Email classification with limited labeled data

Peter Engström 36939

Master's Thesis in Software Engineering

Supervisors: Dragos Truscan, Tanwir Ahmad

Faculty of Science and Engineering

Åbo Akademi

2020

Subject: Software Engineering	
Author: Peter Engström	
Title: Email classification with limited labeled data	
Supervisor: Dragos Truscan	Supervisor: Tanwir Ahmad

Abstract:

In this thesis I evaluate different ways of classifying email messages in the absence of a large number of pre-classified training messages. When using machine learning to classify incoming emails, this lack of pre-labeled emails is problematic. These labels can only be acquired by going through each email and manually labeling it, which for a large number of emails can take very long.

There are existing studies on machine learning for email classification, but the amount of labeled data available for such studies is generally large. However, there have been studies of non-email text classification with smaller amounts of labeled data, namely in the field of semi-supervised learning.

I experiment with supervised and semi-supervised (specifically, self-training) algorithms. The supervised methods used are random forest, naïve Bayes, support-vector machine, and AdaBoost. The experiments are conducted using 1000 labeled training emails. Three classes are used for the classification of data. Class “DB” represents the group that processes emails involving database stored procedures; class “CIM”, emails involving the Customer Interaction Management system; and class “Integration”, emails involving data integration.

The results show that self-training offers no significant advantage over the supervised methods for this task. Of the supervised methods, the support-vector machine achieves the best performance for each of the three classes.

Keywords: Email classification, text classification, machine learning

Date: 20.10.2020	Pages: 40

Contents

1 Introduction

The goal of this thesis is to evaluate different ways of classifying emails in the absence of a large number of pre-labeled emails. The context is that an IT team, providing CRM (Customer Relationship Management) and marketing solutions in an insurance company, has a common mailbox, from which each email must be forwarded to the team member most qualified to handle the email. This process is currently manual, with team members monitoring the mailbox in rotating shifts.

Many of the emails consist of automatic messages from a ticketing system, alerts about potentially malfunctioning processes, etc. In addition, there are internal emails from employees of the same company, emails from business partners of the company, and so on.

While there are many old emails from this mailbox available, there is no information (labels) regarding which team member processed the email. If machine learning is to be used to classify incoming emails, this lack of pre-labeled emails poses a problem. The only way of (accurately) acquiring these labels is by going through each email and labeling it manually, which for a large number of emails (say, more than 1000) may be prohibitively time-consuming.

The thesis is structured as follows: In Chapter 2: Background, the relevant theoretical background is covered. In Chapter 3: Related work, I study what others have published regarding text and email classification. In Chapter 4: Experiments, I define the research question, describe the tools used for conducting the experiments, the data used, and how the hyperparameters of the machine learning classifiers are tuned; I select the methods with which to experiment, and outline the execution of the experiments. In Chapter 5: Results, the results of the experiments are presented graphically, including how well the machine learning classifiers have predicted the email classes, as well as the training and validation times required for each classifier. In Chapter 6: Conclusions, I consider what conclusions can be drawn from the results, such as which classifier appears to be optimal

for our use case. Finally, in Chapter 7: Future research, I present ideas for future research into this topic.

2 Background

This section covers the relevant methods and theoretical background, including machine learning, regression analysis, TF-IDF, and receiver operating characteristic. The methods covered here were selected partly because of their suitability for this particular classification task (for details, see 4.2), and partly because they have been used in similar tasks by others (see Chapter 3).

2.1 Machine learning

According to Weik, *machine learning* is "The ability of a device, such as a computer, to improve its performance based on the results of its past performance" [1, p. 541]. There are three major paradigms of machine learning: supervised, unsupervised, and reinforcement learning (see Figure 1) [2], [3].

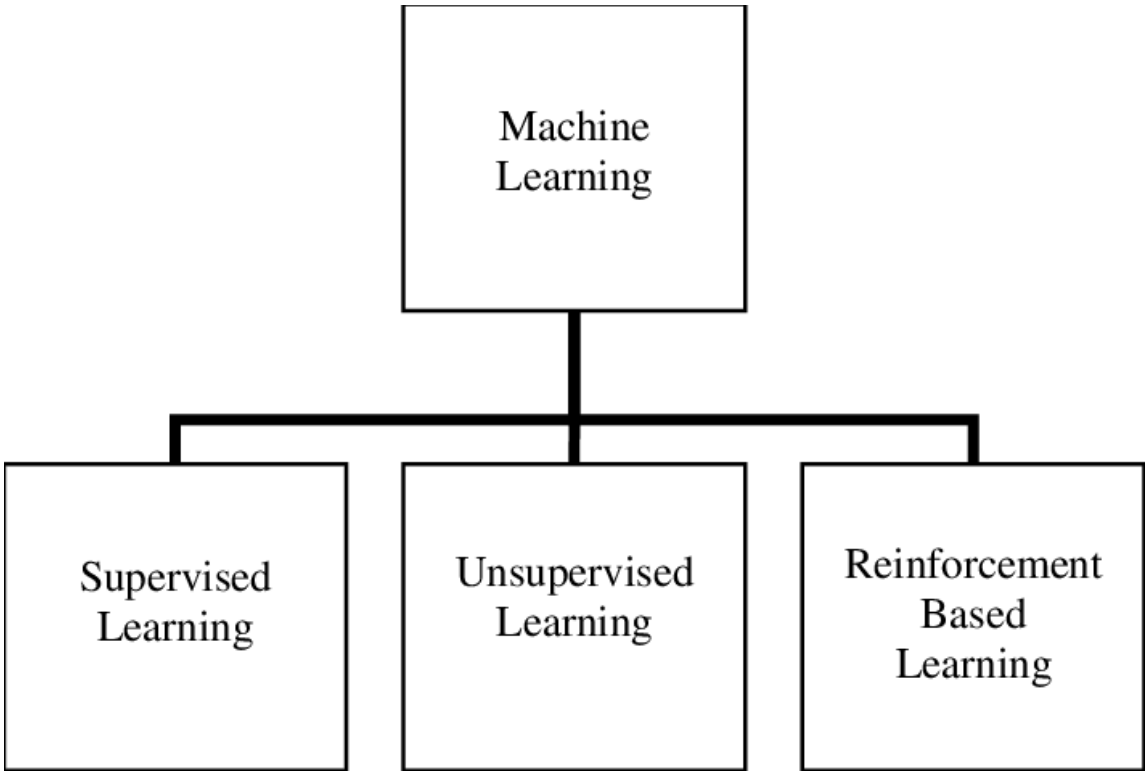

Figure 1: The three major paradigms of machine learning. [2]

Supervised learning means training a machine learning model using only pre-labeled data points, whereas *unsupervised learning* means finding patterns in data without using existing labels. In addition, a combination of the two called *semi-supervised learning* exists. It uses both previously labeled and unlabeled data to train the machine learning model (see Figure 2 for an illustration). In *reinforcement learning*, the learner is given positive feedback for desirable outcomes and negative feedback for undesirable outcomes [4, p. 830]. This section is an overview of each paradigm. [5, pp. 1-2]

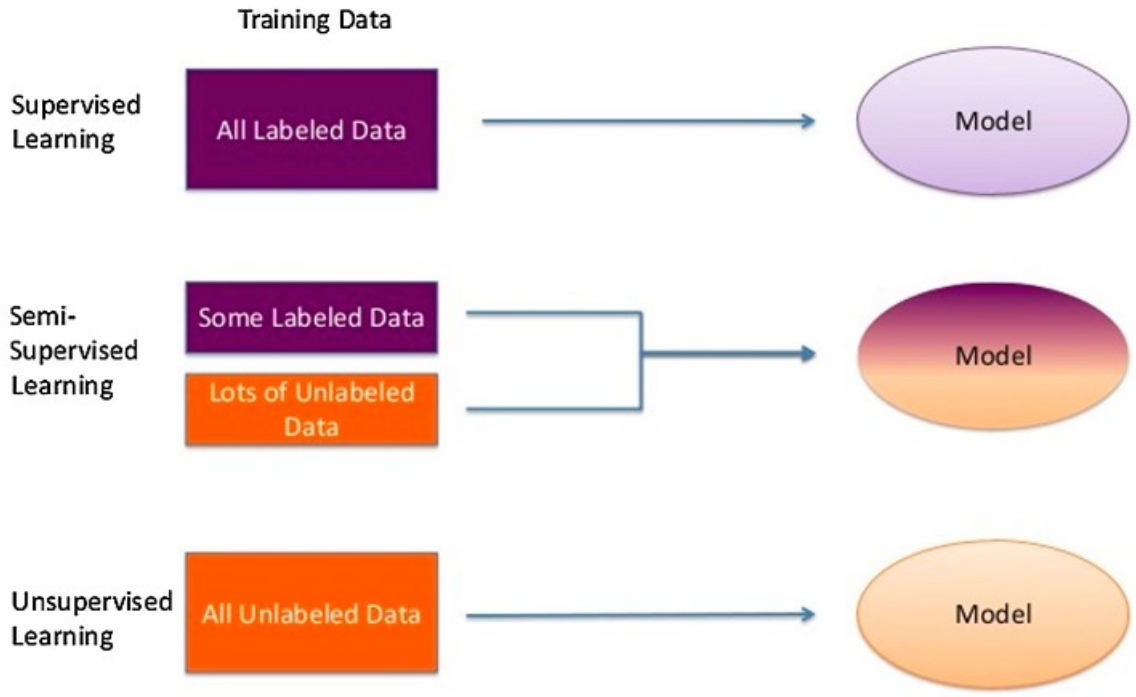

Figure 2: Difference between supervised, unsupervised, and semi-supervised learning. [6]

From the above, it can be concluded that supervised learning is useful when (sufficient) labeled data is available, unsupervised learning is useful for e.g. classification tasks where we do not know the classes in advance, and semi-supervised learning is useful when there is a limited amount of labeled data, and a large amount of unlabeled data. Reinforcement

learning is practical in situations such as games, where labeling input data would be too complicated (for more information, see 2.1.10) [4, pp. 830-831].

2.1.1 Regression analysis

Regression is a measure of how independent and dependent variables are related [7, p. 478]. Since it can be used for predicting the value of a dependent variable, given a value for the independent variable, it is closely related to classification tasks. E.g. if we know that a person has bought a product (independent variable), we can, based on historical data, use this to predict the likelihood of the same person buying another product (dependent variable).

The two types of regression that are considered here are linear regression, since it is the simplest form of regression [4, p. 718] and as such illustrates the concept well, and logistic regression, since it is frequently used for classification tasks [4, p. 727].

2.1.1.1 Linear regression

A simple form of regression is *linear regression*. Assume we have an independent variable x, and a dependent variable y. For instance, x and y could be the size and price of a house, respectively (see Figure 3) [4, p. 718]. For x, we have values $x_1, x_2, \ldots, x_n$, and for y, we have values $y_1, y_2, \ldots, y_n$. Then the function that best describes the relation between x and y is

$$y = bx + a \tag{1}$$

where

$$b = \frac{n\sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \tag{2}$$

and

$$a = \frac{\sum_{i=1}^{n} y_i - b\sum_{i=1}^{n} x_i}{n} \tag{3}$$

These coefficients minimize the squared sum $\sum_{i=1}^{n}(\Delta y_i)^2$, where Δy_i is the vertical distance between (x_i, y_i) and the linear function, see Figure 3. This method was introduced by either Carl Friedrich Gauss or Adrien Marie Legendre around the year 1800 [8]. [9, p. 173]

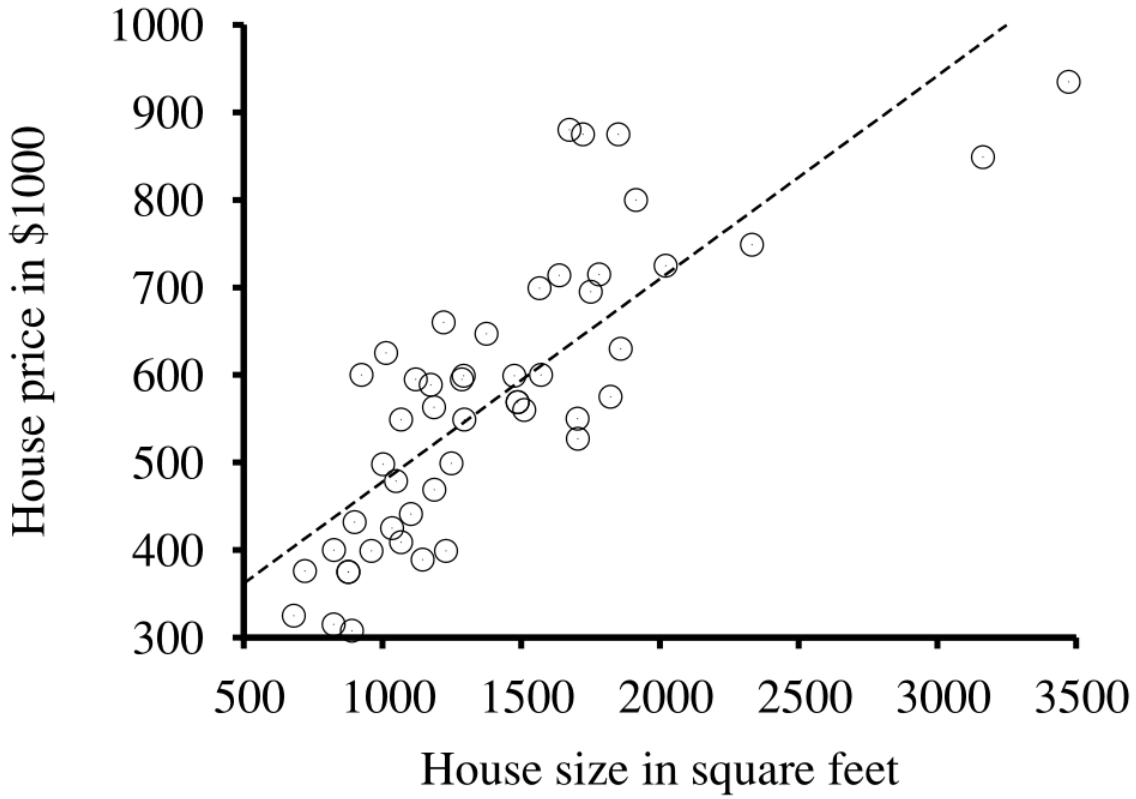

Figure 3: An example of linear regression. [4, p. 718]

