other hand, many samples can be expected, and the assumption implies that the decision boundary cannot pass through the region, since the predicted labels would then be dissimilar for similar samples.

### Manifold assumption

In machine learning problems where the data can be represented in Euclidean space, the observed data points in the high-dimensional input space $\mathbb{R}^d$ are usually concentrated along lower-dimensional substructures. These substructures are known as *manifolds*: topological spaces that are locally Euclidean. For instance, when we consider a 3-dimensional input space where all points lie on the surface of a sphere, the data can be said to lie on a 2-dimensional manifold. The manifold assumption in semi-supervised learning states that (a) data points lie on a lower-dimensional manifold and (b) data points lying on the same lower-dimensional manifold have the same label. Consequently, if we are able to determine on which manifold data points lie, we can infer their class assignments by simply considering which class the labeled data points on the same manifold have.

### A connection to clustering

In semi-supervised learning research, an additional assumption that is often included is the *cluster assumption*, which states that data points belonging to the same cluster belong to the same class [33]. We argue, however, that the previously mentioned assumptions and the cluster assumption are not coordinate but, rather, that the cluster assumption is a generalization of the other assumptions.

Consider an input space $\mathcal{X}$ with some observations $X \subset \mathcal{X}$, drawn from the distribution $p(x)$. A cluster, then, is a set of data points $C \subseteq X$ that are more similar to each other than to other data points in $X$, according to some concept of similarity [2]. Finding clusters corresponds to finding some mapping $X \mapsto \mathcal{Y}$, where $\mathcal{Y}$ is the set of possible cluster assignments. Each element in $\mathcal{Y}$ corresponds to a single cluster. Since we do not have direct access to $p(x)$ to determine a suitable clustering, we need to rely on some concept of similarity between data points in $\mathcal{X}$, according to which we can assign clusters to similar data points.

The concept of similarity we choose, often implicitly, is the only important decision in clustering methods. It uniquely defines the interaction between $p(x)$ and $p(y|x)$. Therefore, whether two points belong to the same cluster can be derived from their similarity to each other and to other points. From our perspective, the smoothness, low-density, and manifold assumptions boil down to different definitions of the similarity between points: the smoothness assumption states that points that are close by in the input space are similar; the low-density assumption states that points in the same high-density area are similar; and the manifold assumption states that points that lie on the same low-dimensional manifold are similar. As such, the semi-supervised learning assumptions can be seen as more specific instances of the cluster assumption: that similar points tend to belong to the same group.

One could even argue that the cluster assumption corresponds to the necessary condition for semi-supervised learning: that $p(x)$ carries information on $p(y|x)$. In fact, assuming the output space $\mathcal{Y}$ consists of all possible clusters, the necessary condition for semi-supervised learning to succeed can be seen to be the necessary condition for clustering to succeed. In other words: if the data points (both unlabeled and labeled) cannot be meaningfully clustered, it is impossible for a semi-supervised learning method to improve on a supervised learning method.

In Chapter 2, we will see how the semi-supervised learning assumptions relate to different semi-supervised learning methods. In particular, the class of intrinstically semi-supervised learning methods, which learn a model by directly incorporating unlabeled data into an objective function, mainly consists of methods that directly act on one of the assumptions.

## 1.4 Contributions and thesis outline

Like in supervised classification, there are very few generic classification algorithms that exhibit good performance on diverse data sets. In many cases, semi-supervised classification algorithms do not outperform their supervised counterparts [113, 160]. In this thesis, we study solutions to this problem and consider the generally relatively robust class of *ensemble methods*, which rely on multiple base learners to improve and stabilize performance [212]. In particular, we focus on semi-supervised *wrapper methods*, which iteratively introduce unlabeled data to an ensemble of supervised classifiers by labeling unlabeled data points with the predictions of the classifiers. Such methods have been extensively studied in the literature [179], but they are usually applied to weak ensembles of supervised classifiers (i.e., ensembles with relatively poor performance). Improving performance over weaker ensembles is both a less challenging and a less interesting task than improving performance over stronger ensembles: there is rarely a reason to use a semi-supervised extension of a weak ensemble when a strong supervised ensemble is also available.

Our contribution to the field of semi-supervised learning is threefold. Firstly, in Chapter 2, we present a comprehensive survey of the field of semi-supervised learning, and semi-supervised classification in particular. The survey constitutes an up-to-date review of the literature, and includes the proposal of a new taxonomy for semi-supervised classification methods.

Secondly, we study semi-supervised extensions to *automated machine learning* (AutoML) in Chapter 3. This branch of machine learning is concerned with automatically selecting and configuring classifiers or ensembles of classifiers for any particular machine learning problem [62, 100, 178]. We propose a semi-supervised wrapper method that can be applied to strong ensembles of diverse classifiers constructed by the AutoML system AUTO-SKLEARN [62]. Our method consists of a single additional training step, where unlabeled data is introduced to the ensemble of supervised base learners. We evaluate this method, which we call *single-step co-ensembling*, on a prominent benchmarking suite of diverse data sets. We achieve an average reduction in error rate of 7%, and performance is improved over the supervised ensemble in over 75% of the multiclass data sets we evaluate.

Thirdly, in Chapter 4, we evaluate the popular semi-supervised ensembling method *co-forest* [110], which extends random forests to the semi-supervised setting. We show that it works well when applied to a weak ensemble of base learners (a forest with few trees), but that it is outperformed by its supervised counterpart when using more base learners. Additionally, we propose a semi-supervised node splitting criterion for decision trees; we show that it improves performance of individual decision trees, but not of random forests.

Finally, in Chapter 5, we present our conclusions. We reflect on the research conducted for this thesis, and discuss our perspective on the field of semi-supervised learning in general.

# Chapter 2

# A Survey on Semi-supervised Learning

Before continuing to the proposal of a new semi-supervised ensemble learning method and the evaluation of a semi-supervised extension to random forests, we review the field of semi-supervised learning. We consider both earlier methods and more recent advances, and provide an in-depth explanation of a broad variety of learning methods. Furthermore, we introduce a new taxonomy for semi-supervised classification methods, shedding light on the goals, characteristics, and assumptions of different methods.

## 2.1 Introduction

The field of semi-supervised learning encompasses a broad spectrum of machine learning problems, including semi-supervised clustering, classification, and regression. A particularly large body of research exists in the field of semi-supervised classification, where a plethora of different methods has been proposed over the past few decades. This survey constitutes an up-to-date review and analysis of the field of semi-supervised learning, with a strong emphasis on semi-supervised classification. The most recent complete survey of the field was published by Zhu in 2005; it was most recently updated in 2008 [216]. The semi-supervised learning book by Chapelle et al. from 2006 [33] and Zhu and Goldberg's introductory semi-supervised learning book from 2009 [221] also provide good bases for studying earlier semi-supervised learning work. More recently, in 2014, Subramanya and Talukdar provided an overview of several graph-based techniques [167]; in 2015, Triguero et al. reviewed and analyzed pseudo-labeling techniques, a class of semi-supervised learning methods [179].

In this survey, we aim to provide the reader with a comprehensive overview of the current state of the field of semi-supervised learning. We present and thoroughly explain both early work in the field and recent advances. We present a new taxonomy for semi-supervised classification methods that captures both the underlying assumptions of each group of methods as well as the way they relate to existing supervised methods.

*This chapter is based on a survey article co-authored with H.H. Hoos. I took primary responsibility for the research conducted for this survey, as well as for writing the manuscript.*

In this, we provide a perspective on semi-supervised learning that allows for a more thorough understanding of different approaches and the connections between them.

Although we aim to provide a comprehensive overview of the field of semi-supervised classification, we cannot possibly cover every method in existence. Due to the expansiveness of the field, this would take away from the key insights which we wish to provide to the reader. Instead, we aim to cover the major developments in the field over the past twenty years.

The rest of this chapter is structured as follows. In Section 2.2, we present our semi-supervised learning taxonomy, which is used as the basis for the following sections. Inductive methods are covered in Sections 2.3 through Section 2.5. We first consider wrapper methods (Section 2.3), followed by unsupervised preprocessing (Section 2.4), and finally, we cover intrinsically semi-supervised methods (Section 2.5). The second part of the taxonomy, consisting of the transductive methods, is presented in Section 2.6. Semi-supervised regression and clustering are discussed in Section 2.7. Finally, in Section 2.8, we provide some prospects for the future of semi-supervised learning.

## 2.2 Taxonomy of semi-supervised classification methods

Over the past two decades, a broad variety of semi-supervised classification algorithms has been proposed. These methods differ in the semi-supervised learning assumptions they use, in the way they incorporate unlabeled data, and in the way they relate to supervised algorithms. In existing categorizations of semi-supervised learning methods, algorithms are usually grouped based on the assumptions on which they are premised. This provides insight into the similarities of learners on similar types of data, but it pushes several other important properties (e.g., whether the learner yields only label predictions or a classification model over the entire input space) to the background.

In this survey, we propose a new way to represent the spectrum of semi-supervised classification methods. Our taxonomy incorporates two crucial characteristics of semi-supervised learning methods. At the highest level, it distinguishes between *inductive* and *transductive* methods, which give rise to distinct optimization procedures: the former attempts to find a classification model, whereas the latter is only concerned with obtaining label predictions for the unlabeled data points at hand. At the second level, it considers the way the semi-supervised learning methods incorporate unlabeled data. This distinction gives rise to three difference classes of inductive methods, each of which is related to supervised classifiers in a different way.

The first distinction we make in our taxonomy, between inductive and transductive methods, is common in semi-supervised learning literature [33, 216, 221]. The former, like supervised learning methods, yield a classification model that can be used to predict the label of previously unseen data points. The latter do not yield such a model, but instead provide predictions directly. In other words, given a data set consisting of labeled and unlabeled data $X_L, X_U \subseteq \mathcal{X}$ with labels $\mathbf{y}_L \in \mathcal{Y}^l$ for the $l$ labeled data points, inductive methods yield a model $f : \mathcal{X} \mapsto \mathcal{Y}$, whereas transductive methods yield a set of predicted labels $\hat{\mathbf{y}}_U$ for the unlabeled data points $X_U$. Accordingly, the optimization problems posed in inductive methods optimize over prediction models,

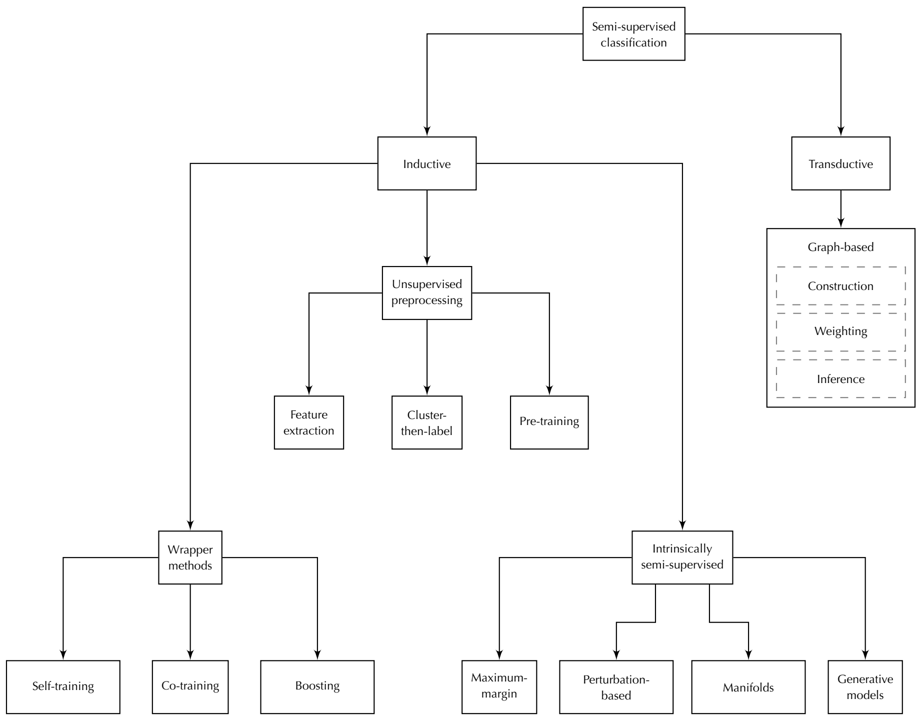


Figure 2.1: Visualization of the semi-supervised classification taxonomy. Each leaf in the taxonomy corresponds to a specific type of approach to incorporating unlabeled data into classification methods. In the leaf corresponding to transductive, graph-based methods, the dashed subitems correspond to distinct phases of the graph-based classification process, each of which has a multitude of variations.

whereas the optimization problems posed in transductive methods optimize directly over the predictions $\hat{\mathbf{y}}_U$.

Inductive methods, which generally extend supervised algorithms to include unlabeled data, are further split in our taxonomy based on the way they incorporate unlabeled data: either in a preprocessing step, directly inside the objective function, or via a pseudo-labeling step. The transductive methods are in all cases graph-based; we group these based on the choices made in separate stages of the learning process. The taxonomy is visualized in Figure 2.1. In the remainder of this section, we will elaborate on the grouping of semi-supervised learning methods represented in the taxonomy. Furthermore, the taxonomy forms the basis for our discussion of semi-supervised learning methods in the remaining sections of this chapter.

### 2.2.1 Inductive methods

Inductive methods aim to construct a classification model that can generate predictions for any sample in the input space. The design of the model can depend on unlabeled data, but the predictions for multiple new, previously unseen examples are independent of each other once the classifier has been trained. This corresponds to the objective in supervised learning methods: a model is built in the training phase and can then be used for predicting the labels of new data points.

#### Wrapper methods

A simple approach to extending existing, supervised algorithms to the semi-supervised setting is to first train classifiers on labeled data, and to then use the predictions of the resulting classifiers to generate additional labeled data. The classifiers can then be retrained on this *pseudo-labeled* data in addition to the existing labeled data. Such methods are known as *wrapper methods*: the unlabeled data is pseudo-labeled by a wrapper component, and a purely supervised learning algorithm, unaware of the distinction between originally labeled and pseudo-labeled data, constructs the final inductive classifier. This reveals a key property of wrapper methods: most of them can be applied to any given supervised base learner, allowing unlabeled data to be introduced in a straightforward manner. They form the first part of the inductive side of the taxonomy, and are covered in Section 2.3.

#### Unsupervised preprocessing

Secondly, we consider unsupervised preprocessing methods, which either extract useful features from the unlabeled data, pre-cluster the data, or determine the initial parameters of a supervised learning model in an unsupervised manner. Like wrapper methods, they can be used with any supervised classifier. However, contrary to wrapper methods, the supervised classifier is only provided with originally labeled data points. These methods are covered in Section 2.4.

**Intrinsically semi-supervised methods**

The last class of inductive methods we consider directly incorporate unlabeled data into the objective function or optimization procedure of the learning method. Many of these methods are direct extensions of supervised learning methods to the semi-supervised setting: they extend the objective function of the supervised classifier to include unlabeled data. Semi-supervised support vector machines (S3VMs), for example, extend supervised SVMs by maximizing the margin on the unlabeled data in addition to the labeled data. There are semi-supervised analogies for, among others, SVMs, Gaussian processes, and neural networks, and we describe these in Section 2.5.

### 2.2.2 Transductive methods

Unlike inductive methods, transductive methods do not construct a classification model over the entire input space. Instead, their predictive power is limited to exactly those samples that it encounters during the training phase. Therefore, no independent training and prediction phases exist in transductive methods. Since supervised learning methods are by definition not supplied with unlabeled data until the testing phase, no clear analogies of transductive algorithms exist in supervised learning.

Since no model of the input space exists in transductive learners, information has to be propagated via direct connections between data points. This observation naturally gives rise to a graph-based approach to transductive methods: if a graph can be defined that connects similar data points, information can then be propagated along the edges of the graph. In practice, all transductive methods we discuss are either explicitly graph-based or can implicitly be understood as such. Although graph-based methods occupy the entire space of transductive methods in our taxonomy, inductive graph-based methods exist as well. Such approaches generally find their motivation in the manifold assumption; we cover them in Section 2.5.3.

Transductive graph-based methods generally consist of three steps: graph construction, graph weighting, and inference. In the first step, the samples $X$ are used to construct a graph where each node represents a data point and pairwise similar data points are connected by an edge. In the second step, these edges are weighted to represent the extent of the pairwise similarity between data points. In the third step, the graph is used to assign labels to the unlabeled data points. Different methods for carrying out these three steps are discussed in detail in Section 2.6.

## 2.3 Wrapper methods

Wrapper methods are some of the oldest and most well-known forms of semi-supervised learning [216]. They utilize one or more supervised base learners and iteratively train them with data consisting of the original labeled data and previously unlabeled data with predictions from earlier iterations of the learners. The latter is commonly referred to as *pseudo-labeled data*. The procedure usually consists of two alternating steps of *training* and *pseudo-labeling*. In the training step, one or more supervised classifiers are trained on the labeled data and, possibly, pseudo-labeled data. In the pseudo-labeling step, the resulting classifiers are used to attach labels to the previously unlabeled samples, pseudo-labeling the samples for which the learners were most confident of their predictions.

A significant advantage of wrapper methods is that they can be used with virtually any supervised base learner. The supervised base learner can be entirely unaware of the wrapper method, which simply passes pseudo-labeled samples to the base learner as if they were regular labeled samples. Although some wrapper methods require the base learner to provide probabilistic predictions, many wrapper methods relying on multiple base learners do not.

A comprehensive survey of wrapper methods was published recently by Triguero et al. [179]. In addition to providing an overview of such methods, they also propose a categorization and taxonomy of wrapper methods. Their taxonomy is based on (1) how many classifiers are used, (2) whether different types of classifiers are used, and (3) whether they use single-view or multi-view data (i.e., whether the data is split into multiple feature subets). The taxonomy provides valuable insight into the space of wrapper methods.

We present a less complex taxonomy, focused on the three relatively independent types of wrapper methods that have been studied in the literature. Firstly, we consider *self-training*, which uses one supervised classifier that is iteratively re-trained on its own most confident predictions. Secondly, we consider *co-training*, an extension of self-training to multiple classifiers which are iteratively re-trained on each other's most confident predictions. The classifiers are supposed to be sufficiently diverse, which is usually achieved by operating on different subsets of the samples or features. Lastly, we consider *pseudo-labeled boosting methods*. Like traditional boosting methods, they build a classifier ensemble by constructing individual classifiers sequentially. Each individual classifier is trained on both labeled data and the most confident predictions of the previous classifiers on unlabeled data.

### 2.3.1 Self-training

Self-training methods (sometimes also called "self-learning" methods) are the most basic of pseudo-labeling approaches [179]. They consist of a single supervised classifier that is iteratively trained on both labeled data and pseudo-labeled data (generated in previous iterations of the algorithm). Let $\tilde{D}_L$ denote the pseudo-labeled data, and let $\tilde{X}_U$ denote the samples that have not been assigned pseudo-labels yet. We initialize $\tilde{D}_L = \emptyset$ and $\tilde{X}_U = X_U$. In each iteration of the self-training algorithm, a supervised classifier is trained on $D_L \cup \tilde{D}_L$. The trained classifier is then used to form predictions for all remaining unlabeled samples $\tilde{X}_U$. The samples on which the classifier is most confident are then removed from $\tilde{X}_U$ and, along with their predicted labels, added to $\tilde{D}_L$. The algorithm is usually iterated until no more unlabeled samples remain to be pseudo-labeled, but other stopping criteria have also been proposed (see, e.g., [149, 179]). A schematic representation of the self-training algorithm is provided in Figure 2.2.

The self-training method was first proposed by Yarowsky as an approach to word sense disambiguation in text documents, predicting the meaning of words based on their context [205]. Rosenberg et al. apply self-training to object detection problems,

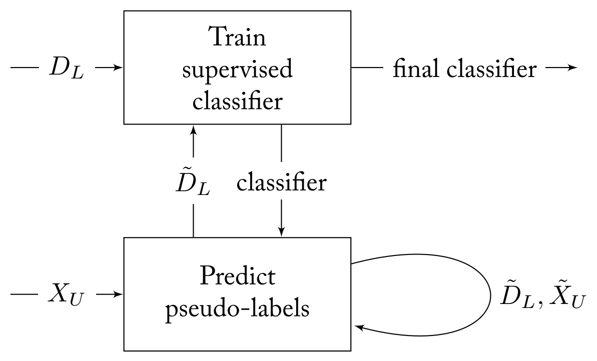


Figure 2.2: Schematic visualization of self-training methods.

and show improved performance over a state-of-the-art (at that time) object detection model [149]. Dópido et al. develop a self-training approach for hyperspectral image classification [58]. They use domain knowledge to select a set of candidate unlabeled samples, and pseudo-label the most informative of these samples with the predictions of the trained classifier. The latter procedure is inspired by active learning [154]: one wants to obtain the labels of the samples that are expected to contribute most to a change in the decision boundary.

Because it selects samples to pseudo-label based on the prediction confidence, self-training is highly dependent on the quality of the probabilistic predictions. In particular, the ranking of prediction probabilities for the unlabeled samples should reflect the true confidence ranking. This means some base learners are inherently better-suited for self-training than others.

Decision trees are a prime example of learning algorithms that, without any modifications or pruning, yield poor prediction probability estimates. This can be largely attributed to two causes: (1) decision trees assign equal probabilities to all instances in a leaf, and (2) leaves in decision tree tend to have few instances. Tanha et al. attempt to improve these probability estimates in a self-training approach to decision trees and ensembles of decision trees [176] in two distinct ways: firstly, they apply several existing methods, such as grafting and Laplace correction, to directly improve prediction probability estimates. Secondly, they use a local distance-based measure to determine the confidence ranking between instances: the prediction confidence of an unlabeled sample is based on the absolute difference in the Mahalanobis distances between the point and the labeled data from each class. They show significant improvements in performance of both decision trees and ensembles of decision trees (random forests) using this method [176].

Leistner et al. also utilize self-training to improve random forests [108]. Instead of labeling the unlabeled samples $\mathbf{x} \in X_U$ with the label predicted to be most likely, they pseudo-label each unlabeled sample independently for each tree according to the estimated posterior distribution $p(y|\mathbf{x})$. Furthermore, they propose a stopping criterion based on the out-of-bag-error: when the out-of-bag-error (which is an unbiased estimate of the generalization error) increases, training is stopped.

If proper probabilistic predictions are available, one can also consider using the probabilities directly. In this case, the self-training approach is iterative and not incremental, as label probabilities for unlabeled data points are re-estimated in each step. In that case, the approach becomes similar to *expectation-maximization* (EM, [53]). It has been particularly studied in the context of *naïve Bayes* classifiers, which are inherently probabilistic [129, 130, 131]. Wu et al. recently applied semi-supervised EM with a naïve Bayes classifier to the problem of detecting fake product reviews on e-commerce websites [199].

Limited studies regarding the theoretical analysis on self-training algorithms exist. Haffari and Sarkar perform theoretical analysis on several variants of self-training, and show a connection between self-training methods and graph-based methods [78]. Culp and Michailidis analyze the convergence properties of a variant of self-training with several base learners, and consider the connection to graph-based methods as well [45].

When a classifier is trained iteratively, optimizing the objective function over individual samples or subsets of samples, an iterative pseudo-labeling approach similar to self-training can be applied. Instead of training the entire classifier, forming predictions on the unlabeled data, and then re-training the classifier, one can also pseudo-label data points throughout the training process. This approach is applied to neural networks by Lee, who proposes the *pseudo-label* approach [107]. Since the pseudo-labels predicted in the earlier training stages are generally less reliable, the weight of the pseudo-labeled samples is increased over time. The *pseudo-label* approach exhibits clear similarities to self-training, but differs in the sense that the classifier is not re-trained after each pseudo-labeling step: instead, it is finetuned with new pseudo-labeled data, and therefore technically deviates from the *wrapper method* paradigm.

### 2.3.2 Co-training

Co-training is an extension of self-training to multiple supervised classifiers. In co-training, two or more supervised classifiers are iteratively trained on the labeled data, adding their most confident predictions to the labeled data set of the other supervised classifiers in each iteration. For co-training to succeed, it is important that the base learners make uncorrelated errors on the unlabeled data. If they do not, their potential to provide each other with useful information is limited. In the literature, this condition is usually referred to as the *diversity* criterion [215].

Zhou and Li provide a survey of semi-supervised learning methods relying on multiple base learners, which they jointly name *disagreement-based methods* [215]. The name stems from the observation that co-training approaches exploit disagreements between multiple learners: they exchange information through unlabeled samples for which different learners predict different labels.

To promote classifier diversity, earlier co-training approaches mainly relied on the existence of multiple different *views* of the data, which generally correspond to distinct subsets of the feature set. For instance, when handling video data, the data can be naturally decomposed into visual data and audio data. Such co-training methods belong to the space of multi-view learning methods, which include a large body of supervised learning algorithms as well. Xu et al. provide a comprehensive survey of such methods [200]. We cover multi-view co-training methods in Section 2.3.2. In many real-world problem scenarios, no distinct views of the data are known a priori. Single-view co-training methods address this problem either by splitting the data into different views automatically, or by promoting diversity among classifiers using different learning algorithms. We cover these methods in Section 2.3.2. We shortly discuss *co-regularization* methods, in which multiple classifiers are combined into a single objective function, in Section 2.3.2.

### Multi-view co-training

The basic form of co-training was proposed by Blum and Mitchell [23]. In their seminal paper, they propose to construct two classifiers which are trained on two distinct views. After each training step, the most confident predictions of each view are added to the set of labeled samples of the other view. Blum and Mitchell apply the co-training algorithm to web page classification for university web pages, where the two views of the data are the web page text and the anchor text in links to the web page from external sources. This algorithm and variants thereof have been successfully applied in several fields, most notably natural language processing [97, 125, 186].

The original co-training algorithm by Blum and Mitchell relies on two main assumptions to succeed: (1) each individual subset of features should be sufficient to form good predictions on the data set, and (2) the subsets of features should be conditionally independent given the class label. The first assumption can be understood trivially: if one of the two feature subsets is insufficient to form good predictions, the corresponding learner can never contribute positively to the combined classifier's performance. The second assumption states that knowledge of a sample's features from either subset does not provide knowledge about the sample's features from the other subset, given the sample's label. This assumption is related to the diversity criterion: if the feature subsets are conditionally independent given the class label, the predictions of the individual classifiers are unlikely to be strongly correlated. Formally, for any sample $\mathbf{x}_i = \mathbf{x}_i^{(1)} \times \mathbf{x}_i^{(2)}$, decomposed into $\mathbf{x}_i^{(1)}$ and $\mathbf{x}_i^{(2)}$ for the first and second feature subset, respectively, the conditional independence assumption amounts to $p(\mathbf{x}_i^{(1)}|\mathbf{x}_i^{(2)}, y_i) = p(\mathbf{x}_i^{(1)}|y_i)$. Dasgupta et al. show that, under the aforementioned assumptions, the individual learners have low generalization error if they agree on their predictions for unlabeled data [48].

In practice, the second assumption is generally not satisfied: even if a natural split of features exists, such as in the experimental setup used by Blum and Mitchell, it is unlikely that information contained in one view provides no information about the other view when conditioned on the class label [59]. Considering the university web page classification example, the anchor text of a link to a web page can indeed be expected to contain clues towards the content of the web page, even if it is known that the web page is classified as a faculty member's home page. For example, if the link's anchor text is "Dean of the Engineering Faculty", one is more likely to find information about the dean of the engineering faculty than about any other person in the web page text. Thus, several alternatives to this assumption have been considered.

Abney shows that a weak independence assumption is sufficient for successful co-training [1]. Balcan et al. further relax the conditional independence assumption, showing that a much weaker assumption, which they name the "expansion" assumption, is sufficient and to some extent necessary [7]. The expansion assumption does not assume conditional independence between the feature sets, but rather assumes that the two views are not highly correlated, and that individual classifiers never confidently make wrong predictions.

Du et al. study empirical methods to determine to what degree the sufficiency and independence assumptions hold [59]. They propose several methods for automatically splitting the feature set into two views, and show that the resulting empirical independence and sufficiency is positively correlated with the performance of the co-trained algorithm, indicating that feature splits optimizing sufficiency and independence lead to good learning algorithms.

### Single-view co-training

As Du et al. show, co-training can be successful even when no natural split in the features is known a priori [59]. This observation is echoed throughout the literature on co-training, and many different approaches to applying co-training in the single-view setting exist.

Chen et al. attempt to alleviate the need for pre-defined disjoint feature sets by automatically splitting the feature set in each co-training iteration [37]. They formulate a single optimization problem closely related to co-training, incorporating both the requirement that the feature sets should be disjoint and the expansion property from Balcan et al. [7]. They show promising results of this approach on a partially synthetic data set, where multiple views of each sample are automatically generated. Wang and Zhou reason about sufficient and necessary conditions for co-training to succeed, approaching co-training from a graph-based perspective, where label propagation is alternately applied to each learner [193]. A downside of this approach is that, although inspired by co-training, it cannot be applied to any unmodified supervised learning algorithm: the co-training style operations are embedded in the objective function, which is directly optimized.

Several suggestions have been made to split single-view data sets into multiple views. For instance, Wang et al. suggest to generate $k$ random projections of the data, and use these as the views for $k$ different classifiers [191]. Zhang and Zheng propose to project the data onto a lower-dimensional subspace using principal component analysis. They then construct the pseudo-views by greedily selecting the transformed features corresponding to the maximal variance [209]. Yaslan and Cataltepe do not transform the data to a different basis, but select the features for each view iteratively, preferring features with high mutual information with respect to the sample labels [206].

Further approaches to apply co-training style algorithms on data sets where no explicit views are available focus on other ways of introducing diversity among the classifiers. For example, one can use different hyperparameters for the supervised algorithms [192, 213], or use different supervised algorithms altogether [67, 201, 211]. Wang and Zhou provide both theoretical and empirical analysis on why co-training can work in single-view settings [192]. They show that the diversity between the learners is positively correlated with their joint performance. Zhou and Li propose *tri-training*, where three classifiers are alternately trained [214]. When two of the three classifiers agree on their prediction of a sample, the sample is passed to the other classifier as a labeled sample. Crucially, tri-training does not rely on probabilistic predictions of individual classifiers, and can thus be applied to a much broader range of supervised learning algorithms.

The authors of the tri-training approach propose to extend it to more than three learners, applying the paradigm to ensembles of decision trees (random forests) [110]. The approach, known as *co-forest*, starts by training the decision trees independently on all labeled data. Then, in each iteration, each classifier receives pseudo-labeled data based on the joint prediction of all other classifiers on the unlabeled data: if the fraction of classifiers predicting a class $\hat{y}_i$ for an unlabeled sample $\mathbf{x}_i$ exceeds a certain threshold, the pseudo-labeled sample $(\mathbf{x}_i, y_i)$ is passed to the classifier. The decision trees are then all re-trained on their labeled and pseudo-labeled data. In the next iteration, all previously pseudo-labeled data is treated as unlabeled again. We note that, as the number of trees tends to infinity, the algorithm transforms into a form of self-training.

Co-forest includes a mechanism for reducing the influence of possibly mislabeled samples in the pseudo-labeling step by weighting the newly labeled samples based on the prediction confidence. Furthermore, it employs a PAC-based approach to determine whether data should be pseudo-labeled at all for a tree in a particular iteration. Deng and Guo attempt to further prevent the influence of possibly mislabeled samples by removing "suspicious" pseudo-labelings [55]. After the each pseudo-labeling step, the prediction for each pseudo-labeled sample $\mathbf{x}_i$ is compared to the (pseudo-)labels of its $k$ nearest neighbors (both labeled and pseudo-labeled). If they are not the same, the pseudo-label is removed from the sample.

In existing literature concerning co-forest, the size of the forest has always limited to six trees. It has been empirically shown that, in supervised random forests, performance can substantially improve as the number of trees grows [134]. In Chapter 4, we investigate the effects of varying the number of trees in the co-forest method.

#### Co-regularization

By passing information in the form of pseudo-labeled data between classifiers, co-training methods decrease disagreement between classifiers. Furthermore, the implicit objective of co-training is to minimize the error rate of the ensemble of classifiers. Sindhwani et al. propose to make these properties explicit in a single objective function [158, 159]. They propose *co-regularization*, a regularization framework in which both the ensemble quality and the disagreement between base learners are simultaneously optimized. In the framework, the objective function consists of both a loss term that penalizes incorrect predictions of the ensemble, and a loss term that directly penalizes different predictions of the base classifiers. In this framework, Yu et al. propose *Bayesian co-training*, which can handle per-view noise [207]. They propose a graphical model for combining data from multiple views, and develop a kernel-based method for co-regularization. This model is extended to handle different noise levels per sample by Christoudias et al. [39].

Co-training can be regarded as a greedy optimization strategy for the co-regularization objective. The ensemble accuracy is maximized by minimizing the loss functions on both hypotheses alternately; the disagreement between classifiers is minimized by propagating predictions from either classifier to the other classifiers as if they were the ground truth. We do note, however, that the general co-regularization objective does not necessitate a solution via a wrapper method; many co-regularization approaches do not utilize wrapper methods at all (see, e.g., [159, 207]).

### 2.3.3 Boosting

Ensemble models are classifiers consisting of multiple base classifiers, which are trained and then used to form combined predictions (see, e.g., [212]). The simplest form of ensemble learning trains $k$ classifiers independently; predictions can then be obtained by aggregating the predictions of these base learners. Besides this trivial approach, two main branches of supervised ensemble learning exist: *bagging* and *boosting* (see, e.g., [212]). In bagging methods, each base learner is provided with a data set consisting of $n$ samples, which are sampled with replacement from the original data set (bootstrapping). The base learners are trained independently. When training is completed, their outputs are aggregated to form the prediction of the combined classifier. In boosting methods, on the other hand, each base learner is dependent on the previous base learners: it is provided with the full data set, but with weights applied to the samples. The weight of a sample is based on the performance of the previous base learners on the sample: samples that were incorrectly classified get assigned larger weights. For the final prediction, the predictions of the classifiers are combined linearly.

Base learners in bagging methods are trained independently by definition. Therefore, the only truly semi-supervised bagging method would apply self-training to individual base learners. Co-training, however, can be seen to be closely related to bagging methods: the only way classifiers interact is by the exchange of pseudo-labeled data; other than that, the classifiers can be trained independently and simultaneously. However, most co-training methods do not use bootstrapping, a defining characteristic of bagging methods.

In boosting methods, on the other hand, there is an inherent dependency between base learners: the classifiers are trained sequentially, and previous base learners influence the construction of later base learners. Such methods can be readily extended to the semi-supervised setting by introducing pseudo-labeled data after each learning step. This gives rise to the class of semi-supervised boosting methods.

Semi-supervised boosting methods have been studied intensively over the past two decades. The success achieved by supervised boosting methods such as *AdaBoost* [63], gradient boosting, and *XGBoost* [38], provides ample motivation to bring boosting to the semi-supervised setting. Furthermore, the pseudo-labeling approach of self-training and co-training can trivially be extended to boosting methods. We outline several semi-supervised boosting methods. Most of these methods are indeed wrapper methods, using fully supervised base learners. The first algorithm we discuss below, *SSMBoost* [73], forms the only exception to this rule: it requires semi-supervised base learners. We consider it for its important contribution as the foundation for semi-supervised boosting methods.

Boosting methods construct a weighted ensemble of classifiers $h_t$ in a greedy fashion. Let $F_{T-1}(\mathbf{x}) = \sum_{t=1}^{T-1} \alpha_t \cdot h_t(\mathbf{x})$ denote the ensemble of classifiers $h_t$ with weight $\alpha_t$. Furthermore, let $\ell(\hat{y}, y)$ denote the loss function for predicting label $\hat{y}$ for a sample with true label $y$. At each iteration of the algorithm, an additional classifier $h_T$ is added to the ensemble with a certain weight $\alpha_T$ such that the cost function

$$\mathcal{L}(F_T) = \sum_{i=1}^{n} \ell(F_T(\mathbf{x}_i), y_i) \tag{2.1}$$

$$= \sum_{i=1}^{n} \ell(F_{T-1}(\mathbf{x}_i) + \alpha_T \cdot h_T(\mathbf{x}_i), y_i) \tag{2.2}$$

is minimized. Note that, at time $T$, the ensemble $F_{T-1}$ is fixed. With particular choices of loss functions, such as $\ell(\hat{y}, y) = \exp(-\hat{y} \cdot y)$, the optimization problem yields a weighted classification problem for determining $h_T$, and allows us to express the optimal $\alpha_T$ in terms of the loss of $h_T$ on the training data.

### SSMBoost

The first effort towards semi-supervised boosting methods was made by Grandvalet et al., who extend AdaBoost to the semi-supervised setting. The initial concept of the semi-supervised boosting algorithm is proposed in [73], and it is extended and motivated from the perspective of gradient boosting in [46]. To achieve this, a loss function is defined for unlabeled data based on the predictions of the current ensemble and on the predictions of the base learner under construction. Experiments are conducted with multiple loss functions; the authors report the strongest results using the expected loss of the new, combined classifier. The weighted error $\epsilon_t$ for base classifier $h_t$ is thus adapted to include the unlabeled data points, causing the weight term $\alpha_t$ to depend on the unlabeled data as well.

Crucially, SSMBoost does not assign pseudo-labels to the unlabeled data points. As a result, it requires semi-supervised base learners to make use of the unlabeled data. This distinguishes the algorithm from other semi-supervised boosting algorithms: SSMBoost is not a wrapper method, but is intrinsically semi-supervised. It is included here because it forms the foundation for all other forms of semi-supervised boosting algorithms, which require no semi-supervised base learners.

### ASSEMBLE

The *ASSEMBLE* algorithm, short for *Adaptive Supervised Ensemble*, alleviates the need for semi-supervised base learners [17]. Instead, it pseudo-labels the unlabeled data points after each iteration, and uses these pseudo-labeled data points in the construction of the next classifier. The authors show that this procedure corresponds to maximizing the classification margin in function space.

Since pseudo-labels are used in ASSEMBLE, it is not trivial to decide which unlabeled data points to pass to the next base learner. Bennet et al. propose to use bootstrapping, sampling, with replacement, $l$ data points from the $l + u$ labeled and unlabeled data points.

