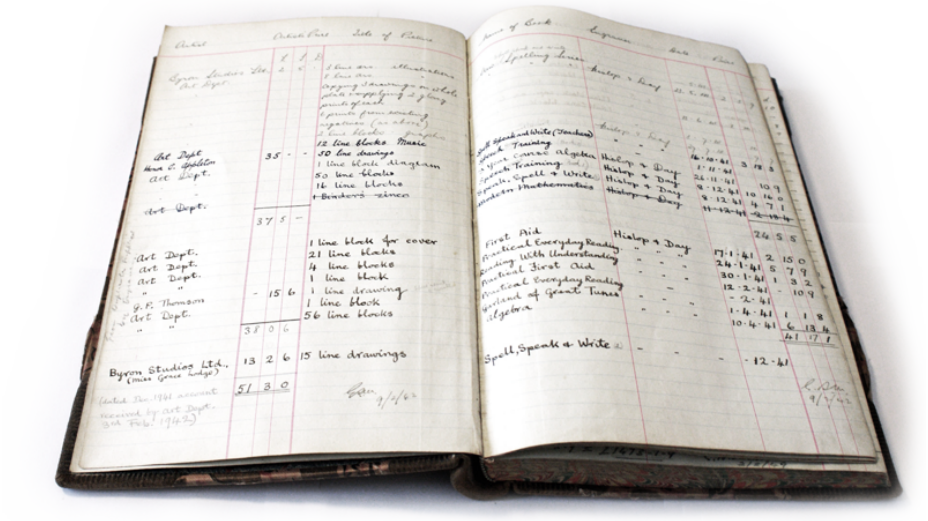
CHALMERS



Using Classification Algorithms for Smart Suggestions in Accounting Systems

Master of Science Thesis in
Computer Science: Algorithms, Languages and Logic

HAMPUS BENGTSSON
JOHANNES JANSSON

Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden 2015

Using Classification Algorithms for Smart Suggestions in Accounting Systems
Hampus Bengtsson & Johannes Jansson

Examiner: Graham Kemp
Supervisor: Olof Mogren

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Cover art: A ledger detailing all work commissioned externally by Holmes McDougall from 1941 to 1983. Adapted from picture by Flickr user Edinburgh City of Print, `https://www.flickr.com/photos/30239838@N04/4268190563`, licenced under CC BY 2.0.

Department of Computer Science and Engineering
Gothenburg, Sweden. June 2015.

Abstract

Accounting is a repetitive task and is mainly done manually. The repetitiveness makes it a suitable target for automation, however not much research has been done in the area yet.

This thesis investigates how two different classification algorithms, Support Vector Machine with Stochastic Gradient Descent training and a Feed-Forward Neural Network, perform at classifying financial transactions based on historical data in an accounting context.

The classification algorithms show promising results but still does not outperform the existing implementation which is simple and deterministic. However, classification itself very much relies on the labels, i.e. how different users have accounted the transactions. As a response to this, we finally give a suggestion on how clustering might be used for the automation of accounting instead.

Keywords: machine learning, multiclass classification, accounting, online learning

Acknowledgements

We would like to thank our both supervisors: our academic supervisor, Olof Mogren, and our advisor at SpeedLedger, Per Brendelökken. Furthermore we want to thank everyone at SpeedLedger for a rewarding and exciting spring!

Hampus Bengtsson & Johannes Jansson, Göteborg 2015-06-04

Contents

1 Introduction

Accounting is still largely a manual labour, though for small companies and associations it is mainly monotonous work. The company SpeedLedger, whom this master thesis is performed in collaboration with, has a service that tries to minimise the manual labour in accounting to make it easier and more efficient.

Giving smart suggestions is an important part of automating the accounting process. Accounting in general and classification of transactions in particular is a mixture of monotonous work and making intelligent decisions. Even though this sounds like a fitting task for machine learning, not much research has been done in this area of accounting [1].

1.1 SpeedLedger

In the system of SpeedLedger, the procedure of classifying transactions works as follows: SpeedLedger has access to the organisations' bank records as well as accounting records[1] and gives a proposal on how to assign accounting codes to new transactions, which is accepted (or rejected, which is done by selecting a different accounting code[1]) by the user[2] as the last step of the process.

SpeedLedger currently has a naïve classifier. When a new transaction is retrieved from the bank, the accounting code of the latest transaction from the same organisation with the same recipient is suggested. If no recipient is found, the accounting code of the latest transaction with identical text is suggested. In the case where no such transaction exists either, no suggestion is made. The focus of this thesis is to evaluate how

[1] For an explaination of accounting terms, see Appendix A.

[2] The terms *user* and *organisation* are largely synonymous. When a distinction is made, the *user* preforms the interaction with the system and the *organisation* owns the actual accounting.

(and which) classification methods can be used to improve the suggestions given and how they perform.

Because of the large amount of data, a requirement for the classifier used is that it can handle incremental learning well as it would be infeasible to keep all of the training data in memory. Incremental learning, in contrast to batch learning, is when a classifier can learn continually from a stream of samples instead of learning on a group of samples at once [2].

1.2 Aim

The aim of the thesis is to compare the performance of different classification algorithms and the naïve approach in the setting of classifying financial transaction with regard to how they should be accounted. This includes measurements of the accuracy of implemented algorithms.

1.3 Problem formulation

The problem is formulated as follows: given the data points, possibly deliver a suggestion. Based on the suggestions given, the goal is to maximise the *quality* and *overall precision* (the terms are described in chapter 5).

To reach our aims, several tasks must be completed. To some extent, the latter tasks build on the earlier.

1.3.1 Collecting attributes

Finding relevant attributes is obviously imperative for succeeding in classification. To begin with, there are attributes straight from the database which are useful: transaction date, transaction amount, transaction text, transaction recipient etc. However, with some preprocessing the possibilities increase, especially when it comes to gathering information about the organisation: relation to the other party, what the line of business is, yearly spending per accounting code, how often each accounting code is used, turnover etc.

1.3.2 Choosing appropriate classification methods

A lot of different classification methods with different properties exist [3][4][5]. They all come with a set of advantages and drawbacks, which make them suitable in different situations – of course with some overlap. Many algorithms were evaluated initially; two were found that fulfilled the requirement of handling incremental learning, and are described in this thesis.

1.3.3 Measuring accuracy

To compare different classification algorithms fairly, one must consider several different aspects of measuring accuracy.

The first aspect is how to make the measurements. There are several possible ways of measuring the accuracy of a classifier. The theory behind the methods for evaluation of classifiers used in this thesis are described in section 2.3.

The next aspect is what to measure. There are several use cases to consider: scores for existing organisations as well as new ones, how the accuracy differs when using some of the data available but not other data et cetera.

Furthermore, there is the aspect of taking into account the possible sources of errors: the existing naïve implementation has an advantage in users possibly accepting suboptimal proposals.

It may also be found valuable to evaluate the accuracy of *ranked suggestions*, where up to n suggestions are presented to the user in order of likelihood. Instead of just measuring the percentage of correct vs. incorrect answers, one could measure the percentage of the first, second, third or none of the suggestions being correct.

1.4 Limitations

In accounting many special cases exist that can complicate the process of automating tedious tasks. This section lists the limitations of what is included in the thesis.

- **BAS chart of accounts only.** Every organisation may choose to deviate from the most widespread chart of accounts (BAS, see Appendix A). We primarily target BAS users as we believe this reduces

a lot of complexity. Furthermore, a general solution is believed to give only a marginal improvement.

- **Single assignment of accounting code only.** We disregard cases where a user needs to assign several accounting codes to a single transaction. Without the accounting records, this is in general not possible even for a human.
- **No dynamic suggestions of VAT codes.** One could imagine cases where different VAT codes (see Appendix A) are desired for a single given accounting code; however this is not the main use case and is not taken into account. Indeed, in the current implementation of the system, VAT codes are statically linked to accounting codes.

1.5 Outline

Chapter 2 describes the theory behind the different parts of the classification process including preprocessing and evaluation. In chapter 3 the implementation phase is described, e.g. the libraries used and how the different steps of preprocessing was done. Chapter 4 shows the setup for the experimental results. Chapter 5 presents the experimental results of the classifiers used. Chapter 6 discusses the results and the work in the thesis and presents possible future improvements. In chapter 7 the thesis is concluded.

2 Theory

This chapter introduces the theory behind different techniques regarding preprocessing of data (for use in machine learning classification algorithms), the classification algorithms, and methods used for evaluation of the performance of classification.

2.1 Preprocessing

It is generally agreed that data preprocessing and preparation is one of the most important and difficult (and also most time-consuming) part of a project in data mining [6].

This section describes different techniques for preprocessing data to make it usable for machine learning classification algorithms.

2.1.1 Feature normalisation and boolean representation

In machine learning, feature normalisation is often applied as a preprocessing step in order to increase the precision [7].

Normalisation to standard score is done using

$$\hat{x} = \frac{x - \mu}{\sigma}, \tag{2.1}$$

where μ is the mean and σ is the standard deviation of the samples [8].

A boolean variable can be represented in multiple ways, $\{-1, 1\}$ and $\{0, 1\}$ are two of them. The method of normalisation to use depends on how booleans are represented, as many machine learning algorithms perform better with an equal distribution for all of the variables in the feature vector.

2.1.2 One-hot encoding

In classification of input data where the value of a certain feature can be any of K categories (categorical features), the feature is commonly represented with a vector of length K where each of the values in the vector can be either 0 or 1. Only the value to be encoded is marked with 1, leaving the others 0 [6][9]. If only a single variable was used and set to an index instead, it would be implicitly and incorrectly implied that there is a relationship between indices that are close to each other.

An example of this is having a feature representing a color where the value could be any of *Red*, *Green* or *Blue*. A possible representation for each of the colors can be seen in Table 2.1. If they were instead encoded as a single variable with the values 1, 2, 3, red and blue would be less similar than red and green.

Color	v_0	v_1	v_2
Red	1	0	0
Green	0	1	0
Blue	0	0	1

Table 2.1: *A one-hot representation for colors.*

This way of encoding categorical features is used in e.g. [10], where 32 different algorithms are compared on multiple datasets.