2.1.1.2 Logistic regression

An alternative to linear regression when the dependent variable y is binary, i.e. 0 or 1, is *logistic regression*, wherein the function that best describes the relation between x and y is

$$y = \frac{e^{bx+a}}{1 + e^{bx+a}} \tag{4}$$

For an example of a typical logistic regression curve, see Figure 4. [10]

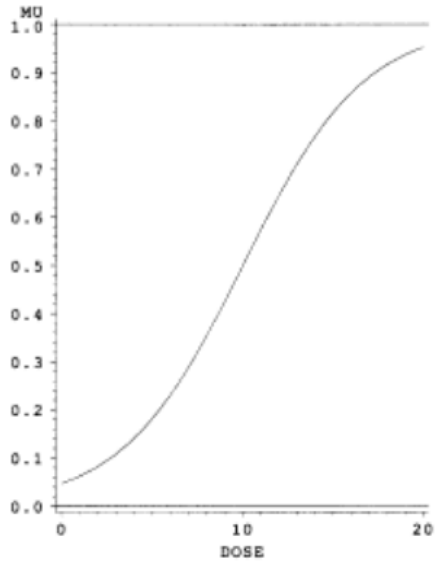

Figure 4: Typical logistic regression curve. [10]

Regression functions can also be used for classification. For example, the logistic function can be used for classification tasks by specifying a threshold value for y, e.g. 0.5. If, for a given input x, the logistic function outputs a value $y < 0.5$, we classify this as 0, otherwise (for $y \geq 0.5$) as 1. Logistic regression is generally more reliable than linear regression for classification tasks, but is slower to converge than linear regression when the data is linearly separable [4, p. 727].

2.1.2 Artificial neural network

The concept of *artificial neural networks* was introduced by McCulloch and Pitts in 1943 [11]. The first computer applying these principles was built by Minsky and Edmonds in 1950. In 1958, Frank Rosenblatt introduced the perceptron, a type of artificial neuron [12, p. 4]. [4, pp. 19-20]

An artificial neural network consists of interconnected artificial neurons. For a graphical representation of an artificial neuron, see Figure 5. It is composed of a central node, which

contains the input function, activation function, and output, as well as links leading into the node and out of it. [4, pp. 727-728]

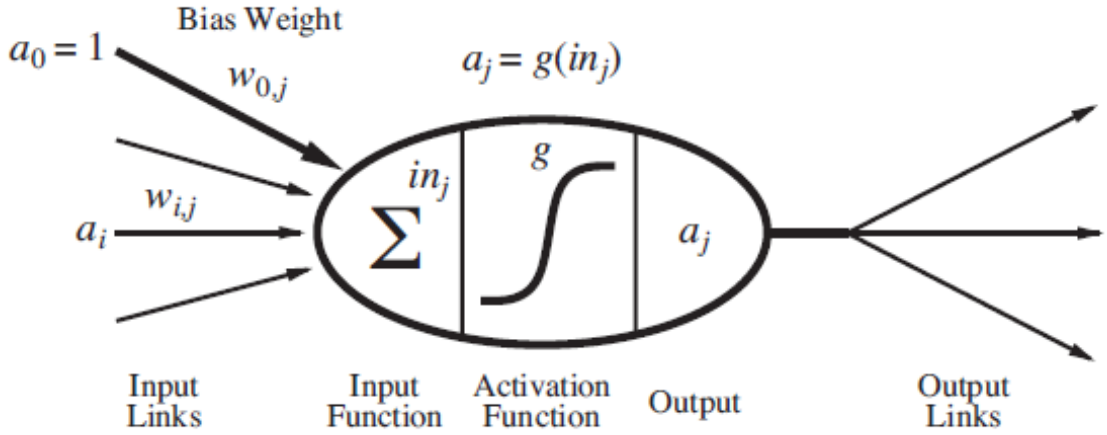

Figure 5: Artificial neuron. [4, p. 728]

According to [4, p. 728], each input link a_i has a certain weight $w_{i,j}$. Here i is the source node and j is the destination node, i.e. each link a_i goes from i to j. In addition, there is a constant input link a_0 which is always 1. First, the node calculates the weighted sum of all input links using the input function:

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i \tag{5}$$

after which the activation function g is used to calculate the output a_j:

$$a_j = g(in_j) \tag{6}$$

A trivial classification task using only a single neuron could be the following: Assume we have a number of reviews a_1, a_2, …, a_n for a film, and each review a_i has a value from 0 to 5 (stars), 0 being the worst and 5 being the best. Each of the inputs is weighted equally, i.e. $w_{i,j} = 1 / n$. Our activation function is

$$g(in_j) = \begin{cases} 1 & \text{if } in_j > 2.5, \\ 0 & \text{if } in_j \leq 2.5 \end{cases} \quad (7)$$

where output 1 means “good” and output 0 means “bad”.

Suppose we disagree with the results of this classifier, e.g. we think a film it has labeled “bad” is good. One way to rectify this is to add a greater weight to the inputs (reviews) with which we tend to agree, and a lesser weight to the inputs with which we tend to disagree. Adjusting the weights of links is how neural networks “learn” [4, pp. 724, 730].

Neural networks learn quickly even with large amounts of data, and are popular for e.g. image classification [13]. However, they require large amounts of data to learn [14], which may place them at a disadvantage for our case.

2.1.3 Decision tree

A *decision tree* is an interconnected set of rules, where each rule is represented by a node in the tree. A rule node takes an input, and based on the input value, branches out into one of its output nodes. [4, p. 698]

Figure 6 displays an example of a decision tree, where the decision is whether to wait for a table at a restaurant, the output value being either “yes” or “no”. The first rule node is based on the number of people in the restaurant. If there are none, we will not wait for a table; if there are some, we will wait for a table; if the restaurant is full, we will branch out into the next node, where the decision is made based on how long the expected waiting time is; and so on. [4, pp. 698-699]

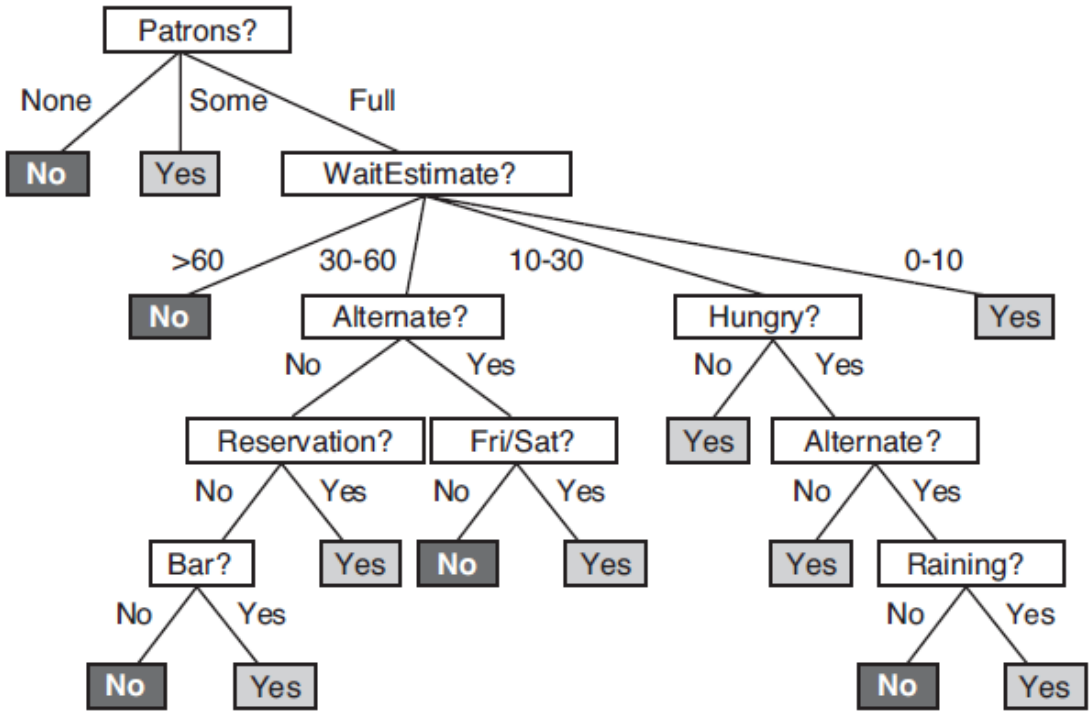

Figure 6: Example of a decision tree. The decision to make is whether to wait for a table at a restaurant, the possible output values being "yes" or "no". [4, p. 699]

The order of the features (e.g. Patrons or WaitEstimate in the above example) is determined by the *information gain* of each feature, when used on a training set. In binary classification tasks, like the above, the information gain for feature A is defined as

$$Gain(A) = B\left(\frac{p}{p+n}\right) - \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k + n_k}\right) \tag{8}$$

where p is the total number of positive samples ("yes") in the remaining training set and n is the total number of negative samples ("no"). Feature A splits the remaining training samples into d subsets; e.g. Patrons splits the samples into three subsets: None, Some, and Full. p_k and n_k are the number of positive and negative samples, respectively, in subset E_k, where $k = 1, 2, \ldots, d$. $B(q)$ is defined as

$$B(q) = -(q \log_2 q + (1-q) \log_2 1 - q)) \tag{9}$$

In the above example, the Patrons feature is used first, because it has the highest information gain for the training set (example training set in Table 1); WaitEstimate is used second, because it has the highest information gain for the remaining training set, etc. [4, pp. 699-704]

Table 1: Training set for restaurant example. [4, p. 700]

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

2.1.4 Ensemble learning

Ensemble learning means combining the predictions of multiple classifiers [4, p. 748]. This section covers the ensemble learning methods random forest and AdaBoost.

2.1.4.1 Random forest

A *random forest* is a collection of decision trees, each using a randomly sampled subset of the original data set. The output of a random forest classifier is the most frequent value of the outputs of each decision tree, i.e. the majority class. [15]

Using the example from 2.1.3, we could randomly sample 12 inputs (restaurants) from the original inputs x_1, x_2, …, x_{12}. This could result in the inputs [x_1, x_1, x_4, x_4, x_5, x_5, x_5, x_6, x_7, x_8, x_{10}, x_{12}], for example. We would use this modified training set to train a decision tree

as in 2.1.3. Then we would repeat the process for another decision tree, but with a different randomly sampled training set, and so on. If we then use these decision trees (or "forest", collectively) to predict the output for some new input, the output would be the average of the outputs of each decision tree. E.g. assume we use three decision trees, and their outputs are [1, 0, 0], with 1 meaning we should wait for a table, and 0 meaning we should not. Since 0 is the majority class, the output of the random forest will be 0.

Random forest is fast for classification of high-dimensional data such as text. However, since implementations of it tend to use only a random subset of features at a time, there is a high probability of excluding important features, resulting in lower accuracy. [16]

2.1.4.2 AdaBoost

AdaBoost (short for **Ada**ptive **Boost**ing) is similar to random forest; however, AdaBoost can for instance use one-level decision trees, also known as decision stumps, which means only one feature (at a time) is used for classification. Samples that have been labeled incorrectly are given more weight in the next classification by a decision stump. When the trees collectively select a class, the more accurate trees have more weight. In addition, instead of using random subsets of the training set, the entire training set is used. [4, pp. 749-751]

For the example in 2.1.3, we could choose Patrons as the feature for our first decision stump. This decision stump would decide to wait if the restaurant had some patrons without being full, and otherwise decide not to wait. Initially, the weight for each sample is $1/n$, where n is the total number of samples; in this case, the initial weight for each sample is therefore 1/12. The first decision stump classifies two samples ($\mathbf{x}_4$ and $\mathbf{x}_{12}$) incorrectly. The total weight *err* of the incorrectly classified samples is then $2 \cdot 1/12 = 1/6$. From this, we can calculate the weight α for this decision stump:

$$\alpha = \ln\left(\frac{1 - err}{err}\right) = \ln\left(\frac{1 - \frac{1}{6}}{\frac{1}{6}}\right) = 1.6 \tag{10}$$

The weight for each of the incorrectly classified samples is set to

$$w_i \leftarrow w_i e^{\alpha} = \frac{1}{12} e^{1.6} = 0.42 \tag{11}$$

Then the weights of all the samples are normalized by dividing them by the sum of the current weights, so that the sum of the samples becomes 1:

$$w_i \leftarrow \frac{w_i}{\sum w_i} \tag{12}$$

This process is then repeated for different features. When classifying a new data point, the majority class of the outputs of each decision stump, multiplied by the stump's weight α, is selected as the class of the data point. See Figure 7 for an illustration of the algorithm. In the figure, **x** is a data point, k_i is the class predicted by tree i for the data point, and k is the final class for the data point, arrived at through weighted voting [17]. [18]

