Vrije Universiteit Amsterdam

Thesis

A comparison of machine learning algorithms using an insufficient number of labeled observations

Author:
Mark Menagie

Supervisors:
Wouter Pepping
Mark Hoogendoorn

Second reader:
Sandjai Bhulai

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

March 2018

Vrije Universiteit Amsterdam

Thesis

A comparison of machine learning algorithms using an insufficient number of labeled observations

Author:
Mark Menagie

Supervisors:
Wouter Pepping
Mark Hoogendoorn

Second reader:
Sandjai Bhulai

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

March 2018

Vrije Universiteit
Faculty of Sciences
De Boelelaan 1105
1081 HV Amsterdam

Deloitte North-West Europe
Risk Advisory
Technology & Data Risk
Gustav Mahlerlaan 2970
1081 LA Amsterdam

Declaration of Authorship

I, Mark Menagie
, declare that this thesis titled, "A comparison of machine learning algorithms using an insufficient number of labeled observations
" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"*There are two ways to get enough. One is to continue to accumulate more and more. The other is to desire less.*"

G.K. Chesterton

Vrije Universiteit Amsterdam

Abstract

Faculty of Science
Business Analytics

Master of Science

A comparison of machine learning algorithms using an insufficient number of labeled observations

by Mark Menagie

Despite the growing interest of organizations to apply machine learning models within their core processes, they often face the problem of having insufficient labeled data available. Labeling data can be a difficult, time-consuming and expensive process, which makes it hard to apply supervised models in such situations. Semi-supervised learning and active learning can be applied to overcome this problem. Both techniques aim to improve classification performance by incorporating unlabeled observations into the training process.

This study provides insights in the performance of committee-based algorithms within semi-supervised learning and active learning. These algorithms use well-known ensemble techniques bagging and boosting, which makes them easily applicable in practice. Despite their common objective of using unlabeled data to improve classification performance, little effort has been made to compare them in literature so far.

In order to study their performance in different realistic situations, multiple committee-based algorithms were applied on four datasets using various percentages of labeled observations. In the extreme situation where only 5% of the data was labeled, it was found that the algorithms which use bagging provide most accurate predictions. The active learning algorithm Query-by-Bagging significantly outperformed the other algorithms in eight out of twelve experiments. However, it was shown that the active learning algorithms suffer most from the presence of outlying observations.

Acknowledgements

This thesis was written as part of the Master Business Analytics at the Vrije Universiteit of Amsterdam. It describes a problem of the company Deloitte Risk Advisory, which is studied during an internship of six months. I am very grateful for the opportunity Deloitte gave me to write my thesis using their collaboration. I would like to thank all my colleagues from Deloitte Risk Advisory for their guidance, enthusiasm and interest in my research.

Furthermore, I would like to thank my supervisors Dr. Mark Hoogendoorn and Wouter Pepping for their support during the research. The feedback they have provided was very useful and they were always available for discussing new ideas. I would also like to thank Dr. Sandjai Bhulai for his time spent as second reader of this thesis. Finally, I would like to thank Manon and my family for their support during the internship and for helping in finalizing this report.

Mark Menagie

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, many organizations recognize the importance of using predictive analysis within their core processes. They use data analytics to obtain insights about what drives their business or their clients. James McCormick [66] forecasts that such insights-driven organizations will grow from $333 billion in revenue in 2015 to $1.2 trillion in 2020. With the increasing interest in insight-driven decision making, companies invest more in artificial intelligence and machine learning every year. Following the survey in [78], 30% of the attendants think that artificial intelligence will be the most disruptive technology in their industry in the next five years. Deloitte recently opened the Artificial Intelligence Center of Expertise (AICE) in order to meet the rapidly increasing demand for artificial intelligence [95]. Despite this growing interest, they often face complex client assignments due to the lack of available data in a manageable structure.

Many machine learning algorithms are available to construct predictive models. Machine learning is a field in data analytics that focuses on the development of mathematical algorithms to predict future events. These algorithms seek to find hidden patterns in large sets of historical data, without being explicitly programmed [83]. A considerable amount of literature within machine learning has been published on classification problems. Classification algorithms aim to assign the correct category to new observations by learning from previous observations. Learning from data where the output value (label) is known for all observations can be considered as *supervised learning*. This machine learning task aims to infer a function from labeled training data, which can be used for mapping new observations [70]. In order to obtain satisfying performance on unseen data, it is important that supervised algorithms are able to generalize well using training data. Ensemble learning (or committee-based learning) is a well known technique to combine multiple classification models into one ensemble classifier. It has proven to outperform the generalization ability of single classifiers [34].

Although supervised classification is a widely used machine learning technique, it requires a fully labeled dataset. In many situations labeling data is a difficult, time-consuming and expensive process which requires knowledge of domain experts [85, 19]. Suppose that the objective is to classify webpages to a category based on its content. Webpage classification [51] is a domain where it is relatively easy to collect unlabeled data by scraping webpages. However, assigning a category to each webpage is highly time-consuming since pages need to be read manually. *Semi-supervised learning* algorithms have been developed to overcome this problem. Semi-supervised algorithms automatically exploit unlabeled observations to find hidden structures that might improve the supervised learning process using limited data. Another technique to improve classification models using insufficient labels is *active learning*. Instead

of automatically selecting unlabeled observations to include in the model, a (usually human) supervisor is involved in the training process within active learning. It assumes that the supervisor is able to provide the correct label for a queried observation [85]. By requesting information about observations which are most uncertain to the classifier, it can be re-trained including newly labeled observations. The aim of active learning algorithms is to request as least as possible manual annotations to achieve a satisfying performance. Over the years, many different algorithms were introduced within semi-supervised learning and active learning. Several promising studies were conducted on so-called committee-based models within both fields [105, 104]. These algorithms aim to obtain strong generalization by incorporating supervised ensemble techniques into semi-supervised learning and active learning.

Since both fields share the same objective of incorporating unlabeled data in classification problems, they are suitable for comparison studies. Stikic et al. [89] compared several state-of-art algorithms from both fields. Perzello et al. [76] focused on SVM-based techniques while performing an extensive comparison between semi-supervised learning and active learning. However, little effort has been made to compare committee-based models so far. The aim of this thesis is to study the performance of such committee-based algorithms within semi-supervised learning and active learning using multiple data sets consisting of limited labeled observations. This research can make an important contribution to solve problems with insufficient labeled data and tries to find an answer to the following research-question:

Which committee-based classification techniques within semi-supervised learning and active learning perform best using multiple data sets with various percentages of labeled observations?

This research-question is supported by the following sub-questions:

- Which algorithms are available within semi-supervised learning and active learning to handle the problem of data with insufficient number of labeled observations?
- How much does the size of the initial labeled training set influence the performance of active learning algorithms?
- Which committee-based algorithms are most robust to outliers in the data?
- How do semi-supervised learning and active learning algorithms perform in comparison to supervised algorithms using various percentages of labeled observations?

This thesis will first discuss background information about ensemble learning, semi-supervised learning and active learning. Next, the applied experiments will be explained in detail. Finally, the results of the research will be presented and further discussed.

Chapter 2

Background

This thesis will focus on committee-based algorithms within semi-supervised learning and active learning. Most of these algorithms are extended supervised ensemble algorithms to be able to handle data with insufficient class labels. Therefore, some necessary background of supervised ensemble techniques will be provided in this chapter first. Next, the concept of semi-supervised learning will be explained and several existing algorithms will be discussed. Finally, we will discuss previous work on active learning and its relevance for this study.

2.1 Ensemble learning

Ensemble learning is a well known technique to combine multiple classification models into one ensemble classifier. An ensemble classifier is constructed by generating multiple base classifiers on training data and then combining the separate predictions using a voting system. The objective of ensemble learning is to develop a model that is able to provide more accurate predictions than each of its single component classifiers. Ensemble classifiers have proven to outperform the generalization ability of single classifiers [34].

Dietterich [34] introduced three main reasons why ensembles can improve the accuracy of its component classifiers:

- *Statistical problem*
  The aim of classification algorithms is to learn from historical training data to find the best classifier f. However, it can be hard to find f using a single classifier when the available amount of training data is insufficient. When this situation occurs, the problem of having a small training set can be addressed by constructing an ensemble of multiple classifiers. By voting for the final hypothesis, the ensemble can find a satisfying approximation of the best solution f.

- *Computational problem*
  Several algorithms perform local search like gradient descent to search for the optimal solution. Even if there is enough training data available (so the statistical problem does not occur), it is possible that these algorithms will converge to a local optimum. Since the techniques to find the global optimum can be computationally intensive, ensemble learning performs multiple local search algorithms that use different starting points in search space. By using different starting points, the ensemble might approximate the optimal solution better than a single classifier.

- *Representational problem*
  It can occur that the optimal classifier f does not exist in the complete search space defined by single classifiers. In such situations, a combination of these classifiers might approximate f more accurately than the component classifiers separately. Figure 2.1 shows an example of a classification problem with two classes that are not separable by a single linear classifier. None of the linear classifiers A, B and C succeed in separating the (+)-class from the (-)-class perfectly. However, for every observation at least two of the classifiers agree about their hypothesis. The ensemble classifier (bold line in Figure 2.1 from [1]) that results from combining the single classifiers is capable of separating both classes accurately.

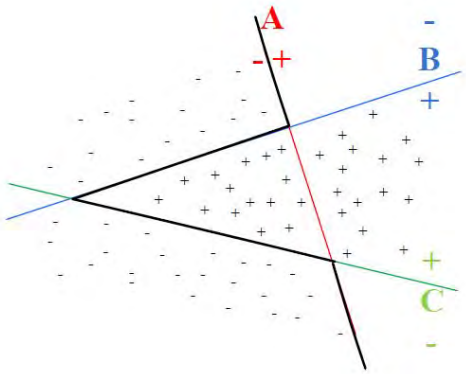


FIGURE 2.1: An illustration of the representational problem using three linear classifiers

Although these problems show that ensemble learning can outperform single classifiers in many situations, ensemble classifiers require the presence of enough diversity in the component classifiers to ensure good performance [55]. The aim is to combine classifiers that make as few errors as possible, but they should make different errors to be able to learn from each other. Diverse component classifiers examine a larger space than one- or multiple identical classifiers. When nearly identical models vote for a prediction, they would all agree and behave like a single classifier.

There are several methods to make classifiers more diverse. *Bagging* [16] and *boosting* [42] are two well-known techniques that adjust the original training data to include diversity in the training process. The former method uses different training sets for every component classifier in parallel, while the latter method trains multiple classifiers in sequence on training sets using different weighted observations. Both techniques can be used to resolve the *bias-variance trade-off*. This is the problem of simultaneously minimizing two types of error that limit supervised models to generalize beyond training data [96]. We can decompose the expected error of an unseen observation x into the following terms:

- *Bias*: an error caused by limitations in the learning method
- *Variance*: an error caused by limitations in the training data
- *Irreducible error*: an error resulting from noise in the problem itself

Both bagging and boosting aim to minimize the expected error, but they focus on different problems. Bagging seeks to reduce variance by resampling the training data,

while boosting aims to minimize the bias by constructing an ensemble that has lower bias than the individual models [40].

2.1.1 Bagging

Bootstrap aggregating or bagging [16] is an ensemble method that trains multiple classifiers on different bootstrap samples of the original training data. For every classifier a random sample with replacement is drawn from training data, containing the same number of observations. Since samples are drawn with replacement, the bootstrap samples can contain duplicate observations. After training each component classifier on a different bootstrapped sample, one of the classes can be assigned to any observation x in the classification phase. In this phase the bagging algorithm performs *majority voting*. All component classifiers $\{h_1, h_2, \ldots, h_N\}$ vote for which of the k classes in C should be assigned to unlabeled observation x. The class that receives most votes will be assigned to x as final prediction.

Breiman [16] has shown that bagging can improve prediction accuracy of so-called unstable algorithms or weak classifiers. Small changes to training data of such weak classifiers can result in large changes in predictions. Breiman [16] explained that Neural Networks and Decision Trees are such unstable algorithms for which bagging could be a solution to improve performance. On the other hand, bagging will have less effect on more stable algorithms like Naïve Bayes and K-Nearest Neighbor [16, 31] since they will not allow enough diversity. The original bagging procedure is shown in Algorithm 1.

Probably the most well-known algorithm that applies bagging is *Random Forest* [17]. The only difference between *Random Forest* and the bagging algorithm using decision trees is the procedure for feature selection. If some features have great influence on the prediction of the class variable, it is likely that they will be selected in most trees in the ensemble. In order to generate a more diverse ensemble of classifiers *Random Forest* selects a random subset of features at each candidate split, which is also called *feature bagging* [17].

2.1.2 Boosting

In contrast to the independent training of classifiers with bagging, boosting aims to build a sequence of classifiers that depend on each other. It trains multiple weak classifiers in sequence on different training sets $L = \{L_1, L_2, \ldots, L_N\}$. Each set contains weighted observations to generate diversity in the learning process. Observations that were misclassified by classifier h_{i-1} will be assigned higher weights than correctly classified observations to force the next classifier h_i to focus more on observations that are hard to classify.

Freund and Schapire [42] introduced one of the first boosting algorithms *AdaBoost*, short for Adaptive Boosting. The objective in each iteration of this algorithm is to reduce the training error of the classifier in the previous iteration. Each iteration $i \in \{1 \ldots N\}$ observation weights D_{i+1} are updated for classifier h_{i+1} in the next iteration. The weights D_{i+1} are multiplied with factor $\frac{\epsilon_i}{1-\epsilon_i}$ for correctly classified observations, while misclassified observations will keep the same weights as in the previous iteration. In this way, the model will focus more on misclassified observations from the previous iterations. Since the update of the weights depends on training

Algorithm 1 *Bagging*

Input: Original training set L, Ensemble size N and the learning algorithm
Output: Ensemble classifier H

Training phase:

1: **for** $i = 1$ to N **do**
2: $S_i = BootstrapSample(L)$
3: $h_i = Learn(S_i)$
4: Add classifier to ensemble H: $H = H \cup h_i$
5: **end for**
6: **return** ensemble $H = \{h_1, h_2, \ldots, h_N\}$

Classification phase (majority voting):

7: Classify unlabeled observation x using all component classifiers $\{h_1, h_2, \ldots, h_N\}$
8: Let $v_{i,k} = \begin{cases} 1 & \text{if } h_i \text{ votes for class } C_k \\ 0 & \text{otherwise} \end{cases}$
9: Obtain votes for all classes $V_k = \sum_{i=1}^{N} v_{i,k}$ for $k = 1, \ldots, C$
10: Choose class with most votes: $H(x) = \arg\max_{1 \leq k \leq C} V_k$ for $k = 1, \ldots, C$
11: **return** prediction $H(x)$

error ϵ_i, the weights are changed more heavily if the training error is high. In addition to changing the weights of all observations, a weight w_i ($0 \leq w_i \leq 1$) is assigned to classifier h_i. Classifier weight w_i is calculated by $-log(\frac{\epsilon_i}{1-\epsilon_i})$ to ensure that classifiers with lower training errors have more influence in the final majority voting procedure. Note that training error ϵ_i must satisfy $0 < \epsilon_i < 0.5$ to make sure that the weights are updated correctly. Therefore, the algorithm terminates if the model either obtains perfect classification or when training error is at least equal to 0.5. This stop-condition can be a drawback of the algorithm when available training data is sparse, since the ensemble might converge to a perfect training error too quickly. Even when a component classifier would be able to classify a small set of training data correctly, it does not necessarily mean that the ensemble model cannot improve on test data anymore.

Dietterich [33] compared *AdaBoost* with the C4.5 Decision Tree algorithm [80] as base-learner to other ensemble methods and concluded that *AdaBoost* is very sensitive to classification noise. If training data contains observations with incorrect labels, *AdaBoost* seems to focus (i.e. assigns high weights) too much on these noisy observations which causes the classifiers to perform poorly. Despite these comments, several experiments [8, 33, 79] have shown that *AdaBoost* can be very successful and especially when using decision trees as base learner. Breiman [14] stated that *AdaBoost* is even one of the best off-the-shelf classification models available. The *AdaBoost* algorithm is shown in Algorithm 2.

In 1998, Breiman [15] introduced a statistical framework to understand the success of boosting algorithms like AdaBoost better. This framework was used by Friedman [44], who introduced *Gradient Boosting Machines* (also called *Gradient Boosting*). Gradient Boosting tries to generalize prediction models by optimizing any arbitrary differentiable loss function L. The chosen loss function measures the costs for incorrect

Algorithm 2 *AdaBoost*

Input: Original training set $\{L_1, \ldots, L_m\}$, ensemble size N and the learning algorithm

Output: Ensemble classifier H

Training phase:

1: Initialize weights of all observations: $D_1(j) = 1/m$, $\forall j \in \{1 \ldots m\}$
2: **for** $i = 1$ to N **do**
3: $h_i = Learn(L, D_i)$
4: Obtain weighted training error of h_i:
$\epsilon_i = \sum_{j=1}^{m} D_i(j) \times I(h_i(x_j) \neq y_j)$
5: **if** $\epsilon_i = 0$ or $\epsilon_i \geq 1/2$ **then**
6: Set final number of iterations $N = i - 1$
7: **break**
8: **end if**
9: Adjust weight of h_i based on training error: $w_i = -log(\frac{\epsilon_i}{1-\epsilon_i})$
10: Update weights of training observations:
$D_{i+1}(j) = \frac{D_i(j)}{Z_i} \times U_i$ where $U_i = \begin{cases} \frac{\epsilon_i}{1-\epsilon_i} & \text{if } h_i(x_j) = y_j \\ 1 & \text{otherwise} \end{cases} \quad \forall j \in \{1 \ldots m\}$
and $Z_i = \sum_{j=1}^{m} D_{i+1}(j)$ as normalization constant
11: **end for**
12: **return** ensemble $H = \{h_1, h_2, \ldots, h_N\}$

Classification phase (weighted majority voting):

13: Classify unlabeled observation x to the class with highest weighted vote:
return prediction $H(x) = \arg\max_{1 \leq k \leq C} P(v_k|x)$
where $P(v_k|x) = \frac{1}{\sum_{i=1}^{N} w_i} \sum_{h_i(x)=v_k} w_i$ for $k = 1, \ldots, C$

predictions. Initially, one single weak classifier $F_m(x)$ (where iteration $m = 0$) is trained and makes predictions for a set of test observations. In every iteration gradient boosting tries to improve the current model by adding another model $h(x)$. This results in a new model $F_{m+1}(x) = F_m(x) + h(x)$. The objective is that this model $F_{m+1}(x)$ is able to predict the class variable y perfectly:

$$F_{m+1}(x) = F_m(x) + h(x) = y$$

We can rewrite this as:

$$h(x) = y - F_m(x)$$

This means that if we want to find the optimal $h(x)$ to add to the current model, it should equal $y - F_m(x)$. This is the difference between predictions and actual values, which are also called the residuals. To find $h(x)$, a model is fitted to the residuals to try to capture their structure. The minimization problem to find $h(x)$ is solved by applying *steepest descent* [77]. This calculates the negative gradient of the loss

function L using the current model F_{m-1}:

$$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla_F L(y_i, F_{m-1}(x_i))$$

Where the step length γ_m is chosen using line search:

$$\arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) - \gamma \frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)})$$

Recently, Chen [23] introduced *Extreme Gradient Boosting* (XGBoost). It follows the principles of gradient boosting, but is implements several improvements in speed and performance. The improvement in performance is mainly caused by the introducing more regularization into the gradient boosting model, which is better able to avoid overfitting on training data. XGBoost became very popular after winning the Higgs Machine Learning Challenge. In 2015, among 29 winning solutions on machine learning competition site Kaggle, XGBoost was used in 17 of them [23].

2.2 Semi-supervised learning

As explained in Section 1, semi-supervised algorithms can be applied to combine both labeled- and unlabeled data when insufficient labeled observations are available. Most of these algorithms automatically search for unlabeled observations that can improve the performance of supervised base-learners, which are initially trained on a limited number of labeled observations. In this section some basic definitions within the semi-supervised setting will be introduced first. Next, the main assumptions on which semi-supervised algorithms depend will be explained and the history of semi-supervised learning will be discussed. In addition, multiple state-of-the-art semi-supervised algorithms will be introduced. Finally, an extensive description of committee-based semi-supervised learning will be given.

2.2.1 Introduction

In the standard semi-supervised learning setting a dataset is provided, which contains observations with a label and observations for which the label is not known. This setting can be formulated as a set of l labeled observations

$$X_l = (x_1, x_2, \cdots, x_l) \quad \text{and its labels} \quad Y_l = (y_1, y_2, \cdots, y_l)$$

and a set of u unlabeled observations without any known labels

$$X_u = (x_{l+1}, x_{l+2}, \cdots, x_{l+u}).$$

In a realistic semi-supervised learning application the number of unlabeled observations clearly outnumber the number of labeled observations $u \gg l$.

Much of the literature on semi-supervised learning distinguishes two main learning types: *inductive learning* and *transductive learning* [108, 19]. Inductive learning is referred to as the general semi-supervised setting, where an algorithm learns a function from a combination of labeled- and unlabeled data to classify unseen data. These new observations can be either a separately stored test set or new data that is coming into the system in an online manner. Transductive learning, introduced by Vapnik [94],

uses unlabeled observations during training as well, but cannot handle unseen data. Transductive algorithms are only able to predict the unlabeled observations that are also included in the training process. In a more formal way, inductive learning uses labeled set $\{X_l, Y_l\}$ and unlabeled set X_u for training and can predict a set of t unseen observations

$$X_t = (x_{n+1}, x_{n+2}, \cdots, x_{n+t}) \quad \text{where } n = l + u$$

Transductive learning also uses labeled set $\{X_l, Y_l\}$ and unlabeled set X_u for training, but can only predict unlabeled observations X_u. Most graph-based semi-supervised algorithms are transductive and will be explained in subsubsection 2.2.3.2 to make transductive learning in practice more clear. A widely used example to illustrate the difference between those two types of semi-supervised learning is as follows: suppose students need to prepare for an exam and the teacher has provided multiple example questions including the correct solutions (i.e. observations with known labels) and some questions without solutions (i.e. unlabeled observations). In a transductive setting, students have to make a take-home exam and answer the provided questions for which no solutions were given. When they study, they will focus on these particular questions to solve them as good as possible. In an inductive setting, students can study the questions including solutions and test their gained knowledge by trying the questions without solutions. They use these test questions to practice for an in-class exam containing similar questions which they have not seen before.

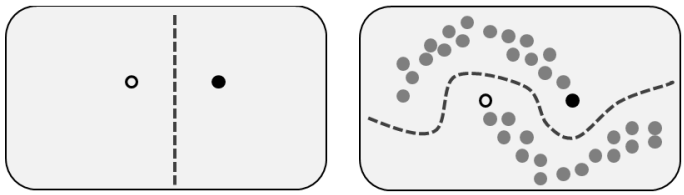
The advantage of transductive learning is that it solves an easier problem than inductive learning, as it only predicts the classes for the test points of interest [19]. However, when the purpose of the application is to make many predictions on large datasets, transductive learning is computationally expensive since it needs to learn on new data every time (new unlabeled observations to predict). Inductive learning algorithms need to be trained once and can be used for new data. Therefore, this thesis will focus on inductive learning to use labeled- and unlabeled data to find a function that can classify unseen data accurately.

2.2.2 Underlying assumptions

Most semi-supervised algorithms depend on several underlying assumptions. In order to use unlabeled observations in a model, some assumptions about the underlying data distribution should be made. Most semi-supervised algorithms rely on at least one of the underlying data assumptions described by Chapelle [19]. In reality it can be hard to determine which model assumptions fit best to the data. It is always important to investigate the problem structure and choose the algorithm that fits the structure best. When it is hard to verify the model assumptions, it might be best to choose an algorithm that is most robust to the underlying assumptions.

