I thank Prof. Marco Saerens and Bertrand Lebichot for the opportunity to work on the active field of research which is graph-based semi-supervised classification and for the idea to extend the hybrid meta-algorithm.
I thank Julian Rox and Valentin Dendoncker for helping me when I was stuck.
I thank my fiancée Cécile Picard for supporting me the whole time.

Contents

Chapter 1

Introduction

Our research question can be formulated as follows:

> Can graph-based semi-supervised classification algorithms be extended to out-of-sample prediction? How can the recently proposed adaptive edge weighting (AEW) technique help with this endeavor?

Before we define the technical terms contained in this question, let's start with some general context. We then introduce the solutions proposed to tackle the research question and highlight how they extend the existing body of knowledge. The third section about managerial implications should motivate the relevance of our research question and clarify the theoretical concepts with practical examples. We end this introduction with an outlook on the following chapters.

1.1 General context of this work

Classification is a type of machine learning. A machine is given a series of training examples where it knows the input data and the associated output: a discrete label which puts the example in a class. Based on those training examples, the machine finds out how to best distinguish the different classes. The machine learns how to predict the labels of other examples than the training set.

If all the training examples come with an output label, we speak of supervised learning. Supervised machine learning algorithms are a mature field of research. A plethora of algorithms and variants thereof is available (Sammut & Webb, 2011).

In the last two years, machine learning has gained increasing public attention. For the better or the worse, artificial intelligence has become a hot topic, often discussed in conjunction with the buzzword *big data* (Stone et al., 2016; LeCun, 2016; Beer, 2016; Ashbrook, Veloso, & Dietterich, 2015; Harris, 2016). Most authors would agree that classification is an important, but not the only, manifestation of artificial intelligence. Will big data, provided we can agree on what the term means, be a game-changer for classification?

Big data can mean potentially two things in the context of classification: There will be more training examples to learn from and/or each training example will be described in more detail (Kuhn, 2015). Not all classification algorithms are well-suited for the future abundance of training examples.

One issue is that not all training examples come with an output label. Such labels may require the judgment of a human expert in which case they are expensive to come by. In other cases, labels

might be available only in delayed fashion. For example, we can know only after the facts which small businesses survived and which defaulted on their loans. In conclusion, unlabeled examples are already more common than labeled ones and this trend will only strengthen in the future.

Semi-supervised classification algorithms are designed in such ways that they can make use of labeled and unlabeled examples. Hence the name semi-supervised.

1.2 Proposed solution

The graph-based semi-supervised algorithms we study in this master thesis are based on assumptions that should make them thrive with big data in form of abundant training examples. Training examples (with and without labels) are transformed into a graph where each node represents an example and is connected to only a few other nodes. Graph based learning is a three-step process:

1. Connect each node to only a few neighboring nodes
2. Compute weights for those connections (edges)
3. Propagate labels over the constructed graph until each node has one

As the name indicates, the newly proposed adaptive edge weighting technique solves the second step of the procedure (Karasuyama & Mamitsuka, 2013a, 2016). The aforementioned three steps are modular, AEW can be combined with different algorithms for the first- and for the last phase.

We do *not* perform a replication of the experiments that established AEW's impressive performance by comparing it to other edge weighting techniques and combining it with different label propagation methods. Other authors have already used AEW in their studies where it has not failed to replicate (Pei, Lyu, Chen, & Chen, 2015; Gong et al., 2016; Fu, Gu, Gong, & Yang, 2016). In our experiments, all graph-based algorithms use AEW.

We are interested in another property of AEW: parsimonious representation of edge weights. AEW optimizes edge weights over only a few decision variables. The resulting optimized parameters are combined with similarity measures of the connected nodes to obtain edge weights. New nodes might thus be connected to the graph. Weights can be estimated based on the AEW parameters.

Such an out-of-sample prediction is usually not possible with graph-based learning. Only nodes that have been on the graph from step 1 onwards can be predicted. If new examples arrive, the whole algorithm needs to be launched again. Most supervised algorithms do not have this limitation. Unless there is an out-of-sample extension for semi-supervised algorithms, the user has to choose between being able to use unlabeled data and being able to quickly predict unseen data.

AEW might be one way to solve this dilemma. Govada, Joshi, Mittal, and Sahay (2015) provide another potential solution. Graph-based algorithms can be combined with other algorithms to yield hybrid meta-algorithms. Each of these algorithms has its own strengths and weaknesses. By combining the algorithms, we hope that the strengths add up and the weaknesses cancel out. More concretely, the graph-based algorithm helps to better use unlabeled data and the other algorithm(s) help to better predict unseen data.

The following examples of real world business cases should help clarify why those extensions to the current body of knowledge are of interest.

1.3 Managerial implications

Let's consider the following example: Assume we manage a large chain of gyms. Clients who want to work out in those gyms can pay for either basic- or premium memberships. The premium membership is much more expensive and includes personal training. Only 3% of members are currently premium members. We think that this number could grow to 10%, conceding that 90% would never want to be premium members. In order to promote the premium membership, we offer one free month of upgrade to premium so that basic members get used to the personal training and continue using it after the trial period.

It would be too expensive to offer the free trial upgrade to everyone. Since the trial would be wasted on 90% of customers, we need a classification algorithm that can identify the most plausible candidates for premium membership. In terms of historic data, we have records of clients that did stay premium members (those current 3%) and clients that reverted to normal membership (another 5%). Those two outcomes are the output labels that help the classifier distinguish good from bad candidates. In supervised learning, we would learn estimated labels of the remaining clients based on those 8%. In semi-supervised learning, we can also use the client records of the remaining 92%. Those numerous cases where the algorithm does not know the outcome are unlabeled data. Semi-supervised learning presumes that unlabeled data helps classification.

The graph-based methods we focus on are traditionally limited to predicting only those unlabeled records that are on the graph at time of construction. This would be an issue for us as gym managers. We know that it is much easier to sell premium memberships at the first contact with a new client rather than to sell upgrades later. Such new customers would not be on an earlier constructed graph. Even if we know the same data for the new records as for the old, we cannot simply add them to the graph (we will discuss later why that is) and therefore not classify them.

We could recompute the whole graph every time a new customer arrives. This is not very practical. Imagine telling a client: "The computer needs about 30 more minutes to tell me whether I should offer you a trial premium membership." Out-of-sample extensions for semi-supervised algorithms address this issue. They allow to extend graph-based learning to unseen records by either making reasonable simplifying assumptions or combining the algorithm with others that can predict unseen data. We will study multiple variants of both approaches in this master thesis.

Not all business decisions are as harmless the last example. Let's therefore now assume we manage a health-insurance company. An experimental cancer treatment has just arrived on the market. It is very expensive and can cure some specific types of cancer that would be deadly otherwise, but is useless for most types. Our resources being limited, we would like to use classification algorithms to identify patients that are most likely to be curable.

With supervised classification, we could use only very few records of patients that have been treated so far and have either been cured or died. With semi-supervised classification however, we can include all our cancer patients into the data-set. [1] The algorithm has many more examples to learn the difference between curable and incurable patients from. Semi-supervised classification could save lives if it allocates resources more efficiently.

We are however faced with an ethical dilemma: Do we always use the most accurate available

[1] Adding records of all our clients, including the healthy ones, would probably be a mistake though. Unlabeled records would be even more numerous but come from a different population than the one that needs to be classified.

classification algorithm or do other criteria such as transparency play a role as well? Imagine a cancer patient who has been denied the new treatment. Are we obliged to share the details of our semi-supervised algorithm? How might a judge rule after learning that the decision to deny treatment has been based on a data-set where 95% of the records do not actually contain an answer to the question if treatment was successful? More intuitive algorithms are available. We could use rule-based classifiers like decision trees which are very comprehensive but cannot use unlabeled records. [2] Which is more unethical, letting patients die for using a transparent but suboptimal algorithm or basing their livelihood on a sophisticated algorithm they cannot possibly understand?

If we want to defend the existence of ever more complex algorithms, like the ones we study in this master thesis, we need to take a utilitarian stance (Arnsperger & van Parijs, 2003). We aim to maximize well-being, in this example the number of saved lives. Classification maximizes expected well-being of a decision that needs to be taken with imperfect information (Oesterheld, 2015; Grau, 2011). If the algorithm that allows us to maximize well-being happens to be obscure, so be it. Such a position is only defensible if we can demonstrate the superiority of the obscure algorithm.

> This is why, in addition to our research question, we will also verify whether semi-supervised learning really does perform better than supervised learning which would ignore the unlabeled records.

Addressing this question only minimally further complicates our experimental setup.

1.4 Outlook

Chapter 2 constitutes a review of related work. We start by defining with more precision the concepts that were introduced until now and by situating graph-based semi-supervised classification algorithms in the field of machine learning. We go on to review relevant existing techniques. For each of the three phases of graph-based learning, we present several techniques that are related to- or can be used with AEW and/or out-of-sample extensions. In anticipation of our complex experimental setup, the last two sections of chapter 2 review performance metrics and statistical methods that will allow us to answer our research question.

In chapter 3, we present the two new techniques related to our research question in detail: AEW and out-of-sample extensions for graph-based algorithms. We explain the motivation behind those techniques and do not shy away from mathematical details and implementation considerations. This is necessary since both techniques are quite computationally complex.

Chapter 4 starts by formally stating our hypotheses. We compare 7 (meta-)algorithms along a line-up of expected performance. We then present 20 data-sets that we use to test those hypotheses. These data-sets should be representative of a wide variety of application scenarios. After defining our experimental setup with the necessary rigor, we analyze the results of our statistical tests.

Chapter 5 discusses the obtained results in light of our research question. This includes the initial research question regarding out-of-sample extension and associated question whether unlabeled data increases classification performance. We also review some of the encountered limitations and constraints and end this document with suggestions for future research.

[2] Anyone could watch 7 min. of video tutorial and understand the basic principle: Lavrenko and Goddard (2014).

Chapter 2

Related work

We begin this literature review laying out the most relevant taxonomies regarding machine learning algorithms and positioning graph-based algorithms we use within those taxonomies. [1] Section 2.2 follows with an in-depth review of the three phases of graph-based learning algorithms. As our research question of whether transductive algorithms can be extended to inductive testing entails rather complex experimental setups, we include in this literature review section 2.3 on performance metrics as well as section 2.4 on how to handle data-sets and to compare algorithms statistically.

2.1 Types of machine learning

This master thesis treats the subject of classification. We limit ourselves to machine learning algorithms that either perform classification tasks or are used in combination with classification algorithms. This narrows the field somewhat, excluding for example association rules algorithms and considering other types like dimensionality reduction algorithms only insofar as they can be used to prepare and help classification. For the purpose of our analysis, we will distinguish classification algorithms along two main dimensions:

- Is the learning supervised, unsupervised or semi-supervised?
- Is the learning transductive or inductive?

While the preceding two are the most relevant taxonomies when dealing with graph-based algorithms, other distinctions should not go unmentioned (Han, Kamber, & Pei, 2011; Rohrer, 2016; Ayodele, 2010). We will refer to them on occasion. With no claim to exhaustiveness, the following list should provide a first idea how graph-based algorithms compare to others:

- Some algorithms construct *explicit models*, others do not.
  Transductive algorithms usually do not create explicit models.
- Some algorithms rely on a single model, others use *ensembles* of models. When extending our transductive algorithms to the inductive setting, we will use such ensembles.
- Some algorithms are limited to *binary classification*, others are not. The graph-based algorithms we use can inherently handle multiple classes. We hence avoid combining them with other algorithms such as SVMs which do not have that inherent capability.

[1] We could also use the terms categorization or classification of algorithms. That would be confusing when writing about classification algorithms.

- Some algorithms can assign potentially *multiple class labels per record*.
  We consider only the scenario where each record gets assigned a single label.
- Some algorithms provide *classification confidence* or probability, others do not.
  Graph-based algorithms can broadly be seen as providing classification confidence.
- Some algorithms have embedded capabilities to *rank variables* by relevance, most do not.
  Our graph-based algorithms do not have this capability which i.e. decision trees possess.
- Some algorithms support *multi-view* learning. For example, a social media profile contains user generated data (one view that contains structured data like age and location but also unstructured data like opinion posts) and the user's connections to friends (another view, graph-based) (Xu, Tao, & Xu, 2013). We limit ourselves to the single view case of data that is not graph-based and either structured in nature or that has been structured before we hand it to our algorithms (i.e. by information retrieval on text data (S. Weiss, Indurkhya, Zhang, & Damerau, 2010)). The graph-based algorithms we use create a graph out of data that is not already a graph. It would also be possible to use them directly on graph-based data, but this is out of scope here.
- Some algorithms have an inherent capability of dealing with *missing values* for some variables and records. We assume that no data is missing or that default values can be used if needed.

A bit of nomenclature: Throughout the following chapters we will use the terms record, instance and data point as synonyms describing rows in a data-set. Speaking less technically, records are sometimes referred to as examples, input, observations or cases. When those records are represented on a graph, they are also called nodes or vertices and are connected by edges. A graph is directed when it is asymmetric. Undirected graphs are symmetric. Columns of a data-set are called variables or features and are referred to as dimensions in a mathematical context. One column of the data-set is the so-called class variable, it contains discrete labels to be learned, output. Classes can be referred to as categories. The other columns are predictors.

Let's define the two most relevant ways of categorizing graph-based classification algorithms in the following subsections.

2.1.1 Supervised-, unsupervised- and semi-supervised learning (SSL)

In machine learning, supervision refers to the record's output labels. When the learning process is guided by such labels, we speak of supervised learning. Classification and regression algorithms are traditionally supervised. The learning algorithm is fed with labeled examples. Based on the available input data and the corresponding labels, the algorithm searches for commonalities within classes and/or relevant differences across classes. This venture can be seen as a form of artificial intelligence since no human operator needs to tell the algorithm how to distinguish the classes. The machine learns.

Unsupervised learning refers to algorithms that do not use class labels. Clustering is a typical example of unsupervised learning and most dimensionality reduction techniques do not use labels either. Although we do not focus on unsupervised learning here, sub-routines and pre-processing steps of the algorithms we shall study can be unsupervised. Conventionally, as soon as one step in an algorithm requires supervision, the entire algorithm is called supervised.

In many scenarios, labels are scarce and expensive because human experts need to provide them. Unlabeled records are more easily available but cannot be used by a supervised algorithm. Unsupervised techniques could be used on data-sets with scarce labels. However, these techniques cannot use the labels where they are available. Semi-supervised learning can use labeled and unlabeled examples. A semi-supervised classifier will only work if at least one labeled example per class is available. Usually, the unlabeled part of the data-set is much larger than the labeled part (Sammut & Webb, 2011) which motivates the research of semi-supervised algorithms. Semi-supervised learning loosely imitates the way humans learn. As an example, parents would point to some cars and buses (the input data) and tell their child which one they are (the label). The child would observe many more examples of cars and buses without supervision. This unlabeled data helps the child to get a more representative view of the classes of vehicles.

Some sub-routines of a semi-supervised algorithm may well be unsupervised and therefore not able to use the few available labels. Other subroutines may be supervised and therefore not able to use the unlabeled records. Early attempts at semi-supervised learning are wrappers around existing (un)supervised algorithms. Coming from one side, it is possible to use unsupervised clustering to gain as many groups of records as there are classes in the data-set. Each group is given a unique label through a heuristic based on the available labeled records within it (Demiriz, Bennett, & Embrechts, 1999). Coming from the other side, we can get to a semi-supervised setting by using only supervised algorithms iteratively. This so-called self learning adds predicted labels to unlabeled records and then iterates with a larger training set which treats them as labeled records (Triguero, Garcia, & Herrera, 2015). Other well known semi-supervised classifiers are transductive support vector machines (Vapnik, 1999; Junhui Wang, Shen, & Pan, 2007) and the graph-based approaches that we focus on in this master thesis. Zhu (2005), Chapelle, Scholkopf, and Zien (2006) review the available semi-supervised classification algorithms up until ten years ago. Our literature review will add more recent and detailed information where necessary.

Our associated research question is to check the premise of semi-supervised learning that adding unlabeled records helps classification performance. Relatively few studies have tried to answer this question so far. Singh, Nowak, and Zhu (2009) present a theoretical analysis according to which unlabeled data helps in some situations, but not in others. The authors did not consider the important manifold assumption for graph-based learning (see p. 15). Y.-F. Li and Zhou (2015) tackles the question with an experimental study, but regarding transductive SVMs and not graph-based learning. They receive mixed results as well. Chapelle et al. (2006, ch. 4) mention some older papers hat have studied the impact of unlabeled data. They predate the graph-based algorithms that we are interested in and can therefore not have used them (Shahshahani & Landgrebe, 1994; N. Friedman, Geiger, & Goldszmidt, 1997; Nigam, 2001). Results were mixed. No paper concluded that unlabeled data always helps classification performance.

2.1.2 Inductive- and transductive learning

Inductive machine learning is not to be confused with the mathematical notion of proof by induction. Induction shall be understood in the broad classical sense: Based on an observed sample (the training data) we draw conclusions about a population (the whole input space and how any record in it shall be classified) (Borchert, 2006). When the input space contains continuous variables or

a high number of categorical ones, it is not possible to enumerate all possible records in advance. Induction is tractable since the algorithm creates an explicit function from the input space to the output space. This function will be applied to unseen data for out-of-sample prediction.

Vapnik (1999) introduces the terms transduction and transductive learning to machine learning. In contrast to the term induction, transduction does not have a long history in philosophy. [2] Instead of learning a classification function from the training data and then applying this function to predict test data, transductive algorithms go directly from the training data to the test data. Since no classification function is being learned, transductive learning solves a smaller problem than inductive learning. Transduction does not need to worry about how an arbitrary record should be classified and cannot predict unseen test data. Vapnik makes the argument that we should not feel obliged to solve a bigger problem than necessary.

According to Chapelle et al. (2006, ch. 25), the dimensions of transductive/inductive learning on the one hand and supervised/semi-supervised learning on the other hand are independent, orthogonal. This may be the case in theory. In practice though, the concepts are correlated since many semi-supervised algorithms are transductive in nature. Graph-based algorithms are typically semi-supervised where many other algorithms are limited to a supervised setting. At the same time, graph-based algorithms are only transductive where other algorithms can be used inductively (W. Liu, Wang, & Chang, 2012). Jebara, Wang, and Chang (2009) even use the words transductive and semi-supervised synonymously. Transductive supervised algorithms are very rare.

2.2 Graph-based learning

Graph-based learning can be seen as a three-step procedure that we shall review in detail:

1. Based on structured data, generate a graph consisting of nodes and edges
   Nodes of the generated graph correspond one to one to the data-set records
2. Give weights to the graph edges
3. Propagate labels on the graph to estimate classes of unlabeled nodes

Different techniques are available at each step. AEW is situated at the second step. The three steps are modular and can be combined in different ways. Nevertheless, some combinations work better than others. We will focus on those techniques, that combine well with AEW.

A. Subramanya and Talukdar (2014) provide a review of graph-based semi-supervised learning. As this subject is at the heart of our research question, we will still go to the original papers for specific details. The authors of this short book and the corresponding slides are mostly known for their work on the third step of the procedure (see P. P. Talukdar and Crammer (2009)), whereas we concentrate on the second step. We cannot limit ourselves to this source.

Before we dive deeper into the algorithms, let's agree on mathematical notation. To avoid confusion, we adopt the nomenclature from the original AEW papers Karasuyama and Mamitsuka (2013a, 2016). This notation can also be found in prominent papers like D. Zhou, Bousquet, Lal, Weston, and Schölkopf (2004), Zhu, Ghahramani, Lafferty, et al. (2003), W. Liu et al. (2012). Graph-based classification can be framed in the following way without losing generality:

[2] The term transduction is used with entirely different meanings in biology and in electrical signal processing though.

"Suppose that we have n feature vectors $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, where $\mathbf{x}_i \in \mathbb{R}^p$. An undirected graph $\mathcal{G}$ is generated from $\mathcal{X}$, where each node (or vertex) corresponds to each data point $\mathbf{x}_i$. The graph $\mathcal{G}$ can be represented by the adjacency matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ where (i,j)-element W_{ij} is the weight of the edge between $\mathbf{x}_i$ and $\mathbf{x}_j$. [...] Let $\mathbf{F} \in \mathbb{R}^{n \times c}$ be a scoring matrix which gives estimation of labels, where c is the number of classes or clusters. [...] Suppose that the first l data points in $\mathcal{X}$ are labeled by $\mathcal{Y} = \{y_1, \ldots, y_l\}$, where $y_i \in \{1, \ldots, c\}$. [...] The scoring matrix $\mathbf{F}$ gives an estimation of the label of $\mathbf{x}_i$ by $\arg\max_{k \leq c} F_{ik}$. [...] $\mathbf{Y} \in \mathbb{R}^{n \times c}$ is the label matrix with $Y_{ik} = 1$ if $\mathbf{x}_i$ is labeled as $y_i = k$; otherwise, $Y_{ik} = 0$" (Karasuyama & Mamitsuka, 2016, p. 4)

Note that this means $Y_{ik} = 0 \ \ \forall k$ when dealing with an unlabeled example i. [3]

Let's look at the available techniques phase by phase.

2.2.1 Graph construction

In graph-based learning, the first step of graph construction is sometimes called graph sparsification because the resulting graphs are very sparse. Every node is connected to typically only a handful to a few dozen other nodes, not to most nodes of a large graph. This is the result of constraints which limit the degrees of nodes. Before it can be decided which connections to keep, we need to compute the similarities of pairs of nodes on the graph.

The quality of the constructed graph is very important to classification performance (Zhu, Ghahramani, Lafferty, et al., 2003; F. Wang & Zhang, 2008). A graph is of high quality when there are many strong connections within classes and only a few weak connections between classes. Nevertheless, graph construction is generally unsupervised learning. It does not use class information.

We shall first investigate how graphs can be constructed from feature vectors through distance measures. We then present different sparsification methods based on distance measures and will end this section by explaining how graph construction influences subsequent classification performance and which assumptions were implicitly made.

Distance and dissimilarity The following graph construction methods all rely on distance- or, more generally, on dissimilarity measures between nodes. Let's note a dissimilarity function between two records i and j as $d(\mathbf{x}_i, \mathbf{x}_j)$. The corresponding similarity function can be defined as $s(\mathbf{x}_i, \mathbf{x}_j) = 1/(\varepsilon + d(\mathbf{x}_i, \mathbf{x}_j))$ in case of unscaled distances. [4] In case of scaled dissimilarities between 0 and 1, the similarity $s(\mathbf{x}_i, \mathbf{x}_j) = 1 - d(\mathbf{x}_i, \mathbf{x}_j)$ (Han et al., 2011, ch. 2). We usually pre-process data before computing dissimilarities. Many data-sets combine different types of variables, we call them heterogeneous data-sets. According to Stevens (1946), there are 4 types:

- *Nominal* categorical variables
- *Ordinal* categorical variables with a hierarchy of categories
- *Interval* continuous variables

[3] We have slightly changed the notation from Y_{ij} in Karasuyama and Mamitsuka (2016) (likely a typographical error) to Y_{ik} which is more consistent with other notations like F_{ik} and avoids confusion when used in conjunction with W_{ij}.

[4] ε is a relatively small number which shall only prevent division by 0.

- *Ratio* continuous variables with a meaningful notion of zero

Nominal- and ordinal variables are called discrete or categorical while interval- and ratio variables are continuous. It is possible to encode all variable types, including nominal- and ordinal ones, as real numbers without loss of generality. It needs to be remembered what they represent though. The algorithm needs to know which variable is of which type and how the encoding of categorical variables as numbers was done.

Interval and ratio type variables are considered equivalent for machine learning purposes. All relevant operations can be performed on both and they are already real numbers. In data-sets that contain only such variables, distances between records can directly be computed using the Euclidean distance (if $q = 2$) or the more general Minkowski distance in equation (2.1). The squared Mahalanobis distance in equation (2.2) is similar to the Euclidean distance but takes correlations between variables into consideration. $\Omega \in \mathbb{R}^{p \times p}$ denotes the covariance matrix of the variables (A. Jain & Dubes, 1988).

$$d(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{d=1}^{p} |x_{id} - x_{jd}|^q \right)^{1/q} \tag{2.1}$$

$$d(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i - \mathbf{x}_j\right)^\top \Omega^{-1} \left(\mathbf{x}_i - \mathbf{x}_j\right) \tag{2.2}$$

Gower (1971) has developed a method to scale and transform mixed types of variables into an integrated similarity measure which has been extended by Podani (1999). [5] If we want to construct a similarity based graph on heterogeneous data, we can do the following for variables $k \in \{1, \ldots, p\}$:

- Continuous variables need to be scaled between 0 and 1 such that no one variable will dominate for unwanted reasons such as differing measurement units. Scaling is done by dividing any given difference by the range of the variable. Outliers should be removed before scaling.

$$d(x_{ik}, x_{jk}) = \frac{x_{ik} - x_{jk}}{\max_\alpha \ x_{\alpha k} - \min_\omega \ x_{\omega k}} \tag{2.3}$$

- Nominal variables have $d(x_{ik}, x_{jk}) = 0$ when both records fall into the same category, 1 if not.
- Asymmetric binary variables are a special case. Positive matches ($x_{ik} = x_{jk} = 1$) are given 0 dissimilarity. Mismatches ($x_{ik} \neq x_{jk}$) are given dissimilarity 1. Negative matches ($x_{ik} = x_{jk} = 0$) are not counted. In such a case, the dissimilarity between two records is measured on the remaining variables. This recognizes that having a mutual binary property is usually a more meaningful indicator of similarity than mutually not having one (Jaccard, 1912).
- For an ordinal variable, the possible states are ranked with $r_{ik} \in \{1, \ldots, M_k\}$. The dissimilarity can then be expressed as the scaled difference of ranks:

$$d(x_{ik}, x_{jk}) = \frac{|r_{ik} - r_{jk}|}{M_k - 1} \tag{2.4}$$

- In case of missing data, similarities between two records are measured on remaining variables.

[5] We use a library function called `daisy`. It was first written in Fortran for the book Kaufman and Rousseeuw (1990, p. 52). Wrappers around the original functions are now available for R (Maechler, Rousseeuw, Struyf, Hubert, & Hornik, 2016; Struyf, Hubert, & Rousseeuw, 1997) and MATLAB (Verboven & Hubert, 2010, 2005).

This procedure scales $\mathbf{x}_i \in \mathbb{R}^p$ to $\mathbf{x}_i \in [0,1]^p$. It thereby equalizes the weights of variables. [6] While it is desirable to scale away variable weightings that stem from measurement units and other causes that are similarly irrelevant to classification, we might want to give some variables explicitly more weight since they are known to be very relevant for classification. This can be achieved by re-scaling the data with weights $\omega \in \mathbb{R}^p$ after the standardization. We are now ready to compute $d(\mathbf{x}_i, \mathbf{x}_j)$ as the euclidean distance between the relevant scaled variables of $\mathbf{x}_i$ and $\mathbf{x}_j$.

Image classification, which is essential to such tasks as computer vision and autonomous driving, can also be done using real valued feature vectors. Each pixel of a digital image is usually stored in 8 or 10 bits for each of three color channels. A picture of p_x times p_y pixels can be represented in the high dimensional space of $\mathbb{R}^{p_x \times p_y \times 3}$. Euclidean distances could be computed on such raw data, but this is seldom the best choice. Pixel-wise euclidean distances do not reflect the position of pixels in the image. For instance, images with intricate detail will be found similar to much less detailed images that have a comparable average color. Conversely, images with the same kind of detail will be found dissimilar when only the color tone differs. Custom dissimilarity measures exist to address this issue. Jiang Wang et al. (2014) propose an image dissimilarity ranking computed by deep neural networks. Other dissimilarity measures like the cosine similarity (the normalized dot product between two vectors) and the χ^2 distance are popular for high dimensional data in text mining (Jebara et al., 2009; Deza & Deza, 2014). Such specialized dissimilarity measures are not easily compatible with the further usage of distances during AEW (see p. 38).

Computing dissimilarities among nodes has an asymptotic temporal complexity of $\mathcal{O}(pn^2)$. There are $\frac{n(n-1)}{2}$ node pairs and each dissimilarity is computed over p dimensions. This is too complex for big data scenarios with millions of records. Delalleau, Bengio, and Le Roux (2005), W. Liu, He, and Chang (2010), W. Liu et al. (2012) propose a solution called anchor graphs. Only $m \ll n$ nodes are so-called anchors and found through k-means clustering. Relevant computations are done for m instead of n nodes. As m can be held independent from n, complexity drops to $\mathcal{O}(pn)$.

Knowing to compute dissimilarities between nodes, we now look at different graph sparsification methods. We assume that a full distance or dissimilarity matrix $\mathbf{d} \in \mathbb{R}^{n \times n}$ has been constructed. P. P. Talukdar (2009) provides a review of graph sparsification methods (sometimes combined with edge weighting methods). We focus mostly on those methods that are related to- or usable in conjunction with AEW.

kNN graph k nearest neighbor techniques can be used not only for instance-based learning such as classification by majority voting, but also for sparsifying a graph. We keep edges only between a node i and its k nearest neighbors. In contrast to inductive kNN classification, transductive kNN graph-based methods need to find the nearest neighbors of every node, not only of the new nodes arriving at test-time. Indeed, instance based kNN classification is called lazy learning since it only stores the data-set and defers computations to test-time. With kNN graphs used in transductive learning, there is no test-time.

[6] Differences in nominal variables will almost always be given more weight than differences in continuous- or ordinal variables with multiple states. In case of non match, the dissimilarity on nominal variables is always 1 whereas it is only a fraction of 1 for other variable types. Are nominal variables really more important than other types?

$$W_{ij} = \begin{cases} 1, & \text{if } j \in \mathcal{N}_i \text{ OR } i \in \mathcal{N}_j, \\ 0 & \text{otherwise} \end{cases} \quad (2.5) \qquad W_{ij} = \begin{cases} 1, & \text{if } j \in \mathcal{N}_i \text{ AND } i \in \mathcal{N}_j, \\ 0 & \text{otherwise} \end{cases} \quad (2.6)$$

It is possible that a node i is one of the kNN of another node j but not vice versa. This happens with nodes at central locations which are close neighbors of more than the k nodes that can be considered as their neighbors. In order to have a symmetric graph, both edges will be set to 1 or to 0 in these cases: $W_{ij} = \max(W_{ij}, W_{ji})$ as in equation (2.5) and (2.7) or $W_{ij} = \min(W_{ij}, W_{ji})$ as in equation (2.6). With the first, and more commonly used, case, a node can have more than k neighbors: $\sum_j W_{ij} \geq k \quad \forall i$. With the second solution $\sum_j W_{ij} \leq k \quad \forall i$. In both cases, the graph is undirected and self-loops are not allowed. $\mathcal{N}_i$ denotes the set of nearest neighbors of node i.

Equation (2.7) could be solved by branch and bound (Rardin, 1998). In practice, there are simpler methods. We can take the dissimilarity matrix $\mathbf{d}$ and sort each row while keeping track of the node indices in a separate matrix. Using quicksort this takes $\mathcal{O}(n \log n)$ per row in the average case and $\mathcal{O}(n^2)$ in the worst case. Sorting the whole matrix and then cutting it after $k+1$ columns would be wasteful though. We can instead perform and exhaustive search for the smallest row element and cut and paste it and its node index into a separate matrices. Iterating this procedure $k+1$ times guarantees us to find the kNN of a node in $\mathcal{O}(kn)$ operations. Repeating this for n rows takes at worst $\mathcal{O}(kn^2)$ operations which may or may not be more than the previous $\mathcal{O}(pn^2)$ for creating matrix $\mathbf{d}$. We fill the indices as binary weights into a sparse matrix $\mathbf{W}$ according to the AND or the OR rule and manually remove the self edges by setting the diagonal to zero.