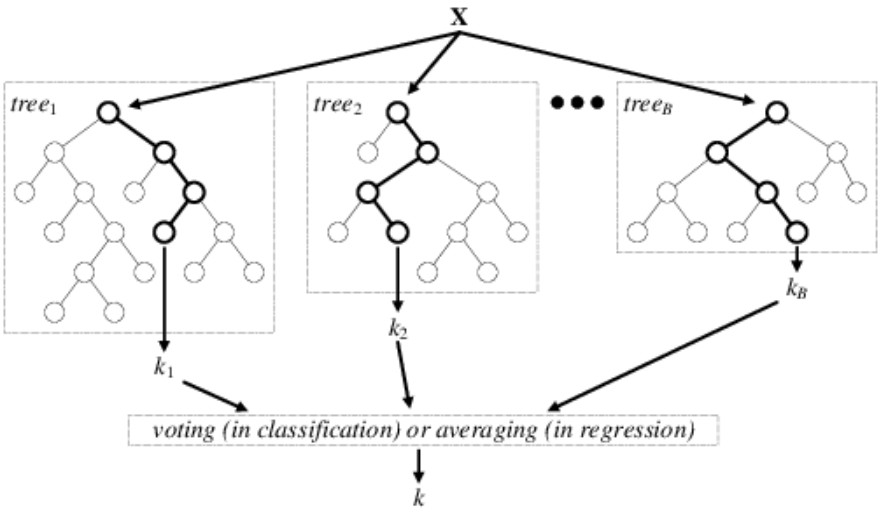

Figure 7: Illustration of AdaBoost algorithm. [17]

AdaBoost has the advantage of rarely overfitting, and it performs well for text classification [19]. However, it has problems classifying noisy data. We can mitigate this by using stop words (for further information, see 2.2). [20]

2.1.5 Naïve Bayes

Naïve Bayes is a classifier based on Bayes' rule, which was introduced by Thomas Bayes [21]. The classifier is called naïve because it assumes statistical independence between variables [4, pp. 497-499]. Its naïve assumptions give it an edge with regard to speed and ease of implementation, at the expense of predictive power. Nevertheless, naïve Bayes does not perform much worse than more advanced methods [22]. It is a popular choice for text classification tasks. [23]

The classifier for naïve Bayes is

$$C_{predicted} = \underset{c}{\operatorname{argmax}} P(C = c) \prod_{j=1}^{m} P\left(X_j = u_j \middle| C = c\right) \tag{13}$$

where $P(C = c)$ is the probability of class c based on its relative frequency in the training data, m is the number of features, and $P(X_j = u_j \mid C = c)$ is the probability that feature X_j has value u_j, given that the class is c. [24]

For example, assume we have the input data X = (Outlook = Overcast, Temperature = Hot, Humidity = High, Wind = Strong), and we want to predict whether the output class for this is "play tennis" (yes) or "do not play tennis" (no). We also have data for previous inputs and their outputs, see Table 2. Therefore, we must determine the largest of

$$P(C = yes) \prod_{j=1}^{4} P\left(X_j = u_j \middle| C = yes\right) \tag{14}$$

and

$$P(C = no) \prod_{j=1}^{4} P\left(X_j = u_j \middle| C = no\right) \tag{15}$$

Table 2: Example training data for naïve Bayes. [24]

Day	Outlook	Temperature	Humidity	Wind	PlayTennis (classification)
D_1	Sunny	Hot	High	Weak	No
D_2	Sunny	Hot	High	Strong	No
D_3	Overcast	Hot	High	Weak	Yes
D_4	Rain	Mild	High	Weak	Yes
D_5	Rain	Cool	Normal	Weak	Yes
D_6	Rain	Cool	Normal	Strong	No
D_7	Overcast	Cool	Normal	Strong	Yes
D_8	Sunny	Mild	High	Weak	No
D_9	Sunny	Cool	Normal	Weak	Yes
D_{10}	Rain	Mild	Normal	Weak	Yes
D_{11}	Sunny	Mild	Normal	Strong	Yes
D_{12}	Overcast	Mild	High	Strong	Yes
D_{13}	Overcast	Hot	Normal	Weak	Yes
D_{14}	Rain	Mild	High	Strong	No

The training set has 9 samples with $C = yes$, so $P(C = yes) = 9/14$. Inserting into (14):

$$P(C = yes)\prod_{j=1}^{4} P(X_j = u_j | C = yes)$$

$$= \frac{9}{14}\prod_{j=1}^{4} P(X_j = u_j | C = yes)$$

$$= \frac{9}{14} P(Overcast|C = yes)\ P(Hot|C = yes)\ P(High|C = yes)\ P(Strong|C = yes)$$

$$= \frac{9}{14}\cdot\frac{4}{9}\cdot\frac{2}{9}\cdot\frac{3}{9}\cdot\frac{3}{9} = \frac{648}{91854}$$

Similarly, inserting $P(C = no) = 5/14$ into (15):

$$P(C = no)\prod_{j=1}^{4} P(X_j = u_j | C = no)$$

$$= \frac{5}{14} \prod_{j=1}^{4} P(X_j = u_j | C = no)$$

$$= \frac{5}{14} \cdot \frac{0}{5} \cdot \frac{2}{5} \cdot \frac{4}{5} \cdot \frac{3}{5} = 0$$

As 648/91854 is greater than 0, this input would be classified as “yes”. [24]

2.1.6 *k*-nearest neighbors

The *k-nearest neighbors* algorithm was first formulated by Fix and Hodges [25]. It performs well for data with few dimensions, but has issues with high-dimensional data [4, p. 739]. The following is the algorithm: y is an unlabeled data point whose label we want to predict. For each labeled data point x_i we have, calculate the distance between y and x_i. Order these distances in ascending order, then select the data points associated with the k first ones. Evaluate what is the most frequent label among these k data points, and assign that label to y. [26]

The distance metric used in *k*-nearest neighbors is commonly Euclidean distance, also known as Cartesian distance. This is defined as

$$\mathrm{d}(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{16}$$

where $\mathbf{a} = [a_1, a_2, \ldots a_n]$, $\mathbf{b} = [b_1, b_2, \ldots b_n]$, and n is the number of dimensions (in this case, features). See Figure 8 for an illustration in two-dimensional space. [7, p. 191]

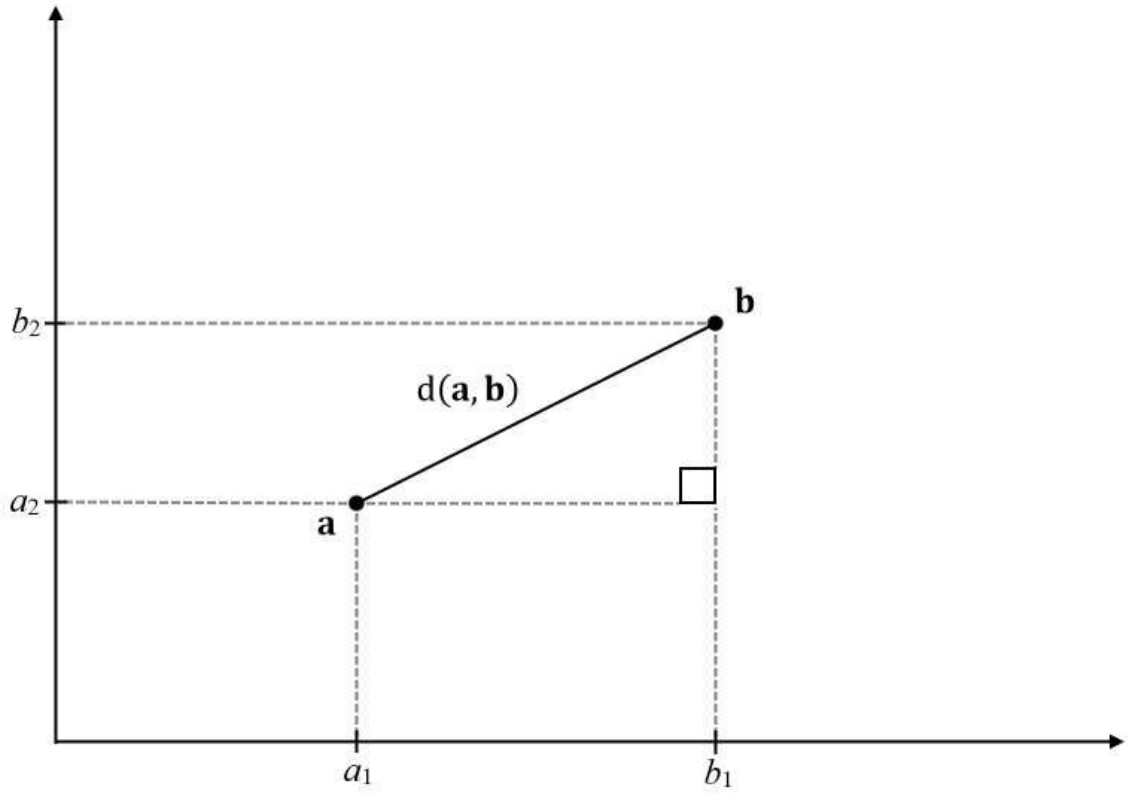

Figure 8: Illustration of Euclidean distance in two-dimensional space. $\mathrm{d}(\mathbf{b}, \mathbf{a}) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2}$

2.1.7 Support-vector machine

The *support-vector machine* is a method that handles high-dimensional data well, which makes it a good candidate for text classification [27]. However, its training and classification times tend to be high [28].

The method classifies data points by separating data points of different classes with a hyperplane, which in two-dimensional space is a line, in three-dimensional space a two-dimensional plane, and so on [7, p. 265]. The hyperplane is selected by maximizing the distance from the nearest data point of any class to the hyperplane; see Figure 9. [29]

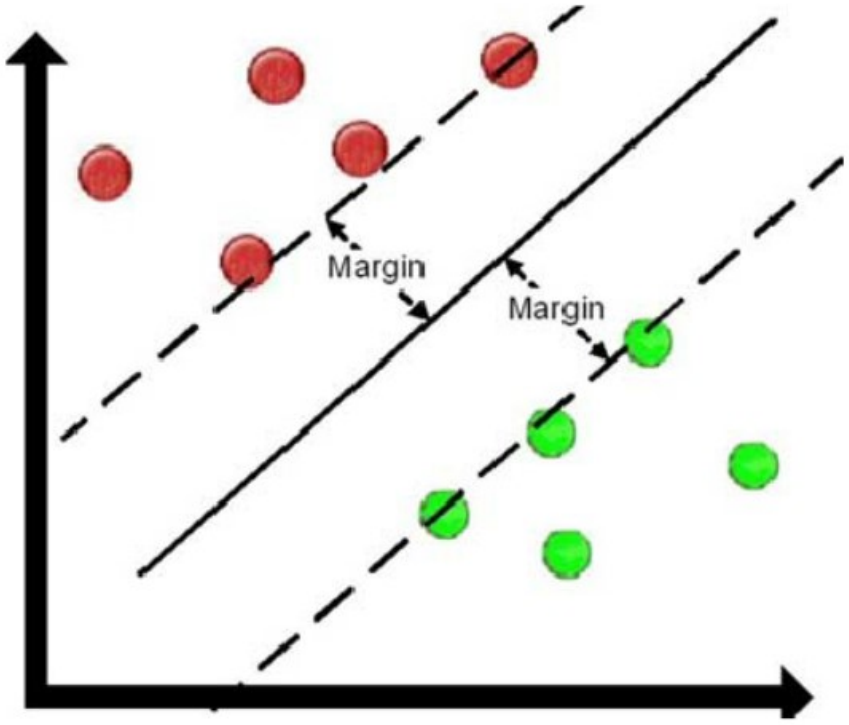

Figure 9: Example of support-vector machine separation of data points. The middle (solid) line is the optimal hyperplane. [29]

If the hyperplane is defined as

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \tag{17}$$

where **x** and **w** are *M*-dimensional vectors and b is a scalar, then the optimal hyperplane can be determined by minimizing

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{M}\xi_i \tag{18}$$

where C is the so-called penalty parameter, and ξ_i is the distance between the margin and the data points that are on the wrong side of the margin. [30]

2.1.8 Unsupervised learning

Unsupervised learning is the discovery of patterns in data without the use of existing labels. One type of unsupervised learning is *clustering*, which is the organizing of data into groups or "clusters". An important work in this field is *Principles of Numerical Taxonomy* by Robert Sokal and Peter Sneath, in which clustering is used for biological classification of organisms [31]. [32, pp. 6-8]

An example of a clustering method is *k*-means. The *k* stands for the number of clusters into which the data points should be clustered. The following is the *k*-means algorithm. For each of the *k* clusters to be computed, a vector called a "mean" is randomly chosen. Then each data point *x* in the data set is assigned to the cluster of its nearest mean:

$$k = \underset{i}{\operatorname{argmin}}[d(m_i, x)] \tag{19}$$

where *k* is the cluster to which *x* is assigned, m_i is the mean for cluster *i*, and *d* is the distance function. Then each mean *m* is updated:

$$m = \frac{\sum_{j=1}^{n} x_j}{n} \tag{20}$$

where x_j is the j^{th} data point assigned to cluster *m*, and *n* is the total number of data points assigned to *m*. [33, pp. 285-286]

2.1.9 Semi-supervised learning

Semi-supervised learning utilizes both labeled and unlabeled data to train a machine learning model, which makes it a combination of supervised and unsupervised learning. An advantage of semi-supervised learning is that it can yield high classification accuracy even with small amounts of training data. However, it can also hurt performance if the amount of labeled data is high enough [34].

Self-training, also known as self-learning or self-labeling, is arguably the oldest form of semi-supervised learning. It has been in use since at least 1965 [35]. In self-training, a supervised classifier is first trained on the labeled training data. The classifier is then used

to predict labels for the unlabeled training data. Those of the newly labeled data points that have been labeled with sufficiently high confidence (over a specified threshold), are then added to the original labeled dataset. This augmented dataset is then used to retrain the classifier, which is then again used to predict labels for the unlabeled data which have not yet been added to the labeled data, and so on. [5, p. 3] [36]

2.1.10 Reinforcement learning

In *reinforcement learning*, a learner is given feedback based on its performance: if it performs well, it is given positive feedback, if poorly, negative. This is useful in situations such as games, where labeling input data would be too complicated [4, pp. 830-831]. However, reinforcement learning has its drawbacks, such as often requiring large amounts of data [37] and performing poorly with high-dimensional data [38], such as text.