### SemiBoost

The latest iteration of semi-supervised boosting algorithms is *SemiBoost*, proposed by Mallapragada et al. [123]. It solves the problem of selecting samples to use in the base learners by relying on the manifold assumption. In a method similar to label propagation, each unlabeled sample is assigned a pseudo-label, and the corresponding prediction confidence is calculated. Then, a weighted sample, based on the prediction confidences, is taken from the pseudo-labeled samples, and added to the set of labeled samples for training the next base learner. SemiBoost is successfully applied to object tracking in videos by Grabner et al. [71].

SemiBoost uses the normal boosting classification model, expressing the final label prediction as a linear combination of the predictions of the individual learners. Its loss function, however, is highly dissimilar from previously described semi-supervised boosting methods. Mallapragada et al. argue that a successful labeling of the test data should conform to the following three requirements. Firstly, the predicted labels of the unlabeled data should be consistent for unlabeled samples that are close together. Secondly, the predicted labels of the unlabeled data should be consistent with the labels of closeby labeled data points. And, thirdly, the predicted labels for the labeled samples should correspond to their true labels. This is posed as a constrained optimization problem, where the first two requirements are captured by the objective function, and the latter requirement is imposed as a constraint. In other words, the SemiBoost algorithm uses boosting to solve the optimization problem

$$\begin{aligned} \underset{F_T}{\text{minimize}} \quad & \mathcal{L}_L(\hat{\mathbf{y}}, A) + \lambda \cdot \mathcal{L}_U(\hat{\mathbf{y}}, A) \\ \text{subject to} \quad & \hat{y}_i = y_i, \qquad\qquad i = 1, \ldots, l, \end{aligned} \tag{2.3}$$

where $\mathcal{L}_U$ and $\mathcal{L}_L$ are the loss functions expressing the inconsistency across the unlabeled and the combined labeled and unlabeled data, respectively, and $\lambda \in \mathbb{R}$ is a constant defining the weight of the inconsistencies. $A$ is an $n \times n$ symmetric matrix defining the pairwise similarities between samples.

The formulation in Equation 2.3 is very similar to the cost functions encountered in *graph-based methods* (see Section 2.5.3 and Section 2.6). The objective function promotes samples on the same manifold to receive the same label prediction. In graph-based methods, however, one usually does not distinguish between labeled-unlabeled and unlabeled-unlabeled pairs.

### Other semi-supervised boosting methods

The three methods discussed so far form the core of semi-supervised boosting research. Aside from these methods, some further research has been conducted on semi-supervised boosting. Chen and Wang propose *RegBoost*, which, like SemiBoost, includes local label consistency in its objective function [36]. In RegBoost, this term is also dependent on the estimated local density of the marginal distribution $p(x)$. Several attempts have been made to extend the label consistency regularization to the multiclass setting [175, 181].

## 2.4 Unsupervised preprocessing

The second category of inductive methods we discuss is *unsupervised preprocessing* methods, which, unlike wrapper methods and intrinsically semi-supervised methods, use the unlabeled data and labeled data in two separate stages. Typically, the unsupervised stage comprises either the automated extraction or transformation of sample features from the unlabeled data (*feature extraction*), the unsupervised clustering of the data (*label-then-cluster*), or the initialization of the parameters of the learning model (*pre-training*).

### 2.4.1 Feature extraction

Since the early days of machine learning, feature extraction has played an important role in classifier construction. Feature extraction methods attempt to find a transformation of the input samples such that the performance of the classifier improves or such that the construction is computationally more efficient. Feature extraction is an expansive research topic by itself; several books and surveys exist regarding the topic. We focus on a small number of particularly important techniques, and refer the reader to the existing literature on feature extraction methods for more information (see, e.g., [77, 156]).

Many feature extraction methods operate without supervision, i.e., without taking into account sample labels. *Principal component analysis*, for example, transforms the input samples to a different basis such that they are linearly uncorrelated and orders the principal components based on their variance [196]. Other traditional feature extraction algorithms operate on the labeled data, and try to extract features with high predictive power (see, e.g., [77]).

Recent semi-supervised feature extraction methods have mainly been focused on finding latent representations of input samples using deep neural networks (in Section 2.5.2, we provide a basic explanation of neural networks). The most prominent example of this is the *autoencoder*: a neural network with one or more hidden layers that has the objective to reconstruct the provided input. By including a hidden layer with relatively few nodes, usually called the *representation* layer, the network is encouraged to find a way to compactly represent input samples. Once the network is trained, the compact representation of each sample constitutes its extracted features.

The network can be considered to consist of two parts: the encoder $h$, which maps the sample $\mathbf{x}$ to its latent representation $h(\mathbf{x})$, and the decoder $g$, which attempts to map the latent representation back to the original sample vector $\mathbf{x}$. The network is trained by optimizing a loss function penalizing the *reconstruction error*: a measure of inconsistency between the input sample $\mathbf{x}$, and the reconstructed input sample $g(h(\mathbf{x}))$. Once the network is trained, the latent representation of any sample $\mathbf{x}$ can be found by simply propagating it through the first part of the network to obtain $h(\mathbf{x})$. A popular type of autoencoders is the *denoising autoencoder*, which is trained on noisy versions of samples, penalizing the reconstruction error of the reconstructions against the noiseless originals [184]. Another variant, the contractive autoencoder, directly penalizes the sensitivity of the autoencoder to perturbations in the input [147].

Autoencoders attempt to find a lower-dimensional representation of the input space without sacrificing substantial amounts of information. Thus, they inherently act on the assumption that the input space contains lower-dimensional substructures on which the data lie. Furthermore, when applied as a preprocessing step to classification, they assume that two samples on the same lower-dimensional substructure have the same label. These obervations indicate that the assumptions relied on by autoencoders are closely related to the semi-supervised *manifold assumption*.

In some domains, data is not inherently represented as a meaningful feature vector. Since many common classification methods require such a representation, feature extraction is a necessity in those cases. The feature extraction step, then, consists of finding an *embedding* of the entity in a vector space by taking into account the relations between different inputs. Examples of such approaches can be found in natural language processing [41, 126] and network science [75, 137, 187].

### 2.4.2 Cluster-then-label

Clustering and classification have traditionally been regarded as relatively disjoint research areas. However, many semi-supervised learning algorithms use principles from clustering approaches to guide the classification process. *Cluster-then-label* approaches form a group of methods that explicitly join the clustering and classification processes: they first apply an unsupervised or semi-supervised clustering algorithm to all available data, and use the resulting clusters to guide the classification process.

Goldberg et al. first cluster the labeled data and a subset of the unlabeled data [66]. Then, they independently train a classifier for each cluster on the labeled data in the cluster. Finally, the unlabeled data points are classified using the classifier of their respective cluster. In the clustering step, they construct a graph over the data points using the Hellinger distance; size-constrained spectral clustering is then applied to the resulting graph. Since the clustering is only used to segment the data, after which individual learners are applied to each cluster, the approach supports any supervised base learner.

Demiriz et al. first cluster the data in a semi-supervised manner, favoring clusters with limited label impurity, and use the resulting clusters in classification [52]. Dara et al. propose a more elaborate preprocessing step, applying self-organizing maps [98] to the labeled data in an iterative fashion [47]. The unlabeled data points are then mapped, yielding a cluster assignment for each of them. If the cluster to which an unlabeled sample is mapped contains only samples of the same label, the unlabeled sample is pseudo-labeled with that label. This process can be iterated, after which the resulting label assignments can be used to train an inductive classifier (Dara et al. train a multilayer perceptron). As such, the approach can also be regarded as a form of wrapper method (see Section 2.3).

### 2.4.3 Pre-training

In pre-training methods, unlabeled data is used to guide the decision boundary towards potentially interesting regions before applying supervised training to the model parameters.

This approach naturally applies to deep learning methods, where each layer of the hierarchical model can be considered a latent representation of the input data. The most commonly known algorithms corresponding to this paradigm are *deep belief networks* and *stacked autoencoders*. Both methods are based on artificial neural networks, and aim to guide the parameters (weights) of a network towards interesting regions in model space using the unlabeled data, before finetuning the parameters with the labeled data.

Pre-training approaches have deep roots in the field of deep learning. Since the early 2000's, neural networks with multiple hidden layers (deep neural networks) have been gaining an increasing amount of attention. However, due to their high number of tunable parameters, training these networks was often difficult: convergence tended to be slow, and the optimization procedures tended to generalize poorly [61]. Before solutions to this problem were found in the form of weight sharing, regularization methods, and different activation functions, unsupervised preprocessing methods were a popular approach to mitigate this problem. Consequently, the associated research we present in this section mainly stems from the first decade of the 2000s; the approaches have since been generally superseded by the aforementioned other solutions to training deep neural networks. However, the principles of these methods still apply, and are still to some extent used in other methods (such as ladder networks, see Section 2.5.2).

Deep belief networks consist of multiple stacked restricted Boltzmann machines (RBM), which are trained layer-by-layer with unlabeled data in a greedy fashion [85]. The resulting weights are then used as the initialization for a deep neural network with the same architecture but with an additional output layer, enabling the model to be trained in a supervised manner on the labeled data.

Stacked autoencoders are very similar to deep belief networks, but they use autoencoders as their base models instead of RBMs. The autoencoders are trained layer-by-layer, where the encoding $h(\mathbf{x})$ of each autoencoder is passed as the input to the next autoencoder. The next autoencoder is then trained to reconstruct the output of the previous autoencoder. Finally, the trained autoencoders are combined, an output layer is added as is done in deep belief networks, and the resulting network is trained on the labeled data in a supervised manner. The paradigm works with multiples types of autoencoders, including denoising autoencoders [184] and contractive autoencoders [147].

Erhan et al. provide an empirical analysis of deep belief networks and stacked autoencoders, and suggest that unsupervised pre-training guides the model towards regions in model space that provide better generalization [61].

Deep neural networks are often motivated from the perspective that they learn a more high-level representation of the data at each layer. Thus, each layer of the network can be considered to contain a different representation of the input data. Both deep belief networks and stacked autoencoders attempt to guide the model in the extraction of these hierarchical representations, pushing the model towards the extraction of representations that are deemed informative. As such, pre-training methods are closely related to the unsupervised feature extraction methods described earlier. The crucial difference between the two methods, however, is that unsupervised pre-training methods allow the pre-trained parameters to be changed in the supervised finetuning phase, whereas these parameters are fixed in unsupervised feature extraction methods. We emphasize that, should the parameters determined in the pre-training phase be fixed in the finetuning phase, pre-training and unsupervised feature extraction are conceptually identical.

## 2.5 Intrinsically semi-supervised methods

In this section, we consider inductive machine learning algorithms that directly optimize an objective function with components for labeled and unlabeled samples. These methods, which we refer to as *intrinsically semi-supervised* methods, do not rely on any intermediate steps or supervised base learners. Usually, they are extensions of existing supervised methods to include unlabeled samples in the objective function.

Generally, these methods either explicitly or implicitly rely on one of the semi-supervised learning assumptions (see Section 1.3). For instance, maximum-margin methods rely on the low-density assumption, and most semi-supervised neural networks rely on the smoothness assumption. We start this section with an overview of the earliest intrinsically semi-supervised classification methods, namely maximum-margin methods. We proceed with perturbation-based methods, which directly incorporate the smoothness assumptio. These encompass most semi-supervised neural networks. Next, we consider manifold-based techniques, which either explicitly or implicitly approximate the manifolds on which the data lie. Lastly, we consider generative models.

### 2.5.1 Maximum-margin methods

Maximum-margin classifiers attempt to maximize the distance between the samples and the decision boundary. This approach corresponds to the semi-supervised low-density assumption: when the margin between all samples and the decision boundary is large (save for some outliers), the decision boundary must be in a low-density area [14]. This gives rise to an obvious extension of maximum-margin methods to the semi-supervised setting: one can incorporate knowledge from the unlabeled data to determine where the density is low, and, thus, where the margin is large.

#### Support vector machines

The most prominent example of supervised maximum-margin classifiers is the *support vector machine* (SVM): a classification method that attempts to maximize the distance from the decision boundary to the points closest to it, while encouraging samples to be correctly classified. It was one of the first maximum-margin approaches to be studied in the semi-supervised setting, and it has been studied extensively since.

We briefly introduce supervised SVMs, but refrain from providing an extensive introduction. For more information on SVMs, we refer the reader to [20]. The objective of SVMs is to find a decision boundary that maximizes the *margin*, which is defined as the distance between the decision boundary and the sample or samples closest to it. Samples are allowed to violate the margin (i.e., lie between the margin and the decision boundary or even past the margin on the wrong side of the decision boundary) at a certain cost. SVMs supports implicit mapping of samples to higher-dimensional feature spaces using the so-called *kernel trick*.

Formally, the objective of SVMs is to find a weight vector $\mathbf{w} \in \mathbb{R}^d$ with minimal magnitude and a bias variable $b \in \mathbb{R}$ such that $y_i \cdot (\mathbf{w}^\intercal \cdot \mathbf{x}_i + b) \geq 1 - \xi_i$ for all samples $\mathbf{x}_i \in X_L$. Here, $\xi_i \geq 0$ is called the "slack variable" for sample $i$, which allows the sample to violate the margin at some cost. This cost is incorporated into the objective function. The corresponding optimization problem can be formulated as follows,

$$\begin{aligned} \underset{\mathbf{w},b,\boldsymbol{\xi}}{\text{minimize}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C \cdot \sum_{i=1}^{l} \xi_i \\ \text{subject to} \quad & y_i \cdot (\mathbf{w}^\mathsf{T} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, l, \\ & \xi \geq 0, \quad i = 1, \ldots, l, \end{aligned} \tag{2.4}$$

where $C \in \mathbb{R}$ is some constant scaling factor for the penalization of samples violating the margin. If $C$ is large, the optimal margin will generally be narrow, and if $C$ is small, the optimal margin will generally be wide. Thus, $C$ acts as a regularization parameter, governing the trade-off between the complexity of the decision boundary and the predictive accuracy on the training set.

The concept of semi-supervised SVMs, or S3VMs, is similar: we want to maximize the margin, and we want to correctly classify the labeled data (implicitly incorporated into the objective function via the margin violation cost). However, in the semi-supervised setting, an additional objective becomes apparent: we also want to minimize the number of unlabeled samples violating the margin. Since the labels of the unlabeled data are unknown, unlabeled samples violating the margin are penalized based on their distance to the closest margin.

The intuitive extension of the optimization problem for S3VMs thus becomes

$$\begin{aligned} \underset{\mathbf{w},b,\boldsymbol{\xi}}{\text{minimize}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C \cdot \sum_{i=1}^{l} \xi_i + C' \cdot \sum_{i=l+1}^{n} \xi_i \\ \text{subject to} \quad & y_i \cdot (\mathbf{w}^\mathsf{T} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \quad i = 1, \ldots, l, \\ & |\mathbf{w}^\mathsf{T} \cdot \mathbf{x}_i + b| \geq 1 - \xi_i, \quad i = l+1, \ldots, n, \\ & \xi_i \geq 0, \quad i = 1, \ldots, n, \end{aligned} \tag{2.5}$$

where $C' \in \mathbb{R}$ is the margin violation cost associated with unlabeled samples.

S3VMs were proposed by Vapnik [183], who motivated the problem from a more transductive viewpoint: instead of optimizing only over the weight vector, bias and slack variables, they proposed to also optimize over the label predictions $\hat{\mathbf{y}}_U$. The constraint for the unlabeled data was formulated similarly to the constraint for the labeled data, but with the predicted labels $\hat{\mathbf{y}}_U$. Though different at first sight, this formulation is equivalent to the optimization problem we describe in Equation 2.5, since any labeling $\hat{\mathbf{y}}_U$ can only be optimal if, for each $\hat{y}_i \in \hat{\mathbf{y}}_U$, the sample $\mathbf{x}_i$ is on the correct side of the decision boundary (i.e., $\hat{y}_i \cdot (\mathbf{w}^\mathsf{T} \cdot \mathbf{x}_i + b) \geq 0$). Otherwise, a better solution could be obtained by simply inverting the labeling of the sample.

The extension of SVMs to the semi-supervised setting carries one significant disadvantage: the optimization problem becomes non-convex. Training S3VMs thus becomes an NP-hard problem. Most efforts in the study of S3VMs have been focused on efficiently training them.

Initial efforts showed promising results in applying S3VMs, but only to small data sets. For instance, Bennett and Demiriz proposed to use the L1-norm instead of the

L2-norm in the objective function and pose the problem as a mixed integer programming problem [16]. The earliest widely used optimization approach was introduced by Joachims [90], who proposed to solve the optimization problem by starting with a random assignment of $\hat{\mathbf{y}}_U$ and a low value for $C'$. Each iteration of the algorithm then consists of three steps. Firstly, the supervised SVM optimization problem corresponding to the current label assignment $\hat{\mathbf{y}}_U$ is solved. Secondly, the algorithm inverts the labels of each pair of samples where inverting the labels would improve the objective function until no more such pairs exist. Lastly, $C'$ is increased. The algorithm terminates when $C'$ reaches a predefined required parameter value for $C'$ specified by the user.

Other approaches to solving S3VMs have been proposed. For instance, several studies have proposed convex relaxations of the minimization function, which can be solved using semidefinite programming methods. The earliest such approach was proposed by De Bie and Cristianini [49, 50]; later, this approach was extended to the multiclass setting by Xu and Schuurmans [202]. These approaches, however, do not scale to large amounts of data due to their time complexity.

Chapelle et al. provide an overview of optimization procedures for S3VMs up until 2008 [34], and broadly categorize S3VM optimization methods into two categories: *combinatoral methods*, finding the label assignment $\hat{\mathbf{y}}_U$ that can minimize the objective function, and *continuous methods*, directly solving the optimization problem using label assignments $\hat{y}_i = \text{sign}(\mathbf{w}^\intercal \cdot \mathbf{x}_i + b)$. The approaches we have thus far described are all combinatoral approaches. However, the formulation we provide in Equation 2.5 corresponds to the continuous approach. Such an approach is taken, for example, with the Concave-Convex Procedure (CCCP), which decomposes the non-convex objective function into a convex and a concave component, and iteratively solves the optimization problem by replacing the concave component with a linear approximation at the current solution [34, 40].

Other continuation methods make use of the fact that the optimization problem in Equation 2.5 can be reformulated as an optimization problem without constraints. This stems from the fact that, if a labeled point $\mathbf{x}_i \in X_L$ does not violate the margin, $\xi_i = 0$ in the optimal solution. If it does violate the margin, $\xi_i = 1 - y_i \cdot (\mathbf{w}^\intercal \cdot \mathbf{x}_i + b)$. For unlabeled data points $\mathbf{x}_i \in X_U$, $\xi_i = 0$ if it does not violate the margin, and $\xi_i = 1 - |\mathbf{w}^\intercal \cdot \mathbf{x}_i + b|$ if it does violate the margin. Thus, the optimization problem can be reformulated as

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \frac{1}{2}||\mathbf{w}||^2 + C \cdot \sum_{i=1}^{l} \max\left(0, 1 - y_i \cdot f(\mathbf{x}_i)\right) \\ & + C' \cdot \sum_{i=l+1}^{n} \max\left(0, 1 - |f(\mathbf{x}_i)|\right), \end{aligned} \tag{2.6}$$

where $f(\mathbf{x}_i) = \mathbf{w}^\intercal \cdot \mathbf{x}_i + b$.

Such approaches include $\nabla$TSVM by Chapelle and Zien [35], who create a smooth approximation of Equation 2.6 by squaring the loss for the labeled data points, and by approximating the loss for the unlabeled data points with a Gaussian. This optimization problem is then solved by gradient descent, where $C'$ is gradually increased from some value close to 0 to its intended value. Chapelle et al. take a similar approach in [32], where they keep $C'$ fixed and use a continuation approach to transform the objective function from using only the labeled data to the final objective function.

As is the case for most semi-supervised learning methods [160], S3VMs are not guaranteed to perform better than their supervised counterparts. Specifically, if one of the underlying assumptions of the semi-supervised learning method is violated, there is a large risk of degrading performance when introducing the semi-supervised objective. In the case of S3VMs, many highly diverse decision boundaries may exist that pass through a low-density area and achieve reasonable classification performance on the labeled data. Consequently, one can expect the generalization performance of such classifiers to exhibit significant variance.

Li and Zhou propose to mitigate this problem by considering a diverse set of low-density separators and choosing the separator that performs best under the worst possible ground truth [113]. Like all S3VM implementations, their method is premised on the assumption that the optimal decision boundary lies in a low-density area. Their algorithm, called S4VM (safe S3VM), consists of two stages. Firstly, a diverse set of low-density decision boundaries is constructed. Secondly, the decision boundary with maximal worst-case performance gain over the supervised decision boundary is chosen as the S4VM's decision boundary. This problem formulation limits the probability that the solution found by S4VMs exhibits performance worse than the corresponding supervised SVM.

The performance gain is formulated as the resulting increase in the number of correctly labeled samples, minus the increase in the number of incorrectly labeled samples. The latter term is multiplied with a factor $\lambda \in \mathbb{R}$, governing the amount of risk of degradation in performance one wishes to take. Formally, they define the score function $J(\hat{\mathbf{y}}, \mathbf{y}, \mathbf{y}^{\text{svm}})$ for a set of predicted labels $\hat{\mathbf{y}}$, ground truth $\mathbf{y}$, and supervised SVM predictions $\mathbf{y}^{\text{svm}}$ as

$$J(\hat{\mathbf{y}}, \mathbf{y}, \mathbf{y}^{\text{svm}}) = earn(\hat{\mathbf{y}}, \mathbf{y}, \mathbf{y}^{\text{svm}}) - \lambda \cdot lose(\hat{\mathbf{y}}, \mathbf{y}, \mathbf{y}^{\text{svm}}), \tag{2.7}$$

where $earn$ and $lose$ denote the increases in correctly labeled samples and incorrectly labeled samples, respectively. The optimal label assignment $\bar{\mathbf{y}}$ in the worst-case true labeling of samples can then be found as

$$\bar{\mathbf{y}} \in \underset{\mathbf{y} \in \{\pm 1\}^u}{\arg\max} \left[ \min_{\hat{\mathbf{y}} \in \mathcal{M}} J(\mathbf{y}, \hat{\mathbf{y}}, \mathbf{y}^{\text{svm}}) \right], \tag{2.8}$$

where $\mathcal{M}$ is the set of all candidate label assignments such that the corresponding decision boundary cuts through a low-density area. Due to the optimization over the possible label assignments, this optimization problem is NP-complete. The authors propose a convex relaxation of the problem to find a candidate solution. Premised on the assumption that the true label assignment is indeed in this set, the authors prove that, if $\lambda \geq 1$, the performance of S4VM is never lower than the performance of the corresponding SVM. They validate this finding empirically, and show that their implementation achieves performance improvements over SVM that are similar to S3VM implementations, but that, contrary to the other implementations, performance never significantly degrades relative to supervised SVMs.

Clearly, the formulation of the second stage of the optimization is not limited to support vector machines. Indeed, it could theoretically be applied to any other semi-supervised learning algorithm. Li and Zhou additionally propose to perform both stages simultaneously in a deterministic annealing approach [113].

### Gaussian processes

The notion of margin maximization is directly incorporated into support vector machines, and it should thus not come as a surprise that they are easily extended to the semi-supervised setting. However, similar efforts have been made with other supervised methods as well. One such extension is proposed by Lawrence and Jordan, who consider an extension of *Gaussian processes* to handle unlabeled data [105].

Gaussian processes are a family of non-parametric models that estimate the posterior probability over the function $f$ mapping points in the input space to a continuous output space. When used for classification purposes, this output is in turn mapped to the label space $\mathcal{Y} = \{-1, 1\}$. In the learning phase, $f$ is learned as the function that maximizes the likelihood of observing the data points $((\mathbf{x}_i, y_i))_{i=1}^{l}$ given $f$. The resulting model can be considered an $l$-dimensional Gaussian distribution over the label vector $\mathbf{y}$ of the input data points, where $l$ is the number of labeled data points. Predictions for previously unseen data points $\mathbf{x}^*$ can then be made by the model by evaluating the posterior probability of the data point's class label, conditioned on the observed data points $X$, their associated labels $\mathbf{y}$, and the observed data point $\mathbf{x}^*$. The associated covariance matrix is the Gram matrix obtained from all $l + 1$ data points using some kernel function $k$.

Lawrence and Jordan extend Gaussian processes to the semi-supervised case by incorporating the unlabeled data points into the likelihood function [105]. Specifically, the likelihood for an unlabeled data point $\mathbf{x}$ is low when it is close to the decision boundary (i.e., when $f(\mathbf{x})$ is close to 0), and high when it is far away from the decision boundary. The space of possible labels is expanded to include a *null category*; the posterior probability of this null category is high around the decision boundary. By imposing the constraint that unlabeled data points can never be mapped to the null category, the model is explicitly discouraged from choosing a decision boundary that passes through a high-density area of unlabeled data points. In other words, unlabeled data points should be far away from the decision boundary.

This extension of Gaussian processes to the semi-supervised setting has an interesting side effect: contrary to supervised Gaussian processes, introducing additional (unlabeled) data can increase the posterior variance. In other words, additional data can increase uncertainty. This effect stems from the observation that the likelihood function for a single unlabeled data point $\mathbf{x}^*$ can be bimodal if $f(\mathbf{x}^*)$ is close to 0.

### Density regularization

Another way of encouraging the decision boundary to pass through a low-density area is to explicitly incorporate the amount of overlap between the estimated posterior class probabilities into the cost function. When there is a large amount of overlap, the decision boundary passes through a high-density area, and when there is a small amount of overlap, it passes through a low-density area. Several approaches have been proposed to use this assumption to regularize the classification objective function.

Grandvalet et al. propose to formalize this in the *maximum a posteriori* (MAP) framework by imposing a prior on the model parameters, favoring parameters inducing small class overlap in the predictive model [33, 72]. In particular, they use Shannon's conditional entropy as a measure of class overlap. The prior is weighted by a constant $\lambda \in \mathbb{R}$. The resulting objective is generally non-convex. The authors propose solving the optimization problem by means of deterministic annealing. The entropy regularization method can be applied to any supervised learning method based on maximum-likelihood; the authors report experiments using logistic regression.

Corduneanu and Jaakkola propose to directly incorporate $p(x)$, estimated from the unlabeled data, into the objective function [42]. They propose a regularizer that reflects the belief that, in high-density areas, the posterior probability of $y$ conditioned on $x$ should not vary too much. They cover the entire domain in multiple, small regions, and regularize the objective function by the sum of the mutual information between labels and inputs in each of these regions, weighted by the estimated density in the region. Their work is an extension of the work by Szummer and Jaakkola [170].

Liu et al. propose to incorporate the prior density into the node splitting criterion of decision trees [118, 119]. At each decision tree node, they attempt to find a hyperplane to split the data, penalizing high-density areas. They approximate $p(x)$ by using Gaussian kernel density estimators. They conduct experiments with random forests consisting of 100 of the resulting semi-supervised decision trees. They report significant performance improvements over supervised random forests in several data sets.

#### Pseudo-labeling as a form of margin maximization

Depending on the base learner used, the self-training approach described in Section 2.3 can also be regarded a margin-maximization method. For instance, when using self-training with supervised SVMs, the decision boundary is iteratively pushed away from the unlabeled samples. Even though the unlabeled data are not explicitly incorporated into the loss function, the underlying assumption which is exploited is, like with S3VMs, the low-density assumption.

### 2.5.2 Perturbation-based methods

The smoothness assumption entails that a predictive model should be invariant to local perturbations in its input. This means that, when we augment a data point $\mathbf{x}$ with noise to produce the noisy sample $\tilde{\mathbf{x}}$, the predictions for the noisy input and the clean input should be similar. Since this expected similarity is not dependent on the true label for the data points, we can make use of unlabeled data.

Many different methods exist to incorporate the smoothness assumption into a given learning algorithm. For instance, one could apply noise to the input data points, and incorporate the difference between the clean and the noisy predictions into the loss function. Alternatively, one could implicitly apply noise to the data points by perturbing the model itself. These type of methods comprise what we refer to as *perturbation-based methods*, which are based on perturbations of the input or the model.

Perturbation-based methods are often implemented with neural networks. Due to their straightforward incorporation of additional (unsupervised) loss terms in their objective function, they are extendable to the semi-supervised case with relative ease. In recent years, neural networks have received renewed interest due their successful application in various application areas (see, e.g., [41, 101, 106]). As a result, interest in semi-supervised neural networks has risen as well. In particular, neural networks with many layers, so-called deep neural networks, have shown interesting extensions to the semi-supervised case. These intrinsically semi-supervised neural networks differ from the neural networks used for feature extraction, which we discussed in Section 2.4.1: the unlabeled data is incorporated directly into the optimization objective, rather than in a separate preprocessing step. Before continuing our discussion of such methods, we provide a short, general introduction to neural networks for the reader not too familiar with them. For a more extensive introduction to (deep) neural networks, we refer the reader to [69].

**Neural networks**

A neural network is a computing system that computes an output vector by propagating an input vector through a network of simple processing elements with weighted connections between them. These simple processing elements are called *nodes*, which contain an *activation function* that transforms their input. Nodes are usually grouped together into *layers*, where nodes from each layer are only connected to nodes from adjacent layers. The output vector is calculated by propagating the input vector, which constitutes the first layer, through the weighted connections of the network. The output of each node is calculated by applying its activation function to the weighted sum of its inputs.

In supervised neural networks, the network weights are generally optimized to calculate the desired output value for a given input vector. Considering the classification task, let $f : \mathbb{R}^d \mapsto \mathbb{R}^{|\mathcal{Y}|}$ denote the vector-valued function modeled by a neural network, mapping an input vector $\mathbf{x} \in \mathbb{R}^d$ to a $|\mathcal{Y}|$-dimensional output vector, where $\mathcal{Y}$ denotes the set of possible classes. The function $f$ is modeled by a neural network consisting of one or multiple layers, and $f(\mathbf{x})$ is the output obtained by propagating the input $\mathbf{x}$ through the network and evaluating the activations of the final layer.

A loss function $\ell$ is then defined, calculating the cost associated with output layer activations $f(\mathbf{x})$ for a sample $\mathbf{x}$ with true label $y$. The complete loss function, then, is defined as

$$\mathcal{L}(W) = \sum_{i=1}^{l} \ell(f(\mathbf{x}_i; W), y_i), \tag{2.9}$$

where $W$ denotes the weight matrix of the matrix, where the element at position $(i, j)$ denotes the weight between node $i$ and $j$. The explicit notion of the parametrization of $f$ by $W$ is often omitted for conciseness.

The network's weights are iteratively optimized by passing input samples through the network and propagating the share of one or more samples in the cost $\mathcal{L}$ backwards through the network. In this process, known as backpropagation, the network weights are updated using gradient descent or a similar method to minimize the loss [69]. To converge, the network generally needs to pass over the entire training set multiple times. One such pass over the entire training set is known as an *epoch*.

### Semi-supervised neural networks

The simplicity and efficiency of the backpropagation algorithm for a great variety of loss functions make it attractive to simply add an unsupervised component to $\mathcal{L}$. This approach, which can be considered a form of regularization over the unlabeled data, is employed by virtually all semi-supervised deep learning methods. Furthermore, the hierarchical nature of representations in deep neural networks make them a viable candidate for other semi-supervised approaches. If deeper layers in the network express increasingly abstract representations of the input sample, one can argue that unlabeled data could be used to guide the network towards more informative abstract representations. This argument can be readily implemented in deep neural networks through the smoothness assumption, and gives rise to perturbation-based semi-supervised neural networks.

### Ladder networks

The first such approach is the semi-supervised *ladder network*, proposed by Rasmus et al. [143]. It extends a feedforward network to incorporate unlabeled data by using the feedforward part of the network as the encoder of a denoising autoencoder, adding a decoder, and including a term in the loss function penalizing the reconstruction cost. The underlying idea is that the network should be able to discover useful features for the unlabeled data (i.e., ones that are predictive of the label).

Consider a feedforward network with $K$ hidden layers. We denote the inputs of a layer $k$ (after normalization) as $\mathbf{z}^{(k)}$, and the layer's activations (i.e., after applying the activation function) as $\mathbf{h}^{(k)}$. The weights are stored in the weight matrix $W$. In a regular feedforward network, the loss function of a sample is calculated by comparing the activations of the final layer $f(\mathbf{x}_i) = \mathbf{h}^{(K)}$ to the sample's label $y_i$ with $\ell(f(\mathbf{x}_i), y_i)$. For the same reason, when referring to layer inputs and activiations, we do not explicitly mention the sample $\mathbf{x}$, nor the parametrization $W$ (e.g., we write $\mathbf{h}^{(k)}$ for the activation vector of the $k$-th layer in a neural network with weights $W$ for an input sample $\mathbf{x}$). As is shown in Equation 2.9, the final loss function for the network is then $\mathcal{L} = \sum_{i=1}^{l} \ell(f(\mathbf{x}_i), y_i)$.

Ladder networks add an additional term to $\mathcal{L}$, penalizing the sensitivity of the network to small perturbations of the input. This is achieved by treating the entire network as the encoder part of a denoising autoencoder: isotropic Gaussian noise with mean 0 and fixed variance is added to the input samples, and the existing feedforward network is treated as the *encoder* part. A *decoder* is then added alongside it, which is supposed to take the final-layer representation $\mathbf{h}^{(K)}$ of the noisy input sample $\tilde{\mathbf{x}}$, and transform it to reconstruct $\mathbf{x}$. To achieve this latter goal, a *reconstruction cost* is added to the loss function of the network (which also still includes the original term for labeled data), allowing it to be trained with backpropagation. This reconstruction loss can, of course, be applied to unlabeled data. Although the autoencoder component of ladder networks is highly similar to regular denoising autoencoders, it differs from them in two ways.

Firstly, the ladder network injects noise not only at the input layer, but at every layer. We denote the noisy inputs of a layer $k$ as $\tilde{\mathbf{z}}^{(k)}$, and the resulting activations as $\tilde{\mathbf{h}}^{(k)}$. The supervised loss component for each sample then becomes $\ell(\tilde{\mathbf{h}}^{(K)}, y)$: i.e., the loss function is evaluated against the output for the noisy sample. Note that, in the testing phase, the output is generated without adding any noise.

Secondly, ladder networks utilize a different reconstruction cost calculation. Where regular denoising autoencoders only penalize the difference between the clean input $\mathbf{x}$ and the reconstructed version $\hat{\mathbf{x}}$ of the noisy input $\tilde{\mathbf{x}}$, the ladder network also penalizes local reconstructions of the hidden representations of the data. To do so, they enforce the decoder to have $K$ layers, the same number of layers as the original network (the encoder). Each of these layers is also required to have the same number of neurons as the corresponding layer in the encoder. As a sample passes through the encoder, noise is added to the layer inputs at each layer. Then, at each layer in the decoder, the reconstructed representation $\hat{\mathbf{z}}^{(k)}$ is compared to the hidden representation $\mathbf{z}^{(k)}$ of the clean sample $\mathbf{x}$ at layer $k$ in the encoder. This, of course, requires each sample to pass through the network twice: once without noise (to obtain $\mathbf{z}$), and once with noise (to obtain $\tilde{\mathbf{z}}$ and and the reconstructed $\hat{\mathbf{z}}$).

The final semi-supervised cost function of ladder networks then becomes

$$\mathcal{L}(W) = \sum_{i=1}^{l} \ell(f(\mathbf{x}_i), y_i) + \sum_{i=1}^{n} \sum_{k=1}^{K} \text{ReconsCost}(\mathbf{z}_i^{(k)}, \hat{\mathbf{z}}_i^{(k)}), \tag{2.10}$$

where $\text{ReconsCost}(\cdot, \cdot)$ is defined as the norm of the difference between the two normalized latent vectors.

Through their penalization of reconstruction errors, ladder networks effectively attempt to push the network towards extracting interesting latent representations of the data. The method is premised on the assumption that a latent representation $\mathbf{h}^{(K)}$ that is useful for reconstructing $\mathbf{x}$, is also useful for predicting the sample label. Rasmus et al. show that ladder networks achieve state-of-the-art results on image data sets with partially labeled data, including MNIST [143]. Interestingly, they also show improvements when using only labeled data.

Rasmus et al. also propose a simpler, computationally more efficient variant of ladder Networks. This method, generally referred to as the $\Gamma$-model in the literature, only includes the reconstruction cost for the top-level layer. Therefore, no full decoder needs to be constructed.

As outlined above, several modifications to regular neural networks are introduced in ladder networks, and their individual contributions are not immediately distinguishable. An extensive empirical study of the different components of ladder networks and their influence on performance was conducted by Pezeshki et al. [138]. They show that the reconstruction cost at the first layer of the neural network, combined with the introduction of noise at the first layer, is one of the key drivers of the success of ladder networks.

### Pseudo-ensembles

Instead of explicitly perturbing the input data, one can also perturb the neural network model itself. Robustness in the model can then be promoted by imposing a penalty on the difference between the activations of the perturbed model and the activations of the original model. Bachman et al. propose a general framework for this approach, where an unperturbed *parent model* with parameters $\boldsymbol{\theta}$ is perturbed to yield one or more *child models* [5]. In this framework, which they call *pseudo-ensembles*, the perturbation is obtained from a stochastic process $p(\xi)$, from which model noise $\xi$ can be drawn. The perturbed model $f_{\boldsymbol{\theta}}(x;\xi)$ is then generated based on the unperturbed parent model $f_{\boldsymbol{\theta}}(x)$ and the model noise. The semi-supervised loss function then consists of a supervised part and an unsupervised part. The supervised component optimizes the expected loss of a perturbed model for labeled samples, and the unsupervised component optimizes the consistency across perturbed models for the unlabeled samples.

