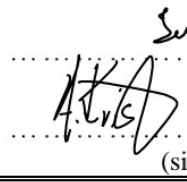
University of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study program/specialization: Computer Science	Spring semester, 2018 Open / ~~Confidential~~
Author: Ivan-Louis Miranda Husebø, Andreas Kvist	.....  ..... (signature of author)

Programme coordinator: Krisztian Balog

Supervisor(s): Krisztian Balog (UiS) and Magnus Folde Glendrage (Bouvet)

Title of Master's Thesis:
Decreasing Manual Workload by Automating SAP Travel Expense Workflows

Credits: 30 ECTS

Keywords: Travel Expense Claims, SAP, process automation, OCR, Machine Learning, Neural Networks.	Number of pages: 84 + supplemental material/other: - Biblography - Appendices - code.7z Stavanger, June 15, 2018

Acknowledgements

We would like to extend a thanks to our external supervisor **Magnus Folde Glendrage** for providing a stern view on the progress of the thesis making sure we continuously worked hard. Also a thanks to **Deepa Jose** for answering all our questions about cluster tables.

Our internal supervisor **Krisztian Balog** should also be greatly credited for guiding us in the process of writing and researching in the thesis.

Another thanks would also have to be made to our team and business unit from Bouvet Norge AS and as well as The Norwegian Government Agency of Financial Management, first of all for making this thesis possible, but also for allowing us to use their systems and providing critical information in processes we studied.

Lastly a thanks to our family and friends, especially **Marie Valdal Tømmerås**, for enduring, motivating us to push forward and always having our backs.

Abstract

In the 21st century, efficiency is a key focus for several organisations, and because of this, machine learning and process automation is getting a lot of attention. The Norwegian Government Agency of Financial Management deals with a large amount of travel expense claims every year, which makes them a possible point of interest for process automation. The claims are at this point approved by manual labour, but we will research the possibility of automating parts of this process by using historical data extracted from the SAP backend system used. To perform this automation, several machine learning methods will be tested to perform a classification on the data. As there are attachments involved in a lot of the claims, Optical Character Recognition will be used to perform a processing of these. We failed to produce a solution that could perform good classification on the extracted data, but our results prove that there it is possible to solve this problem in an optimal manner.

Keywords: Travel Expense Claims, SAP, process automation, OCR, Machine Learning, Neural Networks.

Contents

Chapter 1

Introduction

This thesis is a collaboration between two students from the University of Stavanger, Bouvet Norge AS (Bouvet) and The Norwegian Government Agency of Financial Management[1] (DFØ). The students are currently employed by Bouvet and are working on multiple projects as consultants for DFØ.

1.1 Background

Every year, DFØ receives more than $500,000$ refund requests in the form of travel expense claims (TECs). These include a destination, dates, a purpose, optional stopovers, comments and attachments in addition to the different kinds of expense items and allowances. DFØ provides multiple different ways of requesting refunds, including a mobile application developed by Bouvet[2]. There are multiple steps involved in a refund request, starting when the need of a trip rises and ending when the requested refunds are placed into the employee's bank account. In some cases, the employee needs to fill in a travel application which, when approved, allows the employee to fill in a TEC after the trip is finished. Each claim is approved in two steps by two different employees. The first step is usually performed by a human resources (HR) professional, and the second is usually a superior to the applicant. The HR professional will validate the TEC and look for errors, and the superior will for instance confirm that the applicant actually went on the trip that he or she claims and that they have been allowed to take said trip.

[1] `https://www.dfo.no`
[2] `https://dfo.no/kundesider/l%C3%B8nn/df%C3%B8-app`

The whole process can potentially take a few months. Consider that it takes around two minutes to review and approve a TEC and that an average governmental employed HR professional salary is $488,000\ NOK$ [15]. The result is expenses for 18 full-time employees, which is $18 * 488,000 = 8,784,000\ NOK$ in salary annually, not including any additional expenses that may be included in a realistic situation.

The backend system is an Enterprise Resource Planning (ERP) system developed by SAP SE[3]. The system stores the latest data from each TEC and in some cases, a portable document format (PDF) of a previous version. The system also provides an endpoint that users connect to through the applications. Programs can be executed internally in the same system, and these programs can access all database tables that are utilised by each customer, given that the user has the correct authorisations.

The objective of this thesis is to create a proof of concept for an automatic validation of a TEC, where a system performs a check in comparison with other previously classified TECs. The General Data Protection Regulation (GDPR), and other rules and regulations regarding handling of personal information will also have to be taken into consideration as we develop a solution, as the data we use to classify a TEC may be considered personal information.

1.2 Solution

We wish to dive into the process of approving TECs and see if any steps in this process can be automated. To do this, we will to explore previously submitted data for TECs in DFØ's database, extract it and see if it can be used to train a machine learning algorithm. As we see potential in saving a lot of resources, both human and financial, we want to find out if we are able to find a combination of tools that can eliminate the human element of approving TECs so the employees that are involved in these processes can spend their time elsewhere. In general, we want to see if the data we find contain enough high-quality information to train some algorithm, so it can decide on whether to approve or reject a TEC with certainty above some threshold, e.g. 95%. If the resulting certainty is below this threshold, it can be forwarded to a human resource that can manually process it.