The following is a simple example of reinforcement learning. Suppose we are playing tic-tac-toe, where players take turns drawing symbols on a 3×3 board; one player draws "X", the other "O" (see Figure 10 for an example). The player who first manages to place three Xs or Os in a row wins. [39, p. 10]

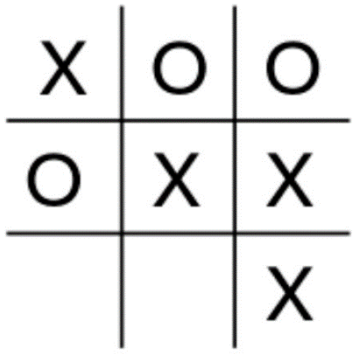

Figure 10: A game of tic-tac-toe. [39, p. 10]

We are training a reinforcement learning algorithm to play as X. The algorithm sets a value for each state of Xs and Os in the game; the value represents the probability of X winning the game, given that state. Therefore, a state with three Xs in a row has a value

of 1, and a state with three Os in a row has a value of 0. All other states initially have a value of 0.5. [39, p. 11]

We play several games against O. During a game, we usually select our next move (which becomes the next state in the game) by selecting the state with the highest value. However, on occasion we may select a random state instead, in order to explore the consequences. In either case, after making the move, the value of the pre-move state is changed so it is closer to the value of the post-move state. How much the value is changed depends on the implementation. Through multiple games, the values gradually approach the actual probability of us winning the game from that state, and the optimal move is therefore the one with the highest value. [39, pp. 11-13]

2.2 TF-IDF

Term frequency-inverse document frequency (TF-IDF) is a method of weighting words in documents. Advantages of TF-IDF include simplicity and efficiency, while disadvantages include the inability to recognize relationships between words, such as synonyms [40].

Term frequency (TF) is defined as follows:

$$\mathrm{TF}_{wd} = \frac{f_{wd}}{\max_a(f_{ad})} \tag{21}$$

where f_{wd} is the number of occurrences of word w in document d, and $\max_a(f_{ad})$ is the largest number of occurrences of any word in document d. *Inverse document frequency* (IDF) is defined as

$$\mathrm{IDF}_w = \log_2\left(\frac{N}{n_w}\right) \tag{22}$$

where N is the total number of documents, and n_w is the number of those documents in which word w occurs. The TF-IDF weight of a word in a document is the product of the word's TF and IDF, i.e.

$$\text{TF-IDF}_{wd} = \mathrm{TF}_{wd} \cdot \mathrm{IDF}_w \tag{23}$$

It may sometimes be advantageous to discard some words which are not useful in classifying documents, such as “the”, “a”, or “an”. These are called *stop words*. [41, p. 8]

An alternative form of TF called *sublinear TF scaling* is sometimes used, wherein 1 + log TF_{wd} is substituted for TF_{wd}. The rationale for this is that, even if a word occurs f_{wd} times in a document, it is not necessarily f_{wd} times as important. [42]

2.3 Receiver operating characteristic

There are different ways of measuring how well classifiers predict the classes of data points. *Receiver operating characteristic* (ROC) is a useful metric for classification tasks where the classes are unevenly distributed. [43]

ROC has two dimensions: true positive rate (*TPR*) and false positive rate (*FPR*). *TPR*, also known as *recall* or *sensitivity*, is defined as:

$$TPR = \frac{TP}{P} \tag{24}$$

where *TP* (true positives) is the number of data points correctly classified as positive, and *P* (positives) is the total number of positive data points, i.e. including those falsely classified as negative. Similarly, *FPR* is defined as:

$$FPR = \frac{FP}{N} \tag{25}$$

where *FP* (false positives) is the number of data points incorrectly classified as positive, and *N* is the total number of negative data points (including those falsely classified as positive). [43]

A point on the ROC curve consists of the *TPR* and *FPR* for a given confidence threshold; by confidence threshold is meant the level of confidence required for a data point to be classified as positive. Figure 11 shows an example of an ROC curve. The number next to a cross denotes the confidence threshold. [43]

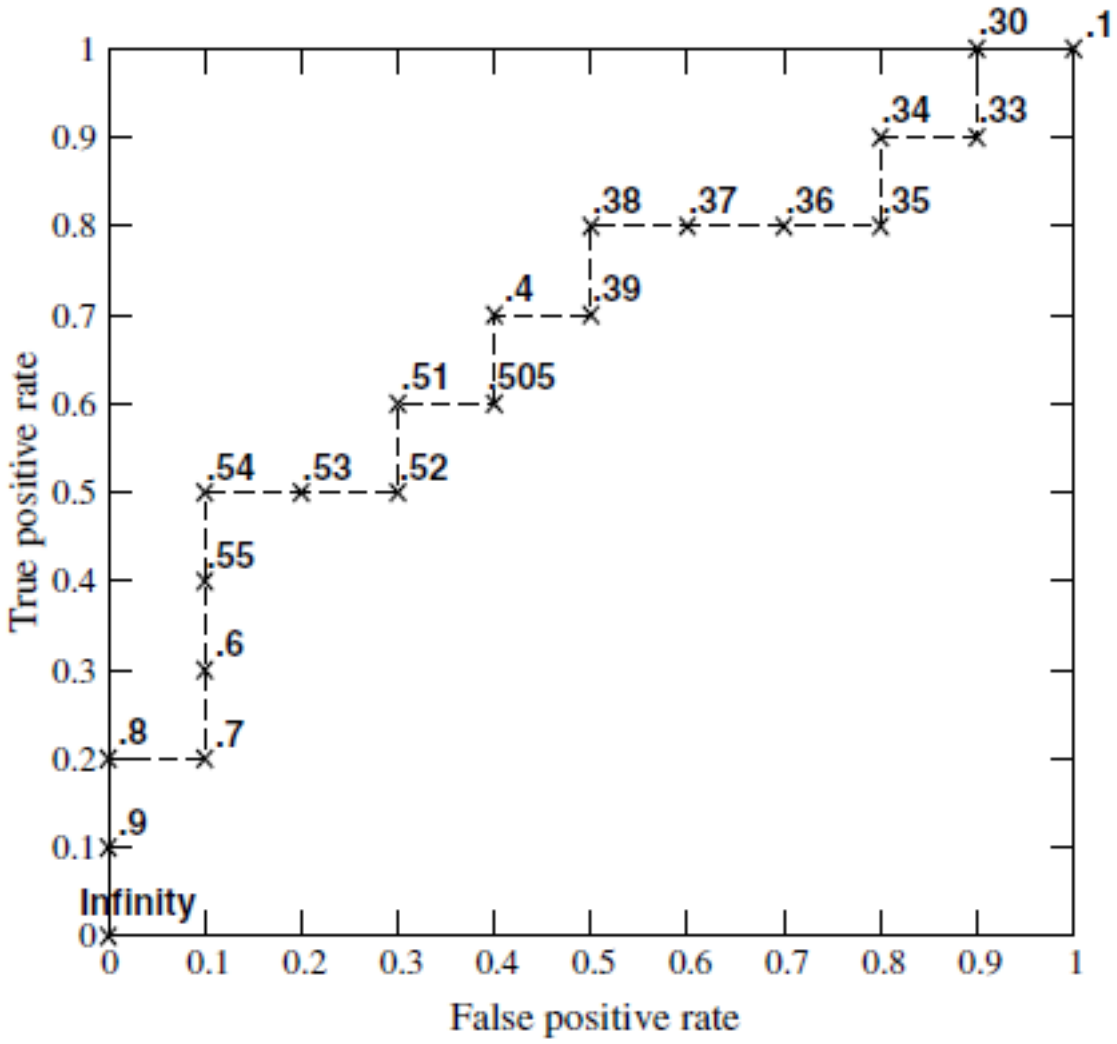

Figure 11: Example of an ROC curve. [43]

The *area under the ROC curve* (AUC), normalized by dividing it by the total area where ROC is defined, is commonly used as a performance metric for classifiers, with a larger AUC meaning better performance [43].

3 Related work

The amount of labeled data available for studies involving the use of machine learning for email classification is generally large. See, for instance, the work by Klimt and Yang [44], where they use a labeled dataset of 200 399 items, representing email messages, with the label in this case being the folder to which each message belongs. However, there have been studies of text classification in general (i.e. not just email messages) with smaller amounts of labeled data, namely in the field of semi-supervised learning. An example of this is the study by Nigam et al. [34], where 300 labeled samples are used to train a semi-supervised model. They conclude that adding unlabeled documents to the dataset increases the accuracy of the classifier, as illustrated in Figure 12.

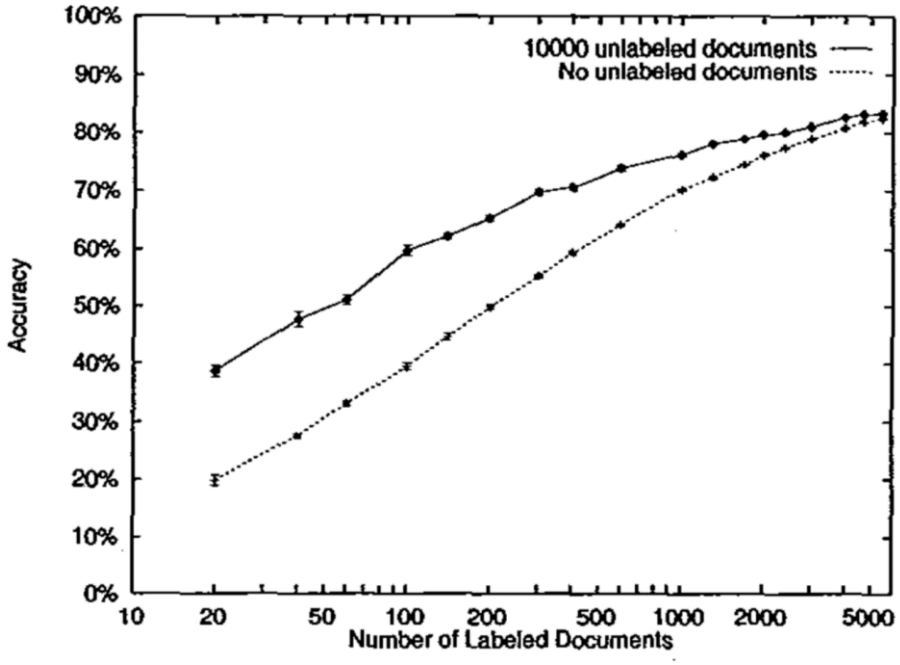

Figure 12: Semi-supervised classifier accuracy as a function of the number of labeled and unlabeled documents. [34]

Bahgat et al. [45] list several supervised learning methods used for spam email filtering, including naïve Bayes, random forest, and multilayer perceptron. Among their findings is that random forest can achieve an accuracy of 92%.

Several papers have been published where the authors compare different methods of email classification. For instance, Gomes et al. [46] compare naïve Bayes to hidden Markov model (HMM) and find that HMM is more accurate for email classification. Yitagesu and Tijare [47] present an overview of methods for email classification and find that naïve Bayes and support-vector machine are potentially effective for this. Phadke [48] successfully uses "self-learning" to classify emails based on user preferences, with an accuracy of 93.26% and an error rate of 6.552%. Kiritchenko et al. [49] successfully incorporate temporal information (based on email timestamps) to decrease classification error. Cormack [50] finds that sequential compression models, as well as on-line versions of logistic regression and support-vector machines, are among the most effective methods of spam filtering.

Ayodele et al. [51] successfully use unsupervised learning (using a custom approach utilizing word frequencies) to group and summarize emails; they achieve an accuracy of 98%. Cheng and Li [52] use graph-based semi-supervised learning for spam filtering, with an average AUC score of 0.933. Mujtaba et al. [53] find that supervised methods are the most widely used technique for email classification, and that the support-vector machine is the most popular of these.

My approach is different from the above, in that I use a relatively small number of emails for the classification, whereas the numbers in most of the above studies have been large, except for e.g. Nigam et al. [34], but they use Usenet postings instead of emails, and only use one supervised method (naïve Bayes), while I compare several. Whereas many studies of email classification focus on spam filtering, such as Bahgat et al. [45] (see above), I use it to determine the appropriate recipient (see Chapters 1 and 4 for further information).

4 Experiments

The research question I aim to answer in this thesis is "*what is the best-performing method of classifying emails, when the number of pre-labeled training emails is at most 1000?*" This chapter describes the setup used in the experiments to answer this research question, including the dataset and tools used.

4.1 Dataset

The data contains automated messages from a ticketing system, alerts about potentially malfunctioning processes, etc. These emails tend to contain much technical jargon. In addition, there are internal emails from employees of the same company, emails from business partners of the company, and so on. The vast majority of all the emails are in English, while a few are in Finnish or Swedish.

Three classes are used for the classification of data, each representing a group of email recipients, the group being determined by who is most qualified to process a given email. Figure 13 shows how the classes are distributed among the 1200 labeled emails. Class "DB" represents the group that processes emails involving database stored procedures; class "CIM", emails involving the Customer Interaction Management system; and class "Integration", emails involving data integration. There may be emails of other types as well, but since it is most practical to forward every email to one of these groups, we do not use a separate category for miscellaneous emails, but instead assign emails to the category to which they are most closely related.