Consider a neural network with $K$ layers, and let $f^k_{\boldsymbol{\theta}}(x)$ and $f^k_{\boldsymbol{\theta}}(x;\xi)$ denote the $k$-th layer activations of the unperturbed and the perturbed model, respectively. The expectation of the joint loss function of the pseudo-ensemble for neural networks then becomes

$$\mathop{\mathbb{E}}_{(x,y)\sim p(x,y)} \mathop{\mathbb{E}}_{\xi\sim p(\xi)} \mathcal{L}(f_{\boldsymbol{\theta}}(x;\xi), y) \tag{2.11}$$

$$+ \mathop{\mathbb{E}}_{x\sim p(x)} \mathop{\mathbb{E}}_{\xi\sim p(\xi)} \sum_{k=1}^{K} \lambda_k \cdot \mathcal{V}_k \left(f^k_{\boldsymbol{\theta}}(x), f^k_{\boldsymbol{\theta}}(x;\xi)\right), \tag{2.12}$$

where the consistency loss $\mathcal{V}_k$ penalizes differences between the activations of the unperturbed and perturbed models at the $k$-th layer for the same input sample; $\lambda_k$ is the relative weight of that particular cost term. The expectations can be approximated by replacing the expectations over $p(x)$ and $p(x,y)$ with the true samples and by sampling from the noise process. Bachman et al. propose to gradually increase $\lambda$ over time, in effect placing more weight on the supervised objective in early iterations.

One particularly popular noise process is *dropout*, which randomly sets weights to 0 (i.e., removes connections in the neural network) in each training iteration [164]. In its originally proposed form, it was only applied to the supervised loss component. However, Wager et al. [185] and Bachman et al. [5] showed that it can be cast as regularization and, as such, be applied to unlabeled data as well.

The framework proposed by Bachman et al. is not limited to semi-supervised settings: the supervised term in the loss function can be applied to any supervised learning problem. Furthermore, a similar approach could be applied to other learning algorithms than neural networks, although the per-layer activation comparison would have to be replaced by an alternative. Of course, since neural networks are entirely parametrized by connection weights, they offer a relatively straightforward implementation of model perturbation.

### Π-model

Instead of comparing the unperturbed parent model with the perturbed models in the loss function, one can also compare the perturbed models directly. A simple variant of this approach, where two perturbed models are trained, is suggested by Laine and Aila [103]. They use dropout [164] as the perturbation process, and penalize the differences in the final layer activations of the two models with the squared loss. The weight of the unsupervised term in the loss function starts at 0, and is gradually increased. This model, which they name the Π-model, can be seen as a simple variant of pseudo-ensembles.

### Temporal ensembling

Since the noise process is a random process, the entire neural network model can be considered a stochastic model. With the Π-model, this random process is regularized by penalizing the difference in output of two runs of the sampled model on the same input. This idea can be extended to more than two perturbed models. Such an approach is taken by Sajjadi et al., who additionally perturb the input samples with random transformations [151]. Of course, that will increase the running time of each training iteration quadratically in the number of perturbations. Pseudo-ensembles solve this problem by comparing the perturbed model activations to the activations of the unperturbed model.

In the same paper in which they propose the Π-model, Laine and Aila propose a different approach to combining multiple perturbations of the model: they compare the activations of the neural network at each epoch to the activations of the network at previous epochs [103]. In particular, after each epoch, they compare the output of the model to the exponential moving average of the outputs of the model in previous epochs. Since the model weights are changed in each iteration, this cannot be considered a form pseudo-ensembling, but it does act on similar principles: the model output is smoothed over model perturbations. Based on the procedure of penalizing the difference in the model outputs at different points in the training process, Laine and Aila name this model *temporal ensembling*.

Temporal ensembling can be considered an extension of the Π-model. However, instead of comparing $f(\mathbf{x}; \xi)$ to $f(\mathbf{x}; \xi')$ for $\xi, \xi' \sim p(\xi)$, it is compared to the exponential moving average of final-layer activations in previous epochs. Since the loss function for unlabeled data points depends on the model output in previous iterations, temporal ensembling is closely related to pseudo-labeling methods such as the *pseudo-label* approach [107] and self-training. The crucial difference, however, is that the entire set of final-layer activations is compared to the activations of the previous model, whereas self-training approaches and *pseudo-label* convert these outputs to a single, hard prediction (the pseudo-label).

### Mean Teacher

When training a neural network using temporal ensembling, unlabeled data points are incorporated into the learning process at large intervals. Since the activations of each sample are only generated once per epoch, it takes a long time for the activations of unlabeled samples to influence the inference process. Tarvainen and Valpola attempt to overcome this problem by considering moving averages over model weights, instead of moving averages over model activations [177].

They suggest to calculate the exponential moving average of model weights at each training iteration, and compare the resulting final-layer activations to the final-layer activations when using the latest set of model weights. Furthermore, they impose noise on the input samples to increase robustness. Formally, consider a neural network with weights $\theta_t$ at iteration $t$, and a set of averaged weights $\theta'_t$. The loss function $\ell$ for an unlabeled sample, then, is calculated as $\ell(\mathbf{x}) = ||f(\tilde{\mathbf{x}}, \theta'_t) - f(\tilde{\mathbf{x}}', \theta_t)||^2$, where $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$ are two noise-augmented versions of $\mathbf{x}$. After updating $\theta$ using backpropagation, $\theta'_{t+1}$ is calculated by $\theta'_{t+1} = \alpha \cdot \theta'_t + (1 - \alpha) \cdot \theta_{t+1}$, where $\alpha$ is the decay rate.

Tarvainen and Valpola evaluate their method on two image data sets using different amounts of labeled data. They compare their approach to several other methods, and report performance comparable to the $\Pi$-model and temporal ensembling.

### 2.5.3 Manifolds

Perturbation-based methods make direct use of the smoothness assumption, penalizing differences in a classifier's behavior under slight changes in the input. However, one can imagine that not all minor changes to the input should yield similar outputs. In particular, if the data lie on lower-dimensional manifolds, one can expect the classifier to be insensitive only to minor changes along the manifold. This observation corresponds to the manifold assumption, which forms the basis of a significant body of intrinsically semi-supervised learning algorithms.

An $m$-dimensional manifold is a subspace of our original input space that locally resembles Euclidean space $\mathbb{R}^m$. Reiterating the definition from Section 2.1, the manifold assumption states that (a) data points lie on a lower-dimensional manifold and (b) data points lying on the same lower-dimensional manifold have the same label. Formally, the first part of the manifold assumption states that each conditional probability distribution $p(x|y = k)$ for class $k$ has a structure corresponding to the union of one or more Riemannian manifolds $\mathcal{M}$. The second part, then, states that points on the same Riemannian manifold $\mathcal{M}$ should have the same label. When these assumptions hold, one can expect that knowledge of the manifolds in our input space can aid in classification.

In this section, we consider two general types of methods utilizing the manifold assumption. Firstly, we consider *manifold regularization* techniques, which define a graph over the samples and implicitly penalize differences in classifier predictions for samples with small geodesic distance. Secondly, we consider *manifold approximation* techniques, which explicitly estimate the manifolds $\mathcal{M}$ on which the data lie and optimize an objective function accordingly.

#### Manifold regularization

Consider a labeled data point $\mathbf{x}_i$ and an unlabeled data point $\mathbf{x}_j$, and assume $\mathbf{x}_i$ lies on some manifold $\mathcal{M}$. Then we expect $\mathbf{x}_j$ to be assigned the same label as $\mathbf{x}_i$ if it also lies on manifold $\mathcal{M}$. If it does indeed lie on the same manifold, there is, by definition of the manifold, a path along the manifold between $\mathbf{x}_i$ and $\mathbf{x}_j$ (a so-called *geodesic*). Since we assume that all data points on the same manifold have the same label, and that the conditional probability distribution $p(x|y = y_i)$ corresponds to this manifold, we expect there to be more samples $\mathbf{x}^*$ located on the manifold.

If we have enough unlabeled samples, we can thus expect there to be some "path" from $\mathbf{x}_j$ to $\mathbf{x}_i$, passing through other labeled or unlabeled samples, such that each path segment has a relatively small length. We can concretize this notion of a path by defining a *graph* over all data points, connecting samples that are close together in the original input space with some possibly weighted edge. This is the underlying principle of *graph-based methods*, which also form the basis of transductive semi-supervised learning (see Section 2.6).

This approach is proposed by Belkin et al., who formulate a general manifold regularization framework [12, 13]. They consider a kernel $K : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ with a corresponding hypothesis space $\mathcal{H}_K$ and an associated norm $|| \cdot ||_K$. For supervised problems, then, they formulate the following general optimization problem:

$$\underset{f \in \mathcal{H}_K}{\text{minimize}} \quad \sum_{i=1}^{l} \ell(f(\mathbf{x}_i), y_i) + \gamma \cdot ||f||_K^2, \tag{2.13}$$

for some loss function $\ell$ on labeled data. Here, $\gamma$ denotes the relative influence of the smoothing term. This objective function simultaneously penalizes misclassifications and promotes smoothness of the predictive function. For semi-supervised problems, they add an additional unsupervised regularization term, penalizing differences in label assignments for pairs of samples that have a direct edge between them in the graph. Implicitly, they thereby encourage samples on the same approximated manifold to receive the same label prediction.

This unsupervised regularization term gives rise to the class of *manifold regularization* methods. Consider a similarity graph with symmetric weighted adjacency matrix $W$, where $W_{ij}$ denotes the similarity between samples $\mathbf{x}_i$ and $\mathbf{x}_j$ ($W_{ij} = 0$ if the samples are not connected). Let $D$ denote the degree matrix, which is a diagonal matrix with $D_{ii} = \sum_{j=1}^{n} W_{ij}$. The manifold regularization term $||f||_I^2$ is then defined as

$$||f||_I^2 = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} (f(\mathbf{x}_i) - f(\mathbf{x}_j))^2. \tag{2.14}$$

The regularization term can be expressed as $\mathbf{f}^\intercal \cdot L \cdot \mathbf{f}$, where $L = D - W$ is the graph Laplacian and $\mathbf{f} \in \mathbb{R}^n$ is the vector of evaluations of $f$ for each $\mathbf{x}_i$. The final optimization problem, including the manifold regularization term, becomes

$$\underset{f \in \mathcal{H}_K}{\text{minimize}} \quad \frac{1}{l} \sum_{i=1}^{l} \ell(f(\mathbf{x}_i), y_i) + \gamma \cdot ||f||_K^2 + \gamma_U \cdot ||f||_I^2, \tag{2.15}$$

where $\gamma_U$ determines the relative influence of the manifold regularization term.

This general framework leads to semi-supervised extensions of popular supervised learning algorithms, such as *Laplacian support vector machines* (LapSVM), where the loss function $\ell$ is defined as the hinge loss, i.e., $\ell(\hat{y}, y) = \max(1 - y\hat{y}, 0)$. The supervised objective of LapSVM maximizes the margin, and the unsupervised objective maximizes consistency of predictions along the estimated manifolds. In the paper proposing LapSVM, Belkin et al. [13] suggest to solve the optimization problem in the dual form, similar to popular solving techniques for supervised SVMs; the resulting time complexity is $O(n^3)$. Melacci and Belkin suggest solving the optimization problem in its primal form [124]. They combine an early stopping criterion with a preconditioned conjugate gradient, reducing the time complexity to $O(c \cdot n^2)$ for some $c$ which is empirically shown to be significantly smaller than $n$. Qi et al. suggest to extend *twin SVMs*, which optimize two SVM-like objective functions to yield two nonparallel decision boundaries (one for each class) [88], to include the LapSVM regularization term [141]. Sindhwani et al. extend manifold regularization to the co-regularization framework [158, 159] (see Section 2.3.2). They propose to construct two classifiers using an objective function similar to the objective function of LapSVM for two different views. Niyogi provides some theoretical analysis on the manifold regularization framework, and analyzes its usefulness in semi-supervised learning [132].

Zhu and Lafferty propose to incorporate a manifold regularization term in a generative model [222]. They express the data-generating distribution as a mixture model, where the manifold is locally approximated by a mixture model component. The loss function they optimize consists of a regularizer over the graph and a generative component. Weston et al. incorporate a manifold regularization term in deep neural networks [194]. They propose several methods to incorporate the manifold structure using an auxiliary *embedding* task, which encourages the latent representations in the neural network to be similar for similar samples. Furthermore, they suggest to include a regularization term that explicitly pushes the latent representations of nonsimilar data points (defined as not being neighbors in the underlying graph) further apart. This approach is applied to hyperspectral image classification by Ratle et al. [144].

The graph construction process is nontrivial, and involves many hyperparameters. For instance, one can use different ways of determining edge weights, but also different connectivity criteria ($k$-NN, for instance, is a popular choice). This makes the performance of manifold regularization methods highly dependent on hyperparameter settings. Geng et al. attempt to overcome this problem by first selecting a set of candidate Laplacians using different hyperparameter settings. They then pose the optimization problem as finding the linear combination of Laplacians that minimizes the manifold regularization objective [64]. Formally, let there be $m$ candidate Laplacians $L_1, \dots, L_m$. Assume that the optimal manifold $L^*$ lies in the convex hull of $L_1, \dots, L_m$, i.e., $L^* = \sum_{j=1}^{m} \mu_j \cdot L_j$ with $\sum_{j=1}^{m} \mu_j = 1$ and $\mu_j \geq 0$ for $j = 1, \dots, m$. Since each $L_j$ is a valid graph Laplacian, their linear combination is a valid graph Laplacian as well. Using exponential weights in the Laplacian, the manifold regularization term $||f||_I^2$ then becomes

$$\begin{aligned} ||f||_I^2 &= \mathbf{f}^\intercal \cdot L \cdot \mathbf{f} && (2.16) \\ &= \mathbf{f}^\intercal \cdot (\sum_{j=1}^{m} \mu_j L_j) \cdot \mathbf{f} && (2.17) \\ &= \sum_{j=1}^{m} \mu_j \cdot ||f||_{I(j)}^2, && (2.18) \end{aligned}$$

where $||f||_{I(j)}^2$ is the manifold regularization term for candidate Laplacian $L_j$. This final regularization term is then used in the original optimization problem from Equation 2.15, with the addition of a regularization term $||\mu||^2$ to prevent the optimizer from overfitting to one manifold, and the constraint that $\sum_{j=1}^{m} \mu_j = 1$. The objective function is then optimized with respect to $\mu$ and $f$, which Geng et al. propose to do in an EM-like fashion (i.e., fixing one and optimizing the other alternatingly). Their approach, which they call *ensemble manifold regularization*, is shown to be superior to LapSVM when applied to the SVM objective function on both synthetic and real-world data sets [64].

Aside from the method proposed by Geng et al. [64], graph construction methods have mainly been studied in the context of transductive semi-supervised learning. We cover these methods extensively in Section 2.6.

### Manifold approximation

Manifold regularization techniques introduce a regularization term that directly uses the fact that manifolds locally represent lower-dimensional Euclidean space. However, one can also consider a two-stage approach, where the manifold is first explicitly approximated and then used in a classification task. This is the approach taken by *manifold approximation* techniques, which construct an explicit representation of the manifold. We note that such approaches have a close relation to, and can in some cases even be considered as, semi-supervised preprocessing (see Section 2.4).

Rifai et al. propose such an approach, where the manifolds are first estimated using *contractive autoencoders* (CAE, [147]), after which a supervised training algorithm is applied to use the extracted manifolds [146]. CAEs are a variant of autoencoders that, in addition to the normal reconstruction cost term in autoencoders, penalize the derivatives of the output activations with respect to the input values. By doing so, they penalize the sensitivity of the learned features to small perturbations in the input without relying on sampling these perturbations (like denoising autoencoders do). Rifai et al. claim that CAEs do not merely penalize sensitivity to small perturbations in the input, but that they in fact penalize small perturbations of the input data along the manifold [147]. They argue that this effect occurs due to the balance of promoting reconstruction and penalizing sensitivity to inputs. In other words, they claim to act directly on the manifold assumption.

The loss function $\mathcal{L}$ utilized by contractive autoencoders with reconstruction cost

$\ell(\cdot, \cdot)$ is

$$\mathcal{L} = \sum_{i=1}^{n} [\ell(g(h(\mathbf{x}_i)), y_i)] + \lambda \cdot ||J||_F^2, \tag{2.19}$$

where $||J||_F$ is the Frobenius norm of the Jacobian matrix of the outputs with respect to the inputs, i.e., the sum of the squared partial derivatives of each output activation with respect to each input value. Rifai et al. additionally propose to penalize the Hessian of the output values. Due to the computational complexity of its explicit calculation, they propose to approximate it as the difference between the Jacobians corresponding to small perturbations of the input.

Using *singular value decomposition*, they estimate the tangent plane at each input point to approximate the actual manifolds. As a result, the distance between two data points along the manifold can be estimated, and can be used in classification via, for example, $k$-nearest neighbors. Additionally, they suggest to use a deep neural network pre-trained with multiple, stacked contractive autoencoders, where an additional term is added to the loss function, explicitly penalizing sensitivity of the outputs to perturbations along the tangent plane.

A manifold can be described as a collection of overlapping *charts*, each having a simple geometry, that jointly cover the entire manifold. Such a collection of charts is known as an *atlas*. Pitelis et al. suggest to approximate these charts explicitly, associating each with an affine subspace [139, 140]. They alternate between assigning data points to charts, and choosing the affine subspace best matching the data for each chart. The charts are initialized by using principal component analysis on a set of random subspaces. From this, they obtain both a set of charts and a soft assignment of points to charts (since points can be associated with more than one chart). From these charts and soft assignments, kernels are generated that are then used in SVM-based supervised learning.

### 2.5.4 Generative models

The aforementioned methods are all *discriminative*: their only goal was to infer a function that can classify data points. In some cases, they yield probabilistic predictions; in others, they only yield the most likely class to assign. In all cases, they approach the classification problem without explicitly modelling any of the data-generating distributions. In contrast, the primary goal of methods based on *generative* models is to model the process that generated the data: it explicitly constructs a model of the input space. When the generative model is conditioned on the label $y$, it can also be used for classification.

#### Mixture models

If prior knowledge about $p(x, y)$ is available, generative models can be very powerful. For instance, consider the case where we know that our data $p(x, y)$ is comprised of a mixture of $k$ Gaussian distributions, each of which corresponds to a certain class. Most discriminative methods would not be able to properly incorporate this prior information. Instead, one would be best served by simply fixing the model as a mixture of $k$

Gaussian components. Each component $j = 1, \ldots, k$ has 3 parameters: a weight $\pi_j$ (where $\sum_{j=1}^{k} \pi_j = 1$), mean vector $\boldsymbol{\mu}_j$, and covariance matrix $\boldsymbol{\Sigma}_j$; the most likely parameters can then be inferred (e.g., via *expectation-maximization* (EM) [53]). This model is generative: it models the distribution $p(x, y)$, from which samples $(\mathbf{x}, y)$ can be drawn. The model can then also be used for classification: since the inference procedure yields an estimate $\hat{p}(x|y)$ of each distribution $p(x|y)$, one can simply assign the class $c$ to an unlabeled sample $\mathbf{x}_i \in X_U$ that maximizes $\hat{p}(\mathbf{x}_i|y_i = c) \cdot p(y_i = c)$. Note that, in the case of Gaussian mixture models described earlier, $p(y_i = c) = \pi_c$.

The application of mixture models to generative modelling comes with several caveats [44, 216]. First of all, the mixture model should be identifiable: each distinct parameter choice for the mixture model should determine a distinct joint distribution, up to a permutation of the mixture components. Secondly, mixture models hinge on the critical assumption that the assumed model is correct. If the model is not correct, i.e., the true distribution $p(x, y)$ cannot be modeled by the assumed model, unlabeled data may hurt performance rather than improve it.

In real-world data, the model correctness assumption rarely holds. Therefore, using mixture models for generative modelling can prove difficult. Some approaches exist to mitigate these problems (e.g., by varying the influence of unlabeled data in EM [131]). However, the rigidity of mixture models has caused attention to shift to more flexible generative models.

### Generative adversarial networks

In recent years, a new type of learning paradigm known as *generative adversarial networks* (GAN) has been proposed for simultaneous construction of generative and discriminative learners [70]. Generally implemented using neural networks, they simultaneously train a generative model, tasked with generating samples that are difficult to distinguish from samples stemming from the true distribution, and a discriminative classifier, tasked with predicting whether samples are generated from the true distribution ("real" samples) or not ("fake" samples). The generator can then be used to produce novel samples that closely resemble those obtained from the true distribution.

The discriminator $D$ with parameters $\boldsymbol{\theta}^{(D)}$ and generator $G$ with parameters $\boldsymbol{\theta}^{(G)}$ are trained simultaneously to optimize a single objective function. Crucially, the discriminator's goal is to *minimize* the objective function, whereas the generator's goal is to *maximize* it. The discriminative function $D$ expresses the probability that a sample $\mathbf{x}$ is a true sample; the generative function $G$ generates a sample $\mathbf{x}$ from a noise vector $\mathbf{z}$ sampled from some distribution $p_{\mathbf{z}}(\mathbf{z})$. The optimization problem then consists of two terms. The first term expresses the discriminator's ability to identify true samples as such, and its optimization involves only the discriminator. The second term expresses the discriminator's ability to identify fake samples, and its optimization involves both the discriminator and the generator. Formally, this optimization problem amounts to determining

$$\min_{G} \max_{D} \quad V(D, G) = \mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))], \tag{2.20}$$

where the parametrizations of $D$ by $\boldsymbol{\theta}^{(D)}$ and $G$ by $\boldsymbol{\theta}^{(G)}$ are omitted for conciseness.

The generator and the discriminator are trained simultaneously. In each training step, multiple true samples are taken from the training data, and multiple fake samples are generated using $G$ by sampling from $p_{\mathbf{z}}(\mathbf{z})$. The discriminator's and generator's respective parameters $\boldsymbol{\theta}^{(D)}$ and $\boldsymbol{\theta}^{(R)}$ are then adjusted to simultaneously optimize the empirical objective function using gradient descent [68].

GANs are naturally unsupervised: they train a generative model on unlabeled data, using a discriminative classifier to assess the quality of the generator. However, extensions exist to support classification in GANs. Proposed but not implemented in the original GAN paper [70], these methods also use a generator and a discriminator, but train the discriminator to identify different classes instead of only distinguishing true samples from fake samples. As such, GANs naturally extend to the semi-supervised case: the purely discriminative component of the loss term (the first term in Equation 2.20) can easily be extended to incorporate the sample's true label when it is known.

Starting from the perspective of clustering, Springenberg proposes *CatGAN*, a semi-supervised extension of GANs [163]. They suggest extending the discriminator to incorporate label information for the available labeled samples. Instead of the two outputs in the original formulation of GANs, they propose to use $K$ outputs for the discriminator, corresponding to the $K$ possible clusters/classes. The discriminator is then trained to (1) be certain of class assignment for true samples and to (2) be uncertain of class assignments for fake samples. The generator is trained to (1) generate samples that obtain highly certain class assignments from the discriminator and to (2) assign equal probability to all classes in the generative process. The resulting discriminator can be used to cluster data points into $K$ different clusters. Incorporating labeled data and thus extending CatGAN to the semi-supervised setting, Springenberg sets $K = |\mathcal{Y}|$ and extends the discriminator's loss function to include the cross-entropy loss for labeled samples [163].

Salimans et al. extend GANs to the semi-supervised setting by using $|\mathcal{Y}|+1$ outputs [152]. Outputs $1, \ldots, |\mathcal{Y}|$ correspond to the individual classes; output $|\mathcal{Y}| + 1$ corresponds to "fake" samples. The loss function is adapted to include the cross-entropy loss of the prediction given the true label for the labeled samples. The rest of the loss function does not need to be changed significantly: when presented with an unlabeled sample, the discriminator's estimate of the sample being a "true" sample can be calculated as $\sum_{c=1}^{|\mathcal{Y}|} D_c(\mathbf{x})$ for sample $\mathbf{x}$, where $D_c(\mathbf{x})$ is the value of output $c$ for the discriminator. Odena independently proposed the same idea around the same time [133].

For an extensive overview of GANs, their applications and their extensions, we refer the reader to the notes of Goodfellow's 2016 NIPS tutorial on GANs [68].

### Variational autoencoders

Aside from GANs, further efforts have been made towards constructing semi-supervised deep generative models in recent years. One notable example is the proposal of *variational autoencoders* (VAEs) and their application to semi-supervised learning.

Proposed by Kingma and Welling, variational autoencoders are a type of latent variable model, treating each data point $\mathbf{x}$ as being generated from a vector of latent variables $\mathbf{z}$ [95]. Traditional latent variable models such as autoencoders generally yield a model with a highly complex distribution $p_{\mathbf{z}}(\mathbf{z})$, which makes it very difficult to use them for sampling. Crucially, VAEs assert that the distribution over latent variables is a very simple one: a standard, multivariate Gaussian distribution ($p_{\mathbf{z}}(\mathbf{z}) = \mathcal{N}(0, I)$). The possible transformation of the simple $p_{\mathbf{z}}(\mathbf{z})$ to a more complex distribution of latent variables is then left to a deep neural network. The outputs of the neural network form the parameters of a distribution $p_{\mathbf{x}|\mathbf{z}}(\mathbf{x}|\mathbf{z})$. Again, this is asserted to be a simple distribution, such as a Gaussian with the neural network's outputs as its mean vector and covariance matrix $\sigma I$ (where $\sigma$ is a hyperparameter). This part of the VAE is called the *decoder*: given a latent vector $\mathbf{z}$, it generates a distribution $p_{\mathbf{x}|\mathbf{z}}(\mathbf{x}|\mathbf{z})$ parameterized by the outputs of the neural network.

To train the network, an *encoder* is introduced to model $p_{\mathbf{z}|\mathbf{x}}(\mathbf{z}|\mathbf{x})$. Again, it is assumed to be some simple distribution like a Gaussian with diagonal covariance matrix, parametrized by another neural network. The neural networks are jointly trained, minimizing a combined loss function consisting of (1) the KL divergence between the posterior $p_{\mathbf{z}|\mathbf{x}}(\mathbf{z}|\mathbf{x})$ and the prior $p_{\mathbf{z}}(\mathbf{z})$ and (2) the reconstruction loss of the autoencoder's output for input samples. For brevity, we do not go into too much detail regarding the training procedure of VAEs, which includes a nontrivial backpropagation step due to the sampling procedure; instead, we refer the reader to the VAE tutorial by Doersch [57].

Kingma et al. propose a two-step model to use VAEs for semi-supervised learning [94]. In the first step, a VAE is trained on both the unlabeled and labeled data to extract meaningful latent representations from data points. By itself, this can be seen as an unsupervised preprocessing step, allowing the latent representations to be used by any supervised classifier. In the second step, they implement a VAE in which the latent representation is augmented with the sample label vector $\mathbf{y}_i$, which is the one-hot encoded true label for labeled samples and is treated as an additional latent variable for unlabeled samples. In addition to the decoder, a classification network is introduced that infers the label predictions [94].

## 2.6 Transductive methods

The semi-supervised learning methods described in the previous sections were all inductive algorithms: their primary goal was to use both labeled and unlabeled data to construct a prediction model able to provide label predictions for data points in the entire input space. In inductive learners, we can therefore clearly distinguish between a training phase and a testing phase: in the training phase, labeled data $(X_L, \mathbf{y}_L)$ and unlabeled data $X_U$ are used to construct a classifier. In the testing phase, this resulting classifier is used to independently classify the unlabeled or other, previously unseen data points.

In this section, we discuss the second major class of semi-supervised learning methods: *transductive* algorithms. Unlike inductive algorithms, transductive algorithms do not produce a prediction model that is defined over the entire input space. Rather, transductive methods only yield a set of predictions for the set of unlabeled data points provided to the learning algorithm. Unlike for inductive methods, we thus cannot distinguish between a training phase and a testing phase: transductive algorithms are provided with labeled data $(X_L, \mathbf{y}_L)$ and unlabeled data $X_U$, and output predictions $\hat{\mathbf{y}}_U$

for the unlabeled data.

Transductive methods typically define a graph over all data points, both labeled and unlabeled, encoding the pairwise similarity of data points with possibly weighted edges [217]. An objective function is then defined and optimized to achieve two goals:

1. The predicted labels for labeled samples should match the true labels.
2. Similar data points, as defined via the similarity graph, should have the same label predictions.

In other words, these methods encourage consistent predictions for similar data points while taking into account the known labels. It is clear that a close similarity exists between these methods and the inductive manifold-based methods from Section 2.5.3. Both methods construct a graph over the data points and use pairwise similarity between data points to approximate more complex structures. The only major difference between them is that the inductive methods seek to obtain a classification model defined over the entire input space, whereas transductive methods only yield predictions for the provided unlabeled data points. Collectively, these methods are often referred to as *graph-based* methods [216].

In Section 2.5.3, we focused on the interpretation and motivation of graph-based techniques from the theoretical perspective of manifolds. The motivation for transductive graph-based methods, however, has generally been driven directly by the two optimization criteria outlined above. This section, in which we discuss transductive semi-supervised learning, follows that line of reasoning.

## 2.6.1 A general framework for graph-based methods

Graph-based semi-supervised learning methods generally involve three separate steps: graph creation, graph weighting, and inference [89, 117]. In the first step, nodes (representing data points) in the graph are connected to each other based on some similarity measure. In the second step, the constructed edges are weighted, yielding a weight matrix. The first two steps are commonly referred to jointly as the graph construction phase. After graph construction, we have a graph consisting of a set of nodes $V = \{v_1, \dots, v_n\}$, corresponding to the data points, and an $n \times n$ weight matrix $W$ containing the edge weights for all node pairs. An edge weight of 0 indicates that no edge is present. In the remainder of this section, we use the terms *node* and *data point* interchangeably in the context of graph-based methods.

Once the graph is constructed, it is used to form predictions $\hat{\mathbf{y}}_U$ for the unlabeled data points. The general form of objective functions for transductive graph-based methods contains one component for penalizing predicted labels that do not match the true label and another component for penalizing differences in the label predictions for connected data points. Formally, given a supervised loss function $\ell$ for the labeled data and an unsupervised loss function $\ell_U$ for pairs of labeled or unlabeled data points, transductive graph-based methods attempt to find a labeling $\hat{\mathbf{y}}$ that minimizes

$$\lambda \cdot \sum_{i=1}^{l} \ell(\hat{y}_i, y_i) + \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \cdot \ell_U(\hat{y}_i, \hat{y}_j), \tag{2.21}$$

where $\lambda$ governs the relative importance of the supervised term. Furthermore, some graph-based methods impose an additional unary regularization term on the unlabeled predictions. This general framework for graph-based methods allows for a multitude of variations in each of its steps. The formulation is commonplace in graph-based methods, and most graph-based inference algorithms can be shown to fit into this framework [15, 167]. It is also present in the manifold regularization framework by Belkin et al. [12] which was discussed in Section 2.5.3.

The spectrum of graph-based semi-supervised learning methods can be effectively structured based on the different approaches in the two main phases, i.e., *graph construction* and *inference*. Early work on graph-based methods focused mainly on the second problem, leaving graph construction as a scarcely studied topic. Zhu noted that this imbalance might be unjust, and that graph construction can have a significant impact on classifier performance [216]. Later work has addressed this imbalance, and graph construction has since become an area of substantial research interest [51].

Graph-based transductive methods were introduced in the early 2000s, and graph-based inference methods were particularly intensively studied during the subsequent decade. As such, a large portion of the significant research that has been conducted in this field is covered in the semi-supervised learning survey by Zhu [216]. Furthermore, Subramanya and Talukdar recently published a book on graph-based methods [167]. These studies cover the literature on graph-based methods well. In light of this, we have chosen to base our review of earlier work in this area heavily on the book by Subramanya and Talukdar, Zhu's survey, and Zhu's doctoral thesis [217]. Following the general chronological order of research in the field of graph-based methods, we begin by outlining different approaches to solving the inference problem. After that, we provide an overview of the research conducted on graph construction.

### 2.6.2 Inference in graphs

The inference process in transductive methods consists of forming predictions $\hat{\mathbf{y}}_U$ for the unlabeled data points $X_U$. If the predicted labels of the labeled samples are not fixed to the true labels in the inference process, the optimization considers the entire set of predicted labels $\hat{\mathbf{y}}$.

A multitude of approaches has been suggested for tackling the optimization of Equation 2.21. Generally, the optimization methods are dependent on the specific choices of the loss functions $\ell$ and $\ell_U$ and the trade-off parameter $\lambda$. Furthermore, some methods infer only the most likely label assignment $\hat{\mathbf{y}}$, while others infer marginal probabilities $p(y_i = c)$ of assigning a particular label $c$ to individual samples. Combined, these variations give rise to a plethora of different perspectives on the graph-based inference problem.

Although the general objective function from Equation 2.21 applies to the multi-class setting as well, many graph-based methods do not naturally extend beyond binary classification. The inference methods we discuss mostly consider the binary classification case. In discussing binary graph-based methods, we assume $\mathcal{Y} = \{-1, 1\}$ unless explicitly stated otherwise.

**Hard label assignments: graph min-cut**

The first graph-based semi-supervised classification method was proposed by Blum and Chawla, who use a min-cut-based method for binary classification [21]. They experiment with graph construction using $k$-nearest neighbors and the $\epsilon$-neighborhood (connecting pairs of data points with distance smaller than $\epsilon$). They keep the edge weights fixed and uniform, but experiment with changing the weight of edges between unlabeled data points relative to other edges.

Once the graph is constructed, they treat the optimization problem from a min-cut perspective. They add a single source node $v_+$, connected with infinite weight to the positive samples, and a single sink node $v_-$, connected with infinite weight to the negative samples. Finding the minimum cut, then, corresponds to finding the set of edges with minimal combined weight that, when removed, result in a graph with no paths from the source node to the sink node. All unlabeled nodes in the resulting graph that are in the component containing $v_+$ are labeled as positive, and all unlabeled nodes that are in the component containing $v_-$ are labeled as negative.

The min-cut approach can be seen to minimize the general objective function of Equation 2.21 as $\lambda$ approaches infinity (fixing the predictions on labeled data points to their true labels) and $\ell_U(\hat{y}_i, \hat{y}_j) = \mathbb{1}_{\{\hat{y}_i \neq \hat{y}_j\}}$, where $\mathbb{1}$ is the indicator function. Note that, assuming labels 0 and 1 are used, the loss function for unlabeled samples corresponds to quadratic cost, i.e., $\mathbb{1}_{\{\hat{y}_i \neq \hat{y}_j\}} = (\hat{y}_i - \hat{y}_j)^2$. We can write the corresponding objective function as

$$\lambda \cdot \sum_{i=1}^{l} (\hat{y}_i - y_i)^2 + \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \cdot (\hat{y}_i - \hat{y}_j)^2. \tag{2.22}$$

Note that this objective function can be written in an alternate form, using the graph Laplacian $L = D - W$ (where $D$ is the diagonal matrix containing the degree for node $i$ at $D_{ii}$) as follows:

$$\lambda \cdot \sum_{i=1}^{l} (\hat{y}_i - y_i)^2 + 2 \cdot \hat{\mathbf{y}}^\intercal \cdot L \cdot \hat{\mathbf{y}}. \tag{2.23}$$

Pang and Lee use the min-cut for classification in the context of sentiment analysis [135]. They note that, instead of fixing the predicted labels of labeled data to their true labels, one can also assign finite weight to the connection between labeled data points and $v_+$ and $v_-$, indicating confidences in either classification from the perspective of the individual data point.

The min-cut approach can easily lead to degenerate cuts, yielding a solution where almost all unlabeled samples are in the same component. This behavior originates from the fact that more balanced cuts generally have more potential edges to cut: when a cut yields a split into negative nodes $V^-$ and positive nodes $V^+$, the number of edges to cut is potentially $|V^+||V^-|$. Joachims proposes to normalize the objective function of min-cut based on this potential number of edges being cut [91]. Spectral methods are then used to solve the resulting optimization problem.

Since the optimization of the min-cut algorithm optimizes over a binary vector, no probabilities can be extracted from this optimization. To solve this problem, Blum et al. propose to construct an ensemble of min-cut classifiers, each finding the minimum cut on a randomly perturbed version of the constructed graph, adding noise to the edge weights [22]. The prediction probabilities are then simply calculated by the fraction of classifiers predicting a label.

**Probabilistic label assignments: Markov random fields**

The lack of a principled, efficient way of estimating classification probabilities is a fundamental disadvantage to the min-cut approach to graph-based inference. In many cases, we wish to estimate the probability $p(y_i = c)$ that an unlabeled sample $\mathbf{x}_i$ has label $c$. Standard min-cut, however, only provides hard classifications (i.e., it only outputs class labels and no probabilities). Approaching graph-based methods from the perspective of Markov random fields provides a potential solution to this problem. Note that in the following paragraph, in a slight abuse of notation, we use $X$ and $x$ to denote (realizations of) random variables rather than data points.

The Hammersley-Clifford theorem states that a probability distribution $P(X)$ for random variables $X_1, \ldots, X_n$ corresponds to a Markov random field if a graph $G$ exists such that the joint probability density $P(X = x)$ can be factorized over the (maximal) cliques of $G$ [80]. In other words, $X$ corresponds to a Markov random field formed by $G$ if

$$P(X = x) = \frac{1}{Z} \prod_{c \in C_G} \psi_c(x_c), \tag{2.24}$$

where $Z$ is a normalization constant, $C_G$ is the set of cliques in $G$, $\psi_c$ is an arbitrary function, and $x_c$ is the subset of random variables in clique $c$.

Using the Hammersley-Clifford theorem, we can show that the general minimization for graph-based methods, formulated in Equation 2.21, can be expressed in the form of a Markov random field. Let $G$ denote the graph with weight matrix $W$ obtained in the graph construction phase, and let $\hat{Y} = \{\hat{Y}_1, \ldots, \hat{Y}_n\}$ be random variables corresponding to the predicted labels (i.e., 0 or 1) for samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$. We extend $G$ by connecting each node $\hat{Y}_i$ corresponding to a labeled sample $\mathbf{x}_i$ to an auxiliary node $Y_i'$, corresponding to a random variable which can only attain the true label $y_i$. We denote the entire set of random variables, or nodes, as $Y = \hat{Y} \cup Y'$, where $Y'$ is the set of all auxiliary nodes. Since the auxiliary nodes can only attain the corresponding true label, $p(Y = \mathbf{y}) = p(\hat{Y} = \hat{\mathbf{y}})$, where $\hat{\mathbf{y}}$ is the set of predictions for our (labeled and unlabeled) samples.

This situation is depicted in Figure 2.3. The filled nodes $\hat{Y}$ and the edges between them correspond to the original graph $G$; the unfilled nodes with plus- and minus-signs in them represent the auxiliary nodes $Y'$, and are connected only to the corresponding filled node.