2.1.3 Bag of words

A convenient way to represent text in classification is the bag of words approach [11]. Each word is represented by a distinct entry in the feature vector to describe e.g. the number of occurrences of that specific word in the text to be classified. An example of the bag of words representation can be seen in Figure 2.1 where only the presence or absence of a word is shown in the example on the left, and the number of occurrences of a certain word is shown in the example to the right.

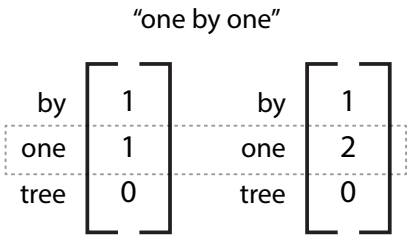


Figure 2.1: *Example of binary (left) and regular (right) bag of words representation of a string. Note the difference for the word "one", which occurs twice. The vector length is equal to the dictionary size.*

This encoding well represents the fact that the strings in the transaction and payment texts (described in section 4.1) are short – only a few words – and that most words are essentially labels describing a transaction. By definition, the main drawback is that the dictionary of possible words to represent must be known in advance (something that is not always the case in online learning). Another drawback is that the length of the feature vector grows linearly with the number of distinct words in text corpuses, and thus it may grow to infeasible sizes.

2.1.4 Feature hashing

A way to handle the problems with the bag of words representation is to utilise hashing of the attribute words [12][13]. In its basic form it works much like a hash table, where a modulo operation decides the appropriate entry for the hash.

The advantage of this representation is that any text can be represented with a fixed number of variables. The drawback is that some information is lost due to hash collisions, as in Figure 2.2 where a hashing function has calculated the hash value for the different strings and the hashing of the words *by* and *one* create a hash collision.

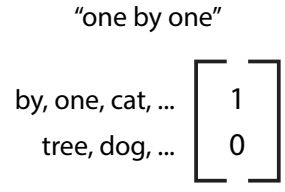


Figure 2.2: *An example of feature hashing. The vector is shorter than the dictionary size, so the words may clash.*

2.2 Classification methods

In machine learning, classification is a kind of supervised learning, where the machine learning algorithm is trained on n samples $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, where x_i is a vector containing the different features and y_i the class of the vector. The class is often defined as $y_i \in \{-1, 1\}$ for binary classification and $y_i \in \{1, ..., K\}$ for multiclass classification [9][11].

During classification, the algorithm is given a sample represented by a feature vector x and should return the class y of that sample based on what it has learnt from training.

In this section we describe the theory behind the classification algorithms used in this master thesis and why they are relevant to the thesis.

2.2.1 Linear classification using Stochastic Gradient Descent

In linear (binary) classification, such as when using Support Vector Machines (SVM) [14], samples can be classified using a so called decision function [15]:

$$f_w(x) = w^T \cdot x + b, \tag{2.2}$$

where w is the weight vector which decides how much each feature weighs in during classification, x is the sample to be classified, and b is a bias. In the following section it will be presented how the decision function can be optimised using stochastic gradient descent.

Gradient descent

In statistical learning theory, gradient descent is a method that can be used for minimising the empirical risk R_{emp} [16]:

$$R_{emp}(f_w) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_w(x_i), y_i) \tag{2.3}$$

Here, n is the number of samples and $\ell(\hat{y}, y)$ is the loss function, which is used to estimate the error of predicting the class $\hat{y}$ when the class

really was y. Some of the commonly used loss functions described in [17] are

$$\text{Hinge loss: } \ell(\hat{y}, y) = max(0, 1 - y\hat{y}) \tag{2.4a}$$

$$\text{Logistic loss: } \ell(\hat{y}, y) = ln(1 + exp(-y\hat{y})) \tag{2.4b}$$

$$\text{Modified huber loss: } \ell(\hat{y}, y) = \begin{cases} max(0, 1 - y\hat{y})^2 & \text{for } y\hat{y} \geq -1 \\ -4y\hat{y} & \text{otherwise} \end{cases} \tag{2.4c}$$

Using gradient descent the weight vector of the decision function is updated according to Equation 2.5 during training.

$$w_{t+1} = w_t - \eta_t \frac{1}{n} \sum_{i=1}^{n} \nabla_{w_t} \ell(f_{w_t}(x_i), y_i) \tag{2.5}$$

η_t is the learning rate which decides to what extent the weight vector should be affected by the gradient of the empirical risk.

Stochastic gradient descent

Stochastic gradient descent (SGD) is a simplification of the gradient descent where, instead of averaging the gradients of the loss function applied to multiple samples, a single sample (x_t, y_t) is selected randomly and the weight vector is updated [18]:

$$w_{t+1} = w_t - \eta_t \nabla_{w_t} \ell(f_{w_t}(x_t), y_t) \tag{2.6}$$

This method of approximating the empirical risk was suggested in the 1950s [19] and in later years it has been shown that this method performs well for large scale machine learning tasks [20].

The process of randomly selecting the next sample to use for updating the weight vector makes SGD appropriate for online learning; as the task of classifying transactions involves datasets containing millions of samples, SGD is a good candidate for handling the task at hand.

During training the weight vector is updated according to Equation 2.6 but often a regularization term $R(w)$ is added to penalise overly complex models (to avoid overfitting) [21]:

$$w_{t+1} = w_t - \eta_t(\nabla_{w_t}\ell(f_{w_t}(x_t), y_t) + \alpha\nabla_{w_t}R(w_t)), \tag{2.7}$$

where α is a positive hyperparameter. The bias b is updated in a similar manner.

Classification is done using the decision function (Equation 2.2) by looking at the sign of the result [15]. Multiclass classification can be done in multiple ways; e.g. One-versus-Rest classification, which is a way to reduce the multiclass problem into several binary class problems using one classifier per class. The class of a new sample is then predicted by using the class of the classifier scoring highest [15].

This classifier requires a weight matrix with the dimensions $C \times F$ where C is the number of classes and F the number of features.

In the rest of the thesis, this classifier will be denoted as SVM-SGD.

2.2.2 Feed forward neural networks

A feed forward neural network (FFNN) consists of several artificial neurons, constrained by how they are interconnected. The general information in this section is taken from [22]. There are several models of neurons, and the model we refer to in this section derives from the perceptron described by Rosenblatt in 1958 [23], which in turn derives from the 1943 McCulloch-Pitts neuron [24] (MPN). Models of artificial neurons resemble their biological counterparts in that they both have weighted inputs and perform a simple computation in order to form a single output signal.

An artificial neuron has n inputs $x_1 \dots x_n$, n weights $w_1 \dots w_n$, a bias b and one output y. The output is computed as

$$y = \phi\left(\sum_{i=1}^{n} x_i w_i - b\right),$$

where ϕ is the *activation function*. The bias may be implemented by adding

a fixed input $x_0 = 1$ and a weight $w_0 = b$. This gives the function

$$y = \phi \left(\sum_{i=0}^{n} x_i w_i \right), \tag{2.8}$$

making the implementation of bias more transparent.

The early models use the Heaviside step function as an activation function, and the MPN has a fixed threshold rather than a variable bias. MPN also has other constraints, such as discrete positive[1] weights. The perceptron was the first model with a learning rule; the weights and thresholds for the MPN were chosen manually.

Today, artificial neurons are used more flexibly. Any activation function is valid, but it must be differentiable in order to use backpropagation (as we will see in the following sections). One important class is *sigmoidal functions* which have upper and lower bounds, and transition smoothly between them. There is also *rectified linear activation* [25] and variants such as the smooth approximation called *softplus* [26]; these are (approximately) zero for negative input and (approximately) linear for positive input.

Topology

Neurons can in general be connected to each other in any way. A network in which the neurons make up a directed acyclic graph is a feed forward neural network. In contrast, if any cycles exist in the network graph, it is called a recurrent neural network (RNN). In a layered FFNN, the neurons are arranged in several layers: an input layer, an output layer and optionally one or more hidden layers. Subsequently, we will be speaking about layered FFNNs unless otherwise noted.

Shallow neural networks with one or two hidden layers are typically used for standard classification; at least one, with a non-linear activation function, is needed in order to implement output functions that are not linearly separable. There are also deep neural networks (DNN) [27] for learning complex patterns, where each layer is supposed to raise the level of abstraction one step. For example: an image may be represented as pixels; these pixels may form edges and lines, which in turn may compose shapes, finally allowing object recognition. The techniques for training DNNs differ somewhat from training shallow networks because of the higher

[1] Inhibitory input is allowed as well, and overrides any other input, forcing negative output.

number of layers. We will focus on shallow networks in the following sections about training.

Each layer is fully connected to the next, and there are no other connections between neurons. See Figure 2.3 for an example of an FFNN with a single hidden layer; each arrow between neurons represents a weighted input (i.e. $x_i w_i$ in Equation 2.8). The number of weights necessary for a network with k layers, where the number of neurons in each layer is $L_1 \dots L_k$, is thus given by

$$\sum_{i=1}^{k-1} L_i L_{i+1}, \tag{2.9}$$

which equals $L_{hidden} \cdot (L_{input} + L_{output})$ for a network with one hidden layer. When $L_{hidden} \ll L_{input}$ and $L_{hidden} \ll L_{output}$, the memory requirement for a FFNN is much less than that of an SVM.

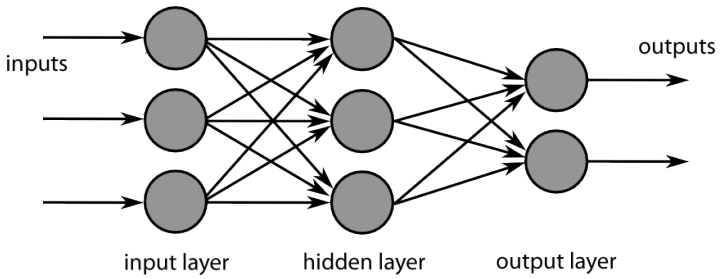


Figure 2.3: *A feed forward neural network with one hidden layer. Image by Wikimedia user Chrislb, licenced under CC BY-SA.*

Under these conditions, feeding a sample to the neural network can be implemented as a series of matrix multiplications (and of course calling the activation functions). Note that the process of feeding an input vector is completely separate from training.