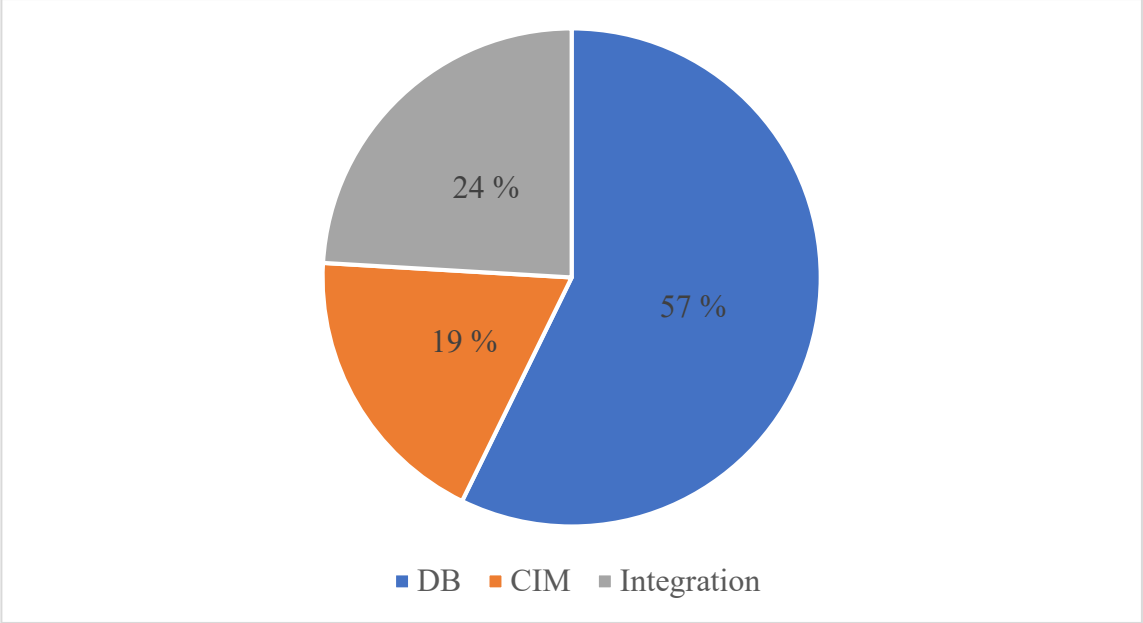

Figure 13: Distribution of email categories.

The data is exported from Microsoft Office Outlook 365 to a comma-separated values (CSV) file. In total, there are 8204 rows (emails). Using a Python script, the unused columns are removed, so that only the message subject and body, concatenated into a single column, remain. A script is then used to randomly sample 1200 emails to be labeled, while the rest is saved to a new file. I then manually label the 1200 emails, of which 1000 are randomly sampled for use in testing, and the remaining 200 are used for validation.

4.2 Selection of methods

Since artificial neural networks generally require large amounts of data to learn well [14], I will exclude them from the experiments. I will also exclude *k*-nearest neighbors, since it is ill-suited for high-dimensional data [4, p. 739] like text; logistic regression is excluded for the same reason [54]. I will include random forest [16], naïve Bayes [23], the support-vector machine [27], and AdaBoost [19] in the experiments, since each of these is well suited either for text classification in particular, or high-dimensional data in general.

Self-training is a promising candidate, since semi-supervised learning excels when there is little labeled data [34]. Unsupervised methods such as *k*-means clustering do not seem appropriate for my case, since I use predefined categories. Reinforcement learning also seems inappropriate, since this problem is not too complex for supervised learning, and because reinforcement learning often requires large amounts of data [37], and has issues with high-dimensional data [38] like text.

I will use TF-IDF to transform the emails into numerical data, since it is a simple and efficient method [40]. Since most of the documents in our dataset are in English, and some methods such as AdaBoost may be especially sensitive to "noisy" data (see 2.1.4.2), some of the most common English words will be used as stop words. In addition, I will use sublinear TF scaling, following the reasoning in 2.2. AUC will be used to evaluate the performance of methods, since it is useful when the classes are unevenly distributed (like in this case; see Figure 13) [43].

4.3 Running the experiments

The experiments are conducted using Python 3.7.1, and the scikit-learn 0.21.3 library. Scikit-learn is an open-source Python machine learning library, first released in 2007 [55]. Scikit-learn itself uses the Python mathematics library SciPy [56], which in turn relies on the NumPy library for multidimensional arrays [57]. The hardware used is an Intel Core i7-4700MQ CPU, and 8 GiB of DDR3 SDRAM.

For hyperparameter tuning, a *grid search* is used. Grid search means trying every possible combination of values sampled from each hyperparameter [58, p. 72]. The selection of parameters, and the ranges of values used in the grid search, vary widely between different classification methods, and will be discussed in greater detail later in this section.

k-fold cross-validation is a method of reducing bias when evaluating the AUC score of a classifier. It is performed as follows: The data is divided into *k* sets of equal size. *k*-1 of these will form the training set that will be used to train the machine learning model (for more information, see 2.1), while the remaining one is selected as the test set that will be used to evaluate the classifier's AUC score. The process is repeated *k* times, each time

with a different set as the test set, and the rest as the training set. The classifier's final AUC score is calculated as the average of the AUC scores of each evaluation round. [4, p. 708]

I perform the validation (tuning of classifier parameters) and testing (training and evaluation of the classifier) using 5-fold cross-validation with 200 and 1000 manually labeled and randomly chosen samples, respectively. For the self-training, I additionally use the remaining 7004 unlabeled emails.

4.3.1 Random forest experiments

Some interesting parameters for random forest are `n_estimators` and `criterion`, since they control the number of decision trees and the criterion for measuring the importance of a feature, respectively. For `n_estimators`, values 10^0, 10^1, …, 10^4 were tried; for `criterion`, the possible values are `'gini'` and `'entropy'`. `'gini'` stands for Gini impurity, whereas `'entropy'` means using information gain (see 2.1.3 for more on the latter) [59].

The optimal value for `n_estimators` was 1000; for criterion, `'gini'`. In general, a larger number of trees (`n_estimators`) yields better performance [60].

For the AUC scores from the random forest tests, see Figure 14. The classifier was best at categorizing emails belonging to class CIM, with an AUC score of 0.974, while DB and Integration had AUC scores of 0.957 and 0.964, respectively. The average AUC score was 0.965.

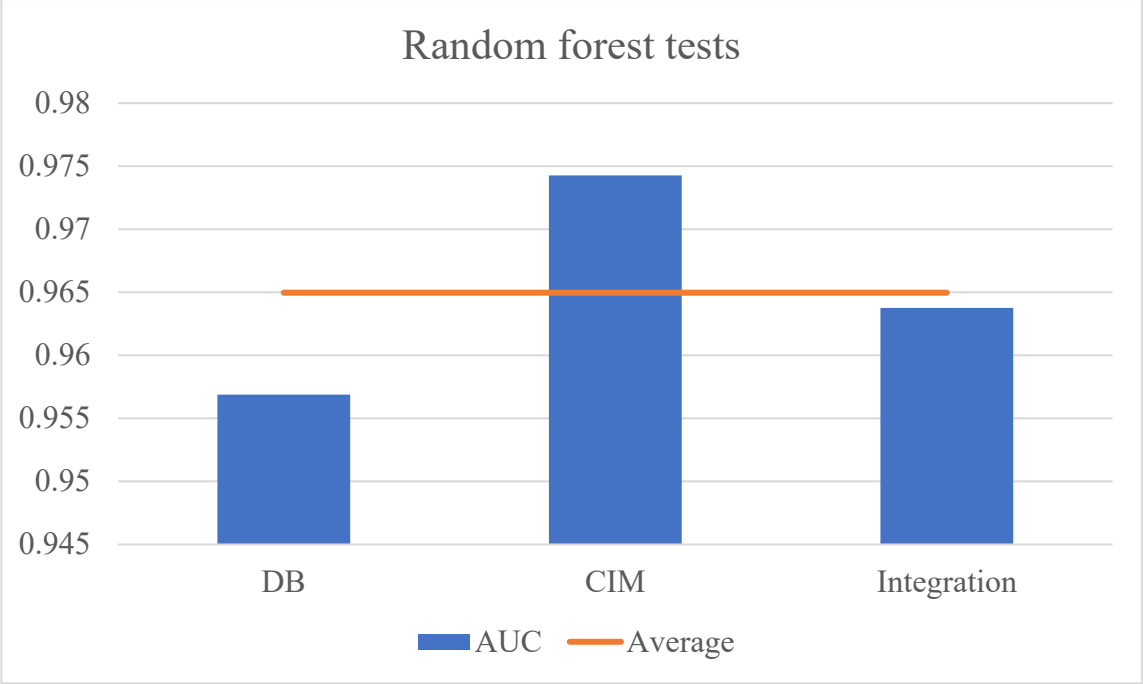

Figure 14: AUC scores for each class with random forest.

4.3.2 Naïve Bayes experiments

For Naïve Bayes, some important parameters are `alpha`, because it sets the amount of smoothing, and `fit_prior`, because it determines whether class prior probabilities are learned [59]. For `alpha`, 100 values between 10^{-10} and 10^{10} spaced evenly on a logarithmic scale were tried; `fit_prior` is either true or false [59]. The best performing values were `alpha` = 0.0004641588833612782 and `fit_prior` = true.

For the AUC scores from the naïve Bayes tests, see Figure 15. The classifier was best at categorizing emails belonging to class Integration, with an AUC score of 0.98, while the AUC scores for DB and CIM were 0.97 and 0.96, respectively. The average AUC score was 0.97.

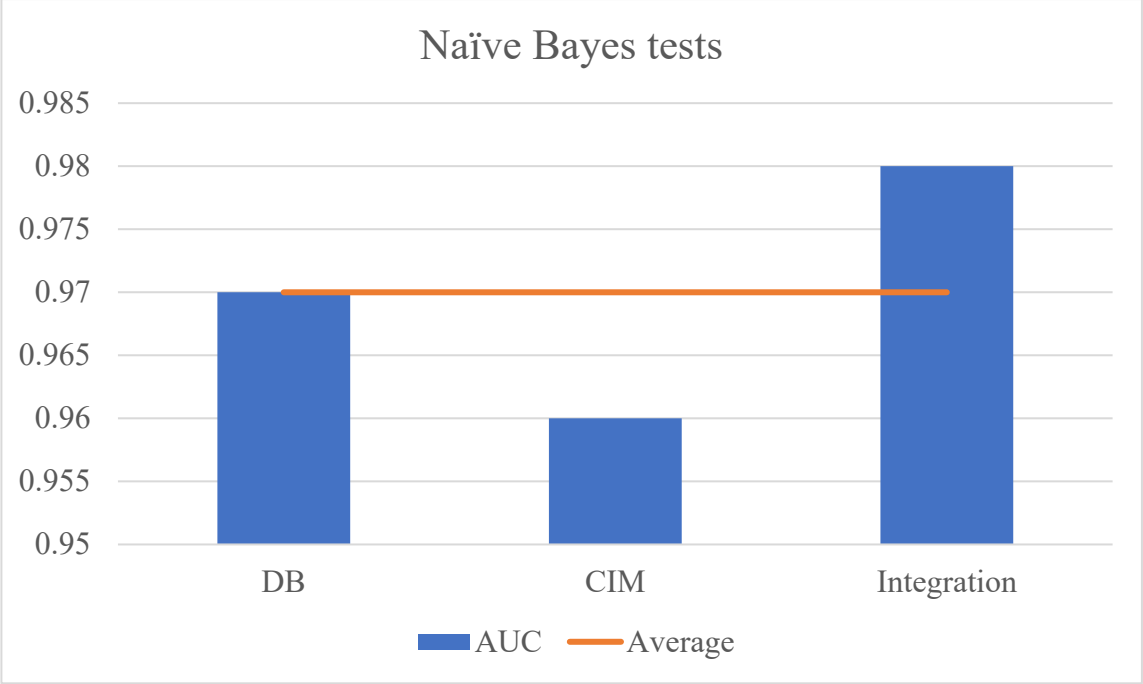

Figure 15: AUC scores for each class with naïve Bayes.

4.3.3 Support-vector machine experiments

Some interesting parameters for the support-vector machine are `C`, because it is the penalty parameter discussed in 2.1.7; `tol`, because it sets the stopping criteria tolerance; and `max_iter`, because it determines the maximum number of iterations [59].

For `C`, values 10^{-1}, 10^{0}, …, 10^{3} were tried; for `tol`, 10^{-1}, 10^{-2}, …, 10^{-5}. The best-performing parameter values were `C` = 10^{3} and `tol` = 0.01. A value of 10 000 for `max_iter` was enough for convergence.

For the AUC scores from the support-vector machine tests, see Figure 16. The classifier was best at categorizing emails belonging to class DB, with an AUC score of 0.984, while the AUC scores for CIM and Integration were 0.9823 and 0.9824, respectively. The average AUC score was 0.9829.

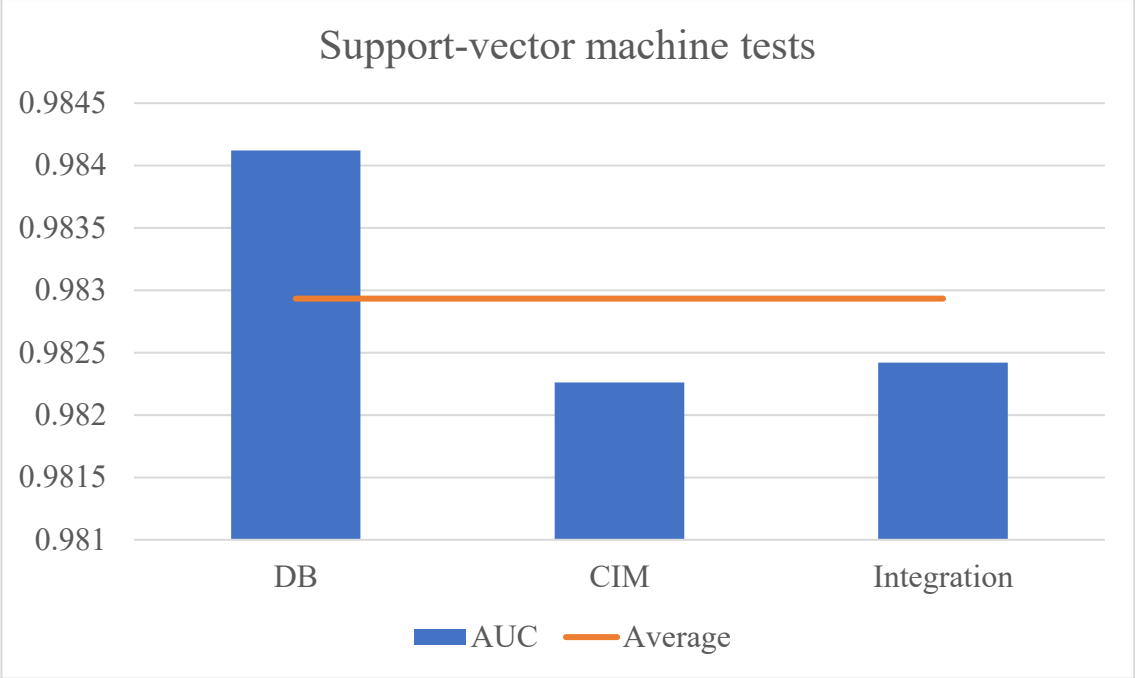

Figure 16: AUC scores for each class with the support-vector machine.