Recall that a clique is a subset of nodes where all pairs of nodes are connected to each other by an edge. A maximal clique, then, is a clique that cannot be expanded, i.e., to which no nodes can be added such that the resulting subset of nodes also forms a clique. We note that every pair of nodes that is connected by an edge is part of at least

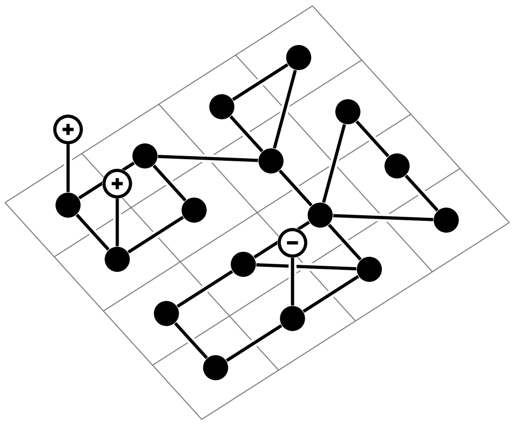


Figure 2.3: Example of an undirected graphical model for graph-based classification. Filled nodes and edges between them correspond to the original graph $G$. Unfilled nodes with plus- and minus-signs correspond to auxiliary nodes to labeled data.

one clique. Thus, if we can find an expression of the form

$$\frac{1}{Z} \prod_{(u,v)\in E} \psi_{\{u,v\}}(\{u,v\}) \tag{2.25}$$

for $P(\hat{Y} = \hat{\mathbf{y}})$, the probability distribution corresponds to a Markov random field. We proceed to show that we can express the cost function from Equation 2.21 in a way such that minimizing it corresponds to maximizing the probability $P(\hat{Y} = \hat{\mathbf{y}})$. We can distinguish between two different types of edges: those between two normal nodes $u, v \in \hat{Y}$, and those between a normal node and its auxiliary node ($u \in \hat{\mathbf{Y}}, v \in \mathbf{Y}'$, or the other way around). Let us define $\psi(\cdot)$ for these two cases independently:

$$\psi(\{\hat{y}_i, \hat{y}_j\}) = \exp(-W_{ij} \cdot \ell_U(\hat{y}_i, \hat{y}_j)) \qquad \text{if } v_i, v_j \in \hat{Y}, \tag{2.26}$$

$$\psi(\{\hat{y}_i, y'_i\}) = \exp(-\ell(\hat{y}_i, y'_i)) \qquad \text{if } v_i \in \hat{Y}, v_j \in Y'. \tag{2.27}$$

The unnormalized probability $Z \cdot P(\hat{Y} = \hat{\mathbf{y}})$ then becomes

$$\prod_{(u,v)\in E} \psi_{\{u,v\}}(\{u,v\}) = \exp\left( - \sum_{y'_i \in Y'} \ell(\hat{y}_i, y'_i) - \sum_{\hat{y}_i, \hat{y}_j \in \hat{Y}} \ell_U(\hat{y}_i, \hat{y}_j) \right). \tag{2.28}$$

The normalization constant $Z$ can be calculated by summing over all possible configurations of $Y$. Although this is computationally too expensive for all practical purposes, the normalization constant is irrelevant in the context of maximum-likelihood estimation. The negative logarithm of the unnormalized probability, then, is exactly equal to the general loss function for graph-based methods from Equation 2.21. Maximizing the probability $P(Y)$, we obtain the *mode* of the Markov random field, i.e., its most likely configuration $\mathbf{y}$. This solution is exactly the solution found when minimizing the min-cut objective [21].

In the inductive semi-supervised classification setting, classifier predictions are independent, i.e., $p(\mathbf{y}) = p(y_1)p(y_2)\cdots p(y_n)$. In transductive, graph-based methods, however, this is generally not the case: predictions are dependent on each other. The most probable label assignment $\mathbf{y}$, thus, generally does not correspond to the label assignment minimizing the expected error rate. To find the latter, each sample $\mathbf{x}_i$ would have to be assigned the label $c$ that maximizes the marginal probability $p(y_i = c)$. Unfortunately, finding the marginal probabilities of a random field is not trivial.

Zhu and Ghahramani attempt the calculation of the marginal probabilities via *Markov chain Monte Carlo* (MCMC) sampling [219]. Experimenting with Metropolis and Swendsen-Wang sampling, they report lacking computational efficiency. Getz et al. use the multicanonical MCMC method to compute the marginal probabilities [157].

**Efficient probabilistic label assignments: Gaussian random fields**

There is no closed-form solution for calculating the marginal probabilities in the Markov random field with binary labels described before. However, when the random variables $\hat{Y}$ are relaxed to the space of real numbers, a closed-form solution does exist. This approach is proposed by Zhu et al., who fix the labels of the labeled samples and use quadratic cost for the pairs of predictions $\hat{y}_i, \hat{y}_j \in \mathbb{R}$ [220]. The resulting objective function is identical to the objective function used in the min-cut formulation (see Equation 2.22), except for the relaxation of the predictions to the space of real numbers.

Using real-valued predictions with a quadratic loss function, the exponential form for $p(\hat{Y} = \hat{\mathbf{y}})$ is a multivariate Gaussian distribution. Thus, a closed-form solution for the mode of the field, which equals its mean, exists. Furthermore, the marginal probability distributions $p(y_i = c)$ are Gaussian as well, allowing for computation of the label predictions minimizing the error rate. This is why the random field is called a *Gaussian random field*.

Recall from Section 2.5.3 that we defined the graph Laplacian as $L = D - W$, where $D$ is the degree matrix (i.e., a diagonal matrix with the vertex degrees at the diagonal). Zhu et al. show that the prediction function is *harmonic*, i.e., $L\hat{\mathbf{y}}$ equals 0 at unlabeled data points, and is equal to the true label at labeled data points [220]. The predicted label at each unlabeled data point is equal to the average of the predictions of its neighbors, i.e.,

$$\hat{y}_i = \frac{1}{D_{ii}} \sum_{v_j \in \mathcal{N}(v_i)} W_{ij} \cdot \hat{y}_j, \quad \text{for } i = l+1, \ldots, n, \tag{2.29}$$

where $\mathcal{N}(v)$ denotes the neighborhood of node $v$, that is, $\mathcal{N}(v) = \{u \in V : (u,v) \in E\}$. Furthermore, the solution is unique and $\hat{y}_i \in [0,1]$ for each $i$. Thus, label predictions can be easily obtained from the solution using thresholding.

Computation of the marginals of the Markov random field involves an inversion of the submatrix $L_U$ corresponding to the unlabeled data points in the graph Laplacian. This is computationally expensive for large numbers of unlabeled data points. Several other approaches have been proposed for finding the solution to the harmonic function, including loopy belief propagation and a conjugate gradient method [220].

Before proposing the Gaussian random fields approach to graph-based methods, Zhu et al. introduced the *label propagation* algorithm for inference on graphs [218]. It is an iterative algorithm that computes soft label assignments $\hat{y}_i \in \mathbb{R}$ by pushing (*propagating*) the estimated label at each node to its neighboring nodes based on the edge weights. In other words, the new estimated label at each node is calculated as the weighted sum of the labels of its neighbors. In matrix notation, let

$$A_{ij} = \frac{W_{ij}}{\sum_{v_k \in \mathcal{N}(v_i)} W_{ik}} \tag{2.30}$$

denote the transition matrix. The label propagation algorithm then consists of two steps, which are repeated until the label assignment $\hat{\mathbf{y}}$ converges. Starting with initial label assignment $\hat{\mathbf{y}}$, which is random for the unlabeled data points and clamped to the true labels for the labeled data points:

1. Propagate labels from each node to the neighboring nodes: $\hat{\mathbf{y}} = A^\intercal \cdot \hat{\mathbf{y}}$.
2. Clamp the predictions of the labeled data points to the corresponding true labels.

As Zhu et al. show, the algorithm converges to the harmonic function solution described earlier, and is guaranteed to converge [15, 217]. They also show that the label propagation approach can be interpreted as a random walk with transition matrix $A$, which stops when a labeled node is hit. Wu et al. cast this procedure in a framework they call *partially absorbing random walks* where they, instead of deterministically stopping when a labeled node is hit, stochastically determine whether to stop (*absorb*) or continue the random walk [198]. The label propagation approach is closely related to the *Markov random walks* approach by Szummer and Jaakkola [169]. Belkin et al. consider a similar objective function, and they provide some theoretical analysis [11]. Azran proposed a random walk approach where walks originate in unlabeled nodes, and the labeled nodes are absorbing states. The probability that an unlabeled data points attains a certain label is then derived from the probability that a walk starting from the unlabeled node ends up in a labeled node of the corresponding class as the random walk length approaches infinity [4].

### Handling label noise and irregular graphs: local and global consistency

The Gaussian random fields method has two drawbacks [167]. Firstly, since the true labels are clamped to the labeled data points, it does not handle label noise well. Secondly, in irregular graphs, the influence of nodes with a high degree is relatively large. An approach closely related to the Gaussian random fields method that addresses these two issues is proposed by Zhou et al. [210]. It is commonly known as the *local and global consistency* (LGC) method, referring to the observation that graph-based methods promote consistency of labels on manifolds (global) and nearby in the input space (local).

They address the first issue by refraining from clamping the true labels to the labeled data points. Instead, they penalize the squared error between the true label and the estimated label. They address the second issue by regularizing the penalty term for unlabeled data points by the node degrees. Furthermore, they regularize the predictions for the unlabeled data points by pulling them towards zero [15]. We can write the corresponding objective function in the general form as

$$\sum_{i=1}^{l}(\hat{y}_i - y_i)^2 + \sum_{i=l+1}^{n} \hat{y}_i^2 + \lambda_U \cdot \sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij} \cdot \left(\frac{\hat{y}_i}{\sqrt{D_{ii}}} - \frac{\hat{y}_j}{\sqrt{D_{jj}}}\right)^2, \tag{2.31}$$

where $\lambda_U$ governs the weight of the penalization of inconsistencies in label predictions between neighbors in the graph.

Note that, like for the min-cut and MRF objectives, the last term of the objective function can be expressed using matrix notation. The only difference in that term is that LGC uses the normalized graph Laplacian $\tilde{L} = D^{-\frac{1}{2}} \cdot L \cdot D^{-\frac{1}{2}}$ instead of the unnormalized Laplacian $L = D - W$. Like Gaussian random fields, this formalization admits a closed-form solution and a relatively efficient iterative approach to optimization. In this algorithm, the label vector $\hat{\mathbf{y}}_{t+1}$ at iteration $t+1$ is calculated based on the label vector at iteration $t$, using the update rule

$$\hat{\mathbf{y}}_{t+1} = \alpha \cdot \tilde{L} \cdot \hat{\mathbf{y}}_t + (1-\alpha) \cdot \mathbf{y}, \tag{2.32}$$

where $\mathbf{y}$ is 0 for the unlabeled data points, and $\alpha$ governs the relative importance of the calculated label vector versus the base label vector $\mathbf{y}$. This algorithm is often referred to as *label spreading*.

### Further research on graph-based inference

The previously described approaches, and in particular label propagation, have been the *de facto* standard approach to the inference phase in graph-based semi-supervised classification. Several variants and extensions to the approach have been proposed, which we briefly summarize here.

Baluja et al. apply graph-based methods to recommender systems (in particular, video suggestions to users) by considering a random-walk-based method [8]. They propose *adsorption*, a heuristic algorithm for predicting the label $\hat{y}_i$ of node $i$ by performing a random walk starting at node $v_i$. At each step in the random walk, the walk can either continue to the next step (*continue*), accept the label of a labeled node as the prediction (*injection*), or explicitly predict no label (*abandonment*). The last option corresponds to a dummy prediction, which specifically indicates that the transducer is unable to output a confident prediction. The option chosen by the algorithm is dependent on two hyperparameters, governing the relative frequencies of the three options. Heuristic approaches to hyperparameter selection have been proposed by Baluja et al. [8] and Talukdar et al. [172]. The algorithm has successfully been applied to video recommendation, but is difficult to analyze theoretically due to its many heuristic components. Talukdar and Crammer attempt to analyze the algorithm, and show that no objective function exists that it is optimized by the adsorption process [171]. They propose a modified version of adsorption that does have a corresponding objective function, and propose efficient iterative methods to solve it.

The previously described graph-based methods can be sensitive to class imbalance [216]. Zhu et al. propose to adjust the classification threshold such that the predicted label proportions correspond to predefined label proportions [220]. Wang et al. propose an optimization scheme that is less sensitive to noise in the true labels and that mitigates the problem of sensitivity to class imbalance by altering the influence of labeled samples based on the label proportions [189]. They modify the objective function to optimize over both real-valued predictions and binary label assignments, and penalize difference between the real-valued predictions and the binary predictions. They proceed to optimize the objective function by alternatingly optimizing the real-valued and binary label assignments. In later work, the authors consider the same approach from a graph max-cut perspective [190].

In *structured output learning*, the labels of samples cannot be captured using simple binary or real-valued representations. For instance, the output labels might be better represented with histograms or probability distributions in some cases (e.g., when predicting the relative traffic density at a location over a 24-hour cycle). Subramanya and Bilmes propagate discrete probability distributions through a graph based on the KL-divergence between the distributions of different nodes [165, 166]. As an alternative to KL-divergence, Solomon et al. propose to use the Wasserstein distance to measure the similarity between the discrete distributions of neighboring nodes [162].

## 2.6.3 Graph construction

Arguably, graph construction is the most important aspect of graph-based methods: in order for inference to work, the constructed graph must successfully encode local similarities. In the early days of research in the area of graph-based methods, research was mainly focused on the inference phase, and graph construction was not well-studied [216]. In recent years, however, this has changed. Extensive experiments have been conducted on different graph construction methods, and new methods have been introduced [51, 89, 167].

Since the nodes of the graph correspond to the data points (both labeled and unlabeled), the graph construction phase amounts to forming edges between nodes (yielding the adjacency matrix) and attaching weights to them (yielding the weight matrix). In many cases, the similarity measure governing the connectivity between nodes is also used in the weight matrix.

### Adjacency matrix construction

The first step in constructing the graph is the creation of an adjacency matrix, whose elements indicate the presence of edges between pairs of nodes. Three popular data-independent methods for constructing graphs exist and are outlined below. We note that the first two methods, $\epsilon$-neighborhood and $k$-nearest neighbors, are *local* in the sense that the neighbors of a node can be determined independently for each node. In other words, the construction of $v_i$'s neighborhood does not influence the construction of $v_j$'s neighborhood (unless node $v_i$ is a neighbor of $v_j$). The third method, $b$-matching, on the other hand, optimizes a global objective, and nodes that are far apart can significantly influence each other's connectivity.

**$\epsilon$-neighborhood.** One of the first methods to be used in graph construction was the $\epsilon$-neighborhood method, which simply connects each node to all nodes to which the distance is at most $\epsilon$ [21]. In other words, an edge between $\mathbf{x}_i$ and $\mathbf{x}_j$ is created if and only if $d(\mathbf{x}_i, \mathbf{x}_j) \leq \epsilon$, where $d(\cdot, \cdot)$ is some distance measure (usually the Euclidean distance). Clearly, this method is highly dependent on the choice of $\epsilon$ and the distance measure. Furthermore, since $\epsilon$ is fixed, it does not work well if the scale of patterns in the input data distribution varies. Because of these limitations, the $\epsilon$-neighborhood method is rarely used in practice [51, 89].

**$k$-nearest neighbors.** The most common graph construction method for transductive methods is the $k$-nearest neighbors method, where each node is connected to its $k$ nearest neighbors in the input space according to some distance measure [21]. Using vanilla $k$-nearest neighbors, however, a problem arises: since $k$-nearest neighbors is not symmetric, we do not necessarily obtain the required undirected graph. Thus, some additional processing is necessary to obtain an undirected graph. Two options are commonly considered: one (*symmetric* $k$-nearest neighbors) constructs an edge if $i$ is in the $k$-neighborhood of $j$ or vice versa, and the other (*mutual* $k$-nearest neighbors) constructs an edge if $i$ and $j$ are both in each other's $k$-neighborhood [51]. The difference between the $\epsilon$-neighbors and $k$-nearest neighbors methods has been extensively studied by Maier et al. in the context of clustering methods [122].

**$b$-matching.** The postprocessing step used when constructing the graph with $k$-nearest neighbors generally results in a graph where not all nodes have exactly $k$ neighbors. If *symmetric* $k$-nearest neighbors is used, it often occurs that some nodes have much higher degrees than others. Jebara et al. show that this can negatively impact the final performance of the classifier [89]. They propose an edge construction method that enforces the regularity of the constructed graph, i.e., that each node has the same number of neighbors, and that the nodes have exactly the requested number of edges. The approach they use is inspired by *matching*, a concept from graph theory where one tries to find a subset of edges in a graph such that the edges do not share vertices. In their method, referred to as $b$-matching, the objective is to find the subset of edges in the complete graph such that (1) each node has degree $b$ and (2) the sum of the edge weights is maximized.

Note that, in the original paper by Jebara et al., instead of maximizing the sum of edge weights, the objective is in fact to minimize the sum of the distances between the remaining edges. However, since they define the distance matrix $C$ with $C_{ij} = \sqrt{W_{ii} + W_{jj} - 2W_{ij}}$, these notions are equivalent. The corresponding optimization problem they formulate is:

$$\begin{aligned} \underset{\hat{A}\in\mathbb{B}^{n\times n}}{\text{minimize}} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}\cdot C_{ij} \\ \text{subject to} \quad & \sum_{j=1}^{n} A_{ij} = b \qquad i=1,\ldots,n, \\ & A_{ii} = 0 \qquad i=1,\ldots,n, \\ & A_{ij} = A_{ji} \qquad i,j=1,\ldots,n. \end{aligned} \tag{2.33}$$

It can be shown that this corresponds to the optimization problem solved by $k$-nearest neighbors, with the addition of the constraint that $A_{ij} = A_{ji}$. This ensures that a symmetric graph is constructed such that no postprocessing step is required. However, no efficient algorithm for solving the $b$-matching optimization problem is known: the best known algorithm has time complexity $O(n^{2.5})$ and is premised on some assumptions that are not always satisfied in real-world scenarios [87].

### Graph weighting

The graph weighting phase, which forms the second step of graph construction, determines the weights for the edges in the graph. In many cases, the weights correspond to the similarity measure used for constructing the edges. For instance, a Gaussian kernel is often used both to determine the connectivity of the graph via $k$-nearest neighbors and for the edge weights. In that case, the graph construction process is usually considered as consisting of weighting and *sparsification*. Firstly, a complete adjancency matrix $K$ is constructed using some kernel function $k$ for all pairs of nodes such that $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Then, the weight matrix $W$ is obtained by sparsification, i.e., by removing edges from $K$.

Several methods for edge weighting have been suggested in the literature. One of the most popular weighting schemes is Gaussian edge weighting ([51, 89]), where

$$W_{ij} = \exp\left(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right), \tag{2.34}$$

and $\sigma^2$ is the variance of the Gaussian kernel. Note that this corresponds to an isotropic Gaussian kernel; a non-isotropic Gaussian kernel can also be used. Hein and Maier suggest a local variant of Gaussian edge weighting for $k$-nearest neighbor graph construction, where the variance for a pair of nodes $i$ and $j$ is based on the maximum distance to $i$ and $j$'s nearest neighbors [84]. They define the weight as

$$W_{ij} = \exp\left(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{(\max(h_i, h_j))^2}\right), \tag{2.35}$$

where $h_i = \max_{v_k\in\mathcal{N}(v_i)} ||\mathbf{x}_i - \mathbf{x}_k||^2$, i.e., the maximum squared distance between $i$ and its neighbors. Blum and Chawla suggest altering the importance of different features in the similarity calculation based on their information gain [21]. Jebara et al. experiment with binary weights, where $W_{ij} = 1$ if nodes $i$ and $j$ are connected, and $W_{ij} = 0$ otherwise [89]. We note that, in all cases described above, $W_{ij} = 0$ for unconnected nodes.

The approaches described above determine edge weights $W_{ij}$ based solely on the pairwise similarity of nodes $\mathbf{x}_i$ and $\mathbf{x}_j$. However, it is also possible to take a the entire neighborhood of a node into account when determining edge weights. Wang and Zhang introduced the *linear neighborhood propagation* (LNP) algorithm [188]. Their graph construction method assumes that the graph should be constructed such that any sample $\mathbf{x}_i$ can be approximated as a linear combination of its neighbors. In other words, they assume that any sample $\mathbf{x}_i$ can be represented as

$$\mathbf{x}_i = \sum_{v_j \in \mathcal{N}(v_i)} W_{ij} \cdot \mathbf{x}_j + \boldsymbol{\epsilon}_i \tag{2.36}$$

for some vector $\boldsymbol{\epsilon}_i$ of low magnitude. In this equation, the unknowns are the weights $W_{ij}$ of the contributions of each neighbor to the approximation of $\mathbf{x}_i$. The approach by Wang and Zhang consists of estimating $W$ such that the difference between the approximated sample values and the true sample values is minimized, while ensuring that the weights are positive and that the sum of the edge weights for each node some up to 1. The resulting optimization problem they pose is then:

$$\begin{array}{lll} \underset{W \in \mathbb{R}^{n \times n}}{\text{minimize}} & \sum_{i=1}^{n} ||\mathbf{x}_i - \tilde{\mathbf{x}}_i||^2 & \\ \text{subject to} & \sum_{v_j \in \mathcal{N}(v_i)} W_{ij} = 1 & i = 1, \ldots, n \\ & W_{ij} \geq 0 & i, j = 1, \ldots, n \end{array} \tag{2.37}$$

where $\tilde{\mathbf{x}}_i = \sum_{v_j \in \mathcal{N}(v_i)} W_{ij} \cdot \mathbf{x}_j$. The optimization problem is identical to *locally linear embedding* (LLE, [150]) with the addition of the two constraints. LNP can be solved via a series of quadratic programming problems (one for each node). Crucially, this depends on the fact that edge weight symmetry is not enforced, i.e., it is not necessarily the case that $W_{ij} = W_{ji}$. Because of this, all weights $W_{ij}$ are independent of weights $W_{kj}$ for $k \neq i$.

Karasuyama and Mamitsuka combine locally linear embedding with a local similarity measure to obtain the edge weights [93]. In particular, given a pre-constructed graph (for instance, with $k$-nearest neighbors), they calculate the weight between two connected nodes with the Gaussian kernel; the covariance of this kernel is determined via locally linear embedding. The authors use a diagonal covariance matrix for the Gaussian kernel, which is chosen such that it minimizes the local reconstruction error, i.e., the difference between $\mathbf{x}_i$ and the linear combination of its neighbors.

Liu and Chang construct the weight matrix with a modification of the *symmetric $k$-nearest neighbors* method: two nodes are connected if either of them is in the other's $k$-neighborhood, but the weight of the two connections is summed if they are both in each other's neighborhoods [115]. In other words, the modified weight matrix $W$ is constructed based on the original weight matrix $\hat{W}$ as follows:

$$W_{ij} = \begin{cases} \hat{W}_{ij} + \hat{W}_{ji}, & \text{if } v_i \in \mathcal{N}(v_j) \text{ and } v_j \in \mathcal{N}(v_i) \\ \hat{W}_{ji}, & \text{if } v_i \in \mathcal{N}(v_j) \text{ and } v_j \notin \mathcal{N}(v_i) \\ \hat{W}_{ij}, & \text{otherwise} \end{cases} \tag{2.38}$$

De Sousa et al. compare the influence of several of these methods on the performance of transductive algorithms [51]. In particular, they compare Gaussian weights, the locally normalized Gaussian approach by Hein and Maier [84], and LNP [188]; somewhat surprisingly, their best results are obtained using Gaussian weights.

**Simultaneous graph construction and weighting**

The LNP algorithm described earlier (see Section 2.6.3) assumes that the graph structure (i.e., which pairs of nodes are connected) is known and fixed, and determines the weights of the edges of each node locally, assuming each node can be reconstructed as a linear combination of its neighbors. Instead of fixing the graph structure, however, one can also simultaneously infer the graph structure and the edge weights by linearly reconstructing nodes based on *all* other nodes.

Such an approach was first proposed by Yan and Wang [203], based on the *sparse coding* approach formulated for face recognition by Wright et al. [197]. Their approach finds, for each node $\mathbf{x}_i$, a coefficient vector $\mathbf{a} \in \mathbb{R}^n$, denoting the contributions of all other nodes to $\mathbf{x}_i$'s reconstruction. The reconstruction, then, is calculated as $\tilde{\mathbf{x}}_i = (X')^\intercal \cdot \mathbf{a} + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is the error vector. $X' \in \mathbb{R}^{n \times d}$ denotes the full data matrix, but with a row of zeroes at index $i$ (since a node cannot contribute to its own reconstruction). Note that, unlike the LNP reconstruction from Equation 2.37 where only predetermined neighbors contribute to the reconstruction, all $n-1$ other nodes can contribute to the reconstruction. The corresponding basic optimization problem attempts to minimize, for each sample, the norm of the difference $\boldsymbol{\epsilon}$ between the reconstruction and the true data. Crucially, Yan and Wang use the L1-norm. This is the second major difference with LNP, which uses the L2-norm and thus promotes non-sparse solutions.

To prevent an underdetermined system of equations in some cases, the final optimization problem penalizes both the norm of the reconstruction coefficients and the noise vector. Let $B = [(X')^\intercal, I_d]$ be the concatenation of the data matrix $X$ and the $d \times d$ identity matrix $I_d$. The reconstruction $\tilde{\mathbf{x}}_i$ of input sample $\mathbf{x}_i$ can then be calculated as $\tilde{\mathbf{x}}_i = B\mathbf{a}'$. Here, $\mathbf{a}'$ consists of the coefficient vector $\mathbf{a} = [a'_1, \ldots, a'_n]$ and the noise vector $\epsilon = [a'_{n+1}, \ldots, a'_{n+d}]$. The final optimization problem for finding the optimal coefficients is then defined as follows for each node $\mathbf{x}_i$:

$$\begin{aligned} \underset{\mathbf{a}' \in \mathbb{R}^{n+d}}{\text{minimize}} \quad & ||\mathbf{a}'||_1 \\ \text{subject to} \quad & B \cdot \mathbf{a}' = \mathbf{x}_i, \end{aligned} \tag{2.39}$$

where $|| \cdot ||_1$ is the L1-norm. Now, let $\mathbf{a}_i$ denote the coefficient vector found for node $i$. The final graph is then constructed by simply adding an edge between nodes $i$ and $j$ if and only if $a_{ij} \neq 0$, and setting the edge weights to the magnitude of the coefficient, i.e., $W_{ij} = |a_{ij}|$. We note that this approach does not yield an undirected graph.

However, Yan and Wang show that when quadratic loss is used for the unlabeled term, i.e., $\ell_U(\hat{y}_i, \hat{y}_j) = (y_i - y_j)^2$, a closed-form solution exists for the optimal labeling $\hat{\mathbf{y}}_u$. A variant of the sparse coding method is proposed by He et al., who impose a constraint that all coefficients be nonnegative to the objective from Equation 2.39 [83].

The coefficient vector $\mathbf{a}$ can be seen as an encoding of the data point $\mathbf{x}_i$. From this perspective, one would expect similar data points to have similar encodings. Zhuang et al. capture this preference by constructing a matrix $A$ from all encodings $\mathbf{a}_1, \dots, \mathbf{a}_n$ and regularizing the objective function by its rank [223]. Based on a well-known clustering method called *low-rank representation* (LRR, [114]), the regularization term penalizes coefficient matrices of high rank. The low-rankness of the matrix captures global structures in the data, while sparsity captures the local structure among data points. The resulting optimization problem, which includes the nonnegativity constraint and penalizes the L0-norm of the coefficients, is NP-hard; Zhuang et al. propose a convex relaxation leading to an objective function identical to the *sparse coding* objective function from Equation 2.39, but with the addition of the nonnegativity constraint and a surrogate for the rank-regularization term.

Despite excellent empirical results, the motivation for using the contribution coefficients $\mathbf{a}$ as graph weights remains somewhat unclear. As an alternative, Li and Fu use the reconstruction coefficients of pairs of samples to measure their similarity [111, 112]. In particular, they build a matrix of encoding vectors that is both sparse and of low rank, capturing global structures in the data, and base the similarity of data points on their distance to each other in encoding space. Additionally, they impose the constraint that all nodes have equal degree to promote sparsity and regularity of the graph.

### 2.6.4 Scalable transductive learning

Many of the graph construction and inference methods discussed thus far suffer from a lack of scalability [117]. Graph construction methods commonly have $O(n^2)$ time complexity (for instance, $k$-nearest neighbors has time complexity $O(k \cdot n^2)$); inference methods generally have time complexity $O(n^3)$ for obtaining the exact solution and $O(n)$ for approximate solutions. This can make it difficult to apply graph-based methods in real-world applications with large quantities of unlabeled data. Liu et al. provide an overview of approaches for making graph-based methods more scalable [117].

To tackle the scalability problem, several approaches have been proposed for efficiently constructing smaller graphs on which inference can be performed. These approaches rely on finding a set of $m \ll n$ *prototype* or *anchor* points to express the structure in the data more compactly. When these anchor points are found, the inference process can proceed over the anchor points, after which each unlabeled data point can be classified based on the inferred labels of nearby anchor points. In this framework, Zhang et al. propose to use a low-rank approximation to the adjacency matrix in the graph regularization term [208].

A commonly used approach called *anchor graph regularization* was proposed by Liu et al [116]. Their method seeks to find a set of anchor points $\mathbf{u}_1, \dots, \mathbf{u}_k$ and corresponding label assignments such that, for each data point $\mathbf{x}_i \in X$, the true label can be expressed as a linear combination of the labels of the anchor points. They choose the positions of the anchor points using $k$-means clustering, and construct a graph connecting each data point to its closest $s$ anchors. The corresponding weights are defined via locally linear embedding (see Section 2.6.3); these are then used to construct a graph over all data points. The inference process indirectly optimizes the predictions for the data points by optimizing a graph-based objective function defined over the predictions for the anchor points.

### 2.6.5 From transduction to induction

Since transductive methods do not provide a classification model over the entire input space, the entire algorithm needs to be rerun when a prediction for a new data point is desired. Due to the computational complexity of transductive algorithms, this is undesirable in many cases. This problem has not been studied extensively in the literature, but some potential solutions have been proposed.

The first type of approach is to find the optimal label prediction for previously unseen data points based on the objective function of the transductive algorithm. Such approaches fix the transductive predictions, and use the resulting graph to predict the label of previously unseen data points [15, 216]. Considering the general objective function from Equation 2.21, the optimal label assignment for the new data point can be calculated efficiently: assuming we can calculate the graph weights $W_{ij}$ for $j = 1, \ldots, n$, we can efficiently optimize the objective function with respect to only the predicted label of the new data point. The label assignment $\hat{y}_i$ minimizing the cost function is then given by the weighted average of its neighbors' predictions:

$$\hat{y}_i = \frac{\sum_{j=1}^{n} W_{ij} \cdot \hat{y}_j}{\sum_{j=1}^{n} W_{ij}}. \tag{2.40}$$

The second type of approach for building an inductive classifier is to treat the pseudo-labeled predictions as true labels, and to train a supervised classifier based on these predictions. This approach is taken by Kveton et al., who use the min-cut approach to obtain the optimal labels, and train a supervised SVM using the combined labeled and unlabeled data [102]. One can consider using a transductive approach with probability estimates, such that unlabeled samples can be weighted in the supervised learning algorithm. This approach can also be applied to inductive learners that have a computationally expensive prediction phase: we can train an inductive semi-supervised learning method on all available data, and pass its predictions for the unlabeled data along with the labeled data to a computationally more efficient classifier [180]. The efficient predictor can then be used to make predictions on new, previously unseen data points.

### 2.6.6 Classification in network data

In some real-world problems, data is inherently represented as a graph. Such data, which is commonly referred to as *network data*, arises in the context of social networks, scientific collaborations, disease spread networks, company structures, etc. In such networks, nodes generally represent entities (such as people), and edges represent relations between them (such as friendships). The field that studies such data is commonly known as network science [9].

In such network data, graph-based transductive methods are an obvious choice for performing inference. Node classification in particular can be considered a regular transductive semi-supervised learning task, and is broadly applied to problems in social network analysis and natural language processing [173, 204]. Although there is a considerable amount of overlap between these fields, the semi-supervised learning and network science communities have operated rather independently. Of course, significant differences also exist between data that is inherently given in the form of a network and graphs that are inferred from input vectors based on some similarity measure.

Sen et al. provide an overview of inference techniques for node classification in network data. They emphasize the difference between *local* classification, where each node is classified individually based on its neighbors (possibly iteratively), and *global* classification, where a global, joint objective is optimized [153]. They specifically consider the *iterative classification algorithm* (ICA), which constructs a local, supervised classifier for each node and assigns it the most likely label based on its neighbors and their labels [121, 128]. This procedure is iterated until the predictions in the entire network stabilize. Yang et al. propose a neural network-based approach that simultaneously predicts a node's labels and its context using node embedding [204]. They extend this approach to the inductive setting by expressing the embedding as a function of a node's features (and not its context). The context is predicted using a random walk; similar approaches to that problem have been previously studied [137, 174]. Several approaches have been suggested to generalize convolutional neural network architectures to network data (see, e.g., [29, 60, 96]).

Network-based methods generally attempt to find a way to represent the network-based data as vectors, allowing for inductive inference [204]. Interestingly, this can be considered the inverse task of what most semi-supervised graph-based methods attempt to do, which is to construct a graph based on vector data. These complementary approaches highlight the difference between 'standard', tabular data and data specified natively in the form of a network.

## 2.7 Related areas

Although semi-supervised classification comprises the vast majority of semi-supervised learning research, studies have been conducted in other subfields as well. Most notably, these include semi-supervised regression and semi-supervised clustering, which we cover in limited detail below. We do not cover other, more distantly related machine learning scenarios, such as active learning [154].

### 2.7.1 Semi-supervised regression

In classification problems, the label space $\mathcal{Y}$ is categorical and, in virtually all cases, finite. In regression problems, on the other hand, the output value space is continuous. Although some semi-supervised classification methods can be naturally applied to the regression setting, most cannot.

A class of methods that can be rather easily extended to the regression setting is that of *graph-based* methods (see Section 2.6). Many such methods model a real-valued function in an intermediate step and incorporate the real-valued predictions in a regularization term in the objective function. These real-valued predictions can be readily applied to the regression scenario (see, e.g., [11, 43]).

The second class of methods that can be readily applied to regression problems is the class of *wrapper methods* (see Section 2.3). Although relatively little research has been conducted in this direction, one can trivially apply paradigms such as self-training and co-training to regression methods. In fact, like in supervised classification methods, any supervised regressor can be used with a wrapper method. Zhou and Li propose a co-training algorithm for semi-supervised regression [213]. They construct two $k$-nearest neighbors regressors on the labeled data which then iteratively pass each other pseudo-labeled data. The labeling confidence, which is used to select data points to pseudo-label, is estimated based on the regressors' consistency with the labeled data.

### 2.7.2 Semi-supervised clustering

Semi-supervised classification is a relatively well-defined task, where one is presented with both fully labeled data and completely unlabeled data. In semi-supervised clustering, however, the supervised information can take different forms. For instance, there can be *must-link* (two samples are known to be in the same cluster) and *cannot-link* (two samples are known to be in different clusters) constraints [104]. On the other hand, it is also possible that some cluster assignments are known beforehand.

An example of the incorporation of the latter type of information is the use of labeled data for cluster *seeding*. Basu et al. propose to initialize the clusters based on the data points for which the cluster assignments are known [10]. For every cluster, they initialize the cluster centroid for the $k$-means algorithm to the mean feature values of the data points known to belong to that cluster. They also propose an alternative of this approach where the cluster assignments of the labeled data points are kept fixed in the $k$-means procedure.

Like semi-supervised regression, semi-supervised clustering is a minor research area when compared to semi-supervised classification. For a more extensive overview of semi-supervised clustering methods, we refer the reader to the recent semi-supervised clustering survey by Bair [6] and the older survey on clustering methods by Grira et al. [74].

## 2.8 Conclusions and future perspectives

In this survey, we have presented an overview of the field of semi-supervised learning. Covering both methods from the early 2000s and more recent advances, our survey constitutes an up-to-date review of the research field. Furthermore, we have presented a new taxonomy for semi-supervised classification methods, distinguishing between the different objectives of the approach (*transductive* versus *inductive*) and the way learners incorporate unlabeled data (i.e., *wrapper methods*, *unsupervised preprocessing*, and *intrinsically semi-supervised methods*).

Early research in the field of semi-supervised learning mainly focused on wrapper methods (Section 2.3) and semi-supervised extensions of traditional supervised algorithms (such as support vector machines, see Section 2.5). Graph-based methods (Section 2.5.3 and Section 2.6), have been extensively researched over the past two decades. They are perhaps the most intuitive semi-supervised learning method, explicitly incorporating the similarity of different unlabeled data points in a principled way. However, they still pose computational challenges. In recent years, semi-supervised learning has developed along the same lines as supervised learning. We have seen a boost in research regarding semi-supervised neural networks, both in the form of unsupervised preprocessing (Section 2.4.3) and in the form of semi-supervised regularization (Section 2.5.2). Additionally, deep generative models have been extended to the semi-supervised setting (Section 2.5.4).

From our perspective, one of the most important issues to be resolved in semi-supervised learning is the potential performance degradation caused by the introduction of unlabeled data. Although receiving relatively little attention in the literature (likely due to publication bias, as noted by Zhu [216]), many semi-supervised learning methods only perform better than their supervised counterparts or base learners in specific cases [113, 160]. Moreover, the potential performance degradation is generally much more significant than the potential improvement, especially in machine learning problems where good performance is achieved with purely supervised learning. We believe that this is one of the main reasons for the dearth of applications of semi-supervised learning methods in practice when compared to supervised learning.

A notable exception are the recent advances in semi-supervised neural networks, which are generally perturbation-based (see Section 2.5.2). They incorporate the relatively weak *smoothness assumption* (i.e., minor variations in the input space should only cause minor variations in the output space). Empirically, these methods have shown to be robust, consistently outperforming their supervised counterparts. Additionally, incorporating unlabeled data into their optimization procedure is relatively efficient computationally: it involves the same type of backpropagation procedure used for labeled samples.