The aforementioned procedure is called exhaustive search or brute force because it compares all nodes witch each other. J. H. Friedman, Bentley, and Finkel (1977) proposes a faster implementation that recursively partitions the k dimensional input space with a kd-tree. Each node is compared to only a small subset of plausible neighbor candidates, defined by the tree partitions. This method works best with a large number of records in a low-dimensional space. The computational advantage disappears in high-dimensional spaces, starting at 10 or 20 dimensions, which is a common occurrence in machine learning. [7]

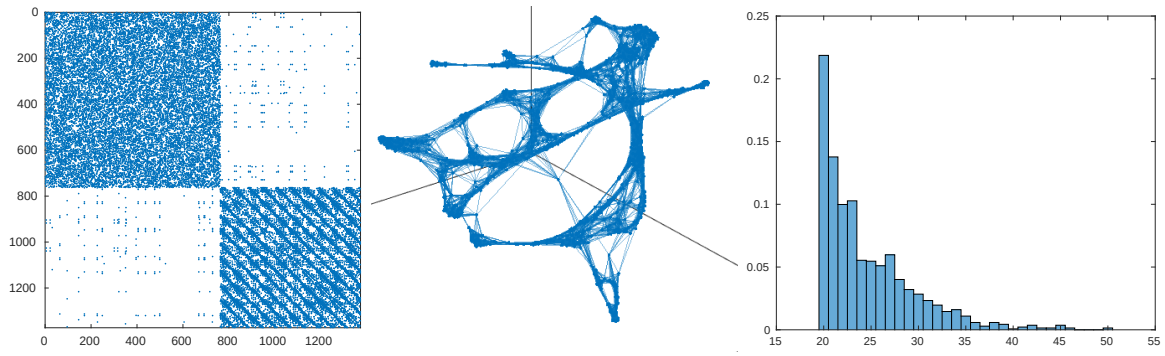
Nearest neighbor search is used not only in machine learning, but also with databases in general and for compression. Locality sensitive hashing (LHS) allows tractable search for approximate nearest neighbors (ANN) across large collections of high dimensional objects:

> "The basic idea is to hash the points from the database so as to ensure that the probability of collision [identical hashes for different objects] is much higher for objects that are close to each other than for those that are far apart."
> (Gionis, Indyk, Motwani, et al., 1999, p. 518)

> "A hash function is a well-defined deterministic algorithm that takes as input data of arbitrary length and produces a short fixed-length digital representation of the data, or a digest, as its output." (Henderson, 2009, p. 1287)

Voluntary collisions may sound counterintuitive. When used in encryption, hashes are not supposed to collide, here they are. LHS works well with 50 or more dimensions and has recently been

[7] Matlab uses kd-trees per default for 10 or fewer dimensions. Furthermore, the implementation of function `daisy` that we use for Gower's metric compares all nodes anyway.



(a) adjacency matrix $\mathbf{W}$ (b) 3D representation (c) Node degree histogram

Figure 2.1: Binary graph constructed on the `banknotes` data-set (sorted by class) There are very few edges between classes, we can almost see separate components. According to equation (2.5), nodes can have a degree ≥ 20

applied to graph-based learning: Y. Weiss, Torralba, and Fergus (2009) use such hashing in unsupervised fashion. Jun Wang, Kumar, and Chang (2012) extend to the semi-supervised case, using available labels for hashing. This latter paper is also a review of hashing methods for approximate nearest neighbor search.

kNN graphs can contain multiple disjoint components, subsets of nodes which are not connected by any path of edges (Eppstein, Paterson, & Yao, 1997). This can happen when the underlying classes are well separated and is more likely when small values of k are chosen. One might think that it is good to have multiple components. In the ideal case, we would have as many components in the graph as we have classes in the data-set. Every component would contain only vertices from one class. Knowing that the graph construction algorithm does not look at the few available class labels and that real-world data is seldom perfectly separable, it is preferable to aim for graphs with only one component. The quality of the graph for classification will depend on having many intraclass edges and only few interclass edges. The subsequent step of edge weighting shall also try to attribute smaller weights to those interclass edges.

Figure 2.1 shows a graph constructed from the `banknotes` dataset downloaded from the UCI machine learning repository (Lichman, 2013). On the left, we see the sparsity pattern of the binary adjacency matrix built with kNNG ($k = 20$). Records are ordered by class. Almost all connections are within a class, the graph is of good quality. In the middle, we see a 3D projection of the graph and on the right a histogram of node degrees based on equation (2.5).

$$\begin{aligned} \min_{\mathbf{W}\in\{0,1\}^{n\times n}} \quad & \textstyle\sum_{ij} W_{ij} d_{ij} && (2.7) \\ s.t. \quad & \textstyle\sum_j W_{ij} \geq k && \forall i \\ & W_{ii} = 0 && \forall i \\ & W_{ij} = W_{ji} && \forall i,j \end{aligned} \qquad \begin{aligned} \min_{\mathbf{W}\in\{0,1\}^{n\times n}} \quad & \textstyle\sum_{ij} W_{ij} d_{ij} && (2.8) \\ s.t. \quad & \textstyle\sum_j W_{ij} = b && \forall i \\ & W_{ii} = 0 && \forall i \\ & W_{ij} = W_{ji} && \forall i,j \end{aligned}$$

ϵ neighborhood graph Alternatively to connecting the k nearest neighbors of a node, however far apart they may be, we can also construct a hyper-sphere of the same radius ϵ in $\mathbb{R}^p$ around each node and connect all nodes included in it. Doing so, we risk graphs that are not very sparse if ϵ is

too large and graphs with disconnected components if ϵ is too small. When there are regions of differing density it may be difficult to find a compromise that avoids both issues.

Maier, Luxburg, and Hein (2008) show that ϵ neighborhood graphs generally lead to weaker clustering performance than kNN graphs.

Silva and Zhao (2016, p. 108) propose to combine kNN and ϵ neighborhood graphs by connecting a node to at least k neighbors if it lies in a sparse region or to more than k neighbors in a dense region where the ϵ hypersphere contains more. The authors thus hope to combine the respective strengths of both approaches.

b-matching Especially with high dimensional data, we run the risk of having hub nodes (Radovanović, Nanopoulos, & Ivanović, 2010; Zimek, Schubert, & Kriegel, 2012). If we use kNN graphs, these nodes will have $\gg k$ edges and make the graph unbalanced. b-matching addresses this issue.

A perfect b-matching of a set of nodes is a graph where each node is met by exactly b vertices (Floudas & Pardalos, 2008, p. 106-108) (Padberg & Rao, 1982). Jebara et al. (2009) propose to use b-matching for semi-supervised learning. The problem can be expressed as in equation (2.8).

There is no theoretical guarantee why b-matching graphs should be better for classification than kNN graphs. Experimental results in Jebara et al. (2009) show that by equating $b = k$ for different values, b-matching most of the time outperforms kNN because it is more robust to changes of b.

Huang and Jebara (2007, 2011) propose a faster way to compute a b-matching for bipartite graphs. The graph is separated into labeled nodes and unlabeled nodes. The b-matching finds the b best labeled nodes for each unlabeled node. Unlabeled nodes are then labeled by majority vote among the connected labeled nodes. This algorithm cannot make use of unlabeled data for classification and is therefore one of few transductive supervised methods. [8] This approach cannot be combined with AEW which allows edges between unlabeled nodes. [9]

LRR A new technique called low rank representation (LRR) (Lin, Chen, & Ma, 2010) has recently been applied to graph-based semi-supervised learning by S. Yang, Wang, Wang, Han, and Jiao (2013). The objective is to represent the data matrix $\mathbf{X}$ by a base dictionary $\mathbf{D}$ multiplied by a low rank combination of coefficients in matrix $\mathbf{Z}$. This approach is parameter free. The mathematical adjustments to make the NP-hard problem equation (2.9) tractable and useful in a SSL context are complex.

We will not investigate this further since AEW is incompatible with LRR. LRR performs at least the first two steps of graph-based learning and sometimes all three steps in a very specific manner, preventing modularity.

LRR has been combined with b-matching (S. Li & Fu, 2013) and used to establish similarity measures (S. Li & Fu, 2015). Peng, Lu, and Wang (2015) review the already quite numerous variants of LRR.

[8] One might argue that connections between other unlabeled nodes with labeled nodes indirectly influence which remaining labeled nodes an unlabeled node can connect to. The algorithm might be called semi-supervised in that very narrow sense.

[9] See Sanghavi, Malioutov, and Willsky (2008) for a fast algorithm for unipartite b-matching and Fremuth-Paeger (2016) for an open source implementation.

$$\min_{\mathbf{Z}} \ \text{rank}(\mathbf{Z}) \qquad (2.9)$$
$$s.t. \ \ \mathbf{X} = \mathbf{DZ}$$

Now that we know different ways to sparsify a graph, let's look at different assumptions that this approach implicitly makes and how those assumptions influence the practical performance of such graph-based classification algorithms.

The smoothness- and manifold assumption Graph construction is done exclusively on the input features and without using the labels in $\mathcal{Y}$. This makes the following implicit assumption:

> "*If input features and labels share the same adjacency matrix* (i.e., sharing the same local structure), the minimization of the objective function [later, in step 2 of the procedure] should estimate the adjacency matrix by accurately propagating labels of graph nodes." (Karasuyama & Mamitsuka, 2016, p. 9)

Graph construction presupposes this link between input features and labels. This is the *smoothness assumption* according to which similar nodes should have similar label estimations. Labels change smoothly on the graph. The *manifold assumption* states that records are densely concentrated in well separated regions (manifolds) of the input space. The fact that the manifolds should be well separated by regions of low density is sometimes given the separate name "cluster assumption" and sometimes understood to be implicit in the manifold assumption. We will follow the latter convention. The manifold assumption is mostly used in graph-based learning but also for some deep neural networks (I. G. Y. Bengio & Courville, 2016). The smoothness assumption is local in nature while the manifold assumption, depending on the size of the manifolds, can be global with respect to the input space (Chapelle et al., 2006, ch. 1.2).

> "[A manifold is] a geometric object which locally [not always globally] has the structure (topological, homological, etc.) of $\mathbb{R}^n$ or some other vector space." (Hazewinkel, 1997)

Machine learning algorithms search for lower dimensional manifolds which are embedded in the much higher dimensional input data space:

> "[The term manifold is] used more loosely to designate a connected set of points that can be approximated well by considering only a small number of degrees of freedom, or dimensions, embedded in a higher-dimensional space. Each dimension corresponds to a local direction of variation." (I. G. Y. Bengio & Courville, 2016, p. 163)

Learning the manifold structure should not be understood too literally. A graph-based algorithm will not output explicit descriptions of the dimensionality, size and embedding of the manifolds. It would be inductive if it did since unseen records would then only need to be compared with these explicit manifold regions. Instead, the graph-based algorithm will connect records that are close to each other. If the manifold assumption holds true, this reconstructs the underlying manifolds.

The synthetic two moons data-set is a simple illustration of manifolds (D. Zhou et al., 2004). Two one dimensional manifolds are warped in the shape of moons and embedded, with some noise,

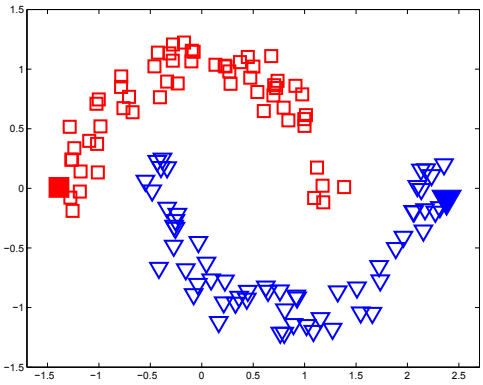


Figure 2.2: Two moons data-set
An illustration of 1D manifolds embedded in 2D space
(D. Zhou, Bousquet, Lal, Weston, & Schölkopf, 2004, p. 2)

in a two dimensional space (see figure 2.2). Locally, those embedded manifolds resemble a one dimensional straight line, but globally the moon shape diverts from such tangent lines.

Practical examples of manifolds include grayscale image recognition and scans for text recognition, where images do not cover the entire input space $\mathbb{R}^{p_x \times p_y}$, but only specific regions. We do not expect training data to take whichever form in the input space, most of which would be white noise. We assume that data comes from a limited number of low dimensional manifolds. Ideally, there are as many well separated manifolds as there are classes c.

Curse of dimensionality The manifold assumption allows to avoid the so-called curse of dimensionality, an umbrella term that covers among others the following issues (Sammut & Webb, 2011):

- The number of training examples n needs to grow exponentially when the number of dimensions p grows linearly (Hughes, 1968). This would be the case if the algorithm needs to search for interesting variations along all possible directions in $\mathbb{R}^p$. By restricting ourselves to lower dimensional manifolds, we need much fewer training examples.
- When the dimension of an euclidean space increases, the distance of a record to its nearest neighbor approaches the distance to the furthest record. This so-called distance concentration effect hurts the performance of any distance-based algorithms. This is however much less problematic in data-sets where the examples lie in well separated clusters or manifolds (Beyer, Goldstein, Ramakrishnan, & Shaft, 1999; Zimek et al., 2012).

It is advantageous to have many unlabeled nodes. The point cloud within the manifolds will be denser and the distances from nodes to their neighbors with the same label will be smaller compared to inter-class distances. Not every node needs to be similar to every other node of the class. It suffices when a path across the manifold can be constructed because every node has some close neighbors of the same class. Nodes from different classes being measured as similar is a problem though, falsely attributed labels can be propagated. Hence the need for the smoothness assumption in addition to the manifold assumption.

Potential weaknesses of graph-based learning Even under the manifold- and smoothness assumption, the following vulnerabilities of graph-based learning remain:

- Graph construction cannot detect noise in a variable.
  Noise manifests through variations in a feature vector which are not linked to labels. Usually, those are rare but large deviations (i.e. from encoding errors) (Sammut & Webb, 2011). [10] Graph construction does not look at labels and thus cannot tell the signal from the noise in the variables. The smoothness assumption implicitly excludes the presence of such noise. Nevertheless, it is worth noting that this might be a strong assumption.
- Graph construction assumes that all variables are relevant and helpful to classification.
  Since it cannot detect noise in a variable, graph construction cannot eliminate variables which are only white noise and have no signal. There might even be variables which predict another partition of the records than the one that we wish to classify on. Graph construction is vulnerable to those kinds of parasitic variables that harm classification performance.
- Graph construction cannot detect colinearity among variables and treats them independently.

The first vulnerability can to an extent be addressed in the second step of the procedure: Some edge-weighting techniques make the graph less vulnerable to over-fitting and potential propagation of existing noise in the feature vectors. The newly proposed AEW is one such technique.

Feature selection & engineering The performance of graph-based learning highly depends on the quality of the input graph, which in turn depends on the quality of the feature vectors. In practice, this means that graph-based methods perform well on some data-sets and less well on others. Nevertheless, some precautions can be taken.

Feature selection and feature engineering techniques can help with the latter two vulnerabilities of the bullet list. In supervised feature selection, simple statistical tests can give an indication of the usefulness of variables for classification. Categorical variables yield a confusion matrix when compared to the class label. They can be ranked according to their p-values on χ^2 tests. Continuous variables can be ranked according to p-values on t-tests. Some supervised inductive algorithms like decision trees have an embedded capability of detecting the most useful variables. Using such techniques on the labeled data before starting the graph construction can already eliminate some useless variables. Selecting features upfront and independently from the graph-based algorithm corresponds to what Kohavi and John (1997) call the filter approach. As opposed to the wrapper approach which would require to iterate the graph-based algorithm with different combinations of features.

Reducing the number of features is not only useful when there are irrelevant and parasitic features, it also helps reducing the computational complexity of the dissimilarity measures in high dimensional data-sets. Principal component analysis can reduce the number of variables and also eliminate colinearity based on eigenvalue decomposition (Pearson, 1901). For text mining, latent semantic indexing reduces the number of dimensions which would otherwise be in the tens of thousands of words in a given language. It is based on singular value decomposition and deals with synonyms and polysemous words (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990; Hofmann, 1999).

[10] Variations that are small and random but common (i.e. continuous variables which cannot be measured with certainty) are benign in machine learning applications and therefore usually not even called noise.

We can also use unsupervised feature selection techniques. These techniques were originally developed for clustering where labels are not available, but they might also be preferable to supervised feature selection in our case since they can make use of the whole data-set. In Dy and Brodley (2004), the goal is to find feature subsets that distinguish "interesting" natural clusters. The authors suggest to determine the feature subset at the same time with the number of clusters, which we cannot change in classification. Analogous to semi-supervised learning, Zhao and Liu (2007) propose semi-supervised feature selection through spectral analysis which can use the available labels and the unlabeled records to choose the best feature subset.

2.2.2 Edge weighting

Now that a sparse graph has been constructed, two main approaches to edge weighting exist: similarity-based edge weighting (usually through Gaussian kernels) and local linear embedding. The AEW technique that we will analyze in section 3.1 is essentially a combination of similarity- and local linear embedding based edge weighting.

Gaussian kernel It comes intuitively to mind that edge weights should depend on the same distances/dissimilarities among nodes that were already used for graph sparsification. Similar nodes should be joined by large weights. The most popular way to do this is through the so-called Gaussian kernel in equation (2.10). Zhu and Ghahramani (2002) named the kernel after the probability density function of the famous normal distribution equation (2.11). Belkin and Niyogi (2001) call the same function a heat kernel since the solution to the heat equation in euclidean space takes Gaussian form. Labeled nodes can metaphorically be seen as heat sources that diffuse their labels to neighbors.

Numerous other kernel functions do exist and could be applied to edge weighting (Shawe-Taylor & Cristianini, 2004), but this area of research has not gained much interest. For some types of data that require specific dissimilarity measures, it might not even be possible to use equation (2.10). We will look deeper into this issue in section 3.1.1 p. 38 since it equally applies to AEW.

$$W_{ij} \leftarrow W_{ij} \times \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{2\sigma^2}\right) \qquad (2.10) \qquad \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \qquad (2.11)$$

Zhu and Ghahramani (2002) propose to compute the bandwidth parameter σ based on the following heuristic: A minimum spanning tree according to the Kruskal (1956) algorithm is constructed over the graph. Let's reuse dissimilarities between nodes $d(\mathbf{x}_i, \mathbf{x}_j)$ as they have been computed to construct the binary graph. Initially, all nodes are singleton components. This greedy algorithm starts with the shortest edge lengths (smallest dissimilarities) and proceeds to the longer ones, connecting two nodes only when they belong to different components. The first edge that connects nodes with differing labels has length d_0 and determines $\sigma = d_0/3$. Knowing that only outliers are more than 3 standard deviations apart from the mean in a normal distribution, this will set the edge weight close to zero. Subsequent edge weights will be even closer to zero, disconnecting nodes with differing labels. This heuristic needs labels in order to work reliably. If labels are scarce, d_0 tends to be too large. Since the chance of connecting any two labeled nodes is small when labels are rare,

differently labeled nodes will be connected too late in the heuristic. Most edge weights, even the desirable ones between nodes in the same manifold, will be small.

Zhu and Ghahramani (2002) also propose to use multiple σ_d parameters, one per variable in the data-set, and optimize them taking the minimum entropy of classification confidence on unlabeled nodes as objective function. Optimal parameters should minimize equation (2.12) where $p_i(k)$ is the estimated probability of node i belonging to class k and where the $n-l$ unlabeled nodes come after the l labeled nodes in the data-set. H is small when every node is classified confidently. Optimizing σ parameters is exactly what we will do with AEW in section 3.1. In contrast to AEW, the entropy optimization uses label estimations and therefore integrates step 2 and 3 of graph-based learning, losing modularity.

$$H = \frac{1}{n-l} \sum_{i=l+1}^{n} \sum_{k=1}^{c} -p_i(k) \log p_i(k) \tag{2.12}$$

According to D. Zhou et al. (2004), it remains difficult to estimate σ_d when only few labels are available. F. Wang and Zhang (2008) further remark that even small inaccuracies in estimating σ can have large negative consequences for classification performance.

Local linear embedding LLE has been proposed by Roweis and Saul (2000), Saul and Roweis (2003) as a dimensionality reduction technique in three steps:

1. A sparse binary graph is constructed (see section 2.2.1)
2. In the high dimensional space, compute weights that reconstruct each node by its neighbors
3. Compute vectors in a lower dimensional space that are best reconstructed by those weights

When using local linear embedding for classification, as done by F. Wang and Zhang (2008), Daitch, Kelner, and Spielman (2009), Cheng, Liu, and Yang (2009), we do not need to compute the vectors in the lower dimensional space and can stop after step 2. We will therefore not explain the last step in detail. Equation (2.13) shows how the weights are used as the decision variables of the constrained optimization ($j \sim i$ are all nodes j that are connected to a node i):

$$\begin{aligned} \min_{\mathbf{W}} \quad & \textstyle\sum_{i=1}^{n} \left\| \mathbf{x}_i - \sum_{j \sim i} W_{ij} \mathbf{x}_j \right\|_2^2 & \\ \text{s.t.} \quad & \textstyle\sum_{j \sim i} W_{ij} = 1 & \forall i \\ & W_{ij} = 0 & \forall i \quad \forall j \notin \mathcal{N}_i \\ & W_{ii} = 0 & \forall i \end{aligned} \tag{2.13}$$

The objective is to minimize the sum of square errors committed when doing a linear approximation of each node by its neighbors. LLE can be optimized in closed form and is not subject to local minima. Classification can only use an existing manifold structure in the input data if k, b or ϵ have been chosen large enough to allow an approximation of $\mathbf{x}_i$ their neighbors. When nodes are reconstructed based on fewer neighbors, a single neighbor gains in influence. Neighborhoods that are too small therefore lead to high variance in $\mathbf{W}$, noise sensitivity. At the same time, neighborhoods need to be small enough to remain on the manifold, even if this manifold is embedded with curvature in the high-dimensional input space. If a neighborhood is too large, it cannot follow the

curvature (Izenman, 2009, ch. 16.6) (Y. Bengio, Delalleau, & Roux, 2006, ch. 11.6). Figure 2.3a shows an acceptable size. Manifold curvature lets the euclidean distances to nearest neighbors, as they were used to construct the graph, differ from the geodesic distances on the manifold. Points which are far apart in the manifold (if we "roll it out") can be closer in the high dimensional space (see figure 2.3b). Because of this, nodes can be seen as direct neighbors when they should not. In the worst case, manifold curvature within the neighborhood can also connect nodes as neighbors which do not lie on a same manifold. This introduces bias in $\mathbf{W}$. There are several variants of LLE:

- Contrary to Gaussian kernels, LLE does not guarantee that edge weights reflect node similarity. When two very similar nodes are connected to the same neighbor, the weights of those two connections can differ considerably. Saul and Roweis (2003) introduce a regularization term to address this issue. However, this regularization parameter is difficult to tune.
- F. Wang and Zhang (2008) introduce linear neighborhood propagation (LNP), a close variant of LLE with two additional constraints: The matrix $\mathbf{W}$ must be symmetric and contain only positive edge weights.
- Daitch et al. (2009) introduce the hard-graphs variant of LLE where each node degree is constrained to be > 1 and sparsification is done at the same time as edge weighting. This variant as well only allows for positive weights and symmetric edges.
- α soft graphs are a variant of hard graphs where the degree is allowed to be ≤ 1 for some nodes in sparse regions. The hyperparameter α needs to be tuned (Daitch et al., 2009).

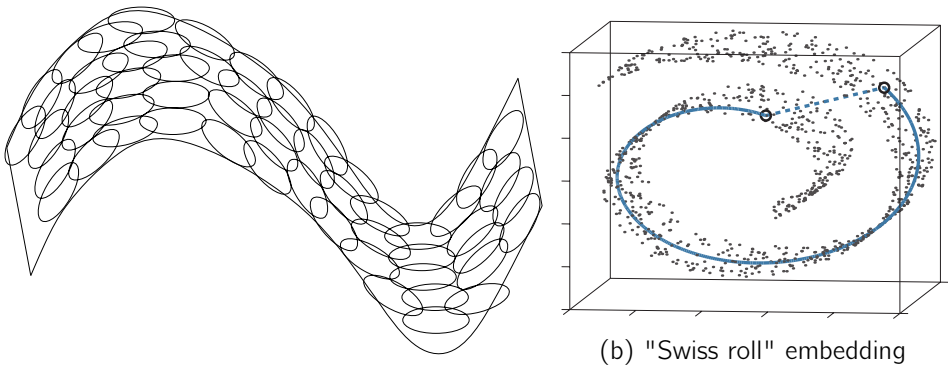
In their review, Chen and Liu (2011) come to the conclusion that LLE's noise sensitivity as well as its inability to generalize to unseen data remain unsolved problems. We shall try to address both these problems: AEW in section 3.1 makes edge optimization less sensitive to noise and section 3.2 proposes inductive extensions to unseen data. Other recent attempts are Kong, Ding, Huang, and Nie (2012) who propose iterative LLE and Elhamifar and Vidal (2011) who integrate sparsification and LLE, allowing neighborhoods of different sizes depending on the manifold.

Both presented edge weighting techniques and their variants maintain the preceding graph sparsification: Equation (2.10) multiplies the existing binary edges with weights. Equation (2.13) has an explicit sparsity constraint. This makes the first two steps of graph-based learning modular. Techniques can be combined in different ways, including the third phase that we shall study now.

2.2.3 Label propagation

Having sparsified and weighted the graph, it should now feature well connected groups of nodes with the same class labels. Connections between groups should be few and far between. Within each group, many nodes do not have a label. The objective of the third and last phase of graph-based learning is to classify the unlabeled records by propagating labels over the graph. Some authors use the term label propagation narrowly, referring to only specific variants of algorithms. There is no semantic consensus. We will use the term more broadly since all the following techniques propagate labels in some form over the graph. [11]

[11] On page 62 you find a table that enumerates all algorithms in this master thesis.



(a) Curvature of a 2D manifold's embedding in 3D space compared to the size of the graph neighborhoods (Y. Bengio, Delalleau, & Roux, 2006, p. 53)

(b) "Swiss roll" embedding The distance on the manifold is longer than in the embedding space (Tenenbaum, De Silva, & Langford, 2000, p. 2321)

Figure 2.3: Manifold embedding

Two widely accepted label propagation methods have already been combined with AEW: LLGC, which allows to change existing labels and HGF, which does not.

On the following pages, we briefly present multiple other techniques:

- GTAM provides even more ways to diverge from existing labels than LLGC.
- Graph Mincuts are of historic relevance, the method precedes HGF and then LLGC.
- Markov random walks can serve to propagate labels but have parameter tuning issues.
- MAD has a similar goal as GTAM, robustness, but a very different approach to it.
- Eigenvector based methods can take class proportions and misclassification costs into account while propagating, but are proposed only for the binary case.
- Manifold regularization is interesting with respect to our research question. It allows for out-of-sample predictions but is limited to binary classes.
- MP extends label propagation to scenarios where multiple labels per record are correct or where the supervision is performed by label probabilities and not hard labels.

We end this section introducing regularization techniques that address class- or cost-imbalance.

All variants use undirected graphs since the preceding sparsification and edge weighting methods do not produce directed graphs. If we wanted to use label propagation on an existing weighted directed graph (i.e. social networking data), we could use the technique from D. Zhou, Huang, and Schölkopf (2005). Karasuyama and Mamitsuka (2013b) review algorithms that propagate labels over multiple graphs, which represent multiple views of the same records, and proposes a method that detects the most useful graphs for classification among the lot. This is out of scope for this master thesis since AEW does not integrate well with pre-weighted graphs.

Learning with local- and global consistency (LLGC) D. Zhou et al. (2004) reflect the smoothness- and the manifold assumption. Propagating labels across a graph achieves local consistency if the label estimations (which are not binary predictions but label confidences stored in matrix $\mathbf{F}$) are similar for strongly connected nodes, this reflects the smoothness assumption. Global consistency is achieved when none or at most a few of the initially labeled nodes get assigned conflicting labels.

There is a trade-off between those two types of consistency. It is possible to accept some global inconsistency (to change existing labels) if this helps with the local smoothness. The possibility of relabeling nodes does not necessarily acknowledge that they were wrongly labeled in the first place, it can also mean that their labels are correct but unusual and will therefore be ignored in favor of smoothness. An unusually labeled node in this context is a node which does not respect the manifold assumption as it lies close to or within the manifold of another class.

Label smoothness can be measured as in equation (2.14). Karasuyama and Mamitsuka (2016) define the compromise between local and global consistency through objective function equation (2.15). $\| \quad \|_F$ denotes the matrix Frobenius norm defined as $\|\mathbf{M}\|_F = \left(\sum_{ij} M_{ij}^2\right)^{1/2}$. The first term of the objective penalizes label confidence differences between nodes that are connected. The weight of the penalty is proportional to the square difference in label confidences for each class and to the weight of the edge between the nodes. The second term penalizes label estimations that depart from the record label and is again proportional to the square difference. We optimize a continuous square loss function with two terms. $\mathbf{Y}$ is the binary label matrix (see p. 9). If no label is available for a record, all label estimations are penalized, which prevents overconfidence. At the same time, the smoothness term prevents degenerate solutions where $\mathbf{F}$ is zero for all classes of unlabeled records. Large values of parameter λ give precedence to global- over local consistency and vice versa.

$$\sum_{k=1}^{c}\sum_{i=1}^{n}\sum_{j\sim i} W_{ij}(F_{ik} - F_{jk})^2 = \text{trace}(\mathbf{F}^\top \mathbf{L}\mathbf{F}) \tag{2.14}$$

$$\mathbf{F} = \arg\min_{\mathbf{F}} \quad \text{trace}(\mathbf{F}^\top \mathbf{L}\mathbf{F}) + \lambda\|\mathbf{F} - \mathbf{Y}\|_F^2 \tag{2.15}$$

Knowing $\mathbf{F}$, classification decisions are stored in $\hat{\mathbf{Y}} \in \{0,1\}^{n\times c}$ according to equation (2.16):

$$\hat{Y}_{ik} = \begin{cases} 1 & \text{if} \quad k = \arg\max_{\kappa\leq c} \quad F_{i\kappa}, \\ 0 & \text{otherwise} \end{cases} \tag{2.16}$$

Equation (2.15) uses the graph Laplacian matrix $\mathbf{L}$ for undirected graphs without self-loops. There are several variants of this matrix. Karasuyama and Mamitsuka (2016) mention the graph Laplacian $\mathbf{L}$ in equation (2.17), a random walk normalized version in equation (2.18) and a symmetric normalized version in equation (2.19). The symmetric normalized version balances a graph which would otherwise systematically have larger edge weights in dense regions than in sparse regions of the input space $\mathbb{R}^p$. The degrees of all nodes are stored in a diagonal matrix $\mathbf{D} \in \mathbb{R}^{n\times n}$ such that $D_{ii} = \sum_{j=1}^{n} W_{ij} = \sum_{j\sim i} W_{ij}$, the sum of all edges between node i and connected nodes j. Since $\mathbf{D}$ is a diagonal matrix, $\mathbf{D}^{-1}$ is simply a diagonal matrix with the reciprocal elements of $\mathbf{D}$ and $\mathbf{D}^{-1/2}$ a diagonal matrix with the reciprocals of the square roots of elements of $\mathbf{D}$.

$$\mathbf{L} = \mathbf{D} - \mathbf{W} \tag{2.17}$$

$$\mathbf{L}^{rw} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W} = \mathbf{D}^{-1}\mathbf{L} \tag{2.18}$$

$$\mathbf{L}^{sym} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} \tag{2.19}$$

The graph Laplacian and the symmetric normalized Laplacian are the most used variants for classification. Equation (2.20) and (2.22) show their content in the case of a weighted adjacency matrix $\mathbf{W}$. If $\mathbf{W}$ has only positive weights, $\mathbf{L}$ is positive semidefinite, which makes equation (2.15) convex (Chung, 1997). For L^{sym}, we assume that there are no singleton nodes with degree 0. This is a reasonable assumption since label propagation would not work in this case and graph construction algorithms can prevent it. The sparse structure of the Laplacian simplifies computations. Laplacian sparsity allows to reduce equation (2.15) to a linear system, as described in Delalleau et al. (2005) for the binary case, which can be solved in nearly linear time by methods exposed in Spielman and Teng (2014).

$$L_{ij} = \begin{cases} D_{ij} & \text{if } i = j, \\ -W_{ij} & \text{if } j \in \mathcal{N}_i \text{ OR } i \in \mathcal{N}_j, \\ 0 & \text{otherwise} \end{cases} \quad (2.20) \qquad L^{sym}_{ij} = \begin{cases} 1 - \frac{W_{jj}}{D_{jj}} & \text{if } i = j, \\ -\frac{W_{ij}}{\sqrt{D_{ii}D_{jj}}} & \text{if } j \in \mathcal{N}_i \text{ OR } i \in \mathcal{N}_j, \\ 0 & \text{otherwise} \end{cases} \quad (2.22)$$

$$L^{rw}_{ij} = \begin{cases} 1 & \text{if } i = j, \\ -\frac{W_{ij}}{D_{ii}} & \text{if } j \in \mathcal{N}_i \text{ OR } i \in \mathcal{N}_j, \\ 0 & \text{otherwise} \end{cases} \quad (2.21) \qquad S_{ij} = \begin{cases} \frac{W_{ij}}{\sqrt{D_{ii}D_{jj}}} & \text{if } i = j \text{ OR } j \in \mathcal{N}_i \text{ OR } i \in \mathcal{N}_j, \\ 0 & \text{otherwise} \end{cases} \quad (2.23)$$

Alternatively, we can use the closed form solution proposed by D. Zhou et al. (2004). It uses a matrix $\mathbf{S} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$ which resembles the symmetric normalized Laplacian in that $\mathbf{L}^{sym} = \mathbf{I} - \mathbf{S}$ and is also sparse, see equation (2.23). $\mathbf{S}$ stems from spectral clustering where $\mathbf{L}^{sym}$ in its original form would lead to problems later in the algorithm (Ng, Jordan, Weiss, et al., 2002). [12] Two variants are proposed to compute $\mathbf{F}$: equation (2.24) and (2.25).

$$\mathbf{F} = \underbrace{(\mathbf{I} - \alpha\mathbf{S})^{-1}}_{\text{graph kernel}}\mathbf{Y} \quad (2.24) \qquad \mathbf{F} = (\mathbf{D} - \alpha\mathbf{W})^{-1}\mathbf{Y} \quad (2.25)$$

The recommended value for the regularization parameter $\alpha \in [0, 1]$ is 0.99 and corresponds to $\lambda = \frac{1}{\alpha} - 1 = 0.0101$. During the iterations which the closed form solution represents, equation (2.26), very much weight is given to the information a node receives from its neighbors (local consistency).

$$\mathbf{F}^{(t+1)} = \alpha\mathbf{S}\mathbf{F}^{(t)} + (1 - \alpha)\mathbf{Y} \quad (2.26)$$

Karasuyama and Mamitsuka (2016) show that matrix $\mathbf{F}$ can be expressed as the local averaging form equation (2.27) when the graph Laplacian $\mathbf{L}$ is used:

$$F_{ik} = \frac{\sum_{j \sim i} W_{ij} F_{jk}}{D_{ii} + \lambda} + \frac{\lambda Y_{ik}}{D_{ii} + \lambda} \qquad \forall k \in \{1, \ldots, c\} \quad \forall i \in \{1, \ldots, n\} \quad (2.27)$$

Computational complexity of LLGC should be at worst $\mathcal{O}(kn^2) = \mathcal{O}(n^2)$ since the number of neighbors k does not rise with n (Y. Bengio et al., 2006, p. 38). In both equation (2.24) and (2.25), only the last multiplication with $\mathbf{Y}$ involves labels. The costly inversion of the matrices in

[12] Taking $\mathbf{S}$ instead of $\mathbf{L}^{sym}$ only changes the eigenvalues from $1 - \lambda_i$ to λ_i. It does not affect the normalization. We suspect that it should not make a difference, in the context of label propagation, to take $\mathbf{L}^{sym}$. Y. Bengio et al. (2006, p. 38) even call $\mathbf{S}$ the normalized graph Laplacian. Yet, we will not temper with D. Zhou et al. (2004)'s recommendation to take $\mathbf{S}$.

the parentheses is unsupervised learning. The first part of the RHS of equation (2.24) yields a graph kernel or diffusion kernel (Smola & Kondor, 2003). This means that the way in which labels are propagated does not depend on the available labels. Practically, the computational complexity can be greatly reduced in scenarios where the label propagation is performed iteratively on a graph, changing only the available labels at each iteration.