2.2.2.1 Smoothness assumption

The smoothness assumption indicates that if observations in a high-density region are close to each other, they are more likely to belong to the same class. This assumption seeks to generate a decision boundary in low-density regions, where not many observations are present. Figure 2.2 from [99] shows an example where this assumption moves the decision boundary to a low-density region when incorporating unlabeled data in the learning process. The observations in the top cluster are separated by a low-density region from the observations in the bottom cluster. If the observations within each cluster are close to each other according to a chosen distance metric, the



(a) (b)

FIGURE 2.2: (a) shows the decision boundary when a classifier learned from labeled data only. In (b) unlabeled data was incorporated in the learning process.

smoothness assumption would indicate that the drawn decision boundary separates observations from two different classes.

2.2.2.2 Cluster assumption

The cluster assumption is a special case of the smoothness assumption as it states that observations are likely to belong to the same class if they are in the same cluster. When a cluster algorithm has been able to separate data into different clusters, labeled observations can be used to assign a class to each cluster. Since clusters are often sets of observations which are located in high-density regions, this assumption can be seen a special case of the smoothness assumption. Decision boundaries in high-density regions would separate clusters in multiple segments, so this assumption also seeks to move the decision boundary to low-density regions.

2.2.2.3 Manifold assumption

Curse of dimensionality is a common problem in machine learning problems. It means that when high-dimensional data is available, many observations are needed to ensure that the data contains all possible combinations of possible values. Many semi-supervised algorithms assume that unlabeled observations are similar as 'close' labeled observations. The measurement of this distance highly depends on the distance metric used. With high-dimensional data, it is very hard to choose a distance metric that gives an accurate indication of the closeness of observations. For such data sets, the manifold assumption can overcome this problem. Following [19], the manifold assumption indicates that (high dimensional) data lie (roughly) on a low-dimensional manifold. If we are able to reduce the number of dimensions drastically by finding an underlying manifold of the data, the distance between observations can be measured more accurately.

2.2.3 Algorithms

Over the years, many different semi-supervised algorithms were developed. All these algorithms incorporate unlabeled observations to improve classification performance when insufficient labeled observations are available. Despite the common objective, the approaches to use unlabeled data differ strongly. Semi-supervised algorithms can be divided into four different categories:

- Iterative models
- Margin-based models

- Graph-based models
- Committee-based models

Please note that all categories will be explained in an inductive setting except graph-based models as they are usually transductive.

2.2.3.1 Iterative models

2.2.3.1.1 Self-Training

Iterative models aim to build accurate classifiers by improving initial weak classifiers in an iterative process. *Self-Training* [84], also known as self-labeling, is one of the first iterative algorithms that was developed for semi-supervised learning. Self-Training initially trains a classification algorithm on a set of labeled observations. The resulting classifier makes predictions for the unlabeled observations. The observations that are predicted with most certainty are added to the training set and the classifier is trained on this new set again. This iterative process repeats until a stopping condition is met, like a maximum number of iterations or a certain level of convergence. It is unavoidable that the classifier makes misclassifications on the set of unlabeled observations. If these misclassifications would be added to the training set, the added noise will affect the classifier negatively. Therefore, the main challenge of Self-Training is to find a metric to select high confidence predictions accurately.

The advantage of this method is that any classification algorithm can be used. Since the model needs to be trained multiple times, it might be best to use models that are not too computationally expensive though. Two main drawbacks of this algorithm are the selection of non-informative observations and the algorithms sensitivity to outliers [1]. The difference between a non-informative observation and an outlier is that the former belongs to one of the classes, while the latter does not belong to any class at all. As both type of observations tend to lie far from the decision boundary, it is likely that Self-Training adds them to the training data.

The idea of self-training was introduced a long time ago in studies from Scudder [84] in 1965, Fralick [41] in 1967 and Agrawala [4] in 1970. Later, several Self-Training applications were performed for different classification tasks. In 1995, Yarowsky [102] applied the algorithm for a natural language processing task. He tried to construct a model for word sense disambiguation, where it seeks to classify the correct meaning of words which are ambiguous. Riloff et al. [81] tried to improve information extraction (IE) systems by building a Self-Training algorithm to identify subjective language, which often causes IE systems to result in false hits. Rosenberg [82] developed a Self-Training object detection model, since it is a very time-consuming and difficult task to collect labeled objects in this field.

2.2.3.1.2 Generative models

Just like Self-Training models, *generative models* belong to the category of iterative models. Generative models assume that the unlabeled data follows the same underlying distribution as the labeled data. The model estimates each class by an identifiable mixture model (i.e. Gaussian mixture) using labeled data. Next, unlabeled observations are used to estimate the parameters of the distributions. Since generative models assume that both classes can be estimated by a different mixture model, they follow the semi-supervised cluster assumption 2.2.2.1. Generative models are mostly

applied in an iterative manner, where the iterative *Expectation-Maximization* algorithm (EM) [32] is used to tune the model parameters. Initially, the mixture model is trained on all labeled observations. The resulting model is used to classify all unlabeled observations in the expectation step. In the maximization step the model is re-trained on both originally labeled observations and newly labeled observations (i.e. it maximizes the likelihood of the parameters). This procedure is repeated until convergence. Although generative models can obtain accurate predictions using a low number of labeled observations, Cozman et al. [29] have shown that classification performance can degrade if the distribution of the unlabeled data does not follow the model assumption. This makes it hard to apply generative models in realistic situations. In real-life scenarios it is usually hard to find the underlying distribution of a data set. In literature, the used mixture model differs for most applications. A mixture of multinomial Naive Bayes distributions is usually used in text classification applications [73, 26]. For image classification Gaussian mixture models are often used to combine labeled- and unlabeled images [86, 101].

2.2.3.2 Margin-based models

Margin-based models follow the semi-supervised smoothness assumption 2.2.2, as they aim to move the decision boundary to low-density regions by maximizing the margin of both labeled- and unlabeled observations. Most of these models are based on the supervised classification algorithm *Support Vector Machine* (SVM) [27]. The first extension of SVMs to semi-supervised problems is the *Transductive Support Vector Machine* (TSVM) [52]. This algorithm implements a transductive setting [94], where test data is included in the learning process to find a maximum margin. This idea can also be used in an inductive setting, where the SVM combines labeled- and unlabeled data to define a function to classify unseen observations. This inductive version of the algorithm is called S^3VM [10] and uses a mixed integer programming approach to solve the corresponding optimization problem. A graphical illustration of the S^3VM algorithm is shown in Figure 2.3. Semi-supervised margin based models strongly depend on the assumption that unlabeled observations from different classes are separated by a large margin. If data does not meet this assumption it is not possible to shift the decision boundary to a low-density region. The computational expensive behavior is another difficulty of SVM-based approaches. Therefore, several algorithms were studied which approximate the exact SVM solution [21, 20, 108]. Finally, Persello and Bruzzone proposed a semi-supervised algorithm *Progressive Semi-Supervised SVM* (PS^3VM) by integrating concepts from active learning [76]. Besides a discussion about the results of PS^3VM they have compared several margin-based models from both semi-supervised learning and active learning, which is obviously relevant for this research.

2.2.3.3 Graph-based models

Semi-supervised graph-based models construct a graph containing all labeled- and unlabeled observations as nodes. The nodes are connected by edges, which represent similarity between the observations. Figure 2.4(a) from [92] shows a similarity graph used by such algorithms. Graph-based algorithms try to distribute information through the graph in order to label all observations [19]. Most graph-based semi-supervised algorithms are transductive, since adding unseen observations to an existing graph might change the graph structure and can change the predicted labels of observations from set U again.

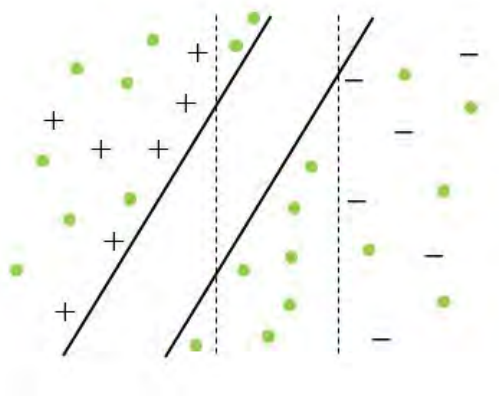
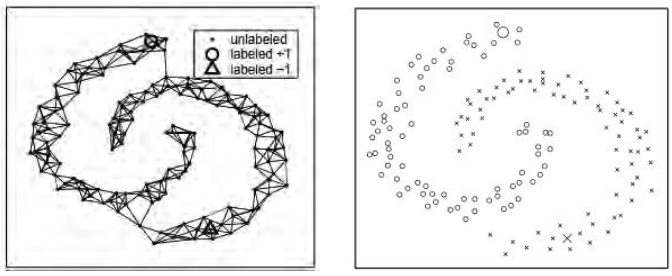


FIGURE 2.3: An illustration of the S^3VM algorithm. The dotted lines illustrate the margin using labeled data only, which separates the positive (+) observations from the negative (-) observations. The solid lines illustrate the margin when the green unlabeled observations are taken into account.

All existing graph-based algorithms use a given similarity graph as input and formulate an optimization problem to search for an optimal propagation of labels through the graph. This type of algorithms highly depend on the similarity graph, but graph construction is considered as a hard task and is not included in the algorithms [108]. Most existing graph-based algorithms estimate a function which minimizes the loss on labeled examples and follow the smoothness assumption as explained in subsection 2.2.2. The different algorithms mostly differ from each other by the loss-function that is minimized [108]. Label propagation [110] is one of the most well-known graph-based algorithms within semi-supervised learning. It uses few labeled observations as sources to propagate labels through the graph and predict unlabeled observations. Another widely used graph-based algorithm is Markov Random Walks [92]. It starts random walks from each observation through the connected graph and stops when it ends up in a labeled observation. Eventually it calculates for each unlabeled node the probability that it belongs to a certain class, based on many of such random walks. Figure 2.4 shows a) the similarity graph used in the experiment by [92] and 2) the resulting classification by the algorithm.



(a) (b)

FIGURE 2.4: (a) illustrates a similarity graph used for the experiments in [92]. Two labeled observations are marked as a cross and a triangle and the remaining unlabeled observations are marked as small dots. (b) shows the resulting perfect classification using Markov Random Walks.

2.2.3.4 Semi-supervised learning with committees

As discussed in Chapter 1, this thesis will focus on committee-based algorithms within semi-supervised learning and active learning. The main objective of machine learning algorithms is to be able to achieve strong generalization (Section 2.1). When an algorithm learns from training data it is essential that the algorithm will not build a complex model which can obtain satisfying predictive performance on the training data only (i.e. overfitting the training data). Since both ensemble learning and semi-supervised learning have proven to obtain strong generalization, the question raised whether these successful approaches can be combined. In literature, the combination of ensemble learning and semi-supervised learning is mostly referred to as *semi-supervised learning with committees* [1] and *semi-supervised learning by disagreement* [105, 104]. For the remainder of this report, we will use the expression semi-supervised learning with committees.

In a study conducted by [105] and [104], it was shown why the use of unlabeled data with multiple classifiers can improve classifier performance. First they reported that classifier combination can improve results when single classifiers cannot improve using unlabeled observations. They conducted several theoretical analyses that compare semi-supervised algorithms using single classifiers and semi-supervided algorithms using ensembles. Furthermore, they stated that unlabeled data can improve ensemble classifiers by introducing more diversity in the learning process. They measured ensemble diversity using multiple- data sets and diversity metrics. Their results show that incorporating unlabeled data into an ensemble model increases diversity in most of the cases, which is important for the performance of the ensemble models. Despite these findings, only a few attempts were made to apply this field of semi-supervised learning [106, 59, 63, 30, 22, 11].

2.2.3.4.1 Co-Training with natural views

The idea of using multiple classifiers in semi-supervised learning was first studied by Blum and Mitchell with the *Co-Training* [12] algorithm. This algorithm assumes that data can be split into two independent and redundant feature sets (also called views). Each view x^1 and x^2 is assumed to be independent given the class label and should contain enough information to construct a strong classifier. The latter assumption should be satisfied since both classifiers learn from the different views and try to exchange useful information to improve each other. Both classifiers aim to select the most-confident predictions from unlabeled set U and add them to each others labeled set. Figure 2.5 provided in [108], illustrates the usefulness of Co-Training when the model assumptions are satisfied. The circles indicate high-confident observations based on the x^1 view, since they are located far from the decision boundary. However, if these observations are plotted in the x^2 view they are distributed more randomly. In this way, the classifier learning from view x^1 can add useful information to the other classifier by providing these high-confident observations including predicted labels. The algorithm can be found in Algorithm 3.

One field where Co-Training is often used is web-page classification, since web-page data can be split into two independent views. One view contains features about the text that is shown on the web-page, while the other view consists of anchor text in hyperlinks on other web-pages that link to the specific page. It is easy to collect text from web-pages by web scraping (unlabeled data), but it is a time-consuming task to label web-pages by assigning the correct subject to the page based on its content. The

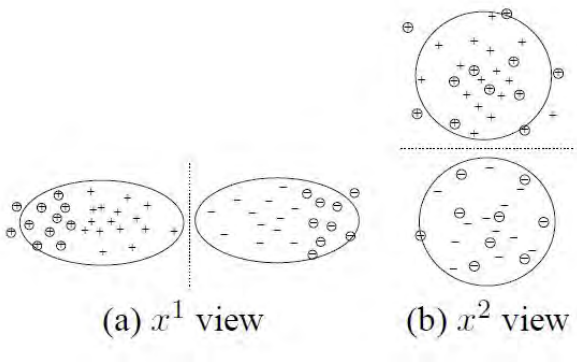


FIGURE 2.5: The circles indicate high-confident observations based on features from view x^1. The same observations are plotted using features from view x^2 and show a different distribution.

availability of two different feature views and a limited number of labeled web-pages, makes it a perfect application for Co-Training. Despite this example with two existing feature sets, most data sets contain only one feature set in most realistic situations. Nigam and Ghani [72] have studied the behavior of Co-Training on different data sets. Some of these data sets contain a natural feature split, while others are general data sets with just one natural feature set. Their results have shown that the results highly depend on its assumptions about the available feature views. This causes the Co-Training algorithm to have limited applicability in real-life problems.

Algorithm 3 *Co-Training*

Input: Labeled data L, unlabeled data U, feature sets (views) x^1, x^2 ,base learning algorithm, maximum number of iterations T, selection size k

Training phase:

Train initial classification models ${h_1}^0$ and ${h_2}^0$ on views x^1 and x^2 respectively
$L_1 = L_2 = L$
for $i = 1$ to T **do**
  if U is empty **then**
    $T = t - 1$ and *stop*
  end if
  for $v = 1$ to 2 **do**
    Predict observations in U using ${h_v}^{t-1}$
    Select k most-confident observations S_v from ${h_v}^{t-1}$ and add to L_{3-v}
    Remove S_v from U
  end for
  Re-train ${h_1}^t$ on L_2 and ${h_2}^t$ on L_1
end for

Classification phase:

return prediction $\dfrac{h_1(x)^T + h_2(x)^T}{2}$

2.2.3.4.2 Co-Training with random views

Despite this shortcoming of the algorithm, Blum and Mitchell have laid an important foundation for the use of multiple classifiers within semi-supervised learning. In addition to their research on the Co-Training assumptions, Nigam and Ghani [72] studied the use of random feature splits. Instead of using data with natural feature splits, they used standard data sets and generated two random feature splits to meet the Co-Training requirement. Using these standard data sets, Co-Training with random feature splits competed with the generative model EM. However, the results of Co-Training using the natural feature splits did clearly outperform the random feature splits. Since the splits are drawn randomly, it is likely that two random features are correlated too much. This might make it hard for both classifiers to select useful observations to improve each other.

2.2.3.4.3 Co-Training with artificial views

In order to obtain feature splits which meet the Co-Training setting better, Feger and Koprinska [39] introduced the method *maxInd* to construct two artificial views. MaxInd is a graph-based method which aims to minimize the dependence between two feature sets. First, it constructs an undirected graph with features as nodes and the conditional mutual information between features as the graph edges. The conditional mutual information between two features, explained in [61], can be calculated using multiple features and the class variable. Next, the graph is split into two equally sized sub-graphs while minimizing the sum of the cut edges (i.e. minimizing the dependence between both sub-graphs). Despite this minimization, experiments in [39] have shown that maxInd does not outperform random feature splits on many datasets. One reason might be that the available class variable is sparse. This causes the calculation of the conditional mutual information to be inaccurate. Another possible reason for the surprising results can be that the performance of Co-Training not only depends on dependence **between** two views, but also relies on dependence **within** views [39]. If the dependency between views is low but the dependency of features within views is high, the classifiers might suffer from training on a highly correlated feature set. This might cause classifiers to be inaccurate, which makes them unable to train each other accurately.

2.2.3.4.4 Co-Training with single views

After several different approaches to split single view data into independent feature sets to make data suitable for Co-Training, Goldman and Zhou [47] studied the idea of using single view data in Co-Training style algorithms.

Statistical Co-Learning
First, they introduced the algorithm *Statistical Co-Learning* [47], which uses a single feature set only. This Co-Training based algorithm trains two different classification models on the initially labeled observations. Both classifiers select the most-confident unlabeled observations and add them to the training set of the other classifier including predicted label. In each iteration 10-fold cross validation is performed to construct confidence intervals for the predictions of unlabeled observations. Using these confidence intervals, statistical tests are performed to 1) dnde which observations should be labeled and 2) which final prediction should be made by combining both classifiers. One major drawback is that statistical tests are used to decide which observations

to label, while usually the available labeled observations are insufficient in a semi-supervised setting to perform such tests.

Democratic Co-Learning
To overcome these drawbacks, Goldman and Zhou [103] introduced *Democratic Co-Learning*. This method has many similarities with their previous study, but instead of two classifiers it uses an ensemble of at least three different algorithms. A combination of majority voting and confidence measurements are used to select observations that should be labeled and added to training data. This approach does not require any statistical tests to select observations anymore. Confidence intervals are still being constructed using 10-fold cross validation to find confident predictions. This causes the algorithm to be still computationally expensive.

Tri-Training
In order to avoid the use of the expensive 10-fold cross validation in every iteration, Zhou and Li [106] studied an approach that still combined concepts from Co-Training but focused more on ensemble learning. They introduced the algorithm *Tri-Training* that constructs an ensemble of three classifiers. All component classifiers include the same algorithm (i.e. base learner), but the ensemble is made diverse by applying bagging (described in subsection 2.1.1) in the first iteration. Let $H = \{h_1, h_2, h_3\}$ be the ensemble initially trained on three bootstrapped samples of originally labeled observations L. In each iteration a subset of unlabeled observations U are selected by component classifiers h_j and h_k $(j, k \neq i)$ under certain conditions and added to the training set of h_i if both h_j and h_k agree about their predictions. In the next iteration, the classifiers are re-trained on the training sets including selected observations with predicted labels.

Although Tri-Training neither depends on data with two independent feature sets nor restricts the base learner to be a tree-based classifier, it still contains some important drawbacks [59]. First, the algorithm restricts the ensemble size to three classifiers. If the algorithm would allow to add more classifiers to the ensemble, it might improve generalization. Furthermore, bagging on training sets is only performed in the first iteration. This allows all component classifiers to become more similar every iteration, which stops them from searching different areas of the solution space.

In [106] Tri-Training was compared to Co-Training (using random feature splits) and to two Self-Training algorithms. In the experiments, multiple classification data sets were used with 40%, 60% and 80% of unlabeled data. The performance metric was the final improvement of each algorithm compared to the accuracy of the supervised base learner. In the most extreme situation with 80% unlabeled data, Tri-Training outperformed the other algorithms on 11 out of 12 data sets using J4.8 decision trees as base learner. Using BP neural networks, Tri-Training obtained the highest improvement for only 3 data sets and using Naive Bayes for 7 out of 12 data sets.

Co-Forest
Since they were aware of the shortcomings of Tri-Training, Zhou and Li [59] extended their ideas by introducing the *Co-Forest* algorithm. In order to get more diversity in the ensemble, Co-Forest is initialized with the well-known ensemble algorithm Random Forest (subsection 2.1.1). First, all N random trees are trained on bootstrapped samples from originally labeled observations L. Then for each classifier h_i, where $i = \{1, \ldots, N\}$, the concomitant ensemble H_i is constructed by combining all

component classifiers except h_i itself. In every iteration, H_i predicts all unlabeled observations from U and selects some of these observations to add to training data of h_i for the next iteration. H_i uses majority voting to estimate its confidence for each observation x from U. The confidence should be higher than a predefined threshold θ. However, even when the confidence meets this requirement, some additional conditions should be met to avoid that too many observations from U will be selected. This would increase the risk of adding misclassifications to training data and affect performance negatively. These conditions are based on research conducted by Angluin and Laird [6]. They studied the relationship between the size of training data m, the noise rate in training data η and the worst-case error rate ϵ. Zhou and Li [59] used this research to formulate some additional conditions in the Co-Forest algorithm.

Let $\hat{e}_{i,t}$ be the error rate of concomitant ensemble H_i in iteration t. $L'_{i,t}$ is defined as a set of unlabeled observations from $U'_{i,t}$ where the confidence of H_i exceeds a predefined threshold θ and $W_{i,t}$ is the sum of the confidences of the observations in $L'_{i,t}$. Following [59], the following condition should be met in every iteration to continue the algorithm:

$$\frac{\hat{e}_{i,t}}{\hat{e}_{i,t-1}} < \frac{W_{i,t-1}}{W_{i,t}} < 1$$

Even if $\hat{e}_{i,t} < \hat{e}_{i,t-1}$ and $W_{i,t} > W_{i,t-1}$ are satisfied, it might still occur that the equation $\hat{e}_{i,t}W_{i,t} < \hat{e}_{i,t-1}W_{i,t-1}$ is violated if $W_{i,t}$ is much larger than $W_{i,t-1}$. Therefore, $L'_{i,t}$ is subsampled to force $W_{i,t}$ to be less than $\frac{\hat{e}_{i,t-1}W_{i,t-1}}{\hat{e}_{i,t}}$. This avoids selecting too many observations to predict for the next iteration. For more details on these conditions, please refer to [59]. All steps of Co-Forest can be found in more detail in Algorithm 4. The algorithm terminates when none of the component classifiers has improved during an iteration. This is determined by the estimated error of each concomitant ensemble in every iteration. During the initialization of the algorithm, the error is calculated using the out-of-bag estimation [17]. In the next iterations the error is measured using training data. This causes the error to be an underestimation, which can force the algorithm to stop too early since it falsely expects that no improvement can be made in further iterations.

Despite this comment, the experiments in [59] show promising results. For nearly all data sets with various percentages of labeled observations, Co-Forest improves the initial accuracy after incorporating unlabeled observations in the training process. A pairwise two-tailed t-test was performed to show that these improvements are significant under the significance level of 0.05. Furthermore, Co-Forest achieves higher average accuracy than multiple supervised and semi-supervised algorithms for almost all data sets. The average accuracy is measured using 10-fold cross validation. The experiment did not include any statistical test to confirm the expectation that Co-Forest outperforms the other algorithms significantly though.