[3] `https://www.sap.com`

1.3 Main contributions

The main work in this thesis is connected to the following:

- Explore the TEC workflow and identify points that can be automated.
- Identify the location of all TEC data.
- Extract data from backend and construct a working dataset.
- Find the best suited machine learning algorithm(s) for our needs through testing.
- Train this(these) algorithm(s) with old TECs to predict the outcome of new ones.
- Analyse results and calculate the efficiency of the solution(s).
- Compare the efficiency of a neural network compared to a simple classifier using hand crafted features.
- Utilise an already existing solution for extracting data from images of receipts.

1.4 Organising

We have used several productivity tools to organise our workflow. We have used Bitbucket[4] for version control, Slack[5] for communications, Trello[6] to keep track of and delegate tasks, OneDrive[7] for storing and sharing other documents. Finally, we have used Draw.io[8] and Geogebra[9] for creating figures and models.

Our code is written mostly in Python[10], and to keep the code structured we have created modules that contain the core code. To execute the code, we have used Jupyter[11] as it allows us to keep a Python kernel open and execute sections of code at a time. The other programming language we have used is ABAP[12], which is developed by SAP and is used solely in SAP systems.

[4] https://bitbucket.org/
[5] https://slack.com/
[6] https://trello.com/
[7] https://onedrive.live.com/
[8] https://www.draw.io
[9] https://www.geogebra.org
[10] https://www.python.org
[11] http://jupyter.org/
[12] https://www.sap.com/developer/topics/abap-platform.html

Table 1.1: Work distribution

Task	Andreas	Ivan-Louis
OCR	0%	100%
Classifiers	100%	0%
Organising	40%	60%
SAP development	60%	40%
Background research	50%	50%
Writing	50%	50%

1.4.1 Distribution of Work

Our thesis is part of a pilot project at the University of Stavanger that allows two students to write a master's thesis together. Because we are two authors, we had to delegate the work of certain sections to be able to separate our contributions. We have cooperated in some chapters and/or sections and in others, we have divided our work completely. Table 1.1 gives an overview of the work distribution in the thesis.

1.5 Outline

Related work and background will take into consideration papers and articles that relates to our thesis, and at the same time give details on the methods, frameworks and classifiers we have used.

Problem statement and overview gives a deeper technical introduction to our thesis' focus and gives a quick overview of work conducted in relation to this.

Classifiers contains the description of the data extraction process and data pre-processing. It also have in depth explanations of the chosen features and the classifiers we have tested, both for creating a baseline as well as creating neural networks.

Optical Character Recognition will explain the usage of the Optical Character Recognition in our thesis. It shows our analysis of the potential it has within our problem parameters, and the final results.

Evaluation contains some evaluations of the different tasks we have performed in the thesis.

Conclusion and future work has the conclusions that we have made in our thesis as well as some plans for additional work that we can perform at some point in the future.

Chapter 2

Related Work and Background

In this chapter we will introduce some work related to ours, descriptions of algorithms and techniques used in machine learning as well as a few popular machine learning frameworks used today.

2.1 Machine Learning

Machine Learning is a concept of using an algorithm to process and find patterns in large amounts of data. If the amount of data is sufficiently large enough it may be referred to as *big data*. Big data is often categorised as when it becomes difficult to store, search, analyse or share the data. These patterns are stored in a model that can be used further to predict the outcome of a new cases of the same data structure. For instance, given a model that been *trained* on large amounts or insurance claims that are either classified as fraudulent or legitimate, it can be able to predict whether a new insurance claim is fraudulent or not based on the parameters that it received. Application of machine learning methods to large databases is called *data mining* and the analogy being large amounts of raw material extracted from a mine that is filtered to recover smaller amounts of valuable material [1, section 1.1].

2.1.1 Supervised and unsupervised learning

Machine learning is practised in two main categories: supervised and unsupervised. In supervised learning, an algorithm maps an input to a corresponding output. This is the most used method of machine learning in every day use, as it is used for predictions like stock prices, whether an MRI image of an organ has a tumour or not, or for recognising faces or speech [1, section 1.1]. Super-