Training as an optimisation problem

Training a feed forward neural network is an optimisation problem; the goal is to find the weight matrices W that minimise the error rate E_W of the classifier. Classical optimisation methods are infeasible because of two main problems. The first problem is the curse of dimensionality, since the number of weights grows linearly with the number of inputs and outputs. The second is that E_W cannot be expressed analytically, nor is it convex,

and so must be approximated by classifying a number of samples, which is resource intensive if a good approximation is sought.

The number of hidden layers, and the number of neurons in each layer, are hyperparameters that are important for the FFNN performance regardless of how the weights themselves are computed. There are several ways to find a good topology, out of which genetic algorithms is one possibility [28].

Training with backpropagation

Backpropagation is a method made specifically for training neural networks. It is done by computing the output error and feeding it backwards through the network (hence the name, *back*wards *propagation* of errors) in order to compute the gradient of the activation function (hence the requirement of a differentiable function). This is coupled with an optimisation method to actually improve the weights.

There are many ways to use the gradient, for example *gradient descent* and *stochastic gradient descent*. With gradient descent, the whole test set is first backpropagated without changing any weights. The gradients are then weighted together, and a small step is then taken in this direction by updating the network weights. This approximates the gradient well, but is computationally expensive. Stochastic gradient descent is a small variation where the weights are instead updated after each backpropagation of the samples in random order. This means some steps may be taken uphill on the error function surface, which may be beneficial for avoiding local minima. Compare this way of using stochastic gradient descent to subsection 2.2.1.

There are variations of backpropagation. One is to add *momentum*; in each update, a fraction of the last step in the search space is added to the current position, $\eta(W_t - W_{t-1})$, in order to increase the speed of convergence, especially when the gradient is small. The magnitude of the step is denoted η. Another variation is weight decay; in each update, all weights are decreased with a factor d. This makes weights that are seldom used decrease over time.

Since the optimisation is only guided by the gradient, backpropagation is sensitive to local minima but relatively fast if initiated near a good solution. How well the weight updates follow the gradient depends on the choice of optimisation method, however. Note that the error rate E_W is not measured directly but rather the activation function of the output layer is used.

Training with particle swarm optimisation

Backpropagation avoids both the dimensionality problem and expensive approximation problem by using the greedy (stochastic) gradient descent algorithm. There are, however, other stochastic optimisation methods suitable for tackling the problem. In this section, particle swarm optimisation (PSO) is described.

PSO has a number of particles, each of which has a position and velocity in the search space. Unlike backpropagation, it does not rely on the objective function being differentiable. Rather, for each sampled point in the search space, E_W has to be estimated by classifying a sufficient number of samples. The particles are systematically directed towards the global and local best found so far. Since the particles are guided by their velocities rather than a local gradient, it is easy to guide the tradeoff between exploration and exploitation (i.e. wide searching in hope of finding more global minima and local searching in order to improve the current best solution, respectively).

These properties make PSO good at finding globally favourable solutions, but slower towards the end of the search. There are also other pleasant properties: it has few parameters compared to backpropagation, and it is trivially parallel. It is also less sensitive to the exact value of bias value x_0, for example.

PSO has been shown to train FFNN faster than backpropagation [29], however under somewhat different circumstances. The examined network had around a dozen weights, while we have over a million; it evaluates a one-dimensional function, while our dimensionality is in the tens of thousands; it is tested with 21 and 201 samples, while we use thousands or millions. Furthermore, the measure used was the number of operations until a specific training error was achieved, not the minimum error achieved.

2.3 Evaluation

This section presents an introduction to relevant theory regarding different ways for evaluating and measuring the accuracy of classifiers and how to compare different classifiers.

2.3.1 Performance measures and confusion matrices

The results of a classification algorithm can be shown in a confusion matrix [30] (see Table 2.2). In the matrix the distribution of the classifications

		Predicted			
		C_1	C_2	$\cdots$	C_k
Actual	C_1	n_{11}	n_{12}	$\cdots$	n_{1k}
	C_2	n_{21}	n_{22}	$\cdots$	n_{2k}
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	C_k	n_{k1}	n_{k2}	$\cdots$	n_{kk}

Table 2.2: *Confusion matrix for multiclass classification.*

is shown by the class predicted by the classifier on one axis and the actual class on the other. Using the values from Table 2.2, different values used when measuring the performance of a classifier can be found:

$$TP_c = n_{cc} \tag{2.10a}$$

$$FP_c = \sum_{i=1}^{k} n_{ic} - TP_c \tag{2.10b}$$

$$FN_c = \sum_{i=1}^{k} n_{ci} - TP_c \tag{2.10c}$$

$$TN_c = \sum_{i=1}^{k}\sum_{j=1}^{k} n_{ij} - TP_c - FP_c - FN_c \tag{2.10d}$$

where TP_c is the true positive for the class c, FN_c the false negative, FP_c the false positive and TN_c the true negative.

Some of the measures for multiclass classification are [30]:

$$\text{Average accuracy: } \frac{\sum_{c=1}^{k} \frac{\mathrm{TP}_c+\mathrm{TN}_c}{\mathrm{TP}_c+\mathrm{FN}_c+\mathrm{FP}_c+\mathrm{TN}_c}}{k} \tag{2.11a}$$

$$\text{Error rate: } \frac{\sum_{c=1}^{k} \frac{\mathrm{FP}_c+\mathrm{FN}_c}{\mathrm{TP}_c+\mathrm{FN}_c+\mathrm{FP}_c+\mathrm{TN}_c}}{k} \tag{2.11b}$$

$$\text{Precision}_\mu\text{: } \frac{\sum_{c=1}^{k} \mathrm{TP}_c}{\sum_{c=1}^{k}(\mathrm{TP}_c + \mathrm{FP}_c)} \tag{2.11c}$$

$$\text{Recall}_\mu\text{: } \frac{\sum_{c=1}^{k} \mathrm{TP}_c}{\sum_{c=1}^{k}(\mathrm{TP}_c + \mathrm{FN}_c)} \tag{2.11d}$$

For a classification problem where each sample only has one label each (the common version of multiclass classification), $\text{precision}_\mu = \text{recall}_\mu$ [31].

2.3.2 Cross-validation

A common way to measure the accuracy of a classifier is using cross-validation. In cross-validation [32][33][34], the data set is divided into k different groups of samples. The classifier algorithm is then trained on all but one of the k subsets and tested on the last subset. The procedure is then done over again using each of the k subsets as a testing set once. The accuracy of the classifier is estimated as the average of the accuracies for the different subsets.

In Figure 2.4 the partitioning of the sample set in 3-fold cross-validation is shown.

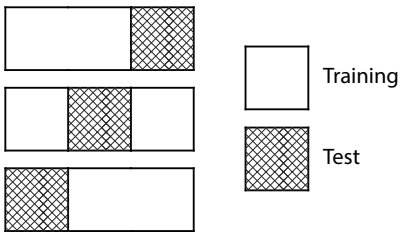


Figure 2.4: *3-fold cross validation.*

2.3.3 McNemar's test

In McNemar's test [32][35], two classifiers are compared by dividing the classified samples into two categories per classifier: correctly classified and incorrectly classified samples. The contingency table seen in Table 2.3 is then constructed from the four different combinations of outcomes the two classifiers can have combined.

	C_A correct	C_A incorrect
C_B correct	a	b
C_B incorrect	c	d

Table 2.3: *Contingency table for McNemar's test between classifiers C_A and C_B.*

Using the values from the filled table the *McNemar's statistic* is acquired:

$$\chi^2 = \frac{(b-c)^2}{b+c} \tag{2.12}$$

To be able to reject the null hypothesis that both classifiers have the same error rate ($b = c$), χ^2 has to be statistically significant and if it is, it is possible to say that one of the classifiers C_A or C_B performs better than the other.

2.3.4 Welch's t-test

The Welch's t-test [36] is a test for comparing the means of two populations which may have different variance (unlike the *Student's t-test* where both populations have to have the same variance [37]). The test statistic is described in Equation 2.13.

$$t = \frac{\bar{x_1} - \bar{x_1}}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{2.13}$$

Here $\bar{x_1}$ and $\bar{x_2}$ are the sample means, s_1^2 and s_2^2 the sample variances and n_1 and n_2 the sample sizes.

3 Implementation

A classifier takes a data point as input and gives a classification as output. In fact, in our case, the classifier returns several classes, ranked by confidence. Thus, the flow of classification is as follows: create data points, feed them to the classification algorithm, and choose one of the returned classes.

A data point, in this thesis, is all information about a transaction and the organisation *at the point in time when the user chooses an accounting code* (i.e. provides the correct classification for that transaction) together with the label. As we will see in sections 3.1 and 3.2, a single data point can be represented in different ways.

The process of creating data points, feeding it to classifiers and interpreting the results is depicted in Figure 3.1 and Figure 3.2.

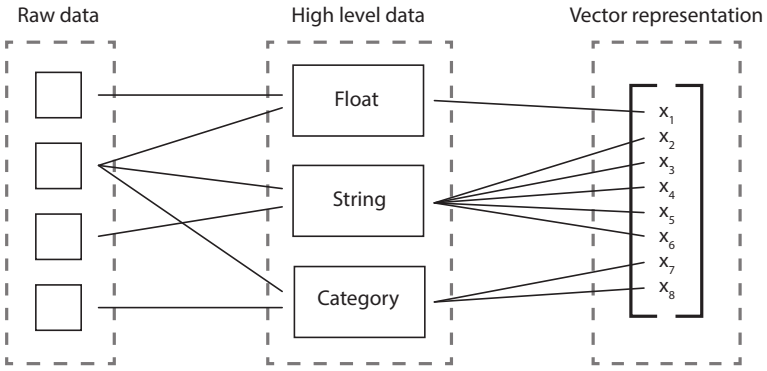


Figure 3.1: *The process of creating the data points (in the form of a vector representation) from the data in the database. First the raw data is converted to high level attributes (described in section 3.1) and then further transformed into the vector representation (see section 3.2).*

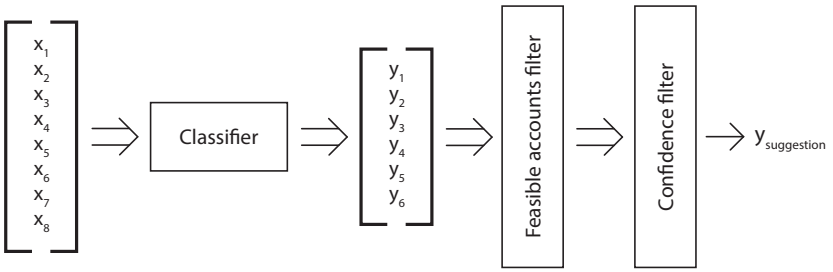


Figure 3.2: *Feeding the vector representation to the classifier (section 3.3) and possibly selecting a label to suggest based on how the results from the classifier were filtered (section 3.4). The* y *vector contains the confidence for each output class.*

This chapter describes how the theory seen in chapter 2 is used in order to implement this process. Finally, in section 3.5, a method for online adaption to single organisations is presented. A closer look on the actual data and machine learning parameters is made in chapter 4.