2.2.3.4.5 Other committee-based algorithms

The previous committee-based algorithms were all extensions of the Co-Training algorithm. These algorithms mostly differ in the approach to construct a set of initial classifiers and methods to select confident predictions from a set of unlabeled observations, but all are methods that iteratively select unlabeled observations the ensemble is most confident about. Not all algorithms that combine ensemble learning and semi-supervised learning follow the structure of Co-Training though. This subsection

Algorithm 4 *Co-Forest*

Input: Labeled set L, unlabeled set U, confidence threshold θ and ensemble size N
Output: Ensemble classifier H

Training phase:

Construct a random forest consisting of N random trees
for $i = 1$ to N **do**
    $\hat{e}_{i,0} \leftarrow 0.5$
    $W_{i,0} \leftarrow 0$
end for
$t \leftarrow 0$
while at least one of the random trees changes **do**
    $t \leftarrow t + 1$
    for $i \in \{1, \ldots, N\}$ **do**
        $\hat{e}_{i,t} \leftarrow EstimateError(H_i, L)$
        $L'_{i,t} \leftarrow \phi$
        if $\hat{e}_{i,t} < \hat{e}_{i,t-1}$ **then**
            $U'_{i,t} \leftarrow SubSampled(U, \frac{\hat{e}_{i,t-1} W_{i,t-1}}{\hat{e}_{i,t}})$
            for each $x_u \in U'_{i,t}$ **do**
                if $Confidence(H_i, x_u) > \theta$ **then**
                    $L'_{i,t} \leftarrow L'_{i,t} \cup \{(x_u, H_i(x_u))\}$
                    $W_{i,t} \leftarrow W_{i,t} + Confidence(H_i, x_u)$
                end if
            end for
        end if
    end for
    for $i \in \{1, \ldots, N\}$ **do**
        if $\hat{e}_{i,t} W_{i,t} < \hat{e}_{i,t-1} W_{i,t-1}$ **then**
            $h_i \leftarrow LearnRandomTree(L \cup L'_{i,t})$
        end if
    end for
end while

Classification phase (majority voting):

$H^*(x) \leftarrow \arg\max_{y \in label} \sum_{i:h_i(x)=y} 1$

explains some committee-based semi-supervised learning algorithms that deviate from this structure.

Semi-Supervised MarginBoost (SSMBoost)
MarginBoost [64] is an extension of supervised ensemble algorithm AdaBoost (see Algorithm 2). It aims to maximize a cost function based on the margin of all observations. The margin of an observation can be defined as the difference between the total weight assigned to the correct label and the largest weight assigned to an incorrect label [64]. The larger this margin, the more confident the ensemble is about a classification. Since the definition of the margin depends on the actual label, it cannot be calculated but should be estimated for unlabeled observations. Therefore, MarginBoost was extended to a semi-supervised algorithm *SSMBoost* [30]. In this algorithm the cost function of MarginBoost was updated to estimate the margin for

unlabeled data as well. SSMBoost requires the base learner to be a generative model in order to allow the EM algorithm to estimate the margin for unlabeled data. This limits the applicability of the algorithm since it is not possible to use other supervised algorithms as base learner. d'Alché et al. [30] performed experiments that show that the algorithm is not able to improve AdaBoost on all datasets for various amounts of labeled observations. When at least 10% of the data is labeled, AdaBoost en SSMBoost obtain similar results on three data sets. However, SSMBoost outperformed AdaBoost when only 5% of the observations were labeled (which the authors indicate as the most realistic situation). On the Ringnorm data set SSMBoost achieved an average error rate of 6.9%, while AdaBoost obtained no less than an average error rate of 28.7% with 5% labels.

ASSEMBLE
Bennet et al. [11] introduced the adaptive semi-supervised ensemble algorithm *ASSEMBLE* to solve some limitations of SSMBoost. ASSEMBLE does not require a semi-supervised generative model as base learner, but suits any cost-sensitive classification model (i.e. a classification model that allows to weight observations). That makes ASSEMBLE much more applicable in general. It also maximizes the margin, but it does this by assigning pseudo-labels to unlabeled observations in every iteration. Pseudo-labels are predicted labels that are assigned to unlabeled observations by a chosen similarity metric and are considered as being correct. ASSEMBLE then uses both labeled- and pseudo-labeled observations to boost the ensemble, by focusing more on misclassified observations in the next iteration.

This basic idea of ASSEMBLE can be adjusted to many variations. One of the variations Bennet et al. [11] proposed, is *ASSEMBLE.AdaBoost*. It adjusts AdaBoost to allow unlabeled observations in the learning process by fitting it in the ASSEMBLE structure. During initialization, pseudo-labels are assigned by performing Nearest Neighbor [28] classification. Since no boosting has been performed yet, the initial weights parameter β of labeled observations is set much higher than the weights of unlabeled observations. It is likely that this will change during the boosting process, as misclassifications (based on labels and pseudo-labels) are assigned higher weights in the next iteration. As can be seen in Algorithm 5, the weights D_{t+1} for the next iteration $t + 1$ are calculated as in AdaBoost (Algorithm 2). Misclassifications get higher weights in the next iteration, to focus more on observations that are hard to predict. In step 15 of ASSEMBLE the unlabeled- and labeled observations are sampled to the size l of the originally labeled observations. This allows the algorithm to keep training times similar as AdaBoost. But due to the limited number of labeled observations, it can hurt the performance of the algorithm.

ASSEMBLE.AdaBoost uses all good properties of AdaBoost and just provides a framework to use boosting in a semi-supervised situation. In 2001 ASSEMBLE won the NIPS Unlabeled Data Competition using a decision tree as base learner. Experimental results from [11] also show that it performs as well or better than AdaBoost on several datasets using various number of labeled observations. It should be noted that this experiment does not include any statistical tests to verify whether improvements were significant or not. For the remainder of this thesis, we will refer to ASSEMBLE.AdaBoost as ASSEMBLE.

DECORATE
So far we have discussed ensemble methods bagging and boosting to construct diverse

Algorithm 5 *ASSEMBLE.AdaBoost*

Input: Labeled set L, unlabeled set U, ensemble size N, misclassification cost parameter β

Output: Ensemble classifier H

Let $l := |L|$ and $u := |U|$

Let $D_1(i) := \begin{cases} \beta/l & \text{if } i \in L \\ (1-\beta)/u & \text{if } i \in U \end{cases}$

Let $y_i := c$ where c is the class of the nearest neighbor point in L for $i \in U$.

Let $h_1 := \mathcal{L}(L+U, Y, D_1)$

for $t := 1$ to N **do**

  Let $\hat{y}_i := h_t(x_i)$, $i = 1, \ldots, l+u$

  $\epsilon = \sum_i D_t[y_i \neq \hat{y}_i]$, $i = 1, \ldots, l+u$

  if $\epsilon > 0.5$ **then**

    Stop

  end if

  $w_t = 0.5 * log(\frac{1-\epsilon_t}{\epsilon_t})$

  Let $H_t := H_{t-1} + w_t h_t$

  Let $y_i = H_t(x_i)$ if $i \in U$

  Update weights of training observations as in AdaBoost:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times U_t \text{ where } U_t = \begin{cases} \frac{\epsilon_t}{1-\epsilon_t} & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases} \quad \forall i \in \{1, \ldots, l\}$$

$$\text{and } Z_t = \sum_{i=1}^{l} D_{t+1}(i) \text{ as normalization constant}$$

  S = $Sample(L+U, l, D_{t+1})$

  $h_{t+1} = \mathcal{L}(S, Y, D_{t+1})$

end for

return H_{T+1}

ensembles that generalize well. Melville and Mooney [67] proposed a committee-based semi-supervised algorithm that generates artificial observations to make ensembles more diverse. *DECORATE* initially trains a classification model on labeled observations and then generates random artificial observations in each iteration. In order to ensure that the artificial data increases the diversity of the ensemble, the artificial observations are labeled so as to differ from the current ensemble. The artificial data is first labeled by the current ensemble. Then the inverse of this resulting probability is used to assign a label to each artificial observation. Next, a classifier learns from both originally labeled- and artificially labeled data. This process ensures the ensemble to be diverse, but artificial data can hurt performance as well. Therefore, a classifier is only added to the ensemble if the accuracy of the ensemble on training data does not decrease. Melville and Mooney [67] have shown that DECORATE outperforms Bagging, Random Forest and AdaBoost if the number of labels is small. On larger data sets, it still outperforms bagging and Random Forest and its results are competitive to AdaBoost.

2.2.4 Summary semi-supervised learning

In order to develop accurate supervised machine learning models, a sufficient amount of labeled observations is required. However, in many real-life scenarios it is time-consuming, expensive or difficult to gather such labels. Semi-supervised learning

algorithms aim to combine information from labeled- and unlabeled observations. This makes it possible to apply classification models in much more applications. Many semi-supervised algorithms have been developed, but most of them highly depend on assumptions (subsection 2.2.2).

Since supervised ensemble learning (section 2.1) has proven to generalize well [34], its ideas have been used in several semi-supervised algorithms over the years. It all started with Co-Training [12], which performance relies on the presence of two independent and redundant feature sets in the data. Next, some algorithms were proposed that try to adjust data to the Co-Training format by generating random- or artificial feature splits. In follow-up studies, limitations of these approaches were solved by constructing diverse ensembles which can handle unlabeled observations. Committee-based algorithms mostly deviate from each other in the way they introduce and maintain diversity in the learning process. Bagging, boosting and artificial observations are all studied and have shown promising results. The main question we will try to address during this research is whether these algorithms can compete with similar active learning algorithms.

2.3 Active learning

This final section of this background chapter will discuss the field of active learning. This technique is developed to cope with problems where it is expensive or difficult to obtain labeled data. Although active learning can be used in the same situations as semi-supervised learning, both techniques differ substantially. As semi-supervised learning seeks to use the distribution of large quantities of unlabeled data, active learning aims to query a minimum number of labels to a (usually human) *oracle* [3] to incorporate in the learning process of a classification model.

The oracle is supposed to provide the correct label for the queried observations based on his/her domain knowledge. Active learning algorithms aim to query observations that contain useful information for the classification model to add to the training data including its correct label. Using the most informative training data only, models might be able to distinguish different classes earlier. The most challenging part of active learning is to determine which unlabeled observations contain most useful information to improve the classifier. Many different *query strategies* have been proposed to select most informative observations from the unlabeled data. The query strategy is the main difference between most active learning algorithms. Much literature available about active learning is discussed in the literature survey by Burr Settles [85], which is highly recommended to read for more details on this topic.

Active learning algorithms have been applied in many different areas. Text classification is an application where active learning is often used [65, 93]. It can be very time consuming to classify large amounts of text into different groups based on its content if it needs to be read and interpreted manually. For example, if entire textual documents need to be reviewed manually to determine whether they meet certain requirements or not, it can be an expensive task to have them read by domain experts. Another common application is speech recognition [48]. For instance, when the objective is to translate audio files from rare languages, it can take quite some time to translate such files manually. Active learning can help in such situations by only requiring manual judgment for a few texts, documents or audio files.

In this section, first three scenarios will be introduced in which active learning can be used. Next, several query strategies will be explained which all search for the most informative observations in a different way. Finally, multiple practical considerations will be discussed that should be taken into account with any active learning application.

2.3.1 Active learning scenarios

Following literature, active learning can be divided into the following three scenarios [85, 3]:

- Membership query synthesis
- Stream-based selective sampling
- Pool-based sampling

In all scenarios the algorithm queries labels for unlabeled observations to an oracle. However, in each scenario labels are gathered and processed differently. Within each scenario, different query strategies can be used to search for the most informative observations. These strategies will be discussed in section 2.3.1.3.

2.3.1.1 Membership Query Synthesis

Membership query synthesis is a scenario where the model actively synthesizes observations from the entire input space, and does not necessarily sample from the actual data [5]. The problem of this technique is that such synthetic observations can be hard to interpret by human oracles. For instance, for an image recognition task the oracle is asked to provide a label to a selected image based on the content of the image. However, when a group of pixels is combined to generate a synthetic image, it is very likely that the new image is not meaningful for human oracles [85]. Stream-based selective sampling and pool-based sampling were introduced to overcome this problem.

2.3.1.2 Stream-based selective sampling

This technique randomly samples an observation from the available training data [24]. Then the classification model decides whether this sample is informative or not using a query strategy. If the sample meets the informativeness requirements of the classifier, it will be queried to an oracle and added to the training data including the provided label. The classifier can make this choice using different query strategies (see Section 2.3.1.3). Since observations are sampled one by one, it can be seen as a stream-based process.

The advantage of stream-based selective sampling compared to membership query synthesis is that the sample comes from the actual training data. For applications with limited memory or processing power, it might be beneficial to process observations one by one [85]. However, since the classifier can only make a query decision based on one sample, it cannot compare the sample to alternative samples that might be even more informative [65].

2.3.1.3 Pool-based sampling

Instead of sampling one observation at a time and deciding whether to query it or not, pool-based sampling ranks all unlabeled observations from a pool of unlabeled observations U based on the prediction uncertainty of the current model trained on labeled pool L [58]. The observation with most uncertain prediction is then queried to an oracle and added to labeled training set L including the provided label. The general pseudo-code of pool-based sampling can be found in Algorithm 6. Within this active learning framework there are many query strategies available.

Algorithm 6 *Pool-based sampling*

Input: Labeled data L, unlabeled data U, base learning algorithm $h(x)$, query strategy Q and number of observations to be queried v

1: Train initial classification model $h(x)$ on labeled observations of the form (x_i, y_i) from L
2: **while** stop-condition is not met **do**
3: Predict class labels for all observations in U using current classifier $h(x)$
4: Rank observations from U based on Q and store in R
5: Query the label for the v most uncertain observations from R to the oracle
6: Add v observations including new label to L and remove them from U
7: Train $h(x)$ on L
8: **end while**
9: Return model $h(x)$

The advantage of pool-based sampling compared to stream-based selective sampling is that it compares all unlabeled observations and it does not depend on random sampling from the underlying distribution. Since it needs to compare all observations from U every iteration, it is more computationally expensive though. Furthermore, it is more likely to select outliers. Since outliers behave differently than the general observations, the algorithm might be uncertain about their label and select them [58]. Outliers do not contain useful information about the separation of class labels, so they might influence the performance of the model negatively.

Despite these two limitations, pool-based sampling is the most common type of active learning. The ranking of all observations in U may use more computation time, but it is more likely to select informative observations that can improve classification models [85]. Therefore, the remainder of this chapter will use the pool-based setting to discuss active learning techniques.

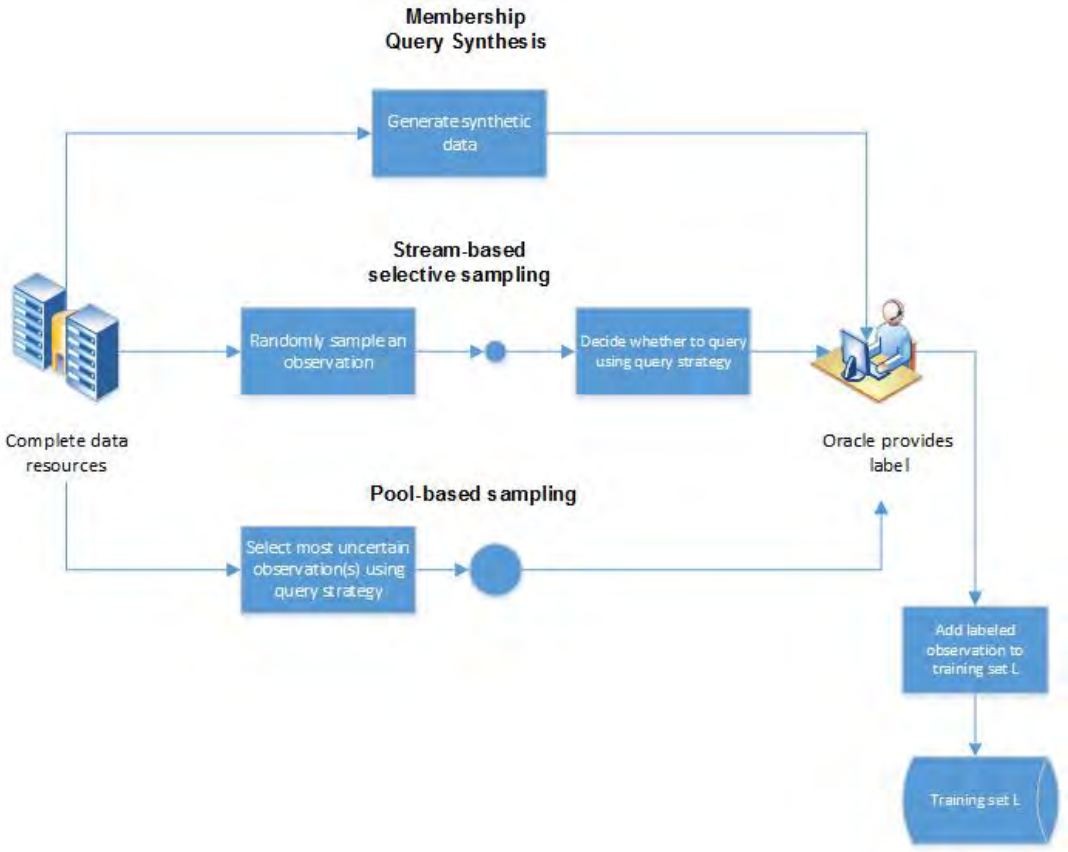


FIGURE 2.6: A schematic overview of three different active learning scenarios.

2.3.2 Query strategies

The available pool-based active learning algorithms mostly differ from each other in the query strategy that is used. These algorithms can again be divided into three categories based on the query strategy that they use [3]:

- *Heterogeneity-based models*
  Such models explore uncertain regions of unlabeled data or try to find observations that differ a lot from observations that have been used for training so far. These models do not focus on the possible influence of additional observations on the classifier performance.

- *Performance-based models*
  In contrast to heterogeneity-based models, performance-based models mainly focus on the possible effect of additional observations on the performance of the current classifier. They aim to measure the effect of querying a new observation to an oracle.

- *Representativeness-based models*
  Representativeness-based models seek to query observations that follow the underlying population of training instances as good as possible.

In section 2.2 we mostly focused on committee-based semi-supervised algorithms. These type of algorithms aim to select the *most* confident unlabeled observations and add them to the training set including their predicted label. This approach can be seen

as the opposite of active heterogeneity-based models, as they seek to query the *least* confident unlabeled observations including their provided label. Therefore, we will explain heterogeneity-based models like *uncertainty sampling*, *query-by-committee* and *expected model change* in more detail in this section. The performance-based- and representativeness-based models will be discussed more briefly.

2.3.2.1 Heterogeneity-based models

These models aim to query uncertain unlabeled observations in order to improve classifier performance. The models measure the uncertainty of predictions in different ways, which will be described in this section.

2.3.2.1.1 Uncertainty sampling

Probably the simplest and most common query strategy within active learning is *uncertainty sampling* [56]. It assumes that the most informative observation is the observation that the current classifier is most uncertain about. If the classifier would know its label, it would be easier to distinguish different classes. First, it trains an initial classification model using a small set of labeled observations L. This classifier predicts the label for all unlabeled observations in U. The classifier queries its most uncertain observation x^* to an oracle. This observation including its correct label y^* is added to the initial training set and finally the classifier is re-trained on the updated training set $L = L \cup (x^*, y^*)$. This process continues iteratively until a chosen stop-condition is reached.

In order to measure the uncertainty of the predictions, it is most straightforward to use probabilistic learning models as base learner. For binary classification, this would indicate that the observation with posterior probability of being positive closest to 0.5 will be queried [56, 57]. For problems with K classes ($K > 2$), the most uncertain point x^* can be found by measuring uncertainty as follows:

$$x^* = \arg\max_x 1 - P_\theta(\hat{y}|x) \tag{2.1}$$

where $\hat{y} = \arg\max_y P_\theta(y|x)$ (i.e. the class label with the maximum posterior probability for observation x under model θ).

A drawback of this uncertainty measure for problems with more than two classes is that it only takes the certainty of the most probable label into account. If the certainty of more labels is considered as well, it can be measured whether the classifier is easily able to chose between the most probable labels for an observation. *Margin sampling* [85] considers the two most probable class labels $\hat{y}_1$ and $\hat{y}_2$ by measuring the difference between their posterior probabilities. A small margin indicates that the classifier is uncertain about the choice between class labels $\hat{y}_1$ and $\hat{y}_2$:

$$x^* = \arg\min_x P_\theta(\hat{y}_1|x) - P_\theta(\hat{y}_2|x) \tag{2.2}$$

Probably the most used uncertainty measure is the *entropy* [87]. The entropy of observation x can be measured by using the Shannon entropy:

$$Ent = -\sum_{i=1}^{K} P_\theta(y_i|x) log(P_\theta(y_i|x)) \tag{2.3}$$

The higher this entropy value, the higher the uncertainty about the prediction for observation x. Therefore, the entropy in Equation 2.3 needs to be maximized to obtain most uncertain observation x^*.

In order to calculate the prediction uncertainty, the algorithm should provide class probabilities as output. Although the possibility exists to adjust certain algorithms to provide probabilistic outputs [93, 45], this requirement limits the applicability of uncertainty sampling. Another drawback is illustrated in Figure 2.7 from [85]. Since uncertainty sampling is only focused on the uncertainty of classifier predictions, it might occur that outliers are queried to the oracle. If such outliers are added to the training data, it will not help the classifier to separate classes more accurately.

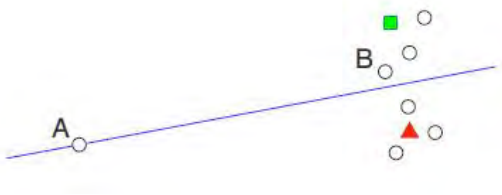


FIGURE 2.7: An example when selective sampling does not work. Since outlier A lies on the decision boundary, selective sampling will query it to the oracle. In this situation, unlabeled observation B contains more information to separate the triangle-class from the square-class though.

2.3.2.1.2 Query-by-committee

Another common query strategy within active learning is *query-by-committee* (QBC) [43]. This algorithm trains a committee of classifiers $H = \{h_1, \ldots, h_N\}$ that is initially trained on labeled observations from L. All component classifiers from H vote for the labeling of unlabeled observations from U. The observation that the component classifiers disagree most about is considered as most uncertain. If all component classifiers would predict the same label for each observation, it would not make sense to measure disagreement between them. Therefore, the committee should consist of diverse component classifiers [7]. The first two proposed approaches for generating a committee for QBC [7] are the *version space* approach and the *random sampling* approach.

Following Mitchell [69], if two models of the same algorithm but with different parameters agree on the predictions for all labeled observations, but disagree on a unlabeled observation, this unlabeled observation lies in the uncertainty region. Such observations are unknown to the overall model class (i.e. the version space) and will be queried by the classifier. Finding the version space is a very computationally expensive task, which makes it hard to use in practice. Random sampling is the standard approach in QBC [7]. It generates a committee by sampling the model parameters from a probability distribution which fits the training data. Fitting a probability distribution to the labeled training data is not a straightforward procedure, which limits the applicability of this approach as well.

To make QBC more applicable, Abe and Mamitsuka [2] used the well-known ensemble methods bagging (section 2.1.1) and boosting (section 2.1.2) to generate a diverse

committee for QBC. They introduced the algorithms *query-by-bagging* and *query-by-boosting* [2], which combine bagging and boosting with uncertainty measurements from QBC respectively. Melville and Mooney [68] proposed another technique to create a diverse committees within the QBC framework. Their algorithm *ACTIVE DECORATE* seeks to obtain diversity into the committee by incorporating synthetic observations into training data. Despite the good performance of this algorithm in experiments conducted by Melville and Mooney [68], synthesizing observations is less straightforward than the use of well-known techniques bagging and boosting. Therefore, especially QBag and QBoost are interesting for the purpose of this study due to their applicability and their similar committee generation techniques as some semi-supervised algorithms. Before discussing these algorithm in more detail, the most common uncertainty measures within the QBC framework will be explained.