Graph transduction via alternating minimization (GTAM) Jun Wang, Jebara, and Chang (2008) introduce a variant of LLGC that is useful when the available labels are known to contain mistakes. In our context, this may be relevant with iterative algorithms where estimated labels are reused as training examples for later iterations. It may not be sufficient or ideal to reluctantly allow labels to be reassigned if the gain in smoothness offsets the loss in global consistency in equation (2.15). If we know for a specific data-set, that labeled records which are located close to or in the manifold of another class are not just unusual outliers but encoding mistakes, we can allow our optimization to change those labels in the matrix $\mathbf{Y}$. Equation (2.28) optimizes $\mathbf{F}$ and $\mathbf{Y}$ simultaneously. The authors succeeded in making this mixed integer programming problem tractable via alternating minimization.

$$(\mathbf{F}, \mathbf{Y}) = \underset{\mathbf{F}\in\mathbb{R}^{n\times c}, \mathbf{Y}\in\{0,1\}^{n\times c}}{\arg\min} \quad \frac{1}{2}\operatorname{trace}\left(\mathbf{F}^\top \mathbf{L}\mathbf{F} + \lambda\left(\mathbf{F}-\mathbf{Y}\right)^\top\left(\mathbf{F}-\mathbf{Y}\right)\right) \tag{2.28}$$

Alternating minimization is an iterative optimization technique where a problem can be subdivided into two problems (here $\mathbf{F}$ and $\mathbf{Y}$) each of which are convex and converge towards a global optimum. Iterations alternate between both subproblems, each time using information from the last step on the respectively other subproblem. The combined problem would typically not have been convex (Niesen, Shah, & Wornell, 2009; P. Jain, Netrapalli, & Sanghavi, 2013).

Harmonic Gaussian fields (HGF) are introduced by Zhu, Ghahramani, Lafferty, et al. (2003) for usage in conjunction with Gaussian kernels weights, but can be used on other graphs as well. From equation (2.29), we see that the objective function optimizes label smoothness exactly as in equation (2.15) for LLGC, but does not weigh this smoothness against deviations from the initial labeling. Instead, labels are explicitly constrained to never depart from an initial label if such a label is available for a record.

$$\begin{aligned} \mathbf{F} = \arg\min_{\mathbf{F}} \quad & \sum_{k=1}^{c}\sum_{i=1}^{n}\sum_{j\sim i} W_{ij}(F_{ik}-F_{jk})^2 = \operatorname{trace}(\mathbf{F}^\top\mathbf{L}\mathbf{F}) \\ \text{s.t. } & F_{ik} = Y_{ik}, \qquad \forall k \in \{1,\ldots,c\} \quad \forall i \in \mathcal{L} \end{aligned} \tag{2.29}$$

The term harmonic indicates that for unlabeled records, matrix $\mathbf{F}$ can be expressed as the local averaging form equation (2.30) when the graph Laplacian $\mathbf{L}$ is used. No record differs from the average of its neighbors.

$$F_{ik} = \frac{\sum_{j\sim i} W_{ij}F_{jk}}{D_{ii}} \qquad \forall k \in \{1,\ldots,c\} \quad \forall i \in \mathcal{U} \tag{2.30}$$

HGF proposed for the first time in graph-based semi-supervised learning a matrix **F** which features label estimates on a continuum. Blum and Chawla (2001) had earlier optimized a similar objective function in the categorical scenario. The relaxation from categorical to continuous label estimates not only provides label confidences, but also allowed for faster computation since loopy belief propagation constitutes a closed form solution of equation (2.29) (Y. Weiss, 2000).

Graph Mincuts Blum and Chawla (2001) propose the following for binary classification. Multi-class extensions are NP-hard (Zhu, Ghahramani, Lafferty, et al., 2003, p. 4).

1. Construct a sparse symmetric weighted graph. Some nodes are labeled, some arenot
2. Add two nodes which are source nodes for the two classes
3. Connect all labeled nodes with infinite weight to their class source node
4. Find a set of edges that disconnect the two source nodes if dropped
   Minimize the sum of the weights in that set of edges to be dropped
5. Label unlabeled nodes according to the connected source node

The fourth step is NP-hard, heuristics are available (Boykov, Veksler, & Zabih, 2001).

Markov random walks Jaakkola and Szummer (2002) propose a Markov random walk approach to label propagation. Based on the probabilities of ending up in a node j when starting in node i and having made t steps on the graph, we can deduce the probabilities of having started in any node i given that we are in node j. This is assuming that each starting point is chosen with the same probability $1/n$. Each unlabeled node is assigned the label which maximizes the probability of having started the random walk in a labeled node of that class. The algorithm suffers from two issues: Firstly, the number of steps t is crucial to classification and yet difficult to tune. If t is too low, labels can only propagate in small spherical clusters and not across longitudinal manifolds. If t is too large, the potential starting points become indistinguishable and the class with the most available labels is almost always chosen. Secondly, we need to estimate an a priori probability of labels for each unlabeled point. Two approaches are proposed for this, both of which depend on available labels and cannot be expected to work well when those are scarce.

Modified adsorption (MAD) P. P. Talukdar and Crammer (2009) propose a novel way to use random walks for label propagation. At each node i encountered during the walk, one of three things happens:

- The walk continues to another node with probability p_i^{cont}
- The label of node i is injected to the starting node of the walker with probability p_i^{inj}
- The walk is abandoned, assigning a dummy label to the start node with probability p_i^{abdn}

p_i^{abdn} is high on unreliable nodes such as hub nodes with high degrees, making MAD robust to poorly constructed graphs. AEW achieves a similar robustness one step earlier in the process by reducing weights of edges connected to unreliable nodes. The mathematical details of MAD are out of scope here since there is not much need to combine it with AEW. The MAD objective function

is convex and can be solved in closed form. It contains a smoothness term and a global consistency term, similar to LLGC, and a third term regulating p_i^{abdn}.

Orbach and Crammer (2012) generalize the concept of recognizing unreliable nodes. The authors use an objective function very similar to P. P. Talukdar and Crammer (2009) and output for each node and class not only how confident the classifier is to label the node this way (that which can be found in $\mathbf{F}$ for other algorithms), but also a measure of variability of said confidences. Unreliable nodes are nodes where the confidences for all labels are close to uniform and/or nodes where confidences have high variability. This method uses 3 parameters that need tuning.

Eigenvector based methods Belkin and Niyogi (2002) propose an algorithm called Laplacian regularization that is based on eigenvector decomposition of the unnormalized Laplacian $\mathbf{L}$. The algorithm makes the assumption that the classification function is only defined within the manifolds. Parameter m needs to be tuned. Consider the binary case with labels $y = +1$ and $y = -1$:

1. Construct a sparse symmetric weighted graph. Some nodes are labeled, some are not
2. Compute m eigenvectors $\mathbf{e}_1, \ldots, \mathbf{e}_m \in \mathbb{R}^n$ of $\mathbf{L}$ corresponding to the m smallest eigenvalues
3. Compute coefficients $a_1, \ldots, a_m \in \mathbb{R}$ over all labeled nodes $i = 1, \ldots, l$ through equation (2.31)

$$\underset{a_1,\ldots,a_m}{\arg\min} \quad \sum_{i=1}^{l} \left(y_i - \sum_{\zeta=1}^{m} a_\zeta \mathbf{e}_\zeta(i) \right)^2 \tag{2.31}$$

4. Estimate labels for unlabeled records i through equation (2.32)

$$y_i = \text{sign} \left(\sum_{\zeta=1}^{m} a_\zeta \mathbf{e}_\zeta(i) \right) \tag{2.32}$$

Joachims et al. (2003) propose a similar method based on m eigenvectors $\mathbf{e}_2, \ldots, \mathbf{e}_{m+1}$ of $\mathbf{L}$ starting with the smallest eigenvalues. Eigenvalues are replaced by monotonically increasing values. The algorithm takes class proportions (estimated from labeled training data) and misclassification costs into consideration when minimizing a squared loss function.

Manifold regularization Manifold regularization unifies learning with kernels and graph-based label propagation by finding a kernel that is adapted to the geometry of the data distribution (Belkin, Niyogi, & Sindhwani, 2005, 2006; Belkin & Niyogi, 2008; Sindhwani, Niyogi, & Belkin, 2005). In the most general form, objective function equation (2.33) contains in the first term a loss function V over labeled data. The second term penalizes the norm of the function f and imposes smoothness on possible solutions. The third term should reflect the intrinsic structure of the data, labeled and unlabeled, and penalize departure from it. In equation (2.34), we have taken a quadratic loss function for the first term and the well known smoothness term from LLGC, HGF and others to enforce local consistency. This variant is called Laplacian regularized least squares, it extends RLS by incorporating unlabeled data. $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n)]^\top$.

$$f = \underset{f}{\arg\min} \quad \frac{1}{|\mathcal{L}|} \sum_{i \in \mathcal{L}} V(\mathbf{x}_i, Y_i, f) + \gamma_A \|f\|_K^2 + \gamma_I \|f\|_K^2 \tag{2.33}$$

$$= \underset{f}{\arg\min} \quad \frac{1}{|\mathcal{L}|} \sum_{i \in \mathcal{L}} (f(\mathbf{x}_i) - Y_i)^2 + \gamma_A \|f\|_K^2 + \gamma_I \mathbf{f}^\top \mathbf{L} \mathbf{f} \tag{2.34}$$

1. Construct a weighted undirected graph over labeled and unlabeled records
2. Chose a Kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ and construct the Gram matrix $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
3. Set parameters $\gamma_A \to 0$, $\gamma_I \to 0$ and $\gamma_I \gg \gamma_A$ for inductive extension of label propagation
4. Compute α according to a square loss function equation (2.35)
   $\mathbf{J}$ is a binary diagonal matrix that reflects in J_{ii} whether record i is labeled.

$$\alpha = \left(\mathbf{JK} + \gamma_A|\mathcal{L}|\mathbf{I} + \frac{\gamma_I|\mathcal{L}|}{(|\mathcal{L}| + |\mathcal{U}|)^2}\mathbf{LK}\right)^{-1}\mathbf{Y} \tag{2.35}$$

5. Use inductive output function equation (2.36) on unseen records $\mathbf{x}$

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x}) \tag{2.36}$$

Drawbacks of manifold regularization are the need to chose a kernel and perhaps tune its parameters, to invert a n^2 matrix in equation (2.35) and the fact that, being based on kernels, it is limited to binary classification. That last point is in contrast with most graph-based algorithms.

Measure propagation (MP) Amarnag Subramanya and Bilmes (2008, 2009, 2011) propose a method that uses Kullback-Leibler divergence instead of a square loss function to optimize global consistency and smoothness. $D_{KL}(P||Q) = \sum_i P_i \log \frac{P_i}{Q_i}$ measures the expected logarithmic difference between probabilities. We use a variant for discrete distributions since this expected difference of probabilities is measured on a discrete amount of labeled nodes. Each probability P or Q spans a c dimensional space since it represents label confidences for all available classes (for example, $\mathbf{F}_{i\cdot} \in \mathbb{R}^{1\times c}$ denotes a row of matrix $\mathbf{F}$). Thanks to the KL divergence, the known labels in the global consistency term can be expressed as soft probability distributions instead of binary values. This helps when there are multiple correct labels per record and in case the human experts have an educated guess, but no certainty regarding the label. Objective function equation (2.37) has a third term that rewards the sum of entropy of the optimized label confidences for all nodes i. It pulls towards a uniform distribution unless the other two terms outweigh it. This prevents degenerate solutions where all records receive the same label. λ and υ need to be tuned.

$$\min_{\mathbf{F}} \quad \sum_{i\in\mathcal{L}} D_{KL}(\mathbf{F}_{i\cdot}||\mathbf{Y}_{i\cdot}) + \lambda \sum_{i=1}^{n}\sum_{j\sim i} W_{ij} D_{KL}(\mathbf{F}_{i\cdot}||\mathbf{F}_{j\cdot}) + \upsilon \sum_{i=1}^{n}\sum_{k=1}^{c} F_{ik}\log F_{ik} \tag{2.37}$$

Equation (2.37) is convex and can be reformulated in a way that makes it solvable via alternating minimization between neighboring nodes. In addition, the authors propose a node reordering algorithm that helps reduce RAM requirements, improves parallelization and might be used with other label propagation techniques than MP. Thanks to these innovations, Amarnag Subramanya and Bilmes (2011) were able to propagate labels across a graph with more than 10^8 nodes.

Regularization to class- and cost-imbalance Class imbalance is usually only an issue when there is also cost imbalance. Minimizing the expected costs of misclassification allows us to ignore

accurate classification of classes that are in a small minority unless the costs of failing to identify a member of said minority class should be much higher than other misclassification costs. In semi-supervised classification, we are faced with another issue: Do we even know if we have class imbalance? The few available labels might not be representative of the class proportions in the rest of the data-set. If we can estimate the class proportions some other way, should we compare them to the results of label propagation and make adjustments? Joachims et al. (2003) integrate class- and cost imbalances in the optimization algorithm itself. When using any other algorithm, we can still make post-hoc adjustments.

Class mass normalization (CMN) assumes that the known class proportions should be reflected in the label propagation outcome (Zhu, Ghahramani, Lafferty, et al., 2003). The authors recognize that, absent specific domain knowledge, labeled training records are the way to know those proportions. Let's assume a prevalence of q_k for class k among labeled nodes: $\sum_{k=1}^{c} q_k = 1$. We define the mass of a class k after label propagation as $\sum_{i=1}^{n} F_{ik}$ and classify a record i as class k if its classification confidence times class prevalence divided by class mass is the highest among available labels (equation 2.38).

$$\hat{Y}_{ik} = \begin{cases} 1 & \text{if } \; k = \arg\max_{\kappa \leq c} \quad F_{i\kappa} \times q_\kappa \Big/ \sum_{\eta=1}^{n} F_{\eta\kappa} \\ 0 & \text{otherwise} \end{cases}, \tag{2.38}$$

The classification confidences provided by LLGC or HGF can also be used in case of cost imbalances in the same way as with supervised algorithms. In the binary case, we can simply adjust the classification threshold to make the classifier more sensitive to the class with higher false negative costs. Other adjustments that use weights to prioritize records or that under- or over-sample specific classes are not easy to use in semi-supervised learning. Since we have only few labeled training examples, it may not be wise to under-sample any of them and since we do not know the true class proportions, we cannot know which weights to assign to records (Kuhn & Johnson, 2013, ch. 16).

Jun Wang et al. (2008) propose to replace the label matrix $\mathbf{Y}$ by a normalized version $\mathbf{Z} = \mathbf{VY} \in \mathbb{R}^{n \times c}$ for GTAM in equation (2.28). $\mathbf{V}$ is a diagonal matrix with elements computed as in equation (2.39). Since $\sum_{i=1}^{n} Z_{ik} = 1 \; \forall k \in \{1, \ldots, c\}$, the total diffusion of each class from its labeled nodes outwards is normed to unity. Within each class however, some nodes can diffuse their labels better than others as it should be. Like CMN, this prevents single classes from dominating the label propagation. Instead of assuming that the labeled records accurately reflect the classes, we now assume that each class should have similar possibilities to diffuse its labels. This is still an assumption. The authors show that it yields better results on some data-sets than no normalization or CMN. There is no reason to believe that $\mathbf{Z}$ can only be used in equation (2.28), it may also replace $\mathbf{Y}$ for LLGC in equation (2.15) or the corresponding closed forms and for HGF in equation (2.29).

$$V_{ii} = \sum_{k=1}^{c} \left\{ \left(Y_{ik} \sum_{j \sim i} W_{ij} \right) \Big/ \left(\sum_{\eta=1}^{n} Y_{\eta k} \sum_{\zeta \sim \eta} W_{\eta\zeta} \right) \right\} \quad \text{and} \quad V_{ij} = 0 \quad \forall i \; \forall j \neq i \tag{2.39}$$

2.3 Classification performance metrics

Having reviewed relevant techniques along the three steps of graph-based learning, we now examine performance metrics that are potentially relevant for our use-case. Our experimental setup as theoretically derived in section 2.4 and later practically described in section 4.3 influences our choice regarding performance metrics. For an exhaustive and formal comparative analysis of performance metrics, see Sokolova and Lapalme (2009), Sokolova, Japkowicz, and Szpakowicz (2006). A general overview of the most used techniques can be found in Han et al. (2011, ch. 8.5), Zaki and Meira (2014, ch. 22) and Kuhn and Johnson (2013, ch. 11).

2.3.1 Binary classes

When dealing with binary classes, the choice of performance metrics is much larger than in the general case with c classes. It is seldom the best option to simply use classification accuracy or error rate. Four classification outcomes are possible, as is represented in table 2.1:

	Predicted class	
True class	**1**	**0**
1	True Positive (TP)	False negative (FN)
0	False Positive (FP)	True Negative (TN)

Table 2.1: Binary confusion matrix, compares predicted class label with the true label The four possible outcomes have names which are used in other performance metrics

The *sensitivity* in equation (2.40), also called *true positive rate* (TPR), refers to the proportion of positive observations that are correctly identified, the ratio between the number of positive observations correctly identified (TP) and the total number of positive observations (P=TP+FN)

The *specificity* in equation (2.41), also called *true-negative rate* (TNR), refers to the proportion of negative observations that are correctly identified as negative, the ratio between the number of negative observations correctly identified (TN) and the total number of negatives (N=TN+FP).

$$\text{sensitivity} = \text{TPR} = \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2.40)$$

$$\text{specificity} = \text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}} \quad (2.41)$$

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2.42)$$

$$\text{F} = \frac{2 \times \text{prec.} \times \text{rec.}}{\text{prec.} + \text{rec.}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} \quad (2.43)$$

There is typically a trade-off between sensitivity and specificity as well as between precision and recall. Instead of asking the user which criteria is more important, some performance metrics integrate both measures. The F-score in equation (2.43) is the harmonic mean between precision and recall. ROC curves provide another integrated performance measure.

ROC When a classifier ranks classification confidence, i.e. estimated probabilities of records to belong to the positive and negative class, one can use receiver operating characteristic curves.

> "Advantages [of using ROC] include decision boundary adaptation to imbalanced misallocation costs, the ability to fix some classification errors, and performance evaluation

in imprecise, ill-defined conditions where costs, or prior probabilities may vary."
(Landgrebe & Duin, 2007)

In order to plot a ROC curve, one performs a sweep over classification thresholds. We start at the bottom right corner where we classify all observations as negative by setting the threshold for the positive class at 1. All the TN are correctly identified (a specificity of 1) but none of the TP are (a sensitivity of 0). We gradually decrease the threshold and hope that the sensitivity will increase (much) faster than the specificity decreases which would bring us closer to the upper left corner. Finally, we end with a threshold of 0 where all observations are deemed positive (a specificity of 0 and a sensitivity of 1) (Fawcett, 2006).

A random classifier has a ROC curve like a straight diagonal line from the bottom left to the upper right. We expect a classifier to perform better than a random model and the ROC curve should lie above the diagonal. The accuracy of a given model is measured by the area under the curve (AUC). If one threshold exists which separates classes perfectly, AUC will be 1.

2.3.2 Multiple classes

When dealing with multiple classes, the range of available performance metrics is smaller. Since the graph-based learning algorithms that we are interested in natively support multiple classes, we need to find an adequate performance metric for that situation. To begin with, we define a confusion matrix $\mathbf{C} \in \mathbb{N}^{c \times c}$ as a generalization of table 2.1. We can chose between ROC, κ and F-measure:

ROC It is possible to extend the usage of binary performance metrics like ROC AUC to c classes (Landgrebe & Duin, 2007). This is done in the ways in which one can also extend any binary classifier to multiple classes: one versus all (OvA) and one versus one (OvO) classification (Trevor Hastie, Tibshirani, et al., 1998). In the one-versus-all scenario, there need to be c classifications of records into "class c" and "not class c". In the one-versus-one scenario, $\frac{c \times (c-1)}{2}$ binary classifications need to be made. Afterwards, there needs to be a voting or averaging procedure to aggregate those results (Tax & Duin, 2002). Since the most relevant classifier used for this research, label propagation, is per default capable of distinguishing multiple classes, we want to avoid further complicating our experimental setup just because the performance metric makes it necessary to train multiple classifiers.

Cohen's κ statistic has been developed to rate the agreement between two raters (Cohen, 1960). In classification, those two raters are the predicted- and the true class distribution of records. The term O in equation (2.44) makes reference to the observed agreement between raters, the classification accuracy. E reflects a baseline called expected agreement or no-information rate. In a data-set with c classes, expected agreement could be set to $1/c$, the expected agreement of a random classifier when all classes are equally probable. In this case, κ would simply be proportional to O. If there is severe class imbalance, it would be better to define E as the proportion of the most prevalent class in the data-set. This level of agreement can always be achieved by classifying all records as this most prevalent class. Cohen (1960)'s definition of κ is based on the marginal totals of the confusion matrix, see equation (2.44).

$$\kappa = \frac{\mathrm{O} - \mathrm{E}}{1 - \mathrm{E}} \in [-1, 1] \quad \text{where}$$
$$\mathrm{O} = \frac{1}{n} \sum_{\gamma=1}^{c} C_{\gamma\gamma} \qquad \text{and} \qquad \mathrm{E} = \frac{1}{n^2} \sum_{\gamma=1}^{c} \left\{ \sum_{\Gamma=1}^{c} C_{\gamma\Gamma} \times \sum_{\Gamma=1}^{c} C_{\Gamma\gamma} \right\} \tag{2.44}$$

κ measures which proportion of the possible agreement above expected random agreement a classifier has reached. Negative values of κ indicate a classifier that actively seeks to make mistakes and are comparable to AUC values < 0.5 which are equally rare. $\kappa = 0$ indicates a random classifier. Starting at $\kappa > 0.3$, we can speak of reasonable agreement (Kuhn & Johnson, 2013, p. 256).

Fatourechi et al. (2008) conclude that κ is a more robust and versatile performance metric than accuracy. κ does not make assumptions of priority among sensitivity or specificity and precision or recall. Unlike ROC AUC, κ does not rely on multiple confusion matrices, it is simpler to compute.

However, κ does not lend itself easily to experimental setups in semi-supervised learning. The key parameter E should not be estimated over training-set(s) since the few available labeled records do not necessarily reflect a-priori class probabilities. It is also unclear whether it is advisable to compute multiple estimates of κ, one per test-set, and to perform statistical tests on them in the way that it can be done with accuracy and ROC AUC measures.

F-measure The multi-class F-measure can be computed from a single confusion matrix. We extend equation (2.43) to compute F-measures with regard to each class. For example, a false positive with regard to a certain class γ is defined as a record classified as γ when the true class would have been another (no matter which other one). Equation (2.45) shows the exact computations. We can then compute c different F-measures with regard to the classes and average them as in equation (2.46). Y. Yang and Liu (1999) call this macro-averaging because it averages F-measures over classes and not single records as it is the case with micro-averaging. Macro-averaging considers each class as equally important, even if it contains fewer records than other classes.

$$\mathrm{TP}_\gamma = C_{\gamma\gamma} \qquad \mathrm{FP}_\gamma = \textstyle\sum_{\Gamma=1}^{c} C_{\Gamma\gamma} - \mathrm{TP}_\gamma \qquad \mathrm{FN}_\gamma = \textstyle\sum_{\Gamma=1}^{c} C_{\gamma\Gamma} - \mathrm{TP}_\gamma \qquad \forall \gamma \in 1, \ldots, c \tag{2.45}$$

$$\mathrm{F} = \frac{1}{c} \sum_{\gamma=1}^{c} \frac{2\mathrm{TP}_\gamma}{2\mathrm{TP}_\gamma + \mathrm{FP}_\gamma + \mathrm{FN}_\gamma} \tag{2.46}$$

These F-measures are the most viable multi-class performance metric for our experiments.

2.4 Experimental setup

The objective of our experimental setup is to compare algorithms. Data-sets are a means to this end. This is very different from other studies where the objective is to find the best classifier for a given data-set. In this latter case, much of the emphasis would be put on fine-tuning the classifiers to the data-set, which we tend to avoid. Comparing algorithms across data-sets requires more sophisticated statistical techniques according to Salzberg (1999). The semi-supervised nature of our algorithms adds further complications, prohibiting the usage of some established techniques.

Data-set handling In order to make statistically significant pronouncements about comparative classification performance of algorithms, we need multiple estimates of performance for each algorithm. Each of those estimates is based on a different split of the data into training- and test records. Pairwise t-tests can then assess whether the performance difference between a pair of algorithms is statistically significant. There are three main methodologies of data-set handling to gain the required estimates: (Kuhn & Johnson, 2013, ch. 4.4)

- repeated training/test splits
- cross validation
- bootstrapping

Bootstrapping uses sampling of training- and test-sets with repetition. It does not combine well with data-sets where training records are subdivided into labeled and unlabeled ones. The popular 632(+) variants include performance on labeled training records into their assessment. This does not integrate with transductive algorithms and unlabeled data (Efron, 1983; Efron & Tibshirani, 1997). There is thus no need to look further into this otherwise interesting technique.

Repeated training/test splits Intuitively, we might want to split the available data repeatedly into a training-, and a test-set in order to gain a series of performance measurements from test-sets.

This simple approach has statistical flaws. Any kind of t-test requires i.i.d. samples, which is not a given in machine learning applications. Repeated training/test splits yield significant overlap regarding the data points used for training and testing. One sample point in the t-test corresponds to the test-set performance measure of one training/test split. This lack of sample independence can inflate Type I errors, exceeding the accepted level α. Especially when underlying classification error rates are high, t-tests based on repeated training/test splits will falsely assert performance differences between algorithms (Dietterich, 1998, sec. 3.3).

Cross-validation (CV) is a common approach to compute the expected value and variance of a given performance measure. As defined in Zaki and Meira (2014, p. 562):

> "K-fold CV divides the data set into K equal-sized parts, called folds. Each fold is, in turn, treated as the test-set, with the remaining folds comprising the training-set."

CV suffers from overlap between samples regarding the data points used for training, but not for testing. When the number of folds is quite large ($K > 20$), the bias of the estimator is small but the variance large since the test-set is small. On the other hand, too small values of $2 < K < 5$ also increase the variance because the composition of the training set is unstable. $K = 10$ is accepted as a good compromise (Kohavi et al., 1995). Unfortunately, 10 samples are generally not sufficient for a t-test. Because a normal distribution of the sample measures of classification performance cannot be assumed, the central limit theorem requires at least 30 sample points for their sample mean to be normally distributed (Wackerly, Mendenhall, & Scheaffer, 2007). This number of samples can be obtained by repeating the 10-fold CV 3 times. Repetitions introduce dependence regarding the test-sets which a simple CV does not have.

CV integrates well with inductive semi-supervised learning. The out-of-sample test-sets are defined by the rotating folds and within each training-set, there can be labeled and unlabeled records.

Thanks to the central limit theorem, we do not require a specific distribution (normal or otherwise) of our sample points of test-set performance measures. Not only accuracy, but also ROC AUC and F-measures can be statistically compared over cross validations. Forman and Scholz (2010) do warn that ROC and F-measures can be undefined on certain folds where there are no positive records. This biases the results but does only happen with strong class imbalance, small data-sets or too many folds. This problem can be avoided through stratified sampling, which we shall investigate now.

Stratified sampling of data points means in general to impose constraints as to what constitutes a valid sample. For instance, there may be sub-populations (strata) all of which need to be adequately represented in the sample(s) (Cochran, 1977). Stratification becomes necessary when purely random sampling is at an increased risk of not attaining this adequate representation of strata. In machine learning, this is the case when some values of an attribute are very rare. For example, with strong class imbalance, we should make sure that the rarest (and often most interesting) class is present in the training-set as well as in the test-set. Otherwise, we could not assess whether an algorithm can learn to detect that class. Another reason to use stratified sampling arises with sparsely labeled data. We need at least one example of each class label in each possible training-set in order to be able to propagate this label. Constraints imposed on a stratified sample should be held to a minimum such that the sample remains close to random. When too many elements of the samples are constrained, overlap between samples increases which biases Type I errors of statistical tests run on those samples upwards (Dietterich, 1998). The larger the number of observations in a data-set is compared to the number of constraints, the more probable it becomes that a randomly selected sample satisfies the constraints. In such cases, it is possible to obtain a stratified sample by reshuffling the sample only a few times until the constraints are met. In other cases, manual interference with the data-set composition might be necessary.

To avoid the aforementioned issue with non i.i.d. samples, non-parametric methods like McNemar's test on comparative confusion matrices can be used instead of t-tests (Dietterich, 1998). This approach is implicitly limited to accuracy/error rate as performance metrics. McNemar's test counts the number of times each of both investigated algorithms made a classification error where the other one did not and compares it with the expected equal number of errors under the null hypothesis of equal performance. This procedure cannot deal with class- and cost-imbalance like t-tests on ROC AUC or F-measures can.

Comparing algorithms with t-tests Until now, we have found out that our needs are best served by the performance metric ROC AUC for binary classes and F-measure for multiple classes and repeated (slightly stratified) cross validation as experimental setup. In the following, we shall investigate which statistical tests we can perform on the thusly generated data. Numerous variants of t-tests have been proposed for different use-cases including classification. We are mostly interested in the three following characteristics:

Paired samples All three presented methodologies for data-set handling can create so-called paired samples. Paired samples happen when re-using each particular data-set split for each algorithm that has to be compared. Instead of testing one algorithm on x data-set splits and another algorithm on y different data-set splits, all algorithms are tested on the same number of identical splits. A special variant of t-tests exists for such cases: Differences are computed on a sample-point per sample-point basis and then a single sample t-test is performed. [13] With paired samples, performance variation due to different data-set handling is eliminated and classification performance is isolated as the source of variation. When repeated measures of sampled data-splits are positively correlated, which is almost always the case, this increases the t-tests' ability to detect existing differences, their power. This difference in power stems from the way the sample variance in the denominator of the test statistic is computed (over precomputed sample-point differences). Computing the mean over differences instead of the difference between two means in the numerator of the test statistic yields the same result. On the other hand, the power of paired sample t-tests decreases because of lower degrees of freedom ($n-1$ instead of $2n-2$ if there are n sample points). According to Zimmerman (1997), this disadvantage is small and routinely outweighed. We should use paired sample tests with paired experimental setups.