3.1 Selecting and processing the attributes to a high-level representation

From the database, useful data were extracted to a number of high-level features suited for further processing. Features such as *turnover*, *part of financial year* (when the transaction was made) and *proportion of usage for different account numbers* which were not stored directly in the database were calculated from the raw data (presented in subsection 4.1.1) to a high-level representation. The final high-level features are described in subsection 4.1.2.

3.2 Vector representation

Many classification algorithms require that the input is in numeric form, and several attributes of seemingly high importance were strings or categories (such as the transaction text and the recipient number). There are multiple ways to represent this high-level data as a low-level vector of floating numbers (as described in section 2.1). In this section it is described how the vector representations of such data in this project was done. The different variables were all normalised according to subsection 2.1.1.

Text features and categorical features

Different categorical features were preprocessed in different ways. The *organisation form* (corporation, trading company etc) was represented using the one hot technique (described in subsection 2.1.2) as the possible values for that feature were limited to a pre-known number of different categories (extracted from the name of the BAS chart of accounts version used by the company). Other categorical features and text was represented using feature hashing (described in subsection 2.1.4) since the size of the category and dictionary, respectively, is not known in advance.

Missing values

Most data points lack values for some attributes. The easiest approach of handling such data points is to simply omit them, though this would lead to a loss of the majority of the data points and was not considered.

Another simple approach to represent this is to add a binary variable to indicate that data is missing (as done in [38]). The other variables related to the attribute are set to static, low information values (typically zeroes).

Other, more elaborate ways to handle missing values exist [39][40][41], but because of the time limit it was decided to go with the approach of adding an extra variable.

3.3 Classification

Many machine learning algorithms are easily available in different libraries [4][42]. As the focus of this master thesis was not to re-invent the wheel by implementing existing classification algorithms, but to evaluate how different algorithms would perform in the problem at hand, more attention was given to find libraries that were easy and efficient to use.

This section describes how the different machine learning techniques were used in the thesis and the libraries in which they are implemented.

3.3.1 Neural networks

One of the classifiers is a feed forward neural network trained with back-propagation, described in subsection 2.2.2. Both the neural network itself

and the backpropagation is implemented in PyBrain.

PyBrain is an easy-to-use Python library for using different types of neural networks. We chose this library because it works out of the box. It allows for expressing many types of networks; one may for example add arbitrary connections between neurons. The essential hyperparameters are easy to configure.

There are, however, drawbacks as well. The documentation is subpar, and there has not been any development of new features for the last few years. It is also not very fast: pure math operations use NumPy, but much of the code is pure Python. There are forks such as cybrain and ARAC that aim to make the implementation faster, but we have not evaluated these.

3.3.2 Support vector machines with stochastic gradient descent

As described in subsection 2.2.1, the use of a linear model with Stochastic Gradient Descent (SGD) learning has been shown to give good performance in settings with many samples and high dimensional feature vectors.

In this thesis, the implementation of the linear model (in our case an SVM [14]) using SGD called *SGDClassifier* in the library Scikit-learn [4] was used. The implementation allows for tuning of the different hyperparameters needed: the loss function, the α-value, the regularisation term etc. The *softmax* function [15] was applied to the score for different classes to get the output in the range $(0, 1)$ to represent the confidence of the classifier.

Scikit-learn is a widely used machine learning library for Python with easy to use documentation and a lot of examples. It has continuous development and a large community of developers contributing[1].

3.4 Interpreting classifications

To simulate the behaviour of the live system of SpeedLedger and for improved precision, a so called *feasible accounts filter* was used. Each user has its own chart of accounts, meaning only a subset of all possible accounts are valid; the feasible accounts filter simply removes the invalid accounts from the list of suggestions.

[1] See `https://github.com/scikit-learn/scikit-learn`

To further improve the suggestions from the classifier another filter was added: to only give suggestions when the confidence of the classifier for the sample was above a given limit. This would improve the accuracy of suggestions actually given, which is good in the system of SpeedLedger where bad suggestions might decrease the credibility of the classifier.

Another post-processing step made was to find the top N classes given from the classifier and compare them against the actual class of the corresponding sample. It would be possible for SpeedLedger to adapt the software to be able to give several suggestions, and as such it was interesting to get measurements for different values of N.

3.5 Adapting to single organisations

As different organisations do their accounting differently, a process of adapting a classifier to single organisations was developed. This process was used to improve the precision of the classifiers when a new organisation is encountered. The process works as follows:

First select k different organisations that will simulate new ones. Then train the classifier continuously on samples (drawn randomly) from all organisations except the k ones mentioned above. When a certain limit is reached (e.g. based on time or number of epochs) the training is stopped and a copy for each of the k organisations is created.

Each of the k classifiers specialises on a certain organisation by performing online learning, i.e. testing one sample at the time from that organisation (with the samples ordered chronologically) and then training on the sample just tested on. To be able to learn more from each sample the classifier is allowed to train on N samples back in time from the current sample (in section 4.3, the choice of training on previous samples is denoted as *batchlearning* and N as *size of batch*).

4 Experimental setup

This chapter describes the raw data available at SpeedLedger, the resources with which the experiments were run and which parameters were used in the classification algorithms.

4.1 Available data

In SpeedLedger's system, two kinds of entities describing financial transactions exist: *bank transactions* and *payments*. A bank transaction describes an event that has occurred on the organisation's bank account, e.g. when money for an outgoing invoice has been received or when a customer has paid an invoice. A payment describes the purchase itself, and as such is possibly created before any exchange of money has occurred.

Payments are only created for organisations that use the accrual method (see Appendix A), because separate records are created for when an invoice is received and when it is paid [43][44]. A payment will thus eventually have a connection to a bank transaction, unless it is cancelled, as the bank transaction describes the event of actually transferring the money.

4.1.1 Raw data in database

In the database of SpeedLedger, the following columns exist for *bank transactions* and *payments* (note that only columns relevant for this project are shown):

Bank transaction

- transaction date
- transaction amount
- transaction text

- journal entry id

Payment

- payment amount
- recipient id
- payment date
- payment note
- payment category
- journal entry id

In addition to the information about the actual transaction or payment, there is information about the organisation making them. This could provide context for the transactions, possibly making them easier to classify for the classifiers:

Organisation

- organisation id
- organisation form (corporation, trading company etc)
- addresses of customers to the organisation
- start and end of financial years

In the database, the entire accounting records of the different organisations exist, which allows for features describing historic properties of the organisations or properties that change over time, e.g. turnover or how the usage is distributed between accounting codes.

4.1.2 High-level features

The features that were processed (according to section 3.1) from the raw data in the database together with the type of the feature can be seen in Table 4.1.

Many features describe the current state of the organisation. While running on historical data, this had to be stored for every data point (i.e. every relevant point in time); however, when used in the live system, only the current state of the organisation needs to be maintained.

Description	Type
Which part of the month	Float
Which part of the financial year	Float
Amount	Float
Transaction text	String
Payment category	String
Recipient number	Categorical
Turnover	Float
Organisation id	Categorical
Organisation form (corporation, trading company etc)	Categorical
How large portion of the customers are international customers	Float
How frequently each accounting code is used	Floats
The amount of money accounted on each accounting code	Floats
In which extent the customer uses group accounts versus detailed accounts (see Appendix A for explanations of the terms)	Float

Table 4.1: *The high-level features used*

4.2 Computational resources

The classification algorithms were run on laptops with mid-range hardware. The hardware on the laptops were:

Laptop 1: Intel® Core™ i5-3317U CPU @ 1.70GHz, 8 GB RAM

Laptop 2: Intel® Core™ i3-2367M CPU @ 1.40GHz, 6 GB RAM

During the thesis it was discussed to use more computationally capable hardware, but because of time constraints this was not done.

4.3 Parameters used

The parameters used in the different classification algorithms were found using randomised parameter optimisation [45]. For both of the classifiers, the parameters searched are listed and the ones chosen are highlighted. The parameters for SVM-SGD are shown in Table 4.2 and 4.3 and for FFNN in Table 4.4 and 4.5. Training on many/single organisations refer

to the process described in section 3.5.

Description	Values
Loss function	{**hinge**, modified huber}
Regularisation term (penalty)	{None, L1, **L2**, Elastic net[1]}
α[2]	{10^{-5}, 10^{-4}, 10^{-3}, $\mathbf{10^{-2}}$, 10^{-1}, 10^{-0}}
Learning rate[3]	{constant, **optimal**, inverse scaling}
η_0[4]	{0.25, 0.5, 0.75}
power$_t$[4]	{0.1, 0.3, 0.5, 0.7, 0.9}

Table 4.2: *SVM-SGD: When training on many organisations.*

Description	Values
Alpha value	{10^{-5}, 10^{-4}, $\mathbf{10^{-3}}$, 10^{-2}, 10^{-1}, 10^{-0}}
Learning rate	{**constant**, optimal, inverse scaling}
η_0	{**0.25**, 0.5, 0.75}
t [5] [6]	{1, 10, 100, 1000}
Batchlearning	{**True**, False}
Size of batch	{3, **5**, 7}

Table 4.3: *SVM-SGD: When training on a single organisation (the parameters not listed are the same as in Table 4.2).*

The learning rates listed in Table 4.2 and 4.3 are defined as in Equation 4.1 in the implementation in Scikit-learn used in this thesis[7].

$$\text{Constant: } \eta_t = \eta_0, \tag{4.1a}$$

$$\text{Optimal: } \eta_t = \frac{1}{t + t_0}, \tag{4.1b}$$

$$\text{Inverse scaling: } \eta_t = \frac{\eta_0}{t^{power_t}}, \tag{4.1c}$$

[1] Elastic net is a combination of L1- and L2-regularisation

[2] See Equation 2.7

[3] η_t in Equation 2.5

[4] Used only when learning rate is constant or inverse scaling, thus no value was selected

[5] Used only when learning rate is optimal or inverse scaling, thus no value was selected

[6] A counter for the number of samples trained on so far. For the learning rates optimal and inverse scaling, a higher t value gives a lower learning rate, see Equation 4.1. Setting it to a lower value when starting to train on a single organisation thus increases the learning rate.