Vote entropy is one approach to measure disagreement within the committee predictions and is related to the entropy measurement used within uncertainty sampling. Suppose $V(y_i)$ are the number of votes for class y_i ($i \in \{1, \ldots, K\}$) and N are the total number of component classifiers within the committee. Vote entropy substitutes the number of votes for a class label y_i with respect to the committee size N into the Shannon entropy from Equation 2.3. The higher the vote entropy, the more disagreement exists within the committee. Maximizing it, would result in the observation x^* with most disagreement:

$$x^* = \arg\max_x -\sum_i \frac{V(y_i)}{N} log \frac{V(y_i)}{N} \tag{2.4}$$

One limitation of vote entropy is that it does not take the confidence of the predictions into account, as it only considers a vote. *Jensen-Shannon (JS) divergence* [65] is an approach to overcome this limitation as it also considers the confidence of the predictions. Let w_i be the weight of the i^{th} classifier and P_i be the class probability distribution for observation x by the i^{th} classifier. Given the vote entropy $Ent(P)$ from Equation 2.3 and N component classifiers, JS divergence can be formulated as follows:

$$JS(P_1, \ldots, P_N) = Ent(\sum_{i=1}^{N} w_i P_i) - \sum_{i=1}^{N} w_i Ent(P_i) \tag{2.5}$$

High values of the JS divergence indicate high uncertainty, so this function needs to be maximized.

QBag

An important aspect of the QBC framework is to generate a diverse committee of classifiers. QBag applies bagging (Section 2.1.1), which applies bootstrap sampling on the originally labeled observations L. Each component learning algorithm is trained on one of the T bootstrap samples from $\{L_1, \ldots, L_T\}$ and predicts the class labels for unlabeled observations from U. In order to estimate the disagreement within the committee about these predictions, QBag uses the margin disagreement measure (see Equation 2.2). This is the difference between the number of votes for the most predicted class and the second-most predicted class. It does not measure the disagreement for all observations in U, but it draws a random sample of candidates of size R first. The observation with highest disagreement measure is queried to the oracle. After the oracle has provided the correct label for these observations, they are added to the current labeled set L and removed from U. Finally, the component classifiers are re-trained using new set L and this process continues until a predefined

stop-condition is met. After the last query, majority voting is applied to obtain a final hypothesis for a new observation. The pseudo-code of QBag can be found in Algorithm 7.

Algorithm 7 *Query-by-bagging*

Input: Labeled data L, unlabeled data U, base learning algorithm $h(x)$, number of queries N, number of bootstrap samples T, number of query candidates R

Initialization:

1: Apply bagging on L and obtain bootstrap samples $\{L_1, \ldots, L_T\}$

Training phase:

2: **for** $i = 1, \ldots, N$ **do**
3: Train $h(x)$ on each bootstrap sample $\{L_1, \ldots, L_T\}$ and obtain ensemble $H(x) = \{h_1(x), \ldots, h_T(x)\}$
4: Construct a set D containing R random observations from U
5: Pick observation x^* from D with smallest disagreement margin: $x^* = \arg\min_{x \in D} |\sum_{h_t(x)=0} H(x) - \sum_{h_t(x)=1} H(x)|$
6: Query the label y^* for x^* to an oracle
7: Add (x^*, y^*) to each sample in $\{L_1, \ldots, L_T\}$ and remove from U
8: **end for**

Classification phase:

Apply *majority voting* and obtain prediction $H(x)$

QBag mainly differs from the standard QBC framework during the generation of a committee. The variance of predictions in QBC is caused by the difference in model parameters, while the variance in QBag is caused by training on different bootstrap samples. Abe and Mamitsuka [2] have not explained why R candidates are sampled randomly, instead of measuring disagreement for all observations in U. This choice is probably made to reduce computation time of the algorithm, but it might have negative influence on the performance due to the random candidate selection.

Abe and Mamitsuka [2] compared QBag to active learning algorithm QBoost and supervised algorithms C4.5 [80] and boosted C4.5 using AdaBoost [42]. They defined a target error rate and analyzed when each algorithm reaches this error for multiple datasets. Their experiment shows that QBag reaches this target error much faster than C4.5 and AdaBoost on all data sets and slightly faster than QBoost on 5 out of 8 data sets. Although they conclude that QBag performs *significantly* better than C4.5 and AdaBoost [42], they do not make any comments about statistical tests to verify this statement. Besides that, they only focus on the comparison between active learning algorithms and supervised algorithms, but they do not discuss the difference between the results of QBag and QBoost.

QBoost

QBoost can be described as the active learning version of supervised algorithm AdaBoost. It iteratively queries the unlabeled observation x^* with the minimum disagreement margin between both class labels, adds it to labeled set L and trains the AdaBoost algorithm on the new set $L = L \cup (x^*, y^*)$. The difference with the QBag algorithm is that QBoost does not sample the initial labeled set, but it generates variance in its predictions by the weighted voting scheme of AdaBoost (see section 2.1.2).

In fact, QBoost trains a new AdaBoost model in each iteration and its predictions are used to query an uncertain observation. In this way, the AdaBoost algorithm is trained on a data set containing more useful observations every iteration. The final hypothesis of QBoost for an unseen observation is obtained by majority voting of the resulting AdaBoost classifier. The pseudo-code of QBoost can be found in Algorithm 8.

QBoost has shown that it is able to achieve similar performance as passive learners while using way less labeled data [2, 68, 88]. As mentioned in the previous section, QBoost needs more labels to reach a target error than the QBag algorithm though. No statistical tests were conducted for significant differences between algorithms.

Algorithm 8 *Query-by-boosting*

Input: Labeled data L, unlabeled data U, base learning algorithm $h(x)$, number of queries N, size ensemble model T, number of query candidates R

Training phase:

1: **for** $i = 1, \ldots, N$ **do**
2: Train an AdaBoost model using (L, $h(x)$, T) and get committee $H(x)$.
3: Construct a set D containing R random observations from U
4: Pick observation x^* from D with smallest disagreement margin:
$x^* = \arg\min_{x \in D} |\sum_{h_t(x)=0} H(x) - \sum_{h_t(x)=1} H(x)|$
5: Query the label for x^* to an oracle
6: Add x^* to L and remove from U
7: **end for**

Classification phase:
Apply *majority voting* and obtain prediction $H(x)$

2.3.2.1.3 Expected Model Change

This active learning approach differs from uncertainty sampling and QBC as is does not seek to query the most uncertain observation but it aims to select the observation that will have most effect on the current model if we would know its label. Aggarwal [3] states that this model belongs to the heterogeneity-based model category, since it queries the observations that are most different from the observations that the classifier has seen so far. However, as it measures the expected change of the model, it can be argued that it is linked to the performance-based models as well.

The expected effect on the current model can be measured by the change in the gradient with respect to the current model parameters. Therefore, this approach is only suitable for gradient-based models. Stochastic Gradient Descent is a well-known technique to find model parameters θ that minimize a chosen loss function [18]. It iteratively modifies parameters θ by calculating the negative gradient decent of the loss function $\ell(\theta)$ with respect to each training observation (x_i, y_i):

$$\theta := \theta - \alpha \frac{\partial \ell_{x_i}(\theta)}{\partial \theta} \quad \text{for } i = 1, \ldots, N \tag{2.6}$$

where α is the learning rate of gradient descent.

Let's consider now that we add a candidate observation x^+ to labeled training set L, and we are interested in the expected model change for new set $L^+ = L \cup (x^+, y^+)$. The expectation of the loss function can then be defined as the sum of the loss over training observations and the expected loss of the candidate observation:

$$\epsilon_{L^+} = \sum_{i=1}^{N} \ell[f(x_i, y_i)] + \ell_{x^+}[f(x^+, y^+)]. \tag{2.7}$$

Now the model change $H(x^+)$ after including x^+ needs to be measured using the gradient of the loss at x^+:

$$H(x^+) = \alpha \frac{\partial \ell_{x^+}(\theta)}{\partial \theta}. \tag{2.8}$$

$H(x^+)$ needs to be maximized over all observations from unlabeled set U to find the observation that results in the maximum change of the current model:

$$x^* = \arg\max_{x \in U} ||H(x)||. \tag{2.9}$$

However, in the situation of active learning the actual class label of observations from U is not known to measure the model change. Therefore, the expected change over all possible labels $y^+ \in \{y_1, \ldots, y_K\}$ is used to approximate the model change. The observation with maximum model change can then be defined as follows:

$$x^* = \arg\max_{x \in U} \sum_{k=1}^{K} P(y_k|x) || \frac{\partial \ell_{x^+}(\theta)}{\partial \theta} ||, \tag{2.10}$$

where $P(y_k|x)$ is the conditional probability of label y_k given observation x using the current model. For more details, please see [18].

Although it is an advantage that this query strategy measures the expected impact of each unlabeled observation on the current model, it can be computationally expensive if the data contains many features or if unlabeled set U is large [85].

2.3.2.1.4 Exponentiated Gradient Exploration for Active Learning

Recently, Bouneffouf [13] proposed a sequential active learning algorithm called *exponentiated gradient (EG)-active*. He states that this algorithm can improve any active learning algorithm by adding an optimal random exploration [13]. Earlier work from Osugi et al. [74] proposed a method to balance between exploitation and exploration in order to improve uncertainty sampling. Uncertain observations are often located near the obtained decision boundary using the current model [74]. Heterogeneity-based models exploit the region close to this decision boundary to provide more information to the model about this uncertain region. However, if the model has not seen any labeled observations from one of the classes, it can occur that the model provides misclassifications for this entire class. This situation is illustrated in Figure 2.8 from [74]. The purpose is to separate the negative class (blue circles) from the positive class (red crosses) as accurately as possible. Suppose that an active learning algorithm has not seen any of the labeled negative observations from the class in the lower-right corner of the input space. It will probably generate a decision boundary that separates the negative class in the upper-left corner from all other observations. Standard active learners will only exploit the region around this decision boundary

to make it more accurate, but they will not explore further regions (the negative observations in the lower-right corner).

Therefore, Osugi et al. [74] proposed a modification to uncertainty sampling. Every iteration, a random choice is made between exploitation (e.g. active query strategy) and random exploration (e.g. querying observations from unseen regions of the input space). By adding exploration into the active learning process, unseen regions like the lower-right negative class in the example might be learned better. The probability of choosing exploration rather than exploitation is updated based on classification improvement after adding explored observations in previous rounds. Despite this updating rule, no optimization techniques were used to update the probability of using exploration. Bouneffouf [13] proposed a method that applies exponentiated gradient [53], which increases this probability if it leads to an improvement. Another improvement compared to [74] is that (EG)-active can be used to improve *any* active learning algorithm instead of uncertainty sampling only. This approach is most suitable when the target classes are spread over many different regions in input space and the active learner has not seen every region yet [74].

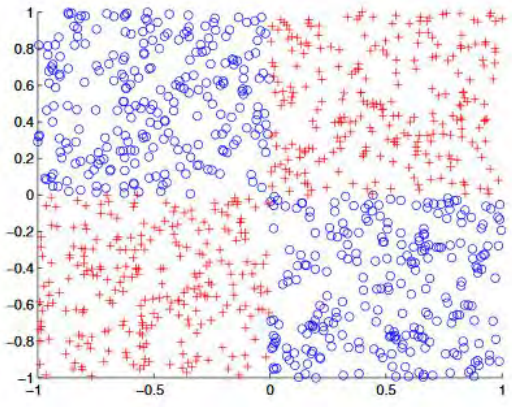


FIGURE 2.8: An example training set where it can be beneficial to combine active- exploitation and exploration. The blue circles are negative observations and the red crosses are positive observations.

2.3.2.2 Performance-based models

Instead of focusing on uncertain observations, performance-based models seek to measure the possible effect of adding new observations to the training data of the current classifier. The advantage of this approach is that it is less likely to query outliers which do not represent the labeled training set.

2.3.2.2.1 Expected Error Reduction

This model is not focused on measuring the expected model change, but it measures the expected reduce of the models generalization error. It uses the unlabeled set U as validation set and queries the observation that will minimize the error if it would be added to labeled training set L. The problem of this approach is that the current model adds each unlabeled observation from U to L iteratively to measure the expected error. This means that for every query, the model needs to be re-trained for all unlabeled observations in U. And for each of these observations the correct label is not known, so the expected error needs to be measured for every possible labeling

of the observations. This makes this model extremely computationally expensive, which makes it hard to apply in practice [85, 3].

2.3.2.2.2 Expected Variance Reduction

In order to reduce the computational effort of expected error reduction, the model can be adjusted by just focusing on variance reduction. The generalization error can be expressed as the sum of the label *noise*, model *bias* and *variance* [46]. The noise depends on the method to obtain labels, as it might occur that some effects in this process add noise to the labels of the data. The bias term is the error caused by the chosen model class. If the model class (e.g. a linear model) does not fit the data perfectly (e.g. a non-linear), the related error is expressed in the bias term. Finally, the variance term expresses the variance of the classification model. Changing the parameters of a classifier can have effect on the variance term, but will not have effect on the noise and bias. Since we have most influence on changing the parameters, we can aim to only minimize the variance instead of the complete generalization error. Since variance can be expressed in closed form for multiple classification models [25, 62], computational complexity can be reduced compared to the expected error reduction model. Despite this improvement, it should be noted that these models are still much more inefficient than simpler techniques as uncertainty sampling and QBC [85].

2.3.2.3 Representativeness-based models

The main drawback of heterogeneity-based models is their sensitivity to outliers. Representativeness-based models are developed to overcome this problem. These models seek to query unlabeled observations when they follow the general distribution of all observations in U only. Density-based models can be used to estimate the underlying distribution of U. If the current model is very uncertain about candidate observation x^+, but it deviates too much from the distribution of the other unlabeled observations, it is less likely to be queried. The representativeness component is multiplied to the loss function of a heterogeneity-based model $H(U)$ like uncertainty sampling or QBC:

$$O(x^+, U) = H(x^+) \times R(x^+, U) \tag{2.11}$$

The term $R(x^+, U)$ can be any measure that defines the similarity between candidate observation x^+ and the other unlabeled observations from U. McCallum and Nigam [65] added the generative model EM to uncertainty sampling to capture the distribution of unlabeled observations better. Nguyen et al. [71] performed a cluster algorithm before applying uncertainty sampling. Under the assumption that observations from the same cluster are likely to belong to the same class [19], they avoid querying observations from the same class every iteration. As these models are based on heterogeneity-based models, they are less computationally expensive than performance-based models. The applicability of these models highly depends on the chosen similarity measure to verify the representativeness of a candidate observation.

2.3.3 Practical considerations for active learning applications

In order to meet the active learning objective to train classification models more efficiently by focusing on informative observations, several choices should be considered regarding the query strategy and the oracle. To make active learning useful in practice, there should be a trade-off between costs to query observations and classification

performance. In this section the main practical considerations are discussed which can have major effect on this trade-off.

2.3.3.1 Noisy oracle

One of the main assumptions of active learning is that the oracle is always able to provide the correct label for any unlabeled observation. However, this assumption might be violated in real-life situations. Observations can contain features that make it hard to classify them. These features can be difficult to interpret by an automated oracle (i.e. data is labeled by chemical- or biological experiments) or even by human domain experts. Human oracles might also provide noisy labels because of tiredness or distraction during the labeling process [85]. Donmez et al. [35] assigned different noise levels to multiple oracles and took this into account in the query process. In practice it might be good to consider to compensate for the risk of noisy oracles. However, for the purpose of this thesis it is assumed that the oracle is able to provide correct labels for each query.

2.3.3.2 Batch queries

Besides focusing on classification performance, the computational effort of active learning algorithms is often of great importance in practice. If models are re-trained after adding each queried observation, active learning can be computationally expensive if the amount of features or observations is high. In such situations, one can choose to query batches of observations instead of single observations each iteration. In this way, more newly labeled observations are added to the training set every iteration and the number of times the model needs to be re-trained can be reduced. Despite the computational advantages, classification performance might suffer from batch querying. If a single observation is being queried, it can provide useful information for a certain region in input space. The next iterations, the model can query an observation from a different uncertain region using the obtained information from previous queries. Using single queries is therefore more efficient, as it does not query observations from the same uncertain region in one iteration.

2.3.3.3 Stop-condition

In most active learning algorithms we will discuss, the model keeps querying observations until it reaches a predefined maximum number of queries. This raises the question after how many queries the model needs to stop. In every active learning application, the ideal trade-off between labeling effort and classification performance should be considered. Active learning algorithms can stop when the improvement of the model within an iteration gets below a certain threshold. Another possibility is to define a labeling budget which may not be exceeded. The active learning algorithm will then proceed until the labeling budget has been reached. In real-life applications this is the most common stop-condition [85], but it might also be combined with other stop-conditions.

2.3.4 Summary active learning

Active learning is a machine learning field that is commonly used when it is expensive or hard to label data. It follows the same objective as semi-supervised learning as it incorporates unlabeled observations into the training of classification models to improve their performance. There are major differences between both techniques

though. Semi-supervised algorithms try to expand training data with unlabeled data that follows the underlying distribution of the labeled data as good as possible. Active learning explores uncertain regions of unlabeled data to expand the classifiers knowledge of the input space. The available active learning algorithms mostly differ from each other in the way they select informative observations (i.e. the query strategy). This chapter introduced many terms used within active learning. In order to make the discussed approaches more clear, we summarized them in Table **??**. Please note that this table only includes techniques within the pool-based sampling category of active learning.

The performance of active learning algorithms highly depends on the application it is used for. It is important to find a good trade-off between labeling costs, classification performance and computational complexity. Performance-based models can be very useful when focusing on querying as few as possible observations to the oracle. Simpler heterogeneity-based models are much more applicable in real-life situations due to the lower computational effort of the algorithms. For this reason, this thesis will focus on heterogeneity-based models and in particular on algorithms that follow the QBC framework. QBC algorithms use a disagreement measure within a committee of classifiers to query labels for uncertain observations. This can be seen as the complement of semi-supervised committee-based algorithms, which use unlabeled data the component classifiers do most agree about.

2.4 Summary

This chapter has discussed multiple algorithms within the fields of semi-supervised learning and active learning. In order to compare algorithms from both fields in this study, multiple algorithms were selected based on 1) their performance in previous studies and 2) their applicability in real-life scenarios. Committee-based models incorporate well-known ensemble techniques into semi-supervised learning and active learning, which makes them relatively easily applicable in practice. Furthermore, little effort has been made to compare committee-based models so far. Therefore, the aim of this thesis is to study the performance of committee-based algorithms within semi-supervised learning and active learning in different scenarios regarding the available labeled data.

First, the semi-supervised learning algorithms Co-Forest (Algorithm 4) and ASSEMBLE (Algorithm 5) were selected for our experiments. Experiments [11, 59] have shown that both algorithms are able to improve supervised algorithms and other semi-supervised algorithms on average, although the results were not validated with statistical tests. Co-Forest and ASSEMBLE generate diverse committees by applying well-known techniques bagging and boosting respectively. This makes them easier to implement than algorithms like DECORATE (Section 2.2.3.4.5), which generates artificial observations to construct diverse committees. Both algorithms seek to incorporate the most confident predictions from an unlabeled pool of observations into the training process.

The complement of committee-based semi-supervised algorithms is the active learning framework QBC (Section 2.3.2.1.2). Within this framework, several techniques are available to generate a diverse committee first. Algorithms QBag (Algorithm 7) and QBoost (Algorithm 8) use bagging and boosting respectively to construct the initial

committee of classifiers. Abe and Mamitsuka [2] introduced these algorithms to make QBC easier to implement in practice, which is the main reason that these algorithms were chosen. In contrast to the chosen semi-supervised algorithms, QBag and QBoost aim to select the most uncertain predictions and query their corresponding label to an oracle. The selected algorithms and the corresponding ensemble techniques to generate committees can be found in Table 2.1.

Committee generation	**Semi-supervised learning**	**Active learning**
Bagging	Co-Forest	QBag
Boosting	ASSEMBLE	QBoost

TABLE 2.1: The algorithms that were selected for this study

Chapter 3

Experimental setup

To answer the research questions of this study, multiple selected algorithms were applied on various datasets. The algorithms were implemented using the open source programming language Python, version 2.7.12. Due to the time expensive computations of the experiments, they were run on the Lisa system installed and maintained by SURFsara [91]. The Lisa system is a cluster computer consisting of several hundreds of multi-core nodes running the Linux operating system [91]. As a student at the Vrije Universiteit, it was possible to gain access to this system.

3.1 Algorithms

As discussed in Section 2.4, multiple committee-based algorithms from semi-supervised learning and active learning were selected based on their applicability and their performance in previous studies. Since the selected algorithms use bagging and boosting in combination with unlabeled data, the supervised algorithms Random Forest and AdaBoost were used as benchmarks as they use these ensemble techniques as well. To generate the initial committee, all algorithms make use of a base learner. Following Breiman [16], weak classifiers are suitable to use as base learner to make committees diverse, as small changes to training data can result in large changes in their predictions. He explained that neural networks and decision trees are such weak algorithms for which bagging and boosting can improve classification performance. Therefore, a decision tree was used as base learner for all algorithms implemented in this research. In all experiments the optimized version of the C4.5 algorithm, provided by the Python package Scikit learn [75], was used to train decision trees. Besides using the C4.5 algorithm as base learner, it was used as supervised benchmark as well. However, the decision tree was able to use the fully labeled datasets.

3.2 Datasets

Multiple datasets were used in the experiments. The datasets *Wisconsin Diagnostic Breast Cancer* (WDBC) and *German credit* were obtained from the UCI Machine Learning Repository [60]. Both datasets originate from a different domain (i.e. health care and banking) and have different data properties like number of observations, number of features and balance between target classes. The artificial *Two Moons* dataset was generated using the open source Python package Scikit-learn [75]. Table 3.1 provides an overview of the properties of all datasets. More details about each dataset can be found in the coming subsections and all features are listed in Appendix A. No further data preparation tasks like feature engineering and outlier removal has been performed, as it is assumed not to be relevant for the purpose of this study.

3.2.1 Wisconsin Diagnostic Breast Cancer

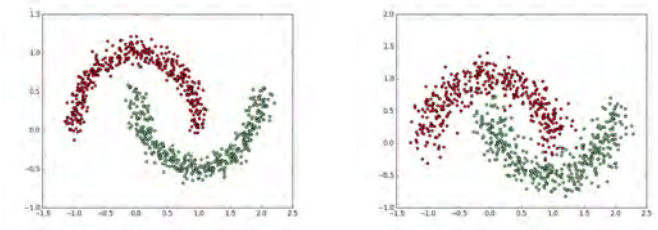
The WDBC dataset was first developed to increase the speed, correctness and objectivity of the diagnosis process for breast cancer [90]. In total, 569 images were taken of small drops of fluid from breast tumors. All images were analyzed, which resulted in ten real-valued features about the cell nucleus. The mean, standard error and largest values of these features were computed, resulting in 30 numerical features. These features can be found in Appendix A. The information was stored into a dataset suitable for machine learning classification models. Out of the 569 tumors, 212 (37%) were labeled as malignant and 357 (63%) were labeled as benign. The objective is to predict whether a tumor is malignant or benign.

3.2.2 German Credit

The german credit dataset was generated to predict whether a customer is 'good' (i.e. no credit risk) or 'bad' (i.e. credit risk) according to personal- and credit features [49]. The dataset contains 1000 rows, where every row represents a customer who takes credit at a bank. The class distribution is imbalanced, as only 30% of the customers is labeled as credit risk. The original dataset contains both numerical and categorical features. Since the Decision Tree algorithm available in the Python Scikit-learn package cannot handle categorical features, another available dataset was used where the categorical features are converted to numerical features using label encoder. The conversion of categorical features to numerical values when the categories have no natural ordering, might result in a slight bias. Since the same dataset was used for all algorithms, it was assumed that this would not affect the results in favor of any algorithm. Appendix A shows all features and their possible values. Please refer to [49] for more details about this dataset.