Corrected resampled t-tests Nadeau and Bengio (2003) developed a corrected resampled (also called corrected repeated) t-test for CV. Let's assume we have a K-fold CV repeated R times such that $K \times R \geq 30$. First, we compute $d_{fr} \quad \forall f \in \{1, \ldots, K\} \; \forall r \in \{1, \ldots, R\}$, the performance difference of both algorithms on each paired sample-point corresponding to the folds and repetitions. We want to test if those differences are significantly different from zero. The sample mean and variance are computed the usual way:

$$\hat{\mu}_d = \frac{1}{K \times R} \sum_{f=1}^{K} \sum_{r=1}^{R} d_{fr} \qquad (2.47) \qquad \hat{\sigma}_d^2 = \frac{1}{(K \times R) - 1} \sum_{f=1}^{K} \sum_{r=1}^{R} (d_{fr} - \hat{\mu})^2 \qquad (2.48)$$

Nadeau and Bengio (2003) propose a correction that takes care of the Type I error inflation with CV t-tests, especially for high values of K and R. The following adjusted test statistic should be compared with regular student tables for $(K \times R) - 1$ degrees of freedom:

$$t = \hat{\mu}_d \Bigg/ \sqrt{\left(\frac{1}{K \times R} + \frac{1/K}{1 - 1/K} \right) \hat{\sigma}_d^2} \qquad \text{instead of the usual} \qquad t = \hat{\mu}_d \Bigg/ \sqrt{\frac{\hat{\sigma}_d^2}{K \times R}} \qquad (2.49)$$

According to Bouckaert and Frank (2004), this test is not only more powerful, but also more repeatable than those from Dietterich (1998): When the same data-set is handled in the same way to compare identical algorithms, only the random assignment of single records to the folds can lead to different outcomes. The corrected resampled t-test is less vulnerable to this kind of instability, more repeatable. We will use corrected resampled t-tests with our CV.

[13] Attention: The wording *pairwise* t-test makes reference to the two-by-two pairing of in our case algorithms of which there are multiple (that which is called "treatments" in the medical literature from which these statistical methods stem) whereas *paired sample* t-test makes reference to pairing sampled data-set splits (individuals) which are repeatedly measured across algorithms (treatments). We will use both concepts combined.

Familly Wise Error Rate (FWER) The fact that $\binom{\aleph}{2} = \frac{\aleph!}{2!(\aleph-2)!} = \frac{\aleph \times (\aleph-1)}{2} = m$ pairwise combinations of $\aleph$ algorithms exist makes for a complication. If those m pairwise tests were done independently with the same confidence level $1-\alpha$ each, the confidence level of the combined hypothesis that none of them has a Type I error would be $(1-\alpha)^m \ll (1-\alpha)$. The Bonferroni correction makes sure the FWER of m tests does not exceed α, allocating $[1-(1-\alpha)]^{1/m} \approx \frac{\alpha}{m}$ per hypothesis (Dunn, 1961).

Holm's method is a more powerful procedure to detect existing differences while controlling for FWER (Holm, 1979). We start by ordering the m hypotheses in order of ascending p-values. Consider the k^{th} hypothesis in that series, the first for which p-value $> \alpha/(m+1-k)$. All null hypotheses up to k not included are rejected, null hypotheses k to m cannot be rejected.

Comparing algorithms across data-sets Most papers in machine learning compare algorithms based on multiple data-sets. Yet those classification performances are seldom aggregated with a rigorous statistical test. Instead, informal statements about performance on several data-sets are the norm. To help with this issue, Demšar (2006) proposes multiple methods of comparing classifier performance across data-sets and was further extended by Garcia and Herrera (2008) to better accommodate multiple classification algorithms.

ANOVA Paired tests of multiple algorithms on each of multiple data-sets correspond to the experimental setup of a two way mixed ANOVA (much used in medicine and social sciences). In our case, the algorithmic factor is measured within subjects whereas the data-set factor is measured between subjects. Unfortunately, the sphericity assumption, stating that variances of the differences between the groups of the within-subjects factor must all be equal, is routinely violated by machine learning data (Demšar, 2006). This renders the results of the ANOVA unreliable. In any case, we would still have needed post-hoc tests like those from Tukey (1949) to obtain more granular results than the ANOVA itself. Such post-hoc tests are t-tests and not formally linked to the ANOVA.

Sign test Intuitively, we might want to count the data-sets on which an algorithm won or lost over another. Under the null hypothesis of equal performance, both algorithms should win $D/2$ points out of D data-sets. This simple counting allows for the so called sign-test (Demšar, 2006). When D is large, equal performance would yield an expected normal distribution of points with mean $D/2$ and standard deviation $\sqrt{D}/2$. This allows to construct confidence intervals and test at a given level of α. [14] When multiple algorithms are to be compared pairwise, a Bonferroni or Holm adjustment needs to be made. When D is smaller than 30, computations should be based on the binomial distribution (Salzberg, 1997). Note that we should count all wins and losses and not only statistically significant wins and losses according to previous t-tests. Otherwise, we would be stacking two significance tests. The combined test would be too conservative and have very low power. Since the sign test only counts wins and losses on data-sets, it does not matter if some data-sets use different performance metrics than others (depending on whether $c > 2$).

Wilcoxon's signed rank test is another way of comparing performance across data-sets without assuming any specific sample distribution (Wilcoxon, 1945). It is more powerful than the sign

[14] This approach is very similar to McNemar's test. The only difference is that we are comparing over data-sets instead of data-points. Our reservation of not being able to use adequate performance metrics would apply here.

test since it ranks performance differences instead of just counting them. This test can only be used when all data-sets share the same performance metric. Otherwise, performance differences according to one metric might be systematically larger/smaller than according to another metric which would bias the ranking. A paired t-test across data-sets would be even more powerful than Wilcoxon's signed rank test, but the assumption of i.i.d. samples is clearly violated when those samples come from different data-sets. [15]

In addition to this quantitative aggregation, a more qualitative analysis of the results on different data-sets is also of interest. It is here that the p-values of the pairwise tests on single data-sets, which have been ignored when aggregating over data-sets, come into play again.

- Does a classifier perform particularly well/badly on specific types of data-sets?
  (data with class imbalance, high dimensionality, large number of classes, noisy data, ...)
- Does a given classifier win/loose by narrow or by wide margins on a data set?
- Does a classifier have consistent performance across data-sets or does it show large variability?

From the selection of performance measures to the comparison across data-sets have we found ways to address the known shortcomings of the "de-facto standard" approach to comparing machine learning algorithms as described in Japkowicz and Shah (2011, ch. 1.3):

- The relevance of performance metrics is sometimes lacking. We have found ROC AUC and multi-class F-measure which make no unreasonable assumptions with regard to our semi-supervised setup and which integrate specificity and sensitivity or precision and recall.
- Sample points are not independent and identically distributed.
  We address this issue with corrected resampled t-tests.
- Type I errors inflate in multiple comparisons.
  We use Holm's procedure to address this.
- Aggregation/averaging of the results is attempted with binomial sign tests

In case of doubt, we prefer to be conservative in our choice of methods since we want to avoid inflated type I errors (i.e. corrected resampled t-tests). Among the statistically valid methods, we chose the most powerful (i.e. Holm's method over the Bonferroni correction).

Chapter 4 describes the concrete choices of algorithms, parameters and data-sets with which this methodology will be used. Before we come to that, let us in the following chapter 3 review in detail the novel technique of adaptive edge weighting as well as the inductive extensions of transductive learning which are at the center of this master thesis.

[15] The main issue here is the identical distribution of samples. The issue with CV was sample independence.

Chapter 3

New techniques

We consider two new approaches in this master thesis: adaptive edge weighting and inductive extensions to transductive algorithms. Adaptive edge weighting adds a viable method to the second step of graph-based learning. Alternatives are much more needed for that second step than for the previous- and the subsequent one where many variants exist.

Inductive extensions to graph-based algorithms would make it possible to use them in scenarios where they cannot realistically be used so long as they remain transductive. This is notably the case where new cases arrive frequently and need to be classified right away.

We analyze two main approaches to inductive extension. Firstly, a recently proposed solution which achieves inductive extension by combining the graph-based algorithm with one or more other algorithms. Secondly, an older solution which makes some simplifying assumptions that allow for inductive prediction. This second solution gains new relevance since it is compatible with AEW but not with the competing LLE. AEW is linked to our research question regarding inductive extension.

3.1 Adaptive edge weighting

For the rest of this document, we assume a graph which is sparse, binary and symmetric has been constructed. A few remarks regarding robustness to violation of these assumptions:

- AEW can be used on fully connected graphs. The weights of unwanted edges connecting very dissimilar nodes would be optimized to very low values. This would mean to start with the second step of the three step procedure of graph-based learning, but would be a waste of computation time. The efficiency of AEW depends on graph sparsity. Once a connection has been sparsified away, it cannot be brought back. Only non-zero weights W_{ij} will be considered.
- AEW can be used on graphs with pre-optimized edge weights. This would be a waste of computation time since the initial edge weights would not be taken into consideration by the AEW optimization, but simply overwritten. For this reason, it may not make much sense to use AEW on data that comes already in the form of a weighted graph.
- If the graph was undirected, as is usually the case, it will stay so after the AEW. AEW might also be used with directed graphs, reconstructing each node from only either outgoing- or incoming neighbors (the second case seems more logical). D. Zhou et al. (2005) can then be used to propagate labels on such a graph.

3.1.1 Parametrized edge weights

As the name indicates, AEW optimizes edge weights adaptively through optimization parameters σ_d in equation (3.1). The numerators of the fractions represent node dissimilarity. x_{id} is the value of record i variable d. When the numerators in the sum are large relative to the denominators, the edge weight will be small because the nodes are dissimilar. Even though there is exactly one parameter σ_d per variable, this does *not* represent standard deviations but rather a kernel bandwidth as in equation (2.10).

$$W_{ij} \leftarrow W_{ij} \times \exp\left(-\sum_{d=1}^{p} \frac{(x_{id} - x_{jd})^2}{\sigma_d^2}\right) \tag{3.1}$$

Before we go on to optimize σ_d, we shall mention two issues with the edge weights function:

Local scaling Zelnik-Manor and Perona (2004) introduce the local scaling kernel. Sample points are seldom evenly distributed. They form dense and less dense neighborhoods. The edge weights in equation (3.1) do not reflect this. Nodes in dense regions will systematically have stronger connections because their dissimilarities (the numerators) are smaller. If we want to propagate labels through neighborhoods of varying density, we should scale the density out of our edge weight equation to gain a more balanced graph. Equation (3.2) introduces estimates of the density around i and j into the edge weight W_{ij}. These densities s_i can for example be estimated by node i's dissimilarity with its k^{th} nearest neighbor (Zelnik-Manor & Perona, 2004) or else by the average dissimilarity between i and its k nearest neighbors (Jegou, Harzallah, & Schmid, 2007). Even though s_i represent densities around nodes on the graph, we are not computing shortest path distances on the graph itself but rather distances between the nodes as represented in the input space $\mathbb{R}^p$. However, if the graph accurately represents the manifold and if the neighborhoods are small enough compared to a possible curvature of the manifold, those two measures should not differ too much. For robustness, we add a small value to all s_i. It avoids division by zero in equation (3.1) in case there are more than k identical records in the data-set.

Karasuyama and Mamitsuka (2016) found that classification performance is in most cases better with the local scaling kernel. We will use equation (3.2) for edge weighting. The same study also found that using the local scaling kernel in combination with a normalized symmetric graph Laplacian gives slightly worse results than using only the local scaling kernel. Both adjustments can, but do not have to be combined since they perform a similar normalization of distances. Local scaling balances edge weights which leaves less normalization for the symmetric graph Laplacian $\mathbf{L}^{sym}$.

$$W_{ij} \leftarrow W_{ij} \times \exp\left(-\sum_{d=1}^{p} \frac{(x_{id} - x_{jd})^2}{s_i s_j \sigma_d^2}\right) \tag{3.2}$$

Distance metric After having been used in graph sparsification (section 2.2.1), dissimilarity of nodes continues to play a crucial role for AEW. For consistency reasons and in order to deal with heterogeneous data, the same distance or dissimilarity metric that has previously been used to sparsify the graph should continue to be used for the edge weights in equation 3.1 or 3.2.

This can pose an issue with for example Gower (1971)'s metric. We cannot take the overall

dissimilarity measure and divide it by $\sum_{d=1}^{p} s_i s_j \sigma_d^2$ because a sum of ratios is not equal to a ratio of sums. In other words, we need the variable wise elements which constitute our dissimilarity measures. We can compute square differences on the scaled feature vectors. This is coherent with Gower's dissimilarity measure as long as ordinal, interval, ratio or symmetric binary variables are concerned. Nominal variables could be transformed into sets of symmetric binary dummy variables. Simply computing euclidean distances on their scaling to $[0, 1]$ does otherwise not reflect Gower's dissimilariy measure. Asymmetric binary variables make the number of dimensions p vary from record to record which is not reflected by scaled variables. [1]

A compromise can be made: The variable wise distance metrics used in AEW may not perfectly reflect the dissimilarity metric used to sparsify the graph. It is better to use the most appropriate dissimilarity measure for sparsification and then the best available variable wise distance measure for AEW than to use the latter for both steps.

AEW is not in principle limited to the usage of Gaussian kernels as presented here. We could use a different kernel function if the type of data we use makes it necessary. Provided a kernel function has a number of parameters that can serve as decision variables for the AEW optimization, it might be used. The number of optimization parameters should reflect the dimensionality of the data p and therefore neither grow with the number of records n, nor be a small constant. Other kernels are seldom used for purely similarity based edge weighting and have, to our knowledge, never been combined with AEW. Let's see how those parameters can be optimized.

3.1.2 Optimization

Objective function 3.3 is used to determine the optimal parameters $\sigma = \{\sigma_1, \ldots, \sigma_p\}$. Note that it does not use labels at all. The second step in the three step procedure of graph-based learning is thus still unsupervised. Only the third step will be semi-supervised. Equation (3.3) measures how well the nodes i can be described as linear combinations of their graph neighbors $j \sim i$, "how well the graph fits the input features manifold" (Karasuyama & Mamitsuka, 2016). As with LLE, we minimize the sum of square errors committed when approximating each node $\mathbf{x}_i$ as a linear combination of its neighbors. This approximation is noted $\hat{\mathbf{x}}_i$ and called a local linear patch (llp). We use edge weights as the coefficients of the llp. We need to divide by the node degree since $\sum_{j\sim i} W_{ij} \neq 1$.

The edge weights depend on $\{\sigma_1, \ldots, \sigma_p\}$ which change at every iteration of the optimization, as do the node degrees. We need to recompute them every time and resubstitute them into equation (3.3).

$$\begin{aligned}
\min_{\sigma} \text{obj}(\sigma) = \min_{\sigma} \quad \sum_{i=1}^{n} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2 \quad &= \min_{\{\sigma_1,\ldots,\sigma_p\}} \quad \sum_{i=1}^{n} \left\| \mathbf{x}_i - \frac{1}{\sum_{j\sim i} W_{ij}} \sum_{j\sim i} W_{ij}\mathbf{x}_j \right\|_2^2 \\
&= \min_{\{\sigma_1,\ldots,\sigma_p\}} \quad \sum_{i=1}^{n} \left\| \mathbf{x}_i - \frac{1}{\sum_{j\sim i} \exp\left(-\sum_{d=1}^{p} \frac{(x_{id}-x_{jd})^2}{s_i s_j \sigma_d^2}\right)} \sum_{j\sim i} \exp\left(-\sum_{d=1}^{p} \frac{(x_{id}-x_{jd})^2}{s_i s_j \sigma_d^2}\right) \mathbf{x}_j \right\|_2^2
\end{aligned} \tag{3.3}$$

[1] Solving this last issue is not easy to implement since Verboven and Hubert (2010) do not grant access to the elements which constitute Gower's measure. Short of re-implementing Gower's metric, we can only use the standardized variables for euclidean distance computations.

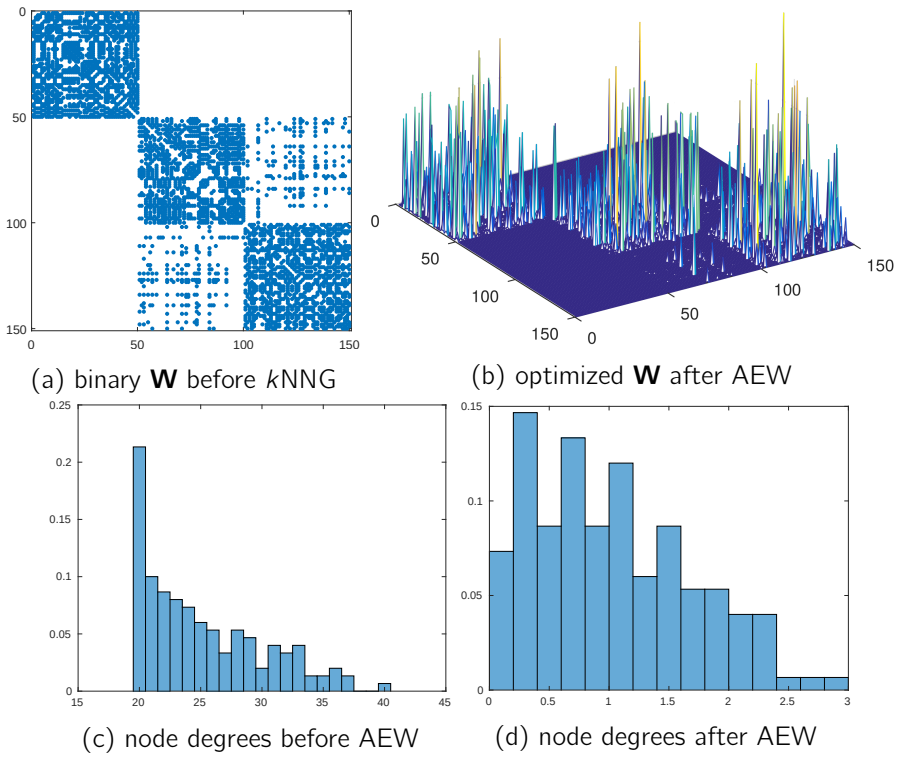
When the graph sparsification was done well and $k \ll n$, the optimization can decrease the objective function only moderately. With some data-sets, no less than 70% of the intial value in obj(σ_{start}) are achievable. With fully connected graphs however, the optimized value will be only $1 - 2\%$ of the starting value. The optimized values of the objective function are similar in both cases which shows that AEW can compensate a poor graph sparsification. Conversely, if graph quality is measured by the objective function, this quality can be achieved to a good extent by binary sparsification. If all the nodes are already connected to a small set of very similar neighbors, then there is less room to further optimize by re-balancing the connection weights relative to each other (be it directly or through intermediary σ parameters).

Note that the value of the objective function is not proportional to n. Sometimes, it even decreases when n increases. For example, the starting value obj(σ_{start}) as well as the optimized value can be smaller when taking an entire data-set compared to their counterparts based on only a part of the same data-set. This is counterintuitive since equation (3.3) is a sum of n non-negative contributions. Each llp is constructed from a limited amount of neighbors (depending on sparsification). If the number of records increases, the density of nodes in the manifold regions of the input space increases. The llp can be constructed out of increasingly similar neighbors which decreases the reconstruction error (the number of neighbors being held constant irrespective of n). The decreasing size of each contribution to equation (3.3) can outweigh the increasing number of contributions. This is a reason to believe that AEW should work well with high values of n in big data scenarios.

When using kNN according to equation (2.5) for graph construction, some nodes are hubs and have $\gg k$ connections. AEW with its capability of reducing the weights of unwanted edges neutralizes this potential disadvantage of kNN graphs. Figure 3.1 illustrates this on the famous Fisher (1936) data-set.

Comparison with other edge weightings The AEW formula for edge weights, equation (3.1), uses the Gaussian kernel similarity formula equation (2.10) with one parameter per variable. Contrary to Gaussian kernels, AEW provides a principled way of optimizing its σ_d parameters. This optimization can be done without labels which is a considerable advantage over the traditional use of Gaussian kernels in case labels are very sparse. There is no need to complicate one's experimental setup in order to tune σ and less of a threat of deteriorating classification performance if one narrowly misses the ideal parameter value.

The LLE objective function equation (2.13) on p. 19 is very similar to equation (3.3). The only difference is that equation (2.13) does not need to divide by the node degrees because they are constrained to 1. The difference between LLE and AEW does not lie in the objective, but in the decision variables that allow to optimize the objective. AEW optimizes edge weights more indirectly and with fewer degrees of freedom than LLE. LLE optimizes $\mathbf{W}$ directly to minimize the linear reconstruction error. It can optimize around nk parameters in a sparse undirected graph or up to n^2 parameters in a fully connected asymmetric graph. This makes LLE noise-sensitive and prone to over-fitting. With AEW, the edge weights are optimized over only p parameters and then computed based on resulting node similarity. Even in high dimensional data-sets where $p > n$, we



(a) binary **W** before kNNG (b) optimized **W** after AEW

(c) node degrees before AEW (d) node degrees after AEW

Figure 3.1: Adjacency matrix **W** of the famous Fisher (1936) data-set with $k = 20$
The records are ordered by class label. The AEW optimization has clearly reduced the weight of interclass connections (those which are not in the squares around the diagonal).
The histograms show lower node degrees and a more balanced graph after AEW.

usually do not have $p > n^2$, especially not with high values of n thanks to unlabeled data. [2] AEW is less noise sensitive than LLE.

AEW essentially uses LLE's objective function to optimize one parameter per variable which is then put into a kernel similarity function to compute edge weights. This combination considers node similarity and reconstruction error. Because of this hybrid approach, AEW cannot be optimized in a closed form.

Karasuyama and Mamitsuka (2016) show that, under the manifold assumption and when using the unnormalized Laplacian **L**, optimizing equation (3.3) not only minimizes reconstruction error of the inputs, but will also achieve a good trade-off between two error terms of the reconstruction of outputs labels from neighboring labels. The first error term of label reconstruction is proportional to the reconstruction error of the inputs. It can be reduced by connecting more neighbors. The second error term is proportional to the reconstruction error on the underlying low dimensional manifold. It increases when too many edges exist. With LLE, it is difficult to make that trade-off. A hyperparameter needs to be tuned (Kong et al., 2012). Because AEW optimizes equation (3.3) by going through similarity parameters, it provides an unsupervised yet principled way to connect only the right nodes.

While AEW promises ameliorations over existing techniques, it still cannot deal very well data-sets where a large number of variables are white noise and/or correlated with each other in complex

[2] Even if $p > n^2$, as is imaginable with pixel-wise high resolution images, AEW would become very computationally intensive. It would probably be best to reduce the number of dimensions by cropping and down-scaling the images.

ways. The `CoIL2000` data-set is such a case. It contains not only strong class- and cost imbalance but also many variables which describe the records very indirectly and poorly and which are correlated (van der Putten, 2000; Van Der Putten & Van Someren, 2004). AEW combined with kNNG and label propagation performs very poorly, but so do most other algorithms without previous feature engineering.

Numerical stability Equation (3.2) as substituted into (3.3) is prone to numerical instability. When the numerators are very big compared to the denominators, which is to say when the nodes are very dissimilar, the expression approaches $\exp(-\infty) = 0$. This behavior is somewhat wanted because it allows AEW to further sparsify a graph which may have too many edges. In the extreme case, we could even hand a fully connected binary graph to AEW.

However, if, for at least one node, all the connections are weighted with zero, we divide by a node degree of zero in the objective function. This can happen when using the local scaling kernel and scaled dissimilarity measures at the same time. [3] When the starting values for σ_d are too low compared to the numerator, the optimization cannot start since the objective function is undefined. Some values of the objective function that are computed during the iterative optimization might be undefined as well. It is advisable to use an implementation that can recover from such errors.

Karasuyama and Mamitsuka (2016), Gretton, Borgwardt, Rasch, Schölkopf, and Smola (2006) propose the median dissimilarity of all the connected node-pairs as the scalar starting value and insert it to each element of the σ_d vector. When edge weights are computed with local scaling kernel, the same scaling should be applied to the dissimilarities that determine the starting point. This prevents division by zero.

Gradient Karasuyama and Mamitsuka (2016) provide an analytic expression of the gradient of the objective function as equation (3.7) which we shall derive here. Before computing derivatives of the objective function, we can rewrite the square vector norm as a matrix product. When $\mathbf{x}$ is a row vector, we have $\|\mathbf{x}\|_2^2 = \mathbf{x}\mathbf{x}^\top$. For column vectors, $\|\mathbf{x}\|_2^2 = \mathbf{x}^\top\mathbf{x}$. We will assume that $\mathbf{x}_i$ and $\hat{\mathbf{x}}_i$ are row vectors since data-sets are usually stored in this way. Let's look at the derivative of the objective function with respect to a parameter σ_d. The mathematical expressions are the same for all parameters of σ, only the indices change.

$$\begin{aligned}\frac{\partial obj}{\partial \sigma_d} &= \frac{\partial}{\partial \sigma_d}\left[\sum_{i=1}^{n}(\mathbf{x}_i - \hat{\mathbf{x}}_i)\times(\mathbf{x}_i - \hat{\mathbf{x}}_i)^\top\right] \\ &= \sum_{i=1}^{n}\frac{\partial}{\partial \sigma_d}\left[(\mathbf{x}_i - \hat{\mathbf{x}}_i)\times(\mathbf{x}_i - \hat{\mathbf{x}}_i)^\top\right]\end{aligned} \tag{3.4}$$

The derivative of a sum equals the sum of a derivative. The n elements which constitute our derivative are independent. We can make abstraction of the sum and keep analyzing only one element associated with one node i. Furthermore, the element wise derivative of a tranposed vector equals the transpose of its element wise derivative: $\frac{\partial}{\partial \sigma_d}\left[\mathbf{x}^\top\right] = \left[\frac{\partial}{\partial \sigma_d}\mathbf{x}\right]^\top$. When taking the

[3] Although linguistically similar, these concepts are different and it should be possible to use both together. Scaled dissimilarities essentially scale data from $\mathbb{R}^p$ to $[0,1]^p$. Having done that, there can still be regions of lower and regions of higher point density in the $[0,1]^p$ space. The need for local scaling is thus independent of scaled/unscaled data.

derivative of element i in the equation (3.4), we use the rule of derivation for products: $\frac{d}{dx}(u \times v) = u \times \frac{dv}{dx} + \frac{du}{dx} \times v$. In our case, row vector u is the transpose of column vector v and it does not matter which one of the two vectors we take the element wise derivative of before we multiply them to get a scalar. The expression simplifies:

$$\begin{aligned}
&\frac{\partial}{\partial \sigma_d}\left[(\mathbf{x}_i - \hat{\mathbf{x}}_i) \times (\mathbf{x}_i - \hat{\mathbf{x}}_i)^\top\right] \\
&\quad = (\mathbf{x}_i - \hat{\mathbf{x}}_i) \times \frac{\partial}{\partial \sigma_d}(\mathbf{x}_i - \hat{\mathbf{x}}_i)^\top + \frac{\partial}{\partial \sigma_d}(\mathbf{x}_i - \hat{\mathbf{x}}_i) \times (\mathbf{x}_i - \hat{\mathbf{x}}_i)^\top \\
&\quad = 2(\mathbf{x}_i - \hat{\mathbf{x}}_i)\left[\frac{\partial}{\partial \sigma_d}(\mathbf{x}_i - \hat{\mathbf{x}}_i)\right]^\top
\end{aligned} \tag{3.5}$$

The transposed term in the simplified expression of equation (3.5) is the only term that contains a derivative. Since $\mathbf{x}_i$ does not depend on σ_d, but $\hat{\mathbf{x}}_i$ does, we substitute in the following way:

$$\begin{aligned}
\frac{\partial}{\partial \sigma_d}(\mathbf{x}_i - \hat{\mathbf{x}}_i) &= -\frac{\partial}{\partial \sigma_d}\hat{\mathbf{x}}_i = -\frac{\partial}{\partial \sigma_d}\left[\frac{1}{D_{ii}}\sum_{j\sim i} W_{ij}\mathbf{x}_j\right] \\
&= -\left[\frac{\partial}{\partial \sigma_d}\left\{\frac{1}{D_{ii}}\right\} \times \sum_{j\sim i} W_{ij}\mathbf{x}_j + \frac{1}{D_{ii}} \times \frac{\partial}{\partial \sigma_d}\left\{\sum_{j\sim i} W_{ij}\mathbf{x}_j\right\}\right] \\
&= -\left[\frac{\partial}{\partial \sigma_d}\{D_{ii}\}\frac{-1}{D_{ii}^2} \times \sum_{j\sim i} W_{ij}\mathbf{x}_j + \frac{1}{D_{ii}} \times \sum_{j\sim i}\left(\frac{\partial}{\partial \sigma_d}\{W_{ij}\}\mathbf{x}_j\right)\right] \\
&= \frac{1}{D_{ii}}\left[\frac{\partial D_{ii}}{\partial \sigma_d}\underbrace{\frac{1}{D_{ii}} \times \sum_{j\sim i} W_{ij}\mathbf{x}_j}_{=\hat{\mathbf{x}}_i} - \sum_{j\sim i}\frac{\partial W_{ij}}{\partial \sigma_d}\mathbf{x}_j\right]
\end{aligned} \tag{3.6}$$

After substituting the value of $\hat{\mathbf{x}}_i$ in equation (3.6), we have applied the product derivation formula again. In the next step, the derivative of $1/D_{ii}$ is computed knowing that D_{ii} represents an expression depending on σ_d. The other derivative in the product formula can be passed inside the sum it is applied to. In the last step, we bring the factor $-1/D_{ii}$ outside the main parentheses. We see that the first term of the product formula contains again $\hat{\mathbf{x}}_i$. We can now put our results back together to describe the derivative of the objective function. Only the index of σ_d changes with respect to the components of the gradient in equation (3.7).

$$\frac{\partial obj}{\partial \sigma_d} = -2\sum_{i=1}^{n}\frac{1}{D_{ii}}(\mathbf{x}_i - \hat{\mathbf{x}}_i)\left(\sum_{j\sim i}\frac{\partial W_{ij}}{\partial \sigma_d}\mathbf{x}_j - \frac{\partial D_{ii}}{\partial \sigma_d}\hat{\mathbf{x}}_i\right)^\top \tag{3.7}$$

In Karasuyama and Mamitsuka (2016), the gradient does not contain this factor "-2". This might be a voluntary abstraction since $\nabla kf = k\nabla f$ when k is a constant. We only need to find the derivatives of edge W_{ij} and D_{ii} with respect to σ_d (equations 3.8 and 3.9). Note that in Karasuyama and Mamitsuka (2016), the derivative of W_{ij} does not contain the local scaling terms s_i and s_j. This is a little misleading. While we are not obliged to use a local scaling kernel, once we use it, we also need to use it in the expression of the gradient.

$$
\begin{aligned}
\frac{\partial W_{ij}}{\partial \sigma_d} &= \frac{\partial}{\partial \sigma_d}\left[\exp\left(-\sum_{\delta=1}^{p}\frac{(x_{i\delta}-x_{j\delta})^2}{s_i s_j \sigma_\delta^2}\right)\right] \\
&= \underbrace{\exp\left(-\sum_{\delta=1}^{p}\frac{(x_{i\delta}-x_{j\delta})^2}{s_i s_j \sigma_\delta^2}\right)}_{=W_{ij}} \times \underbrace{\frac{\partial}{\partial \sigma_d}\left(-\sum_{\delta=1}^{p}\frac{(x_{i\delta}-x_{j\delta})^2}{s_i s_j \sigma_\delta^2}\right)}_{\text{only the term with } \delta=d \text{ is } \neq 0} \\
&= W_{ij}\frac{-(x_{id}-x_{jd})^2}{s_i s_j}\frac{\partial}{\partial \sigma_d}\frac{1}{\sigma_d^2} \quad = \quad 2W_{ij}\frac{(x_{id}-x_{jd})^2}{s_i s_j \sigma_d^3}
\end{aligned}
\tag{3.8}
$$

The derivative of a node's degree D_{ii} is a simple sum of the derivatives of the connected edges:

$$
\frac{\partial D_{ii}}{\partial \sigma_d} = \sum_{j\sim i}\frac{\partial W_{ij}}{\partial \sigma_d} = \sum_{j\sim i} 2W_{ij}\frac{(x_{id}-x_{jd})^2}{s_i s_j \sigma_d^3} \tag{3.9}
$$

Optimization algorithm We use an algorithm of the Quasi-Newton type called BFGS (Floudas & Pardalos, 2008) for unconstrained optimization. It can be used to find roots of non-linear multi-valued functions or, as in our case, optima of single valued functions with mutliple independent variables. Finding the optimum of a single-valued function is mathematically equivalent to finding the root of its multi-valued gradient. In order to do this, BFGS estimates the Hessian of the single-valued function (or the Jacobian of the multi-valued function if we were to search roots) numerically based on the gradient. At each iteration, BFGS first finds the optimal direction based on the gradient and Hessian. Once the direction is established, it performs a line search to determine the optimal step-size and updates the estimated Hessian.