A second potential remedy for the lack of robustness of semi-supervised learning methods lies in the application of *automated machine learning* (AutoML) to the semi-supervised setting. Recently, interest has significantly increased in automatically selecting and configuring learning algorithms for a given classification problem. These approaches include meta-learning and automated algorithm selection and hyperparameter optimization. Recently, these approaches have been successfully applied to supervised learning (see, e.g., [62, 178]). They have, however, not been applied to semi-supervised learning yet. In Chapter 3, we propose a semi-supervised ensembling method and apply it to ensembles constructed with AutoML.

Lastly, we expect the strong distinction between the fields of clustering and classification to fade. Fundamentally, both of these topics can be seen as a specification of semi-supervised learning to the case where either only labeled data or only unlabeled data is present. When we can confidently reason about the connections between the marginal distribution $p(x)$ and the conditional distribution $p(y|x)$, both unlabeled and labeled data can be confidently used in learning algorithms. The recent rise in popularity of generative models (see Section 2.5.4) can be seen as evidence for this paradigm shift.

Ultimately, we expect the incorporation of unlabeled data to be a vital step in the progress of machine learning and artifical intelligence. To uncover the intricate and complex structures underlying the data model, the machine needs to be able to infer patterns between observations about which it receives no explicit labeling information. Semi-supervised learning, which aims to provide mechanisms to build such connections, will be an important tool in further developing machine learning.

# Chapter 3

# Automated Single-step Co-ensembling

Most machine learning algorithms cannot be used off the shelf, requiring significant amounts of hyperparameter tuning to achieve acceptable performance. In recent years, automated machine learning systems have started to tackle this problem by automatically selecting and configuring machine learning algorithms for any data set. AUTO-SKLEARN is a state-of-the-art automated machine learning system that uses meta-learning and advanced ensemble selection methods to robustly construct learner ensembles. We propose to improve the ensemble constructed by AUTO-SKLEARN using semi-supervised learning. In particular, we introduce a generic co-training procedure called *co-ensembling*, which can be applied to any ensemble of more than two base learners. We show that we can consistently improve the performance of the ensemble constructed by AUTO-SKLEARN in multiclass classification problems by applying a single co-ensembling iteration. Furthermore, we show that this procedure generally outperforms multi-step co-ensembling, which exhibits decreasing performance and increasing performance variance as the number of iterations is increased. On a large benchmarking suite of diverse multiclass classification data sets, single-step co-ensembling yields a relative reduction in error rate of 7% on average; we improve the performance of the ensemble on over 75% of the multiclass classification problems we consider in our experiments.

## 3.1 Introduction

Traditionally, applying machine learning techniques to real-world problems has required substantial human effort in finding and configuring a suitable learning algorithm for the problem at hand. This approach, where an expert considers a given problem or data set and manually configures a machine learning algorithm to tackle the problem, is not scalable when considering the ever-growing range of machine learning applications. Consequently, the demand for machine learning systems that require little or no human interference has grown substantially.

*Automated machine learning*, or AutoML, is the branch of machine learning that

*This chapter is based on an article co-authored with H.H. Hoos. I took primary responsibility for the research conducted for this article, as well as for writing the manuscript.*

considers the automation of finding an algorithm and a corresponding hyperparameter setting that performs well on a given problem (see, e.g., [178]). It can be seen as a generalization of the hyperparamater optimization problem, which attempts to find the optimal set of hyperparameters for a given learning algorithm and data set. Recent advances in automated machine learning combine meta-learning, which reasons about the performance of algorithms based on data set characteristics and previous experiments, with Bayesian optimization methods [161] and ensemble learning [62].

Ensembles of learners have been long known to outperform individual learners in most supervised learning scenarios [212]. By training multiple, diverse classifiers on the same problem or on small variations of the same problem and combining their predictions, both overall performance and stability can be improved over individual learners. The popular random forest model, for instance, is an ensemble model [26]. For ensembles to outperform individual learners, it is a necessary condition that the learners are (1) accurate by themselves and (2) diverse (i.e., they make uncorrelated errors) [28, 56, 82]. AutoML methods that construct ensembles of classifiers satisfy the latter property either implicitly through Bayesian optimization [178] or explicitly in an ensemble construction phase [62].

To the best of our knowledge, automated machine learning techniques have not yet been applied to semi-supervised learning. This can be partially attributed to the lack of standardized implementation toolkits for semi-supervised learning, which do exist for supervised learning (e.g., Weka [79], scikit-learn [136]). Furthermore, semi-supervised learning is not applied as broadly as supervised learning, and no clearly defined set of methods is commonly applied in practice. Although wrapper methods (see Section 2.3) could potentially be incorporated into the algorithm and hyperparameter space relatively easily, their computational complexity can prove to be a limiting factor due to their iterative nature and the large number of unlabeled data points in typical real-world problems.

Instead of incorporating semi-supervised learning algorithms or paradigms in the algorithm and hyperparameter space, one could also consider employing semi-supervised learning in a separate training step after constructing supervised classifiers with AutoML. In particular, one could apply semi-supervised ensemble methods to the ensemble constructed by the AutoML system. We note that the diversity and accuracy conditions for ensembles of supervised models also apply to semi-supervised ensemble methods [215]. For instance, the traditional co-training algorithm, which trains an ensemble of two classifiers simultaneously by exchanging pseudo-labeled data points, promotes diversity between the learners by training them on different subsets of the input features [23]. In this work, we use this knowledge to train a semi-supervised ensemble of learners obtained through AutoML, using a co-training style algorithm to re-train them with both labeled and pseudo-labeled data.

Our main contribution is the proposal of a single-step co-training procedure where an ensemble of classifiers is re-trained a single time using labeled and pseudo-labeled data. The algorithm, which we call *single-step co-ensembling*, trains an ensemble of $K$ classifiers on labeled data and uses each sub-ensemble of $K-1$ classifiers to pseudo-label data for the remaining classifier. Each classifier is then re-trained on the labeled data and its pseudo-labeled data. By applying only a single co-ensembling iteration on the entire set of unlabeled data points, we overcome a common problem with wrapper methods, where repeated pseudo-labeling iterations can introduce significant variance in performance and even degrade performance over the purely supervised ensemble [179]. We apply the co-ensembling algorithm to ensembles constructed by AUTO-SKLEARN, which is a prominent, state-of-the-art automated machine learning system. In multiclass classification problems, we are able to significantly improve the predictive power of the ensemble with a single co-ensembling iteration.

The rest of this chapter is structured as follows. In Section 3.2, we explain the concept of AutoML and the AUTO-SKLEARN system. We then introduce our single-step co-ensembling algorithm, along with a general framework for co-ensembling, in Section 3.3. Our experiments and results are outlined in Section 3.4, followed by conclusions and discussion provided in Section 3.5.

## 3.2 AutoML: auto-sklearn

Given a learning problem, the goal of AutoML systems is to automatically infer a learner that is well-suited to tackle this problem. Since a broad variety of learning algorithms exists, and most of these algorithms depend on multiple hyperparameters that significantly influence the performance of the learner, this is a challenging task. However, recent work in AutoML has shown that reliable, broadly applicable AutoML systems can be constructed that yield impressive performance across a broad range of machine learning problems [62, 100, 178].

The AutoML problem can be cast as an optimization problem where some cost function, expressing the performance of the learner on the input data set, has to be minimized. The cost function is an approximation of the generalization error. It is obtained by splitting the data into training and test sets and evaluating the test set performance of a learner trained on the training set. We denote these $k$ training and test sets, which are usually obtained via cross-validation or holdout, by $D^{(i)}_{\text{train}}$ and $D^{(i)}_{\text{test}}$, respectively. The loss function of some algorithm $A$ on the test set, when trained on the training set, is then denoted as $\mathcal{L}(A, D^{(i)}_{\text{train}}, D^{(i)}_{\text{test}})$

Let $\mathcal{A} = \{A^{(1)}, \ldots, A^{(R)}\}$ denote the space of available machine learning algorithms, and let $\mathbf{\Lambda}^{(1)}, \ldots, \mathbf{\Lambda}^{(R)}$ denote their respective hyperparameter spaces. Together, these form a hierarchical search space of all available algorithm configurations, known as the *configuration space*. The AutoML optimization problem, then, can be formulated as follows:

$$\underset{A^{(j)} \in \mathcal{A}, \mathbf{\Lambda} \in \mathbf{\Lambda}^{(j)}}{\text{minimize}} \quad \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\left(A^{(j)}_{\mathbf{\Lambda}}, D^{(i)}_{\text{train}}, D^{(i)}_{\text{test}}\right), \tag{3.1}$$

where $A^{(j)}_{\mathbf{\Lambda}}$ denotes learning algorithm $A^{(j)}$ with hyperparameters $\mathbf{\Lambda}$.

Traditional hyperparameter optimization approaches tackle the same problem, but fix the algorithm $A_j$. Examples of common hyperparameter optimization approaches include grid search, random search [18], and, more recently, Bayesian optimization methods [161]. The more general task of combined algorithm selection and hyperparameter optimization (known as *CASH*) involves a much larger configuration space and is generally approached using Bayesian optimization methods.

The first generic AutoML approach was Auto-Weka [100, 178], which constructs a configuration space consisting of Weka [79] machine learning algorithms and feature selection methods. The algorithm space contains multiple ensemble methods accepting arbitrary base classifiers, which means the total configuration space size is exponential in the number of base classifiers allowed by these ensemble methods. The authors limit this to 5 base classifiers.

Another popular, more recent AutoML system is auto-sklearn [62], which is based on the Python machine learning toolkit scikit-learn [136]. It takes the ensemble construction process outside of the configuration space, instead constructing the ensemble in a postprocessing step. A greedy ensemble selection procedure is employed, which iteratively adds the classifier to the ensemble that minimizes the objective function for the resulting full ensemble [31]. This greatly reduces the size of the configuration space. auto-sklearn also employs meta-learning [25]: by running the optimization procedure for long stretches of time on a large, diverse set of data sets, a model is constructed for the posterior performance distribution conditioned on data set characteristics. This model is used for new problems, initialising the Bayesian optimization procedure to promising configurations.

A variety of Bayesian optimization procedures exists [155]. Both Auto-Weka and auto-sklearn use the sequential model-based optimization method *SMAC* to find the optimal algorithm and hyperparameters. SMAC builds a random forest model that captures the dependence of the loss $\mathcal{L}$ on the algorithm $A$ and hyperparameter settings $\mathbf{\Lambda}$. In each iteration, SMAC selects a new, promising configuration to evaluate by considering the configuration with the maximum expected improvement over the current best configuration. The resulting evaluation is then used (in addition to the previously obtained evaluations) to re-train the random forest model. This process is repeated until a prespecified evaluation budget is exhausted. Crucially, SMAC consecutively evaluates each configuration on the $k$ different train-test sets, which allows the algorithm to terminate further evaluation of low-performance configurations at an early stage.

auto-sklearn is one of the most prominent AutoML systems. In the original paper, the authors show that it compares favorably to both Auto-Weka and Hyperopt-sklearn [99] on a broad variety of data sets [62]. Furthermore, it was the winning solution to the 2015 AutoML challenge [76].

## 3.3 Co-ensembling

We propose to enhance the ensemble generated by auto-sklearn using unlabeled data. Specifically, we use every sub-ensemble of $K-1$ classifiers to pseudo-label data for the remaining classifier. Each classifier is then re-trained on the labeled data and its newly obtained, pseudo-labeled data. Our approach falls into the group of *wrapper methods*, i.e., semi-supervised learning methods that use pseudo-labeling to re-train supervised classifiers (see Section 2.3). More specifically, it can be seen as a variant of co-training where more than two classifiers are trained. The pseudo-labeling approach we use for our semi-supervised method, where sub-ensembles form predictions for the remaining classifier, is very similar to the pseudo-labeling approaches used by *tri-training* [214] and *co-forest* [110] (see Chapter 4).

We note that we have also considered integrating wrapper methods into the configuration space of AUTO-SKLEARN. We integrated self-training into the configuration space, activating it based on a Boolean hyperparameter in the configuration space. However, most likely due to the widely observed weak performance of self-training or due to its computational inefficiency, we were not able to obtain any performance improvements. More information about these experiments is provided in Appendix B.

Co-training was one of the earliest semi-supervised learning paradigms. In its originally proposed form, it trained two classifiers independently on two different feature subsets of the data [23]. In each co-training iteration, the classifier obtained in the previous iteration is used to make predictions for the unlabeled data, which are pseudo-labeled with the predictions and passed to the other classifier. The final classifier is then simply constructed as the majority voting procedure using the two base classifiers. This approach relies on the existence of multiple *views* of the data, i.e., feature subsets, being available. Of course, this situation is not too common in machine learning problems. Since the original proposal of co-training by Blum and Mitchell, a plethora of co-training style methods has emerged. Although many of them consider the multi-view setting, single-view co-training has also received attention. Several approaches have been proposed that train base learners with different hyperparameters on the same data, or that use different base learners altogether [179].

### 3.3.1 Co-training assumptions

All co-training methods have one thing in common: they rely on the *diversity* of the base classifiers [28, 215]. In multi-view learning, this requirement is satisfied by using two uncorrelated or weakly correlated feature subsets. In single-view learning, the diversity has to be enforced in the classifiers themselves. In addition to the diversity criterion, co-training methods assume that the individual learners can, by themselves, yield accurate predictions. In co-training literature, the former condition is commonly referred to as *independence*, and the latter condition is commonly referred to as *redundancy* [215]. We note that these conditions coincide with the necessary conditions for supervised ensemble methods: the base learners need to be *diverse* and *accurate* [56, 82].

Finding an ensemble of diverse, accurate classifiers is not a trivial task: since each base learner needs to be accurate by itself, the corresponding algorithm and hyperparameter configuration task is even more complex than when using only a single classifier. We propose to mitigate this problem by applying the co-training paradigm to the ensemble generated by AUTO-SKLEARN, making use of the observation that the necessary conditions for successful classifier ensembles are identical to the necessary conditions for co-training to succeed.

### 3.3.2 General co-ensembling framework

We outline a simple framework for co-training using multiple classifiers. The framework does not use sample weighting, probabilistic predictions for individual classifiers, or bootstrapping. As such, it support supervised base learners of any type. We name this framework *co-ensembling*, referring to the application of co-training style optimization to any existing ensemble of classifiers. It is similar to the *co-forest* approach by Li and Zhou, who also use sub-ensembles of $K - 1$ classifiers to pseudo-label data for the remaining classifier [110]. However, their approach employs a stopping criterion based on an estimate of the generalization error; they bootstrap the initial, labeled data for each classifier to obtain the *out-of-bag error* in each iteration. Additionally, they use a subsampling step to limit the amount of pseudo-labeled data points. Our framework works with any ensemble of classifiers, and does not require bootstrapping. This is highly advantageous in scenarios where few base learners are available.

The final single-step co-ensembling procedure we propose fits nicely into our co-ensembling framework. In accordance with the Programming by Optimization (PbO) paradigm [86], we refrain from making design choices where not strictly necessary. The resulting framework contains several hyperparameters that allow for automatic configuration. Extensive configuration of these hyperparameters is beyond the scope of this work; we limit ourselves to optimization of the confidence threshold hyperparameter (see Section 3.4.4). We do note that automatic configuration with multi-step co-ensembling could be computationally expensive, since each co-ensembling iteration has the same computational complexity as the training phase of the ensemble of supervised base learners. Pseudo-code for the co-ensembling algorithm is provided in Algorithm 1.

The general co-ensembling procedure for base learners $h_1, \ldots, h_K$ proceeds as follows. Firstly, all classifiers are independently trained on the labeled data (lines 2 to 5). For each classifier $h_k$, we keep track of the unlabeled data points $U_k$ that can still be pseudo-labeled (i.e., that weren't pseudo-labeled in a previous iteration). This set of candidate samples is initialized to the full set of unlabeled data points. Secondly, we iteratively introduce unlabeled data to each of the classifiers (lines 6 to 16). We refer to one of these iterations as a *co-ensembling iteration*.

In each co-ensembling iteration, all ensembles of $K - 1$ classifiers pseudo-label samples for the remaining classifier. To select the samples to pseudo-label, the sub-ensemble $H_k$, consisting of all classifiers in the ensemble except for $h_k$, is evaluated on all remaining unlabeled samples $U_k$ for classifier $k$ (line 8). From the samples on which $H_k$'s confidence (i.e., the fraction of the $K - 1$ classifiers predicting the pseudo-label) exceeds a certain threshold $\theta$, the most confident samples are selected up to a certain limit *max_pl* (line 9). Each classifier $k$ is then re-trained on both the labeled samples and these selected unlabeled samples with their pseudo-labels from $H_k$ (lines 10 and 11). Depending on the setting of the Boolean hyperparameter *retain_pl* (governing whether pseudo-labeled samples should be retained in later iterations), the pseudo-labeled samples are removed from the set of unlabeled data points $U_k$ that can still be pseudo-labeled for classifier $k$ (lines 12 to 14).

### 3.3.3 Single-step co-ensembling

Although receiving relatively little attention in the literature (likely due to publication bias, as noted by Zhu [216]), many semi-supervised learning methods only perform better than their supervised counterparts or base learners in specific cases [113, 160]. This is also the case for wrapper methods, including self-training and co-training (see, e.g., [110, 179] and Chapter 4). Moreover, the potential performance degradation is generally much more significant than the potential improvement, especially in machine learning problems where good performance is achieved with purely supervised learning.

**Algorithm 1** Co-ensembling

```
Input:
    Labeled data $L = ((\mathbf{x}_i, y_i))_{i=1}^{l}$
    Unlabeled data $U = (\mathbf{x}_i)_{i=l+1}^{n}$
    Ensemble of classifiers $H = h_1, \ldots, h_K$
    Confidence threshold $\theta$
    Maximum number of iterations num_iter
    Maximum number of pseudo-labeled samples per iteration max_pl
    Whether to retain pseudo-labeled samples in the next iteration retain_pl
Output: Trained ensemble $H'$

1:  procedure Co-ensembling
2:      for $k = 1, \ldots, K$ do
3:          $U_k \leftarrow U$
4:          $h_k = \text{Train}_k(L)$
5:      end for
6:      for $t = 1, \ldots,$ num_iter do
7:          for $k = 1, \ldots, K$ do
8:              $U'_k \leftarrow \{\mathbf{x}_i \in U_k : \text{Confidence}(H_k, \mathbf{x}_i) \geq \theta\}$
9:              $U^*_k \leftarrow$ Select max_pl highest-confidence samples from $U'_k$
10:             $L'_k = \{(\mathbf{x}_i, H_k(\mathbf{x}_i)) : \mathbf{x}_i \in U^*_k\}$
11:             $h_k = \text{Train}_k(L \cup L'_k)$
12:             if retain_pl then
13:                 $U_k \leftarrow U_k \setminus U^*_k$
14:             end if
15:         end for
16:     end for
17:     return $(h_1, \ldots, h_K)$
18: end procedure
```

The main cause of potential performance degradation in wrapper methods is the iterative nature of the algorithms: when the incorrect pseudo-labeling of a sample significantly alters the decision boundary in an early iteration of the algorithm, this mistake will be amplified throughout subsequent iterations. Furthermore, the diversity of the ensemble, which is a necessary condition for the ensemble to perform well, is explicitly weakened in each co-training iteration. When passing pseudo-labeled data between classifiers, the correlation in the errors they make will generally increase.

We propose a simple solution to this problem: we only apply a single co-training iteration, but use the entire set of unlabeled data (i.e., without explicitly limiting the number of pseudo-labeled samples). The only constraint we impose is that the prediction confidence of the sub-ensemble is above some threshold $\theta$, to ensure that only confidently predicted samples are pseudo-labeled.

## 3.4 Experiments and results

We assess our single-step co-ensembling approach by applying it to the ensemble generated by AUTO-SKLEARN. This allows us to directly compare its peformance to the performance of AUTO-SKLEARN. We first describe our experimental setup in detail. Then, we show that using the ensemble constructed by AUTO-SKLEARN in an unweighted majority voting scheme improves performance over the original, weighted ensemble. We apply our co-ensembling procedure to this unweighted ensemble, and report considerable performance improvements on multiclass data sets. Furthermore, we demonstrate that the improvements diminish as the number of co-ensembling iterations grows. On the basis of this evidence for the viability of single-step co-ensembling, we conduct extensive experiments with single-step co-ensembling. We demonstrate that it can reliably produce significant performance improvements on multiclass data sets.

### 3.4.1 Experimental setup

Both automated machine learning systems and semi-supervised wrapper methods need to be applicable to a broad variety of data sets. We therefore evaluate the performance of our algorithms on a diverse suite of data sets with different characteristics, the *OpenML 100* [19]. This benchmarking suite consists of roughly 100 data sets; at the time of our experiments, it contained 52 binary and 45 multiclass data sets. A broad range of data set sizes is represented, each data set consisting of between 100 and 100,000 samples. Further information about the benchmarking suite is provided in Appendix C.

Our main experiments are conducted with 10% and 20% labeled data fractions; this constitutes the training set that is passed to AUTO-SKLEARN. It will be further split into different subsets for AUTO-SKLEARN's training procedure. The rest of the data remains unlabeled and constitutes the testing data. The features of these data points are, of course, used in the co-ensembling procedure. Varying the amount of labeled data relative to the number of data points in the data set is common when experiments are conducted on data sets with diverse sample sizes [179]. Due to computational limitations, some of our preliminary experiments are only conducted in the 20%-labeled setting, which provides the supervised base algorithms with more data and promotes stability in our experiments.

For each data set, we conduct 10 experiments with different labeled/unlabeled (i.e., train/test) splits to obtain confident performance estimates. Each experiment proceeds as follows. The data is split into labeled and unlabeled sets randomly (according to the prespecified ratio of labeled to unlabeled data). Then, the labeled training data is used by AUTO-SKLEARN to construct a weighted ensemble of classifiers. The evaluation of this ensemble on the test data yields the performance of AUTO-SKLEARN. We also evaluate the ensemble without weights on the testing data. Then, we apply single-step co-ensembling to the unweighted ensemble (if it has at least 3 base learners), passing it the features of the available unlabeled data points. The resulting ensemble is evaluated on the unlabeled data. This approach is similar to the transductive semi-supervised learning setting (see Section 2.6), where the goal of the learner is to infer the labels of the unlabeled data points it encountered in the training phase. However, our approach produces a classifier that is defined over the entire input space, as opposed to providing only the predictions for the unlabeled data points.

Our experiments are conducted with AUTO-SKLEARN version 0.3.0. Each run of AUTO-SKLEARN is performed in a distributed manner on 8 computing nodes, one of which is used exclusively as a controller, aggregating results from the other nodes and constructing the final ensemble. Each node receives a wall-clock time budget of 2 hours (10% labeled data) or 4 hours (20% labeled data); the time limit for each individual run (i.e., the construction of a single classifier) is limited to 10% of the total budget per node. RAM is limited to 2.5 GB per node.

The co-ensembling procedure, which refines the ensemble constructed by AUTO-SKLEARN, is run with a varying number of iterations *num_iter*. In our final single-step co-ensembling experiments, the number of iterations is set to 1. No explicit limit is imposed on the number of data points to pseudo-label in this single iteration ($max\_pl = \infty$). In accordance with hyperparameter optimization experiments carried out for single-step co-ensembling (see Section 3.4.4), the confidence threshold $\theta$ is set to 0.7. We note that this fixed threshold can be considered to be more strict for data sets with more classes, where the labeling confidence can be expected to be lower.

#### Construction of the final experiment set

Not all experiments we conduct are usable for performance comparisons. We outline the reasons for this, and elaborate on the experiments that needed to be discarded. All results we report are, unless explicitly stated otherwise, based on the experiments remaining after this selection step.

Firstly, some data sets are ill-suited for semi-supervised learning experiments due to their lack of sufficient data points per class. Since we need to split our data set into a small labeled data set and a large unlabeled data set to emulate common semi-supervised learning scenarios, the initial data set needs to contain sufficient samples per class. If it doesn't, no meaningful optimization procedure can be applied by AUTO-SKLEARN. We therefore discard data sets with fewer than 10 samples per class on average when 20% of the data is labeled. This leaves 91 data sets.

Secondly, the AutoML phase of the training process is susceptible to errors. For instance, AUTO-SKLEARN might suggest configurations that cause errors in SCIKIT-LEARN, and computing nodes may crash. After discarding crashed experiments, we are left with 697 valid experiments in the 10%-labeled setting and 710 valid experiments in the 20%-labeled setting.

Thirdly, co-ensembling requires two or more learners in each sub-ensemble to be able to estimate prediction confidence. Thus, three or more learners are required in total for co-ensembling to proceed. Discarding those ensembles with fewer than 3 learners, we obtain our final set of 396 experiments in the 10%-labeled setting and 496 experiments in the 20%-labeled setting. This constitutes a weakness in our method: if the ensemble constructed by AUTO-SKLEARN does not adhere to this specification, we cannot reliably improve performance. We note that the final number of experiments can vary slightly based on the hyperparameter settings; the numbers reported above are based on the single-step co-ensembling experiments with a confidence threshold of 0.7.

All of these steps are independent of the results of the co-ensembling procedure: we only omit infeasible and erroneous experiments, which can be identified as such before starting the co-ensembling procedure.

### 3.4.2 Supervised ensemble weighting

AUTO-SKLEARN constructs an ensemble of classifiers from the set of candidate classifiers by greedily selecting the classifier that minimizes the validation error of the ensemble [31, 62]. Classifiers are in principle unweighted, but can be selected multiple times. When the ensemble can no longer be improved or a predefined ensemble size limit is reached, ensemble construction is finalized: classifiers that occur multiple times are grouped, yielding a weighted ensemble.

In our co-ensembling experiments, we treat all classifiers in the ensemble equally. In other words, we do not use the classifier weights in our co-ensembling approach. Interestingly, our experimental results show that, even in the supervised setting, this approach yields better performance than the weighted ensemble constructed by AUTO-SKLEARN. The aggregated results of these experiments in both the 10%-labeled and 20%-labeled settings are provided in Table 3.1. To obtain these results, we grouped the results per data set and calculated the mean error rates of the different methods and the mean change in error rate. These results are then averaged to yield the final performance figures. To show that AUTO-SKLEARN is able to find high-quality classifier ensembles, we also compare it to random forests [26] with 100 trees. In the comparisons with random forests, any data sets with missing values are omitted since random forests do not natively handle missing data.

We assess the statistical significance of our results by applying the Wilcoxon signed-rank test [195] with a significance level of 5% to the list of mean error rates for our data sets, as is common when comparing two classifiers on multi data set experiments [54]. Interestingly, we find that the performance difference between AUTO-SKLEARN and random forests is significant in the 20%-labeled setting but not in the 10%-labeled setting. This can possibly be explained by the lack of sufficient labeled data AUTO-SKLEARN can use to properly evaluate its solutions.

The performance improvement of the unweighted AUTO-SKLEARN ensemble over the regular, weighted AUTO-SKLEARN ensemble and over random forests is significant

Table 3.1: Performance comparison of random forests, AUTO-SKLEARN (ASKL), and unweighted AUTO-SKLEARN on the OpenML 100 benchmarking suite. Both absolute error rates and relative error rate differences are reported; results are averaged over all data sets.

(a) Relative changes in error rate, averaged over all data sets, and win/tie/loss rates.

| Method | Fraction labeled | |
|---|---|---|
| | 10% | 20% |
| **ASKL vs Random forest** | | |
| Mean rel. change | -6.4% | -14.0% |
| Win/tie/loss | 42/0/32 | 51/0/23 |
| **ASKL (unweighted) vs ASKL** | | |
| Mean rel. change | -8.0% | -5.1% |
| Win/tie/loss | 57/2/19 | 56/1/21 |
| **ASKL (unweighted) vs Random forest** | | |
| Mean rel. change | -12.5% | -18.8% |
| Win/tie/loss | 46/1/27 | 54/1/19 |

(b) Absolute error rates, averaged over all data sets.

| Method | Fraction labeled | |
|---|---|---|
| | 10% | 20% |
| Random forest | 0.196 | 0.173 |
| ASKL | 0.188 | 0.160 |
| ASKL (unw.) | 0.182 | 0.154 |

in both labeled data fractions. Furthermore, the win/tie/loss rates clearly show that it outperforms both the weighted ensemble and random forests in a large majority of cases. This shows that, even when using very few data points, AUTO-SKLEARN yields a well-performing ensemble. We stress that our results only include AUTO-SKLEARN ensembles consisting of more than 2 learners (see Section 3.4.1). When the ensemble contains only 2 learners, one can expect the weighted ensemble to outperform the unweighted ensemble: the potential influence of a weak learner with low weight is, in that case, potentially very large. Preliminary experiments (not reported here) substantiate that claim.

### 3.4.3 Multi-step co-ensembling

Before continuing to the evaluation of single-step co-ensembling, the centerpiece of our experiments, we evaluate multi-step co-ensembling. Using the trained, unweighted ensemble from AUTO-SKLEARN as a starting point, we run the co-ensembling procedure with *num_iter* $= 20$ iterations. These experiments are conducted in the 20%-labeled setting. We compare the performance of the resulting ensemble to the unweighted base ensemble after each iteration. We assess both the mean relative change in error rate and its variance as the number of iterations grows.

The results are depicted in Figure 3.1. As is clearly visible, the performance improvement obtained when applying a single co-ensembling iteration is subtantial in multiclass data sets: a mean performance improvement of over 7% is obtained after the first co-ensembling iteration. As the number iterations grows, however, the improvement diminishes. In binary data sets, performance is not noticeably improved at all. However, the procedure exhibits the same behavior as in multiclass data sets when varying the number of co-ensembling iterations: performance deteriorates as the number of iterations grows, and the best performance is obtained when using just a single co-ensembling iteration.

The progressive degradation in performance improvement could potentially be caused by the previously discussed effects of co-ensembling. Firstly, the diversity between the classifiers decreases as the number of co-ensembling iterations grows. Secondly, incorrect pseudo-labels obtained through confident misclassifications can be amplified throughout the co-ensembling procedure.

Considering Figure 3.1, we notice that the variance in the relative change in error rate grows significantly throughout the co-ensembling process. Large variance in the performance change when applying a wrapper method to a base learner signifies a lack of robustness and a relatively high probability of substantial performance degradations. In general, we want the mean improvement to be large and the variance in the mean improvement to be low. Both of these measures are optimized when using only a single co-ensembling iteration, confirming our hypothesis.

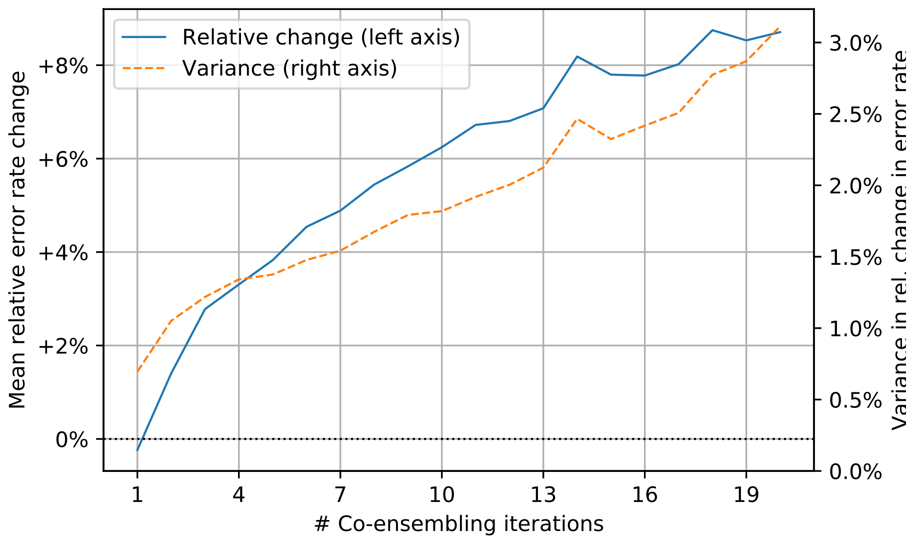
### 3.4.4 Single-step co-ensembling

Having established the ability of co-ensembling to improve the performance of the ensemble constructed by AUTO-SKLEARN, we proceed to a broader evaluation of the performance of single-step co-ensembling. Firstly, we assess the influence of the confidence threshold hyperparameter $\theta$ on the performance of single-step co-ensembling. Then, using the optimal confidence threshold from these experiments, we elaborate on the performance of single-step co-ensembling in both the 10%-labeled and 20%-labeled settings. We show that single-step co-ensembling provides a substantial performance improvement on multiclass data sets, and that it does so reliably, improving performance on a large majority of data sets.

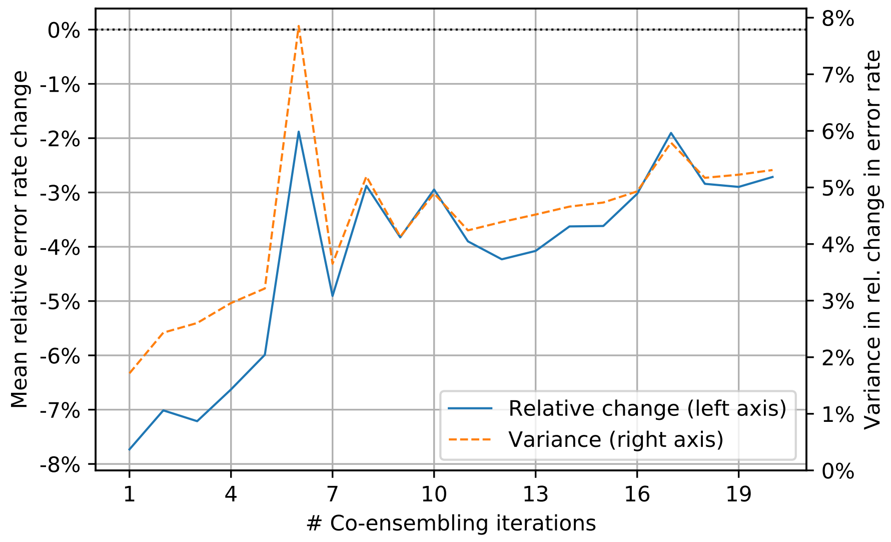
#### Confidence threshold

The co-ensembling framework we have presented is relatively generic: it allows for varying numbers of iterations, limits on the number of pseudo-labeled samples per iteration, and different confidence thresholds. Having evaluated the influence of the number of co-ensembling iterations in the previous, we now investigate the influence of the confidence threshold on the performance of co-ensembling. Our co-ensembling method does not impose restrictions on the number of samples to pseudo-label per iteration: from our perspective, there is no compelling reason to impose such as limit unless driven by computational limitations.

To evaluate the impact of the confidence threshold $\theta$ on the performance of co-ensembling, we conduct experiments with different thresholds in the 20%-labeled setting. The thresholds we evaluate range from 0.5, such that at least half of the classifiers in the sub-ensemble have to agree to suggest a data point for pseudo-labeling, to 0.9. The results of these experiments are shown in Figure 3.2. As is visible from this graph, no confidence thresholds provide substantial performance improvements in binary data



(a) Multi-step co-ensembling results for binary data sets.



(b) Multi-step co-ensembling results for multiclass data sets.

Figure 3.1: Ensemble performance on OpenML 100 data sets, using multiple co-ensembling iterations. We plot the mean and variance of the relative change in error rate between the unweighted AUTO-SKLEARN ensemble and the refined ensemble from co-ensembling.

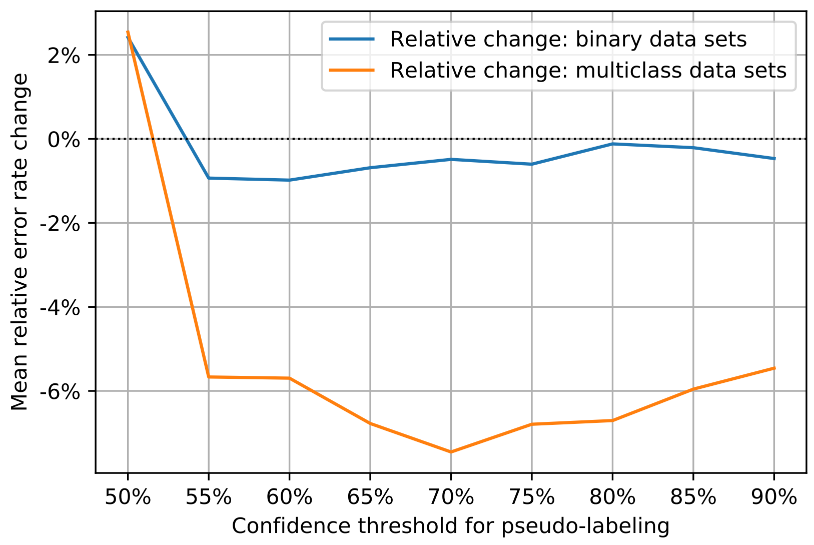


Figure 3.2: Performance evaluation of single-step co-ensembling with different confidence thresholds in the 20%-labeled setting.

sets. For multiclass data sets, the distribution is unimodal, and the largest performance gains are attained with $\theta = 0.7$.

It is not evident why such large differences exist between the performance of co-ensembling on binary data sets and the performance of co-ensembling on multiclass data sets, even when varying the confidence threshold. We identify two potential causes. Firstly, it is possible that an agreement of multiple base learners on a prediction inherently carries more significance in multiclass data sets. In binary data sets, and especially when the ensemble is relatively small, it is more likely that a sub-ensemble of learners agrees on a prediction by chance. Secondly, it is possible that the ensemble diversity in binary classification problems is generally smaller than in multiclass classification problems. In that case, the base learners would often be unable to exchange useful information. Further investigation of these potential causes is beyond the scope of this work, but remains an interesting topic for future research.

## Results

Using the previously established confidence threshold of 0.7 and a single co-ensembling iteration, we conduct extensive experiments in both the 10%-labeled and 20%-labeled settings. A summary of our results is provided in Table 3.2. Supporting the results of our earlier experiments, we see that the average reduction in error rate of single-step co-ensembling is 7.5% in the 20%-labeled setting and 6.8% in the 10%-labeled setting. Comparing this to the performance improvement of the unweighted AUTO-SKLEARN

Table 3.2: Performance comparison between the unweighted AUTO-SKLEARN ensemble and the ensemble refined with single-step co-ensembling. Includes mean absolute error rates and mean error differences.