3.2.3 Two Moons

The two moons dataset can be generated using the Scikit-learn package in Python [75]. This dataset consists of two artificially generated half circles. It is often used to illustrate clustering and classification algorithms due to its simple two-dimensional structure. Furthermore, it is possible to add Gaussian noise to the data to verify whether classification models are prone to noisy data. For our experiments we generated the dataset *Two Moons (0.08)*, where the noise parameter (i.e. the standard deviation of the added Gaussian noise) was set to 0.08. In order to study the behavior of the algorithms when both classes are less easily separable, the *Two Moons (0.15)* was generated with the noise parameter set to 0.15. A visualization of both datasets is shown in Figure 3.1.



(a) (b)

FIGURE 3.1: Visualization of the datasets (a) Two Moons (0.08) and (b) Two Moons (0.15).

Dataset	Num. observations	Num. features	Num. classes	Percentage minority class
WDBC	569	32	2	37%
German credit	1000	20	2	30%
Two moons (0.08)	1000	2	2	50%
Two moons (0.15)	1000	2	2	50%

TABLE 3.1: Selected datasets including their properties used for this study.

3.3 Experiments

In order to answer the research questions of this study, multiple experiments had to be performed to compare semi-supervised learning and active learning as fair as possible. As both techniques combine unlabeled- and labeled data in a different manner, it is hard to compare them without giving any advantage to one of them. In this research, it was assumed that the techniques were compared as fair as possible by providing exactly the same number of labeled observations to both of them.

Semi-supervised algorithms are initially trained on the available set of labeled observations and add unlabeled observations including their predicted pseudo-labels to the training set in each iteration. The active learning algorithms are initially trained using only a few labeled observations and try to improve performance by requesting labels until a certain labeling budget has been reached. The advantage of active learning is that it can select useful observations to use for training, while the performance of semi-supervised learning algorithms depend on the randomly selected initial training set. However, semi-supervised learning is able to use additional confident predictions to enhance its training data, while active learning is restricted to use the selected labeled data only. In order to minimize the influence of the random initial training set for semi-supervised learning, all experiments were performed 500 times.

In order to validate the performance of each algorithm, the entire dataset was randomly split into a training set containing 80% of the data and a test set containing the remaining 20% of the data using stratified sampling to ensure that the class distributions remained the same. These training- and test sets for each dataset were used in all experiments, to ensure that model validation was always performed on independent observations that were not used to train the models before. Since these datasets are usually used for supervised classification tasks, a certain amount of known labels were randomly removed in each run to make it suitable for the purpose of this study. The experiments were performed using 5%, 10% and 20% of the available labels in the training set. For the remainder of this thesis, we will correspond to these percentages as the 'labeled rate' or 'budget'.

3.3.1 Performance metric

Since some of the datasets have an imbalanced class distribution, the accuracy paradox [109] might occur when the accuracy (i.e. the ratio of correct predictions to the total number of predictions) is used as performance metric. Therefore, the F1-score (also known as the F-measure) was used as performance metric in all experiments. The F1-score takes the balance between true positives (TP), false positives (FP), true

negatives (FN) and false negatives (FN) into account, which can make the performance measure fairer. The F1-score can be calculated as follows:

$$\text{F1-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

where

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \text{ and recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Since the WDBC dataset and the German Credit dataset have imbalanced class distributions, it is important to define which target class is assumed to be the 'positive' class. For the WDBC dataset the objective is to use machine learning to detect malignant tumors. Therefore, the malignant target class was assumed to be the positive class. For the German Credit dataset, the main focus is to identify customers who might have a credit risk. The customers with credit risk were therefore assumed to be the positive class for this dataset.

3.3.2 Size of initial training data active learning

In the standard active learning setting, the algorithms would initially be trained using just a few labeled observations and they would request labels for unlabeled observations until the predefined budget has been reached. Suppose the initial training set consists of only 10% of the total (limited) labeling budget. Using such a limited initial training set, it might be hard for committee models to select unlabeled observations that can provide useful information to distinguish the target classes. In some situations, it might be beneficial for active learning algorithms to start with more randomly selected labeled observations to be better able to query informative data from the start of the algorithm. Therefore, an additional experiment was conducted to find the best starting point for each active learning algorithm. For all labeled rates (5%, 10% and 20%), QBag and QBoost were provided 10%, 25%, 50% and 75% random observations from these labeled rates as initial training set. For instance, using a labeled rate of 5% the starting points of 10%, 25%, 50% and 75% are equivalent to 0.5%, 1.25%, 2.5% and 3.75% of the total training data respectively. From these starting points, the algorithms could request labels until the labeled rate was reached. Table 3.2 shows all label percentages used in this experiment. It should be noted that this additional analysis was performed using the separate test set as performance validation. This might cause that the chosen sizes of the initial training data were not entirely independent of the final performances of the algorithms.

		Labeled rate		
		5%	10%	20%
Initial size	10%	0,5%	1%	2%
	25%	1,25%	2,5%	5%
	50%	2,5%	5%	10%
	75%	3,75%	7,5%	15%

TABLE 3.2: All labeled percentages used in active learning experiments

3.4 Hyperparameter tuning

In order to boost the performance of each algorithm, their hyperparameters were optimized first. Grid-search was used to find the best set of hyperparameters from a manually chosen subset of the hyperparameter space of each algorithm. These candidate hyperparameter values for supervised algorithms C4.5, AdaBoost and Random Forest were chosen based on recommendations from Scikit-learn [75], as their implementations were used in our experiments. The subsets of hyperparameter values for the semi-supervised learning and active learning algorithms were obtained from literature [11, 59, 2]. Grid-search uses brute-force search to enumerate all possible candidate sets of hyperparameters [97] and validates their performance using cross-validation on the 80% training data [50].

Since we were mostly interested in the hyperparameters that result in best performance using limited number of labels, grid-search was performed for each algorithm on every dataset using labeled rates 5%, 10% and 20% separately. This resulted in many different sets of hyperparameters, which can be found in Appendix B. In some cases multiple hyperparameter sets achieved similar results in terms of the F1-score, but differed in terms of computational effort. In such situations, the hyperparameters with lowest runtime were selected.

3.4.1 Hyperparameters base learner

Besides the use of the C4.5 algorithm as supervised benchmark, it was also used as base learner for all other algorithms. Preliminary runs showed that the resulting hyperparameters values from the grid-search in the fully supervised scenario did not necessarily yield the best results for the other algorithms with C4.5 as base learner.

Boosting algorithms AdaBoost, ASSEMBLE and QBoost seemed to perform best when the decision tree hyperparameters did not allow the tree to become very large. If the size of the decision tree is not limited by its hyperparameters and the available training data is sparse, it is very likely that it will overfit the training data. As explained in Section 2.1.2, AdaBoost assigns weights to the observations and to its component classifiers using the training error. If the component classifiers (i.e. decision trees) overfit on the training data, the training error might be extremely small or even equal to zero. In the latter scenario, the weights will not be updated properly. As shown in Algorithm 2, the update rule for the weights of each component classifier is:

$$-log(\frac{\epsilon_i}{1-\epsilon_i})$$

If the training error ϵ_i would be equal to zero, this update rule would be undefined. In this scenario, the AdaBoost implementation of Scikit-learn [75] assigns a weight of zero to the component classifier that achieves perfect training accuracy. If all component classifiers obtain perfect classification, the implementation assigns a weight of one to the first component classifier and no weight to the remaining component classifiers. This indicates that the committee consists of only one overfitting decision tree, which can limit the performance of the model heavily. To avoid using decision trees with a training error of zero, it is important to force the C4.5 algorithm to generate decision trees of limited size.

In contrast to boosting, the bagging algorithms QBag and Co-Forest suffered from limitations on the size of decision trees. Although preliminary runs showed that the random component trees do overfit the small set of training observations, the combination of these random trees performed much better when trees were allowed to become large. Since all random component trees were trained on different bootstrap samples, it often occurred that multiple trees disagreed about their predictions. In this way, the combination of multiple overfitting component classifiers resulted in a committee model which did not overfit the small training set.

Due to this different behavior of the algorithms regarding the hyperparameters of the C4.5 algorithm, the values for hyperparameters *max_depth* and *min_samples_leaf* were adjusted for the boosting and bagging algorithms separately. As the name indicates, the max_depth limits the maximum depth of the tree. If no value is assigned to max_depth, nodes are expanded until all leaves are pure or until leaves contain less than the minimum number of samples required to split an internal node [75]. The hyperparameter min_samples_leaf denotes the minimum number of samples required to split an internal node [75]. For boosting algorithms max_depth was set to 2 and min_samples_leaf was set to 5 to avoid overfitting of the separate decision trees. For bagging algorithms Co-Forest and QBag the max_depth was not limited, while the min_samples_leaf was set to 1.

3.4.2 Committee size

The committee size is another important hyperparameter to be taken into account. Following Zhou et al. [107], large committees do not always obtain better performances than small committees. If the committee consists of many component classifiers while the training data is sparse, the risk of overfitting the training data can be higher than when using smaller committee sizes. Besides that, using large committees results in more computational effort. To limit the runtime of the many experiments and to lower the risk of overfitting the small training sets, the possible committee sizes in the grid-searches were set to 10, 20 and 50.

3.4.3 Initialization of the ASSEMBLE algorithm

Besides the hyperparameters of its component classifiers, multiple hyperparameters need to be set for the initialization of ASSEMBLE. During the initialization, nearest neighbor classification is used to assign labels to unlabeled observations. During the grid search, nearest neighbor classification using majority voting from Scikit-learn [75] and KD Trees [98] were used to find the nearest neighbors of each unlabeled observation. For both techniques, the number of neighbors were taken into account in the grid search. For the KD Trees, the leaf size was varied in the grid search as well. Finally the values for hyperparameter β (see Section 2.2.3.4.5) were varied to estimate the best initial weights for the unlabeled observations.

3.4.4 Number of active learning queries per iteration

In Section 2.3.3.1 the concept of batch querying was introduced. In each iteration, active learning algorithms select the most uncertain predictions and query them to an oracle. One important question is how many observations should be queried each iteration. To speed up the labeling process, multiple observations can be queried at a time. The risk is that the algorithm selects multiple observations from the same uncertain area of the input space [85]. In this way the labeling budget would not be

used as efficient as the algorithm does not explore different uncertain regions each iteration. Figure 3.2 illustrates the possible problem of inefficient batch querying that might occur. This figure was generated during one of the QBoost runs on the Two Moons (0.08) dataset. In figure (a) it is obvious that the current model is not able to separate the red observations from the green observations accurately yet. QBoost queries the yellow marked observations to an oracle and adds them including their correct (green) labels to the training set. Although figure (b) shows that the decision boundary has improved for that specific region, it might have gained the same result by querying only one of these uncertain observations. This would have saved budget to label two labels in different uncertain regions.

However, batch querying might be useful when the model forces the queried observations to be located in different areas of the input space. In this way different uncertain areas can be explored, while saving computational effort by reducing the number of times the model needs to be retrained. However, since we have not further investigated any possibilities to achieve this, both active learning algorithms were allowed to query one label per iteration in all experiments.

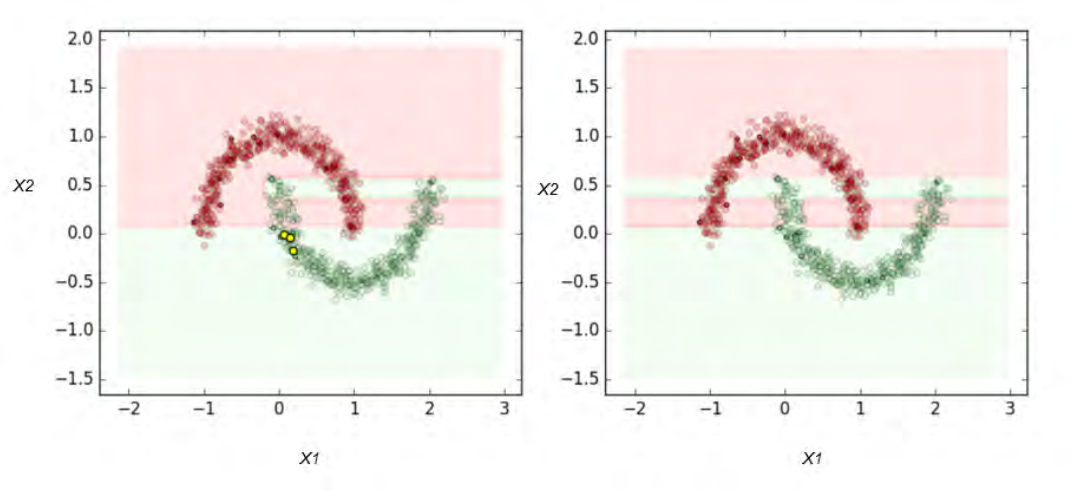


FIGURE 3.2: An illustration of inefficient batch querying for active learning. The border between the red plane and green plane is the decision boundary of the current classification model. Figure (a) illustrates the decision boundary of the model *before* querying three new labels. Figure (b) illustrates the adjusted decision boundary *after* re-training including newly gathered labels.

3.5 Model validation

The experiments described in Section 3.2.3 resulted in 500 F1-scores for each algorithm, each dataset and each labeled rate. To validate the performance of each algorithm, the average F1-scores with a 95% confidence intervals were calculated for each situation first. Bootstrapping [38] was used to estimate the confidence intervals, as this technique does not assume normality of the samples. Every sample of 500 F1-scores was obtained using a different subset of labeled- and unlabeled observations from the training set, which is equivalent to bootstrapping. The 95% confidence intervals were then computed for each sample X as follows:

$$CI_{lower}(X) = max(0, percentile(X, 2.5\%))$$
$$CI_{upper}(X) = min(1, percentile(X, 97.5\%))$$

Although the average F1-scores and confidence intervals certainly give insight in the performance of each algorithm, no conclusions can be drawn about any significant difference between their performances without any statistical tests. The usual approach to test for significant difference between two groups (i.e. algorithms) without assuming normality of the groups is the *Wilcoxon Signed-Rank* test. As six algorithms were compared in this study, this test cannot be used. The *Kruskal-Wallis test* [54] is a non-parametric test (i.e. it does assume normality) and can be used for comparing *two or more* independent groups of equal or different sizes [54]. The null-hypothesis of this test is that the mean ranks of the groups are the same. Rejecting the null-hypothesis indicates that at least one group stochastically dominates one another group [54].

The Kruskal-Wallis test does not provide information about which group is stochastically dominant though. Since we are mostly interested which algorithm performs best in each situation, post hoc *Dunn's test* [37] was applied to analyze pairwise comparisons. For both the Kruskal-Wallis test and the Dunn's test a significance level of 5% was used. The problem of analyzing pairwise comparisons, is the risk of a high Type I error. A Type I error is the incorrect rejection of a true null hypothesis [100]. A high Type I error in this study would mean that there is high risk of finding significant differences between two algorithms, while actually the difference is not significant under a chosen significance level. When only one hypothesis is tested under significance level 0.05, it means that there is a probability of 0.05 of making a Type I error. However, we aim to test $\binom{6}{2} = 15$ hypothesis in this study. This means that there is a probability of making a Type I error of:

$$\begin{aligned} P(\text{Type I error}) &= 1 - P(\text{no significant results}) \\ &= 1 - (1 - 0.05)^{15} \\ &\approx 0.54 \end{aligned}$$

There are several methods available to adjust the significance level to lower the probability of a Type I error. The *Bonferroni method* [36] corrects the significance level by dividing significance level α by the number of hypothesis. However, the Bonferroni correction can be quite conservative, which might lead to high false negatives [9] (i.e. concluding non-significance while there is a significant difference between two algorithms). The *False Discovery Rate* (FDR) [9] has proven to have less negative effects on the false negative rate. The FDR estimates a rejection region by assuring that FDR $< \alpha$ on average [9]. Therefore, the FDR correction was used to adjust the p-values from Dunn's test.

Chapter 4

Results

In this chapter the results of this study will be discussed. First, the results of the analysis on different starting points for active learning algorithms will be described. Next, the performances of all algorithms will be compared for each dataset separately. Finally, the results of the statistical tests will be explained to be able to draw conclusions about the significance of the results.

4.1 Size of initial training data active learning

As explained in Section 3.3.1, we analyzed whether the initial number of available labeled observations influences the performance of active learning algorithms. Figure 4.1 shows the average progress of QBag and QBoost on all datasets using labeled rates (i.e. budget) of 5%, 10% and 20% and 500 runs. Interestingly, from both Two Moons figures we can see that QBoost performed much better when using a relatively large part of the budget as initial training set. This result is somewhat counterintuitive, as QBoost was expected to be able to select a training set that is more informative than a randomly selected initial training set (despite the fact that these random training sets contained more observations). This might be partly explained by the inability of AdaBoost to correctly update the weights of the component classifiers and the weights of the observations on extremely small training sets (see Section 3.4.1), which is the component classifier of QBoost. If the initial committee is not diverse enough due to overfitting, QBoost will not be able to select informative unlabeled observations that can improve classification. The algorithm might then select observations from the same area of the input space, while the random selected training size is more spread out over different areas. The labeling starting point did not have much effect on the performance of QBoost for datasets WDBC and German Credit. A possible explanation for this might be that QBoost needs more budget than 10% anyway to obtain satisfying results. Changing the starting point for labeled rates 5% and 10% would therefore not have much effect on the algorithms performance.

Interestingly, QBag shows opposite results as QBoost. It obtained best results using only 10% or 25% of the budget as starting point for datasets WDBC, Two Moons (0.08) and Two Moons (0.15). This indicates that QBag only needs a few initial labeled observations as initial training set. By requiring only a small part of its labeling budget, the algorithm can spend it more efficiently by requesting useful labels for the remaining budget. Only for the German Credit dataset QBag performed slightly better using 75% of its budget as initial training set. This result may be due the imbalanced class distribution of German Credit. The algorithm might tend to overfit on the majority class when initial training data contains only a few examples from the minority class.

Besides the different results for both algorithms, it is worth mentioning that variation in the labeling starting point did affect the results less severe when the labeled rate was 20%. Even when QBoost starts with just a few labels, a budget of 20% seems to be enough to request sufficient labels to achieve good performance. For the situation with a labeled rate of 20%, QBag could only benefit from a low starting point on the WDBC dataset. This result might be explained by the lower number of total observations of this dataset compared to the others. Even with a budget of 20% of the labels, a starting point of 75% for this dataset would be equivalent to an initial training set of only 68 observations.

Please note that the starting points with highest average F1-score were used in the experiment to compare all algorithms, described in the coming sections.

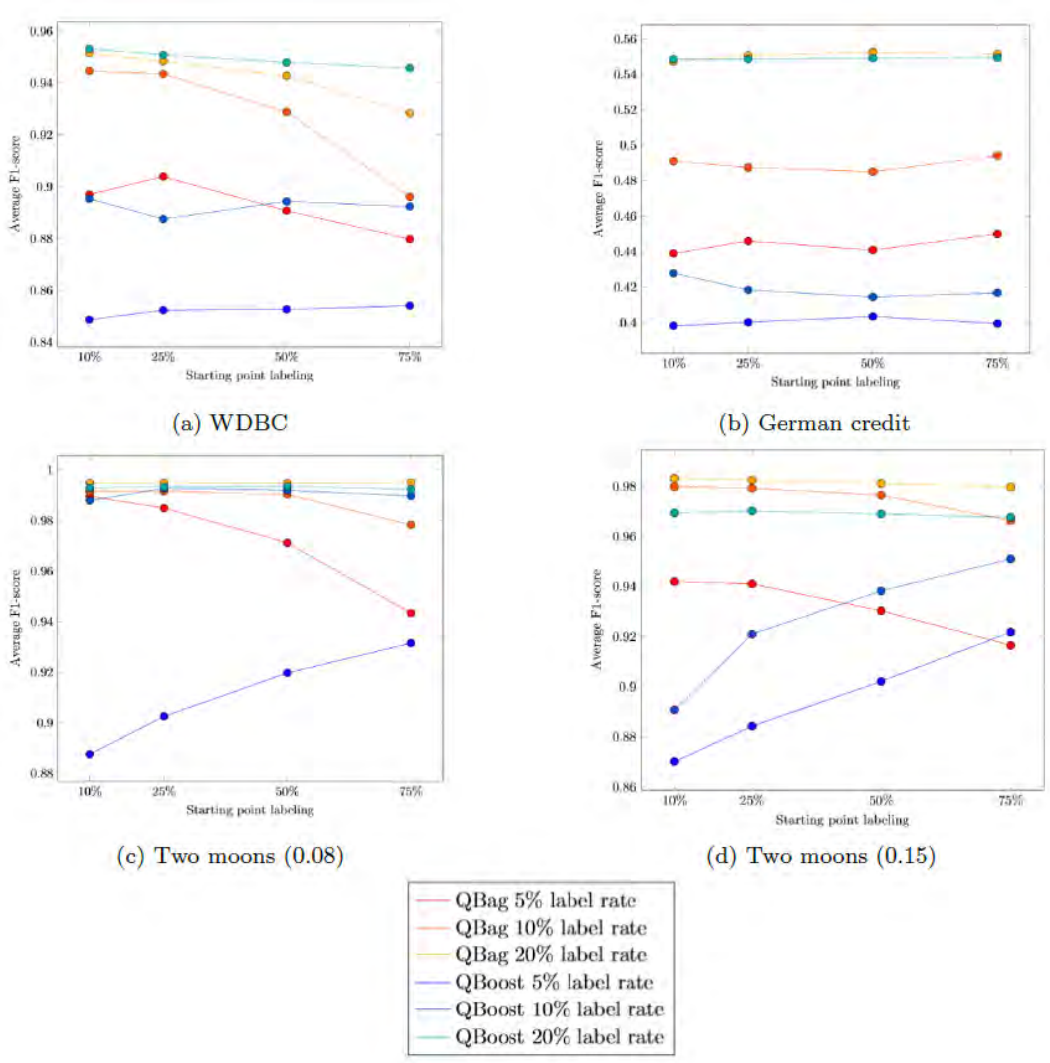
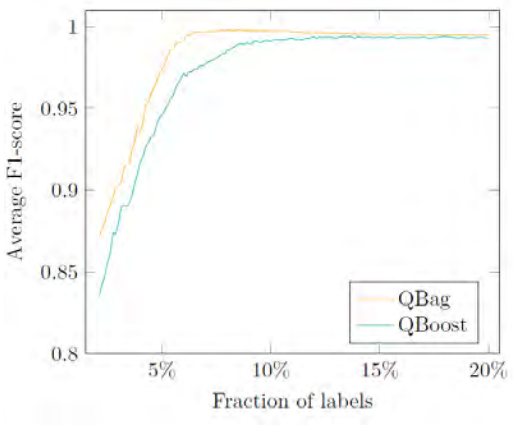


FIGURE 4.1: Each subplot illustrates the average F1-score of active learning algorithms QBag and QBoost using different labeling starting points.

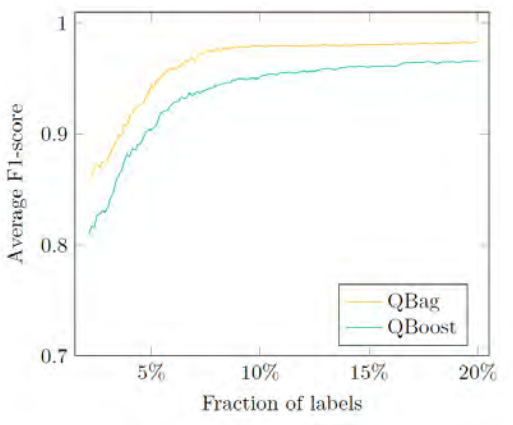
The lack of influence of the labeling starting point when the active learning algorithms have a budget of 20% raises the question whether they need the full budget to obtain good performances. Figure 4.2 shows the average performance of 100 independent runs, where the test score was measured after each time the models were re-trained

on the updated training data. This clearly shows that both algorithms converge to their maximum average F1-score far before using the final budget of 20% labels for both Two Moons datasets. Budget wise, this is a very interesting finding. Although Figure 4.1 shows that QBag and QBoost can also obtain good performances when starting with 75% of the budget of 20% labels, it would not be efficient to use this starting point. If such performances can be achieved earlier when initially using only 10% of the budget, it might save the oracle labeling work. This shows that it would be useful to investigate different stop-conditions than a maximum budget. Another stop-condition based on the training error improvement might avoid the waste of labeling budget.