BFGS works best with convex twice continuously differentiable functions but it has been shown to do reasonably well outside of those conditions (Lewis & Overton, 2009). Alternatively, derivative free optimization algorithms which do not pose such conditions can be used. They are however susceptible to local optima and do generally need more computation time (measured in evaluations of the objective function) than BFGS with numerical gradients (Rios & Sahinidis, 2013).

Unfortunately, we were not able to implement the analytic gradients correctly (with or without the unaccounted factor of "-2" that we find compared to Karasuyama and Mamitsuka (2016)). The optimization quickly falls into what it believes to be local minima. Those are not real minima, the optimizer simply does not look in the right directions. The objective function does not get optimized much. In some cases when the graph is well sparsified, classification performance can still be decent with spurious analytic gradients. With most graphs though, classification will suffer.

We need to find another solution: In absence of the analytic expression of the gradient, the Quasi-Newton algorithm that we use can numerically estimate the derivatives based on central differences. Our only concern when doing this is tractability, not precision. If there is enough time to compute numerical gradients, the results can be trusted. In fact, we know that the implementation of the analytic gradient is incorrect because Matlab compares it to numerically estimated derivatives before using it. This shows implicitly that the numerically computed gradients can be trusted.

Computational complexity The computational complexity of AEW depends on the potential need for a global optimization heuristic. As the objective function cannot be shown to be convex, it is possible to end up in local minima. [4] Our experiments have shown that repeated optimizations on different data-sets were never able to find better minima than the ones from the (possibly adapted to local scaling) median heuristic starting points. We have tested with large amounts of randomly chosen starting values. This is all we can hope for in a starting point heuristic and reduces computational complexity. This result is consistent with a similar finding from LeCun, Bengio, and Hinton (2015) who state that local minima are of very low practical relevance with gradient descent algorithms in artificial neural networks.

Note that the AEW optimization cannot easily be done in parallel. It is iterative in nature. Global optimization heuristics, which launch multiple optimizations, would have made this problem irrelevant, but we are fortunately able to avoid them. This is a real limitation in 2017 were consumer PCs can have up to 10 CPU cores and GPUs are even more parallel. If we want to parallelize AEW, we need to do it at a lower level than the iterations. An iteration takes multiple evaluations of the objective function. Those can be distributed over computing units insofar as they are independent and not part of an iterative line search. This is notably the case when the gradient is numerically estimated in a high dimensional data-set. [5]

At an even lower level, the evaluation of the objective function equation (3.3) accumulates contributions from n nodes which are independent and can be parallelized. The same goes for the derivatives of the objective function in equation (3.4). For very high values of n, this approach allows to spread the computations over $\gg p$ computing units. Depending on the programming language, the speed gains may or may not outweigh the overhead from distributing many small computation jobs in this way. For medium values of n or when fewer than p computing units are available, we can still parallelize the computations during the line-search phase of BFGS.

These challenges in parallelizing are in contrast to the previous- and following steps in graph-based learning. Regarding graph construction, computation of the full distance matrix can be parallelized as distances are independent. [6] Finding k nearest neighbors in n vectors of length n each can be done on up to n parallel computing units. Regarding label propagation, the relevant operations of matrix inversion and solving linear systems are per default parallel in Matlab. When computers become even more parallel in the future, AEW may become more and more a bottleneck in the three step process.

Let's have a closer look at the computational complexity of the optimization when using analytical or numeric gradients. In both cases, the objective function needs to be evaluated multiple times. Each evaluation has $\mathcal{O}(nkp)$ asymptotic time complexity because it needs to loop over all n nodes and their respective $\approx k$ neighbors and compute edge weights based on p variables. k can be held at relatively low values of 10 to 25 and does not need to increase when the size of the problem (n and/or p) does. This can also be seen as $\mathcal{O}(np)$ with sparse graphs and as $\mathcal{O}(n^2p)$ with fully connected graphs. At each iteration of the optimization, the objective function needs to be evaluated at least once. The number of needed iterations does not directly depend on the

[4] Besides this risk, a poor starting point can also lead to an undefined objective function (see paragraph about numerical instability on p. 42) and the need for more iterations before convergence.

[5] The Matlab function `fminunc` has the option `UseParallel` for this purpose.

[6] The older Matlab function `daisy` does not do this though.

size of the problem, but on convergence criteria such as the minimum step size and the minimum improvement compared to the objective value at the last iteration. [7] In practice, between 10 and 100 iteration have been needed. [8]

With analytic gradients, the derivatives as in equation (3.7) can be computed in $\mathcal{O}(nkp^2)$ or $\mathcal{O}(np^2)$ with sparse matrices. The factor n represents the summation over nodes, one factor p represents the fact that each derivative $\frac{\partial W_{ij}}{\partial \sigma_d}$ contains a term W_{ij} which requires $\mathcal{O}(p)$ computations and the other factor p represents the p partial derivatives that need to be computed at each iteration. k is a constant that doesn't increase with n or p.

With numeric gradients, each partial derivative is computed based on central differences. This requires at least $2p$ evaluations of the objective function per iteration. The subsequent line search in the direction of the gradient adds more evaluations of the objective function, but does not depend on n or p. This adds up to $\mathcal{O}(p \times np)$.

The complexity of the objective function combined with the derivatives (however computed) is of $\mathcal{O}(np + np^2) = \mathcal{O}(np^2)$. The number of needed iterations should also not depend on the way in which derivatives are computed. The computation of numeric gradients can better be parallelized than the analytic ones. Which method is faster depends on the value of p compared to the number of available computing units and on the constants hidden behind the asymptotic notation. Whichever method is used, AEW can handle high values of n better than high values of p.

Spatial complexity is not an issue with AEW. The previous step of graph construction and the following step of label propagation need amounts of RAM that increase with $\mathcal{O}(n^2)$ unless special precautions are taken (which is not always possible). [9] AEW works with sparse representations of matrices, which need much less RAM.

Table 3.1 indicates roughly how much time the three phases of graph-based SSL take. We see that the number of iterations and function evaluations is unpredictable. In the appendices, tables (C.21) and following show the computation times of the same four data-sets within our experimental setup. In conclusion, graph construction and label propagation will limit the value of n through spatial complexity and AEW the value of p through temporal complexity.

	n	p	kNNG	AEW	LLGC	$\sum$	# iter.	# evals.
`iris`	150	4	0.007	1.664	0.003	1.673	15	243
`bank`	4119	19	3.143	553.880	4.522	561.548	55	3394
`eeg`	14980	15	56.795	841.145	108.259	1006.199	4	1451
`usps`	1500	241	5.354	3838.796	0.245	3844.395	100	54579

Table 3.1: Computation time in seconds for the 3 phases of graph-based SSL
Data-sets of small- (`iris`), medium- (`bank`) and large size (`eeg`) and high dimensions (`usps`)
The most relevant computations have made use of all 6 CPU cores (Xeon X5650)
kNNG and LLGC might be implemented more efficiently than what is the case here
The last two columns show the number of iterations (max. 100) and function evaluations of AEW

[7] To save time, these tolerances can be set one or two orders of magnitude higher than the Matlab default values. Otherwise, the optimization will perform many iterations yielding only tiny improvements that do not help classification.

[8] This factor can be seen as a constant and does not appear in the asymptotic expression.

[9] The computer used for our experiments has 48Gb of RAM is thereby limited to $n \approx 50000$.

3.2 Inductive extension of transductive algorithms

Having presented the AEW technique, we now tackle the question of out-of-sample predictions. In section 3.2.1, we review the difficulties encountered when attempting induction with graph-based algorithms and some proposed solutions. Section 3.2.2 defines the data-set structure that the newly proposed meta-algorithms from section 3.2.3 use when performing out-of-sample predictions.

3.2.1 Why AEW has not yet been tested inductively

Like many graph-based algorithms, AEW does not lend itself naturally to inductive testing on unseen data. New records on which the algorithm's generalization performance shall be tested would, per definition, not be nodes in the existing graph. Can we nevertheless find a way to add nodes to an already constructed graph without compromising the optimized weights? We always assume that the presence of large amounts of unlabeled data warrant the usage of semi-supervised algorithms over inductive supervised ones that would simply ignore unlabeled data.

In the three-step process (see p. 8), the first step of graph construction is quite compatible with an inductive setting. If for instance kNNG is used, each new node will augment the adjacency matrix by one line and one column. Adding edges between old and new- as well as between new and new nodes will only affect the outer red area in figure 3.2, not the old green and blue parts. This process requires $\mathcal{O}(np)$ computations. [10] Theoretically, the old rows and columns of $\mathbf{W}$, edges between old and old nodes, would need to be revised. Sometimes, the new node replaces an existing node among the kNN and the incumbent k^{th} nearest neighbor becomes the $k+1^{th}$ nearest. s_i then should be updated as well. We can ignore these complications. Especially if we use AEW in step 2, it is not problematic if some nodes have a few more neighbors. The weights of unwanted connections would be optimized close to zero.

The second step cannot be done entirely inductively. With AEW, the set of parameters $\{\sigma_1, \ldots, \sigma_p\}$ is optimized simultaneously and based on all feature vectors, see equation (3.3). All the graph's edges, including the new ones, are optimized simultaneously since they all inform- and depend on $\{\sigma_1, \ldots, \sigma_p\}$.

We fall short of our goal to test AEW in an inductive setting. The fact that $\{\sigma_1, \ldots, \sigma_p\}$ are optimized on all records simultaneously requires compromises. Let's look at ways to bypass AEW's limitation to transductive learning.

We could recompute all the edge weights every time a new observation arises. The computational costs would be prohibitive. Imagine 10^5 nodes in the graph which correspond to customer transactions. 1000 new transactions arrive every day (the unseen data onto which to generalize). 1000 times every day, we would need to re-optimize $\mathbf{W} \in \mathbb{R}^{10^5 \times 10^5}$. Anchor graph learning (W. Liu et al., 2010; W. Liu et al., 2012) aims at making such computations more tractable. It is however not easily compatible with AEW since with anchor graphs, the matrix $\mathbf{W}$ depends on a smaller matrix $\mathbf{Z} \in \mathbb{R}^{n \times m}$. $\mathbf{Z}$ connects all nodes to the anchors and is used for prediction. Any AEW optimizations to $\mathbf{W}$ would not be reflected in $\mathbf{Z}$.

If the application scenario permits it, we could wait until a batch of new nodes has accumulated.

[10] Since we still know the distance s_i of each node to its k^{th} nearest neighbor thanks to local scaling, we need only compare those s_i with the distances to the new node. Each distance computation has $\mathcal{O}(p)$ complexity.

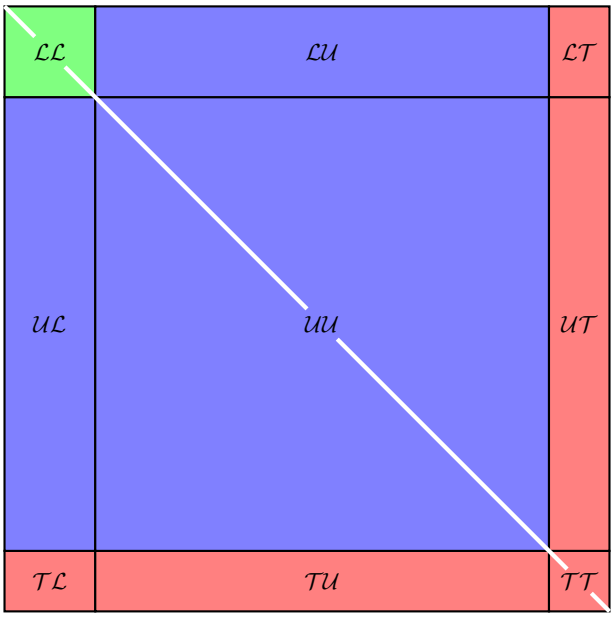


Figure 3.2: Adjacency matrix **W** ordered by labeled- $\mathcal{L}$, unlabeled- $\mathcal{U}$ and test-set status $\mathcal{T}$
The matrix is symmetric, sparse and has zeros on the diagonal
For instance, $\mathcal{UL}$ and $\mathcal{LU}$ connect labeled- to unlabeled training records

We then construct a new graph from scratch which makes sense if the batches are big and arrive rarely. Unseen data would still be used to construct the graph. According to Kuhn and Johnson (2013, ch. 4), the role of a test set is to assess the generalization performance of an algorithm on data that has not been used in any way, shape or form while choosing, constructing and fine-tuning the model and the input variables. Doing so, the test-set simulates real world scenarios where the model should generalize to transactions that are unknowable at time of model construction (i.e. future customers).

Transductive graph-based models do not allow for this strict test-set separation. The graph would always use the information contained in the input features of the new data. However, the argument could be made that this issue is due to the inherent differences between a transductive- and an inductive setting and that any advantages a transductive algorithm might gain from incorporating new unlabeled data in this way are legitimate. The underlying purpose of strictly separating a test-set from the rest of the data is to avoid over-fitting. This is less of a concern with AEW, by design not very noise sensitive or prone to over-fitting due to the parametrization of weights, than with other algorithms such as LLE.

Regarding label propagation, it might be possible to extend to an inductive setting. One approach, manifold regularization, is compatible with AEW and inductive. Within the manifold regularization framework, we could take a Gaussian kernel and reuse the σ parameters from the AEW optimization. We would be limited to binary classification (Belkin et al., 2006). Extending to multiple classes by performing one vs. one or one vs. all classifications would be very computationally complex and classification performance may suffer as well (Eichelberger & Sheng, 2013). Another approach is the induction formula from Delalleau et al. (2005). It is presented for binary classification, but might extend to multiple classes. In order to forgo solving a linear system or inversing a matrix every time new record arises, equation (3.10) is derived from a slightly modified version of

objective function equation (2.15). The global consistency term applies only to labeled examples and which is thus convex with regard to the label confidence of the unseen record $F_{n+1,1}$ (expressed with respect to class 1 with label $y = +1$, class 2 has label $y = -1$). The authors assume that all other label confidences in matrix $\mathbf{F}$ are not influenced by the unseen record $\mathbf{x}_{n+1}$. In the asymptotic case of $n \to \infty$, this is certainly reasonable. The authors show that even with smaller values of n, the assumption has only minimal impact on classification performance. When used on known unlabeled training records, equation (3.10) yields the same results as the transductive propagation.

When used on known labeled training examples, equation (3.10) can yield conflicting values to the label and there is no way to force conformity as was done in equation (2.29). It is no coincidence that the induction formula mirrors the harmonic property of local averages for unlabeled nodes with HGF equation (2.30). As Delalleau et al. (2005) do not compute global consistency on unlabeled examples (such as unseen records), optimizing for $F_{n+1,1}$ considers only local consistency.

$$F_{n+1,1} = \frac{\sum_{j \sim n+1} W_{j,n+1} F_{j,1}}{\sum_{j \sim n+1} W_{j,n+1}} \tag{3.10}$$

This induction formula implicitly uses the function that generated $\mathbf{W}$. It needs to compute weights between old- and new nodes. With AEW, this would make use of equation (2.7) and (3.2), the latter is again based on all records including the new one and needs to be optimized iteratively. Looking at equation (3.2), we see that only the σ_d depend on data from all nodes. We can extend the assumption that $\mathbf{F}$ is not influenced by the unseen record and assume that $\{\sigma_1, \ldots, \sigma_p\}$ are not influenced either. We keep using the parameter set derived from the training data when weighting edges that connect the new node. Zhu (2005, p. 26) calls this approach graph freezing.

Inductive extension should not be possible with local linear embedding where the elements of the adjacency matrix itself are optimized simultaneously through equation (2.13). [11] AEW's parsimonious representation of optimization results through $\{\sigma_1, \ldots, \sigma_p\}$ not only prevents overfitting, but is also useful to extend graph-based learning to an inductive setting.

Augmenting the graph by a batch of new nodes can also be done inductively using frozen parameters. We augment $\mathbf{W}$ by multiple lines and columns and insert binary edges as described at the beginning of the section. Then we use equation (3.2) with the frozen parameters to give weights to those new edges and finally, we perform label propagation. Note that this batch freezing makes less assumptions than using equation (3.10): The parameters σ_d are frozen and carried over, but all the lines in $\mathbf{F}$ are allowed to be influenced by the new nodes since label propagation is repeated. This label propagation is relatively computationally expensive. Freezing in this way makes sense if AEW is much more computationally intensive than label propagation and if new records can be accumulated into reasonable batches. Multiple class labels are supported per default. It remains to be tested how classification performance is impacted.

Section 3.2.3 presents combinations of different types of algorithms into meta-algorithms. They allow for a tractable inductive extension in the general case of multiple classes even if records arrive one by one. Additionally to this convenience, we hope that the combinations of algorithms may benefit from their respective strengths while compensating for weaknesses. Different algorithms use parts of data-sets in different ways, let's therefore first precisely define our data-sets structure.

[11] F. Wang and Zhang (2008) claim they were able to use equation (3.10) with the LLE variant LNP. The authors do however not explain how they were able to estimate the edge weights between old- and new nodes.

3.2.2 Data-set structure

Consider the following data-set structure:

- $\mathcal{L} = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$ training records are available with
  - p features each;
  - the correct labels $\mathcal{Y}_\mathcal{L} = \{y_1, \ldots, y_l\}$ where $y_i \in \{1, \ldots, c\}$ classes;
- $\mathcal{U} = \{\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}\}$ additional training records are available without labels;
- $\mathcal{T} = \{\mathbf{x}_{l+u+1}, \ldots, \mathbf{x}_{l+u+t}\}$ test records are set aside with their correct labels $\mathcal{Y}_t = \{y_{l+u+1}, \ldots, y_{l+u+t}\}$;

In general, $u \gg l$ since the cost of labeling data manually is high. A total of $n = l + u + t$ records are available in $\mathcal{X} = \mathcal{L} \cup \mathcal{U} \cup \mathcal{T}$. Out of those n records, $l + u$ are training records. We do not purposefully introduce label noise and we assume that there is no (substantial) label noise in the data-set. Supervised algorithms cannot make use of the entire data-set as seen in figure 3.3.

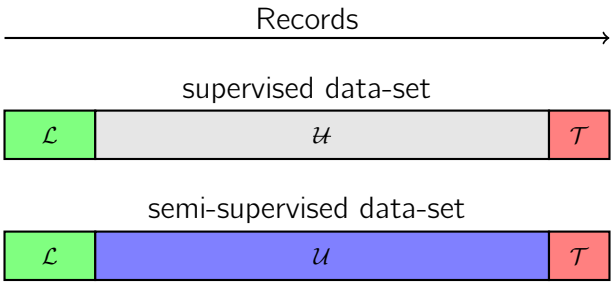


Figure 3.3: Data-set as seen by different algorithms
A supervised algorithm predicts the test-set $\mathcal{T}$ by learning from labeled records $\mathcal{L}$
Only a semi supervised algorithm can also use the many unlabeled records $\mathcal{U}$

3.2.3 Meta-algorithms

In order to compare the performance of transductive- and inductive algorithms on unseen data, we can use combinations of algorithms: meta-algorithms. We describe their general functionality and motivation in the following section and provide unambiguous algorithmic descriptions in pseudo-code. Let's consider a graph-based semi-supervised algorithm $\mathfrak{G}$ and an inductive supervised algorithm $\mathfrak{A}$.

Firstly, algorithms [Induc] and [Transduc] reflect the classic inductive- and transductive settings. $\mathfrak{A}$ cannot make use of unlabeled data $\mathcal{U}$ whereas $\mathfrak{G}$ can. At the same time, $\mathfrak{G}$ needs $\mathcal{T}$ as input data because it cannot generalize to unseen data, which $\mathfrak{A}$ can.

Secondly, algorithm [Freeze] reflects the parameter freezing and batch extension method from section 3.2.1. The first two steps of $\mathfrak{G}$, graph construction and adaptive edge weighting, are performed on the training data. Label propagation to the unlabeled training samples is not of interest since we are only interested in predicting the test-set. We assume that this test-set arrives in one batch and therefore forgo the assumption of freezing $\mathbf{F}$ that equation (3.10) makes. Instead, we add the new nodes to the graph matrix $\mathbf{W}$. We do *not* perform adaptive edge weighting again. We extend the frozen σ parameters from the training data to the test-set and perform label propagation on the augmented graph. Since we make fewer simplifying assumptions than Delalleau et al. (2005), we can reasonably hope for competitive results.

Thirdly, algorithm [Hybrid] shows an iterative combination of $\mathfrak{A}$ and $\mathfrak{G}$ which has very recently been proposed by Govada et al. (2015). Labels are iteratively added to $\mathcal{U}$ based on agreement between $\mathfrak{A}$ (which predicts them inductively) and $\mathfrak{G}$ (which predicts them transductively). If both algorithms agree confidently on a label, it is added and used as training example at the next iteration. In addition to the mere agreement on a label, we consider the geometric mean of both classifier's label confidences and compare it to a cutoff. This geometric mean unifies both confidences in one parameter and conveniently penalizes records with unequal confidences. Govada et al. (2015) look only at the confidence from $\mathfrak{G}$ since they use SVMs which do not easily provide confidences in $\mathfrak{A}$. Using only one confidence measure, the authors conclude the threshold is of low relevance for classification performance. After the iterations have finished, the inductive prediction on the test set is done by $\mathfrak{A}$. Note that such an iterative use of $\mathfrak{G}$ is not an issue regarding computational complexity. If we keep records in the same order, graph construction, AEW and the larger part of label propagation need to be done only at the first iteration.

Fourthly, algorithm [Simple] is a simpler way to combine $\mathfrak{A}$ and $\mathfrak{G}$. It uses each algorithm only once. $\mathfrak{G}$ classifies the unlabeled examples transductively and feeds the most confidently predicted ones to $\mathfrak{A}$, which could otherwise not use them for the induction. In an attempt to simplify the hybrid approach by Govada et al. (2015), we shall see whether the iterations add performance over a one time combination. [Simple] is similar to what Zhu, Lafferty, and Ghahramani (2003) and Zhu and Lafferty (2005) propose. The first paper predicts new records by inductive kNN with respect to the transductively learned records on the graph. In the latter paper, a generative mixture model that can predict unseen records is built over specific nodes on the graph that follows the manifold structure of the data. Another way to simplify [Hybrid] would be to use $\mathfrak{A}$ and $\mathfrak{G}$ simultaneously, but to deliberately limit ourselves to one or two iterations.

Fifthly, meta-algorithm [Self] is quite similar to the iterative hybrid approach. In classic self-learning (see Triguero et al. (2015) for a comprehensive overview), one starts with the labeled data and an inductive algorithm $\mathfrak{A}$. One then iteratively adds labels to the unlabeled examples $\mathcal{U}$ where they can be predicted with the highest confidence. Meta-algorithm [Self] shows a simple version of self learning (categorized as "single view, single learning, single classifier and incremental" by Triguero et al. (2015)).

Finally, [Ext] is an extension of [Hybrid] that uses ensembles of classifiers and majority voting. The idea is to use more than two algorithms (transductive and inductive ones) and to define an agreement threshold $\xi < 1$, as a fraction of the number of used algorithms, above which a label is added to a record. If agreement of all algorithms was required in this scenario, fewer labels would be added the more algorithms are being used. This would defeat the purpose of using $\mathcal{U}$. Conversely, if ξ is sufficiently restrictive, we do not need an additional parameter reflecting the average confidence of this agreement as we did in [Hybrid]. When requiring multiple agreements, the chance of counterproductive random agreement is lower (assuming that the used algorithms are truly different and not close variants of each other). The test-set labels will be attributed according to the highest mean (or equivalently sum) of label confidence among inductive algorithms. Govada et al. (2015) propose to parallelize the training of both algorithms within each iteration. When using multiple algorithms, parallelization can be done even more effectively.

Many of the presented meta-algorithms feed uncertain labels, that a prior algorithm predicted,

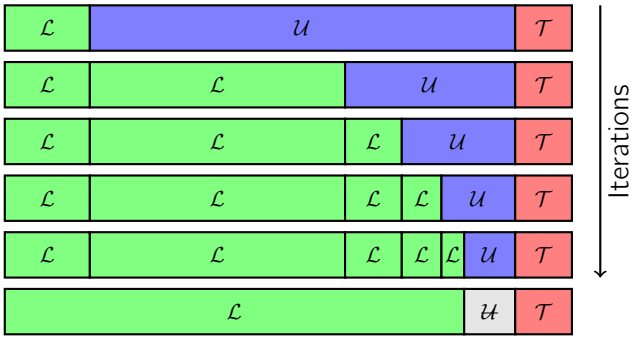


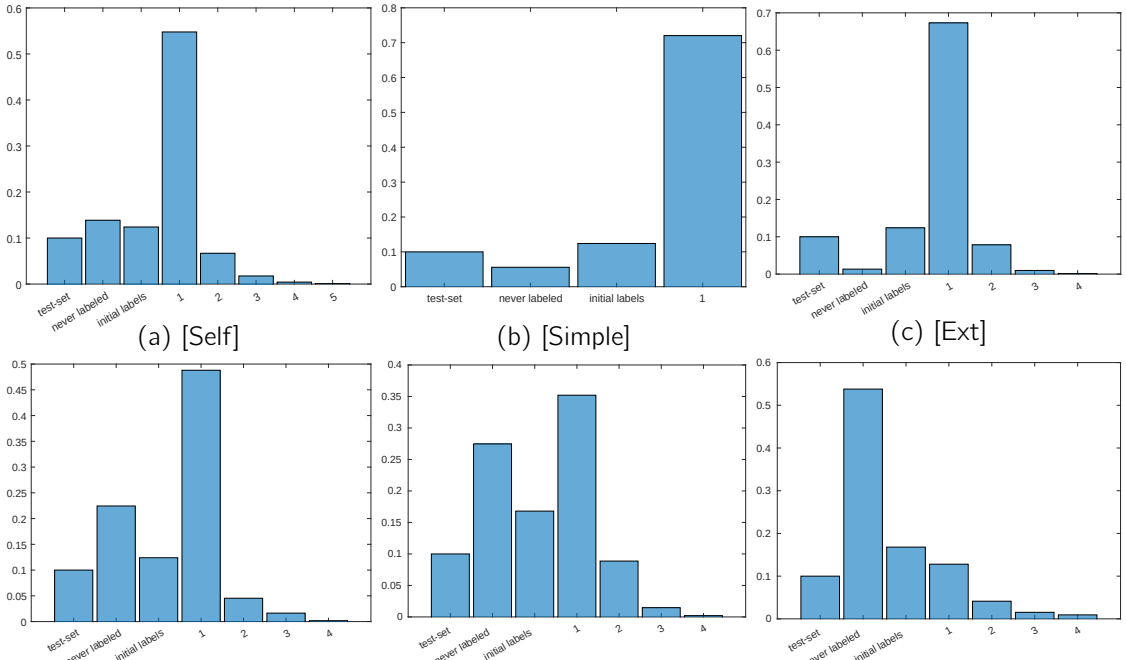
Figure 3.4: Iterative labeling in algorithms [Self], [Hybrid] and [Ext]
Unlabeled records in $\mathcal{U}$ get iteratively predicted and become labeled records in $\mathcal{L}$
After that, the inductive prediction of test-set $\mathcal{T}$ ignores the remainder of $\mathcal{U}$

as training examples into another algorithm. Since the accuracy of the first algorithm is not perfect, this raises the question of how the other algorithms deal with incorrect training labels. There is no way for an algorithm to know that a label is *wrong*, it can at most conclude that a label is *unusual* compared to the rest of the training set. The ability to ignore highly unusual labels is what makes an algorithm resistant to over-fitting (regardless of whether or not the unusual label is known with certainty). Robustness to false labels is therefore strongly correlated with robustness to over-fitting.

A compromise needs to be found regarding the threshold θ used in [Simple], [Self] and [Hybrid]. Note that since θ is used cumulatively with the agreement criteria in [Hybrid], it may be advisable to use a lower cutoff. For [Self], the threshold produces an excessive amount of iterations when set too high (≥ 0.9) or too low (≤ 0.7). 0.85 works well on Fisher (1936)'s dataset. A low threshold will label most records in the first iterations (not always confidently) but then keep on adding a few labels for many iterations. A high threshold leads to increasing proportions of records that will be labeled in later iterations or never. Some records fail to meet the criteria even if classification confidence was decent. Other records are labeled so late that label uncertainty is propagated over many iterations, making the label untrustworthy even if the threshold was high. Many iterations are also computationally expensive. All these scenarios are undesirable and can be dealt with by imposing, in addition to θ, a minimum batch size of new labels per iteration $\phi > 1$ and/or a hard iterations limit ψ.

[Hybrid] might take many iterations if we do not look at classification confidence, especially with large data-sets. Having labeled all the records that are easy to predict in early iterations, later iterations might add only very few labels where both classifiers randomly happen to agree on a label without much confidence. Those labels should not be trusted. Restricting ourselves to adding confidently predicted labels should help select records that are representative of their classes. Irrelevant or worse parasitic records are not labeled. Considering the manifold assumption that $\mathfrak{G}$ is based on, irrelevant records could be outliers that are not close to any manifold and parasitic records could be those that bridge manifolds of different class labels (F. Wang & Zhang, 2008). Looking at only one confidence, as Govada et al. (2015) do, might be enough. However, if $\mathfrak{A}$ is chosen such to provide confidences, there is no reason not to use them.

Figure 3.4 (a-d) illustrates a good compromise: Not too many iterations are necessary and the majority of records is labeled early. Regarding [Hybrid], this small data-set suggests not to use a high threshold θ since it keeps most records unlabeled, to be confirmed.