[7] See `http://scikit-learn.org/stable/modules/sgd.html`

Description	Values
Learning rate	{0.025, 0.05, 0.1, **0.15**, 0.2}
Decay of learning rate	{**0.99**, 0.999, 0.9999}
Momentum	{0.0, 0.05, 0.1, 0.2, **0.35**}
Weight decay	{0.0, **0.01**, 0.05, 0.1}
Minimum learning rate limit	{0.0005, **0.001**, 0.005, 0.01, 0.05}
Topology (neurons in first hidden layer)	{55, 65, 75, **85**, 95}
Topology (neurons in second hidden layer)	{**0**, 35, 70, 120, 300}
Activation function for output layer	{**softmax** [15], sigmoid}

Table 4.4: *FFNN: When training on many organisations.*

Description	Values
Learning rate	{0.005, 0.01, 0.025, 0.05, **0.08**}
Decay of learning rate	{0.99, **0.999**, 1.0}
Momentum	{**0.0**, 0.02, 0.05, 0.1}
Weight decay	{0.0, **0.02**, 0.05, 0.08}
Minimum learning rate limit	{0.0005, 0.001, **0.005**, 0.01, 0.05}
Batchlearning	{**True**, False}
Size of batch	{3, 5, **7**}

Table 4.5: *FFNN: When training on a single organisation (the parameters not listed are the same as in Table 4.4).*

5 Experimental results

In this chapter graphs and measurements of the applied algorithms are shown, together with results of the different implementation choices described in chapter 3. The notation *overall precision* and *quality* will be used, where *overall precision* refers to the precision_μ (as described in subsection 2.3.1) for all samples, and *quality* refers to the precision_μ for only the samples where a suggestion was actually given[1].

Cross-validation (described in subsection 2.3.2) is a common way to measure accuracy in classification. It was not used because it assumes that all samples are equivalent and unordered, though in our setting the samples are ordered which required other ways of measurement.

Together with the prediction for a data point, a classifier returns the confidence for how certain it is that the given prediction is the correct one, as described in Figure 3.2. When the suggestions given by a classifier are ordered by confidence, it can be seen (as in Figure 5.1) that a higher confidence gives higher precision. In sections 5.1 and 5.2 this is used to give suggestions in only a limited amount of times, suggestions which are more probable to be correct than if all suggestions where given.

[1] Recall that the classifier may choose not to give a suggestion.

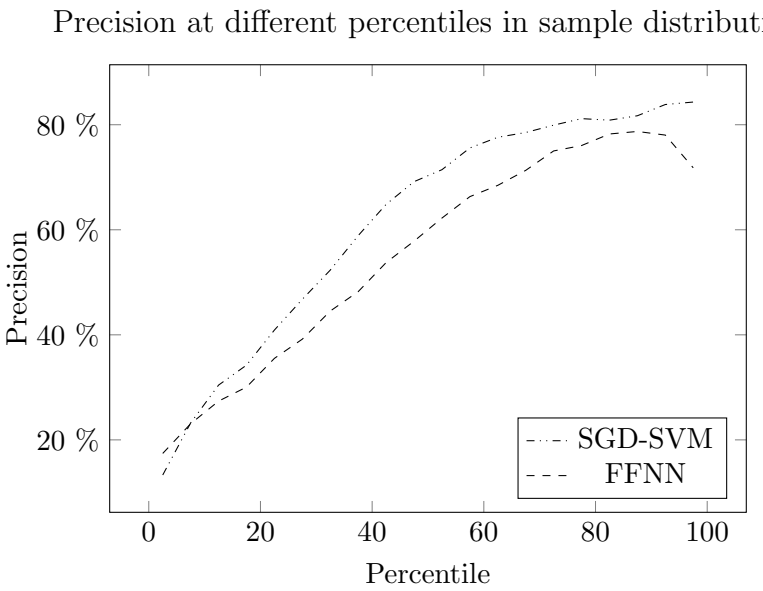


Figure 5.1: *The* overall precision *for different percentiles in sample distribution when samples are sorted on confidence.*

The confidence given by the classifiers lies on an arbitrary scale from 0 to 1, inclusive. Limiting the suggestions given based on the confidence yields different suggestion rates for the different classifiers, which can be seen in Figure 5.2 (where it is clear that a higher confidence limit gives a lower suggestion rate). Because of how the confidence values differ between the classifiers we will present the suggestion rate on the x axises instead of the confidence limits, even though the actual choice of confidence limit will differ between classifiers.

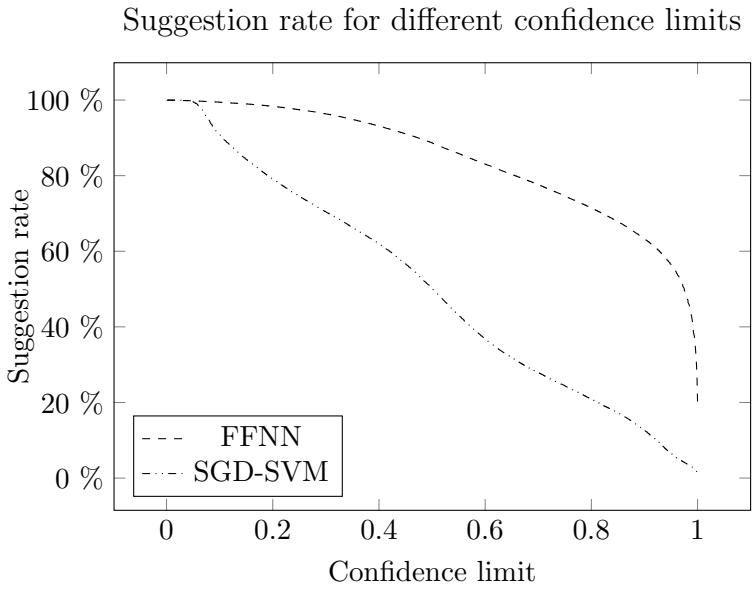


Figure 5.2: *Suggestion rate as a function of confidence limit for the two classifiers.*

5.1 Suggestion rate tradeoff

In Figure 5.3 the tradeoff between a high suggestion rate (which gives a lower *quality*) against a higher *overall precision* is shown. With a low suggestion rate there is more statistical noise, which can be seen in the *quality* for both SVM-SGD and FFNN. The *quality* and *overall precision* for the naïve classifier are shown as marks instead of lines as the suggestion rate for it is an effect of the current implementation and is thus fixed.

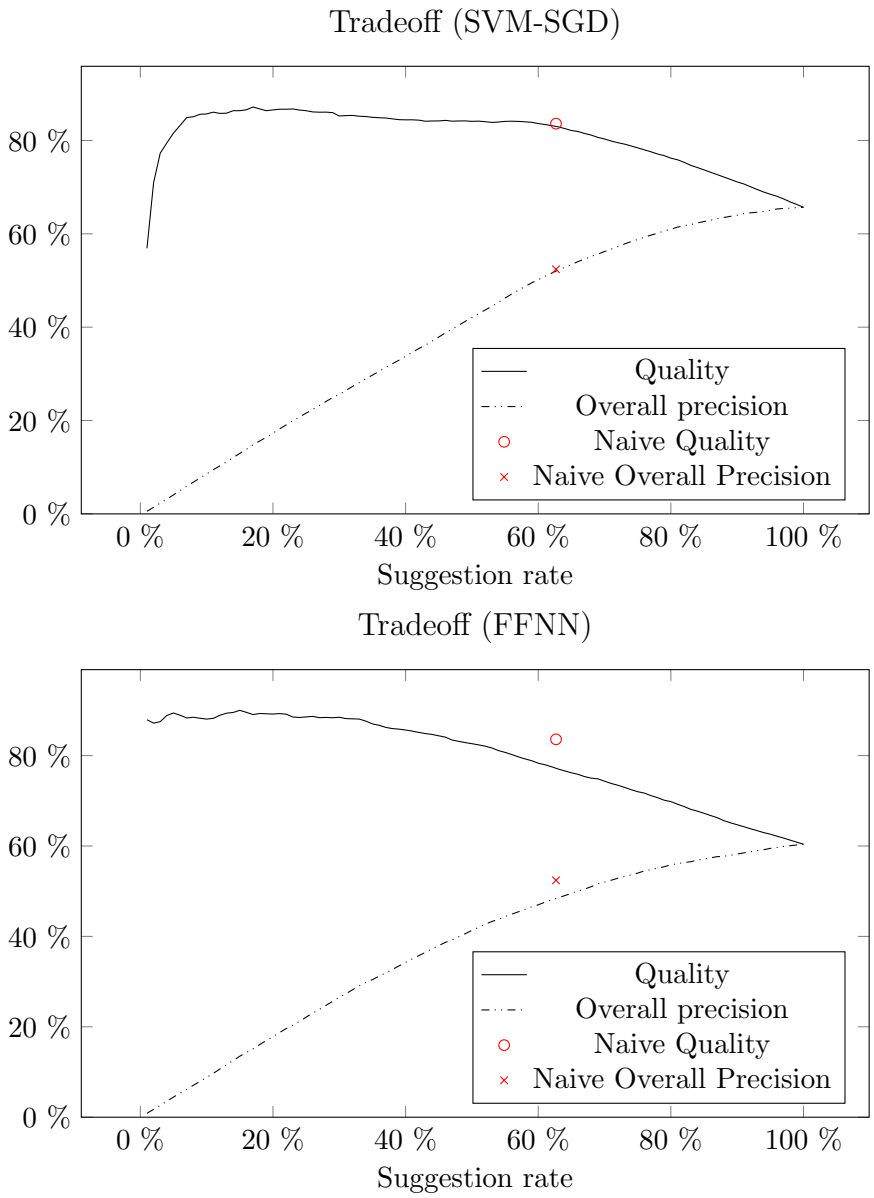


Figure 5.3: *The tradeoff between* quality *and* overall accuracy*. The accuracy is measured as the average of the journal entries at positions 280 to 300, using the data from when adapting the classifiers to single organisations, as described in section 3.5.*

5.2 Accuracy over time

In the system of SpeedLedger, the ability to immediately give suggestions to a new customer is currently missing because of how the naïve classifier gives suggestions based on earlier journal entries from the same organisation. A possible advantage of a machine learning classifier would be that suggestions could be given based on knowledge about other organisations, even when the current organisation is not previously encountered.