4.3.4 AdaBoost experiments

A significant parameter for AdaBoost is `n_estimators`, because it determines the number of decision stumps used [59]. For `n_estimators`, values 10^1, 10^2, …, 10^5 were tried. Of these, 10^5 performed best.

For the AUC scores from the AdaBoost tests, see Figure 17. The classifier was best at categorizing emails belonging to class CIM, with an AUC score of 0.974, while the AUC scores for DB and Integration were 0.970 and 0.961, respectively. The average AUC score was 0.968.

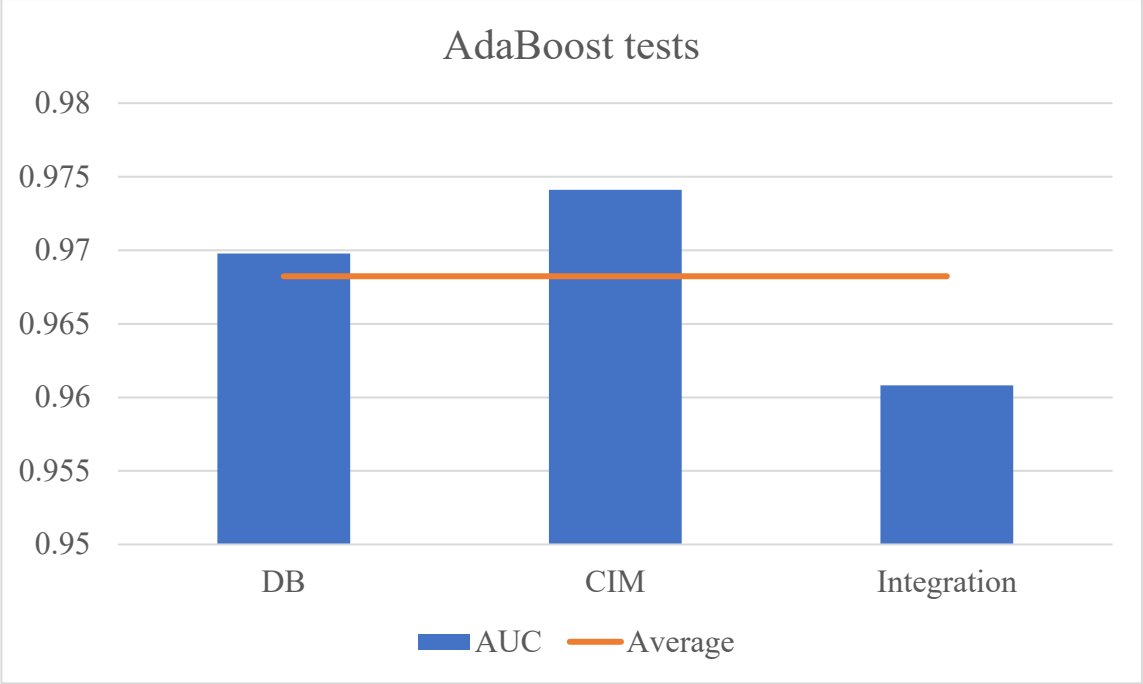

Figure 17: AUC scores for each class with AdaBoost.

4.3.5 Self-training experiments

For self-training, I implemented the algorithm described by [36], using Python 3.7.1; for a summary of the algorithm, see 2.1.9. The parameters to be tuned are `confidence_threshold` and `max_iter`. The former determines which samples will be included in the next iteration of training; the latter determines the maximum number of iterations before giving up. For `confidence_threshold`, values 0, 0.1, …, 0.9 were tried; of these, 0 performed best. For `max_iter`, 20 iterations were enough to ensure convergence.

For the AUC scores from the self-training tests, see Figure 18. The classifier was best at categorizing emails belonging to class Integration, with an AUC score of 0.95, while DB and CIM had AUC scores of 0.94 and 0.93, respectively. The average AUC score was 0.94.

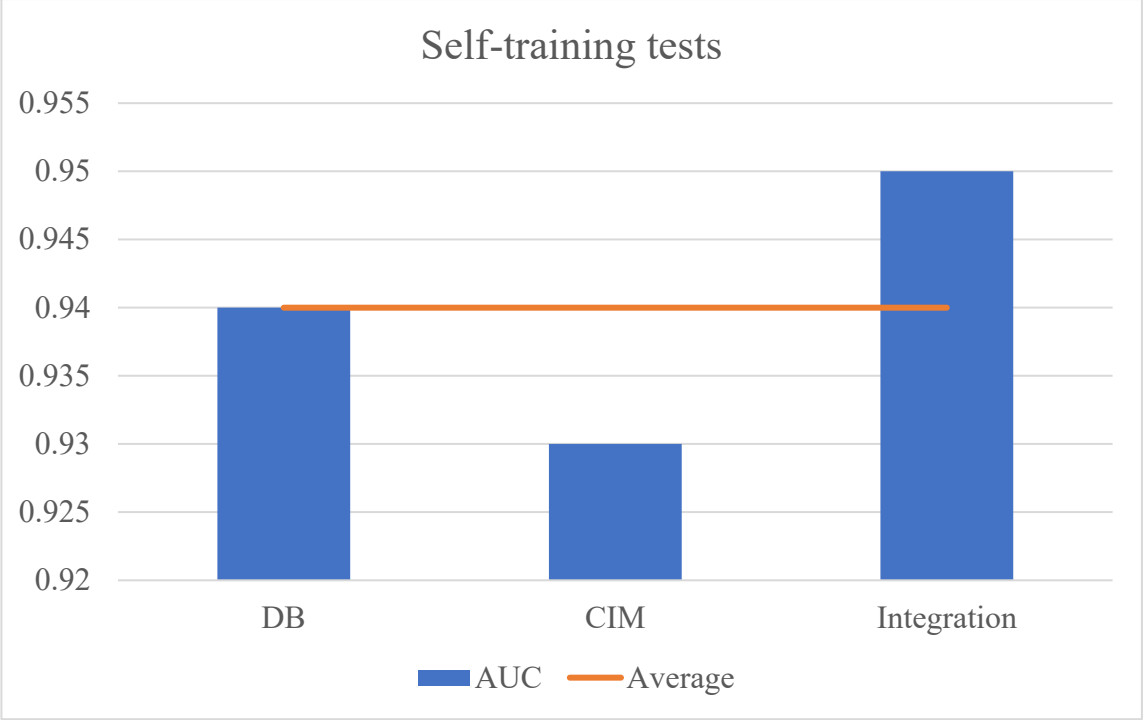

Figure 18: AUC scores for each class with self-training.

5 Results

This chapter presents a summary of the results of the above experiments, including the training and validation times of each classifier, and each classifier's AUC scores grouped by class.

5.1 Training and validation time

See Figures 19 and 20 for the classifier training and validation times, respectively; due to large variation in times, a logarithmic scale is used. A higher bar means the training or validation took a longer time. To reiterate, training means fitting the classifier to a training dataset, while validation means the tuning of classifier parameters.

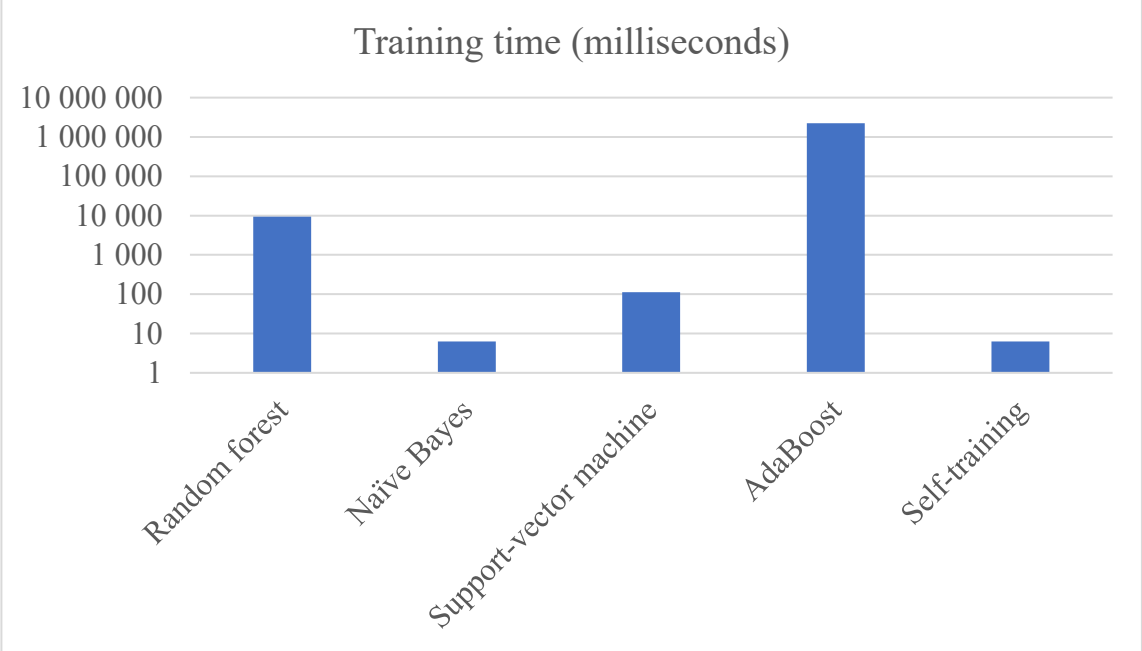

Figure 19: Model training time in milliseconds (logarithmic scale).

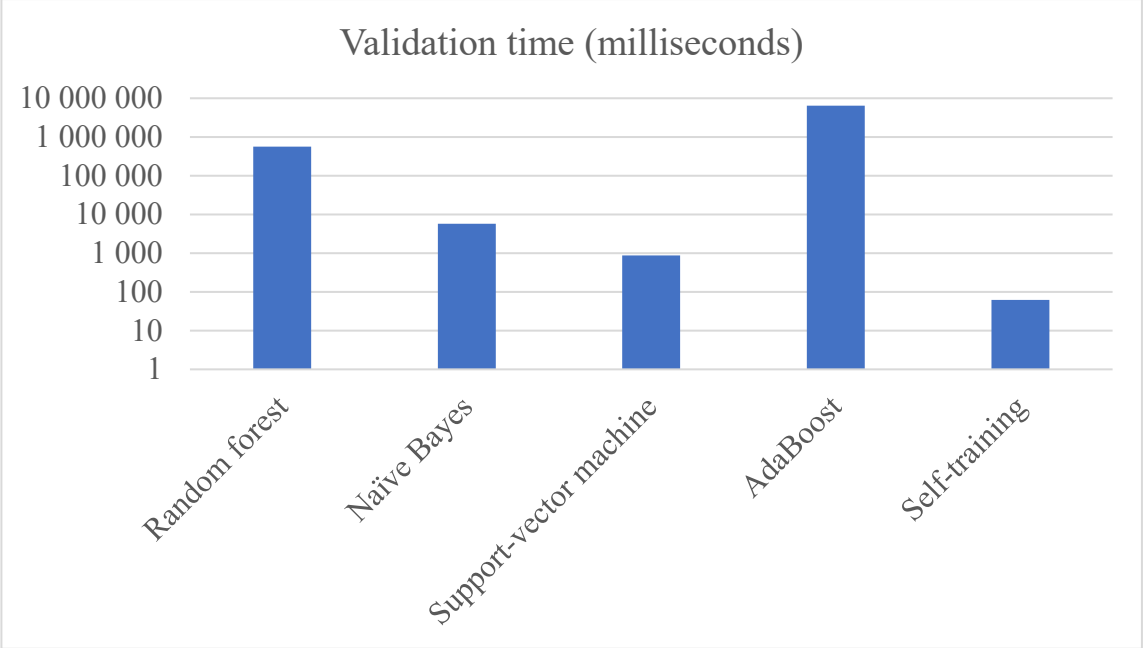

Figure 20: Classifier validation time in milliseconds (logarithmic scale).

5.2 AUC

See Figures 21-23 for the AUC scores for classes DB, CIM, and Integration, respectively. A higher bar indicates a higher AUC score, which means a classifier categorized emails belonging to the class better during testing. For the AUC scores averaged over all three classes, see Figure 24.