(a) [Self] (b) [Simple] (c) [Ext]

(d) [Hybrid] with $\theta = 0.5$ (e) [Hybrid] with $\theta = 0.7$ (f) [Hybrid] with $\theta = 0.85$

Figure 3.5: Probability of a record from Fisher (1936) to be labeled in a given iteration.
From left to right we see the proportions of : test-set records (10%), never labeled records, initially labeled records (15%) and then the iterations
For each of the 4 meta-algorithms, proportions are merged over 10 folds and 3 repetitions.
$\theta = 0.85$ for [Self] and [Simple] and $\xi = 0.75$ for [Ext]

Algorithm [Induc]: Inductive supervised learning

Input: Records $\mathcal{L}$ with labels $\mathcal{Y}_\mathcal{L}$ and unlabeled records $\mathcal{U}$
Supervised algorithm $\mathfrak{A}$

train $\mathfrak{A}$ on $\mathcal{L} \cup \mathcal{Y}_\mathcal{L}$
ignore $\mathcal{U}$ since $\mathfrak{A}$ cannot use it
predict labels of $\mathcal{T}$ with $\mathfrak{A}$

Algorithm [Transduc]: Transductive semi-supervised learning

Input: Records $\mathcal{L}$ with labels $\mathcal{Y}_\mathcal{L}$ and unlabeled records $\mathcal{U}$
Semi-supervised algorithm $\mathfrak{G}$
Unlabeled test records $\mathcal{T}$ `// available from the beginning`

train $\mathfrak{G}$ on $\mathcal{L} \cup \mathcal{Y}_\mathcal{L} \cup \mathcal{U} \cup \mathcal{T}$ **begin** `// training and predicting are merged`
    build binary adjacency matrix **W** with kNNG on $\mathcal{L} \cup \mathcal{U} \cup \mathcal{T}$
    AEW on **W** updates the edge-weights
    predict labels of $\mathcal{U} \cup \mathcal{T}$ simultaneously through label propagation from $\mathcal{Y}_\mathcal{L}$
end

Algorithm [Freeze]: Freezing of graph parameters for quasi inductive extension

Input: Records $\mathcal{L}$ with labels $\mathcal{Y}_\mathcal{L}$ and unlabeled records $\mathcal{U}$
Semi-supervised algorithm $\mathfrak{G}$
Unlabeled test records $\mathcal{T}$ `// available at label propagation only`

train $\mathfrak{G}$ on $\mathcal{L} \cup \mathcal{U}$ **begin** `// we do not use` $\mathcal{T}$ `yet`
  build binary adjacency matrix **W** with kNNG on $\mathcal{L} \cup \mathcal{U}$
  AEW on **W** updates the edge-weights and yields parameters $\sigma_{\mathcal{L}\cup\mathcal{U}}$
end
add $\mathcal{T}$ to **W begin**
  augment **W** with empty lines and columns for $\mathcal{T}$
  add binary edge weights in new lines and columns through kNNG
  give weights to the new edges based on the frozen parameters $\sigma_{\mathcal{L}\cup\mathcal{U}}$
end
predict labels of $\mathcal{U} \cup \mathcal{T}$ simultaneously through label propagation

Meta-Algorithm [Self]: Self-learning

Input: Records $\mathcal{L}$ with labels $\mathcal{Y}_\mathcal{L}$ and unlabeled records $\mathcal{U}$
Supervised algorithm $\mathfrak{A}$
Classification threshold θ
Minimum label batch size ϕ
Iterations limit ψ

labelAdded $\leftarrow \phi$
counter $\leftarrow 0$
while $\mathcal{U} \neq \emptyset$ **AND** *labelAdded* $\geq \phi$ **AND** *counter* $< \psi$ **do**
 `// While there are labels to predict, it was possible to add at least` ϕ `of them at the last iteration and up to the hard iterations limit` ψ
  labelAdded $\leftarrow 0$
  counter $\leftarrow$ counter + 1
  train $\mathfrak{A}$ on labeled records $\mathcal{L} \cup \mathcal{Y}_\mathcal{L}$
  foreach $\mathbf{x}_i \in \mathcal{U}$ **do**
    if $\max_{k \leq c} F_{ik} \geq \theta$ **then** `// add only confidently predicted labels`
      $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathbf{x}_i\}$ `// switch record into labeled training-set`
      $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\mathbf{x}_i\}$
      $\mathcal{Y}_\mathcal{L} \leftarrow \mathcal{Y}_\mathcal{L} \cup \hat{y}_i = k$
      labelAdded $\leftarrow$ labelAdded + 1
    end
  end
end
predict labels of $\mathcal{T}$ with $\mathfrak{A}$ (regardless of θ)

Meta-Algorithm [Simple]: Simple combination of (semi-)supervised algorithms

Input: Records $\mathcal{L}$ with labels $\mathcal{Y}_\mathcal{L}$ and unlabeled records $\mathcal{U}$
Supervised algorithm $\mathfrak{A}$ and semi-supervised algorithm $\mathfrak{G}$
Classification threshold θ

train $\mathfrak{G}$ on $\mathcal{L} \cup \mathcal{Y}_\mathcal{L} \cup \mathcal{U}$, i.e. **begin**
  kNN-G
  AEW
  label propagation
end
foreach $\mathbf{x}_i \in \mathcal{U}$ **do**
  if $\max_{k \leq c} F_{ik} \,/ \sum_{\kappa=1}^{c} F_{i\kappa} \geq \theta$ **then** `// add only confidently predicted labels`
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathbf{x}_i\}$ `// switch record into labelled training-set`
    $\mathcal{Y}_\mathcal{L} \leftarrow \mathcal{Y}_\mathcal{L} \cup \hat{y}_i = k$
  end
end
train $\mathfrak{A}$ on $\mathcal{L} \cup \mathcal{Y}_\mathcal{L}$ `// U is thus ignored`
predict labels of $\mathcal{T}$ with $\mathfrak{A}$

Meta-Algorithm [Hybrid]: Hybrid approach to inductive semi-supervised learning

Input: Records $\mathcal{L}$ with labels $\mathcal{Y}_\mathcal{L}$ and unlabeled records $\mathcal{U}$
Supervised algorithm $\mathfrak{A}$ and semi-supervised algorithm $\mathfrak{G}$
Classification threshold θ, minimum label batch size ϕ and iterations limit ψ

labelAdded $\leftarrow \phi$
counter $\leftarrow 0$
while $\mathcal{U} \neq \emptyset$ **AND** *labelAdded* $\geq \phi$ **AND** *counter* $< \psi$ **do**
  labelAdded $\leftarrow 0$
  counter $\leftarrow$ counter + 1
  train $\mathfrak{A}$ on $\mathcal{L} \cup \mathcal{Y}_\mathcal{L}$ `// train both alg. with as much data as they can use`
  train $\mathfrak{G}$ on $\mathcal{L} \cup \mathcal{Y}_\mathcal{L} \cup \mathcal{U}$, i.e. **begin**
    if *counter==1* **then**
      k-NNG `// the first two parts of G are unsupervised`
      AEW `// they need to be done only once`
    end
    label propagation `// this needs to be done at every iteration though`
  end
  foreach $\mathbf{x}_i \in \mathcal{U}$ **do**
    agreement $\leftarrow \arg\max_{k \leq c} F^{\mathfrak{A}}_{ik} == \arg\max_{k \leq c} F^{\mathfrak{G}}_{ik}$ `// F is label confidence`
    agreementConf $\leftarrow \sqrt{\max_{k \leq c} F^{\mathfrak{A}}_{ik} \times \max_{k \leq c} F^{\mathfrak{G}}_{ik} \,/ \sum_{\kappa=1}^{c} F^{\mathfrak{G}}_{i\kappa}} > \theta$
    if *agreement* **AND** *agreementConf* **then** `// if both alg. agree confidently`
      $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathbf{x}_i\}$ `// switch record into labelled training-set`
      $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\mathbf{x}_i\}$
      $\mathcal{Y}_\mathcal{L} \leftarrow \mathcal{Y}_\mathcal{L} \cup \hat{y}_i = k$
      labelAdded $\leftarrow$ labelAdded + 1
    end
  end
end
train $\mathfrak{A}$ on $\mathcal{L} \cup \mathcal{Y}_\mathcal{L}$ `// the remaining U is thus ignored`
predict labels of $\mathcal{T}$ with $\mathfrak{A}$ (regardless of θ)

Meta-Algorithm [Ext]: Extended hybrid approach to inductive semi-supervised learning

Input: Records $\mathcal{L}$ with labels $\mathcal{Y}_\mathcal{L}$ and unlabeled records $\mathcal{U}$
Supervised algorithms $\mathfrak{A}_1, \ldots, \mathfrak{A}_P$
Semi-supervised algorithms $\mathfrak{G}_1, \ldots, \mathfrak{G}_Q$
Agreement threshold ξ
Minimum label batch size ϕ
Iterations limit ψ

labelAdded $\leftarrow \phi$
counter $\leftarrow 0$
while $\mathcal{U} \neq \emptyset$ **AND** *labelAdded* $\geq \phi$ **AND** *counter* $< \psi$ **do**
  labelAdded $\leftarrow 0$
  counter $\leftarrow$ counter + 1
  for $a = 1$ **to** P **do** `// train all algorithms with as much data as they can use`
    train $\mathfrak{A}_a$ on $\mathcal{L} \cup \mathcal{Y}_\mathcal{L}$
  end
  for $a = 1$ **to** Q **do**
    train $\mathfrak{G}_a$ on $\mathcal{L} \cup \mathcal{Y}_\mathcal{L} \cup \mathcal{U}$
  end
  foreach $\mathbf{x}_i \in \mathcal{U}$ **do**
    $\eta \leftarrow 0$ `// \eta counts the max. number of alg. that agree on any label`
    for $k = 1$ **to** c **do** `// we look at each possible label k to determine \eta`
      $\delta \leftarrow 0$ `// \delta counts the number of algorithms agreeing on a label k`
      for $a = 1$ **to** P **do**
        $\delta \leftarrow \delta + \hat{Y}_{ik}^{\mathfrak{A}a}$ `// \delta \leftarrow \delta + 1 each time an algorithm predicts label k`
      end
      for $a = 1$ **to** Q **do**
        $\delta \leftarrow \delta + \hat{Y}_{ik}^{\mathfrak{G}a}$
      end
      if $\delta > \eta$ **then**
        $\eta \leftarrow \delta$ // $\eta \leftarrow \max_k \delta$
        $\hat{y}_i \leftarrow k$
      end
    end
    if $\frac{\eta}{P+Q} > \xi$ **then** `// if enough algorithms agree on a label for record x_i`
      $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathbf{x}_i\}$ `// switch record into labelled training-set`
      $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\mathbf{x}_i\}$
      $\mathcal{Y}_\mathcal{L} \leftarrow \mathcal{Y}_\mathcal{L} \cup \hat{y}_i$
      labelAdded $\leftarrow$ labelAdded + 1
    end
  end
end
for $a = 1$ **to** P **do**
  train $\mathfrak{A}_a$ on $\mathcal{L} \cup \mathcal{Y}_\mathcal{L}$ `// the remaining \mathcal{U} is thus ignored`
  predict label probabilities $\mathbf{F}^{\mathfrak{A}_a} \in [0,1]^{|\mathcal{T}| \times c}$
end
foreach $\mathbf{x}_i \in \mathcal{T}$ **do**
  $\hat{y}_i \leftarrow \arg\max_{k \leq c} \sum_{a=1}^{P} F_{ik}^{\mathfrak{A}_a}$ `// the label with the highest sum probabiltiy`
end

Chapter 4

Experiments

In the last chapter, we have presented two new techniques: AEW (and how it helps with induction) as well as the inductive semi-supervised meta-algorithms. It is now time to put those techniques to the test. We start by explaining our hypotheses concerning the performance of the meta-algorithms in section 4.1. Then we present the actual datasets on which the performance will be compared in section 4.2. The experimental setup described in section 4.3 follows from the methodological considerations discussed in section 2.3 and 2.4. In section 4.4, we discuss experimental results.

4.1 Hypotheses

Intuitively, we would expect algorithm [Induc] to have the worst performance because it cannot make use of $\mathcal{U}$ at all and is limited to the much smaller $\mathcal{L}$. Chapelle et al. (2006, ch. 4) recommends to always perform a baseline comparison against the supervised case and so we do. On the other end, [Transduc] should have the best performance since only it can use $\mathcal{T}$ as input data. Even if this turns out to be true, the other algorithms are still worth considering since it is impractical to recompute [Transduc] every time that a new record needs to be predicted.

Furthermore, algorithms [Simple] and [Hybrid] may perform better than algorithm [Self] since they make use of a graph-based semi-supervised algorithm $\mathfrak{G}$ as well as a supervised algorithm $\mathfrak{A}$, whereas [Self] uses only $\mathfrak{A}$. Our intuition is less pronounced on this hypothesis compared to the previous ones. [Hybrid] should outperform the [Simple]. Otherwise there is no point using the complex iterative procedure. [Ext] adds further complexity to [Hybrid] which is also only justified if it performs better. Algorithm [Freeze] should perform less well than [Transduc] because it extends the usage of AEW parameters from $\mathcal{L} \cup \mathcal{U}$ to $\mathcal{T}$. On the other hand, it uses information from $\mathcal{T}$ when predicting $\mathcal{T}$ which puts it at an advantage over the other contenders. Let's put these hypotheses to the test:

H1 Algorithm [Transduc] performs better than algorithm [Freeze]

H2 Algorithm [Freeze] performs better than algorithm [Ext]

H3 Algorithm [Ext] performs better than algorithm [Hybrid]

H4 Algorithm [Hybrid] performs better than algorithm [Simple]

H5 Algorithm [Simple] performs better than algorithm [Self]

H6 Algorithm [Self] performs better than algorithm [Induc]

To varying degrees of conviction, we expect classification performance of the algorithms to line up in the following way. Question marks signal comparisons about which we have less confidence.

$$[Transduc] > [Freeze] >^{?} [Ext] > [Hybrid] > [Simple] >^{?} [Self] > [Induc] \quad (4.1)$$

Associated with each hypothesis, we have a null hypothesis $\mathbf{H}_0$ and an alternative hypothesis $\mathbf{H}_A$. Even though we have expectations about how algorithms compare, rejection of the null hypotheses is determined by bilateral statistical tests. This common practice and prevents the experimenter's expectations from interfering with the results. Differences are only half as likely to be significant in this setup compared to one sided tests which would raise suspicions of having adjusted the test's direction after seeing the data (Lombardi & Hurlbert, 2009).

$\mathbf{H}_0$ There is no difference in classification performance

$\mathbf{H}_A$ There is a difference in classification performance

The null hypothesis is rejected for p-values smaller than $\alpha = 0.05$ adjusted for FWER and not rejected based on the available data for p-values larger than that. Comparing all combinations of algorithms would have yielded $\frac{7\times(7-1)}{2} = 21$ hypotheses which is not only cumbersome, but also a problematic regarding statistical power. Correcting for the FWER with 21 hypotheses makes errors of type II likely, not correcting for it errors of type I. Some of those combinations are not of interest. Why would we want to directly compare [Freeze] with [Self] for example? As it is, we can formally compare each algorithm to both of its neighbors in the lineup equation (4.1). Each hypothesis is tested on a data-set per data-set basis with corrected resampled t-tests and then also over all data-sets with binomial tests.

4.2 Data-sets

Table 4.1 shows an overview of the data-sets that will be used. n, p and c are as usual the number of records, variables and classes. $\max_k$ and $\min_k$ refer to the prevalence of the most- and the least common class in the data-set and show if there is class imbalance. Column "SSL" indicates whether this data-set is taken from other SSL papers or books. This circumstance might indicate that the smoothness- and manifold assumptions are met and that the data-set particularly illustrates the strengths of SSL. Most of our data-sets have been collected in real world scenarios, the few artificial ones are indicated in the column right to SSL. They are artificially created to highlight strengths and weaknesses of semi-supervised learning (Chapelle et al., 2006, ch. 21). The other data-sets are mostly taken form the UCI machine learning repository (Lichman, 2013).

For reasons of tractability in light of our available resources, we cannot perform tests on big-data scenarios with millions of records. Nevertheless, most of our data-sets have considerably more records than in many SSL papers and we have also tested some high-dimensional data-sets. We strive to test the algorithms on a realistic mix of real word data-sets. A few remarks:

- We have left the `banknotes` data-set in the mix for transparency reasons. It is useless when it comes to comparing algorithms because it is very easy to predict. After that experience, we have pre-tested the other data-sets in a simpler experimental setup to eliminate those that are very easy- and those that are virtually impossible to predict.

- The original `adult` data-set has 48842 records. We have randomly chosen 33% because of computational limitations. Another version of data-set `spam` with 10 times more records is also available. The other data-sets all contain their original number of records.
- Data-set `spam` originally contains 54 sparse variables that represent term frequencies. We performed a PCA and kept 10 dimensions because this exclusive use of sparse variables causes problems with multiple of our algorithms. Regarding all other data-sets, we keep feature engineering to the strict minimum: Scaling of interval variables when necessary for AEW.
- High dimensional data-sets with $p > 50$ (`mice`, `spam`, `urban` and the SSL group) represent either pixels from images, text or other sensory data. Most of the low dimensional data-sets represent heterogeneous data with different types and scales of variables. It will be interesting to see how the manifold assumption holds under these different conditions.
- The artificial data-set `digit1` is designed to satisfy the manifold assumption, which both g241 data-sets do not. In `g241c`, the records at least lie in (slightly overlapping) clusters with one class per cluster. In `g241n`, they lie in misleading clusters with mixed classes.
- `cardio` can be classified according to 2 distinct taxonomies with 3 and 10 classes.

Name	n	p	c	$\max_k$	$\min_k$	SSL	art.	Source
`adult`	16088	14	2	0.76	0.24			(Kohavi, 1996)
`bank`	4119	19	2	0.89	0.11			(Moro, Cortez, & Rita, 2014)
`banknotes`	1372	4	2	0.56	0.44			(Lichman, 2013)
`bci`	400	117	2	0.50	0.50	x		(Chapelle, Scholkopf, & Zien, 2006)
`biodeg`	1055	41	2	0.66	0.34			(Mansouri, 2013)
`cardio3`	2126	21	3	0.52	0.08			(Ayres-de-Campos, 2000)
`cardio10`	2126	21	10	0.27	0.02			(Ayres-de-Campos, 2000)
`coil`	1500	241	6	0.16	0.16	x		(Chapelle, Scholkopf, & Zien, 2006)
`digit1`	1500	241	2	0.51	0.49	x	x	(Chapelle, Scholkopf, & Zien, 2006)
`eeg`	14980	15	2	0.55	0.45			(Lichman, 2013)
`g241c`	1500	241	2	0.50	0.50	x	x	(Chapelle, Scholkopf, & Zien, 2006)
`g241n`	1500	241	2	0.50	0.50	x	x	(Chapelle, Scholkopf, & Zien, 2006)
`mice`	1080	77	8	0.14	0.10			(Higuera, Gardiner, & Cios, 2015)
`retino`	1151	20	2	0.53	0.47			(Antal & Hajdu, 2014)
`seismic`	2584	19	2	0.93	0.07			(Sikora & Wróbel, 2010)
`spam`	4601	54	2	0.61	0.39			(Lichman, 2013)
`thoracic`	470	16	2	0.85	0.15			(Zięba, Tomczak, Lubicz, & Świątek, 2014)
`urban`	675	147	9	0.18	0.04			(B. Johnson & Xie, 2013)
`usps`	1500	241	2	0.80	0.20	x		(Chapelle, Scholkopf, & Zien, 2006)
`wilt`	4889	5	2	0.95	0.05			(B. A. Johnson, Tateishi, & Hoan, 2013)

Table 4.1: Data-sets and their number of records n, variables p and classes c
$\max_k$ and $\min_k$ indicate class (im)balance between the most- and least common class.
SSL points to data-sets designed for semi-supervised learning and art. to artificial data.

4.3 Experimental setup

Classification performance is assessed exclusively on the test records $\mathcal{T}$. How an algorithm labels or does not label the unlabeled training records $\mathcal{U}$ and how it might even relabel some records in $\mathcal{L}$ will not be assessed. When using LLGC for example, the global consistency allows to change some existing labels. [1] Since $u \gg l$ in many real world scenarios, we use the following proportions: [2]

$$\begin{aligned} l \approx \ell \times n &= 0.15 \times n \\ u \approx \upsilon \times n &= 0.75 \times n \\ t \approx \tau \times n &= 0.10 \times n \end{aligned}$$

l, u and t are approximate numbers because each record will be randomly sampled into one of those buckets. When n is sufficiently large, this will yield proportions very close to the theoretical sampling probabilities ℓ, υ and τ. Note that the labels of $\mathcal{U}$ are known in the source data-sets as well, but will be disregarded for the sake of testing a semi-supervised setup.

We have established in section 2.4 that bootstrapping is not a viable option and that repeated training/test splits are at a disadvantage compared to cross-validation. We will thus use cross-validation with 10 folds and 3 repetitions to get enough samples for the statistical tests. For each fold and repetition, all the algorithms are compared on an identically divided data-set with paired sample corrected t-tests.

Most of the data-sets in table 4.1 are selected without too strong class imbalances (except `bank`, `seismic` and `thoracic`). As recommended by Y. Wang, Xu, Zhao, and Hua (2010), we do not constrain class proportions to be comparable across labeled and unlabeled examples. Some stratification is still necessary because of the label sparsity. We constrain every labeled training-set $\mathcal{L}$ to contain at least one labeled record of each class. One label of each type in each training set guarantees at least the theoretical possibility to propagate that label. This is also the least you would do in a scenario where labeling is expensive and done by human experts. In some training sets, those labels can still fall on much more useful records, allowing more effective label propagation and better classification performance than in other training sets. This is not an issue since all (meta)algorithms are tested on the same splits of the data-set and compared based on paired sample t-tests. It would even be advantageous to directly compare classifiers trained on the same good training-sets and then also on the same less good training-sets, this makes for a more holistic assessment. In addition, we constrain every test-set to contain at least one record of every class (labeled or unlabeled confounded). This is necessary to avoid situations where our performance metrics would be undefined.

Since these are relatively weak constraints, they are implemented by re-shuffling available labels and folds till the constraints are met. We have an upper limit of 100 re-shuffles after which the last shuffle is kept regardless of the constraints. Without this limit, our data-set partitions could not be considered random anymore since they are dictated more by the constraints than by the random

[1] When LLGC is used within the iterative meta-algorithm [Hybrid] for example, label-changes are possible at each iteration. The next iteration would reset all labels within $\mathcal{L}$ to the default labels and then decide anew if some labels shall be changed for better transductive classification at that iteration. The test-set prediction is done with $\mathfrak{A}$ which uses unchanged labels of $\mathcal{L}$.

[2] Note that those proportions are accurately reflected in figure 3.2, 3.3 and 3.4.

number generator. Note that this limit is empirically never reached. When there are, for each class, at least 2.5 times more records than folds, very few re-shuffles are needed. The first try often meets all the constraints. Exceptions would be very small data-sets, very imbalanced data-sets and data-sets with very numerous classes. Those are not the focus of our research.

Repeated cross-validation is easily compatible with our data set structure. As a first step, each record in $\mathcal{X}$ gets randomly assigned a fold $K \in \{1, \ldots, 10\}$ with probability 1/10 each. This makes for the rotating test-sets in the cross validation. In parallel and independently, each record gets assigned a binary status as labeled or unlabeled with probabilities of $\frac{\ell}{\upsilon+\ell} = \frac{1}{6}$ and $\frac{\upsilon}{\upsilon+\ell} = \frac{5}{6}$ respectively. The labeled/unlabeled status of a record is irrelevant while it is used for testing but becomes relevant again once the folds are rotated. [3] The records in the other 9 training folds constitute $\mathcal{L} \cup \mathcal{U}$. After ten rotations, the folds and the label status of records get reassigned for the next repetition of the CV.

See listing SETUP for an unambiguous description in pseudo code of how all those concepts work together when comparing meta-algorithms on one data-set.

We will test along two relevant dimensions:

- The 7 meta-algorithms $\mathfrak{M}_i$
  This is our main object of interest: testing the list of hypotheses on page 57.
- The chosen data-sets in table 4.1
  We are primarily interested in the performance of the machine learning algorithms. It is necessary to test multiple datasets in order to make sure eventual differences in algorithm performance are not just due to outlier performance on a specific data-set.

We want to isolate the differences generated by the usage of the meta-algorithms. Other sources of performance variation should be held as constant as possible. These sources include the techniques used in the graph-based algorithms $\mathfrak{G}$ at all three steps. Karasuyama and Mamitsuka (2016) gives good indications as to which techniques yield the best results: Graph sparsification was done using kNNG according to equation (2.5) with $k = 20$. Similarity is measured with Gower's metric. This k is high enough to allow linear reconstruction of nodes by their neighbors in step 2 and yet low enough to be computationally tractable and hopefully compatible with the manifold assumption. We have chosen the value of k slightly higher than the usual 10 or 15 that we would take if edges were weighted by LLE. This is possible because AEW, like Gaussian kernels, can virtually sparsify unwanted edges away. In Karasuyama and Mamitsuka (2016), AEW achieved the best performance improvements compared to other semi-supervised algorithms when used in conjunction with LLGC. The authors have optimized LLGC parameter λ in a nested CV, which we do not use. We use $\alpha = 0.99$ as proposed by D. Zhou et al. (2004) instead. We use the variant from equation (2.24) and perform no class imbalance regularizations. A local scaling kernel based on the distance to the k^{th} nearest neighbor is used for AEW. The initial AEW parameters σ_{start} are computed according to the median heuristic on scaled distances. We optimize using the BFGS algorithm with only this one starting point, an iterations limit of 100, numeric gradients based on central differences and a stopping tolerance of 10^{-4}. Table 4.2 provides an overview of all relevant (meta-)algorithms and shows which ones are used in the experiements.

[3] This is why we need 1/6 labeled records such that $1/6 \times 0.9 = 15\%$ of all records are labeled training records.

	Name	Type	Section	Used
kNNG	k nearest neighbor graph	sparsification	2.2.1	x
ϵNG	ϵ neighborhood graph	sparsification	2.2.1	
-	b-matching	sparsification	2.2.1	
LRR	low rank representation	sparsification	2.2.1	
-	Gaussian kernels	weighting	2.2.2	
LLE	local linear embedding	weighting	2.2.2	
AEW	adaptive edge weighting	weighting	3.1	x
LNP	linear neighborhood propagation	weighting	2.2.2	
LLGC	learning with local ... and global consistency	propagation	2.2.3	x
GATM	graph transduction through ... alternating minimization	propagation	2.2.3	
HGF	harmonic Gaussian fields	propagation	2.2.3	
-	Graph Mincuts	propagation	2.2.3	
-	Markov random walks	propagation	2.2.3	
MAD	modified adsorption	propagation	2.2.3	
-	Laplacian regularization	propagation	2.2.3	
-	manifold regularization	propagation	2.2.3	
MP	measure propagation	propagation	2.2.3	
[Transduc]	graph-based learning	all 3 phases	4.3	x
[Freeze]	graph freezing	all 3 + induction	3.2.1 & 3.2.3	x
-	induction formula	induction	3.10	
[Induc]	inductive supervised alg.	non-graph	3.2.3 & 4.3	x
[Self]	self learning	non-graph	3.2.3 & 4.3	x
[Ext]	iterative ensemble	meta-algorithm	3.2.3 & 4.3	x
[Hybrid]	iterative combination	meta-algorithm	3.2.3 & 4.3	x
[Simple]	simple combination	meta-algorithm	3.2.3 & 4.3	x

Table 4.2: Overview of the graph-related algorithms presented in this master thesis
Because of modularity of the 3 phases, there are many variants.
The first two columns are the acronym and name of the algorithm.
The third column describes how this algorithm is used.
Then we refer to the section that contains further explanations. In case of multiple entries, the first refers to the theory, the second to more practical considerations.
Finally, we indicate if an algoritm has been used for the experimental setup.
Variants with different names are listed, some algorithms have internal variants that use the same name. Does not list similarity measures for graph sparsification or underlying techniques like optimization (i.e. BFGS within AEW) and sorting algorithms.

Many of our meta-algorithms use a supervised algorithm $\mathfrak{A}$: [Ext], [Hybrid] and [Simple] use it in combination with $\mathfrak{G}$ while [Self] and [Induc] use only $\mathfrak{A}$. This should always be the same supervised algorithm since we are not investigating the influence of the supervised algorithm on classification performance in our experimental setup. The choice is crucial, it can invalidate our conclusions if done poorly. We have decided to use *random forests* for the following reasons:

- Random forests are a stronger classification algorithm than single decision trees (Dietterich, 2000), competitive for today's tasks (Shotton et al., 2013). We need to choose a strong algorithm if our hypothesis testing shall have not only statistical, but also practical significance. Had we chosen a mediocre straw man, [Transduc] would trivially win against [Induc] and the performance of [Ext], [Hybrid] and [Simple] may suffer.
- Random forests are easier to tune than other competitive supervised algorithms like artificial neural networks. A random forest is essentially a collection of decision trees which are trained on different sets of records drawn with replacement. [4] Variable selection at each node is randomized and trees are left un-pruned (Ho, 1995; Breiman, 1996, 2001). [5]

  The only parameter that we really need to tune is the number of trees in the forest. Breiman (1996) was mainly concerned with the minimal number of trees depending on n and p, indicating that more trees should increase classification performance. The only disadvantage would then be computational complexity at training- and test-time. Oshiro, Perez, and Baranauskas (2012) found diminishing returns setting in between 64 and 128 trees. For our data-sets, of comparable size to this analysis, we decide to use 100 trees. [6]
- Random forests are much less prone to over-fitting than artificial neural networks (T. Hastie, Tibshirani, & Friedman, 2009, ch. 11.5.2, 15.3.4). Segal (2004) could find at most mild over-fitting for un-pruned random forests.
- Random forests have native support for multi-class data-sets.
  Another strong candidate, SVMs that Govada et al. (2015) used inductively when introducing the hybrid meta-algorithm, are excluded for this reason (Hsu & Lin, 2002).
- Random forests have an inherent capability to deal with high dimensional data and irrelevant variables (contrary to neural networks) and do not rely on the manifold assumption. The underlying trees choose the best splitting variables based on training data.

  – This makes [Self] and [Induc] strong contenders when compared directly with [Transduc] and [Freeze].
  – This makes [Ext], [Hybrid] and [Simple] stronger because the strength of $\mathfrak{A}$ can compensate for the weaknesses of $\mathfrak{G}$.

[4] All those sets are drawn from the labeled training data in the context of our experimental setup.

[5] Regarding the exact procedures for drawing the different sets as well as regarding the degree of randomness of the trees, we trust the default parameters of the Matlab function `TreeBagger`.

[6] For comparison: An artificial neural network needs to be tuned along topology parameters such as the number of nodes per layer, the number of layers and the connectedness. The learning rate is another important tuning parameter. The number of learning epochs and eventual early stopping criteria need to be specified as well. Those parameters interact with each other and can be very different depending on the data-set (Bashiri & Geranmayeh, 2011). This would have required at least a nested CV as used in Fouss, Francoisse, Yen, Pirotte, and Saerens (2012). Because of the unlabeled data, our experimental setup is complicated enough as it is.

For [Ext], we use label propagation, random forests and 3 more inductive algorithms: multinomial logistic regression, naive Bayes and classic kNN. All these algorithms can handle multiple classes and provide classification probabilities. We use an agreement threshold $\xi = 0.75$. For kNN , we use half as many neighbors as in graph construction (thus 10) as the algorithm does not have the same possibility to optimize unwanted connections away that AEW has. Regarding naive Bayes, we use a kernel smoothing function. The MLE estimation within logistic regression sometimes needs more labeled training examples per class than we have. This happens with high dimensional data-sets, but is not a big problem. If one of the 5 classifiers malfunctions for some records, the other 4 can still label it according to a majority decision. For some data-sets where logistic regression consistently fails, we limit ourselves to the other 4 algorithms.

Parameter ϕ in [Self], [Hybrid] and [Ext] is set to a minimum batch of 5 new labels per iteration in small data-sets ($n < 500$), to 10 in medium sized data-sets ($500 < n < 2000$), to 20 in large data-sets ($2000 < n < 10000$) and to 30 in very large data-sets ($n > 10000$). The maximum number of allowed iterations $\psi = 20$ for the same three algorithms on all data-sets. Parameter θ was difficult to decide. For [Self] and [Simple], the value of $\theta = 0.85$ as tested on Fisher (1936) has been chosen. For [Hybrid], the small `iris` data-set suggested to not use a confidence threshold since most records remain unlabeled. Following that advice might be dangerous on larger data-sets. We have made more tests: figure 4.1

On the `biodeg` data-set, raising θ from 0.5 to 0.7 has increased the number of iterations and decreased the classification performance. Raising the parameter further to 0.85 had a strange effect: Instead of the usual shape where the number of labeled records declines roughly exponentially with the number of iterations, we now have a bump in the numbers of labeled records at iterations 4 and 5. Even though classification performance is on par with $\theta = 0.5$ again, this is very suspicious. On the `banknotes` data-set, θ has low influence. This is consistent with Govada et al. (2015) but probably the consequence of a data-set that is very easy to predict. In order to test the performance of the iterative [Hybrid] algorithm, we need to allow it to iterate and add enough labels. $\theta = 0.5$ seems to be the only scenario where this is possible. For binary classification, this deactivates the confidence threshold. The larger of both label confidences is per definition > 0.5. Only in multi-class settings do we keep a very moderate threshold.

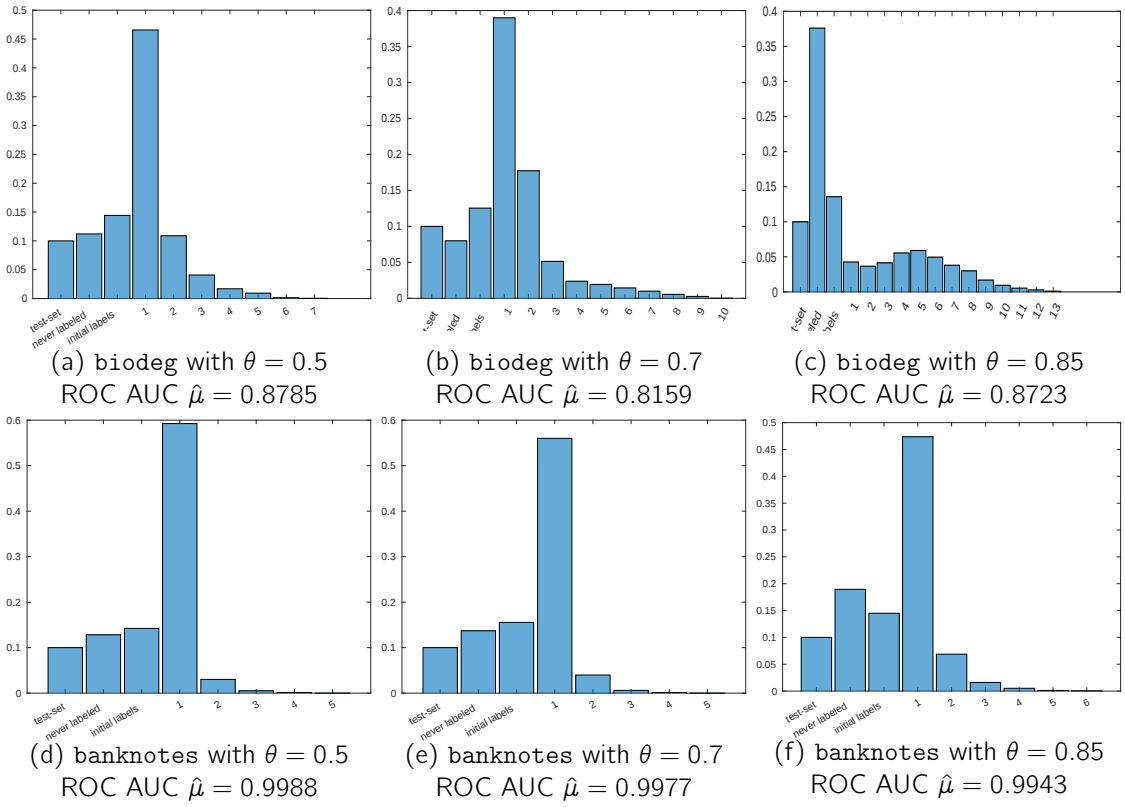
As performance metric, we will be using ROC AUC for binary classification and F-measure for multiple classes. [7] It is true that ROC AUC is mostly used with algorithms that output class probabilities. Label propagation in a graph-based algorithm does not do this: $\sum_k F_{ik} \neq 1$. However, ROC only requires classification confidence measures, not necessarily probabilities. The scoring matrix $\mathbf{F}$ changes smoothly on a given graph (equation (2.14)). The label is chosen by $\arg\max_k F_{ik}$ and the differences between F_{ik} for different k reflect the confidence of that classification.