In Figure 5.4, the *overall precision* for new organisations when using only suggestions with a confidence level above a certain limit is shown and in Figure 5.5 the *quality* is shown. The confidence limits are chosen such that they give a suggestion rate of 70 % for the SVM-SGD and 65 % for the FFNN. The limits are based on the tradeoffs described in section 5.1. The process of training and testing used for this graph is described in section 3.5 with $k = 300$.

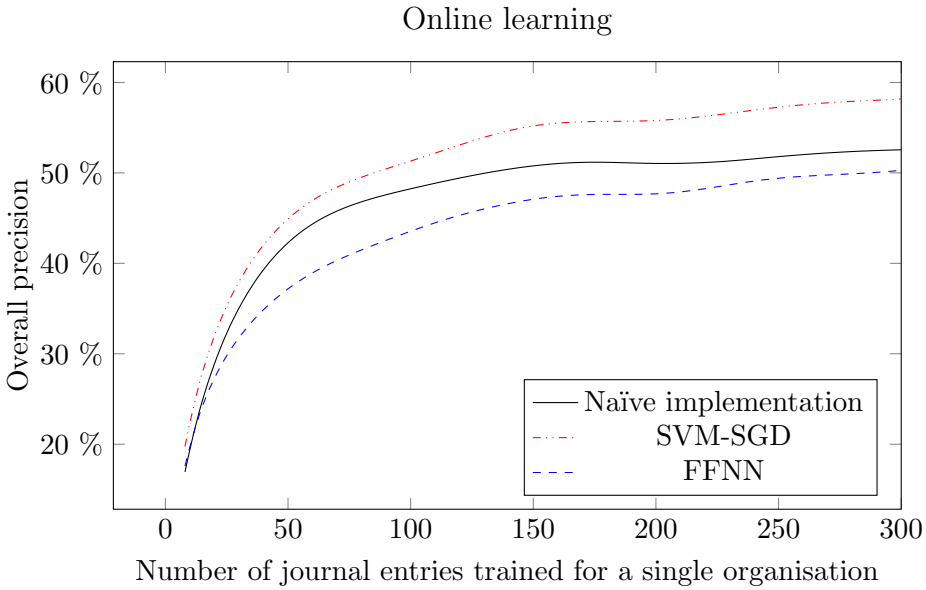


Figure 5.4: *Graph illustrating how different classifiers perform and evolve for organisations it has not encountered before. Every point on the x-axis represents the* ith *journal entry for 300 different organisations. The y-axis shows the* overall precision *of the classifiers for the different entries. Every classifier trains on an entry immediately after classifying it.*

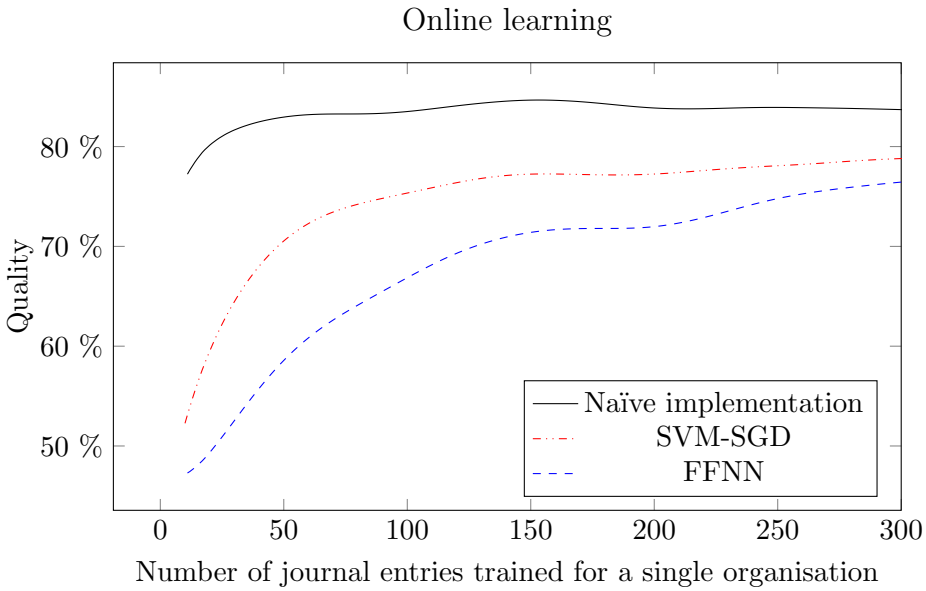


Figure 5.5: *Same as Figure 5.4 but showing* quality*, i.e. the proportion of correct suggestions when a suggestion is actually given.*

5.3 Suggesting multiple accounting codes

As described in section 3.4, a possible post-processing step is to give multiple suggestions based on the confidence of the classifier. Figure 5.6 corresponds to the graphs in Figure 5.3, but shows the quality when giving 1, 3 or 5 suggestions. The confidence of giving N different suggestions together was calculated as the sum of the confidences of the individual suggestions.

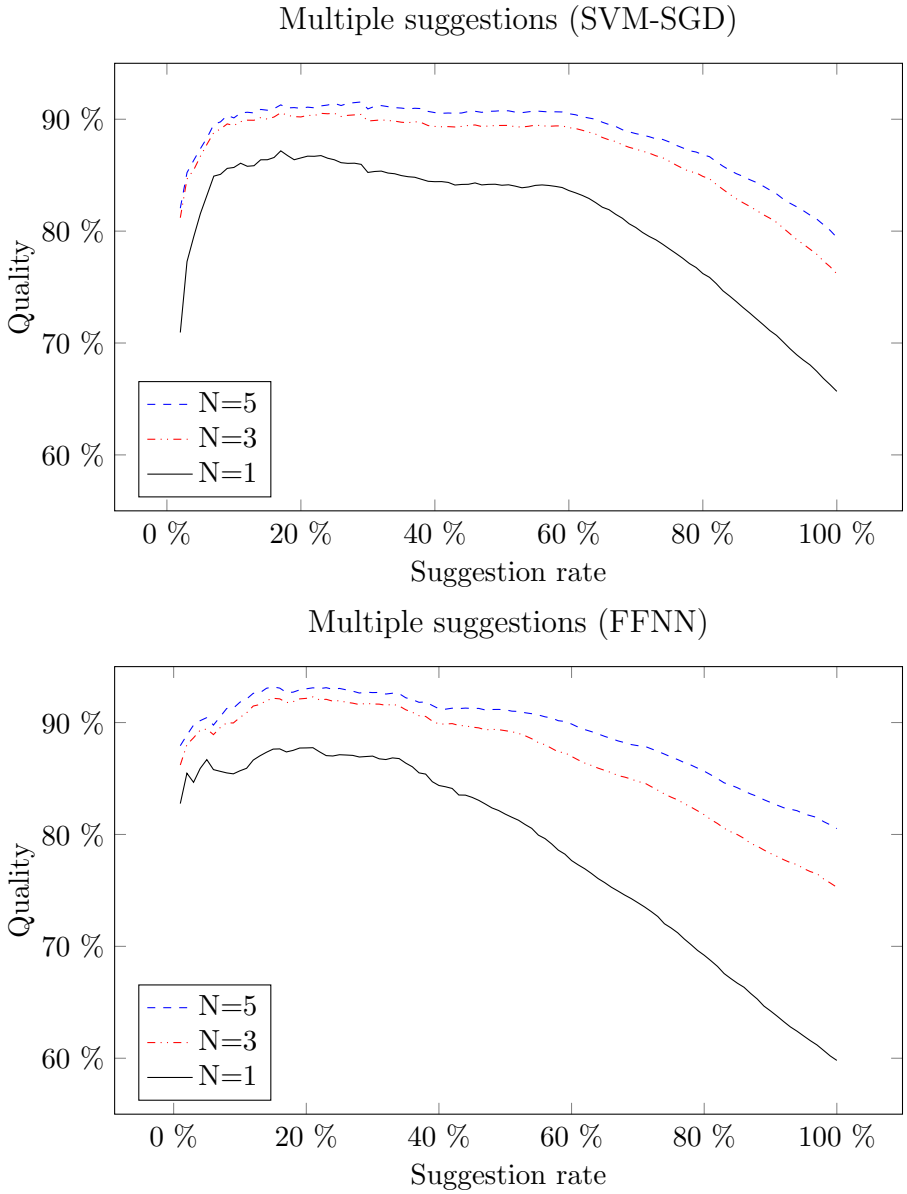


Figure 5.6: *The* quality *as a function of the suggestion rate when giving N suggestions.*

5.4 Other notable results

5.4.1 Accuracy on predefined vs user defined accounting codes

In the BAS chart of accounts, there are both predefined accounting codes and accounting codes which are up to the organisation to decide how to use. This means that each organisation might use the user defined accounting codes very differently. Using Welch's t-test (described in subsection 2.3.4), the classification accuracy of samples with labels among the user defined accounting codes compared to ones with labels belonging to the predefined accounting codes were measured.

The test was run on the SVM (which has been shown to give more accurate suggestions than the FFNN) using the classifier with the same base training as in section 5.2. 10 000 random samples were split into two sets based on if the sample had a label among the user defined accounting codes or not. In the sets, the samples were then split into chunks of 100 samples each and the precision_μ of each chunk was calculated. The means, variances and sizes of the two sets were then used to calculate the t statistic defined in Equation 2.13.