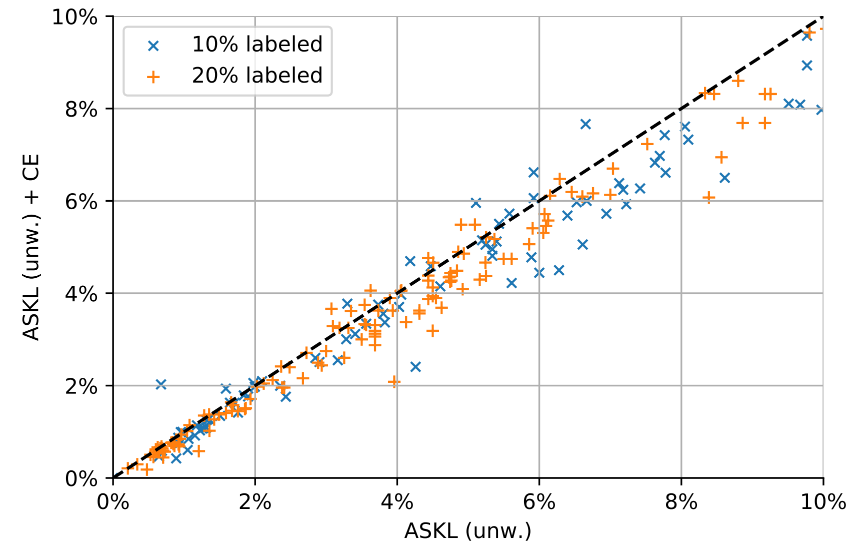
| Method | Fraction labeled | |
|---|---|---|
| | 10% | 20% |
| **Overall** | | |
| Baseline (unw. ASKL) | 0.177 | 0.151 |
| Mean relative change | -2.2% | -3.5% |
| Win/tie/loss | 45/2/31 | 42/2/34 |
| **Binary** | | |
| Baseline (unw. ASKL) | 0.158 | 0.144 |
| Mean relative change | 1.4% | -0.5% |
| Win/tie/loss | 18/1/25 | 16/2/26 |
| **Multiclass** | | |
| Baseline (unw. ASKL) | 0.198 | 0.158 |
| Mean relative change | -6.8% | -7.5% |
| Win/tie/loss | 27/1/6 | 26/0/8 |

ensemble over random forests from Table 3.1, co-ensembling yields a substantial improvement. Assessing statistical experiments with the Wilcoxon signed-rank test with a significance level of 5%, we find that the performance improvements observed in the multiclass data sets are indeed statistically significant; the differences observed in the binary data sets are not.

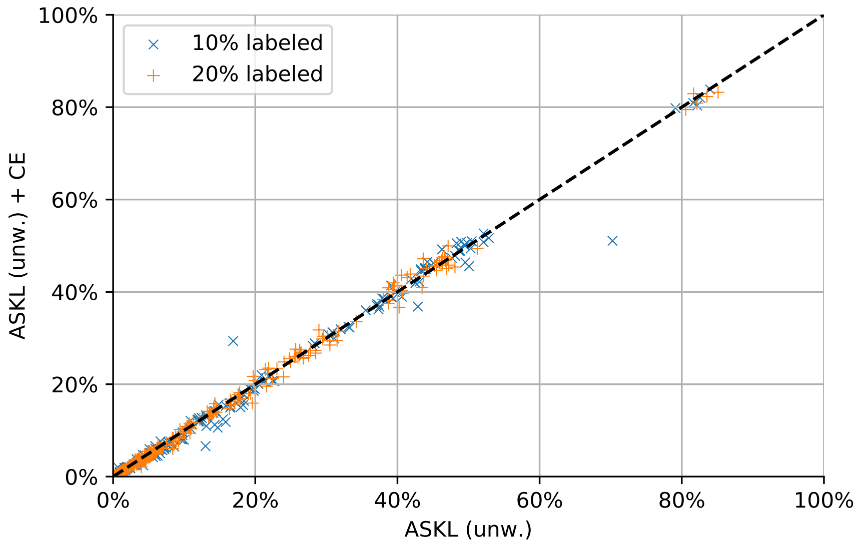
More importantly, however, co-ensembling yields a substantial performance improvement in over 75% of the data sets, both in the 10%-labeled and 20%-labeled settings. This signifies the robustness and genericness of single-step co-ensembling in multiclass problems. We further evaluate the robustness of our procedure by considering the per-data-set results. These are available in full in Appendix A.

We proceed to compare the error rates of the unweighted AUTO-SKLEARN ensemble and the ensemble obtained when using co-ensembling by means of a scatter plot. In Figure 3.3, all valid multiclass experiments are plotted. We provide both a scatter plot of the error rates within the 0 -10% range (in Figure 3.3a) and a full scatter plot (in Figure 3.3b).

These scatter plots provide confirmatory evidence for our earlier observations: co-ensembling is able to improve the unweighted ensemble from AUTO-SKLEARN in most cases. Furthermore, it rarely degrades the ensemble's performance substantially. This is an important feature of successful semi-supervised learning methods: the user should be confident that it does not perform worse than its supervised counterpart. The zoomed-in scatter plot in Figure 3.3a shows that significant performance improvements are obtained in ensembles with error rates in the range of 0 -10%. As the error rates obtained by AUTO-SKLEARN increase, the improvement obtained by co-ensembling appears to



(a) Zoomed-in scatter plot of error rates of valid multiclass experiments with error rates in range 0 -10%, comparing unweighted AUTO-SKLEARN to its co-ensembling extension.



(b) Scatter plot of error rates of all valid multiclass experiments, comparing unweighted AUTO-SKLEARN to its co-ensembling extension.

Figure 3.3: Scatter plots of AUTO-SKLEARN's unweighted ensemble error rates before and after co-ensembling.

decrease (see Figure 3.3b).

Single-step co-ensembling is able to consistently improve the performance of strong ensembles of classifiers in multiclass data sets. Performance improvements are obtained in more than three quarters of our multiclass classification problems, and substantial performance degradation is rarely observed. Furthermore, since co-ensembling retrains the existing ensemble, its time complexity is identical to the original ensemble of classifiers. Because the original, labeled data is always used when the classifiers are retrained, the total running time of the algorithm can be expected to be at least twice the running time of the supervised algorithm. The additional computation time attributed to the increase in labeled data can greatly vary, and depends primarily on the number of pseudo-labeled samples.

## 3.5 Conclusions

In this work, we have introduced a novel semi-supervised learning framework for training classifier ensembles. Our method, *co-ensembling*, iteratively introduces pseudo-labeled data to each of the classifiers in the ensemble by using the predictions of the sub-ensemble of all other classifiers. Our results demonstrate that such methods tend to have increased performance variance as the number of iterations grows. In that light, we proposed single-step co-ensembling, where the co-ensembling procedure is applied a single time, considering all unlabeled data for pseudo-labeling.

We have shown that single-step co-ensembling can consistently improve the performance of strong classifier ensembles on multiclass classification problems. Applying our ensembling method to ensembles constructed by AUTO-SKLEARN, a prominent automated machine learning system, we were able to consistently improve performance in multiclass classification problems. Firstly, we showed that using the ensemble constructed by AUTO-SKLEARN in an unweighted fashion improves performance over the standard, weighted ensemble. Secondly, we used single-step co-ensembling to reduce the error rate obtained by this unweighted ensemble by an average of close to 7% in multiclass classification problems under different fractions of labeled data. Our approach yielded performance improvements in 27 of the 34 multiclass classification problems we evaluated with 10% of the data being labeled. In the 20%-labeled setting, we achieved performance improvements in 26 of the 34 data sets.

The single-step co-ensembling procedure achieves state-of-the-art performance in multiclass automated machine learning problems. In future work, we intend to explore the possibilities of improving the performance in binary classification problems as well, which we have not been able to achieve so far. One promising approach to this problem would be to incorporate the diversity criterion of the ensemble explicitly in the AUTO-SKLEARN ensemble construction process, or even in its Bayesian optimization procedure. Furthermore, it would be interesting to evaluate the performance of the ensemble obtained by co-ensembling on previously unseen data, which corresponds to the fully inductive setting.

# Chapter 4

# Semi-supervised Decision Trees

In Chapter 3, we proposed a new single-step semi-supervised ensembling method for strong classifier ensembles. The method we proposed pseudo-labels data for each classifier by considering the predictions of the ensemble of all other classifiers. This pseudo-labeling approach is similar to the pseudo-labeling approach used by the *co-forest* algorithm [110], which we will discuss in this chapter. Co-forest is an extension of random forests to the semi-supervised setting, introducing unlabeled data to each decision tree based on the predictions of the other trees in the ensemble. It has been shown to outperform supervised random forests on a variety of data sets when using few trees, but no prior work explores their performance using a larger number of trees. We address this open question by evaluating co-forest with a varying number of trees on a benchmarking suite of diverse data sets. We show that, in a large majority of cases, the performance improvement of co-forest over random forests vanishes as the number of trees grows, and is in fact reversed when using more than 10 to 15 trees. Our results indicate that larger, supervised random forests outperform smaller, semi-supervised forests in almost all cases.

In addition to these findings, we propose a novel semi-supervised node splitting criterion for the construction of semi-supervised decision trees. We show that this addition can significantly improve performance for individual decision trees, but fails to yield improvements when the semi-supervised trees are incorporated into a random forest.

## 4.1 Introduction

Decision trees have long been a popular classification and regression model in supervised machine learning [120]. Due to their structured nature, where predictions are formed via sequences of simple decisions, they are easy to understand, implement, and explain. They require little tuning and data preprocessing and can be efficiently trained [81]. Although individual decision trees are susceptible to noise and generalize poorly, multiple decision trees can be combined to yield robust and powerful ensemble methods [212].

In ensemble learning, a distinction is often made between *bagging* and *boosting* methods. In bagging methods, each base learner is trained independently on a bootstrapped training set. When training is completed, predictions are formed by aggregating the results of the base learners. In classification, this is done with majority voting; in regression, the joint predictions are formed by averaging the predictions of the base learners. In boosting methods, learners are constructed sequentially based on the performance of previously constructed learners. After the training phase, predictions are formed by aggregating the predictions of the base learner.

The most well-known bagging model for decision trees is the random forest, which was proposed by Breiman [26]. In addition to bootstrapping the samples for each decision tree, it only considers a (random) subset of features for determining the best split at each node. This randomization step, which is an essential part of the training procedure of random forests, is intended to prevent overfitting. We cover the construction process of supervised decision trees in more detail in Section 4.3.1. Today, random forests are broadly used in machine learning applications [212]. Boosting methods are often applied to decision trees as well: Freund and Schapire introduced the highly popular AdaBoost, which is a general boosting framework that often uses decision trees as base leaners [63]. In recent years, decision trees have gained additional traction in boosting methods with the introduction of the distributed gradient boosting algorithm XGBoost [38].

Over the past two decades, several approaches have been suggested to include unlabeled data in the training process. Most of these approaches incorporate the unlabeled data into the classifiers using a wrapper method (see Section 2.3), re-training a decision tree or random forest multiple times by iteratively introducing unlabeled data labeled by the classifiers from the previous iteration. However, approaches also exist to directly incorporate the unlabeled data into the decision tree. For instance, Liu et al. propose to use an estimate of the sample density in the splitting criterion [118, 119].

Applying the well known self-training algorithm [205] to random forests, Leistner et al. suggest to probabilistically pseudo-label unlabeled samples independently for each tree based on the predictions of the full ensemble [108]. Tanha et al. also apply self-training to decision trees and random forests, using improved confidence predictions for individual decision trees [176]. Boosting methods typically pseudo-label samples for each new tree that is added to the ensemble [17, 123].

Li and Zhou propose an adaption of co-training, a wrapper method where base learners provide each other with pseudo-labeled data [21], to random forests [110]. They propose *co-forest*, which pseudo-labels unlabeled samples for each tree based on the predictions made by the ensemble of all other trees. This approach was extended by Deng and Guo, who attempt to prevent the influence of possibly mislabeled samples by removing "suspicious" pseudo-labelings based on $k$-nearest neighbors [55]. These approaches are compared to other wrapper methods by Triguero et al., who observe impressive performance gains compared to other single-view co-training methods [179].

Crucially, in evaluating co-forest, existing work constructs an ensemble consisting of only 6 trees. However, it has been shown that using more trees for supervised random forests generally yields highly superior performance in practice [134]. In fact, Osha et al. empirically showed that the area under the *receiver operator characteristic* (ROC) curve, which is commonly used to measure classifier performance [24], typically increases monotonically as the number of trees grows. Furthermore, they suggest that it convergences asymptotically. The question naturally arises whether co-forest also improves performance over supervised random forests when using more decision trees.

Here, we address this question and study the performance of co-forest in comparison with random forests as the number of trees grows. Since co-forest is an expansion of random forests (it reduces to a supervised random forest when no unlabeled data is present), we can directly evaluate the performance gain when introducing different amounts of unlabeled data. Comparing results on the OpenML 100 [182], a benchmarking suite of around 100 diverse data sets, we show that co-forest exhibits improved performance over random forests when few trees are used, but that this performance improvement quickly dwindles when the number of trees is increased. In fact, as the number of trees grows, co-forest performs significantly worse than supervised random forests.

Additionally, we study a simple density-estimation approach for incorporating unlabeled data into decision trees. We propose a node splitting approach similar to the approach by Liu et al. [118, 119], but with lower time complexity. We show that this approach improves the performance of decision trees, but yields no improvement compared to supervised random forests when used in an ensemble.

The remainder of this chapter is structured as follows. The co-forest algorithm is outlined in Section 4.2, followed by our new semi-supervised node splitting criterion in Section 4.4.2. We discuss our experiments and results in Section 4.4. Finally, we present our conclusions in Section 4.5.

## 4.2 Co-forest

The co-forest algorithm is a wrapper method for ensembles of decision trees. Wrapper methods iteratively introduce unlabeled data to regular, supervised classifiers by pseudo-labeling unlabeled data points using the classifiers being trained (see Section 2.3). *Self-training*, for instance, takes a base classifier and iteratively re-trains it by pseudo-labeling its most confident predictions in each iteration [205, 149]. *Co-training* uses two base classifiers which pass each other unlabeled data [21], and *tri-training* uses three base classifiers [214].

Co-forest can be seen as an extension of co-training and tri-training to an arbitrary number of base learners. It iteratively introduces unlabeled data to the decision tree classifiers by passing each decision tree the highly confident predictions of the ensemble of all other trees. These pseudo-labeled samples are weighted by the prediction confidence, defined as the fraction of classifiers predicting a particular label, varying the influence of the pseudo-labeled samples in the decision tree construction process based on their prediction confidence. Co-forest utilizes a stopping criterion based on an estimate of the generalization error, which is readily available due to the fact that random forests bootstrap labeled samples independently for each classifier. By comparing the true label of each sample to the label prediction of all trees that were not trained on that particular sample, one can calculate the *out-of-bag error*, an estimate of the generalization error.

The original formulation of the algorithm, as proposed by Li and Zhou, leaves open two questions, regarding (1) the subset of samples to label in the first pseudo-labeling iteration, and (2) bootstrapping of labeled samples and out-of-bag error-measurement [110]. We elaborate on these questions and address them after providing a more detailed, step-by-step overview of the co-forest algorithm.

### 4.2.1 The co-forest algorithm

Let $D_L = ((\mathbf{x}_i, y_i))_{i=1}^{l}$ denote the labeled data points, where $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathcal{Y}$ for each $i = 1, \ldots, n$. Furthermore, let $X_U = (\mathbf{x}_i)_{i=l+1}^{n}$ denote the unlabeled data points. Co-forest construction, then, proceeds as follows.

**Step 1: Random forest construction.** A supervised random forest is constructed. $K$ decision trees are trained on independently bootstrapped subsamples of the labeled data (i.e., for each decision tree, $l$ labeled data points are sampled with replacement). The number of features to consider in each decision tree node is set to $\log_2(m+1)$, where $m$ is the total number of available features, corresponding to Breiman's heuristic (see Section 4.3.1). The error estimate, used in the stopping criterion, is set to 0.5 for each classifier.

**Step 2: Pseudo-labeling.** For each individual tree $h_k, k = 1, \ldots, K$, the ensemble of all other decision trees determines which unlabeled data should be pseudo-labeled for $h_k$. This ensemble, which we will henceforth refer to as the *concomitant* ensemble, is denoted by $H_k$.

We consider the procedure for tree $h_k$ in iteration $t$. Firstly, the out-of-bag error $\hat{e}_{k,t}$ is calculated for the concomitant ensemble $H_k$. If this out-of-bag error is larger than the previous out-of-bag error, the performance of the concomitant ensemble is expected to have deteriorated and no data is pseudo-labeled for $h_k$. If the out-of-bag error has decreased, however, the concomitant ensemble pseudo-labels the unlabeled data points whose prediction confidence exceeds a certain threshold $\theta$. A subsampling scheme is used to limit the number of potential pseudo-labeled data points. This scheme, which is elaborately explained in [110], effectively limits the additional number of samples based on the estimated noise rate to prevent the additional samples from negatively impacting performance [3]. Let $W_{k,t}$ denote the sum of all sample weights for the pseudo-labeled samples for tree $h_k$ at iteration $t$. Then the maximum summed weight of the samples $U'_k$ to consider for pseudo-labeling for tree $h_k$ is calculated as

$$W_{\text{max}} = \frac{\hat{e}_{k,t-1} \cdot W_{k,t-1}}{\hat{e}_{k,t}}. \tag{4.1}$$

$U'_k$ is then determined by sampling from $U$ uniformly at random until the sum of the sample weights reaches $W_{\text{max}}$. Note that, in each iteration, the entire set of unlabeled data is considered for pseudo-labeling, and previous pseudo-labels are discarded.

**Step 3: Re-training.** In the third step, each tree is re-trained with its labeled samples and pseudo-labeled samples. If no trees receive unlabeled data (usually because all out-of-bag error rates have increased), the algorithm halts and the training phase is completed. Otherwise, steps 2 and 3 are repeated.

### 4.2.2 Subsampling and bootstrapping

The explanation above leaves open one major question: how do we determine $W_{\text{max}}$ in the first iteration of pseudo-labeling? At that point in time, $W_{k,t-1}$ is not available since no pseudo-labeling step has occurred before. In the originally proposed algorithm, Li and Zhou initialize $W_{k,0} = 0$, but that would cause $W_{\text{max}} = 0$, thereby preventing any pseudo-labeling from occurring at all. In the original software implementation [109], this problem is addressed by setting $W_{k,0} = \min(\frac{1}{10}|X_U|, 100)$, i.e., at most one tenth of the number of unlabeled samples and at most 100 in total. We use this heuristic as well, but note that imposing a constant limit on the summed weight of the samples makes the impact of the unlabeled data in the co-forest algorithm highly dependent on the data set size.

Secondly, in the original formulation of co-forest, each tree is trained on *all* labeled samples (in addition to the pseudo-labeled samples), instead of just the original in-bag labeled samples. This does not allow for an out-of-bag estimate of the error rate $\hat{e}$ in later iterations, or forces the error estimate to proceed on the training samples (which, of course, yields a highly biased error estimate). This problem is addressed in the source code [109], where each tree is only trained on its in-bag samples and the pseudo-labeled samples.

The entire algorithm, including these modifications, is presented in Algorithm 2. Clarifying some notation, we note that Subsample$(X, H_k, W_{\text{max}})$ denotes the uniform subsampling procedure of samples from $X$, weighted by the confidence estimates of $H_k$, up to a maximum total weight of $W_{\text{max}}$. The TrainTree$_k(D)$ function trains tree $h_k$ on samples $(\mathbf{x}, y) \in D$, where labeled samples have weight 1 and possible pseudo-labeled samples are weighted by the confidence estimates of $H_k$.

## 4.3 Semi-supervised node splitting

In addition to evaluating the performance of co-forest as the number of trees varies, we propose a generic extension of splitting criteria for decision tree construction by incorporating unlabeled data. In particular, we propose to include a loss term in existing splitting criteria that promotes the splitting point to lie in a low-density area. We use one-dimensional kernel density estimation to establish where such areas are. Before elaborating on this procedure, we outline the typical construction process of supervised decision trees.

### 4.3.1 Supervised decision tree construction

In the training phase, decision trees are constructed by iteratively splitting the data into two subsets based on some criterion. Usually, this *splitting criterion* is based on class impurity: the quality of the data split is determined by comparing the class impurity of the two subsets to the class impurity at the parent node. Starting at the root node, which contains all samples, the samples in each node are split into two new nodes. This process continues for all nodes with nonzero impurity until each node contains only samples from a single class. In most decision tree learning algorithms, only one-dimensional

**Algorithm 2** Co-forest

**Input:** Labeled data $D_L = ((\mathbf{x}_i, y_i))_{i=1}^{l}$, unlabeled data $X_U = (\mathbf{x}_i)_{i=l+1}^{n}$, number of trees $K$, confidence threshold $\theta$
**Output:** Trained decision tree ensemble $H$

```
 1: procedure Co-forest
 2:     for $k = 1, \dots, K$ do
 3:         $L_k \leftarrow \text{Bootstrap}(D_L)$
 4:         $h_k = \text{TrainTree}_k(L_k)$
 5:         $\hat{e}_{k,0} \leftarrow 0.5$
 6:         $W_{k,0} \leftarrow \min(\frac{1}{10}|X_U|, 100)$
 7:     end for
 8:     $t \leftarrow 0$
 9:     repeat
10:         $t \leftarrow t + 1$
11:         for $k = 1, \dots, K$ do
12:             $\hat{e}_{k,t} \leftarrow \text{OutOfBagError}(H_k, D_L)$
13:             $L'_{k,t} \leftarrow \emptyset$
14:             if $\hat{e}_{k,t} < \hat{e}_{k,t-1}$ then
15:                 $U'_{k,t} \leftarrow \text{Subsample}(X_U, H_k, \frac{\hat{e}_{k,t-1} \cdot W_{k,t-1}}{\hat{e}_{k,t}})$
16:                 $W_{k,t} \leftarrow 0$
17:                 for each $\mathbf{x}_i \in U'_{k,t}$ do
18:                     if $\text{Confidence}(H_k, \mathbf{x}_i) > \theta$ then
19:                         $L'_{k,t} \leftarrow L'_{k,t} \cup \{(\mathbf{x}_i, H_k(\mathbf{x}_i))\}$
20:                         $W_{k,t} \leftarrow W_{k,t} + \text{Confidence}(H_k, \mathbf{x}_i)$
21:                     end if
22:                 end for
23:             end if
24:         end for
25:         for $k = 1, \dots, K$ do
26:             if $L'_{k,t} \neq \emptyset$ then
27:                 $h_k = \text{TrainTree}_k(L_k \cup L'_{k,t})$
28:             end if
29:         end for
30:     until None of the trees in the random forest receives new data
31:     return $(h_1, \dots, h_K)$
32: end procedure
```

splits (i.e., splits based on a single feature) are considered, benefiting both computational efficiency in the construction phase and understandability in the inference phase.

When only one-dimensional splits are considered, the finite set of potential splitting points with different class impurities can be obtained relatively easily. One can simply consider a sorted list of feature values in the data set for each feature, and calculate the class impurity when using a value between each subsequent pair of feature values as the splitting point. Depending on the specific implementation of the algorithm, a subset of all possible features and splitting points can be used. For instance, random forests consider a random subset of $\log_2(m+1)$ features [26]; *extremely randomized trees* use a random subset of $\sqrt{m}$ features, and consider a single random splitting point for each feature [65]. Since we consider random forests in our experiments, we use Breiman's heuristic [26].

The splitting criterion determines which of the potential splitting features and splitting values is used. A plethora of splitting criteria exist, but it has been noted, both in empirical and in theoretical studies, that the choice of splitting criterion has little influence on the performance of the constructed tree in supervised scenarios [30, 127, 142]. In our work, we use the Gini criterion, which is used in the popular CART decision tree training algorithm [27]. Considering a decision tree node with a set of items $S$ with labels $(y_i)_{i \in S}$, let $\hat{p}_c$ denote the fraction of these items labeled with label $c \in \mathcal{Y}$. The Gini impurity $I(S)$ is then defined as

$$I(S) = 1 - \sum_{c \in \mathcal{Y}} \left[\hat{p}_c^2\right]. \tag{4.2}$$

The feature split yielding the lowest weighted Gini index for the two child nodes is used. Weighting occurs by the number of samples in each node.

The splitting procedure used in decision tree construction algorithms can be seen to optimize an objective function with respect to two parameters: the feature to split by and the feature value to split at. Let $t \in \{1, \ldots, m\}$ denote the index of the feature to split by, and let $\gamma \in \mathbb{R}$ denote the splitting value. Considering a node with samples $S$, we define $S_1$ as the samples $\mathbf{x} \in S$ where $x_t < \gamma$, and $S_2$ as the samples where $x_t \geq \gamma$. The optimization problem, then, can be formulated as follows:

$$\underset{t,\gamma}{\text{minimize}} \quad \mathcal{L}(t,\gamma) = \frac{|S_1|}{|S|} I(S_1) + \frac{|S_2|}{|S|} I(S_2). \tag{4.3}$$

As noted before, the splitting values $\gamma$ that yield distinct values for $\mathcal{L}(s,\gamma)$ can be easily enumerated since we are considering one-dimensional splits. We can simply sort the feature values of all samples for each individual feature prior to tree construction, which can be done in $O(d \cdot n \cdot log(n))$. We denote these potential splitting values by $\Gamma(S,t)$.

The decision tree construction process has been studied extensively, and a plethora of heuristics and tuning methods exist. For example, decision tree construction can be halted when the number of samples in a node is too small, or when a certain impurity has been reached. For an extensive survey of decision tree construction methods, we refer the reader to [148].

### 4.3.2 Regularization by density

One of the assumptions often used in semi-supervised learning is the *maximum-margin* assumption. It states that the decision boundary produced by the classifier should not be close to any data points. This assumption is made explicit in (semi-supervised) support vector machines, where the distance between the boundary and the data is maximized (see Section 2.5). The maximum-margin assumption can also be considered from the perspective of the marginal probability density $p(x)$: if the distance of the decision boundary to samples is large, it probably lies in an area where $p(x)$ is low.

We propose to incorporate this perspective on the maximum-margin assumption directly into the objective function $\mathcal{L}(s, \gamma)$ from Equation 4.3. We note that, for modeling $p(x)$, we need not rely solely on the labeled data: we can use all the unlabeled data to obtain a better estimate of the marginal probability distribution. Of course, obtaining a model of $p(x)$ can be computationally expensive. By using kernel density estimation on the one-dimensional data in each potential split, we overcome this problem.

Consider a node with data points $S$ and potential split feature and split value $t$ and $\gamma$, respectively. Our estimate $\hat{p}_t(x)$ is then an estimate of the density in our node for feature $t$ at point $x$, which can be calculated as

$$\hat{p}_t(x) = \frac{1}{|S|} \sum_{i \in S} k\left(\frac{x - x_{i,t}}{h}\right), \tag{4.4}$$

where $x_{i,t}$ is the value of feature $t$ for sample $\mathbf{x}_i$, $k$ is some kernel function, and $h$ is a hyperparameter that controls the breadth of the influence of each sample (the *scaling factor*). We use the exponential kernel, i.e., $k(d) \propto \exp(-d^2)$.

Our proposed method uses $\hat{p}_t(x)$ directly by incorporating it into the optimization problem in Equation 4.3: we simply add a loss term $\hat{p}_t(x)$, penalizing decision boundaries in high-density areas. This approach poses two potential problems. Firstly, when incorporating $\hat{p}_t(\gamma)$ into the density function, the set of values of $\gamma$ that yield distinct values for $L(s, \gamma)$ is potentially infinite and cannot be easily enumerated anymore. Secondly, even when considering only a limited number of data points, computing $\hat{p}_t(\gamma)$ can be computationally expensive.

We tackle the first problem by only considering splitting points from our original set of potential splitting points. The second problem is the computational complexity of kernel density estimation: the time complexity for calculating $\hat{p}(\gamma)$ for all splitting points $\gamma$ is $O(|S|^2)$. However, since we have access to sorted feature values, we can obtain reasonable estimates of $\hat{p}(x)$ in constant time during training: as most of density of a Gaussian distribution is concentrated around its mean, we only need to consider a small window of samples around $\gamma$ to obtain a reasonable estimate for $\hat{p}(\gamma)$. As such, the densities for all splitting points $\gamma$ for a feature $s$ can be calculated simultaneously in total time $O(|S|)$.

Lastly, we note that we wish to consider the relative estimated densities. Therefore, in our objective function, we normalize $\hat{p}_t(\gamma)$ by the maximum $\hat{p}_t(\gamma)$ for the candidate splitting points for the current feature $t$. Our semi-supervised objective function

$\mathcal{L}'(t, \gamma)$ in the splitting procedure, then, is defined as

$$\mathcal{L}'(t, \gamma) = (1 - \lambda) \cdot \mathcal{L}(t, \gamma) + \lambda \cdot \frac{\hat{p}_t(\gamma)}{\max_{\gamma' \in \Gamma(S,t)} \hat{p}_t(\gamma')}, \tag{4.5}$$

where $\lambda \in [0, 1]$ governs the trade-off between the influence of the supervised and the unsupervised loss terms.

In related work, Liu et al. propose semi-supervised decision trees with *oblique* node splitting: each split is based on multiple features [118]. They also use kernel density estimation to estimate the marginal density along the splitting hyperplane. However, instead of explicitly incorporating an unsupervised objective into the loss function, they use the unlabeled data to modify $\hat{p}_c$ (which is a surrogate for $p(c|S)$). They follow an approach similar to pseudo-labeling, determining the contribution of each unlabeled data point to $\hat{p}_c$ by estimating the probability that the unlabeled sample belongs to class $c$. This probability is estimated using kernel density estimation.

Our method differs from their node splitting criterion in several ways. Firstly, we do not use oblique node splitting, allowing us to enumerate all potential splitting points for the supervised objective. Secondly, we do not pseudo-label the data, but use the estimated marginal density directly. This allows our approach to be used in any node splitting approach, even when it does not use $\hat{p}_c$ (such as maximum-margin splitting criteria). Furthermore, because we use one-dimensional node splitting, we can precompute the sorted feature values for each feature, thereby reducing the time complexity of the kernel density estimation. Like the method proposed by Liu et al. [118], our method naturally extends to random forests. However, as we demonstrate empirically, our method does not improve performance when used in random forests.

## 4.4 Experiments and results

Recall from Section 4.1 that our main goal is to evaluate the performance of co-forest as the number of base learners varies. In particular, we wish to quantitatively assess the performance difference between supervised random forests and their semi-supervised extension co-forest. Furthermore, we wish to evaluate the performance of our semi-supervised node splitting algorithm.

The experimental setup for both problems is similar. Our experiments are conducted on a diverse variety of classification data sets from the *OpenML 100* [19], a benchmarking suite consisting of both real-world and synthetic data sets (for more information about this data set, please consult Section 3.4.1 and Appendix C). Since decision trees are not naturally equipped to handle missing data and we do not wish to potentially influence our results with imputation steps, we discarded 18 data sets with missing values. The remaining 79 data sets are listed in the main results table (Table 4.1). For a full overview of all benchmark suite data sets and their characteristics, we refer to Appendix C.

To evaluate the behavior of co-forest with different quantities of labeled data, experiments are conducted with 5%, 10%, 15%, and 20% of the data being labeled (and the rest unlabeled). Varying the amount of labeled data relative to the number of data points in the data set is common when experiments are conducted on data sets with diverse sample sizes [179].

In outlining the specific experiments and their results, we distinguish between experiments concerning co-forest and experiments concerning semi-supervised node splitting. We cover our experiments on co-forest extensively in Section 4.4.1, and provide a brief overview of the decision tree experiments in Section 4.4.2.

### 4.4.1 Co-forest

For each of the 79 data sets, we conduct experiments on varying numbers of trees and with varying quantities of labeled and unlabeled data. Each of the experiments is conducted 30 times with different train/test splits to obtain confident performance estimates.

Each experiment proceeds as follows. The data is split into labeled and unlabeled sets randomly (according to the prespecified ratio of labeled to unlabeled data). Then, the labeled training data is used to train a supervised random forest (which corresponds with the first step of the co-forest algorithm). This random forest is refined iteratively with unlabeled data using the co-forest algorithm. The resulting ensemble is evaluated on the test set (i.e., the unlabeled data), to yield the final co-forest performance; it is compared against the performance of the random forest constructed in the first step on the same test data.

The number of trees is varied from 2 to 100 with increasing intervals. The difference between each pair of subsequent ensemble sizes is gradually increased to reflect the observation that the addition of a single tree has more influence on smaller ensembles than on larger ensembles. Starting with the real number 2, we iteratively increase the current number with 10% until we surpass 100. We round the obtained numbers, leaving 33 distinct ensemble sizes. The lower limit of two trees does not allow for confidence estimates in the concomitant ensemble, and our results show that this leads to performance degradation. The upper limit of 100 trees corresponds to empirical evidence of random forest convergence [134]. We increment the number of trees in steps of 10% so that the number of experiments in the lower ranges is larger than the number of experiments in the higher ranges; the influence of adding a single additional tree can be expected to decrease as the number of trees grows. The other hyperparameters are kept constant in accordance with the originally proposed co-forest algorithm [110]: the confidence threshold $\theta$ is set to $0.75$, and the maximum number of features to consider at each split is set to $\log_2(m+1)$. We note that the number of trees was fixed to 6 in previous studies of co-forest.

An overview of the relative performance differences between co-forest and random forests with varying number of trees and with different labeled/unlabeled rates is provided in Figure 4.1. As can be seen from this figure, the introduction of unlabeled data to the random forest yields performance improvements when the number of trees is low, but quickly vanishes as the number of trees increases. When more than 10-15 trees are used, the pattern is in fact reversed, and the introduction of unlabeled data only worsens performance of the forest. This pattern can be observed for all labeled/unlabeled ratios. In fact, the performance curves can be seen to cross the x-axis, where the performance difference is 0, at roughly the same number of trees for all ratios. This emphasizes

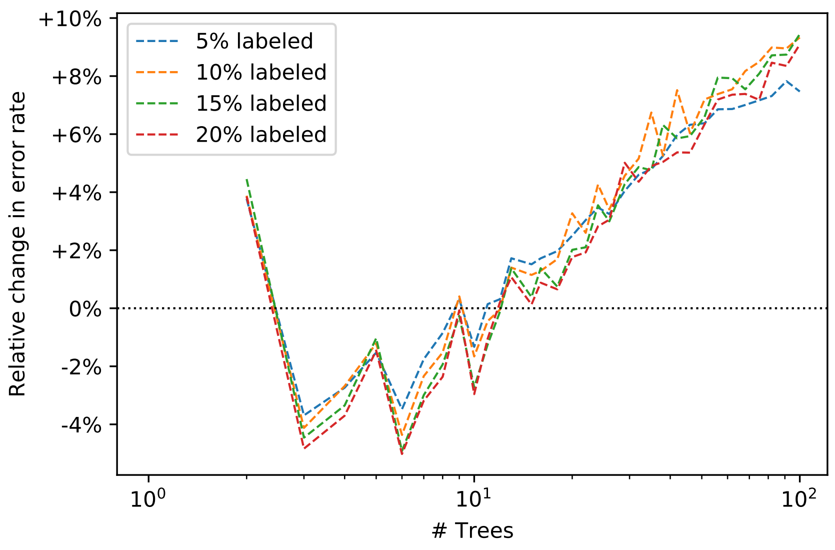


Figure 4.1: Mean difference in error rates between random forest and co-forest with varying number of trees and for different labeled/unlabeled ratios. Negative values indicate that co-forest outperformed random forest.

the dependence of the relative performance of co-forest on the number of trees: independent of the amount of labeled data, co-forest only yields improvements when the number of trees is low.

Interestingly, no clear dependence can be observed in Figure 4.1 between the performance improvement of co-forest and the ratio of labeled to unlabeled data. This could potentially be explained by the heuristic used for limiting the number of samples to consider for pseudo-labeling, as explained in Section 4.2.2: at any time, at most 100 samples are pseudo-labeled, indendendent of the number of labeled and unlabeled data points.

When using few trees (in particular, when using fewer than 10 trees), the performance improvement of co-forest is strongly influenced by the addition or removal of a single tree. For example, when using 4 or 6 trees, the relative reduction in error rate between obtained by co-forest when compared to random forests is around 3-5%; when using 5 trees, the reduction is around 1%.

To evaluate the absolute performance of co-forest and random forests, we plot the absolute mean error rates with varying number of trees and with different labeled/unlabeled rates in Figure 4.2. The behavior observed in the evaluation of the relative performance changes is reflected here as well: as the number of trees grows, the performance gain of co-forest over random forests decreases, and the effect reverses as around 10-15 trees are reached.

The figure also shows that, as expected, using more trees monotonically improves

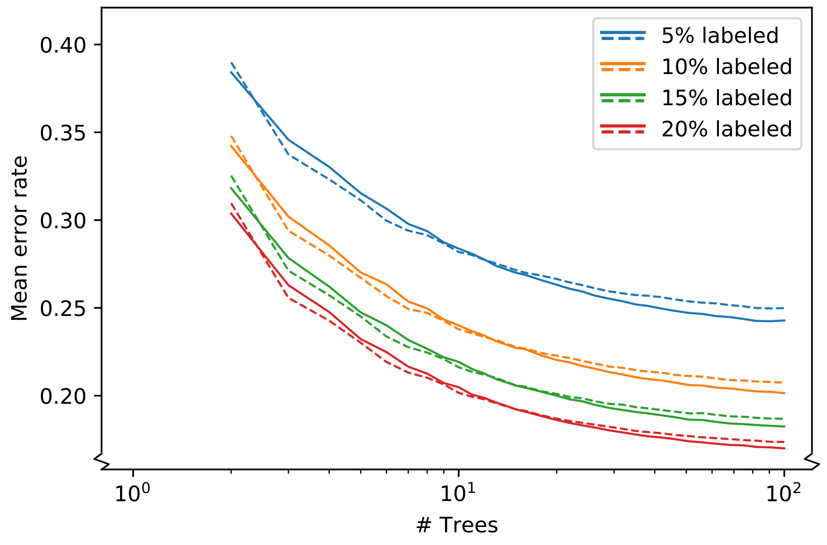


Figure 4.2: Mean error rates of co-forest and random forest with varying number of trees and for different labeled/unlabeled ratios. Solid lines indicate random forest performance; dashed lines indicate co-forest performance.

the mean performance; the same behavior occurs when more labeled data is added. We can also see that, although co-forest improves over random forests when using few trees, we can still obtain greatly superior performance when using larger, supervised random forests. We note that, since the error rates are averaged over all experiments, the influence of experiments on data sets with large random forest error rates is disproportionately large. However, the patterns we observe in these figures are generally reflected in individual data sets as well, as the per-data-set results we discuss later will show.