Even if a specific implementation of ROC AUC requires that the classification confidence measures sum to 1, $\mathbf{F}$ can easily be scaled to be row stochastic such that $\sum_k F_{ik} = 1 \quad \forall i$. Govada et al. (2015, p. 4) even call $\mathbf{F}$ a probability matrix derived from label propagation. It is true that those probabilities are not estimated in the same way as they are by i.e. the random forest algorithms. Any performance differences stemming from those different ways in which probabilities are estimated will be legitimately reflected in the AUC measure. In real application scenarios with

[7] The t-tests on a given data-set always compare performances measured with the same criteria for all algorithms.

varying classification thresholds (which are quite common), such differences would also weigh on the choice of algorithm.

A few remarks about the Matlab implementation can be found in chapter B.



(a) `biodeg` with $\theta = 0.5$
ROC AUC $\hat{\mu} = 0.8785$

(b) `biodeg` with $\theta = 0.7$
ROC AUC $\hat{\mu} = 0.8159$

(c) `biodeg` with $\theta = 0.85$
ROC AUC $\hat{\mu} = 0.8723$

(d) `banknotes` with $\theta = 0.5$
ROC AUC $\hat{\mu} = 0.9988$

(e) `banknotes` with $\theta = 0.7$
ROC AUC $\hat{\mu} = 0.9977$

(f) `banknotes` with $\theta = 0.85$
ROC AUC $\hat{\mu} = 0.9943$

Figure 4.1: Probability of a record to be labeled in a given iteration by [Hybrid]
Proportions are merged over 10 folds. From left to right we see the proportions of : test-set records (10%), never labeled records, initially labeled records (15%) and then the iterations

4.4 Results

Table 4.4a shows which of our hypotheses are supported by which data-sets. If the corresponding null-hypotheses are rejected, we also indicate the direction of the statistically significant difference. Table 4.4b shows for which hypothesis and data-set the algorithm that we thought would win, did win. Statistical significance of the performance differences are explicitly not taken into account here. In an attempt to condense information, table 4.3 uses the content from table 4.4b to make binomial tests across data-sets. Table 4.5 shows the sample mean and sample variance of each algorithm's performance on each data-set. This is not exactly how the tests were done since we use paired-sample tests for reasons of statistical power (see p. 33). The table gives an overview of how large performance differences between algorithms are in order to qualify or not as significant and also how easy or difficult a given data-set is to classify. In the appendices, there are table C.1 and following with the details (test statistics, p-values etc.) of all the hypothesis tests. Table C.25 and following contain the raw data that those tests are based on. The tests are always based on the right sub-table that features ROC AUC or F-measure, never on the accuracy measures.

Let's review the obtained results by hypothesis and end on some related insights.

Meta-Algorithm SETUP: Experimental setup for a given data-set

Input: n records $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$
Corresponding labels $\mathcal{Y}_\mathcal{X} = \{y_1, \ldots, y_n\}$
Meta algorithms $\mathfrak{M}_1, \ldots, \mathfrak{M}_M$
Number of folds K and repetitions R
Probabilities ℓ, υ and $\tau = \frac{1}{K}$ s.t. $\ell + \upsilon + \tau = 1$
Performance metric $p(\hat{\mathbf{Y}}, \mathbf{Y})$

Output: Estimated label matrix $\hat{\mathbf{Y}} \in \{1, \ldots, c\}^{n \times R \times M}$
Classification confidence matrix $\mathbf{Co} \in [0, 1]^{n \times c \times R \times M}$
Classification performance matrix $\mathbf{P} \in \mathbb{R}^{F \times R \times M}$

for $r = 1$ **to** R **do**
  $\omega \leftarrow 0$
  counter $\leftarrow 0$
  while $\omega < 2 \times K \times c$ **AND** *counter* < 100 **do**
    for $i = 1$ **to** n **do**
      $x_i^{fold} \leftarrow$ ceil(unif(0,K)) // assign folds to records
      if *unif(0,1)*$< \frac{\ell}{\ell+\upsilon}$ **then** $x_i^{label} \leftarrow 1$ // assign label status to records
      else $x_i^{label} \leftarrow 0$ // y_i corresponding to $x_i^{label} == 0$ are not deleted
    end
    for $f = 1$ **to** K **do**
      for $k = 1$ **to** c **do**
        if $\exists\ i \mid y_i == k\ \&\ x_i^{label} == 1\ \&\ x_i^{fold} \neq f$ **then** $\omega \leftarrow \omega + 1$
        // check for min. 1 label of each class in each training-set
        if $\exists\ i \mid y_i == k\ \&\ x_i^{fold} == f$ **then** $\omega \leftarrow \omega + 1$
        // check for min. 1 record of each class in each test-set
      end
    end
    counter $\leftarrow$ counter $+ 1$
  end
  for $f = 1$ **to** K **do**
    $\mathcal{T} \leftarrow \{\mathcal{X} | x^{fold} == f\}$ // identify test-set
    $\mathbf{Y}_{fr} \leftarrow \{\mathcal{Y} | x^{fold} == f\}$ // identify test-set labels
    $\mathcal{U} = \{(\mathcal{X} \setminus \mathcal{T}) | x^{label} == 0\}$ // identify unlabeled training records
    $\mathcal{L} = \{(\mathcal{X} \setminus \mathcal{T}) | x^{label} == 1\}$ // identify labeled training records
    $\mathcal{Y}_{fr} \leftarrow \{y | x^{fold} \neq f\}$ // identify training-set labels
    for $y_i \in \mathcal{Y}_{fr}$ **do** $\mathcal{Y}_{fr-i} \leftarrow y_i \times x_i^{label}$ // current training labels
    for $a = 1$ **to** M **do**
      train $\mathfrak{M}_a$ on $\mathcal{L} \cup \mathcal{Y}_{fr} \cup \mathcal{U}$
      $\hat{\mathbf{Y}}_{ra}^{fold==f} \in \{1, \ldots, c\}^{|\mathcal{T}|} \leftarrow$ predicted labels of $\mathcal{T}$ with $\mathfrak{M}_a$
      $\mathbf{Co}_{ra}^{fold==f} \in [0, 1]^{|\mathcal{T}| \times c} \leftarrow$ prediction confidences of $\mathcal{T}$ with $\mathfrak{M}_a$
      $P_{fra} \leftarrow p\left(\hat{\mathbf{Y}}_{ra}^{fold==f}, \mathbf{Y}_{fr}\right)$
    end
  end
end

Hyp.	alg. 1	alg. 2	rej.	cutoff	p-value	wins	losses
H1	[Transduc]	[Freeze]	0	0.0100	0.01182	16	4
H2	[Freeze]	[Ext]	0	0.0500	0.50340	8	12
H3	[Ext]	[Hybrid]	0	0.0167	0.11340	14	6
H4	[Hybrid]	[Simple]	1	0.0083	0.00004	1	19
H5	[Simple]	[Self]	0	0.0250	0.26320	13	7
H6	[Self]	[Induc]	0	0.0125	0.04139	5	15

Table 4.3: Hypothesis testing with binomial sign tests across all data-sets
Columns 1 to 3 show which hypothesis is being tested and the corresponding algorithms
Column 4 shows if the null hypothesis has been rejected
Column 5 shows the cutoff for rejection according to Holm (1979) (global $\alpha = 0.05$)
Column 6 shows the obtained p-value of the bilateral test
Columns 7 and 8 show the number of wins and losses. The number of tries is always 20 and the expected number of wins under the null hypothesis of equally good algorithms 10

H1 The null hypothesis cannot be rejected across data-sets, but only very narrowly so with 16 wins of [Transduc] over [Freeze] against 4 losses. If we look at t-tests on single data-sets, the null hypothesis is rejected only 3 out of 20 times, twice in the direction of our intuition. There is seldom a significant difference between [Freeze] and [Transduc], but perhaps systematically an advantage to [Transduc] which is too small to be significant. Neither the 4 data-sets that [Freeze] does not lose, nor the 3 data-sets with the significant differences have obvious commonalities. In terms of effect size, we do not have a single data-set where [Transduc] would excel and [Freeze] clearly fail or the opposite. [Freeze] remains a strong contender and a valid way to perform out-of-sample prediction in SSL.

H2 With 8 wins to 12 losses, there is no trend across data-sets. Individual data-sets show more interesting results however. [Freeze] has 3 significant wins and [Ext] has 6. Data-sets `g241c` (0.73 to 0.84), `coil` (0.92 to 0.77), `urban` (0.35 to 0.64) and `wilt` (0.78 to 0.97) show very large performance differences. The last three of those data-sets do have either class imbalance or multiple classes and `g241c` is designed to violate the manifold assumption. Such strong differences in both ways indicate that anyone interested in out-of-sample extension with graph-based SSL algorithms should try both approaches on their specific data-set before making a decision.

Both algorithms represent very different approaches to out-of-sample extension. [Freeze] tries to remain as close as possible to [Transduc] and [Ext] combines the graph-based algorithm with many other to compensate respective weaknesses. Computational complexity is also impacted by those differences. If new records arrive in batches, [Freeze] will be more manageable. If new records arrive frequently and individually, [Ext] is more tractable since its complexity resides at training-time, not test-time. Delalleau et al. (2005)'s formula equation (3.10), which makes only slightly more simplifying assumptions than [Freeze], would render graph-freezing with AEW tractable for frequent out-of-sample predictions.

H3 With 14 wins to 6 losses, there is no significant trend across data-sets. The score for significant wins is 7 to 3. Among [Ext]'s wins with the largest margins are 3 artificial data-sets (`g241c`, `g241n`

and `usps`) and `cardio3`. On figure C.4, we see that those are data-sets where [Ext] performed a moderate amount of iterations. This might indicate a slight issue with label reliability in later iterations. GTAM as label propagation technique could make iterations more robust to uncertain labels. It needs to be seen if the computational complexity of GTAM in any but the first iteration compares favorably to LLGC.

H4 With 1 win to 19 losses, the null hypothesis is rejected across data-sets in the direction opposing our intuition. [Hybrid] has no significant win over [Simple]. Considering that the 3 only significant wins of [Hybrid] over [Ext] are also significant losses of [Hybrid] against [Simple], there is no reason to use it. We should either decrease complexity and use [Simple] or increase it and use [Ext].

[Hybrid] is stuck in the middle. This may well be an issue with the θ parameter. We have set it to 0.5 and thereby virtually deactivated it (see the previous section). Looking at figure C.3, we can see no obvious problems. In no case do the amounts of labeled data increase with iterations as they did on the `iris` set. Only with `cardio10`, `coil`, `mice` and `urban` do we have large fractions of never labeled records. Those data-sets contain multiple classes and are among the better performances of [Hybrid]. This is not a bug, but a feature. When there are many classes, even $\theta = 0.5$ plays a role. This would indicate that we should have kept a substantive θ threshold. However, [Simple] performs equally well on those data-sets. The complexity of [Hybrid] would not be justified even if all data-sets behaved like those 4.

H5 With 13 wins to 7 losses, there is no significant trend across data-sets. All the 7 significant differences are wins for [Simple] over [Self]. We should not over-interpret this finding. It might be more of a weakness of [Self] (even compared to [Induc]) than a strength of [Simple]. Comparing figure C.2 with table 4.5, we see that [Simple] can correctly distinguish when it is a good idea to label many records and when not to. Its performance is neither consistently worse on those data-sets where more labels remain unlabeled than get labeled, nor vice versa. The threshold $\theta = 0.85$ works well. This means that the classification confidences as outputted in $\mathbf{F}$ by label propagation are indeed good approximations of label probabilities.

H6 With 5 wins to 15 losses, the null hypothesis fails to be rejected across data-sets. Yet the trend goes in the direction opposite to our intuitions. [Self] has no significant win and 7 significant losses against [Induc]. We cannot recommend self-learning over ignoring of unlabeled data. The issue might once more be parameter θ. Even though, contrary to [Hybrid], we could set a high value for θ and didn't anticipate problems, this parameter is the weak spot of self learning: Contrary to simple there can be many iterations (which do propagate uncertainty) and contrary to [Ext], there is not a multitude of classifiers that need to agree on each label. We should point out that there might be better variants of self-learning than the one we employed. However, Triguero et al. (2015) conclude that no one variant of self learning is systematically the best and that the choice should depend on the data-set.

Related insights [Simple]'s good performance against [Hybrid] and [Self] would indicate a very promising algorithm. Unfortunately, due to our hypothesis design, it has only been tested against

two algorithms that may suffer from parameter tuning issues. It would be interesting to compare it directly with [Ext] to see if the substantial added complexity between the two helps. A direct comparison with [Induc] is of interest for the same reason.

[Hybrid] and [Self] having scored no significant wins, it is not possible to infer conclusions of the type $[Simple] > [Self]$ AND $[Self] > [Induc] \Rightarrow [Simple] > [Induc]$. It would not be statistically sound to add new hypotheses and tests to the lot after having analyzed the results. We are therefore not able to make any statements about statistical significance when comparing [Simple] to [Induc] and to [Ext]. We can at most look at table 4.5 and see in how many cases [Simple] has a higher sample mean than [Ext] or [Induc] a higher sample mean than [Simple]. This would at least tell us with certainty that those are not losses. No statistical test could turn a difference in sample means into a significant difference in the other direction.

[Simple] does in 12 out of 20 cases not lose against [Ext], which is encouraging, but does also in 11 cases not win against [Induc]. This last observation is less a statement about [Simple] than it is one about [Induc], which 14 times out of 20 does not lose against [Ext] and the same against [Transduc]. [Ext] uses multiple algorithms iteratively and [Transduc] has access to the test-set when constructing the graph. Yet either one of them has at most 6 chances out of 20 data-sets to beat an algorithm that ignored over 83% of the training-set and did not have access to the test-set when constructing its model. Among the 6 data-sets where [Transduc] does not necessarily lose against [Induc] are 4 sets specifically selected by Chapelle et al. (2006) for SSL, 2 of which are artificially created. Those other 14 non wins are not necessarily losses but, given the comparative advantages of the algorithms, we would have expected more clear wins than even the maximally possible 6. This warrants further research. We should note, that the 4 data-sets that [Transduc] clearly failed on (`cardio3/10`, `urban` and `wilt`) contained only numeric variables. They did not make a transformation of features necessary for graph sparsification. For AEW, it should have been equally straightforward, but we have ended up scaling all variables since AEW could not optimize its objective function much on the unscaled data. On the other hand, Gower (1971)'s metric seems to integrate well with AEW even though some assumptions needed to be made (38). The heterogeneous data-set `seismic` shows no performance difference between algorithms.

	H1	H2	H3	H4	H5	H6
adult	0	-1	0	-1	0	-1
bank	0	0	1	-1	1	0
banknotes	0	0	0	0	0	0
bci	0	0	0	0	0	0
biodeg	1	0	1	-1	1	0
cardio3	0	-1	1	-1	0	0
cardio10	1	-1	-1	-1	1	-1
coil	0	1	-1	-1	1	0
digit1	0	0	0	0	0	0
eeg	-1	1	-1	-1	1	-1
g241c	0	-1	1	-1	0	0
g241n	1	0	1	-1	0	0
mice	0	0	0	0	0	0
retino	0	0	0	0	0	0
seismic	0	0	0	0	0	-1
spam	0	0	0	0	1	-1
thoracic	0	0	0	0	0	0
urban	0	-1	0	0	0	0
usps	0	1	1	-1	1	-1
wilt	0	-1	1	0	0	-1

(a) Overview of hypothesis testing
Each data-set is subject to Holm's procedure for FWER
(distributing a global $\alpha = 0.05$ among hypotheses)
"1" means that the null hypothesis has been rejected
"-1" means that the null hypothesis has been rejected,
but in the direction opposing our intuition
"0" means that the null hypothesis cannot be rejected

	H1	H2	H3	H4	H5	H6
adult	-	-	+	-	+	-
bank	+	+	+	-	+	-
banknotes	+	-	+	-	+	-
bci	+	-	+	-	+	+
biodeg	+	-	+	-	+	-
cardio3	-	-	+	-	-	-
cardio10	+	-	-	-	+	-
coil	+	+	-	-	+	-
digit1	+	+	+	-	-	+
eeg	-	+	-	-	+	-
g241c	+	-	+	-	-	-
g241n	+	-	+	-	-	+
mice	+	+	+	-	-	+
retino	-	-	+	-	+	-
seismic	+	+	+	-	+	-
spam	+	-	-	-	+	-
thoracic	+	+	-	-	-	+
urban	+	-	-	+	+	-
usps	+	+	+	-	+	-
wilt	+	-	+	-	-	-
#+	16	8	14	1	13	5
#-	4	12	6	19	7	15
$\sum$	12	-4	8	-18	6	-10

(b) Overview of the sign of performance differences
Even the smallest differences are counted
"+" means our intuitions went the right direction
"-" contradicts our intuitions
The counts of $+$ and $-$ are used for statistical testing
$\sum$ provides an *informal* check of our intuitions

Table 4.4: Overview of hypothesis testing over data-sets

		[Transduc]		[Freeze]		[Ext]		[Hybrid]		[Simple]		[Self]		[Induc]	
data-set	metric	$\hat{\mu}$	$\hat{\sigma}$	$\hat{\mu}$	$\hat{\sigma}$	$\hat{\mu}$	$\hat{\sigma}$	$\hat{\mu}$	$\hat{\sigma}$	$\hat{\mu}$	$\hat{\sigma}$	$\hat{\mu}$	$\hat{\sigma}$	$\hat{\mu}$	$\hat{\sigma}$
adult	ROC	0.8304	(0.0106)	0.8384	(0.0122)	0.8692	(0.0081)	0.8641	(0.0086)	0.8848	(0.0073)	0.8821	(0.0082)	0.8968	(0.0056)
bank	ROC	0.7568	(0.0412)	0.7550	(0.0398)	0.7351	(0.0374)	0.6933	(0.0476)	0.7566	(0.0406)	0.7150	(0.0421)	0.7393	(0.0383)
banknotes	ROC	1.0000	(0.0000)	0.9999	(0.0004)	0.9999	(0.0002)	0.9963	(0.0061)	0.9994	(0.0024)	0.9923	(0.0113)	0.9976	(0.0040)
bci	ROC	0.5166	(0.0821)	0.5089	(0.0917)	0.5499	(0.0983)	0.5426	(0.1041)	0.5831	(0.0889)	0.5750	(0.1019)	0.5734	(0.0880)
biodeg	ROC	0.8617	(0.0377)	0.8478	(0.0370)	0.8808	(0.0321)	0.8335	(0.0442)	0.8943	(0.0358)	0.8717	(0.0402)	0.8937	(0.0371)
cardio3	F	0.5857	(0.0660)	0.5918	(0.0837)	0.7034	(0.0573)	0.4831	(0.0841)	0.7637	(0.0446)	0.7731	(0.0443)	0.7947	(0.0418)
cardio10	F	0.4354	(0.0415)	0.3868	(0.0291)	0.4949	(0.0420)	0.6127	(0.0559)	0.6969	(0.0493)	0.6582	(0.0534)	0.6896	(0.0498)
coil	F	0.9211	(0.0262)	0.9157	(0.0272)	0.7735	(0.0514)	0.8522	(0.0392)	0.8831	(0.0284)	0.8567	(0.0368)	0.8663	(0.0395)
digit1	ROC	0.9990	(0.0010)	0.9984	(0.0020)	0.9969	(0.0020)	0.9831	(0.0280)	0.9902	(0.0066)	0.9934	(0.0049)	0.9897	(0.0063)
eeg	ROC	0.7828	(0.0116)	0.7953	(0.0150)	0.7779	(0.0117)	0.8881	(0.0101)	0.9374	(0.0055)	0.8831	(0.0097)	0.9371	(0.0057)
g241c	ROC	0.7639	(0.0368)	0.7330	(0.0398)	0.8438	(0.0343)	0.6068	(0.0525)	0.7933	(0.0327)	0.7950	(0.0313)	0.8002	(0.0356)
g241n	ROC	0.8137	(0.0485)	0.7533	(0.0461)	0.7996	(0.0679)	0.6064	(0.0554)	0.7686	(0.0414)	0.7726	(0.0357)	0.7696	(0.0429)
mice	F	0.7794	(0.0607)	0.7735	(0.0492)	0.7609	(0.0438)	0.7454	(0.0621)	0.7912	(0.0530)	0.7968	(0.0511)	0.7944	(0.0505)
retino	ROC	0.6387	(0.0504)	0.6392	(0.0487)	0.6987	(0.0387)	0.6548	(0.0560)	0.6889	(0.0418)	0.6831	(0.0370)	0.6908	(0.0369)
seismic	ROC	0.7188	(0.0702)	0.7005	(0.0744)	0.6922	(0.0624)	0.6441	(0.0809)	0.6893	(0.0808)	0.6742	(0.0778)	0.7334	(0.0659)
spam	ROC	0.9414	(0.0103)	0.9364	(0.0162)	0.9397	(0.0097)	0.9445	(0.0117)	0.9519	(0.0095)	0.9391	(0.0102)	0.9525	(0.0086)
thoracic	ROC	0.5610	(0.1424)	0.5329	(0.1465)	0.5320	(0.1287)	0.5614	(0.1099)	0.5697	(0.1015)	0.5776	(0.1285)	0.5709	(0.1119)
urban	F	0.4189	(0.1171)	0.3482	(0.0570)	0.6369	(0.0728)	0.7246	(0.0831)	0.7205	(0.0913)	0.6784	(0.0831)	0.7256	(0.0925)
usps	ROC	0.9840	(0.0124)	0.9810	(0.0149)	0.9413	(0.0281)	0.8253	(0.0424)	0.9227	(0.0267)	0.8686	(0.0328)	0.9177	(0.0264)
wilt	ROC	0.7923	(0.0472)	0.7832	(0.0530)	0.9702	(0.0118)	0.9136	(0.0352)	0.9275	(0.0280)	0.9433	(0.0335)	0.9762	(0.0185)

Table 4.5: Overview of sample means $\hat{\mu}$ and sample variances $\hat{\sigma}$ of the performance metrics.
Each cell is based on 10×3 values from the folds and repetitions. t-tests are based on these underlying values in matched pairs.
This table provides an informal impression of which data-set was predicted how well by which algorithm.

Chapter 5

Final discussion, conclusions and future work

Our main research question was whether graph-based semi-supervised classification algorithms can be extended to out-of-sample prediction. The recently proposed adaptive edge weighting technique received special attention since on the one hand, it promises increased classification performance and on the other hand, its parsimonious representation of edge weights through the parameters σ simplifies out-of-sample prediction.

We have tested two main approaches to out-of-sample prediction with graph-based algorithms. Firstly, algorithm [Freeze] computes optimized parameters σ and connects yet unseen data-points using the frozen parameters and the similarity between old- and new nodes. It stays within the framework of graph-based learning and makes only minor adjustments.

Secondly, we can combine a transductive graph-based algorithm and one or more inductive supervised algorithms, possibly in iterative fashion, into meta-algorithms. The final induction of the unseen test-set can be performed by the inductive algorithms. Before that, the algorithms can learn from each other and ideally mutually compensate their weaknesses.

We must also discuss another result. We have compared aforementioned (meta-)algorithms with the simple inductive supervised scenario. Our expectation was that [Induc] would be consistently outperformed by the other algorithms since it ignores most of the training data. This was not the case, casting doubts on performance of semi-supervised classification that need to be addressed.

5.1 Synthesis of the results

The results obtained by [Induc] are a reality check with regard to the usefulness of unlabeled training data. Provided the manifold- and smoothness assumptions are met, unlabeled records should improve classification. According to our results however, this happens at most on data-sets specifically selected and designed for semi-supervised learning. Since those foundational assumptions are stated in rather non-technical, intuitive terms, they cannot formally be tested. [1] The best, and in absence of domain knowledge perhaps only, way to test those assumptions of graph-based semi-supervised learning may be to actually compare performance with inductive supervised algorithms

[1] Testing the smoothness assumption would also necessarily require label information which can be scarce.

on a given data-set. This baseline comparison, recommended by Chapelle et al. (2006, ch. 4), is not often done. Karasuyama and Mamitsuka (2013a, 2016) did not perform one when introducing the new AEW method. Still, the authors insisted on the manifold assumption and selected real world data-sets which contain high dimensional representations of images where it is known to be reasonable. In our experiments however, not even data-sets `mice`, `urban` and `spam` (containing high-dimensional sensory- and text-data) have [Transduc] outperform [Induc]. This would indicate that the manifold assumption constrains the applicability of graph-based learning to only a few select data-sets. We feel that the literature does not insist enough on these limitations. In their review, W. Liu et al. (2012) do not mention understanding and prediction of circumstances in which graph-based learning performs well as an area of ongoing research.

If our observations are representative, graph-based learning is not very adequate for business use-cases like the gym example from the introduction. Data used for business decisions like promotional offerings, targeted advertising or attrition prevention is typically not high-dimensional and homogeneous. All variables are not measured on the same scale like pixels on an image. Instead, businesses have socio-economic data about their customers and log files of their purchasing history.

Furthermore, coming back to our cancer example, the general public will likely see semi-supervised algorithms as obscure. Substantial decisions by such algorithms will only be accepted if they consistently and clearly outperform more intuitive alternatives like the supervised decision trees that we used in our experiments. A utilitarian stance in favor of complex algorithms is not defensible when performance comparisons yield the results we have observed.

While the most likely cause for the observed results is that the manifold assumption restricts the application of graph-based algorithms to a far smaller domain than we thought, some caveats are in order. Firstly, it might be possible that n was so large in our data-sets, that even the labeled training records $\mathcal{L}$ were enough to learn everything there is to learn from training data. This may be the case for some data-sets where all algorithms perform similarly well.

Secondly, it is imaginable that n was too small for semi-supervised algorithms to develop their advantage. A large number of records populates the manifolds more densely and makes propagation easier. If that was true, most SSL papers would introduce new algorithms testing them only on inadequately small data-sets. The size of our data-sets is not below average in this field of research.

Finally, the graph-based algorithms might have suffered from a lack of feature engineering. We have established on p. 17 that this can be an issue and have not performed much feature engineering on our data-sets. If this was the reason, it would be an interesting finding in itself that graph-based algorithms are more vulnerable to lack of feature engineering than random forests are. [2]

Coming back to our original research question of out-of-sample prediction, we can still draw conclusions. Our 20 data-sets in section 4.2 are chosen to represent a balanced mix of application scenarios. This includes 6 high-dimensional image data-sets designed for SSL, plus the aforementioned `mice`, `urban` and `spam`. Other sets containing lower-dimensional ($d < 50$) and heterogeneous data may not play as much to the strengths of graph-based learning. Even if [Transduc] does not outperform [Induc], it often performs similarly well. We can still compare [Transduc] to the different out-of-sample extensions to see whether they can at least keep that level of performance.

[2] Implementation errors of graph-based algorithms can be excluded as an explanation. On easily predictable data-sets like `banknotes`, `usps` and `digit1`, all graph-based algorithms have near perfect scores. That would not happen with flawed code.

We have two distinct approaches to out-of-sample prediction. We can use only a graph-based algorithm or we can use combinations of supervised inductive- and semi-supervised transductive algorithms. Our results have shown that another possibility, self-learning, is unreliable and probably prone to parameter tuning issues. We rule it out.

Firstly, we cannot use [Transduc] directly, unless we relaunched it from scratch every time new records need to be classified. Especially if this includes an AEW optimization, the computational costs are prohibitive. [Freeze] allows out-of-sample prediction with only minimal adjustments to [Transduc]. The parsimonious representation of edge weights in σ through AEW makes this possible. [3] If new records arrive frequently and need to be classified right away, equation (3.10) would speed computations up by making one more simplifying assumption. We have not yet tested how this second assumption impacts classification performance, but [Freeze] is almost on par with [Transduc].

Secondly, meta-algorithms, we have started with Govada et al. (2015)'s suggestion to use [Hybrid] that iteratively predicts unlabeled training records with an inductive and a transductive algorithm and adds labels when those are confidently agreed upon. The final test-set prediction is done inductively without the graph-based algorithm, thus ignoring the remaining unlabeled records. Contrary to the author's conclusions, we suspect that the parameter θ, which represents the minimum label confidence in order to assign this label for the next iteration(s), plays a crucial role. It was difficult to tune this parameter and [Hybrid]'s performance disappointed. However, we also included an extension and a simplification to [Hybrid] into our experimental setup. [Ext] uses multiple algorithms for multiple iterations. An agreement proportion threshold among algorithms avoids the parameter tuning issues. It sometimes outperformed [Freeze], indicating that the contained algorithms may compensate their respective weaknesses. [Simple] on the other hand uses only one transductive algorithm once to feed additional labels to the inductive algorithm which then directly predicts the test-set and also performed well. The options for out-of-sample prediction in the presence of unlabeled training data can be synthesized as in figure 5.1.

5.2 Limitations and constraints

The first constraint to be mentioned is computational complexity. Given our resources, we could use at most a few ten thousand records. We had to sample random parts from data-set `adult` and take the smaller version of `spam`. The entire data-sets would not have fit in RAM and also have taken days or weeks to compute. At our scale, AEW has proven to be the computational bottleneck. This might change at larger scales since AEW's temporal complexity increases only linearly with n whereas graph sparsification and label propagation compute over matrices in $\mathbb{R}^{n \times n}$. It may then be necessary to re-implement Gower's dissimilarity to compute selective distances so that it is compatible with kd-trees for faster graph sparsification. Approximate nearest neighbor search based on hashing may be an alternative as well. Especially since AEW can tolerate receiving superfluous neighbors, it may be better to sparsify a little less strictly with a tractable ANN search. For label propagation, Delalleau et al. (2005)'s formulation of LLGC which yields equal results to D. Zhou et al. (2004) and is more difficult to implement but computes faster should be used.

[3] The alternative Gaussian kernels represent edge weights exactly as parsimoniously, but do not provide as principled an approach to optimize them. With LLE on the other hand, there are as many parameters as there are edges.

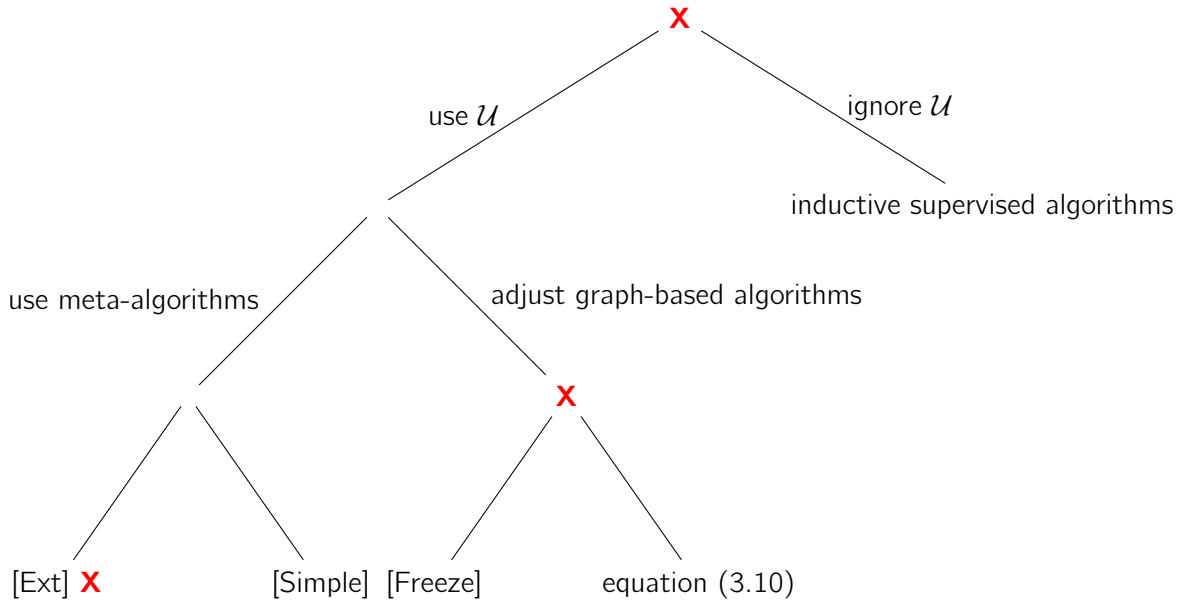


Figure 5.1: Algorithmic choices when facing out-of-sample prediction and unlabeled data
Should we use unlabeled records $\mathcal{U}$ or can we ignore them?
If yes, should we use meta-algorithms or adjust graph-based algorithms? Which variants?
The red **X** indicates an opportunity for future research

Availability of sufficiently large and relevant data-sets an issue. The best data-sets seem to be confidential property of corporations. Had we had more than 20, perhaps 30 or 40, the binomial tests across data-sets might have yielded more conclusive results. On the other hand, it would have made hypothesis testing on a data-set per data-set basis more tedious.