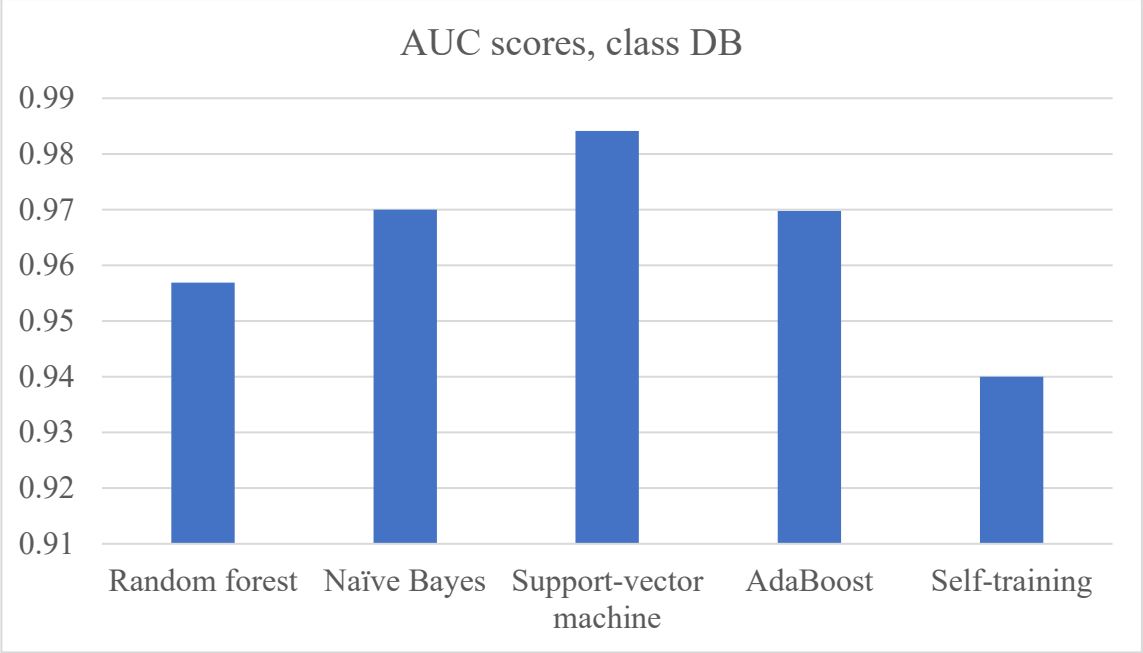

Figure 21: AUC scores for class DB.

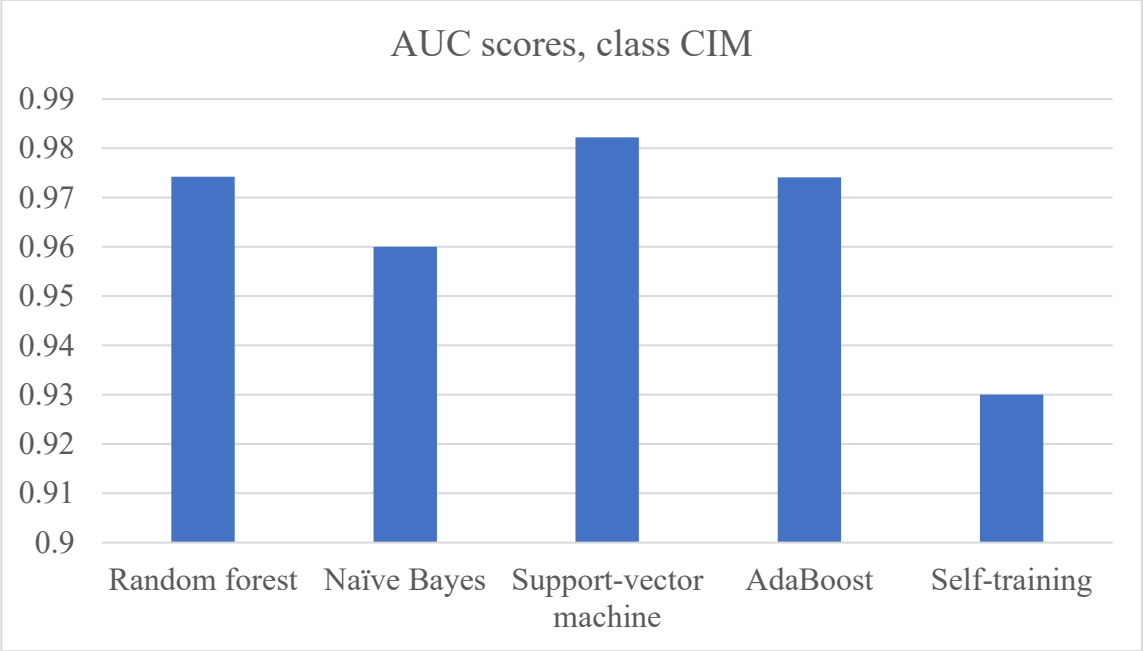

Figure 22: AUC scores for class CIM.

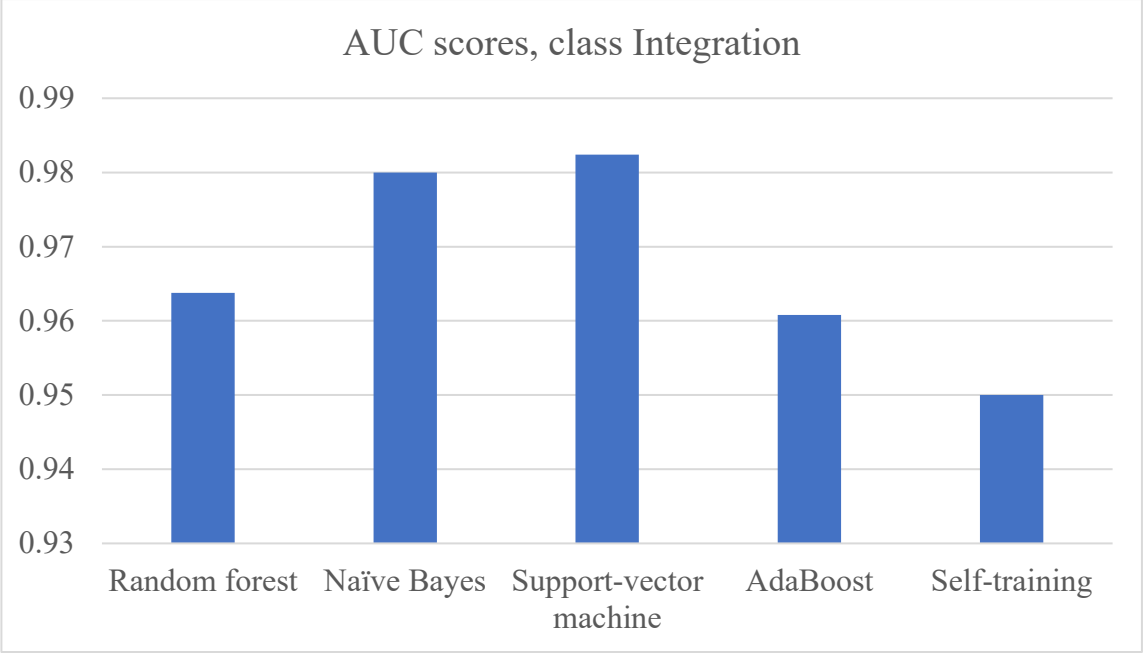

Figure 23: AUC scores for class Integration.

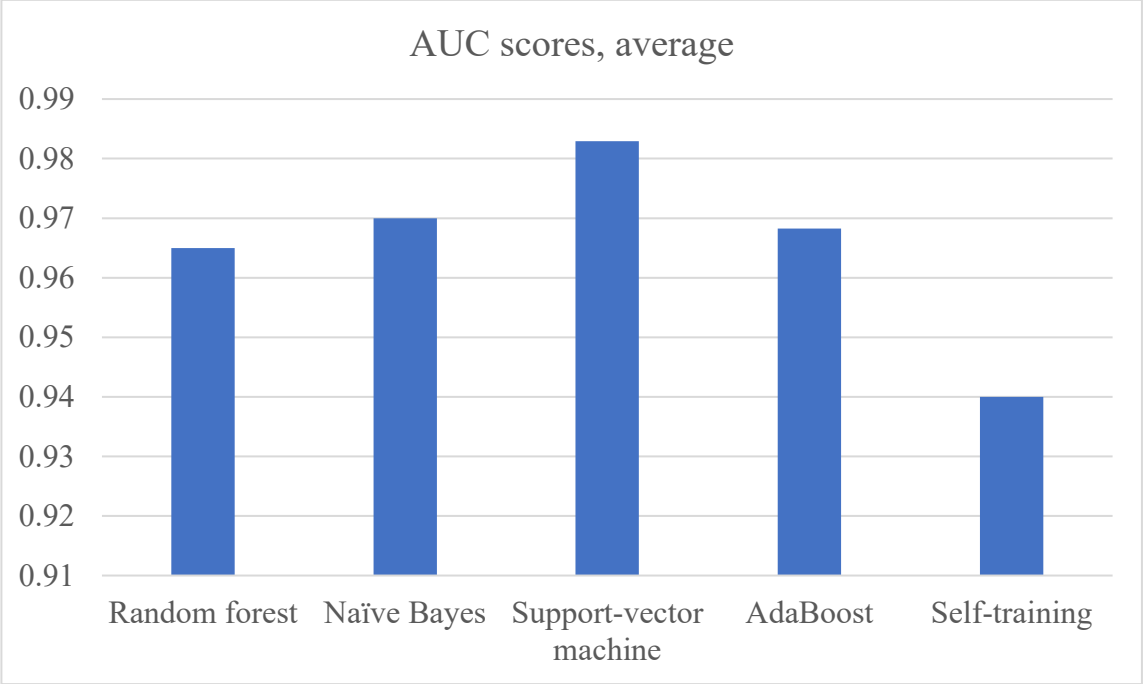

Figure 24: AUC scores averaged over all three classes.

6 Conclusions

Of the methods used, the support-vector machine was the best-performing classifier for all three classes, with AUC scores of 0.98412, 0.98226, and 0.98242 for classes DB, CIM, and Integration, respectively, and an average AUC score of 0.9829. This is better than in some comparable studies of email classification, such as the one by Cheng and Li [52], where they achieved an average AUC score of 0.933; however, there are notable differences from this study, e.g. they use email classification for spam filtering (see Chapter 3 for more information).

Support-vector machines perform well in text classification tasks in general, partly due to the high dimensionality of text; as such, it stands to reason that it performed well in this case [27]. Conversely, self-training did not perform better than the supervised methods, despite having done so in previous studies of text classification with limited labeled data [61].

AdaBoost had the longest training and validation times at 37 and 108 minutes, respectively; naïve Bayes had the shortest training time at 6.25 milliseconds, and self-training had the shortest validation time at 63 milliseconds. Both ensemble learning methods (AdaBoost and random forest) were the slowest in both training and validation.

Since training time is not the most important consideration for our use case, the support-vector machine is the optimal choice, due to its high AUC score and relatively short validation time.

7 Future research

Since self-training did not yield better performance in this case, yet it has been demonstrated to do so in others [61], it would be interesting to investigate the cause of this discrepancy. Could it be due to not having enough unlabeled data points, or because the type of data is different? The dataset used here is quite specific, so it might be useful to experiment with other email sets, other text sets, or non-text datasets.

I have only experimented with a subset of all machine learning algorithms for this task, so testing other methods for email classification could also provide valuable information, for instance advanced neural networks and representation learning approaches, or other semi-supervised methods. It could also be worthwhile to investigate various automated machine learning approaches.

I have only used common English words as stop words in the TF-IDF vectorization. In theory, better performance might be achieved by considering which words the classifiers use to determine class membership, and then filter out the words that are counterproductive.

Since I did not use any kind of feature reduction, it might be interesting to evaluate whether feature reduction would improve performance, especially for methods that are sensitive to the number of features, such as *k*-nearest neighbors [4, p. 739].

It will be necessary to plan the next steps in implementing the models in practical use. Considerations include whether to retrain the model using newer emails as they arrive, and how this could best be accomplished.

Summary in Swedish

Klassificering av e-brev med begränsad mängd förklassificerade data

Målet med denna avhandling är att utvärdera olika sätt att klassificera e-brev då man inte har tillgång till stora mängder förklassificerade data. Kontexten är att en grupp i ett företag har en gemensam brevlåda, från vilken varje e-brev ska vidarebefordras till den gruppmedlemmen som är mest lämpad för att behandla e-brevet. Denna process är för tillfället manuell; gruppmedlemmarna övervakar brevlådan i tur och ordning.

Flera av e-breven består av automatiska meddelanden från ett ärendehanteringssystem, varningar om potentiella fel i processer, och så vidare. Dessutom finns det interna e-brev från anställda i företaget, e-brev från företagets samarbetspartners, etc.

Trots att en stor mängd gamla e-brev från brevlådan är tillgängliga, så finns det ingen information om vilken gruppmedlem som behandlat ett e-brev. Om maskininlärning ska användas för att klassificera inkommande e-brev, så är denna brist på förklassificerade data problematisk. Det enda sättet att få sanningsenliga klasser för dessa e-brev är att manuellt gå igenom vart och ett e-brev och klassificera det, vilket för stora mängder e-brev (till exempel mer än 1000) kan vara oöverkomligt tidskrävande.

I studier där maskininlärning används för klassificering av e-brev, är mängden tillgängliga förklassificerade data vanligen stor. Till exempel använder Klimt och Yang 200 399 e-brev i Enron-datamängden; utdatan i deras studie är mappen som ett e-brev hör till [44]. Trots det har studier utförts kring textklassificering över lag (det vill säga, inte endast e-brev) med mindre mängder förklassificerade data, nämligen inom delvis övervakad maskininlärning. Ett exempel är Nigam et al., som i sin studie använder 300 förklassificerade textdokument för att träna en delvis övervakad maskininlärningsmodell [34].

Enligt Weik är maskininlärning en anordnings (till exempel en dators) förmåga att förbättra sin prestationsförmåga utgående från resultaten av sin tidigare prestation [1, s.

541]. De tre kategorier som är relevanta för denna avhandling är övervakad, oövervakad och delvis övervakad maskininlärning. Övervakad maskininlärning innebär tränandet av en maskininlärningsmodell med hjälp av endast förklassificerade data, medan oövervakad maskininlärning betyder upptäckandet av mönster i data utan användning av förklassificerade data. Delvis övervakad maskininlärning är, som namnet antyder, en kombination av övervakad och oövervakad maskininlärning: den använder sig av både förklassificerade och icke-förklassificerade data för att träna maskininlärningsmodellen. [5, ss. 1-2]

Forskningsfrågan som jag ämnar att besvara i denna avhandling är ”vad är det bästa sättet att klassificera e-brev då antalet förklassificerade e-brev är högst 1000”? Jag experimenterar med övervakade och delvis övervakade algoritmer. Oövervakade metoder såsom *k*-means verkar inte lämpliga, eftersom jag använder mig av fördefinierade kategorier. Anledningen till att inkludera delvis övervakade algoritmer är att de lämpas för fall där mängden förklassificerade data är begränsad. Mer specifikt så används den delvis övervakade metoden som kallas för självträning. De övervakade metoderna som används är Random Forest, Naïve Bayes, Support-Vector Machine och AdaBoost.