We assess the statistical significance of our results by applying the Wilcoxon signed-rank test [195] to the mean error rates for our data sets, as is common when comparing two classifiers on multi-data-set experiments [54]. We use a standard significance level of 5%. We find that the performance difference between co-forests and random forests is statistically significant when using up to 10 to 12 trees, and when using more than 20 to 22 trees. The specific thresholds vary slightly depending on the amount of labeled data.

To compare the performance of co-forest to random forests per data set, we list the mean error rates per data set for varying numbers of trees in Table 4.1. The table reflects experiments conducted with 10% of the data labeled using 6, 12, and 24 trees; we report the error rates and relative differences averaged over 30 experiments per data set and ensemble size. The data confirms that the per data set performance generally corresponds to the global patterns observed earlier: (1) the performance of both co-forest and random forests improves as the number of trees grows, and (2) co-forest is better than random forests when using few trees, but gets progressively worse than random forests as the number of trees grows. The same behavior was observed when conducting these experiments using different labeled data fractions.

We discuss two potential causes for the decreasing effectiveness of co-forest as the number of trees grows.

Firstly, co-forest reduces the diversity of the ensemble of decision trees. Ensemble *diversity*, the property that the base learners in an ensemble make uncorrelated errors, is a determining factor in the improvement ensembles provide over individual learners (see, e.g., [56, 82] and Section 3.3). By repeatedly exchanging pseudo-labeled data between classifiers, co-ensembling explicitly reduces the diversity of the base learners. Although this effect is present in both smaller and larger forests, it is conceivable that it is more pronounced in smaller forests: in larger forests, less variance can be expected in the fraction of learners agreeing on a prediction, thereby causing more unlabeled data points to be pseudo-labeled consistently.

Secondly, the potential performance improvement the introduction of unlabeled data can provide diminishes as the performance of the purely supervised ensemble improves. As we have demonstrated, the error rate of random forests generally decreases as the number of trees grows. When the error rate is high, there are many predictions that could be changed to yield performance improvements. When the error rate is low, this is not the case. Any change in the prediction the ensemble makes has a much higher probability of decreasing the error rate when the original error rate is high. In other words, it is easier to improve weak ensembles of learners than it is to improve strong ensemble of learners.

Thirdly, the out-of-bag error estimate, which is unbiased in bagging methods, may no longer be unbiased as pseudo-labeled data is exchanged between learners. Co-forest relies on the out-of-bag error estimate in its stopping criterion: a tree is only provided with new pseudo-labeled data if its out-of-bag error is lower than in the previous iteration. Thus, incorrect out-of-bag error estimates may influence the performance of co-forest. Let $h_k$ be the learner under consideration, and let $H_k$ be its concomitant ensemble. Furthermore, let $\tilde{L}_k = D_L \backslash L_k$ denote the out-of-bag samples of $h_k$. Then the out-of-bag error estimate for $h_k$ is based on the predictions of $h_k$ on $\tilde{L}_k$. The calculation of the out-of-bag error, then, relies on the assumption that the decision boundary of $h_k$ was not influenced by its out-of-bag samples. Now, assume that the concomitant ensemble $H_k$ pseudo-labels a sample $\mathbf{x}^* \in X_U$ for $h_k$. In that case, the pseudo-label for $\mathbf{x}^*$ is based on the predictions of the learners in $H_k$, which may well be trained on some samples from $\tilde{L}_k$. In the next iteration, $h_k$ is trained on its labeled samples $L_k$ and its newly pseudo-labeled sample $\mathbf{x}^*$. Now, the decision boundary of $h_k$ indirectly depends on samples in $\tilde{L}_k$, violating the out-of-bag error assumption. In smaller forests, one can expect that some samples are included in the labeled data sets of very few base learners. Such samples have limited influence on the pseudo-labeling procedure. Consequently, the bias in the out-of-bag error estimate can be expected to be smaller when fewer trees are used.

Table 4.1: Performance comparison between co-forest (CF) and random forest (RF) with 6, 12, and 24 trees using 10% labeled data. Includes CF and RF mean error rate per data set and relative difference between them.

| Data set | 6 trees | | | 12 trees | | | 24 trees | | |
|---|---|---|---|---|---|---|---|---|---|
| | RF | CF | Diff | RF | CF | Diff | RF | CF | Diff |
| ada-agnostic | 0.194 | 0.187 | **-3.2%** | 0.182 | 0.180 | **-0.9%** | 0.177 | 0.173 | **-2.1%** |
| anacat-authors | 0.125 | 0.081 | **-33.8%** | 0.074 | 0.067 | **-6.0%** | 0.048 | 0.061 | 28.3% |
| anacat-dmft | 0.813 | 0.813 | **-0.0%** | 0.812 | 0.812 | **-0.0%** | 0.813 | 0.813 | 0.0% |
| artificial-characters | 0.438 | 0.438 | 0.1% | 0.414 | 0.413 | **-0.2%** | 0.402 | 0.404 | 0.4% |
| australian | 0.201 | 0.171 | **-13.8%** | 0.169 | 0.151 | **-9.6%** | 0.157 | 0.156 | **-0.2%** |
| balance-scale | 0.239 | 0.245 | 3.6% | 0.227 | 0.234 | 3.8% | 0.217 | 0.230 | 6.3% |
| bank-marketing | 0.109 | 0.108 | **-0.2%** | 0.105 | 0.105 | **-0.4%** | 0.102 | 0.102 | **-0.0%** |
| banknotes | 0.052 | 0.047 | **-7.2%** | 0.046 | 0.049 | 12.4% | 0.042 | 0.049 | 17.9% |
| bioresponse | 0.353 | 0.348 | **-1.2%** | 0.330 | 0.325 | **-1.4%** | 0.310 | 0.311 | 0.3% |
| blood-donors | 0.268 | 0.269 | 0.7% | 0.265 | 0.268 | 0.8% | 0.264 | 0.263 | **-0.6%** |
| car | 0.154 | 0.154 | 0.7% | 0.137 | 0.155 | 13.6% | 0.128 | 0.150 | 18.5% |
| cardiotocography | 0.051 | 0.010 | **-78.6%** | 0.026 | 0.016 | **-32.1%** | 0.017 | 0.017 | 54.2% |
| climate-model | 0.092 | 0.090 | **-2.5%** | 0.091 | 0.086 | **-4.6%** | 0.087 | 0.086 | **-1.1%** |
| cmc | 0.527 | 0.525 | **-0.3%** | 0.508 | 0.508 | **-0.1%** | 0.505 | 0.503 | **-0.4%** |
| cnae-9 | 0.328 | 0.301 | **-7.8%** | 0.269 | 0.253 | **-5.7%** | 0.229 | 0.222 | **-2.3%** |
| collins | 0.657 | 0.657 | 0.0% | 0.584 | 0.586 | 0.4% | 0.543 | 0.545 | 0.4% |
| credit-g | 0.302 | 0.292 | **-3.2%** | 0.289 | 0.288 | **-0.2%** | 0.281 | 0.284 | 1.0% |
| diabetes | 0.302 | 0.279 | **-7.4%** | 0.287 | 0.274 | **-4.4%** | 0.275 | 0.275 | **-0.1%** |
| eeg-eye-state | 0.230 | 0.229 | **-0.6%** | 0.197 | 0.197 | 0.2% | 0.176 | 0.179 | 1.4% |
| electricity | 0.188 | 0.186 | **-0.9%** | 0.170 | 0.171 | 0.2% | 0.162 | 0.163 | 0.5% |
| gas-drift | 0.056 | 0.050 | **-9.8%** | 0.039 | 0.037 | **-4.8%** | 0.031 | 0.031 | 0.8% |
| gesture-phase | 0.531 | 0.531 | 0.0% | 0.502 | 0.502 | 0.0% | 0.477 | 0.478 | 0.2% |
| gina-agnostic | 0.255 | 0.238 | **-6.1%** | 0.213 | 0.207 | **-2.4%** | 0.182 | 0.179 | **-1.1%** |
| har | 0.128 | 0.126 | **-0.9%** | 0.091 | 0.095 | 4.9% | 0.073 | 0.076 | 4.5% |
| hill-valley | 0.494 | 0.494 | **-0.1%** | 0.490 | 0.493 | 0.5% | 0.490 | 0.491 | 0.1% |

Table continues on next page

Table 4.1: Performance comparison between co-forest (CF) and random forest (RF) with 6, 12, and 24 trees using 10% labeled data. Includes CF and RF mean error rate per data set and relative difference between them.

| Data set | 6 trees | | | 12 trees | | | 24 trees | | |
|---|---|---|---|---|---|---|---|---|---|
| | RF | CF | Diff | RF | CF | Diff | RF | CF | Diff |
| ilpd | 0.335 | 0.315 | **-5.3%** | 0.319 | 0.313 | **-1.7%** | 0.318 | 0.311 | **-2.3%** |
| internet-ads | 0.061 | 0.059 | **-3.4%** | 0.054 | 0.059 | 10.3% | 0.050 | 0.056 | 10.6% |
| isolet | 0.338 | 0.341 | 0.9% | 0.242 | 0.242 | 0.2% | 0.175 | 0.177 | 0.9% |
| japanese-vowels | 0.170 | 0.168 | **-1.1%** | 0.125 | 0.126 | 1.2% | 0.102 | 0.104 | 1.5% |
| kc1 | 0.179 | 0.167 | **-5.9%** | 0.170 | 0.162 | **-4.3%** | 0.168 | 0.161 | **-4.4%** |
| kc2 | 0.194 | 0.183 | **-5.4%** | 0.186 | 0.183 | **-1.3%** | 0.187 | 0.187 | 0.1% |
| kr-vs-kp | 0.091 | 0.067 | **-23.7%** | 0.065 | 0.057 | **-11.7%** | 0.057 | 0.053 | **-5.1%** |
| led-display-domain | 0.386 | 0.385 | **-0.1%** | 0.367 | 0.365 | **-0.6%** | 0.346 | 0.350 | 1.4% |
| letter | 0.229 | 0.230 | 0.8% | 0.187 | 0.187 | 0.3% | 0.160 | 0.161 | 0.8% |
| madelon | 0.490 | 0.491 | 0.3% | 0.482 | 0.479 | **-0.7%** | 0.475 | 0.474 | **-0.1%** |
| magic-telescope | 0.168 | 0.166 | **-1.2%** | 0.153 | 0.152 | **-1.1%** | 0.146 | 0.146 | 0.1% |
| mfeat-factors | 0.188 | 0.162 | **-13.1%** | 0.127 | 0.128 | 1.3% | 0.104 | 0.102 | **-1.8%** |
| mfeat-fourier | 0.406 | 0.409 | 0.8% | 0.337 | 0.338 | 0.5% | 0.285 | 0.286 | 0.8% |
| mfeat-karhunen | 0.364 | 0.361 | **-0.5%** | 0.255 | 0.254 | 0.2% | 0.177 | 0.184 | 4.4% |
| mfeat-morph | 0.341 | 0.329 | **-3.5%** | 0.325 | 0.327 | 0.6% | 0.321 | 0.322 | 0.3% |
| mfeat-pixel | 0.202 | 0.177 | **-11.5%** | 0.134 | 0.130 | **-1.9%** | 0.098 | 0.094 | **-3.1%** |
| mfeat-zernike | 0.403 | 0.395 | **-1.9%** | 0.344 | 0.343 | **-0.3%** | 0.308 | 0.310 | 0.6% |
| micro-mass | 0.666 | 0.666 | **-0.0%** | 0.601 | 0.601 | 0.0% | 0.554 | 0.554 | 0.0% |
| mnist-784 | 0.159 | 0.160 | 0.6% | 0.107 | 0.107 | 0.1% | 0.080 | 0.080 | 0.1% |
| monks-1 | 0.249 | 0.250 | 2.0% | 0.212 | 0.218 | 4.9% | 0.202 | 0.235 | 17.1% |
| monks-2 | 0.373 | 0.365 | **-1.9%** | 0.366 | 0.356 | **-2.5%** | 0.361 | 0.355 | **-1.5%** |
| monks-3 | 0.087 | 0.051 | **-35.6%** | 0.060 | 0.042 | **-24.3%** | 0.051 | 0.041 | **-10.8%** |
| mozilla4 | 0.068 | 0.065 | **-5.0%** | 0.064 | 0.062 | **-2.6%** | 0.062 | 0.061 | **-1.4%** |
| nomao | 0.060 | 0.059 | **-0.7%** | 0.052 | 0.052 | **-0.5%** | 0.049 | 0.049 | 1.0% |
| optdigits | 0.146 | 0.137 | **-5.4%** | 0.093 | 0.091 | **-1.8%** | 0.067 | 0.069 | 3.4% |
| ozone-level-8hr | 0.065 | 0.063 | **-2.8%** | 0.063 | 0.062 | **-0.3%** | 0.063 | 0.063 | 0.3% |
| pc1 | 0.076 | 0.073 | **-3.1%** | 0.073 | 0.070 | **-3.6%** | 0.073 | 0.070 | **-3.5%** |
| pc3 | 0.116 | 0.107 | **-7.4%** | 0.112 | 0.104 | **-6.6%** | 0.111 | 0.104 | **-6.1%** |
| pc4 | 0.124 | 0.123 | **-0.7%** | 0.123 | 0.122 | **-0.7%** | 0.121 | 0.121 | 0.3% |
| pendigits | 0.061 | 0.057 | **-7.0%** | 0.044 | 0.044 | **-0.9%** | 0.036 | 0.037 | 3.3% |
| phoneme | 0.185 | 0.180 | **-2.7%** | 0.174 | 0.173 | **-0.4%** | 0.166 | 0.168 | 0.9% |
| plants-margin | 0.811 | 0.811 | 0.0% | 0.751 | 0.751 | 0.0% | 0.696 | 0.696 | 0.0% |
| plants-shape | 0.795 | 0.795 | 0.0% | 0.753 | 0.753 | 0.0% | 0.725 | 0.725 | 0.0% |
| plants-texture | 0.791 | 0.791 | 0.0% | 0.725 | 0.725 | 0.0% | 0.673 | 0.673 | 0.0% |
| qsar-biodeg | 0.220 | 0.212 | **-3.4%** | 0.205 | 0.210 | 2.5% | 0.194 | 0.209 | 8.1% |
| robot-navigation | 0.050 | 0.040 | **-17.8%** | 0.038 | 0.036 | **-3.0%** | 0.029 | 0.034 | 20.0% |
| satimage | 0.153 | 0.152 | **-1.0%** | 0.139 | 0.137 | **-1.4%** | 0.130 | 0.130 | 0.6% |
| scene | 0.173 | 0.177 | 2.9% | 0.163 | 0.175 | 7.6% | 0.161 | 0.176 | 9.7% |
| segment | 0.090 | 0.083 | **-6.7%** | 0.079 | 0.081 | 4.0% | 0.073 | 0.079 | 9.6% |
| semeion | 0.433 | 0.432 | **-0.3%** | 0.332 | 0.332 | 0.3% | 0.267 | 0.268 | 0.6% |
| spambase | 0.106 | 0.094 | **-10.8%** | 0.088 | 0.083 | **-5.6%** | 0.078 | 0.080 | 2.5% |
| splice | 0.230 | 0.223 | **-1.7%** | 0.161 | 0.166 | 4.3% | 0.127 | 0.138 | 9.4% |
| steel-plates | 0.201 | 0.204 | 2.5% | 0.162 | 0.183 | 16.2% | 0.138 | 0.182 | 35.6% |
| sylva-agnostic | 0.041 | 0.048 | 17.2% | 0.036 | 0.044 | 23.9% | 0.033 | 0.043 | 31.0% |
| synthetic-control | 0.235 | 0.194 | **-16.6%** | 0.171 | 0.175 | 5.4% | 0.145 | 0.166 | 15.4% |
| tamilnadu | 0.136 | 0.160 | 26.5% | 0.086 | 0.095 | 17.3% | 0.057 | 0.068 | 26.6% |
| texture | 0.127 | 0.120 | **-5.4%** | 0.099 | 0.100 | 0.6% | 0.086 | 0.091 | 6.8% |
| theorem-proving | 0.538 | 0.538 | 0.0% | 0.521 | 0.521 | 0.0% | 0.510 | 0.510 | 0.1% |
| tic-tac-toe | 0.295 | 0.287 | **-2.2%** | 0.278 | 0.273 | **-1.3%** | 0.257 | 0.267 | 4.0% |

Table continues on next page

Table 4.1: Performance comparison between co-forest (CF) and random forest (RF) with 6, 12, and 24 trees using 10% labeled data. Includes CF and RF mean error rate per data set and relative difference between them.

| Data set | 6 trees | | | 12 trees | | | 24 trees | | |
|---|---|---|---|---|---|---|---|---|---|
| | RF | CF | Diff | RF | CF | Diff | RF | CF | Diff |
| vehicle | 0.401 | 0.388 | **-3.1%** | 0.369 | 0.361 | **-1.8%** | 0.345 | 0.348 | 0.9% |
| vowel | 0.546 | 0.546 | 0.0% | 0.493 | 0.491 | **-0.4%** | 0.457 | 0.462 | 1.3% |
| waveform-5000 | 0.251 | 0.242 | **-3.4%** | 0.209 | 0.206 | **-1.2%** | 0.186 | 0.184 | **-0.7%** |
| wdbc | 0.088 | 0.079 | **-9.1%** | 0.075 | 0.078 | 8.2% | 0.072 | 0.080 | 11.7% |
| wilt | 0.035 | 0.037 | 3.5% | 0.035 | 0.037 | 5.0% | 0.034 | 0.037 | 8.8% |

### 4.4.2 Semi-supervised node splitting

We evaluate the performance of our semi-supervised node splitting criterion on decision trees and random forests with 10 trees. The supervised loss term in the splitting criterion is the Gini impurity. In random forests, we bootstrap the samples for each tree and use $\log_2(m + 1)$ randomly selected features at each split, corresponding to Breiman's proposal for random forests [26]. To limit the influence of unlabeled data points relative to the labeled data points in our loss function, we choose $\lambda = 0.01$. This limitation of the influence of unlabeled data ensures that the unlabeled data can only make a significant difference when there is a small difference in supervised performance between candidate splitting points.

The results of these experiments, averaged over all data sets, are listed in Table 4.2. As can be seen from this table, the semi-supervised decision trees generally outperform supervised decision trees in all labeled data fractions. When using the semi-supervised decision trees in an ensemble, however, they appear to be outperformed by their supervised counterparts. The win/tie/loss rates for the random forests suggest that the actual performance differences between the supervised and semi-supervised implementations are small. A possible explanation for this behavior is that the unsupervised loss term acts as a regularizer, somewhat preventing the learner from overfitting. In random forests, this regularization is not necessary due to the regularization provided intrinsically via both the boostrapping and feature subsampling procedures.

As we did for our co-forest experiments, we apply the Wilcoxon signed-rank test to assess the statistical significance of our results. The test shows that the semi-supervised decision trees perform significantly better in all labeled data fractions ($p < 0.05$), whereas there is no significant difference between the semi-supervised random forests and the supervised random forests ($p \geq 0.05$).

We conclude that our approach to incorporate unlabeled data into the splitting criterion is not well-suited to most real-world problems where the objective is to minimize the error rate. In that case, random forests would almost always exhibit superior performance. However, the semi-supervised decision trees are both more cost-effective and more understandable than random forests in the prediction phase. This potentially warrants further research into semi-supervised node splitting criteria.

Table 4.2: Performance comparison between decision trees and random forests with and without semi-supervised node splitting. Includes mean absolute error rates and mean error differences.

| Method | Fraction labeled | | | |
|---|---|---|---|---|
| | 5% | 10% | 15% | 20% |
| **Decision tree** | | | | |
| Supervised | 0.316 | 0.277 | 0.257 | 0.242 |
| Semi-supervised | 0.312 | 0.274 | 0.253 | 0.239 |
| Mean relative change | -0.7% | -0.6% | -1.3% | -1.0% |
| Win / tie / loss | 49/0/28 | 56/0/21 | 58/0/19 | 59/1/17 |
| **Random forest** | | | | |
| Supervised | 0.282 | 0.241 | 0.220 | 0.207 |
| Semi-supervised | 0.284 | 0.242 | 0.221 | 0.208 |
| Mean relative change | 1.6% | 1.2% | 1.3% | 0.8% |
| Win / tie / loss | 35/1/41 | 39/2/36 | 33/1/43 | 34/1/42 |

## 4.5 Conclusions

In this chapter, we have evaluated the performance of the semi-supervised co-forest approach relative to supervised random forests with varying numbers of trees. We have shown that, when using few trees, co-forest significantly outperforms random forests. As the number of trees grows, however, the performance improvement reduces. When using more than 20 trees, supervised random forests significantly outperform co-forests. Our results also show that the performance of larger, supervised forests generally exceeds the performance of smaller co-forests.

Based on these results, we argue that large, supervised random forests should in most cases be preferred over co-forest in most real-world scenarios. Performance improvements are rarely observed when introducing labeled data to forests consisting of many trees, whereas substantial performance degradations do occur. Furthermore, since co-forest typically includes multiple pseudo-labeling steps in which the trees need to be re-trained, the computational resources required for the training phase in a small co-forest can be expected to match the resources required for training a larger random forest. However, when computational resources in the prediction phase are limited (e.g., when using a pre-trained ensemble in mobile devices), co-forest can provide added value over random forests: when using few trees, co-forest generally outperforms random forests.

We see potential merit in applying a single pseudo-labeling iteration to random forests, as we did for general learner ensembles in Chapter 3. This could mitigate some of the issues encountered with co-forests, and provide a computationally efficient way of incorporating unlabeled data into the ensemble.

In addition to evaluating co-forest, we have introduced a generic approach to incorporate unlabeled data into the objective function of decision trees. Estimating the marginal density with kernel density estimation, we directly penalize decision boundaries in high-density regions by incorporating the density estimate in the splitting criterion. Although this approach did significantly improve the performance of individual decision trees, it failed to yield improvements when using the trees in random forests. However, like supervised decision trees, they have the advantage of being more cost-efficient and more understandable in the prediction phase. Potential directions for future research include the evaluation of other unsupervised loss terms and the optimization of the hyperparameter governing the weight of the unsupervised loss term.

Chapter 5

# Conclusions and Future Work

In this thesis, we have studied semi-supervised ensemble methods. We have provided an extensive literature review of the field of semi-supervised learning, covering the major developments in semi-supervised learning over the past two decades. As part of this survey, we have proposed a new taxonomy for the spectrum of semi-supervised classification methods. The taxonomy reflects both the differences in the objectives of different methods and the way these methods incorporate unlabeled data.

The primary contribution of this thesis is the proposal of a new semi-supervised ensembling method, which can be applied to strong ensembles of supervised learners. In a single-step procedure, we exchange confident predictions for unlabeled samples between sub-ensembles of learners. We applied this method to a state-of-the-art automated machine learning system, AUTO-SKLEARN, and showed that our algorithm consistently improves ensemble performance in multiclass classification problems. We evaluated the performance of the approach on the OpenML 100, a prominent benchmarking suite of diverse classification data sets. Under different labeled data fractions, we achieved an error rate reduction of around 7% on average. Performance improvements were obtained in 75% of the multiclass data sets we evaluated.

Our ensembling method provides some interesting opportunities for further research. In particular, we see great potential in the explicit promotion of ensemble diversity during ensemble construction, or even in the hyperparameter optimization phase. Substantial ensemble diversity, the extent to which base classifiers make uncorrelated errors, is a key criterion for co-ensembling to succeed. In the presence of unlabeled data, diversity could be quantified by comparing the predictions of the classifiers on the data points.

In addition to these contributions, we have investigated co-forest, a popular semi-supervised extension of random forests. We have shown that it performs better than its supervised counterpart when few base learners are used, but that it is outperformed by random forests when more trees are used. This substantiates our claim that using unlabeled data to consistently improve over strong supervised learning methods is a difficult task. Lastly, we have proposed a semi-supervised node splitting criterion for decision trees, which provided performance improvements in individual trees, but not in random forests.

In the broader field of semi-supervised classification, the past few years have shown interesting developments. Recent advances in generative models and neural networks have opened up new ways of using unlabeled data to improve learners. Generative semi-supervised models, which approximate the joint data distribution $p(x, y)$, facilitate a principled way of incorporating all available data, both labeled and unlabeled. Furthermore, the versatility of neural networks in regard to the cost function allows for straightforward inclusion of both supervised and unsupervised cost terms. We expect the incorporation of unlabeled data in such classifiers to become commonplace, but note that potential performance degradation is still a pressing issue: if the assumptions underlying the learning algorithm do not hold, the introduction of unlabeled data can be counterproductive. We have addressed this issue in the context of wrapper methods in this thesis, but expect that more advances will have to be made before semi-supervised learning can be broadly and reliably applied in practice.

# Bibliography

[1] S. Abney. Bootstrapping. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 360–367. Association for Computational Linguistics, 2002.

[2] M. R. Anderberg. *Cluster analysis for applications*. Academic Press, 1973.

[3] D. Angluin and P. Laird. Learning from noisy examples. *Machine Learning*, 2(4):343–370, 1988.

[4] A. Azran. The rendezvous algorithm: Multiclass semi-supervised learning with markov random walks. In *Proceedings of the 24th international conference on Machine learning*, pages 49–56, 2007.

[5] P. Bachman, O. Alsharif, and D. Precup. Learning with pseudo-ensembles. In *Advances in neural information processing systems*, pages 3365–3373, 2014.

[6] E. Bair. Semi-supervised clustering methods. *Wiley Interdisciplinary Reviews: Computational Statistics*, 5(5):349–361, 2013.

[7] M.-F. Balcan, A. Blum, and K. Yang. Co-training and expansion: Towards bridging theory and practice. In *Advances in neural information processing systems*, pages 89–96, 2005.

[8] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly. Video suggestion and discovery for youtube: taking random walks through the view graph. In *Proceedings of the 17th international conference on World Wide Web*, pages 895–904. ACM, 2008.

[9] A.-L. Barabási. *Network science*. Cambridge university press, 2016.

[10] S. Basu, A. Banerjee, and R. Mooney. Semi-supervised clustering by seeding. In *Proceedings of the 19th International Conference on Machine Learning*, pages 27–34, 2002.

[11] M. Belkin, I. Matveeva, and P. Niyogi. Regularization and semi-supervised learning on large graphs. In *International Conference on Computational Learning Theory*, pages 624–638. Springer, 2004.

[12] M. Belkin, P. Niyogi, and V. Sindhwani. On manifold regularization. In *Proceedings of the 10th International Conference on Artificial Intelligence and Statistics*, pages 17–24, 2005.

[13] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7:2399–2434, Dec 2006.

[14] S. Ben-David, T. Lu, D. Pál, and M. Sotáková. Learning low density separators. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, pages 25–32, 2009.

[15] Y. Bengio, O. Delalleau, and N. Le Roux. Label propagation and quadratic criterion. In O. Chapelle, B. Schölkopf, and A. Zien, editors, *Semi-supervised Learning*, chapter 11, pages 193–216. The MIT Press, 2006.

[16] K. P. Bennett and A. Demiriz. Semi-supervised support vector machines. In *Advances in neural information processing systems*, pages 368–374, 1999.

[17] K. P. Bennett, A. Demiriz, and R. Maclin. Exploiting unlabeled data in ensemble methods. In *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 289–296. ACM, 2002.

[18] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[19] B. Bischl, G. Casalicchio, M. Feurer, F. Hutter, M. Lang, R. G. Mantovani, J. N. van Rijn, and J. Vanschoren. Openml benchmarking suites and the OpenML100. *ArXiv e-prints*, 2017.

[20] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[21] A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proceedings of the 18th International Conference on Machine Learning*, pages 19–26, 2001.

[22] A. Blum, J. Lafferty, M. R. Rwebangira, and R. Reddy. Semi-supervised learning using randomized mincuts. In *Proceedings of the 21st international conference on Machine learning*, page 13, 2004.

[23] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

[24] A. P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[25] P. Brazdil, C. G. Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

[26] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[27] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen. *Classification and Regression Trees*. Chapman and Hall, 1984.

[28] G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20, 2005.

[29] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *International Conference on Learning Representations*, 2014.

[30] W. Buntine and T. Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8(1):75–85, 1992.

[31] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Proceedings of the 21st International Conference on Machine learning*, page 18. ACM, 2004.

[32] O. Chapelle, M. Chi, and A. Zien. A continuation method for semi-supervised svms. In *Proceedings of the 23rd international conference on Machine learning*, pages 185–192, 2006.

[33] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning*. The MIT Press, 1st edition, 2006.

[34] O. Chapelle, V. Sindhwani, and S. S. Keerthi. Optimization techniques for semi-supervised support vector machines. *Journal of Machine Learning Research*, 9:203–233, Feb 2008.

[35] O. Chapelle and A. Zien. Semi-supervised classification by low density separation. In *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, pages 57–64, 2005.

[36] K. Chen and S. Wang. Semi-supervised learning via regularized boosting working on multiple semi-supervised assumptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):129–143, 2011.

[37] M. Chen, Y. Chen, and K. Q. Weinberger. Automatic feature decomposition for single view co-training. In *Proceedings of the 28th International Conference on Machine Learning*, pages 953–960, 2011.

[38] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[39] C. M. Christoudias, R. Urtasun, A. Kapoorz, and T. Darrell. Co-training with noisy perceptual observations. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2844–2851. IEEE, 2009.

[40] R. Collobert, F. Sinz, J. Weston, and L. Bottou. Large scale transductive svms. *Journal of Machine Learning Research*, 7:1687–1712, Aug 2006.

[41] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, Aug 2011.

[42] A. Corduneanu and T. Jaakkola. On information regularization. In *Proceedings of the 19th conference on Uncertainty in Artificial Intelligence*, pages 151–158. Morgan Kaufmann Publishers Inc., 2003.

[43] C. Cortes and M. Mohri. On transductive regression. In *Advances in neural information processing systems*, pages 305–312, 2007.

[44] F. G. Cozman, I. Cohen, and M. C. Cirelo. Semi-supervised learning of mixture models. In *Proceedings of the 20th International Conference on Machine Learning*, pages 99–106, 2003.

[45] M. Culp and G. Michailidis. An iterative algorithm for extending learners to a semi-supervised setting. *Journal of Computational and Graphical Statistics*, 17(3):545–571, 2008.

[46] F. d'Alche Buc, Y. Grandvalet, and C. Ambroise. Semi-supervised marginboost. *Advances in neural information processing systems*, 1:553–560, 2002.

[47] R. Dara, S. C. Kremer, and D. A. Stacey. Clustering unlabeled data with soms improves classification of labeled real-world data. In *International Joint Conference on Neural Networks*, volume 3, pages 2237–2242. IEEE, 2002.

[48] S. Dasgupta, M. L. Littman, and D. A. McAllester. Pac generalization bounds for co-training. In *Advances in neural information processing systems*, pages 375–382, 2002.

[49] T. de Bie and N. Cristianini. Convex methods for transduction. In *Advances in neural information processing systems*, pages 73–80, 2004.

[50] T. de Bie and N. Cristianini. Semi-supervised learning using semi-definite programming. In O. Chapelle, B. Schölkopf, and A. Zien, editors, *Semi-supervised learning*, pages 119–135. The MIT Press, 2006.

[51] C. A. R. de Sousa, S. O. Rezende, and G. E. Batista. Influence of graph construction on semi-supervised learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 160–175. Springer, 2013.

[52] A. Demiriz, K. P. Bennett, and M. J. Embrechts. Semi-supervised clustering using genetic algorithms. *Artificial neural networks in engineering*, pages 809–814, 1999.

[53] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society, Series B*, pages 1–38, 1977.

[54] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(Jan):1–30, 2006.

[55] C. Deng and M. Zu Guo. A new co-training-style random forest for computer aided diagnosis. *Journal of Intelligent Information Systems*, 36(3):253–281, 2011.

[56] T. G. Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[57] C. Doersch. Tutorial on variational autoencoders. *ArXiv e-prints*, 2016.

[58] I. Dópido, J. Li, P. R. Marpu, A. Plaza, J. M. B. Dias, and J. A. Benediktsson. Semisupervised self-learning for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 51(7):4032–4044, 2013.

[59] J. Du, C. X. Ling, and Z.-H. Zhou. When does cotraining work in real data? *IEEE Transactions on Knowledge and Data Engineering*, 23(5):788–799, 2011.

[60] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[61] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, Feb 2010.

[62] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In *Advances in neural information processing systems*, pages 2962–2970, 2015.

[63] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[64] B. Geng, D. Tao, C. Xu, L. Yang, and X.-S. Hua. Ensemble manifold regularization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(6):1227–1233, 2012.

[65] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[66] A. B. Goldberg, X. Zhu, A. Singh, Z. Xu, and R. D. Nowak. Multi-manifold semi-supervised learning. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, pages 169–176, 2009.

[67] S. Goldman and Y. Zhou. Enhancing supervised learning with unlabeled data. In *Proceedings of the 17th international conference on machine learning*, pages 327–334, 2000.

[68] I. Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *ArXiv e-prints*, 2017.

[69] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.

[70] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[71] H. Grabner, C. Leistner, and H. Bischof. Semi-supervised on-line boosting for robust tracking. *Computer Vision–ECCV 2008*, pages 234–247, 2008.

[72] Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*, pages 529–536, 2005.

[73] Y. Grandvalet, F. d'Alché Buc, and C. Ambroise. Boosting mixture models for semi-supervised learning. *Artificial Neural Networks - ICANN 2001*, pages 41–48, 2001.

[74] N. Grira, M. Crucianu, and N. Boujemaa. Unsupervised and semisupervised clustering: a brief survey. In *7th ACM SIGMM international workshop on Multimedia information retrieval*, 2004.

[75] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[76] I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, T. K. Ho, N. Macia, B. Ray, M. Saeed, A. Statnikov, et al. Design of the 2015 ChaLearn autoML challenge. In *IEEE International Joint Conference on Neural Networks*, pages 1–8. IEEE, 2015.

[77] I. Guyon and A. Elisseeff. An introduction to feature extraction. *Feature extraction*, pages 1–25, 2006.

[78] G. R. Haffari and A. Sarkar. Analysis of semi-supervised learning with the Yarowsky algorithm. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, pages 159–166, 2007.

[79] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[80] J. M. Hammersley and P. Clifford. Markov fields on finite graphs and lattices. 1971.

[81] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[82] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, 12(10):993–1001, 1990.

[83] R. He, W.-S. Zheng, B.-G. Hu, and X.-W. Kong. Nonnegative sparse coding for discriminative semi-supervised learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2849–2856. IEEE, 2011.

[84] M. Hein and M. Maier. Manifold denoising. In *Advances in neural information processing systems*, pages 561–568, 2007.

[85] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[86] H. H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2):70–80, 2012.

[87] B. Huang and T. Jebara. Fast b-matching via sufficient selection belief propagation. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pages 361–369, 2011.

[88] Jayadeva, R. Khemchandani, and S. Chandra. Twin support vector machines for pattern classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(5):905–910, 2007.

[89] T. Jebara, J. Wang, and S.-F. Chang. Graph construction and b-matching for semi-supervised learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 441–448, 2009.

[90] T. Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the 16th international conference on machine learning*, volume 99, pages 200–209, 1999.

[91] T. Joachims. Transductive learning via spectral graph partitioning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 290–297, 2003.

[92] D. Jurafsky and J. H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, 2009.

[93] M. Karasuyama and H. Mamitsuka. Manifold-based similarity adaptation for label propagation. In *Advances in neural information processing systems*, pages 1547–1555, 2013.

[94] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014.

[95] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2013.

[96] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *ArXiv e-prints*, 2016.

[97] S. Kiritchenko and S. Matwin. Email classification with co-training. In *Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research*, page 8. IBM press, 2001.

[98] T. Kohonen. The self-organizing map. *Neurocomputing*, 21(1-3):1–6, 1998.

[99] B. Komer, J. Bergstra, and C. Eliasmith. Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *31st ICML Workshop on AutoML*, 2014.

[100] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. volume 18, pages 826–830. JMLR, 2017.

[101] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[102] B. Kveton, M. Valko, A. Rahimi, and L. Huang. Semi-supervised learning with max-margin graph cuts. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 421–428, 2010.

[103] S. Laine and T. Aila. Temporal ensembling for semi-supervised learning. In *International Conference on Learning Representations*, 2017.

[104] T. Lange, M. H. Law, A. K. Jain, and J. M. Buhmann. Learning with constrained and unlabelled data. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 731–738. IEEE, 2005.

[105] N. D. Lawrence and M. I. Jordan. Semi-supervised learning via gaussian processes. In *Advances in neural information processing systems*, pages 753–760, 2005.

[106] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[107] D.-H. Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *30th ICML Workshop on Challenges in Representation Learning*, volume 3, page 2, 2013.

[108] C. Leistner, A. Saffari, J. Santner, and H. Bischof. Semi-supervised random forests. In *IEEE 12th International Conference on Computer Vision*, pages 506–513. IEEE, 2009.

[109] M. Li. Co-forest java implementation. `http://lamda.nju.edu.cn/code_CoForest.ashx`, 2007.

[110] M. Li and Z.-H. Zhou. Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 37(6):1088–1098, 2007.

[111] S. Li and Y. Fu. Low-rank coding with b-matching constraint for semi-supervised classification. In *Proceedings of the 23rd international joint conference on Artificial Intelligence*, pages 1472–1478, 2013.