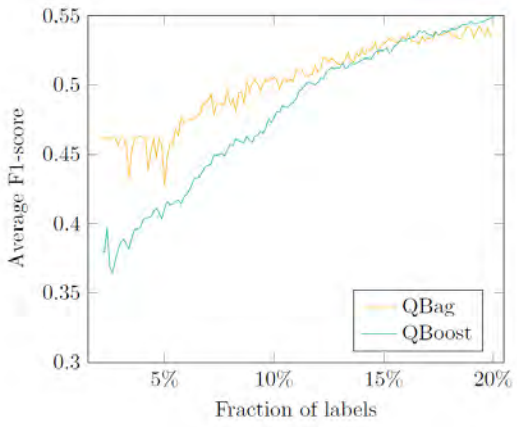
This early convergence was not found using the German Credit dataset. Due to the hard classification task, both algorithms need the full budget to achieve their best performance. This plot was not generated for WDBC since the performance of QBag *does* depend on the starting point when using a budget of 20%.



(a) Two Moons (0.08)



(b) Two Moons (0.15)



(c) German Credit

FIGURE 4.2: Convergence of QBag and QBoost using a total budget of 20% budget and an initial training set of 10% of this budget (i.e. 2% labels)

4.2 Wisconsin Diagnostic Breast Cancer

The average F1-scores after 500 runs of each algorithm using the WDBC dataset are shown in Figure 4.3. When using a labeled rate of 5% and 10%, QBag clearly performed best on average. Using only 10% of the data, the algorithm was already capable of outperforming a decision tree that was allowed to use the fully labeled dataset (the dotted line DT-S in Figure 4.3). The disappointing results of QBoost are consistent with the findings from the analysis described in Section 4.1. The algorithm failed to perform when the labeling budget was limited to only 5% or 10% of the data. When a budget of 20% was available though, which is still just a fraction of fully supervised learning, QBoost was able to outperform all other algorithms on this dataset. The poor result with a labeled rate of 5% might be explained again by the problem that weights of component classifiers and observations were not updated correctly using such sparse training data. This can also explain the poor performance of all algorithms that use boosting when having a labeled rate of 5%. In this situation, the bagging algorithms Random Forest, Co-Forest and QBag obviously outperformed the other algorithms.

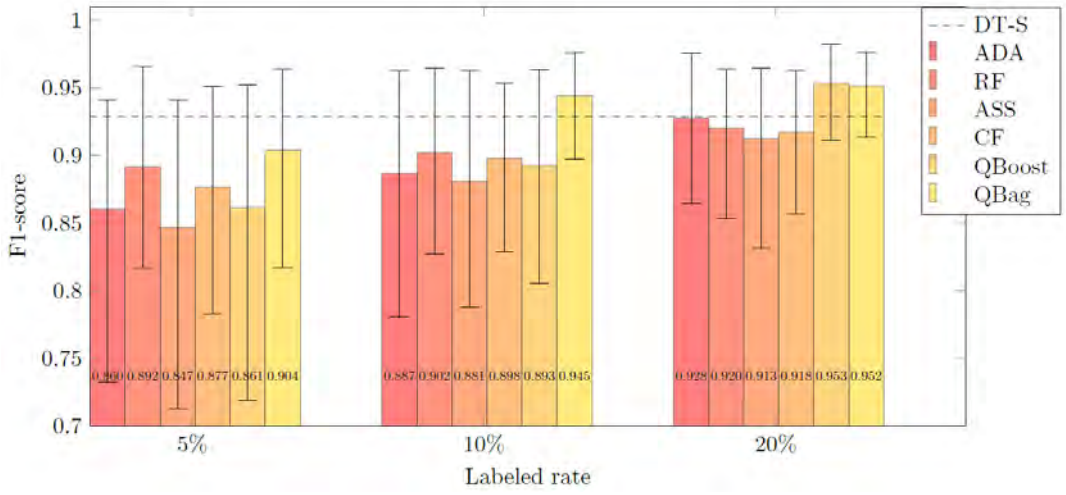


FIGURE 4.3: The average F1-scores for all algorithms with the 95% confidence intervals using the WDBC dataset and using different labeled rates.

It is somewhat surprising that the semi-supervised algorithms ASSEMBLE and Co-Forest were not able to outperform their component classifiers AdaBoost and Random Forest respectively. This rather unanticipated result may be due to this relatively easy classification dataset. Despite several conditions (Section 2.2.3.4.4) that Co-Forest uses to limit the number of selected predictions to add to the training set, it still selects too many predictions for easy classification tasks. This is caused by the initial value of parameter e, which is initially set to 0.5 before any predictions are made. When sampling the predictions to limit the number of selected observations, the algorithm uses the following sample size:

$$\frac{e_{i,t-1} \cdot W_{i,t-1}}{e_{i,t}}$$

where $e_{i,t}$ is the training error of component classifier i in iteration t and $W_{i,t-1}$ is the set of observation weights for component classifier i in iteration $t-1$. If the actual training error in this first iteration ($e_{i,1}$) is much smaller than 0.5 ($e_{i,0}$), the sample

size can become very large. This causes that many predictions are added to the training data, while it is likely that some predictions are noisy. A possible solution would be to assign more realistic initial values to $e_{i,0}$.

As explained in section 2.2.3.4.5, ASSEMBLE assigns pseudo-labels to each unlabeled observation based on their nearest (labeled) neighbors in the first iteration. When only a small amount of initial labels is available, these pseudo-lables might be incorrect. Although the algorithm assigns small initial weights to these observations, the noise might affect the performance of component classifier AdaBoost. These results reflect those of Breiman [14], who stated that AdaBoost is sensitive to noise in labels.

Although the F1-scores for the algorithms did differ on average, we needed to verify whether these differences are significant. First, the Kruskal-Wallis test was performed to test the hypothesis whether the median F1-scores of all algorithms are equal using 5%, 10% and 20% labeled rates. The resulting Chi-Squared statistics and the p-values can be found in Table 4.1. Since all p-values are much lower than significance level 0.05, we can reject the null-hypothesis and state that there are significant differences between the median F1-scores of the algorithms for all labeled rates. The extremely small p-values might be explained by the high number of performance samples for each algorithm. Due to the 500 runs the p-values are all equal to only 2.2e−16, which at least indicates that there is a substantial difference between the performances of the algorithms.

Labeled rate	**Chi-squared**	**p-value**
5%	365.99	2.2e−16
10%	1039.8	2.2e−16
20%	1487.4	2.2e−16

TABLE 4.1: The results of the Kruskal-Wallis test for the WDBC dataset. Each row shows the results of the test using a different labeled rate

To obtain more information about the significance of the performances, multiple comparison Dunn's test with False Discovery Rate correction was performed (as discussed in Section 3.4.4). The p-values of the test for each pair of algorithms are shown in Appendix C for labeled rates 5%, 10% and 20%. The null-hypothesis of pairs with a p-value lower than significance level 0.05 can be rejected. This indicates that there is a significant difference between the performance of a pair of algorithms. Since the p-values of QBag compared to all other algorithms are lower than 0.05 using labeled rates of 5% and 10%, we can conclude that QBag performed significantly better in these scenarios. When more budget is available (i.e. labeled rate of 20%), QBoost was able to significantly outperform all algorithms except QBag. There is no significant difference between both active learning algorithms in that situation.

4.3 German Credit

The average F1-scores after 500 runs of each algorithm using the German Credit dataset are shown in Figure 4.4. The most obvious finding to emerge from the results is that none of the algorithms was able to classify the minority class (e.g. 'bad' customers) accurately. Even the fully supervised decision tree obtained an average

F1-score below 0.6, which indicates that the german credit dataset is hard to classify. It is hard to verify the results on the German Credit dataset with other studies, as different performance metrics were used [22, 68].

As with the WDBC dataset, the bagging algorithms outperformed the boosting algorithms when having a limited labeled rate of 5% or 10%. Where Co-Forest was not able to outperform its supervised counterpart Random Forest using WDBC, it obtained higher F1-scores than Random Forest on this dataset. This difference might be explained by the challenging classification task of the german credit dataset. In contrast to its performance on WDBC dataset, Co-Forest is not too confident about its predictions for the German Credit dataset. This causes the training error of the algorithm to be closer to the initial value of 0.5 for the parameter e, which avoids Co-Forest to sample too many of its predictions for training. Since the algorithm added only a small set of pseudo-labels to its training set, it is less likely that it added noisy predictions to the training data that could hurt the performance of the model.

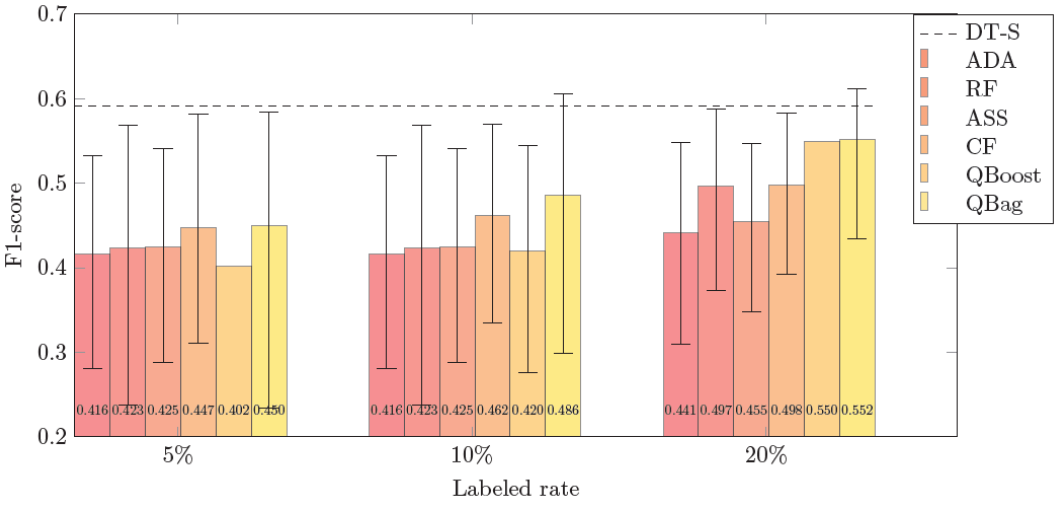


FIGURE 4.4: The average F1-scores with the 95% confidence intervals for all algorithms using the German Credit dataset and using different labeled rates.

These QBoost results reflect its results on the WDBC dataset. It was not able to compete with most other algorithms when using a budget of only 5% or 10%. However, when a budget of 20% is available QBoost seems better able to distinguish useful observations from other observations. In that situation, it outperformed all supervised- and semi-supervised algorithms and it was able to compete with the performance of QBag.

The Kruskal-Wallis test was performed to test whether there are significant differences between the results for all algorithms. Table 4.2 presents the resulting Chi-Squared statistics and the p-values obtained. Again all p-values are much lower than significance level 0.05, which is probably due to the high number of runs. The results of multiple comparison Dunn's test using False Discovery Rate corrections can be found in Appendix C. These results confirm that Co-Forest significantly outperformed all algorithms except QBag using labeled rates of 5% and 10%. The performances of Co-Forest and QBag were not significantly different for labeled rate 5%, but QBag did outperform Co-Forest when having more budget. Furthermore it is interesting to note that even ASSEMBLE.AdaBoost outperformed QBoost significantly using

a labeled rate of 5%, while both depend on AdaBoost. A possible explanation for this finding might be that QBoost selected outliers, since it was not confident about its predictions for these observations. In order to confirm this expectation, more extensive research on the german credit dataset is needed.

Labeled rate	**Chi-squared**	**p-value**
5%	332.95	2.2e−16
10%	512.02	2.2e−16
20%	1997.4	2.2e−16

TABLE 4.2: The results of the Kruskal-Wallis test for the German Credit dataset. Each row shows the results of the test using a different labeled rate

4.4 Two Moons (0.08)

The artificial data set Two Moons (0.08) was mainly used because of its simplicity. This enabled us to visualize the behavior of the active learning algorithms to understand what makes them perform well or not. Figure 4.5 shows the average F1-scores after 500 runs of each algorithm. Using a budget of only 5%, QBag clearly outperformed the other algorithms. Again, QBoost did not achieve comparable results as the other active learning algorithm using 5% of budget.

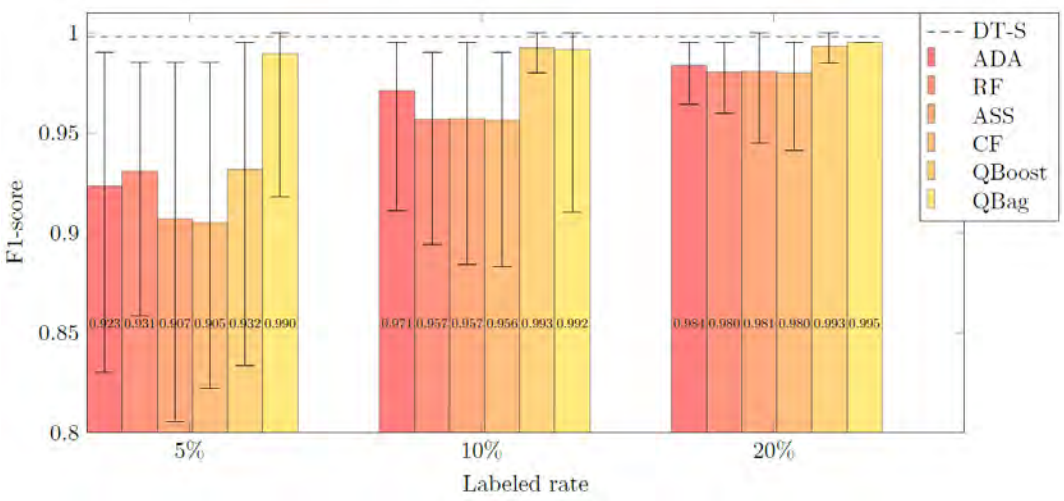
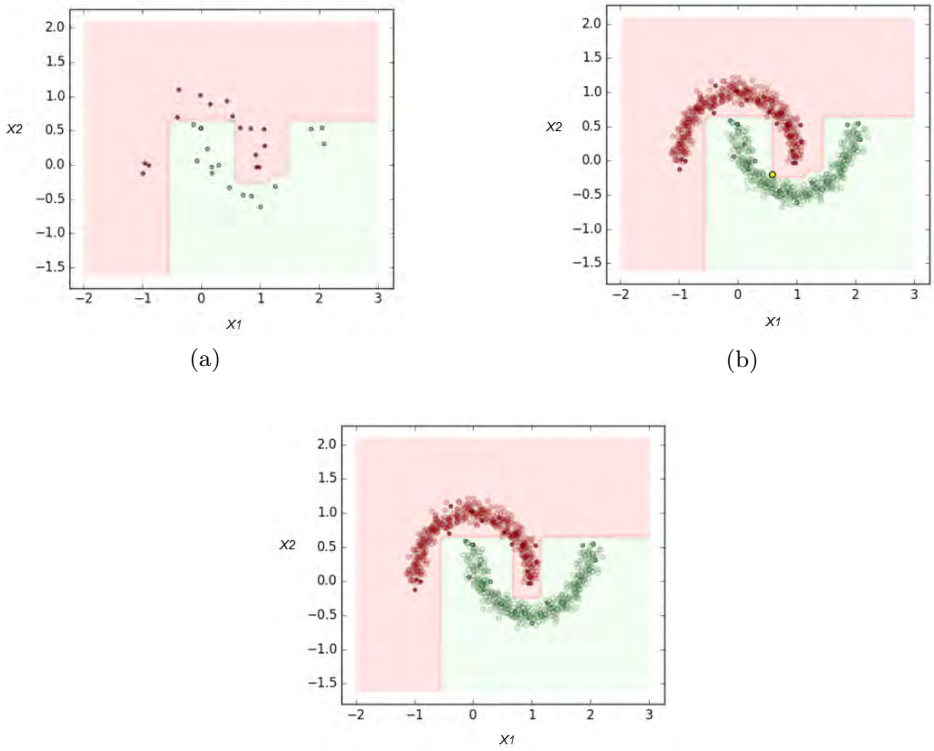


FIGURE 4.5: The average F1-scores with the 95% confidence intervals for all algorithms using the Two Moons (0.08) dataset and using different labeled rates.

Figure 4.6 illustrates one iteration of the QBag algorithm using this dataset. It shows the ability of QBag to query labels for observations close to the decision boundary of the current model, as most component classifiers disagree about their predictions. In this particular example, the decision boundary is shifted more to the red class, since it was provided the green label for an observation close to the decision boundary. Figure 4.7 illustrates three successive iterations of QBoost where three labels were requested from the same region, far from the decision boundary. This shows that QBoost was not able to explore different uncertain areas in this situation. Due to the

lack of diversity within the AdaBoost committee, the model did not have the ability to select useful observations. The requested labels did have only minimal effect on the shape of the decision boundary of the model.



(a) (b)

(c)

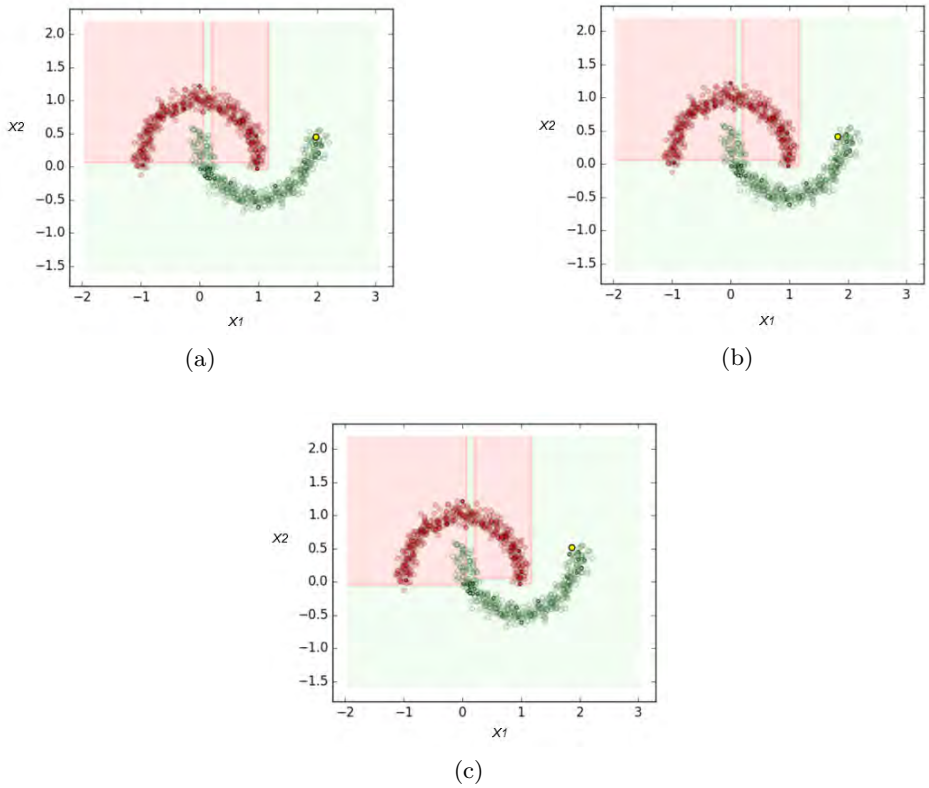
FIGURE 4.6: Visualization of one QBag iteration using Two Moons (0.08). The border between the green area and red area is the decision boundary of the model at that moment. *(a)* shows the labeled observations only, *(b)* shows all training data and the selected observation by QBag marked in yellow and *(c)* shows the adjusted decision boundary after retraining the model.

Furthermore, although the average F1-score of QBag did increase only slightly when raising the labeled rate from 10% to 20%, the size of its confidence interval decreased a lot. Due to the simplicity of this dataset, QBag can nearly always achieve very high performance.

Another result that stands out in Figure 4.5 is the poor performance of both semi-supervised algorithms. Just as for the WDBC dataset, this result may be due to the easy separable classes in this dataset. Co-Forest might have added too many predicted labels to its training set, due to its high performance in the first iteration of the algorithm. If ASSEMBLE.AdaBoost was not able to update its weights correctly, it might have focused too much on correctly classified observations instead of focusing on misclassifications.

Table 4.3 provides the results of the Kruskal-Wallis test for this dataset. As the p-values are much lower than significance level 0.05 again, we can reject the null-hypothesis and conclude that the median F1-scores for all algorithms significantly differ. Again, it should be noted that the p-values are probably extremely low because of the 500 runs. The results from Dunn's test using False Discovery Rate corrections

can be found in Appendix C. Although the average F1-scores in Figure 4.5 indicate that the difference between QBag and QBoost is minimal, the difference is significant. This can be caused by the extremely small confidence interval of QBag, while the performance of QBoost variates more.



(a) (b)

(c)

FIGURE 4.7: Visualization of three successive QBoost iterations. The border between the green area and red area is the decision boundary of the model at that moment.

Labeled rate	**Chi-squared**	**p-value**
5%	1149	2.2e−16
10%	1573.9	2.2e−16
20%	1424.6	2.2e−16

TABLE 4.3: The results of the Kruskal-Wallis test for the Two Moons (0.08) dataset. Each row shows the results of the test using a different labeled rate

4.5 Two Moons (0.15)

The Two Moons (0.15) dataset was used to study the performance of each algorithm when the dataset contains more outliers. Figure 4.8 provides the resulting average F1-scores after 500 runs for each algorithm including the 95% confidence intervals. One interesting finding is that the performance of the active learning algorithms greatly reduced compared to the performance on the Two Moons (0.08) dataset. This finding reflects those of Settles [85], who stated that heterogeneity models are sensitive to outlying observations (see Figure 2.7). As active learning algorithms aim to request labels for observations close to the decision boundary, they might select observations that are located close to the observations from the other class. Figure 4.9 illustrates one iteration of QBag using this dataset. Following the shapes of both classes, the current decision boundary should obviously be adjusted in some areas. QBag requested the label of the yellow-marked observation due to its uncertain prediction. Although this observation lies closer to the red class, it actually belongs to the green class (caused by the added noise). This causes the algorithm to not shift that specific part of the decision boundary more to the middle between both classes.

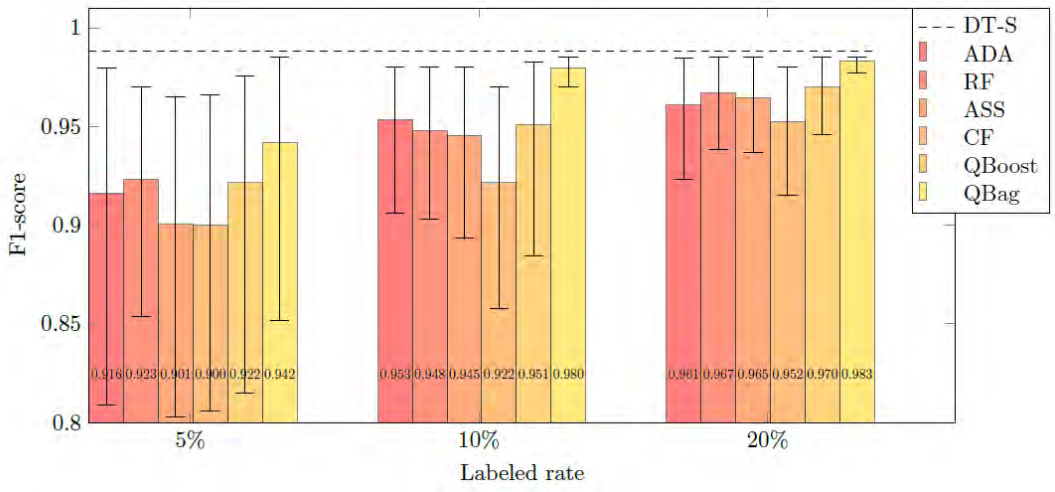


FIGURE 4.8: The average F1-scores with the 95% confidence intervals for all algorithms using the Two Moons (0.15) dataset and using different labeled rates.