Experimenten utförs med hjälp av Python, specifikt modulen scikit-learn. Scikit-learn, som först utgavs 2007, är ett programvarubibliotek av öppen källkod för maskininlärning i Python [55]. Scikit-learn använder sig av SciPy, en Python-modul för matematik, som använder Python-modulen NumPy för flerdimensionella tabeller [57].

Validering och testning utförs m.h.a. 5-delad korsvalidering, med 200 respektive 1000 manuellt klassificerade och slumpmässigt valda e-brev. För självträningen använder jag dessutom 7004 oklassificerade e-brev.

E-breven indelas i tre klasser, av vilka var och en representerar en grupp e-brevsmottagare, där gruppen är den som är mest lämpad för att behandla e-brevet.

Datan exporteras från Microsoft Office Outlook 365 till en CSV-fil. Totalt finns det 8204 rader (e-brev). Med hjälp av ett Python-skript raderas de oanvända kolumnerna, så att endast meddelandets rubrik och brödtext konkatenerade till en enda kolumn kvarstår. Sedan används ett skript för att slumpmässigt välja ut 1200 e-brev för att klassificeras manuellt, medan resten sparas i en ny fil. Sedan klassificerar jag manuellt de 1200 e-

breven, av vilka 1000 slumpmässigt väljs som testdata, och de resterande 200 används för validering.

Av de använda metoderna är Support-Vector Machine den noggrannaste klassificeraren för alla tre klasser, med genomsnittlig AUC på 0,9829. Support-Vector Machine presterar bra i textklassificering över lag, delvis på grund av att text har hög dimensionalitet [27].

AdaBoost har de längsta valideringsoch träningstiderna, 108 respektive 37 minuter; självträning har den kortaste valideringstiden, 63 millisekunder, medan Naïve Bayes har den kortaste träningstiden, 6,25 millisekunder.

Det vore intressant att undersöka varför självträning inte förbättrade klassificeringen i detta fall, trots att den visats göra det i motsvarande fall [61]. Kan detta bero på att mängden oklassificerade data som var tillgänglig inte var tillräcklig, eller att datan var annorlunda? Datamängden som använts här är rätt specifik, så det kunde vara gynnsamt att experimentera med andra slags e-brev, texter som inte är e-brev, eller annan data än text.

Mer arbete behövs för att kunna tillämpa avhandlingens resultat i praktiken. Bland annat måste man överväga om maskininlärningsmodellerna borde tränas om i takt med att nya e-brev tas emot, och hur detta i så fall bäst kunde åstadkommas.

8 Bibliography

[1] M. H. Weik, *Computer Science and Communications Dictionary,* Boston: Springer, 2001.

[2] D. Nandi, A. F. M. S. Saif, P. Paul, K. M. Zubair and S. A. Shubho, "Traffic Sign Detection based on Color Segmentation of Obscure ImageCandidates: A Comprehensive Study," *International Journal of Modern Education and Computer Science,* vol. 10, no. 6, pp. 35-46, 2018.

[3] S. Wang, W. Chaovalitwongse and R. Babuška, "Machine Learning Algorithms in Bipedal Robot Control," *IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews),* vol. 42, no. 5, pp. 728-743, 2012.

[4] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach (3rd Edition),* New Jersey: Pearson Education Limited, 2009.

[5] O. Chapelle, B. Schölkopf and A. Zien, *Semi-supervised learning,* Cambridge: MIT Press, 2006.

[6] R. Mohanasundaram, A. S. Malhotra, R. Arun and P. S. Periasamy, "Deep Learning and Semi-Supervised and Transfer Learning Algorithms for Medical Imaging," in *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, St. Louis, Academic Press, 2019, pp. 139-151.

[7] E. J. Borowski and J. M. Borwein, *Collins dictionary of Mathematics,* London: Collins, 2007.

[8] S. M. Stigler, "Gauss and the invention of least squares," *The Annals of Statistics,* vol. 9, no. 3, pp. 465-474, 1981.

[9] P. Kontkanen, J. Lehtonen, K. Luosto and H. Westermark, *Ellips 6: Sannolikhet och statistik,* Vasa: Ykkös-Offset, 2007.

[10] R. J. Freund, W. J. Wilson and P. Sa, *Regression analysis,* Burlington: Elsevier Science & Technology, 2006.

[11] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics,* vol. 5, no. 4, pp. 115-137, 1943.

[12] F. Rosenblatt, *Principles of neurodynamics. perceptrons and the theory of brain mechanisms,* Buffalo: Cornell Aeronautical Lab Inc, 1961.

[13] D. K. Renuka, T. Hamsapriya, M. R. Chakkaravarthi and P. L. Surya, "Spam classification based on supervised learning using machine learning techniques," in *2011 International Conference on Process Automation, Control and Computing*, Coimbatore, 2011.

[14] H. Peng, S. Thomson and N. A. Smith, "Deep multitask learning for semantic dependency parsing," in *ACL*, Florence, 2017.

[15] A. Liaw and M. Wiener, "Classification and regression by randomForest," *R news,* vol. 2, no. 3, pp. 18-22, 2002.

[16] B. Xu, X. Guo, Y. Ye and J. Cheng, "An Improved Random Forest Classifier for Text Categorization," *JCP,* vol. 7, no. 12, pp. 2913-2920, 2012.

[17] A. Verikas, E. Vaiciukynas, A. Gelzinis, J. M. Parker and M. C. Olsson, "Electromyographic Patterns during Golf Swing: Activation Sequence Profiling and Prediction of Shot Effectiveness," *Sensors,* vol. 16, no. 5, p. 592, 2016.

[18] J. Zhu, H. Zou, S. Rosset and T. Hastie, "Multi-class AdaBoost," *Statistics and its Interface,* vol. 2, no. 3, pp. 349-360, 2009.

[19] P. Nardiello, F. Sebastiani and A. Sperduti, "Discretizing continuous attributes in AdaBoost for text categorization," in *European Conference on Information Retrieval*, Pisa, 2003.

[20] G. Rätsch, T. Onoda and K.-R. Müller, "Soft margins for AdaBoost," *Machine learning,* vol. 42, no. 3, pp. 287-320, 2001.

[21] T. Bayes, "An essay towards solving a problem in the doctrine of chances," *Philosophical Transactions of the Royal Society of London,* vol. 53, pp. 370-418, 1763.

[22] I. Rish, "An empirical study of the naive Bayes classifier," *IJCAI 2001 workshop on empirical methods in artificial intelligence,* vol. 3, no. 22, pp. 41-46, 2001.

[23] J. D. Rennie, L. T. J. Shih and D. R. Karger, "Tackling the poor assumptions of naive bayes text classifiers," *In Proceedings of the 20th international conference on machine learning,* pp. 616-623, 2003.

[24] M. Parsian, *Data Algorithms,* Sebastopol: O'Reilly, 2015.

[25] E. Fix and J. Hodges, "Discriminatory analysis, nonparametric discrimination: Consistency properties," University of California, Berkeley, 1951.

[26] J. M. Keller, M. R. Gray and J. A. Givens, "A fuzzy k-nearest neighbor algorithm," *IEEE transactions on systems, man, and cybernetics,* vol. 4, pp. 580-585, 1985.

[27] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *European conference on machine learning*, Chemnitz, 1998.

[28] B. Catanzaro, N. Sundaram and K. Keutzer, "Fast support vector machine training and classification on graphics processors," in *Proceedings of the 25th international conference on Machine learning*, Helsinki, 2008.

[29] J. Levman, "The support vector machine in medical imaging," in *Support Vector Machines: Data Analysis, Machine Learning and Applications*, New York, Nova Science Publishers, 2011, pp. 1-25.

[30] B.-S. Yang and A. Widodo, "Pattern recognition for machine fault diagnosis using support vector machine," in *Support Vector Machines: Data Analysis, Machine*

Learning and Applications, New York, Nova Science Publishers, 2011, pp. 153-195.

[31] R. R. Sokal and P. H. Sneath, *Principles of Numerical Taxonomy,* San Francisco: W.H. Freeman, 1963.

[32] M. S. Aldenderfer and R. K. Blashfield, *Cluster Analysis,* Thousand Oaks: SAGE Publications, Inc., 1984.

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

[34] K. Nigam, A. McCallum, S. Thrun and T. Mitchell, "Learning to classify text from labeled and unlabeled documents," *Machine Learning,* vol. 39, no. 2/3, pp. 103-134, 1998.

[35] H. J. Scudder, "Probability of Error of Some Adaptive Pattern-Recognition Machines," *IEEE Transactions on Information Theory,* vol. 11, no. 3, pp. 363-371, 1965.

[36] M. I. Sameen, B. Pradhan and S. Lee, "Self-learning random forests model for mapping groundwater yield in data-scarce areas," *Natural Resources Research,* vol. 28, no. 3, pp. 757-775, 2019.

[37] O. Pietquin, M. Geist, S. Chandramohan and H. Frezza-Buet, "Sample-Efficient Batch Reinforcement Learning for Dialogue Management Optimization," *ACM Transactions on Speech and Language Processing,* vol. 7, no. 3, pp. 1-21, 2011.

[38] W. Curran, T. Brys, D. Aha, M. Taylor and W. D. Smart, "Dimensionality Reduced Reinforcement Learning for Assistive Robots," in *AAAI Fall Symposium Series*, Arlington, 2016.

[39] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction,* Cambridge: MIT Press, 1998.

[40] J. Ramos, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, Piscataway, 2003.

[41] J. Leskovec, A. Rajaraman and J. D. Ullman, *Mining of massive datasets,* Cambridge: Cambridge University Press, 2014.

[42] C. D. Manning, P. Raghavan and H. Schütze, *An Introduction to Information Retrieval,* Cambridge: Cambridge University Press, 2009.

[43] T. Fawcett, "An introduction to ROC analysis," *Pattern recognition letters,* vol. 27, no. 8, pp. 861-874, 2006.

[44] B. Klimt and Y. Yang, "The Enron Corpus: A New Dataset for Email Classification Research," in *European Conference on Machine Learning*, Pisa, 2004.

[45] E. M. Bahgat, S. Rady, W. Gad and I. F. Moawad, "Efficient email classification approach based on semantic methods," *Ain Shams Engineering Journal,* vol. 9, no. 4, pp. 3259-3269, 2018.

[46] S. R. Gomes, S. G. Saroar, M. Mosfaiul, A. Telot, B. N. Khan, A. Chakrabarty and M. Mostakim, "A comparative approach to email classification using Naive Bayes classifier and hidden Markov model," in *2017 4th International Conference on Advances in Electrical Engineering*, Hong Kong, 2017.

[47] M. E. Yitagesu and M. Tijare, "Email classification using classification method," *International Journal of Engineering Trends and Technology,* vol. 32, no. 3, pp. 142-145, 2016.

[48] S. G. Phadke, "Email Classification Using a Self-Learning Technique Based on User Preferences," North Dakota State University of Agriculture and Applied Science, 5 November 2015. [Online]. Available: https://library.ndsu.edu/ir/bitstream/handle/10365/25492/Email%20Classification%20Using%20a%20Self-

Learning%20Technique%20Based%20on%20User%20Preferences.pdf?sequence=1. [Accessed 13 June 2020].

[49] S. Kiritchenko, S. Matwin and S. Abu-Hakima, "Email classification with temporal features," in *Intelligent Information Processing and Web Mining*, Zakopane, 2004.

[50] G. V. Cormack, *Email Spam Filtering: A Systematic Review,* Waterloo: Now Publishers Inc., 2008.

[51] T. Ayodele, R. Khusainov and D. Ndzi, "Email classification and summarization: A machine learning approach," in *2007 IET Conference on Wireless, Mobile and Sensor Networks*, Shanghai, 2007.

[52] V. Cheng and C. H. Li, "Personalized Spam Filtering with Semi-supervised Classifier Ensemble," in *2006 IEEE/WIC/ACM International Conference on Web Intelligence*, Hong Kong, 2006.

[53] G. Mujtaba, L. Shuib, R. G. Raj, N. Majeed and M. A. Al-Garadi, "Email Classification Research Trends: Review and Open Issues," *IEEE Access,* vol. 5, pp. 9044-9064, 2017.

[54] P. Sur and E. J. Candès, "A modern maximum-likelihood theory for high-dimensional logistic regression," *Proceedings of the National Academy of Sciences,* vol. 116, no. 29, pp. 14516-14525, 2019.

[55] G. Hackeling, *Mastering machine learning with scikit-learn,* Birmingham: Packt Publishing Ltd., 2014.

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, ... and J. Vanderplas, "Scikit-learn: Machine learning in Python," *Journal of machine learning research,* vol. 12, no. Oct, pp. 2825-2830, 2011.

[57] S. J. Rojas G., E. A. Christensen and F. J. Blanco-Silva, *Learning SciPy for Numerical and Scientific Computing,* Birmingham: Packt Publishing Ltd., 2015.

[58] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow,* Sebastopol: O'Reilly Media, Inc., 2017.

[59] scikit-learn developers, "scikit-learn user guide," 29 July 2019. [Online]. Available: https://scikit-learn.org/0.21//_downloads/scikit-learn-docs.pdf. [Accessed 7 March 2020].

[60] T. Shi and S. Horvath, "Unsupervised learning with random forest predictors," *Journal of Computational and Graphical Statistics,* vol. 15, no. 1, pp. 118-138, 2006.

[61] M. Pavlinek and V. Podgorelec, "Text classification method based on self-training and LDA topic models," *Expert Systems with Applications,* vol. 80, pp. 83-93, 2017.