[112] S. Li and Y. Fu. Learning balanced and unbalanced graphs via low-rank coding. *IEEE Transactions on Knowledge and Data Engineering*, 27(5):1274–1287, 2015.

[113] Y.-F. Li and Z.-H. Zhou. Towards making unlabeled data never hurt. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(1):175–188, 2015.

[114] G. Liu, Z. Lin, and Y. Yu. Robust subspace segmentation by low-rank representation. In *Proceedings of the 27th international conference on machine learning*, pages 663–670, 2010.

[115] W. Liu and S.-F. Chang. Robust multi-class transductive learning with graphs. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 381–388. IEEE, 2009.

[116] W. Liu, J. He, and S.-F. Chang. Large graph construction for scalable semi-supervised learning. In *Proceedings of the 27th international conference on machine learning*, pages 679–686, 2010.

[117] W. Liu, J. Wang, and S.-F. Chang. Robust and scalable graph-based semisupervised learning. *Proceedings of the IEEE*, 100(9):2624–2638, 2012.

[118] X. Liu, M. Song, D. Tao, Z. Liu, L. Zhang, C. Chen, and J. Bu. Semi-supervised node splitting for random forest construction. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 492–499. IEEE, 2013.

[119] X. Liu, M. Song, D. Tao, Z. Liu, L. Zhang, C. Chen, and J. Bu. Random forest construction with robust semisupervised node splitting. *IEEE Transactions on Image Processing*, 24(1):471–483, 2015.

[120] W.-Y. Loh. Fifty years of classification and regression trees. *International Statistical Review*, 82(3):329–348, 2014.

[121] Q. Lu and L. Getoor. Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning*, pages 496–503, 2003.

[122] M. Maier, U. V. Luxburg, and M. Hein. Influence of graph construction on graph-based clustering measures. In *Advances in neural information processing systems*, pages 1025–1032, 2009.

[123] P. K. Mallapragada, R. Jin, A. K. Jain, and Y. Liu. Semiboost: Boosting for semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(11):2000–2014, 2009.

[124] S. Melacci and M. Belkin. Laplacian support vector machines trained in the primal. *Journal of Machine Learning Research*, 12:1149–1184, Mar 2011.

[125] R. Mihalcea. Co-training and self-training for word sense disambiguation. In *Proceedings of the 8th Conference on Computational Natural Language Learning*, 2004.

[126] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[127] J. Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine learning*, 3(4):319–342, 1989.

[128] J. Neville and D. Jensen. Iterative classification in relational data. In *17th AAAI Workshop on Learning Statistical Models from Relational Data*, pages 13–20, 2000.

[129] K. Nigam and R. Ghani. Analyzing the effectiveness and applicability of co-training. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 86–93. ACM, 2000.

[130] K. Nigam, A. McCallum, and T. Mitchell. Semi-supervised text classification using em. *Semi-Supervised Learning*, pages 33–56, 2006.

[131] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine learning*, 39(2):103–134, 2000.

[132] P. Niyogi. Manifold regularization and semi-supervised learning: Some theoretical analyses. *Journal of Machine Learning Research*, 14(1):1229–1250, 2008.

[133] A. Odena. Semi-supervised learning with generative adversarial networks. *ArXiv e-prints*, 2016.

[134] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas. How many trees in a random forest? In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 154–168. Springer, 2012.

[135] B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*, page 271. Association for Computational Linguistics, 2004.

[136] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[137] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[138] M. Pezeshki, L. Fan, P. Brakel, A. Courville, and Y. Bengio. Deconstructing the ladder network architecture. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 2368–2376, 2016.

[139] N. Pitelis, C. Russell, and L. Agapito. Learning a manifold as an atlas. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1642–1649. IEEE, 2013.

[140] N. Pitelis, C. Russell, and L. Agapito. Semi-supervised learning using an unsupervised atlas. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 565–580. Springer, 2014.

[141] Z. Qi, Y. Tian, and Y. Shi. Laplacian twin support vector machine for semi-supervised classification. *Neural Networks*, 35:46–53, 2012.

[142] L. E. Raileanu and K. Stoffel. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93, 2004.

[143] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko. Semi-supervised learning with ladder networks. In *Advances in neural information processing systems*, pages 3546–3554, 2015.

[144] F. Ratle, G. Camps-Valls, and J. Weston. Semisupervised neural networks for efficient hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 48(5):2271–2282, 2010.

[145] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor. *Recommender systems handbook*. Springer, 2011.

[146] S. Rifai, Y. N. Dauphin, P. Vincent, Y. Bengio, and X. Muller. The manifold tangent classifier. In *Advances in neural information processing systems*, pages 2294–2302, 2011.

[147] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning*, pages 833–840, 2011.

[148] L. Rokach and O. Maimon. Top-down induction of decision trees classifiers-a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 35(4):476–487, 2005.

[149] C. Rosenberg, M. Hebert, and H. Schneiderman. Semi-supervised self-training of object detection models. In *7th IEEE Workshop on Applications of Computer Vision*, pages 29–36, 2005.

[150] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[151] M. Sajjadi, M. Javanmardi, and T. Tasdizen. Regularization with stochastic transformations and perturbations for deep semi-supervised learning. In *Advances in neural information processing systems*, pages 1163–1171, 2016.

[152] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[153] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[154] B. Settles. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114, 2012.

[155] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[156] R. Sheikhpour, M. A. Sarram, S. Gharaghani, and M. A. Z. Chahooki. A survey on semi-supervised feature selection methods. *Pattern Recognition*, 64:141–158, 2017.

[157] N. Shental and E. Domany. Semi-supervised learning–a statistical physics approach. In *22nd ICML workshop on Learning with Partially Classified Training Data*, 2005.

[158] V. Sindhwani, P. Niyogi, and M. Belkin. A co-regularization approach to semi-supervised learning with multiple views. In *22nd ICML Workshop on Learning With Multiple Views*, pages 74–79, 2005.

[159] V. Sindhwani and D. S. Rosenberg. An rkhs for multi-view learning and manifold co-regularization. In *Proceedings of the 25th international conference on Machine learning*, pages 976–983, 2008.

[160] A. Singh, R. Nowak, and X. Zhu. Unlabeled data: Now it helps, now it doesn't. In *Advances in neural information processing systems*, pages 1513–1520, 2009.

[161] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[162] J. Solomon, R. Rustamov, L. Guibas, and A. Butscher. Wasserstein propagation for semi-supervised learning. In *Proceedings of the 31st International Conference on Machine Learning*, pages 306–314, 2014.

[163] J. T. Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *ArXiv e-prints*, 2015.

[164] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[165] A. Subramanya and J. Bilmes. Soft-supervised learning for text classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1090–1099. Association for Computational Linguistics, 2008.

[166] A. Subramanya and J. Bilmes. Semi-supervised learning with measure propagation. *Journal of Machine Learning Research*, 12:3311–3370, Nov 2011.

[167] A. Subramanya and P. P. Talukdar. Graph-based semi-supervised learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(4):1–125, 2014.

[168] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[169] M. Szummer and T. Jaakkola. Partially labeled classification with markov random walks. In *Advances in neural information processing systems*, pages 945–952, 2002.

[170] M. Szummer and T. S. Jaakkola. Information regularization with partially labeled data. In *Advances in neural information processing systems*, pages 1049–1056, 2003.

[171] P. P. Talukdar and K. Crammer. New regularized algorithms for transductive learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 442–457. Springer, 2009.

[172] P. P. Talukdar, J. Reisinger, M. Paşca, D. Ravichandran, R. Bhagat, and F. Pereira. Weakly-supervised acquisition of labeled class instances using graph random walks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 582–590. Association for Computational Linguistics, 2008.

[173] C. Tan, L. Lee, J. Tang, L. Jiang, M. Zhou, and P. Li. User-level sentiment analysis incorporating social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1397–1405. ACM, 2011.

[174] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[175] J. Tanha, M. van Someren, and H. Afsarmanesh. An adaboost algorithm for multiclass semi-supervised learning. In *IEEE 12th International Conference on Data Mining*, pages 1116–1121. IEEE, 2012.

[176] J. Tanha, M. van Someren, and H. Afsarmanesh. Semi-supervised self-training for decision tree classifiers. *International Journal of Machine Learning and Cybernetics*, 8(1):355–370, 2017.

[177] A. Tarvainen and H. Valpola. Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems*, pages 1195–1204, 2017.

[178] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM, 2013.

[179] I. Triguero, S. García, and F. Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems*, 42(2):245–284, 2015.

[180] R. Urner, S. Ben-David, and S. Shalev-Shwartz. Access to unlabeled data can speed up prediction time. In *Proceedings of the 27th international conference on machine learning*, pages 641–648, 2011.

[181] H. Valizadegan, R. Jin, and A. K. Jain. Semi-supervised boosting for multi-class classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 522–537. Springer, 2008.

[182] J. Vanschoren, J. N. Van Rijn, B. Bischl, and L. Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

[183] V. N. Vapnik and V. Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[184] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[185] S. Wager, S. Wang, and P. S. Liang. Dropout training as adaptive regularization. In *Advances in neural information processing systems*, pages 351–359, 2013.

[186] X. Wan. Co-training for cross-lingual sentiment classification. In *Proceedings of the 47th Annual Meeting of the ACL*, pages 235–243. Association for Computational Linguistics, 2009.

[187] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.

[188] F. Wang and C. Zhang. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):55–67, 2008.

[189] J. Wang, T. Jebara, and S.-F. Chang. Graph transduction via alternating minimization. In *Proceedings of the 25th international conference on Machine learning*, pages 1144–1151, 2008.

[190] J. Wang, T. Jebara, and S.-F. Chang. Semi-supervised learning using greedy max-cut. *Journal of Machine Learning Research*, 14:771–800, Mar 2013.

[191] J. Wang, S.-w. Luo, and X.-h. Zeng. A random subspace method for co-training. In *IEEE International Joint Conference on Neural Networks*, pages 195–200. IEEE, 2008.

[192] W. Wang and Z.-H. Zhou. Analyzing co-training style algorithms. In *Proceedings of the 18th European Conference on Machine Learning*, pages 454–465. Springer, 2007.

[193] W. Wang and Z.-H. Zhou. A new analysis of co-training. In *Proceedings of the 27th international conference on machine learning*, pages 1135–1142, 2010.

[194] J. Weston, F. Ratle, and R. Collobert. Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1168–1175, 2008.

[195] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

[196] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[197] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):210–227, 2009.

[198] X.-M. Wu, Z. Li, A. M. So, J. Wright, and S.-F. Chang. Learning with partially absorbing random walks. In *Advances in neural information processing systems*, pages 3077–3085, 2012.

[199] Z. Wu, J. Wu, J. Cao, and D. Tao. Hysad: A semi-supervised hybrid shilling attack detector for trustworthy product recommendation. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 985–993. ACM, 2012.

[200] C. Xu, D. Tao, and C. Xu. A survey on multi-view learning. *ArXiv e-prints*, 2013.

[201] J. Xu, H. He, and H. Man. Dcpe co-training for classification. *Neurocomputing*, 86:75–85, 2012.

[202] L. Xu and D. Schuurmans. Unsupervised and semi-supervised multi-class support vector machines. In *Proceedings of the 20th National Conference on Artificial Intelligence*, volume 5, page 13, 2005.

[203] S. Yan and H. Wang. Semi-supervised learning by sparse representation. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 792–801. SIAM, 2009.

[204] Z. Yang, W. W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 40–48, 2016.

[205] D. Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 189–196. Association for Computational Linguistics, 1995.

[206] Y. Yaslan and Z. Cataltepe. Co-training with relevant random subspaces. *Neurocomputing*, 73(10):1652–1661, 2010.

[207] S. Yu, B. Krishnapuram, R. Rosales, and R. B. Rao. Bayesian co-training. *Journal of Machine Learning Research*, 12:2649–2680, Sep 2011.

[208] K. Zhang, J. T. Kwok, and B. Parvin. Prototype vector machine for large scale semi-supervised learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 1233–1240, 2009.

[209] W. Zhang and Q. Zheng. Tsfs: A novel algorithm for single view co-training. In *IEEE International Joint Conference on Computational Sciences and Optimization*, volume 1, pages 492–496. IEEE, 2009.

[210] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in neural information processing systems*, pages 321–328, 2004.

[211] Y. Zhou and S. Goldman. Democratic co-learning. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 594–602. IEEE, 2004.

[212] Z.-H. Zhou. *Ensemble methods: foundations and algorithms*. CRC press, 2012.

[213] Z.-H. Zhou and M. Li. Semi-supervised regression with co-training. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, volume 5, pages 908–913, 2005.

[214] Z.-H. Zhou and M. Li. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1529–1541, 2005.

[215] Z.-H. Zhou and M. Li. Semi-supervised learning by disagreement. *Knowledge and Information Systems*, 24(3):415–439, 2010.

[216] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, University of Wisconsin-Madison, 2005.

[217] X. Zhu. *Semi-supervised learning with graphs*. PhD thesis, Carnegie Mellon University, 2005.

[218] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical Report CMU-CALD-02-107, Carnegie Mellon University, 2002.

[219] X. Zhu and Z. Ghahramani. Towards semi-supervised classification with markov random fields. Technical Report CMU-CALD-02-106, Carnegie Mellon University, 2002.

[220] X. Zhu, Z. Ghahramani, and J. D. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning*, pages 912–919, 2003.

[221] X. Zhu and A. B. Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.

[222] X. Zhu and J. Lafferty. Harmonic mixtures: combining mixture models and graph-based methods for inductive and scalable semi-supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 1052–1059. ACM, 2005.

[223] L. Zhuang, H. Gao, Z. Lin, Y. Ma, X. Zhang, and N. Yu. Non-negative low rank and sparse graph for semi-supervised learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2328–2335. IEEE, 2012.

# Appendix A

# Co-ensembling Results Per Data Set

In Table A.1, the per-data-set results of the co-ensembling experiments (Chapter 3) are displayed. The results in every table cell are aggregated over the individual experiments for that data set and labeled data fraction.

Table A.1: Co-ensembling results per data set. The column "ASKL (unw.)" contains the performance of the unweighted ensemble from AUTO-SKLEARN. The column "+CE" contains the performance of the ensemble obtained when applying single-step co-ensembling to the unweighted ensemble mentioned beore.

| Data set | 10% labeled | | | 20% labeled | | |
|---|---|---|---|---|---|---|
| | ASKL (unw.) | +CE | Diff | ASKL (unw.) | +CE | Diff |
| **Multiclass** | | | | | | |
| anacat-dmft | 0.819 | 0.814 | -0.6% | 0.826 | 0.818 | -1.0% |
| artificial-characters | 0.378 | 0.380 | 0.6% | 0.260 | 0.269 | 3.4% |
| balance-scale | 0.155 | 0.124 | -19.5% | 0.110 | 0.108 | -1.6% |
| car | 0.092 | 0.086 | -6.7% | 0.062 | 0.056 | -10.4% |
| cjs | 0.023 | 0.025 | 51.9% | 0.012 | 0.006 | -51.9% |
| cmc | 0.505 | 0.494 | -2.1% | 0.470 | 0.459 | -2.3% |
| cnae-9 | 0.146 | 0.119 | -18.0% | 0.084 | 0.076 | -9.5% |
| eucalyptus | 0.475 | 0.464 | -2.4% | 0.401 | 0.394 | -1.8% |
| gas-drift | 0.012 | 0.011 | -9.4% | 0.007 | 0.007 | -1.7% |
| gesture-phase | 0.440 | 0.455 | 3.4% | 0.411 | 0.431 | 4.7% |
| har | 0.035 | 0.034 | -4.7% | 0.022 | 0.022 | -4.0% |
| isolet | 0.073 | 0.066 | -10.0% | 0.051 | 0.044 | -12.0% |
| japanese-vowels | 0.043 | 0.043 | -0.7% | 0.026 | 0.025 | -6.0% |
| led-display-domain | 0.352 | 0.343 | -2.5% | 0.300 | 0.285 | -4.9% |

Table continues on next page

Table A.1: Co-ensembling results per data set. The column "ASKL (unw.)" contains the performance of the unweighted ensemble from AUTO-SKLEARN. The column "+CE" contains the performance of the ensemble obtained when applying single-step co-ensembling to the unweighted ensemble mentioned beore.

| Data set | 10% labeled | | | 20% labeled | | |
|---|---|---|---|---|---|---|
| | ASKL (unw.) | +CE | Diff | ASKL (unw.) | +CE | Diff |
| letter | 0.111 | 0.107 | -5.1% | 0.064 | 0.059 | -7.5% |
| mfeat-factors | 0.068 | 0.051 | -25.9% | 0.044 | 0.038 | -12.7% |
| mfeat-fourier | 0.199 | 0.193 | -2.8% | 0.168 | 0.162 | -3.9% |
| mfeat-karhunen | 0.064 | 0.050 | -21.5% | 0.038 | 0.033 | -13.6% |
| mfeat-morph | 0.307 | 0.306 | -0.0% | 0.281 | 0.281 | 0.1% |
| mfeat-pixel | 0.091 | 0.064 | -24.7% | 0.038 | 0.032 | -14.1% |
| mfeat-zernike | 0.209 | 0.201 | -3.5% | 0.198 | 0.187 | -5.5% |
| optdigits | 0.028 | 0.024 | -16.8% | 0.018 | 0.015 | -15.4% |
| pendigits | 0.014 | 0.012 | -13.0% | 0.008 | 0.006 | -15.3% |
| robot-navigation | 0.019 | 0.019 | -0.6% | 0.011 | 0.010 | -3.0% |
| satimage | 0.124 | 0.125 | 1.2% | 0.107 | 0.109 | 2.1% |
| segment | 0.057 | 0.054 | -4.8% | 0.042 | 0.041 | -1.4% |
| semeion | 0.168 | 0.149 | -12.2% | 0.094 | 0.085 | -9.1% |
| splice | 0.063 | 0.063 | -0.1% | 0.051 | 0.052 | 2.2% |
| synthetic-control | 0.043 | 0.024 | -43.5% | 0.028 | 0.023 | -11.8% |
| texture | 0.010 | 0.006 | -34.9% | 0.006 | 0.004 | -30.5% |
| theorem-proving | 0.493 | 0.502 | 1.8% | 0.459 | 0.466 | 1.7% |
| vehicle | 0.306 | 0.305 | -0.3% | 0.237 | 0.241 | 1.9% |
| vowel | 0.473 | 0.418 | -9.4% | 0.228 | 0.221 | -3.2% |
| waveform-5000 | 0.154 | 0.164 | 6.2% | 0.139 | 0.142 | 1.6% |
| | | | | | | |
| **Binary** | | | | | | |
| ada-agnostic | 0.175 | 0.182 | 3.7% | 0.161 | 0.159 | -1.3% |
| adult | 0.139 | 0.142 | 1.8% | 0.137 | 0.142 | 3.5% |
| australian | 0.180 | 0.207 | 13.3% | 0.167 | 0.162 | -1.8% |
| bank-marketing | 0.102 | 0.103 | 1.7% | 0.099 | 0.101 | 2.4% |
| blood-donors | 0.280 | 0.281 | 0.1% | 0.240 | 0.246 | 2.5% |
| climate-model | 0.096 | 0.095 | -1.5% | 0.089 | 0.085 | -4.2% |
| credit-approval | 0.192 | 0.193 | -1.8% | 0.156 | 0.150 | -3.6% |
| credit-g | 0.289 | 0.284 | -1.8% | 0.272 | 0.269 | -1.3% |
| cylinder-bands | 0.402 | 0.435 | 8.5% | 0.339 | 0.359 | 6.2% |
| diabetes | 0.295 | 0.290 | -1.3% | 0.279 | 0.275 | -1.2% |
| dresses-sales | 0.459 | 0.443 | -3.2% | 0.462 | 0.465 | 0.7% |
| eeg-eye-state | 0.071 | 0.065 | -9.2% | 0.057 | 0.057 | 0.3% |
| electricity | 0.143 | 0.147 | 2.8% | 0.111 | 0.117 | 5.3% |
| higgs | 0.291 | 0.300 | 3.1% | 0.288 | 0.301 | 4.5% |

Table continues on next page

Table A.1: Co-ensembling results per data set. The column "ASKL (unw.)" contains the performance of the unweighted ensemble from AUTO-SKLEARN. The column "+CE" contains the performance of the ensemble obtained when applying single-step co-ensembling to the unweighted ensemble mentioned beore.

| Data set | 10% labeled | | | 20% labeled | | |
|---|---|---|---|---|---|---|
| | ASKL (unw.) | +CE | Diff | ASKL (unw.) | +CE | Diff |
| hill-valley | 0.040 | 0.034 | -47.9% | 0.022 | 0.017 | -46.7% |
| ilpd | 0.328 | 0.333 | 1.8% | 0.304 | 0.312 | 3.0% |
| jm1 | 0.192 | 0.193 | 0.3% | 0.188 | 0.202 | 7.2% |
| kc1 | 0.162 | 0.162 | -0.2% | 0.156 | 0.152 | -2.0% |
| kc2 | 0.145 | 0.166 | 14.7% | 0.179 | 0.178 | -0.1% |
| kddcup09-churn | 0.075 | 0.075 | 0.0% | 0.074 | 0.074 | 0.0% |
| kddcup09-upselling | 0.050 | 0.054 | 7.6% | 0.050 | 0.050 | -0.2% |
| kr-vs-kp | 0.036 | 0.037 | 2.4% | 0.017 | 0.018 | 2.4% |
| madelon | 0.292 | 0.290 | -0.3% | 0.213 | 0.221 | 3.5% |
| magic-telescope | 0.136 | 0.137 | 1.1% | 0.129 | 0.130 | 1.3% |
| monks-2 | 0.219 | 0.242 | 13.1% | 0.072 | 0.076 | 20.4% |
| monks-3 | 0.092 | 0.086 | -8.9% | 0.042 | 0.039 | 2.9% |
| mozilla4 | 0.061 | 0.061 | 0.4% | 0.054 | 0.054 | 0.9% |
| nomao | 0.043 | 0.044 | 1.9% | 0.038 | 0.041 | 5.8% |
| ozone-level-8hr | 0.077 | 0.074 | -4.5% | 0.062 | 0.063 | 1.7% |
| pc1 | 0.074 | 0.072 | -2.6% | 0.078 | 0.077 | -1.0% |
| pc3 | 0.167 | 0.150 | -3.3% | 0.122 | 0.119 | -2.3% |
| pc4 | 0.118 | 0.117 | -0.8% | 0.102 | 0.102 | -0.1% |
| phoneme | 0.177 | 0.180 | 1.9% | 0.135 | 0.140 | 3.4% |
| profb | 0.376 | 0.361 | -3.9% | 0.360 | 0.357 | -0.5% |
| qsar-biodeg | 0.192 | 0.205 | 6.1% | 0.164 | 0.167 | 1.5% |
| scene | 0.021 | 0.015 | -28.9% | 0.015 | 0.012 | -11.1% |
| sick | 0.036 | 0.035 | -2.3% | 0.022 | 0.022 | 0.6% |
| spambase | 0.069 | 0.069 | 1.1% | 0.057 | 0.057 | 0.2% |
| speed-dating | 0.150 | 0.154 | 2.4% | 0.141 | 0.142 | 0.7% |
| sylva-agnostic | 0.008 | 0.008 | -6.1% | 0.007 | 0.007 | 1.8% |
| tic-tac-toe | 0.052 | 0.057 | 8.9% | 0.052 | 0.029 | -45.0% |
| wilt | 0.016 | 0.017 | 6.3% | 0.018 | 0.017 | 0.1% |

# Appendix B

# Self-training in auto-sklearn

As part of our preliminary experiments for co-ensembling (Chapter 3), we integrated the self-training wrapper method into the configuration space of AUTO-SKLEARN. A versatile self-training framework was implemented, exposing a variety of hyperparameters. These hyperparameters, which are similar to the hyperparameters in the *co-ensembling* algorithm, are listed in Table B.1.

The experimental sutup was somewhat dissimilar from the experimental setup used in the main experiments. In particular, we used 100 labeled data points per class for each data set, instead of 10% or 20% of the full data set. Furthermore, we did not conduct fully paired tests between the supervised and semi-supervised setup: data splitting into labeled/unlabeled sets was independent for each experiment. Like in our main experiments, we used 8 nodes per experiment. Each node was provided with one hour of computational budget (wall-clock time) and three minutes per individual classifier training run. Our preliminary experiments are conducted on a subset of the OpenML 100 benchmarking suite (see Appendix C) corresponding to the first half of the data sets obtained via the OpenML Python API [182]. These, along with the per-data-set results of the self-training experiments, are listed in Table B.3.

The aggregated results of our experiments are listed in Table B.2. The results show that the incorporation of self-training into the configuration space degrades perfor-

Table B.1: Hyperparameters for the self-training framework.

| Hyperparameter | Description |
|---|---|
| *num_iter* | Number of self-training iterations |
| $\theta$ | Minimum predictive confidence of classifier to pseudo-label sample |
| *max_pl* | Maximum number of samples to pseudo-label per iteration |
| *retain_pl* | Whether to retain previously pseudo-labeled samples in later iterations |

Table B.2: Performance comparison between AUTO-SKLEARN with and without self-training included in the design space. Includes mean absolute error rates and mean error differences.

| Hyperparameter | Description |
|---|---|
| **Overall** | |
| Supervised | 0.162 |
| Semi-supervised | 0.162 |
| Mean relative change | 4.9% |
| Win/tie/loss | 27/0/24 |
| **Binary** | |
| Supervised | 0.163 |
| Semi-supervised | 0.162 |
| Mean relative change | 1.0% |
| Win/tie/loss | 18/0/15 |
| **Multiclass** | |
| Supervised | 0.161 |
| Semi-supervised | 0.161 |
| Mean relative change | 12.1% |
| Win/tie/loss | 9/0/9 |

mance on average. Considering the win/tie/loss rates, both methods win in approximately the same number of cases. Applying the Wilcoxon signed-rank test to the per-data-set results [195], the performance difference between the supervised and semi-supervised AUTO-SKLEARN implementations is not statistically significant ($p \geq 0.05$). The similar win/tie/loss rates, combined with the substantial average performance degradation, indicates that self-training does not lead to better-performing configurations, and that it can prevent the optimization procedure from finding interesting configurations within the allotted time. The latter is likely caused by the computational burden imposed by the self-training procedure.

We note that the large variance in the per-data-set results can be explained by the lack of paired tests: the small amount of labeled data in each experiment causes different experiments to yield highly varying results. As such, we mainly reason about the performance comparison from the perspective of the aggregated results.

Table B.3: AUTO-SKLEARN self-training results per data set. Comparison of AUTO-SKLEARN with ("Supervised" column) and without ("Semi-supervised" column) self-training included in the design space. Includes mean absolute error rates and mean error differences.

| Data set | Supervised | Semi-supervised | Difference |
|---|---:|---:|---:|
| adult | 0.188 | 0.185 | -1.6% |
| artificial-characters | 0.426 | 0.433 | 1.6% |
| australian | 0.149 | 0.155 | 3.9% |
| bank-marketing | 0.123 | 0.119 | -3.1% |
| banknotes | 0.023 | 0.020 | -12.8% |
| bioresponse | 0.284 | 0.319 | 12.3% |
| blood-donors | 0.245 | 0.239 | -2.4% |
| cardiotocography | 0.000 | 0.001 | 61.9% |
| climate-model | 0.104 | 0.106 | 2.5% |
| cnae-9 | 0.056 | 0.057 | 2.6% |
| cylinder-bands | 0.292 | 0.287 | -1.9% |
| dresses-sales | 0.441 | 0.443 | 0.5% |
| eeg-eye-state | 0.301 | 0.319 | 6.2% |
| gas-drift | 0.030 | 0.035 | 16.7% |
| gesture-phase | 0.509 | 0.512 | 0.6% |
| gina-agnostic | 0.220 | 0.185 | -15.8% |
| har | 0.064 | 0.055 | -14.2% |
| hill-valley | 0.036 | 0.024 | -33.8% |
| ilpd | 0.309 | 0.294 | -5.1% |
| internet-ads | 0.058 | 0.067 | 15.5% |
| jm1 | 0.167 | 0.212 | 27.3% |
| kc1 | 0.166 | 0.161 | -2.5% |
| kc2 | 0.189 | 0.169 | -10.8% |
| kddcup09-churn | 0.086 | 0.097 | 12.7% |
| kddcup09-upselling | 0.078 | 0.081 | 5.1% |
| led-display-domain | 0.264 | 0.263 | -0.4% |
| madelon | 0.433 | 0.423 | -2.4% |
| magic-telescope | 0.200 | 0.191 | -4.3% |
| mice-protein | 0.003 | 0.001 | -84.3% |
| micro-mass | 0.098 | 0.093 | -4.7% |
| mnist-784 | 0.084 | 0.101 | 19.9% |
| mozilla4 | 0.109 | 0.078 | -28.3% |
| nomao | 0.099 | 0.089 | -10.2% |
| ozone-level-8hr | 0.073 | 0.082 | 13.3% |
| pc1 | 0.086 | 0.087 | 1.8% |
| pc3 | 0.131 | 0.127 | -2.8% |
| pc4 | 0.100 | 0.129 | 29.3% |
| phoneme | 0.217 | 0.221 | 1.7% |

Table continues on next page

Table B.3: AUTO-SKLEARN self-training results per data set. Comparison of AUTO-SKLEARN with ("Supervised" column) and without ("Semi-supervised" column) self-training included in the design space. Includes mean absolute error rates and mean error differences.

| Data set | Supervised | Semi-supervised | Difference |
|---|---:|---:|---:|
| plants-margin | 0.171 | 0.166 | -3.0% |
| plants-shape | 0.341 | 0.342 | 0.5% |
| plants-texture | 0.175 | 0.183 | 4.6% |
| qsar-biodeg | 0.192 | 0.184 | -3.9% |
| robot-navigation | 0.048 | 0.047 | -2.0% |
| semeion | 0.073 | 0.069 | -4.9% |
| speed-dating | 0.188 | 0.173 | -8.3% |
| steel-plates | 0.006 | 0.009 | 52.4% |
| tamilnadu | 0.000 | 0.000 | 253.6% |
| texture | 0.011 | 0.008 | -29.2% |
| theorem-proving | 0.538 | 0.527 | -1.9% |
| wdbc | 0.053 | 0.044 | -17.4% |
| wilt | 0.036 | 0.041 | 15.6% |

# Appendix C

# Data Sets

All methods we evaluate and propose are intended to be generic, robust machine learning methods, applicable to a broad variety of data sets. In that light, the algorithms should be evaluated on a large number of data sets with diverse characteristics. The OpenML 100 is a benchmarking suite that satisfies these requirements [19]. The benchmarking suite consists of 100 data sets and is publicly accessible via the OpenML platform [182]. We note that the benchmarking suite is somewhat subject to change; at the time of our experiments, three data sets had been removed from the benchmarking suite, presumably due to some inconsistencies in these data sets.

An overview of the data sets used in our experiments is provided in Table C.1. It includes some data set properties, including the number of samples and the number of classes present in the data set.

Table C.1: Data sets used in our experiments. The "ID" column contains the ID of the data set on the OpenML platform. The "Missing values" column indicates whether any feature values are missing in the data sets, which is particularly relevant for the random forest experiments.

| ID | Name | # Samples | # Classes | # Features | Missing values |
|---|---|---|---|---|---|
| 1043 | ada-agnostic | 4562 | 2 | 48 | |
| 1590 | adult | 48842 | 2 | 14 | Yes |
| 458 | anacat-authors | 841 | 4 | 70 | |
| 469 | anacat-dmft | 797 | 6 | 4 | |
| 1459 | artificial-characters | 10218 | 10 | 7 | |
| 40981 | australian | 690 | 2 | 14 | |
| 11 | balance-scale | 625 | 3 | 4 | |
| 1461 | bank-marketing | 45211 | 2 | 16 | |
| 1462 | banknotes | 1372 | 2 | 4 | |
| 4134 | bioresponse | 3751 | 2 | 1776 | |
| 1464 | blood-donors | 748 | 2 | 4 | |
| 15 | breast-w | 699 | 2 | 9 | Yes |
| 21 | car | 1728 | 4 | 6 | |
| 1466 | cardiotocography | 2126 | 10 | 35 | |
| 23380 | cjs | 2796 | 6 | 33 | Yes |

Table continues on next page

Table C.1: Data sets used in our experiments. The "ID" column contains the ID of the data set on the OpenML platform. The "Missing values" column indicates whether any feature values are missing in the data sets, which is particularly relevant for the random forest experiments.

| ID | Name | # Samples | # Classes | # Features | Missing values |
|---|---|---|---|---|---|
| 1467 | climate-model | 540 | 2 | 20 | |
| 23 | cmc | 1473 | 3 | 9 | |
| 1468 | cnae-9 | 1080 | 9 | 856 | |
| 478 | collins | 500 | 15 | 21 | |
| 29 | credit-approval | 690 | 2 | 15 | Yes |
| 31 | credit-g | 1000 | 2 | 20 | |
| 6332 | cylinder-bands | 540 | 2 | 37 | Yes |
| 37 | diabetes | 768 | 2 | 8 | |
| 23381 | dresses-sales | 500 | 2 | 12 | Yes |
| 1471 | eeg-eye-state | 14980 | 2 | 14 | |
| 151 | electricity | 45312 | 2 | 8 | |
| 188 | eucalyptus | 736 | 5 | 19 | Yes |
| 1476 | gas-drift | 13910 | 6 | 128 | |
| 4538 | gesture-phase | 9873 | 5 | 32 | |
| 1038 | gina-agnostic | 3468 | 2 | 970 | |
| 1478 | har | 10299 | 6 | 561 | |
| 23512 | higgs | 98050 | 2 | 28 | Yes |
| 1479 | hill-valley | 1212 | 2 | 100 | |
| 1480 | ilpd | 583 | 2 | 10 | |
| 1176 | internet-ads | 3279 | 2 | 1558 | |
| 451 | irish | 500 | 2 | 5 | Yes |
| 300 | isolet | 7797 | 26 | 617 | |
| 375 | japanese-vowels | 9961 | 9 | 14 | |
| 1053 | jm1 | 10885 | 2 | 21 | Yes |
| 1067 | kc1 | 2109 | 2 | 21 | |
| 1063 | kc2 | 522 | 2 | 21 | |
| 1112 | kddcup09-churn | 50000 | 2 | 230 | Yes |
| 1114 | kddcup09-upselling | 50000 | 2 | 230 | Yes |
| 3 | kr-vs-kp | 3196 | 2 | 36 | |
| 40496 | led-display-domain | 500 | 10 | 7 | |
| 6 | letter | 20000 | 26 | 16 | |
| 1485 | madelon | 2600 | 2 | 500 | |
| 1120 | magic-telescope | 19020 | 2 | 10 | |
| 12 | mfeat-factors | 2000 | 10 | 216 | |
| 14 | mfeat-fourier | 2000 | 10 | 76 | |
| 16 | mfeat-karhunen | 2000 | 10 | 64 | |
| 18 | mfeat-morph | 2000 | 10 | 6 | |
| 20 | mfeat-pixel | 2000 | 10 | 240 | |
| 22 | mfeat-zernike | 2000 | 10 | 47 | |
| 4550 | mice-protein | 1080 | 8 | 81 | Yes |
| 1515 | micro-mass | 571 | 20 | 1300 | |
| 554 | mnist-784 | 70000 | 10 | 784 | |
| 333 | monks-1 | 556 | 2 | 6 | |
| 334 | monks-2 | 601 | 2 | 6 | |
| 335 | monks-3 | 554 | 2 | 6 | |
| 1046 | mozilla4 | 15545 | 2 | 5 | |
| 24 | mushroom | 8124 | 2 | 22 | Yes |
| 1486 | nomao | 34465 | 2 | 118 | |
| 28 | optdigits | 5620 | 10 | 64 | |

Table continues on next page

Table C.1: Data sets used in our experiments. The "ID" column contains the ID of the data set on the OpenML platform. The "Missing values" column indicates whether any feature values are missing in the data sets, which is particularly relevant for the random forest experiments.

| ID | Name | # Samples | # Classes | # Features | Missing values |
|---|---|---|---|---|---|
| 1487 | ozone-level-8hr | 2534 | 2 | 72 | |
| 1068 | pc1 | 1109 | 2 | 21 | |
| 1050 | pc3 | 1563 | 2 | 37 | |
| 1049 | pc4 | 1458 | 2 | 37 | |
| 32 | pendigits | 10992 | 10 | 16 | |
| 1489 | phoneme | 5404 | 2 | 5 | |
| 1491 | plants-margin | 1600 | 100 | 64 | |
| 1492 | plants-shape | 1600 | 100 | 64 | |
| 1493 | plants-texture | 1599 | 100 | 64 | |
| 470 | profb | 672 | 2 | 9 | Yes |
| 1494 | qsar-biodeg | 1055 | 2 | 41 | |
| 1497 | robot-navigation | 5456 | 4 | 24 | |
| 182 | satimage | 6430 | 6 | 36 | |
| 312 | scene | 2407 | 2 | 299 | |
| 36 | segment | 2310 | 7 | 19 | |
| 1501 | semeion | 1593 | 10 | 256 | |
| 38 | sick | 3772 | 2 | 29 | Yes |
| 42 | soybean | 683 | 19 | 35 | Yes |
| 44 | spambase | 4601 | 2 | 57 | |
| 40536 | speed-dating | 8378 | 2 | 120 | Yes |
| 46 | splice | 3190 | 3 | 60 | |
| 1504 | steel-plates | 1941 | 2 | 33 | |
| 1036 | sylva-agnostic | 14395 | 2 | 216 | |
| 377 | synthetic-control | 600 | 6 | 60 | |
| 1505 | tamilnadu | 45781 | 20 | 3 | |
| 40499 | texture | 5500 | 11 | 40 | |
| 1475 | theorem-proving | 6118 | 6 | 51 | |
| 50 | tic-tac-toe | 958 | 2 | 9 | |
| 54 | vehicle | 846 | 4 | 18 | |
| 307 | vowel | 990 | 11 | 12 | |
| 60 | waveform-5000 | 5000 | 3 | 40 | |
| 1510 | wdbc | 569 | 2 | 30 | |
| 1570 | wilt | 4839 | 2 | 5 | |