Another result that stands out is the poor performance of Co-Forest on this dataset. Due to its high confidence about most predictions, the algorithm selected many of these predictions and added them to the training set as pseudo-labels. Due to the noisy structure of the data, it is very likely that it selected some of the outlying observations. This shows again that the major pitfall of Co-Forest is the selection of too many unlabeled observations for relatively easy classification tasks.

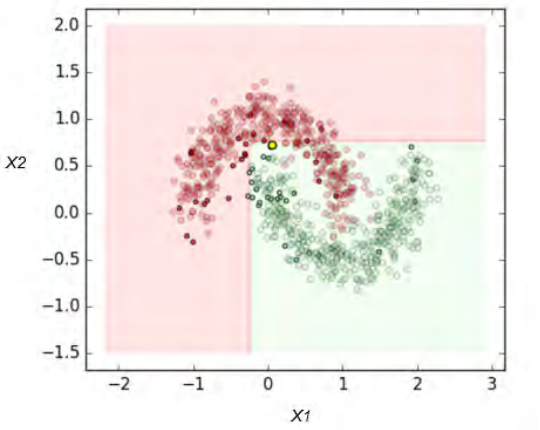


FIGURE 4.9: An illustration of the selection of an outlier by the QBag algorithm. The border between the green area and red area is the decision boundary of the model at that moment.

The p-values in Table 4.4 are all much lower than significance level 0.05. Despite the fact that the extremely low p-values might be caused by the high number of runs, that we can reject the null-hypothesis of the Kruskal-Wallis test and conclude that the median F1-scores of all algorithms significantly differ using this dataset. The results of Dunn's test with False Discovery Rate corrections show that despite the performance reduction of QBag, it still significantly outperformed all other algorithms on this dataset.

Labeled rate	**Chi-squared**	**p-value**
5%	380.62	2.2e−16
10%	1392	2.2e−16
20%	1424.6	2.2e−16

TABLE 4.4: The results of the Kruskal-Wallis test for the Two Moons (0.15) dataset. Each row shows the results of the test using a different labeled rate

4.6 Summary

Table 4.5 shows the algorithms that performed significantly best for each experiment following Dunn's test. This clearly shows that QBag outperformed all other algorithms in eight out of twelve situations. Three experiments resulted in a draw between QBag and another algorithm, while QBag was outperformed by QBoost in only one scenario. QBag has shown to be able to obtain consistent results when insufficient labeled observations are available.

Dataset	**5%**	**10%**	**20%**
WDBC	QBag	QBag	QBag and QBoost
German Credit	QBag and Co-Forest	QBag	QBag and QBoost
Two Moons (0.08)	QBag	QBoost	QBag
Two Moons (0.15)	QBag	QBag	QBag

TABLE 4.5: An overview of all results for each dataset and labeled rate. In cases with two algorithms, there was no significant difference found between them

Chapter 5

Conclusion & Future work

The main purpose of this study was to study the performance of semi-supervised algorithms and active learning algorithms in different situations. First, literature study was performed to investigate which techniques are available within both fields. Several promising studies were conducted on committee-based models, which use supervised ensemble techniques within semi-supervised learning and active learning. Despite that these techniques have proven to obtain strong generalization and to be relatively easy to apply, little effort has been made to compare committee-based models from both fields. Therefore, the research question of this thesis was:

"Which committee-based classification techniques within semi-supervised learning and active learning perform best using multiple data sets with various percentages of labeled observations?"

In order to answer this question, several experiments were performed. First, multiple algorithms were selected from literature based on 1) their performance in previous studies and 2) their applicability in real-life situations. We selected algorithms that use well-known techniques bagging and boosting to generate a committee. Semi-supervised algorithms ASSEMBLE (boosting) and Co-Forest (bagging), active learning algorithms QBoost (boosting) and QBag (bagging) and supervised ensemble algorithms AdaBoost (boosting) and Random Forest (bagging) were implemented. Since the decision tree algorithm C4.5 was used as base learner for all these algorithms, this algorithm was used as benchmark using all labeled data.

Second, it was studied whether the performance of the active learning algorithms depends on the size of the initial training set. Using budgets of 5%, 10% and 20%, the algorithms were provided 10%, 25%, 50% and 75% of their budget as starting points for labeling. The results of this analysis show that QBoost obtains best results when initially using a relatively large part of its budget, while QBag is able to achieve its highest average F1-scores when it is initially trained on just a few labeled observations. The findings for the German Credit dataset deviate from this, which might be explained by the fact that it is a more challenging classification dataset.

Interestingly, the size of the initial training set did not influence the results much when a budget of 20% was available. Further analysis showed that when initially only 10% of this budget was provided, both QBag and QBoost converged to their best performance much earlier using both Two Moons datasets. For the more challenging German Credit classification, the full budget was required to obtain best results. This shows that it would be useful to investigate different stop-conditions besides terminating when reaching the provided labeling budget. When considering

the improvement of the model after each query, the algorithms might be terminated when the improvement stagnates.

Next, all algorithms were performed 500 times using all datasets and labeled rates 5%, 10% and 20%. The results of these experiments have shown that committee-based algorithms within semi-supervised learning and active learning are generally able to improve the performance of supervised algorithms AdaBoost and Random Forest by incorporating unlabeled data. The most obvious finding is that QBag significantly outperformed all other algorithms in eight out of twelve situations. In three of the remaining four situations there was no significant difference between QBag and another algorithm. In only one situation QBoost performed significantly better than QBag.

The second major finding is the poor performance of the boosting algorithms compared to the bagging algorithms. For nearly all datasets with a labeled rate of 5%, the bagging algorithms outperformed their boosting counterparts within their fields (e.g. QBag compared to QBoost). This might be explained by the inability of the component classifier AdaBoost to update its classifier weights and observations weights properly when training data is sparse. A further study could assess this problem by trying more sophisticated boosting component classifiers like XGBoost. Another option is to investigate the possibilities to improve the ability of AdaBoost to update its classifier- and observation weights when its component classifiers obtain perfect classification on sparse data.

Furthermore, this study has confirmed that active learning algorithms are prone to outliers. Besides the performance reduction when adding additional noise to the Two Moons dataset, visualizations of the learning process of QBag and QBoost have shown that both algorithms might focus too much on outlying observations. This would be a fruitful area for further work, as several questions still remain to be answered. It would be interesting study the behavior of the algorithms on datasets with different types of outliers (e.g. local or global outliers) and whether proper data preparation might solve these issues.

The inconsistent results of the Co-Forest algorithm was also an interesting finding. Using datasets WDBC and German Credit, the algorithm was able to outperform most algorithms using budgets of 5% and 10%. However, when target classes were relatively easily separable like with the Two Moons datasets, it suffered from adding too many pseudo-labels to its training set. This study should be repeated using more datasets of varying classification difficulty to verify whether Co-Forest only fails to perform on easy synthetic datasets, or that it is a problem of the algorithm for more datasets.

The final finding is the lack of performance of ASSEMBLE. The developers of the algorithm only tested the algorithm for situations where more budget was available than in this research. Especially using only 5% or 10% of labeled data, it often did not obtain better results than its supervised component classifier AdaBoost. This might be explained by the use of pseudo-labels, which are likely to be noisy when they are provided using sparse training data. Since AdaBoost is prone to labeling noise, the semi-supervised algorithm might suffer. These findings provide insights for future research as it would be interesting to study the use of different component

classifiers in the ASSEMBLE framework and to investigate more accurate techniques to assign pseudo-labels to unlabeled data in the first iteration.

One possible limitation of this study is the possible slight advantage of active learning in the experiments. These algorithms were able to select their labeled data for training, while the supervised- and semi-supervised algorithms were dependent on the randomly selected initial training data. However, the disadvantage of the semi-supervised algorithms may be limited since 1) they use additional unlabeled data besides the labeled training set and 2) the experiments were performed 500 times to limit the effect of the random initial training data.

Notwithstanding this comment, this thesis has provided deeper insight into the performance of committee-based algorithms using insufficient labeled observations. These findings can be of interest to realistic situations with limited available labeled data. The studied algorithms have proven to be suitable for business use cases with a limited labeling budget in terms of performance and applicability.

Appendix A

Dataset properties

Features	**Values**
Patient ID	ID number
Radius	Numerical
Texture	Numerical
Perimeter	Numerical
Area	Numerical
Smoothness	Numerical
Compactness	Numerical
Concavity	Numerical
Concave points	Numerical
Symmetry	Numerical
Fractal dimension	Numerical
Class	1 = Malignant, 0 = Benign

Table A.1: Features WDBC dataset

Features	**Values**
Status existing checking account	1 - 4
Duration in month	Numerical
Credit history	1 - 5
Purpose	1 - 11
Credit amount	Numerical
Savings account/bonds	1 - 5
Present employment since	1 - 5
Installment rate in percentage of disposable income	Numerical
Personal status and sex	1 - 5
Other debtors / guarantors	1 - 3
Present residence since	Numerical
Property	1 - 4
Age in years	Numerical
Other installment plans	1 - 3
Housing	1 - 3
Number of existing credits at this bank	Numerical
Job	1 - 4
Number of people being liable to provide maintenance for	Numerical
Telephone	1 - 2
Foreign worker	1 - 2
Class	1 = Good, 2 = Bad

TABLE A.2: Features German Credit dataset

Appendix B

Hyperparameters

As discussed in Section 3.4.1, the value for hyperparameter *max depth* was set to 2 for boosting algorithms ASSEMBLE, QBoost and AdaBoost and to 'None' for bagging algorithms Co-Forest and QBag. The value for hyperparameter *min samples leaf* was set to 5 for the boosting algorithms, while it was set to 1 for the bagging algorithms.

Dataset	**Hyperparameters**	**5%**	**10%**	**20%**
WDBC	Max. leaf nodes	20	10	5
	Criterion	Entropy	Entropy	Entropy
	Min. samples split	10	2	2
	Class weight	None	None	None
German Credit	Max. leaf nodes	5	5	None
	Criterion	Gini	Entropy	Entropy
	Min. samples split	10	2	20
	Class weight	Balanced	Balanced	Balanced
Two Moons (0.08)	Max. leaf nodes	20	20	20
	Criterion	Entropy	Entropy	Entropy
	Min. samples split	2	2	2
	Class weight	None	None	None
Two Moons (0.15)	Max. leaf nodes	None	None	None
	Criterion	Entropy	Entropy	Entropy
	Min. samples split	2	2	2
	Class weight	None	None	None

TABLE B.1: Hyperparameter values used for the C4.5 algorithm

Dataset	**Hyperparameters**	**5%**	**10%**	**20%**
WDBC	No. of estimators	50	20	50
	Learning rate	1.0	1.0	1.0
German Credit	No. of estimators	20	20	20
	Learning rate	1.0	1.0	1.0
Two Moons (0.08)	No. of estimators	20	50	50
	Learning rate	0.1	1.0	1.0
Two Moons (0.15)	No. of estimators	50	50	20
	Learning rate	0.1	0.1	1.0

TABLE B.2: Hyperparameter values used for the AdaBoost algorithm

The hyperparameter *max features* could become $\sqrt{numberoffeatures}$ or $log_2(numberoffeatures)$ using grid search. These values are represented as *sqrt* and log_2 in Table B.3 respectively.

Dataset	**Hyperparameters**	**5%**	**10%**	**20%**
WDBC	Min. samples leaf	5	10	1
	No. of estimators	50	50	50
	Min. samples split	10	10	2
	Criterion	Gini	Entropy	Entropy
	Max. features	log_2	*sqrt*	*sqrt*
	Max. depth	2	5	5
	Class weight	None	None	None
German Credit	Min. samples leaf	5	5	1
	No. of estimators	20	50	50
	Min. samples split	10	2	2
	Criterion	Gini	Gini	Gini
	Max. features	*sqrt*	log_2	*sqrt*
	Max. depth	2	5	2
	Class weight	Balanced	Balanced	Balanced
Two Moons (0.08)	Min. samples leaf	1	1	1
	No. of estimators	50	50	50
	Min. samples split	2	2	2
	Criterion	Entropy	Entropy	Entropy
	Max. features	*sqrt*	*sqrt*	log_2
	Max. depth	5	5	None
	Class weight	None	None	None
Two Moons (0.15)	Min. samples leaf	1	1	1
	No. of estimators	50	50	50
	Min. samples split	2	2	2
	Criterion	Entropy	Entropy	Entropy
	Max. features	*sqrt*	*sqrt*	*sqrt*
	Max. depth	None	None	None
	Class weight	None	None	None

TABLE B.3: Hyperparameter values used for the Random Forest algorithm

Dataset	**Hyperparameters**	**5%**	**10%**	**20%**
WDBC	Nearest Neighbor method	KDTree	KNN	KNN
	Leaf size	2	5	5
	No. neighbors	1	1	3
	Initial labeled weights	0.95	0.95	0.95
German Credit	Nearest Neighbor method	KNN	KDTree	KDTree
	Leaf size	5	2	5
	No. neighbors	1	3	1
	Initial labeled weights	0.95	0.8	0.95
Two Moons (0.08)	Nearest Neighbor method	KNN	KDTree	KDTree
	Leaf size	2	2	5
	No. neighbors	3	1	3
	Initial labeled weights	0.95	0.95	0.95
Two Moons (0.15)	Nearest Neighbor method	KNN	KDTree	KDTree
	Leaf size	5	5	5
	No. neighbors	3	3	1
	Initial labeled weights	0.95	0.95	0.95

Table B.4: Hyperparameter values used for the ASSEMBLE algorithm

Dataset	**Hyperparameters**	**5%**	**10%**	**20%**
WDBC	No. of estimators	50	50	50
	Confidence threshold	0.8	0.85	0.85
German Credit	No. of estimators	20	20	20
	Confidence threshold	0.85	0.85	0.85
Two Moons (0.08)	No. of estimators	50	50	50
	Confidence threshold	0.7	0.7	0.7
Two Moons (0.15)	No. of estimators	50	50	50
	Confidence threshold	0.7	0.7	0.75

Table B.5: Hyperparameter values used for the Co-Forest algorithm

Dataset	**Hyperparameters**	**5%**	**10%**	**20%**
WDBC	Disagreement measure	Vote Entropy	JS Divergence	Vote Entropy
German Credit	Disagreement measure	Vote Entropy	Vote Entropy	Vote Entropy
Two Moons (0.08)	Disagreement measure	Vote Entropy	Vote Entropy	Vote Entropy
Two Moons (0.15)	Disagreement measure	Vote Entropy	Vote Entropy	Vote Entropy

Table B.6: Hyperparameter values used for the QBoost algorithm.

Dataset	**Hyperparameters**	**5%**	**10%**	**20%**
WDBC	Disagreement measure	Vote Entropy	JS Divergence	Vote Entropy
	No. of estimators	50	50	50
German Credit	Disagreement measure	Vote Entropy	Vote Entropy	Vote Entropy
	No. of estimators	50	50	50
Two Moons (0.08)	Disagreement measure	JS Divergence	Vote Entropy	Vote Entropy
	No. of estimators	20	50	50
Two Moons (0.15)	Disagreement measure	Vote Entropy	JS Divergence	JS Divergence
	No. of estimators	50	50	50

Table B.7: Hyperparameter values used for the QBag algorithm

Appendix C

Dunn's test

C.1 Wisconsin Diagnostic Breast Cancer

	AdaBoost	Assemble	Co-Forest	QBag	QBoost
Assemble	0.00905				
Co-Forest	9.1e−06	1.5e−12			
QBag	2e−16	2e−16	2e−16		
QBoost	0.38472	0.00053	0.00032	2e−16	
Random Forest	2e−16	2e−16	6e−06	4.6e−05	2.5e−16

TABLE C.1: P-values resulting from Dunn's test using False Discovery Rate corrections for the WDBC dataset using a labeled rate of 5%.

	AdaBoost	Assemble	Co-Forest	QBag	QBoost
Assemble	0.04090				
Co-Forest	0.00252	3e−07			
QBag	2e−16	2e−16	2e−16		
QBoost	0.07405	0.00011	0.20366	2e−16	
Random Forest	1.8e−06	5.7e−12	0.08298	2e−16	0.00305

TABLE C.2: P-values resulting from Dunn's test using False Discovery Rate corrections for the WDBC dataset using a labeled rate of 10%.

	AdaBoost	Assemble	Co-Forest	QBag	QBoost
Assemble	4.5e−11				
Co-Forest	1.9e−08	0.350			
QBag	2e−16	2e−16	2e−16		
QBoost	2e−16	2e−16	2e−16	0.381	
Random Forest	3.9e−05	0.015	0.145	2e−16	2e−16

TABLE C.3: P-values resulting from Dunn's test using False Discovery Rate corrections for the WDBC dataset using a labeled rate of 20%.

C.2 German Credit

	AdaBoost	**Assemble**	**Co-Forest**	**QBag**	**QBoost**
Assemble	0.07348				
Co-Forest	9.9e−10	2e−05			
QBag	1.7e−13	3.3e−08	0.21587		
QBoost	0.11582	0.00065	1.4e−14	2e−16	
Random Forest	0.02270	0.61809	0.00015	4.9e−07	0.10001

TABLE C.4: P-values resulting from Dunn's test using False Discovery Rate corrections for the German Credit dataset using a labeled rate of 5%.

	AdaBoost	**Assemble**	**Co-Forest**	**QBag**	**QBoost**
Assemble	0.075				
Co-Forest	2e−16	1.9e−15			
QBag	2e−16	2e−16	4.5e−07		
QBoost	0.407	0.367	2e−16	2e−16	
Random Forest	0.012	0.464	5.3e−13	2e−16	0.102

TABLE C.5: P-values resulting from Dunn's test using False Discovery Rate corrections for the German Credit dataset using a labeled rate of 10%.

	AdaBoost	**Assemble**	**Co-Forest**	**QBag**	**QBoost**
Assemble	0.076				
Co-Forest	2e−16	2e−16			
QBag	2e−16	2e−16	2e−16		
QBoost	2e−16	2e−16	2e−16	0.073	
Random Forest	2e−16	2e−16	0.648	2e−16	2e−16

TABLE C.6: P-values resulting from Dunn's test using False Discovery Rate corrections for the German Credit dataset using a labeled rate of 20%.

C.3 Two Moons (0.08)

	AdaBoost	**Assemble**	**Co-Forest**	**QBag**	**QBoost**
Assemble	8e−07				
Co-Forest	1.2e−09	0.2687			
QBag	2e−16	2e−16	2e−16		
QBoost	0.0094	3.2e−14	2e−16	2e−16	
Random Forest	0.0476	3.3e−12	5.8e−16	4.9e−07	0.5265

TABLE C.7: P-values resulting from Dunn's test using False Discovery Rate corrections for the Two Moons (0.08) dataset using a labeled rate of 5%.

	AdaBoost	**Assemble**	**Co-Forest**	**QBag**	**QBoost**
Assemble	4e−12				
Co-Forest	5.9e−09	0.280			
QBag	2e−16	2e−16	2e−16		
QBoost	2e−16	2e−16	2e−16	0.012	
Random Forest	2.5e−15	0.327	0.041	2e−16	2e−16

TABLE C.8: P-values resulting from Dunn's test using False Discovery Rate corrections for the Two Moons (0.08) dataset using a labeled rate of 10%.

	AdaBoost	**Assemble**	**Co-Forest**	**QBag**	**QBoost**
Assemble	0.65088				
Co-Forest	0.24553	0.45809			
QBag	2e−16	2e−16	2e−16		
QBoost	2e−16	2e−16	2e−16	5.3e−08	
Random Forest	2e−05	0.00013	0.00231	2e−16	2e−16

TABLE C.9: P-values resulting from Dunn's test using False Discovery Rate corrections for the Two Moons (0.08) dataset using a labeled rate of 20%.

C.4 Two Moons (0.15)

	AdaBoost	**Assemble**	**Co-Forest**	**QBag**	**QBoost**
Assemble	2.9e−10				
Co-Forest	1e−11	0.610			
QBag	2e−16	2e−16	2e−16		
QBoost	0.058	2e−16	2e−16	5.5e−15	
Random Forest	0.324	1.8e−13	4.2e−15	2e−16	0.388

TABLE C.10: P-values resulting from Dunn's test using False Discovery Rate corrections for the Two Moons (0.15) dataset using a labeled rate of 5%.

	AdaBoost	**Assemble**	**Co-Forest**	**QBag**	**QBoost**
Assemble	6.6e−07				
Co-Forest	2e−16	2e−16			
QBag	2e−16	2e−16	2e−16		
QBoost	0.37	4.4e−05	2e−16	2e−16	
Random Forest	8.1e−05	0.3109	2e−16	2e−16	0.0023

TABLE C.11: P-values resulting from Dunn's test using False Discovery Rate corrections for the Two Moons (0.15) dataset using a labeled rate of 10%.

	AdaBoost	**Assemble**	**Co-Forest**	**QBag**	**QBoost**
Assemble	0.01307				
Co-Forest	5.5e−10	2e−16			
QBag	2e−16	2e−16	2e−16		
QBoost	7.2e−16	2.4e−08	2e−16	2e−16	
Random Forest	1.7e−09	0.00042	2e−16	2e−16	0.03950

TABLE C.12: P-values resulting from Dunn's test using False Discovery Rate corrections for the Two Moons (0.15) dataset using a labeled rate of 20%.

Bibliography

[1] M.F. Abdel Hady. "Semi-supervised learning with committees: exploiting unlabeled data using ensemble learning algorithms". In: (2016). DOI: 10.18725/oparu-1750. URL: https://oparu.uni-ulm.de/xmlui/handle/123456789/1777.

[2] N. Abe and H. Mamitsuka. "Query Learning Strategies Using Boosting and Bagging". In: *Proceedings of the Fifteenth International Conference on Machine Learning*. ICML '98. Morgan Kaufmann Publishers Inc., 1998, pp. 1–9. URL: http://dl.acm.org/citation.cfm?id=645527.657478.

[3] C.C. Aggarwal et al. "Active Learning: A Survey". In: *Data Classification: Algorithms and Applications*. 2014.

[4] A. Agrawala. "Learning with a probabilistic teacher". In: *Information Theory, IEEE Transactions on* 16.4 (July 1970), pp. 373–379.

[5] D. Angluin. "Queries and Concept Learning". In: *Machine Learning* 2.4 (Apr. 1988), pp. 319–342. ISSN: 0885-6125. DOI: 10.1023/A:1022821128753. URL: https://doi.org/10.1023/A:1022821128753.

[6] D. Angluin and P. Laird. "Learning From Noisy Examples". In: *Machine Learning* 2.4 (Apr. 1988), pp. 343–370. ISSN: 0885-6125. DOI: 10.1023/A:1022873112823. URL: https://doi.org/10.1023/A:1022873112823.

[7] S. Argamon-Engelson and I. Dagan. "Committee-Based Sample Selection for Probabilistic Classifiers". In: *CoRR* abs/1106.0220 (2011). URL: http://arxiv.org/abs/1106.0220.