Statistical soundness has imposed limitations on our experimental setup. In order to use t-tests on classification performances from repeated cross-validation, we need to use corrected resampled t-tests. This already reduces the power of those tests. Furthermore, the need to correct for FWER constrains the amount of variants- and combinations of algorithms that we can test against each other. With seven algorithms, we already had to limit ourselves to comparisons only along a lineup and thus 6 hypotheses instead of 21 when comparing all algorithms among each other. This has limited the types of conclusions that we can statistically draw based on the data. Including more variants would not only have made the analysis of results tedious, it also lowers statistical power since the FWER allocates less of α to each test.

We were able to overcome the difficulties in implementing analytic gradients of the AEW objective function by circumventing them. As we have shown in section 3.1.2, numeric gradients through central differences are as tractable and reliable as their analytic counterparts.

5.3 Future work

We see three concrete research opportunities that can be realized employing only modest resources, a fourth opportunity that requires much more resources and an open question.

The question in which circumstances more unlabeled records improve classification deserves attention. An experimental setup could look as follows: We keep random forests as inductive

algorithm. We increase the ratio between labeled and unlabeled records from 5, as was used in this master thesis, to 10 or 20. We collect more high-dimensional data-sets that supposedly conform to the manifold assumption, including larger ones. We implement other variants of graph-based learning, potentially including b-matching and kNNG for sparsification, LLE and AEW for edge weighting, LLGC, HGF, GTAM and MP for label propagation. Each variant learns transductively on the whole data-set and is directly compared with random forests. Those inductive supervised random forests not only ignore the test-set while training, but also the large unlabeled part of the training-set. If results are similar to those obtained here, they should urge researchers to include a reality check against an inductive supervised algorithm in future studies.

Regarding meta-algorithm [Ext], we could test GTAM instead of LLGC for label propagation. We discovered this insight too late to relaunch all experiments and include it in this master thesis. GTAM improves robustness to unreliable labels in label propagation and seems a natural candidate to be combined with the iterative algorithm [Ext]. At later iterations, there will be potentially unreliable labels to propagate, those that stem from earlier iterations. Once this foreseeable objection to [Ext] is addressed in this way, the technique would add to the currently available solutions for out-of-sample prediction in SSL. Putting the issue whether unlabeled data are helpful in general aside, [Ext] avoids problems with parameter tuning and has fared well against its competitor [Freeze]. To provide a reference point, it should also be tested against [Transduc]. [Simple] can be presented as a far less computationally costly alternative. [4]

Regarding algorithm [Freeze], we could compare it against Delalleau et al. (2005)'s formula equation (3.10) which integrates well with AEW. It would need to be tested if equation (3.10) extends well to multi-class prediction. [Freeze] makes one less assumption than equation (3.10). It recomputes all lines of $\mathbf{F}$ each time it predicts new records and only freezes σ. That makes it slower, but does it also perform better than equation (3.10) which freezes $\mathbf{F}$ and σ? To provide a reference point, both variants of graph freezing should be tested against [Transduc].

There is always the possibility to test the presented algorithms at a larger scale. For example, [Simple] can be used to feed labels to deep convolutional neural networks. Those inductive supervised algorithms are currently among the best if not the best performing classifier for image data (LeCun et al., 2015). However, they need lots of training samples to tune their numerous parameters and labeled image data is much more scarce than unlabeled images. Labeling of images by humans into many classes is expensive and also prone to errors. Image data is usually assumed to respect the manifold assumption, as confirmed by our experiments where [Transduc] fared comparatively well on such data-sets. Therefore, the graph-based algorithm within [Simple] can feed reliable labels to the the inductive neural network. Both algorithms would be very computationally expensive in such a big data scenario.

We were not able to identify principled criteria for choosing between [Freeze], [Ext] and [Simple]. It is not sure that such criteria exist. We suspect that it depends strongly on the data-set and needs to be tried every time before deciding. If there was a more formal test for the manifold- and smoothness assumption, passing this test would make a data-set likely to work well with [Freeze] and failing it would indicate to use the meta-algorithms instead, or only inductive supervised learning.

[4] [Simple] simple does not benefit from GTAM since it uses label propagation only once.

Bibliography

Antal, B. & Hajdu, A. (2014). An ensemble-based system for automatic screening of diabetic retinopathy. *Knowledge-Based Systems*, *60*, 20–27.

Arnsperger, C. & van Parijs, P. (2003). *Éthique économique et sociale*. Collection Repères. La Découverte.

Ashbrook, T., Veloso, M., & Dietterich, T. (2015, July). Managing the artificial intelligence risk. Retrieved from http://www.wbur.org/onpoint/2015/07/16/artificial-intelligence-risk-reward-elon-musk

Ayodele, T. O. (2010). *Types of machine learning algorithms*. INTECH Open Access Publisher.

Ayres-de-Campos, D. (2000). Sisporto 2.0: a program for automated analysis of cardiotocograms. *Journal of Maternal-Fetal Medicine*, *9*(5), 311–318.

Bashiri, M. & Geranmayeh, A. F. (2011). Tuning the parameters of an artificial neural network using central composite design and genetic algorithm. *Scientia Iranica*, *18*(6), 1600–1608.

Beer, K. (2016, May). Macht der algorithmen. Retrieved from http://www.heise.de/newsticker/meldung/heiseshow-Ab-12-Uhr-live-zur-Macht-der-Algorithmen-3218013.html

Belkin, M. & Niyogi, P. (2001). Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Nips* (Vol. 14, pp. 585–591).

Belkin, M. & Niyogi, P. (2002). Using manifold stucture for partially labeled classification. In *Advances in neural information processing systems* (pp. 929–936).

Belkin, M. & Niyogi, P. (2008). Towards a theoretical foundation for laplacian-based manifold methods. *Journal of Computer and System Sciences*, *74*(8), 1289–1308.

Belkin, M., Niyogi, P., & Sindhwani, V. (2005). On manifold regularization. In *Aistats*.

Belkin, M., Niyogi, P., & Sindhwani, V. (2006). Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. *The Journal of Machine Learning Research*, *7*, 2399–2434.

Bengio, I. G. Y. & Courville, A. (2016). *Deep learning*. Book in preparation for MIT Press. Retrieved from http://www.deeplearningbook.org

Bengio, Y., Delalleau, O., & Roux, N. L. (2006). Label propagation and quadratic criterion. In *Semi-supervised learning* (pp. 193–216). Adaptive computation and machine learning. MIT Press. Retrieved from http://www.acad.bg/ebook/ml/MITPress-%20SemiSupervised%20Learning.pdf

Beyer, K., Goldstein, J., Ramakrishnan, R., & Shaft, U. (1999). When is "nearest neighbor" meaningful? In *International conference on database theory* (pp. 217–235). Springer.

Blum, A. & Chawla, S. (2001). Learning from labeled and unlabeled data using graph mincuts.

Borchert, D. (2006). *Encyclopedia of philosophy*. Thomson Gale.

Bouckaert, R. R. & Frank, E. (2004). Evaluating the replicability of significance tests for comparing learning algorithms. In *Pacific-asia conference on knowledge discovery and data mining* (pp. 3–12). Springer.

Boykov, Y., Veksler, O., & Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, *23*(11), 1222–1239.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

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

Chapelle, O., Scholkopf, B., & Zien, A. (2006). *Semi-supervised learning*. Adaptive computation and machine learning. MIT Press. Retrieved from http://www.acad.bg/ebook/ml/MITPress-%20SemiSupervised%20Learning.pdf

Chen, J. & Liu, Y. (2011). Locally linear embedding: a survey. *Artificial Intelligence Review*, *36*(1), 29–48.

Cheng, H., Liu, Z., & Yang, J. (2009). Sparsity induced similarity measure for label propagation. In *Computer vision, 2009 ieee 12th international conference on* (pp. 317–324). IEEE.

Chung, F. R. (1997). *Spectral graph theory*. American Mathematical Soc.

Cochran, W. (1977). *Sampling techniques*. Wiley series in probability and mathematical statistics: Applied probability and statistics. Wiley. Retrieved from http://hbanaszak.mjr.uw.edu.pl/StatRozw/Books/Cochran_1977_Sampling%20Techniques.pdf

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*.

Daitch, S. I., Kelner, J. A., & Spielman, D. A. (2009). Fitting a graph to vector data. In *Proceedings of the 26th annual international conference on machine learning* (pp. 201–208). ACM.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American society for information science*, *41*(6), 391.

Delalleau, O., Bengio, Y., & Le Roux, N. (2005). Efficient non-parametric function induction in semi-supervised learning. In *Aistats* (Vol. 27, *28*, p. 100).

Demiriz, A., Bennett, K. P., & Embrechts, M. J. (1999). Semi-supervised clustering using genetic algorithms. *Artificial neural networks in engineering (ANNIE-99)*, 809–814.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, *7*(Jan), 1–30.

Deza, M. & Deza, E. (2014). *Encyclopedia of distances*. Springer Berlin Heidelberg.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, *10*(7), 1895–1923.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine learning*, *40*(2), 139–157.

Dunn, O. J. (1961). Multiple comparisons among means. *Journal of the American Statistical Association*, *56*(293), 52–64.

Dy, J. G. & Brodley, C. E. (2004). Feature selection for unsupervised learning. *Journal of machine learning research*, *5*(Aug), 845–889.

Efron, B. (1983). Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American Statistical Association*, *78*(382), 316–331.

Efron, B. & Tibshirani, R. (1997). Improvements on cross-validation: the 632+ bootstrap method. *Journal of the American Statistical Association*, *92*(438), 548–560.

Eichelberger, R. K. & Sheng, V. S. (2013). Does one-against-all or one-against-one improve the performance of multiclass classifications? In *Proceedings of the twenty-seventh aaai conference on artificial intelligence* (pp. 1609–1610). AAAI Press.

Elhamifar, E. & Vidal, R. (2011). Sparse manifold clustering and embedding. In *Advances in neural information processing systems* (pp. 55–63).

Eppstein, D., Paterson, M. S., & Yao, F. F. (1997). On nearest-neighbor graphs. *Discrete & Computational Geometry*, *17*(3), 263–282.

Fatourechi, M., Ward, R. K., Mason, S. G., Huggins, J., Schlögl, A., & Birch, G. E. (2008). Comparison of evaluation metrics in classification applications with imbalanced datasets. In *Machine learning and applications, 2008. icmla'08. seventh international conference on* (pp. 777–782). IEEE.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, *27*(8), 861–874.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, *7*(2), 179–188.

Floudas, C. & Pardalos, P. (2008). *Encyclopedia of optimization*. Encyclopedia of Optimization. Springer.

Forman, G. & Scholz, M. (2010). Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. *ACM SIGKDD Explorations Newsletter*, *12*(1), 49–57.

Fouss, F., Francoisse, K., Yen, L., Pirotte, A., & Saerens, M. (2012). An experimental investigation of kernels on graphs for collaborative recommendation and semisupervised classification. *Neural networks*, *31*, 53–72.

Fremuth-Paeger, C. (2016). Goblin graph library. Retrieved from http://goblin2.sourceforge.net/

Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, *3*(3), 209–226.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine learning*, *29*(2-3), 131–163.

Fu, K., Gu, I. Y., Gong, C., & Yang, J. (2016). Robust manifold-preserving diffusion-based saliency detection by adaptive weight construction. *Neurocomputing*, *175, Part A*, 336–347. doi:http://dx.doi.org/10.1016/j.neucom.2015.10.066

Garcia, S. & Herrera, F. (2008). An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, *9*(Dec), 2677–2694.

Gionis, A., Indyk, P., Motwani, R., et al. (1999). Similarity search in high dimensions via hashing. In *Vldb* (Vol. 99, *6*, pp. 518–529).

Gong, C., Tao, D., Maybank, S. J., Liu, W., Kang, G., & Yang, J. (2016). Multi-modal curriculum learning for semi-supervised image classification. *IEEE Transactions on Image Processing*, *25*(7), 3249–3260.

Govada, A., Joshi, P., Mittal, S., & Sahay, S. K. (2015). Hybrid approach for inductive semi supervised learning using label propagation and support vector machine. In *International workshop on machine learning and data mining in pattern recognition* (pp. 199–213). Springer.

Gower, J. C. (1971). A general coefficient of similarity and some of its properties. *Biometrics*, 857–871.

Grau, C. (2011). There is no "i" in "robot": robots and utilitarianism. In M. Anderson & S. Anderson (Eds.), *Machine ethics* (Chap. 25, pp. 451–463). Cambridge University Press.

Gretton, A., Borgwardt, K. M., Rasch, M., Schölkopf, B., & Smola, A. J. (2006). A kernel method for the two-sample-problem. In *Advances in neural information processing systems* (pp. 513–520).

Han, J., Kamber, M., & Pei, J. (2011). *Data mining: concepts and techniques* (3rd). The Morgan Kaufmann Series in Data Management Systems. Elsevier Science.

Harris, S. (2016, June). Can we build ai without losing control over it? Retrieved from http://www.ted.com/talks/sam_harris_can_we_build_ai_without_losing_control_over_it/transcript?language=en

Hastie, T. [T.], Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer series in statistics. Springer.

Hastie, T. [Trevor], Tibshirani, R. et al. (1998). Classification by pairwise coupling. *The annals of statistics*, *26*(2), 451–471.

Hazewinkel, M. (1997). *Encyclopaedia of mathematics*. Encyclopaedia of Mathematics. Springer.

Henderson, H. (2009). *Encyclopedia of computer science and technology*. Facts On File, Incorporated.

Higuera, C., Gardiner, K. J., & Cios, K. J. (2015). Self-organizing feature maps identify proteins critical to learning in a mouse model of down syndrome. *PloS one*, *10*(6), e0129126.

Ho, T. K. (1995). Random decision forests. In *Document analysis and recognition, 1995., proceedings of the third international conference on* (Vol. 1, pp. 278–282). IEEE.

Hofmann, T. (1999). Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international acm sigir conference on research and development in information retrieval* (pp. 50–57). ACM.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, 65–70.

Hsu, C.-W. & Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, *13*(2), 415–425.

Huang, B. C. & Jebara, T. (2007). Loopy belief propagation for bipartite maximum weight b-matching. In *International conference on artificial intelligence and statistics* (pp. 195–202).

Huang, B. C. & Jebara, T. (2011). Fast b-matching via sufficient selection belief propagation. In *Aistats* (pp. 361–369).

Hughes, G. (1968). On the mean accuracy of statistical pattern recognizers. *IEEE transactions on information theory*, *14*(1), 55–63.

Izenman, A. (2009). *Modern multivariate statistical techniques: regression, classification, and manifold learning*. Springer Texts in Statistics. Springer New York.

Jaakkola, M. S. T. & Szummer, M. (2002). Partially labeled classification with markov random walks. *Advances in neural information processing systems (NIPS)*, *14*, 945–952.

Jaccard, P. (1912). The distribution of the flora in the alpine zone. *New phytologist*, *11*(2), 37–50.

Jain, A. & Dubes, R. (1988). *Algorithms for clustering data*. Prentice-Hall Advanced Reference Series. Prentice Hall PTR. Retrieved from http://homepages.inf.ed.ac.uk/rbf/BOOKS/JAIN/Clustering_Jain_Dubes.pdf

Jain, P., Netrapalli, P., & Sanghavi, S. (2013). Low-rank matrix completion using alternating minimization. In *Proceedings of the forty-fifth annual acm symposium on theory of computing* (pp. 665–674). ACM.

Japkowicz, N. & Shah, M. (2011). *Evaluating learning algorithms: a classification perspective*. Cambridge University Press. Retrieved from https://books.google.be/books?id=VoWIIOKVzR4C

Jebara, T., Wang, J., & Chang, S.-F. (2009). Graph construction and b-matching for semi-supervised learning. In *Proceedings of the 26th annual international conference on machine learning* (pp. 441–448). ACM.

Jegou, H., Harzallah, H., & Schmid, C. (2007). A contextual dissimilarity measure for accurate and efficient image search. In *Computer vision and pattern recognition, 2007. cvpr'07. ieee conference on* (pp. 1–8). IEEE.

Joachims, T. et al. (2003). Transductive learning via spectral graph partitioning. In *Icml* (Vol. 3, pp. 290–297).

Johnson, B. A., Tateishi, R., & Hoan, N. T. (2013). A hybrid pansharpening approach and multiscale object-based image analysis for mapping diseased pine and oak trees. *International journal of remote sensing*, *34*(20), 6969–6982.

Johnson, B. & Xie, Z. (2013). Classifying a high resolution image of an urban area using super-object information. *ISPRS journal of photogrammetry and remote sensing*, *83*, 40–49.

Karasuyama, M. & Mamitsuka, H. (2013a). Manifold-based similarity adaptation for label propagation. In *Advances in neural information processing systems* (pp. 1547–1555).

Karasuyama, M. & Mamitsuka, H. (2013b). Multiple graph label propagation by sparse integration. *Neural Networks and Learning Systems, IEEE Transactions on*, *24*(12), 1999–2012.

Karasuyama, M. & Mamitsuka, H. (2016). Adaptive edge weighting for graph-based learning algorithms. *Machine Learning*, 1–29.

Kaufman, L. & Rousseeuw, P. (1990). *Finding groups in data: an introduction to cluster analysis*. Wiley Series in Probability and Statistics. Wiley.

Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. *Ijcai*, *14*(2), 1137–1145.

Kohavi, R. (1996). Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In *Kdd* (Vol. 96, pp. 202–207). Citeseer.

Kohavi, R. & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, *97*(1–2), 273–324. Relevance. doi:http://dx.doi.org/10.1016/S0004-3702(97)00043-X

Kong, D., Ding, C. H., Huang, H., & Nie, F. (2012). An iterative locally linear embedding algorithm. *arXiv preprint arXiv:1206.6463*.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, *7*(1), 48–50.

Kuhn, M. (2015). Applied predictive modeling meet up event. Retrieved from https://www.youtube.com/watch?v=dB-JHhEJvQA

Kuhn, M. & Johnson, K. (2013). *Applied predictive modeling*. SpringerLink : Bücher. Springer New York. Retrieved from http://appliedpredictivemodeling.com/

Landgrebe, T. C. & Duin, R. P. (2007). Approximating the multiclass roc by pairwise analysis. *Pattern recognition letters*, *28*(13), 1747–1758.

Lavrenko, V. & Goddard, N. (2014). Decision tree 1: how it works. Retrieved from https://www.youtube.com/watch?v=eKD5gxPPeY0

LeCun, Y. (2016, April). Yann le cun - chercheur en intelligence artificielle. Retrieved from https://www.franceinter.fr/emissions/les-savanturiers/les-savanturiers-17-avril-2016

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Lewis, A. S. & Overton, M. L. (2009). Nonsmooth optimization via bfgs. *Submitted to SIAM J. Optimiz*, 1–35.

Li, Y.-F. & Zhou, Z.-H. (2015). Towards making unlabeled data never hurt. *IEEE transactions on pattern analysis and machine intelligence*, *37*(1), 175–188.

Li, S. & Fu, Y. (2013). Low-rank coding with b-matching constraint for semi-supervised classification. In *Ijcai*.

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

Lichman, M. (2013). UCI machine learning repository. Retrieved from http://archive.ics.uci.edu/ml

Lin, Z., Chen, M., & Ma, Y. (2010). The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *arXiv preprint arXiv:1009.5055*.

Liu, W., He, J., & Chang, S.-F. (2010). Large graph construction for scalable semi-supervised learning. In *Proceedings of the 27th international conference on machine learning (icml-10)* (pp. 679–686).

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

Lombardi, C. M. & Hurlbert, S. H. (2009). Misprescription and misuse of one-tailed tests. *Austral Ecology*, *34*(4), 447–468.

Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., & Hornik, K. (2016). *Cluster: cluster analysis basics and extensions*. R package version 2.0.4 — For new features, see the 'Changelog' file (in the package source).

Maier, M., Luxburg, U. V., & Hein, M. (2008). Influence of graph construction on graph-based clustering measures. In *Advances in neural information processing systems* (pp. 1025–1032).

Mansouri, K. (2013). Quantitative structure–activity relationship models for ready biodegradability of chemicals. *Journal of chemical information and modeling*, *53*(4), 867–878.

Moro, S., Cortez, P., & Rita, P. (2014). A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, *62*, 22–31.

Nadeau, C. & Bengio, Y. (2003). Inference for the generalization error. *Machine Learning*, *52*(3), 239–281.

Ng, A. Y., Jordan, M. I., Weiss, Y., et al. (2002). On spectral clustering: analysis and an algorithm. *Advances in neural information processing systems*, *2*, 849–856.

Niesen, U., Shah, D., & Wornell, G. W. (2009). Adaptive alternating minimization algorithms. *IEEE Transactions on Information Theory*, *55*(3), 1423–1429.

Nigam, K. P. (2001). *Using unlabeled data to improve text classification* (Doctoral dissertation, Massachusetts Institute of Technology).

Oesterheld, C. (2015, May). Machine ethics and preference utilitarianism. Retrieved from http://reducing-suffering.org/machine-ethics-and-preference-utilitarianism/

Orbach, M. & Crammer, K. (2012). Transductive phoneme classification using local scaling and confidence. In *Electrical & electronics engineers in israel (ieeei), 2012 ieee 27th convention of* (pp. 1–5). IEEE.

Oshiro, T. M., Perez, P. S., & Baranauskas, J. A. (2012). How many trees in a random forest? In *International workshop on machine learning and data mining in pattern recognition* (pp. 154–168). Springer.

Padberg, M. W. & Rao, M. R. (1982). Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, *7*(1), 67–80.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, *2*(11), 559–572.

Pei, X., Lyu, Z., Chen, C., & Chen, C. (2015). Manifold adaptive label propagation for face clustering. *Cybernetics, IEEE Transactions on*, *45*(8), 1681–1691.

Peng, Y., Lu, B.-L., & Wang, S. (2015). Enhanced representation via sparse manifold adaption for semi-supervised learning. *Neural Networks*, *65*, 1–17.

Podani, J. (1999). Extending gower's general coefficient of similarity to ordinal characters. *Taxon*, 331–340.

Radovanović, M., Nanopoulos, A., & Ivanović, M. (2010). Hubs in space: popular nearest neighbors in high-dimensional data. *Journal of Machine Learning Research*, *11*(Sep), 2487–2531.

Rardin, R. (1998). *Optimization in operations research*. Pearson Education, Limited.

Rios, L. M. & Sahinidis, N. V. (2013). Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, *56*(3), 1247–1293.

Rohrer, B. B. (2016). Machine learning algorithm cheat sheet for microsoft azure machine learning studio v6. Retrieved from https://azure.microsoft.com/en-us/documentation/articles/machine-learning-algorithm-cheat-sheet/

Roweis, S. T. & Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, *290*(5500), 2323–2326.

Salzberg, S. L. (1997). On comparing classifiers: pitfalls to avoid and a recommended approach. *Data mining and knowledge discovery*, *1*(3), 317–328.

Salzberg, S. L. (1999). On comparing classifiers: a critique of current research and methods. *Data Mining and Knowledge Discovery*, *1*, 1–12.

Sammut, C. & Webb, G. (2011). *Encyclopedia of machine learning*. Springer.

Sanghavi, S., Malioutov, D., & Willsky, A. (2008). Networking sensors using belief propagation. In *Communication, control, and computing, 2008 46th annual allerton conference on* (pp. 384–391). IEEE.

Saul, L. K. & Roweis, S. T. (2003). Think globally, fit locally: unsupervised learning of low dimensional manifolds. *The Journal of Machine Learning Research*, *4*, 119–155.

Segal, M. R. (2004). Machine learning benchmarks and random forest regression. *Center for Bioinformatics & Molecular Biostatistics*.

Shahshahani, B. M. & Landgrebe, D. A. (1994). The effect of unlabeled samples in reducing the small sample size problem and mitigating the hughes phenomenon. *IEEE Transactions on Geoscience and remote sensing*, *32*(5), 1087–1095.

Shawe-Taylor, J. & Cristianini, N. (2004). *Kernel methods for pattern analysis*. Kernel Methods for Pattern Analysis. Cambridge University Press.

Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., ... Moore, R. (2013). Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, *56*(1), 116–124.

Sikora, M. & Wróbel, Ł. (2010). Application of rule induction algorithms for analysis of data collected by seismic hazard monitoring systems in coal mines. *Archives of Mining Sciences*, *55*(1), 91–114.

Silva, T. & Zhao, L. (2016). *Machine learning in complex networks*. Springer International Publishing.

Sindhwani, V., Niyogi, P., & Belkin, M. (2005). Beyond the point cloud: from transductive to semi-supervised learning. In *Proceedings of the 22nd international conference on machine learning* (pp. 824–831). ACM.

Singh, A., Nowak, R., & Zhu, X. (2009). Unlabeled data: now it helps, now it doesn't. In *Advances in neural information processing systems* (pp. 1513–1520).

Smola, A. J. & Kondor, R. (2003). Kernels and regularization on graphs. In *Learning theory and kernel machines* (pp. 144–158). Springer.

Sokolova, M., Japkowicz, N., & Szpakowicz, S. (2006). Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. In *Australasian joint conference on artificial intelligence* (pp. 1015–1021). Springer.

Sokolova, M. & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, *45*(4), 427–437.

Spielman, D. A. & Teng, S.-H. (2014). Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, *35*(3), 835–885.

Stevens, S. S. (1946). On the theory of scales of measurement. *Science*, *103*(2684), 677–680.

Stone, P., Brooks, R., Brynjolfsson, E., Calo, R., Etzioni, O., & Hager, G. (2016). Artificial intelligence and life in 2030. Retrieved from https://ai100.stanford.edu/2016-report

Struyf, A., Hubert, M., & Rousseeuw, P. J. (1997). Integrating robust clustering techniques in s-plus. *Computational Statistics & Data Analysis*, *26*(1), 17–37.

Subramanya, A. [A.] & Talukdar, P. (2014). *Graph-based semi-supervised learning*. Synthesis digital library of engineering and computer science. Morgan & Claypool. Retrieved from http://graph-ssl.wikidot.com/

Subramanya, A. [Amarnag] & Bilmes, J. (2008). Soft-supervised learning for text classification. In *Proceedings of the conference on empirical methods in natural language processing* (pp. 1090–1099). Association for Computational Linguistics.

Subramanya, A. [Amarnag] & Bilmes, J. (2009). Entropic graph regularization in non-parametric semi-supervised classification. In *Advances in neural information processing systems* (pp. 1803–1811).

Subramanya, A. [Amarnag] & Bilmes, J. (2011). Semi-supervised learning with measure propagation. *Journal of Machine Learning Research*, *12*(Nov), 3311–3370.

Talukdar, P. P. (2009). Topics in graph construction for semi-supervised learning.

Talukdar, P. P. & Crammer, K. (2009). New regularized algorithms for transductive learning. In *Machine learning and knowledge discovery in databases* (pp. 442–457). Springer.

Tax, D. M. & Duin, R. P. (2002). Using two-class classifiers for multiclass classification. In *Pattern recognition, 2002. proceedings. 16th international conference on* (Vol. 2, pp. 124–127). IEEE.

Tenenbaum, J. B., De Silva, V., & Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *science*, *290*(5500), 2319–2323.

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

Tukey, J. W. (1949). Comparing individual means in the analysis of variance. *Biometrics*, 99–114.

Van Der Putten, P. & Van Someren, M. (2004). A bias-variance analysis of a real world learning problem: the coil challenge 2000: The coil challenge 2000. *Machine Learning*, *57*(1-2), 177–195.

van der Putten, P. (2000). The coil challenge 2000. Retrieved from http://liacs.leidenuniv.nl/~puttenpwhvander/library/cc2000/

Vapnik, V. N. (1999). An overview of statistical learning theory. *IEEE transactions on neural networks*, *10*(5), 988–999.

Verboven, S. & Hubert, M. (2005). Libra: a matlab library for robust analysis. *Chemometrics and intelligent laboratory systems*, *75*(2), 127–136. Retrieved from http://wis.kuleuven.be/stat/robust/LIBRA/LIBRA-home

Verboven, S. & Hubert, M. (2010). Matlab library libra. *Wiley Interdisciplinary Reviews: Computational Statistics*, *2*(4), 509–515. Retrieved from http://wis.kuleuven.be/stat/robust/LIBRA/LIBRA-home

Wackerly, D., Mendenhall, W., & Scheaffer, R. (2007). *Mathematical statistics with applications*. International student edition / [Brooks-Cole]. Cengage Learning.

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

Wang, J. [Jiang], Song, Y., Leung, T., Rosenberg, C., Wang, J., Philbin, J., ... Wu, Y. (2014). Learning fine-grained image similarity with deep ranking. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1386–1393).

Wang, J. [Jun], Jebara, T., & Chang, S.-F. (2008). Graph transduction via alternating minimization. In *Proceedings of the 25th international conference on machine learning* (pp. 1144–1151). ACM.

Wang, J. [Jun], Kumar, S., & Chang, S.-F. (2012). Semi-supervised hashing for large-scale search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *34*(12), 2393–2406.

Wang, J. [Junhui], Shen, X., & Pan, W. (2007). On transductive support vector machines. *Contemporary Mathematics*, *443*, 7–20.

Wang, Y., Xu, X., Zhao, H., & Hua, Z. (2010). Semi-supervised learning based on nearest neighbor rule and cut edges. *Knowledge-Based Systems*, *23*(6), 547–554.

Weiss, S., Indurkhya, N., Zhang, T., & Damerau, F. (2010). *Text mining: predictive methods for analyzing unstructured information*. Springer New York.

Weiss, Y. (2000). Correctness of local probability propagation in graphical models with loops. *Neural computation*, *12*(1), 1–41.

Weiss, Y., Torralba, A., & Fergus, R. (2009). Spectral hashing. In *Advances in neural information processing systems* (pp. 1753–1760).

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

Xu, C., Tao, D., & Xu, C. (2013). A survey on multi-view learning. *arXiv preprint arXiv:1304.5634*. Retrieved from http://arxiv.org/pdf/1304.5634.pdf

Yang, S., Wang, X., Wang, M., Han, Y., & Jiao, L. (2013). Semi-supervised low-rank representation graph for pattern recognition. *IET Image Processing*, *7*(2), 131–136.

Yang, Y. & Liu, X. (1999). A re-examination of text categorization methods. In *Proceedings of the 22nd annual international acm sigir conference on research and development in information retrieval* (pp. 42–49). ACM.

Zaki, M. & Meira, W. (2014). *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press.

Zelnik-Manor, L. & Perona, P. (2004). Self-tuning spectral clustering. In *Advances in neural information processing systems* (pp. 1601–1608).

Zhao, Z. & Liu, H. (2007). Semi-supervised feature selection via spectral analysis. In *Proceedings of the 2007 siam international conference on data mining* (Chap. 75, pp. 641–646). doi:10.1137/1.9781611972771.75

Zhou, D., Bousquet, O., Lal, T. N., Weston, J., & Schölkopf, B. (2004). Learning with local and global consistency. *Advances in neural information processing systems*, *16*(16), 321–328.

Zhou, D., Huang, J., & Schölkopf, B. (2005). Learning from labeled and unlabeled data on a directed graph. In *Proceedings of the 22nd international conference on machine learning* (pp. 1036–1043). ACM.

Zhu, X. (2005). *Semi-supervised learning literature survey* (tech. rep. No. 1530). Computer Sciences, University of Wisconsin-Madison.

Zhu, X. & Ghahramani, Z. (2002). *Learning from labeled and unlabeled data with label propagation*. Citeseer.

Zhu, X., Ghahramani, Z., Lafferty, J., et al. (2003). Semi-supervised learning using gaussian fields and harmonic functions. In *Icml* (Vol. 3, pp. 912–919).

Zhu, X. & Lafferty, J. (2005). 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* (pp. 1052–1059). ACM.

Zhu, X., Lafferty, J. D., & Ghahramani, Z. (2003). Semi-supervised learning: from gaussian fields to gaussian processes.

Zięba, M., Tomczak, J. M., Lubicz, M., & Świątek, J. (2014). Boosted svm for extracting rules from imbalanced data in application to prediction of the post-operative life expectancy in the lung cancer patients. *Applied soft computing*, *14*, 99–108.

Zimek, A., Schubert, E., & Kriegel, H.-P. (2012). A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining*, *5*(5), 363–387.

Zimmerman, D. W. (1997). Teacher's corner: a note on interpretation of the paired-samples t test. *Journal of Educational and Behavioral Statistics*, *22*(3), 349–360.