Using the t-distribution, the hypothesis that the means of the two sets are equal were rejected with statistical significance ($p < 0.01$ ‰). The mean of the chunks in the set of samples with user defined accounting codes was 36.38 % and the mean of the chunks in the set of samples with predefined accounting codes was 23.26 %.

5.4.2 Impact of presence of bank transaction info

In practice, payments are known before the transaction has actually occurred (as described in section 4.1). More information is available if doing a suggestion when the money is transferred (i.e. a bank transaction is made and thus occur in the system of SpeedLedger). The additional data made available is the text of the transaction, though in as many as 82 % of all journal entries the transaction text is exactly the same as the payment text. Thus an interesting result to look at is if there is a significant increase in accuracy when using information about both the payment and the bank transaction when evaluating a classifier.

A McNemar's test (described in subsection 2.3.3) was run on the same base classifier as in subsection 5.4.1. Testing was run on 10 000 randomly

drawn samples, first using only the payment information and then also the bank transaction information.

Comparing both runs showed that using bank transaction information gave a statistically significant improvement. However, the improvement was only approximately 0.6 percentage points: with transaction information the accuracy was 24.46%, while not using the transaction information the accuracy was 23.86%.

6 Discussion

The aim of this thesis was to compare the performance of different classification algorithms to the naïve approach, in the setting of classifying financial transaction with regard to how they should be accounted. Comparing the classifiers used in this thesis, Support Vector Machines with Stochastic Gradient Descent learning and Feed Forward Neural Networks, the former shows better performance in the experimental results. However, the naïve classifier still performs on par with, or even better than, the more sophisticated classification algorithms. The different graphs and results are discussed more thoroughly in section 6.3.

The algorithms were used as they showed potential when running preliminary tests on small amounts of data and as they are known to be able to handle large data sets. Some of the limiting factors for the performance of automatic classification are described in section 6.1.

The extraction of data from the database and converting it to a format suitable for machine learning was one of the most time consuming parts of the thesis. If even more work had been put into the preprocessing, e.g. if a more thorough feature selection or text preprocessing had been done, we might have obtained better precision and the time needed for training would probably have been lower [46].

The process of adapting the classifier to single organisations could likely be improved. By splitting the user's data points known thus far into training and test sets, and continuously train until convergence rather than just training on each sample once as they come along, the classifier could possibly achieve higher precision (at the cost of increased demand for computational resources). However, we believe that even bigger improvements are possible by changing the attention to other machine learning methods altogether, as we will see in section 6.4.

6.1 Limiting factors

As briefly mentioned in subsection 1.3.3, there is likely a bias in the labels of the training and test data as a result of users being influenced by the suggestions given by the naïve classifier. To some extent, the classifiers train on what the naïve suggestion was rather than what the user really wanted; this way the score of the naïve classifier is higher relative to the other classifiers than it would otherwise have been.

This was identified as a potential issue early on, and the idea was to train and test only on data points where the naïve classifier did not give a suggestion. This would introduce a new bias that seems to be even worse though: the samples with no naïve suggestion are in fact much harder to classify since they are more likely to be special or irregular in some way. Therefore, the training and test data was kept as is; future measurements, however, ought to be done with independent data (i.e. accounting done without the suggestions from SpeedLedger).

Another, perhaps more important, bias is that the naïve classifier runs in an interactive setting while our testing is done on historical data. In practice, several suggestion might be acceptable. In an interactive setting, this means that any of the acceptable accounting codes may be presented in order to score. In contrast, running on historical data means that only a single suggestion is correct – the one finally chosen by the user. In our results, this favours the naïve classifier, but we do not know by how much since we have not performed interactive testing.

Furthermore, modest computational resources were used for all training (and testing) as seen in section 4.2. There are plenty of possibilities for parallelisms (GPU and/or distributed), but this was excluded from the thesis in order to focus the time on other tasks. The libraries used are also primarily made for ease of use rather than computational heavy lifting, much like other Python libraries. If the concepts of this thesis were to be adapted to a commercial setting, one could expect better results as a result of the classifiers being trained more thoroughly. Note that none of our classifiers ever overfitted.

Last but not least; a premise of using classification is to effectively split all data points into different chunks according to their label, then make each output to be attracted to one chunk and repel all others. This may be counterproductive in our setting where the labels are not always reliable. A different approach will be discussed in section 6.4.

6.2 Promising outlooks

Despite the aforementioned limiting factors, we observe similar precision as the naïve classifier. The maximum possible *overall precision* is far higher and the maximum possible *quality* is somewhat higher than that of the naïve classifier, though they are not achievable at the same time because of the confidence tradeoff described in section 5.1. However, this still suggests the potential of further research into the problem; the extracted features coupled with the vector representation is clearly able to encode the necessary information, which could be reused in the future. We have so far only been able to scratch the surface of what machine learning offers.

6.3 Results commentary

In Figure 5.1 the results are expected; a higher confidence gives a better accuracy, which gives us the ability to use the confidence value as a cutoff in order to increase the *quality* of suggestions. Note in Figure 5.2 that the curves differ a lot, but they both describe a one-to-one relationship between confidence limit and suggestion rate. This difference is the rationale for comparing the classifiers with suggestion rate on the x axis rather than confidence limit.

Figure 5.3 shows the tradeoff between giving only the few best suggestions which gives higher *quality* but with a lower overall precision, or giving many suggestions which gives the opposite result. Since the naïve classifier is a simple deterministic algorithm and does not give varying confidence values, it does not give the option of varying the suggestion rate (and hence the quality and overall precision). For the suggestion rate it actually gets, both values correlate remarkably well with the SVM-SGD; however, thanks to the tradeoff, the SVM-SGD may give a higher overall precision by increasing the suggestion rate. It should be noted that the graph depicts the average for when the curves have settled, i.e. far to the right in Figure 5.4.

As seen in Figure 5.4, the SVM-SGD performs better than both the naïve classifier and the FFNN regarding *overall precision* (with the confidence limits chosen), though in Figure 5.5 we can see that the *quality* is still better for the naïve implementation. There is no value for the confidence limit that gives both better *overall precision* and *quality* than the naïve classifier, mostly because the *quality* remains at slightly lower levels.

Continuing to Figure 5.6, it can be seen that suggesting multiple accounting

codes for each sample would possibly be a good idea; a top three list yields significantly higher results for both the SVM-SGD and the FFNN. It is possible to adapt the naïve algorithm to give several suggestions with relative ease, which ought to prove beneficial if the behaviour is similar to our results.

It is seen in subsection 5.4.1 that the precision is higher for classifying user defined accounting codes rather than the predefined accounting codes, contrary to intuition. We expected the best results for the predefined accounting codes, since each accounting code has the same meaning for every organisation.

Regarding using only information about payment compared to using also bank transaction information as described in subsection 5.4.2, the results are what was expected. Though the accuracy when using also the additional information was not much higher, the difference was statistically significant. The low improvement in accuracy shows that it is not worthwhile waiting for the extra information given by the transaction note.

6.4 Future work

There are many ways and directions that future work in the area can take. In this section we present some of them.

6.4.1 Unsupervised or semi-supervised learning

The label (i.e. accounting code) of a data point depends on which user labelled it, and as such is always going to differ between organisations. Furthermore, labelling done by the users of SpeedLedger's system is biased by the naïve classifier. Because of this, we suggest to look at unsupervised or semi-supervised learning as follows.

Use a clustering algorithm to group data points without their labels. When a user labels a data point, a connection is made between the cluster containing said data point and the corresponding accounting code of the organisation. This connection is unique to the organisation, and is used to suggest the same accounting code to this organisation upon future similar data points that fall into the same cluster. Note that several clusters may point to the same accounting code. Finally, the clustering algorithm may train on the data point, allowing it to improve over time.

The explicit connection between cluster and accounting would likely be

robust, but it would not allow for any initial suggestions. We suggest two main mechanisms for creating preliminary suggestions, i.e. before there is any explicit connection between a cluster and an accounting code for the organisation.

The first is to look at other organisations' explicitly mapped accounting codes for the cluster matching the data point, i.e. accounting code frequency. If most organisations agree on an accounting code for that cluster, it is delivered as a suggestion.

Secondly, the accounting code frequency could be extended with collaborative filtering [47]. Each organisation is characterised by its set of connections between clusters and accounting codes. Organisations that have previously used labels similarly might benefit more from each others' cluster-to-accounting-code-mappings, which is where collaborative filtering comes in handy. Other properties could be used as well, e.g. the industry code used in the Swedish official company registry[1].

There are three predicted advantages to this approach. Firstly, the impact of inter-organisation heterogeneous labelling is minimised. Indeed, it would only be used for the preliminary suggestions. Secondly, adaption to single organisations would be much more efficient in terms of storage and computational needs. Lastly, collaborative filtering is made possible by the explicit representation of connections between clusters and accounting codes. For these reasons, we believe that this is a viable approach for continued research.

6.4.2 General improvements

There are possible actions to take that would improve accuracy regardless of whether classification or the suggested method above is used.

The importance of proper preprocessing of text is thoroughly examined in [48]. One example is stemming which is *"a procedure to reduce all words with the same stem to a common form, [and] is useful in many areas of computational linguistics and information-retrieval work"* [49]. In practice, it means that different variations of the same word would be encoded by the same variable in the vector representation seen in section 2.1. One could also expand this by considering e.g. personal names to be equal.

Both supervised and unsupervised learning requires a powerful model for mapping input to output. The only kind of neural network used in this

[1] See `http://www.sni2007.scb.se/snisokeng.asp`

thesis is a shallow feed forward neural network trained with backpropagation. There are however many other well studied variations of neural networks, such as deep neural networks. There are different ways to train them as well, as seen in subsection 2.2.2, and other ways of initialisation than random [50][51].

If suggestions based on machine learning would be implemented in the live system, we believe that the learning algorithm would profit from having a high suggestion rate in the beginning. Rather to have a short period when a customer is new when the suggestions may have lower quality to more rapidly increase the accuracy, than to have a low suggestion rate. Also, because of the possibility that a user just accepts the suggestion given, it is possible to get a high accuracy of the suggestions as long as the suggestions are "good enough". This is however hard to give evidence for unless testing in a live environment.