[8] E. Bauer and R. Kohavi. "An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants". In: *Machine Learning* 36.1 (1999), pp. 105–139. ISSN: 1573-0565. DOI: 10.1023/A:1007515423169. URL: https://doi.org/10.1023/A:1007515423169.

[9] Y. Benjamini and Y. Hochberg. "Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 57.1 (1995), pp. 289–300. ISSN: 00359246. DOI: 10.2307/2346101. URL: http://dx.doi.org/10.2307/2346101.

[10] K.P. Bennett and A. Demiriz. "Semi-Supervised Support Vector Machines". In: *Advances in Neural Information Processing Systems 11*. Ed. by M. J. Kearns, S. A. Solla, and D. A. Cohn. MIT Press, 1999, pp. 368–374. URL: http://papers.nips.cc/paper/1582-semi-supervised-support-vector-machines.pdf.

[11] K.P. Bennett, A. Demiriz, and R. Maclin. "Exploiting Unlabeled Data in Ensemble Methods". In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD 2002. ACM, 2002, pp. 289–296. ISBN: 1-58113-567-X. URL: http://doi.acm.org/10.1145/775047.775090.

[12] A. Blum and T. Mitchell. "Combining Labeled and Unlabeled Data with Co-training". In: *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*. COLT' 98. ACM, 1998, pp. 92–100. DOI: 10.1145/279943.279962. URL: http://doi.acm.org/10.1145/279943.279962.

[13] D. Bouneffouf. "Exponentiated Gradient Exploration for Active Learning". In: *CoRR* abs/1408.2196 (2014). arXiv: 1408.2196. URL: http://arxiv.org/abs/1408.2196.

[14] L. Breiman. "Arcing classifier (with discussion and a rejoinder by the author)". In: *Ann. Statist.* 26.3 (June 1998), pp. 801–849. DOI: 10.1214/aos/1024691079. URL: https://doi.org/10.1214/aos/1024691079.

[15] L. Breiman. "Arcing classifier (with discussion and a rejoinder by the author)". In: *The Annals of Statistics* 26.3 (June 1998), pp. 801–849. ISSN: 0090-5364, 2168-8966. DOI: 10.1214/aos/1024691079. URL: http://projecteuclid.org/euclid.aos/1024691079.

[16] L. Breiman. "Bagging Predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140. ISSN: 1573-0565. DOI: 10.1023/A:1018054314350. URL: https://doi.org/10.1023/A:1018054314350.

[17] L. Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[18] W. Cai, Y. Zhang, and J. Zhou. "Maximizing Expected Model Change for Active Learning in Regression". In: *2013 IEEE 13th International Conference on Data Mining* (2013), pp. 51–60.

[19] O. Chapelle, B. Schlkopf, and A. Zien. *Semi-Supervised Learning*. 1st. The MIT Press, 2010. ISBN: 0262514125, 9780262514125.

[20] O. Chapelle, V. Sindhwani, and S.S. Keerthi. "Branch and Bound for Semi-Supervised Support Vector Machines". In: *Advances in Neural Information Processing Systems 19*. Ed. by B. Schölkopf, J. C. Platt, and T. Hoffman. MIT Press, 2007, pp. 217–224. URL: http://papers.nips.cc/paper/3135-branch-and-bound-for-semi-supervised-support-vector-machines.pdf.

[21] O. Chapelle and A. Zien. "Semi-Supervised Classification by Low Density Separation". In: *AISTATS 2005*. Max-Planck-Gesellschaft. Jan. 2005, pp. 57–64.

[22] K. Chen and S. Wang. "Semi-Supervised Learning via Regularized Boosting Working on Multiple Semi-Supervised Assumptions". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 33 (2010), pp. 129–143. ISSN: 0162-8828.

[23] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. ACM, 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

[24] D. Cohn, L. Atlas, and R. Ladner. "Improving Generalization with Active Learning". In: *Machine Learning* 15.2 (May 1994), pp. 201–221. ISSN: 0885-6125. DOI: 10.1023/A:1022673506211. URL: https://doi.org/10.1023/A:1022673506211.

[25] D.A. Cohn, Z. Ghahramani, and M.I. Jordan. "Active Learning with Statistical Models". In: *J. Artif. Int. Res.* 4.1 (Mar. 1996), pp. 129–145. ISSN: 1076-9757. URL: http://dl.acm.org/citation.cfm?id=1622737.1622744.

[26] Cong et al. "Semi-supervised Text Classification Using Partitioned EM". In: *Database Systems for Advanced Applications: 9th International Conference, DASFAA 2004, Jeju Island, Korea, March 17-19, 2003. Proceedings*, ed. by Y. Lee et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 482–493. ISBN: 978-3-540-24571-1. DOI: 10.1007/978-3-540-24571-1_45. URL: https://doi.org/10.1007/978-3-540-24571-1_45.

[27] C. Cortes and V. Vapnik. "Support-Vector Networks". In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297. ISSN: 0885-6125. DOI: 10.1023/A:1022627411411. URL: https://doi.org/10.1023/A:1022627411411.

[28] T. Cover and P. Hart. "Nearest Neighbor Pattern Classification". In: *IEEE Trans. Inf. Theor.* 13.1 (Sept. 2006), pp. 21–27. ISSN: 0018-9448. DOI: 10.1109/TIT.1967.1053964. URL: http://dx.doi.org/10.1109/TIT.1967.1053964.

[29] F. Gagliardi Cozman and I. Cohen. "Unlabeled Data Can Degrade Classification Performance of Generative Classifiers". In: *FLAIRS Conference*. 2002.

[30] F. D'alché-buc, Y. Grandvalet, and C. Ambroise. "Semi-supervised MarginBoost". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. Vancouver, British Columbia, Canada: MIT Press, 2001, pp. 553–560. URL: http://dl.acm.org/citation.cfm?id=2980539.2980612.

[31] I. Davidson. "An Ensemble Technique for Stable Learners with Performance Bounds". In: AAAI'04 (2004), pp. 330–335. URL: http://dl.acm.org/citation.cfm?id=1597148.1597203.

[32] A.P. Dempster, N.M. Laird, and D.B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B* 39.1 (1977), pp. 1–38.

[33] T.G. Dietterich. "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization". In: *Machine Learning* 40.2 (2000), pp. 139–157. DOI: 10.1023/A:1007607513941. URL: https://doi.org/10.1023/A:1007607513941.

[34] T.G. Dietterich. *Ensemble Methods in Machine Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–15. DOI: 10.1007/3-540-45014-9_1. URL: https://doi.org/10.1007/3-540-45014-9_1.

[35] P. Donmez, J.G. Carbonell, and J. Schneider. "Efficiently Learning the Accuracy of Labeling Sources for Selective Sampling". In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '09. Paris, France: ACM, 2009, pp. 259–268. ISBN: 978-1-60558-495-9. DOI: 10.1145/1557019.1557053. URL: http://doi.acm.org/10.1145/1557019.1557053.

[36] O.J. Dunn. "Estimation of the Means of Dependent Variables". In: *The Annals of Mathematical Statistics* 29.4 (1958), pp. 1095–1111. ISSN: 00034851. URL: http://www.jstor.org/stable/2236948.

[37] O.J. Dunn. "Multiple Comparisons Using Rank Sums". In: *Technometrics* 6.3 (1964), pp. 241–252. DOI: 10.1080/00401706.1964.10490181. URL: https://www.tandfonline.com/doi/abs/10.1080/00401706.1964.10490181.

[38] B. Efron and R. Tibshirani. "Bootstrap Methods for Standard Errors, Confidence Intervals, and Other Measures of Statistical Accuracy". In: *Statist. Sci.* 1.1 (Feb. 1986), pp. 54–75. DOI: 10.1214/ss/1177013815. URL: https://doi.org/10.1214/ss/1177013815.

[39] Felix F. and Irena K. "Co-training using RBF Nets and Different Feature Splits". In: *The 2006 IEEE International Joint Conference on Neural Network Proceedings* (2006), pp. 1878–1885.

[40] S. Fortmann-Roe. *Understanding the Bias-Variance Tradeoff*. 2012. URL: http://scott.fortmann-roe.com/docs/BiasVariance.html.

[41] S. Fralick. "Learning to recognize patterns without a teacher". In: *Information Theory, IEEE Transactions on* 13.1 (Jan. 1967), pp. 57–64.

[42] Y. Freund and R.E. Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119 –139. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1997.1504. URL: http://www.sciencedirect.com/science/article/pii/S002200009791504X.

[43] Y. Freund et al. "Selective Sampling Using the Query by Committee Algorithm". In: *Mach. Learn.* 28.2-3 (Sept. 1997), pp. 133–168. ISSN: 0885-6125. DOI: 10.1023/A:1007330508534. URL: https://doi.org/10.1023/A:1007330508534.

[44] J.H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. URL: http://www.jstor.org/stable/2699986.

[45] A. Fujii et al. "Selective Sampling for Example-based Word Sense Disambiguation". In: *Comput. Linguist.* 24.4 (Dec. 1998), pp. 573–597. ISSN: 0891-2017. URL: http://dl.acm.org/citation.cfm?id=972764.972766.

[46] S. Geman, E. Bienenstock, and R. Doursat. "Neural Networks and the Bias/Variance Dilemma". In: *Neural Comput.* 4.1 (Jan. 1992), pp. 1–58. ISSN: 0899-7667. DOI: 10.1162/neco.1992.4.1.1. URL: http://dx.doi.org/10.1162/neco.1992.4.1.1.

[47] S.A. Goldman and Y. Zhou. "Enhancing Supervised Learning with Unlabeled Data". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. Morgan Kaufmann Publishers Inc., 2000, pp. 327–334. ISBN: 1-55860-707-2. URL: http://dl.acm.org/citation.cfm?id=645529.658273.

[48] A.L. Gorin, D.Z. Hakkani-Tur, and G. Riccardi. *Method of active learning for automatic speech recognition*. Dec. 2006. URL: https://www.google.com/patents/US7149687.

[49] H. Hofmann. *UCI Machine Learning Repository*. 1994. URL: https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data).

[50] C. Hsu, C. Chang, and C. Lin. *A Practical Guide to Support Vector Classification*. Tech. rep. Department of Computer Science, National Taiwan University, 2003. URL: http://www.csie.ntu.edu.tw/~cjlin/papers.html.

[51] X. Jing et al. "Semi-Supervised Multi-View Correlation Feature Learning with Application to Webpage Classification". In: *AAAI*. 2017.

[52] T. Joachims. "Transductive Inference for Text Classification Using Support Vector Machines". In: *Proceedings of the Sixteenth International Conference on Machine Learning*. ICML '99. Morgan Kaufmann Publishers Inc., 1999, pp. 200–209. URL: http://dl.acm.org/citation.cfm?id=645528.657646.

[53] J. Kivinen and M. Warmuth. *Exponentiated Gradient versus Gradient Descent for Linear Predictions*. Tech. rep. Santa Cruz, CA, USA, 1994.

[54] W.H. Kruskal and W.A. Wallis. "Use of Ranks in One-Criterion Variance Analysis". In: *Journal of the American Statistical Association* 47.260 (1952), pp. 583–621. DOI: 10.1080/01621459.1952.10483441. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1952.10483441.

[55] L.I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004. ISBN: 0471210781.

[56] D.D. Lewis. "A Sequential Algorithm for Training Text Classifiers: Corrigendum and Additional Data". In: *SIGIR Forum* 29.2 (Sept. 1995), pp. 13–19. ISSN: 0163-5840. DOI: 10.1145/219587.219592. URL: http://doi.acm.org/10.1145/219587.219592.

[57] D.D. Lewis and J. Catlett. "Heterogenous Uncertainty Sampling for Supervised Learning". In: *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*. ICML'94. New Brunswick, NJ, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 148–156. ISBN: 1-55860-335-2. URL: http://dl.acm.org/citation.cfm?id=3091574.3091593.

[58] D.D. Lewis and W.A. Gale. "A Sequential Algorithm for Training Text Classifiers". In: *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '94. Springer-Verlag New York, Inc., 1994, pp. 3–12. ISBN: 0-387-19889-X. URL: http://dl.acm.org/citation.cfm?id=188490.188495.

[59] M. Li and Z. Zhou. "Improve Computer-Aided Diagnosis With Machine Learning Techniques Using Undiagnosed Samples". In: *IEEE Trans. Systems, Man, and Cybernetics, Part A* 37 (2007), pp. 1088–1098.

[60] M. Lichman. *UCI Machine Learning Repository*. 2013. URL: http://archive.ics.uci.edu/ml.

[61] D.J. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[62] D.J.C. MacKay. "Information-Based Objective Functions for Active Data Selection". In: *Neural Computation* 4.4 (1992), pp. 590–604. DOI: 10.1162/neco.1992.4.4.590.

[63] P.K. Mallapragada et al. "SemiBoost: Boosting for Semi-Supervised Learning". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 31.11 (Nov. 2009), pp. 2000–2014. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2008.235. URL: http://dx.doi.org/10.1109/TPAMI.2008.235.

[64] L. Mason et al. "Functional Gradient Techniques for Combining Hypotheses". In: *Advances in Large Margin Classifiers*. Ed. by A Smola et al. MIT Press, 2000, pp. 221–246.

[65] A. McCallum and K. Nigam. "Employing EM and Pool-Based Active Learning for Text Classification". In: *Proceedings of the Fifteenth International Conference on Machine Learning*. ICML '98. Morgan Kaufmann Publishers Inc., 1998, pp. 350–358. ISBN: 1-55860-556-8. URL: http://dl.acm.org/citation.cfm?id=645527.757765.

[66] J. McCormick. *Predicitions 2017: Artificial Intelligence Will Drive The Insights Revolution*. Nov. 2016.

[67] P. Melville and R.J. Mooney. "Creating Diversity in Ensembles Using Artificial Data". In: *Journal of Information Fusion: Special Issue on Diversity in Multi Classifier Systems* 6.1 (2004), pp. 99–111. URL: http://www.cs.utexas.edu/users/ai-lab/?melville:if04.

[68] P. Melville and R.J. Mooney. "Diverse Ensembles for Active Learning". In: *Proceedings of the Twenty-first International Conference on Machine Learning*. ICML '04. ACM, 2004, pp. 74–. DOI: 10.1145/1015330.1015385. URL: http://doi.acm.org/10.1145/1015330.1015385.

[69] T.M. Mitchell. "Generalization as search". In: *Artificial Intelligence* 18.2 (1982), pp. 203 –226. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(82)90040-6. URL: http://www.sciencedirect.com/science/article/pii/0004370282900406.

[70] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN: 026201825X, 9780262018258.

[71] H.T. Nguyen and A. Smeulders. "Active Learning Using Pre-clustering". In: *Proceedings of the Twenty-first International Conference on Machine Learning*. ICML '04. Banff, Alberta, Canada: ACM, 2004, pp. 79–. ISBN: 1-58113-838-5. DOI: 10.1145/1015330.1015349. URL: http://doi.acm.org/10.1145/1015330.1015349.

[72] K. Nigam and R. Ghani. "Analyzing the Effectiveness and Applicability of Co-training". In: *Proceedings of the Ninth International Conference on Information and Knowledge Management*. CIKM '00. ACM, 2000, pp. 86–93. DOI: 10.1145/354756.354805. URL: http://doi.acm.org/10.1145/354756.354805.

[73] K. Nigam et al. "Text Classification from Labeled and Unlabeled Documents Using EM". In: *Mach. Learn.* 39.2-3 (May 2000), pp. 103–134. ISSN: 0885-6125. DOI: 10.1023/A:1007692713085. URL: https://doi.org/10.1023/A:1007692713085.

[74] T.T. Osugi, K. Deng, and S.D. Scott. "Balancing exploration and exploitation: a new algorithm for active machine learning". In: *Fifth IEEE International Conference on Data Mining (ICDM'05)* (2005), p. 8.

[75] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[76] C. Persello and L. Bruzzone. "Active and Semisupervised Learning for the Classification of Remote Sensing Images". In: *FLAIRS Conference*. Vol. 52. 11. 2014, pp. 6937–6956.

[77] S. Petrova and A.D. Solov'ev. "The Origin of the Method of Steepest Descent". In: *Historia Mathematica* 24.4 (1997), pp. 361 –375. ISSN: 0315-0860. DOI: https://doi.org/10.1006/hmat.1996.2146. URL: http://www.sciencedirect.com/science/article/pii/S0315086096921461.

[78] PwC. *2017 Global Digital IQ Survey*. 2017. URL: http://www.pwc.com/us/en/advisory-services/digital-iq/assets/pwc-digital-iq-report.pdf#515.

[79] J.R. Quinlan. "Bagging, Boosting, and C4.S". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1*. AAAI'96. Portland, Oregon: AAAI Press, 1996, pp. 725–730. ISBN: 0-262-51091-X. URL: http://dl.acm.org/citation.cfm?id=1892875.1892983.

[80] J.R. Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN: 1-55860-238-0.

[81] E. Riloff, J. Wiebe, and W. Phillips. "Exploiting Subjectivity Classification to Improve Information Extraction". In: *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*. AAAI'05. Pittsburgh, Pennsylvania: AAAI Press, 2005, pp. 1106–1111. URL: http://dl.acm.org/citation.cfm?id=1619499.1619511.

[82] C. Rosenberg, M. Hebert, and H. Schneiderman. "Semi-Supervised Self-Training of Object Detection Models". In: *Proceedings of the Seventh IEEE Workshops on Application of Computer Vision (WACV/MOTION'05) - Volume 1 - Volume 01*. WACV-MOTION '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 29–36. DOI: 10.1109/ACVMOT.2005.107. URL: http://dx.doi.org/10.1109/ACVMOT.2005.107.

[83] SAS. *Machine Learning: What it is and why it matters*. 2016. URL: https://www.sas.com/it_it/insights/analytics/machine-learning.html (visited on 03/29/2016).

[84] H. Scudder. "Probability of error of some adaptive pattern-recognition machines". In: *Information Theory, IEEE Transactions on* 11.3 (July 1965), pp. 363–371.

[85] B. Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison, 2009. URL: http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf.

[86] B.M. Shahshahani and D.A. Landgrebe. "The effect of unlabeled samples in reducing the small sample size problem and mitigating the Hughes phenomenon." In: *IEEE Trans. Geoscience and Remote Sensing* 32.5 (1994), pp. 1087–1095. URL: http://dblp.uni-trier.de/db/journals/tgrs/tgrs32.html#ShahshahaniL94.

[87] C.E. Shannon. "A Mathematical Theory of Communication". In: *SIGMOBILE Mob. Comput. Commun. Rev.* 5.1 (Jan. 2001), pp. 3–55. ISSN: 1559-1662. DOI: 10.1145/584091.584093. URL: http://doi.acm.org/10.1145/584091.584093.

[88] J. Stefanowski and M. Pachocki. "Comparing Performance of Committee Based Approaches to Active Learning". In: *Recent Advances in Intelligent Information Systems* (2009), pp. 457–470. URL: https://pdfs.semanticscholar.org/9ec2/31e3571e3cdfd15598c2e1d303693852a0be.pdf.

[89] M. Stikic, K. Van Laerhoven, and B. Schiele. "Exploring Semi-supervised and Active Learning for Activity Recognition". In: *Proceedings of the 2008 12th IEEE International Symposium on Wearable Computers*. ISWC '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 81–88. ISBN: 978-1-4244-2637-9. DOI: 10.1109/ISWC.2008.4911590. URL: https://doi.org/10.1109/ISWC.2008.4911590.

[90] W.N. Street, W.H. Wolberg, and O.L. Mangasarian. "Nuclear feature extraction for breast tumor diagnosis". In: *Biomedical Image Processing and Biomedical Visualization*. Vol. 1905. July 1993, pp. 861–870. DOI: 10.1117/12.148698.

[91] SURFsara. *Description of the Lisa system*. URL: https://userinfo.surfsara.nl/systems/lisa/description.

[92] M. Szummer and T. Jaakkola. "Partially Labeled Classification with Markov Random Walks". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01. Vancouver, British Columbia, Canada: MIT Press, 2001, pp. 945–952. URL: http://dl.acm.org/citation.cfm?id=2980539.2980661.

[93] S. Tong and D. Koller. "Support Vector Machine Active Learning with Applications to Text Classification". In: *J. Mach. Learn. Res.* 2 (Mar. 2002), pp. 45–66. ISSN: 1532-4435. DOI: 10.1162/153244302760185243. URL: https://doi.org/10.1162/153244302760185243.

[94] V.N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[95] M. Vlaming. "Deloitte has opened the Artificial Intelligence Center of Expertise". In: (Nov. 2017). URL: https://www2.deloitte.com/nl/nl/pages/over-deloitte/articles/deloitte-has-opened-the-artificial-intelligence-center-of-expertise.html.

[96] Wikipedia. *Bias-Variance Tradeoff — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-February-2018]. URL: https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff.

[97] Wikipedia. *Hyperparameter optimization — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-January-2018]. URL: https://en.wikipedia.org/wiki/Hyperparameter_optimization#Grid_search.

[98] Wikipedia. *k-d tree — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-January-2018]. URL: https://en.wikipedia.org/wiki/K-d_tree.

[99] Wikipedia. *Semi-supervised learning — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-November-2017]. URL: https://en.wikipedia.org/w/index.php?title=Semi-supervised_learning&oldid=803365422.

[100] Wikipedia. *Type I and type II errors — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-January-2018]. URL: https://en.wikipedia.org/wiki/Type_I_and_type_II_errors.

[101] B. Xiong, X. Zhang, and W. Jiang. "Semi-Supervised Classification based on Gaussian Mixture Model for remote imagery". In: *Science China Technological Sciences* 53.1 (2010), pp. 85–90. ISSN: 1862-281X. DOI: 10.1007/s11431-010-3211-5. URL: https://doi.org/10.1007/s11431-010-3211-5.

[102] D. Yarowsky. "Unsupervised Word Sense Disambiguation Rivaling Supervised Methods". In: *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics*. ACL '95. Cambridge, Massachusetts: Association for Computational Linguistics, 1995, pp. 189–196. DOI: 10.3115/981658.981684. URL: https://doi.org/10.3115/981658.981684.

[103] Y. Zhou and S.A. Goldman. "Democratic co-learning". In: *16th IEEE International Conference on Tools with Artificial Intelligence* (2004), pp. 594–602.

[104] Z. Zhou. "When semi-supervised learning meets ensemble learning". In: *Frontiers of Electrical and Electronic Engineering in China* 6.1 (2011), pp. 6–16. ISSN: 1673-3584. DOI: 10.1007/s11460-011-0126-2. URL: https://doi.org/10.1007/s11460-011-0126-2.

[105] Z. Zhou and M. Li. "Semi-supervised learning by disagreement". In: *Knowledge and Information Systems* 24.3 (2010), pp. 415–439. ISSN: 0219-3116. DOI: 10.1007/s10115-009-0209-z. URL: https://doi.org/10.1007/s10115-009-0209-z.

[106] Z. Zhou and M. Li. "Tri-Training: Exploiting Unlabeled Data Using Three Classifiers". In: *IEEE Trans. on Knowl. and Data Eng.* 17.11 (2005), pp. 1529–1541. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.186. URL: http://dx.doi.org/10.1109/TKDE.2005.186.

[107] Z. Zhou, J. Wu, and W. Tang. "Ensembling Neural Networks: Many Could Be Better Than All". In: *Artif. Intell.* 137.1-2 (May 2002), pp. 239–263. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(02)00190-X. URL: http://dx.doi.org/10.1016/S0004-3702(02)00190-X.

[108] X. Zhu. *Semi-Supervised Learning Literature Survey*. 2008.

[109] X. Zhu and I. Davidson. *Knowledge Discovery and Data Mining: Challenges and Realities*. Hershey, PA, USA: IGI Global, 2007. ISBN: 1599042525.

[110] X. Zhu and Z. Ghahramani. *Learning from labeled and unlabeled data with label propagation*. 2002. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.3864.