7 Conclusion

During this thesis, the possibility of using classification algorithms in accounting has been looked at. Relevant data from the database of Speedledger was transformed in multiple steps into a vector representation fit for the algorithms. The classifiers were then trained and tested on the preprocessed data to evaluate how they would perform when applied to the live system.

The results from running the algorithms on historical data in the thesis indicates that there is potential for machine learning in the area. For more elaborate results, testing the algorithms in the live system would be needed. There is probably more than one sufficiently good suggestion for each sample which is not possible to test without interactive feedback.

Because of the uncertainty of the labels (as different organisations use the accounting codes in different ways), we believe that other kinds of machine learning such as unsupervised or semi-supervised learning might be a better approach to reach a fully automated accounting process.

A Explanation of accounting terms

This appendix is aimed at people with computer science background but with little or no experience of accounting.

Business transaction Sw. *affärshändelse*. A single atomic economic event; any flow of value within a business (e.g. a transfer of money from a cash register to the bank account) or between parties (e.g. selling a sandwich to a customer). Note the word *flow*; money always has at least one source and at least one destination. Furthermore, the flow from all sources equals the flow to all destinations.

Journal entry Sw. *verifikation*. A structured description of a business transaction, including accounting records and a complete assignment of accounting codes (e.g. "from *food sales* to *cash register*"). It also contains some metadata such as the date. A journal entry is the smallest building block in accounting.

Accounting code Sw. *kontonummer (inom bokföring)*. A four digit number identifying an account. An account is used for describing a specific source or destination of money. Some accounts are generally used as either source or destination (e.g. *buying food* is a destination) while others are used as both (e.g. the bank account is used both for sending and receiving money).

Accounting records Sw. *underlag till verifikat*. Evidence for a business transaction, e.g. a receipt.

Assignment of accounting code Sw. *kontera*. Determining the correct accounting codes for a journal entry. In general this includes several accounting codes, but in the context of this thesis, all accounting codes are implicit except one. The bank account involved is known before the classification, and the VAT code is given by the accounting code chosen by the user. The classification task is to determine the single missing accounting code. Classifying several missing accounting codes is out of scope for this thesis. This is described in-depth in section 1.4.

Accrual method Sw. *fakturametoden*. Separation of date of purchase versus the exchange of money. With the accrual method, one journal entry is created when something is bought or sold without exchanging money yet, i.e. a debt is created. Another journal entry is created upon the exchange of money, i.e. the debt is being paid off.

With the opposite, **cash flow accounting** (Sw. *kontantmetoden*), no debt is ever created; rather, an item is considered bought or sold the day it is paid.

Chart of accounts Sw. *kontoplan*. A list of accounting codes and a policy for how they are supposed to be used. The **BAS chart of accounts** is widespread in Sweden. It is, however, a template rather than a complete chart of accounts: organisations may deviate from it. Indeed, some ranges are explicitly available for organisations to customise according to their needs, so called *user defined accounting codes*.

VAT code Sw. *momskod*. A unique number identifying a specific use of VAT as well as the VAT percentage. Each accounting code is statically assigned at most one VAT code. When an account is used, a VAT account is per default also used according to this VAT code.

Group accounts and detailed accounts Sw. *gruppkonton och detaljkonton*. A group account is an accounting code which gives a more general description of a journal entry while a detailed account gives more information about it. An example in the **BAS chart of accounts** is the group account *5610 Car costs* with some of its detailed accounts *5611 Fuel*, *5612 Insurance and tax* and *5613 Repairs and maintenance*.

Bibliography

[1] S. M. Ul-Huq, "The role of artificial intelligence in the development of accounting systems: A review", *The IUP Journal of Accounting Research & Audit Practices*, vol. 13, no. 2, pp. 7–19, 2014.

[2] J. Daintith and E. Wright, *A dictionary of computing*. Oxford University Press, Inc., 2008.

[3] P. D. Robles-Granda and V. Belik, "A comparison of machine learning classifiers applied to financial datasets", in *Proceedings of the World Congress on Engineering and Computer Science*, vol. 1, 2010.

[4] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python", *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[5] X. Wu *et al.*, "Top 10 algorithms in data mining", *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008.

[6] D. Pyle, *Data preparation for data mining*. Morgan Kaufmann, 1999, vol. 1.

[7] A. K. Jain, R. C. Dubes, *et al.*, *Algorithms for clustering data*. Prentice hall Englewood Cliffs, 1988, vol. 6.

[8] S. Aksoy and R. M. Haralick, "Feature normalization and likelihood-based similarity measures for image retrieval", *Pattern Recognition Letters*, vol. 22, no. 5, pp. 563–582, 2001.

[9] T. Hastie *et al.*, *The elements of statistical learning*, 1. Springer, 2009, vol. 2.

[10] T.-S. Lim *et al.*, "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms", *Machine learning*, vol. 40, no. 3, pp. 203–228, 2000.

[11] T. Joachims, *Learning to classify text using support vector machines: Methods, theory and algorithms*. Kluwer Academic Publishers, 2002.

[12] K. Weinberger *et al.*, "Feature hashing for large scale multitask learning", in *Proceedings of the 26th Annual International Conference on Machine Learning*, ACM, 2009, pp. 1113–1120.

[13] K. Ganchev and M. Dredze, "Small statistical models by random feature mixing", in *Proceedings of the ACL08 HLT Workshop on Mobile Language Processing*, 2008, pp. 19–20.

[14] C. Cortes and V. Vapnik, "Support-vector networks", *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[15] C. M. Bishop *et al.*, *Pattern recognition and machine learning*, 4. springer New York, 2006, vol. 4.

[16] V. Vapnik, *The nature of statistical learning theory*. Springer Science & Business Media, 2000.

[17] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms", in *Proceedings of the twenty-first international conference on Machine learning*, ACM, 2004, p. 116.

[18] L. Bottou and Y. Le Cun, "On-line learning for very large data sets", *Applied stochastic models in business and industry*, vol. 21, no. 2, pp. 137–151, 2005.

[19] H. Robbins and S. Monro, "A stochastic approximation method", *The annals of mathematical statistics*, pp. 400–407, 1951.

[20] L. Bottou, "Large-scale machine learning with stochastic gradient descent", in *Proceedings of COMPSTAT'2010*, Springer, 2010, pp. 177–186.

[21] Y. Tsuruoka *et al.*, "Stochastic gradient descent training for l1-regularized log-linear models with cumulative penalty", in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, Association for Computational Linguistics, 2009, pp. 477–485.

[22] M. Wahde, *Biologically inspired optimization methods: An introduction*. WIT press, 2008.

[23] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.", *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[24] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[25] X. Glorot *et al.*, "Deep sparse rectifier networks", in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP Volume*, vol. 15, 2011, pp. 315–323.

[26] C. Dugas *et al.*, "Incorporating second-order functional knowledge for better option pricing", *Advances in Neural Information Processing Systems*, pp. 472–478, 2001.

[27] J. Schmidhuber, "Deep learning in neural networks: An overview", *CoRR*, vol. abs/1404.7828, 2014. [Online]. Available: `http://arxiv.org/abs/1404.7828`.

[28] D. Stathakis, "How many hidden layers and nodes?", *International Journal of Remote Sensing*, vol. 30, no. 8, pp. 2133–2147, 2009.

[29] V. G. Gudise and G. K. Venayagamoorthy, "Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks", in *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, IEEE, 2003, pp. 110–117.

[30] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks", *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.

[31] V. Van Asch, "Macro-and micro-averaged evaluation measures [[basic draft]]", 2013.

[32] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms", *Neural computation*, vol. 10, no. 7, pp. 1895–1923, 1998.

[33] S. L. Salzberg, "On comparing classifiers: Pitfalls to avoid and a recommended approach", *Data mining and knowledge discovery*, vol. 1, no. 3, pp. 317–328, 1997.

[34] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection", in *Ijcai*, vol. 14, 1995, pp. 1137–1145.

[35] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages", *Psychometrika*, vol. 12, no. 2, pp. 153–157, 1947.

[36] B. L. Welch, "The generalization ofstudent's' problem when several different population variances are involved", *Biometrika*, pp. 28–35, 1947.

[37] K. K. Yuen, "The two-sample trimmed t for unequal population variances", *Biometrika*, vol. 61, no. 1, pp. 165–170, 1974.

[38] Microsoft. (2015). Microsoft neural network algorithm technical reference, [Online]. Available: `https://technet.microsoft.com/en-us/library/cc645901(v=sql.110).aspx` (visited on 06/02/2015).

[39] S. García *et al.*, *Data Preprocessing in Data Mining*. Springer, 2015.

[40] P. K. Sharpe and R. Solly, "Dealing with missing values in neural network-based diagnostic systems", *Neural Computing & Applications*, vol. 3, no. 2, pp. 73–77, 1995.

[41] E. Pesonen *et al.*, "Treatment of missing data values in a neural network based decision support system for acute abdominal pain", *Artificial Intelligence in Medicine*, vol. 13, no. 3, pp. 139–146, 1998.

[42] T. Schaul *et al.*, "Pybrain", *The Journal of Machine Learning Research*, vol. 11, pp. 743–746, 2010.

[43] E. J. McMillar, *Not-for-Profit Budgeting and Financial Management*, 4th. John Wiley & Sons, Jun. 2010.

[44] S. A. McCrary, *Mastering Financial Accounting Essentials: The Critical Nuts and Bolts*. John Wiley & Sons, 2010.

[45] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization", *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 281–305, 2012.

[46] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection", *The Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[47] F. Ricci *et al.*, *Recommender systems handbook*. Springer, 2011, vol. 1.

[48] A. K. Uysal and S. Gunal, "The impact of preprocessing on text classification", *Information Processing & Management*, vol. 50, no. 1, pp. 104–112, 2014.

[49] J. B. Lovins, *Development of a stemming algorithm*. MIT Information Processing Group, Electronic Systems Laboratory, 1968.

[50] D. Nguyen and B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights", in *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, IEEE, 1990, pp. 21–26.

[51] J. Y. Yam and T. W. Chow, "A weight initialization method for improving training speed in feedforward neural network", *Neurocomputing*, vol. 30, no. 1, pp. 219–232, 2000.