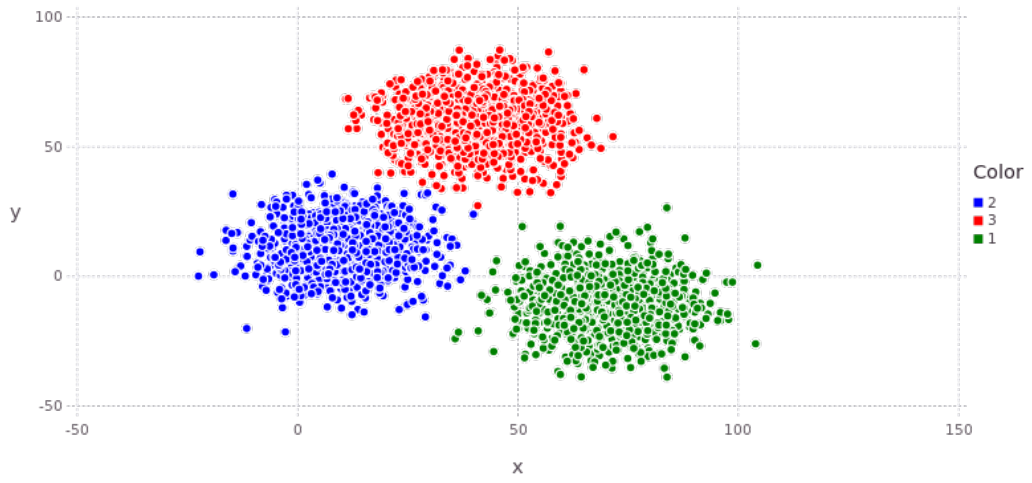


Figure 2.1: Clustering

vised learning can be further split into two different subcategories: *classification* and *regression*. Classification decides on one out of two or more specified outcomes, like whether a given image contains a dog, a cat or something else, whereas regression decides on a continuous value, like predicting a new price for a certain stock. Unsupervised learning does on the other hand not return an output. Instead, it is used to find similarities in the inputs.

2.1.2 Machine learning tasks

Unsupervised learning

A widely used technique in unsupervised learning is *clustering*. Clustering is used to form groups of data that are similar to each other. Figure 2.1 shows a set of data points grouped into three clusters using *k-means clustering*. K-means clustering finds k clusters by calculating *centroids* by finding the mean x and y coordinates of all the points in the cluster. The algorithm continues to execute until the centroids do not change or the amount of changed centroids are below a given number of points in the cluster. The figure is retrieved from Exegetic's blog post about clustering[1]. Clustering can be useful in many different applications. For instance, books can be categorised using clustering to find similar books and group them together in genres, or animals and plants can be categorised into different species[2].

[1] `http://www.exegetic.biz/blog/2015/10/monthofjulia-day-30-clustering/`
[2] `https://www.geeksforgeeks.org/clustering-in-machine-learning/`

Supervised learning

For classification, an example can be the case of deciding whether an insurance claim is fraudulent or legitimate. It is the process of labelling a data point with a class. A data point can have any number of attributes along with a single target attribute (the class). A classifier can be binary, meaning only two outcomes (e.g. fraud or legitimate), or it can have any other number of possible classes. Cars for instance have different prices, widths, lengths, heights, amount of torque and break horsepower. These attributes contribute to specify if the type of car is a small city car, a family car, a sports car or a race car.

Figure 2.2: An image containing multiple different objects that can be recognised by a classifier

Classification can also be used in images like in the popular MNIST dataset[3]. The MNIST dataset contains 70,000 images in total of handwritten digits. Here, the classification task is to classify the digit that is located in the image. In another example, consider the image in Figure 2.2. This image contains a number of different items like a table, a plant, two benches as well as other objects. In a case like this, a classification task could be to recognise the different items located in the image.

Sometimes, the task is not to predict one out of a specified amount of outcomes, but rather a continuous value like when predicting the price for a stock or predicting the price of a house or apartment. In cases like these, classification can be inefficient as there could potentially be way too many output labels so instead we can use regression. For instance, given a dataset with data from house sales in some time period, some of the attributes for each sale can be:

- Size in square meters

[3] `http://yann.lecun.com/exdb/mnist/`

- Condition
- House/Apartment category
- Floor (apartment)
- Number of floors
- Number of rooms
- Neighbourhood/Location
- Price (target label)

This dataset can of course be used in a classification problem to predict one of the other attributes. For regression however, we could predict either the size or the price. In our case, due to the nature of our task, the type of data we have and what we would like to accomplish, we have a classification task at our hands.

2.1.3 Data representation

To allow an algorithm to efficiently find patterns and allow it to perform predictions, the representation and structure of the data used for training is crucial. It is important that each data entry is structured in the same manner and one problem with big data is noise. Noise is essentially a data point with unnecessary or corrupt attribute values. Too much noise causes a classifier to make wrong decisions as it distorts the patterns in the data. Attributes like unique identifiers are useless, so these kinds of attributes have to be removed before processing.

Another type of noise can be found in images used in deep learning, which will be discussed later on. Here, some noise could potentially be designed to make an algorithm believe that something completely different is apparent in the image, whereas a human will have no problems seeing what's actually there. An example noise for this case can for instance make an algorithm believe that an image contains a cabinet, when it actually contains a person. Each attribute in the data set needs to be reviewed and, if necessary, removed. When the data is processed and potentially noisy attributes removed, the data is ready to be applied to a model to extract patterns. We will dive into two different learning approaches, *feature-based* learning and *representation* learning.

2.1.4 Feature-based learning

In feature-based learning, patterns are found with the use of user-defined features that specify what to look for in the data. These features are built by using the attributes in the data e.g. the attributes alone, a combination of multiple attributes or by utilising bins to group values in an attribute. Feature-based learning uses these features to learn a function $y = f(x)$ that will fit the training data in the best possible way [9]. By learning a function, we refer to finding the parameters of e.g. a linear function $f(x) = ax + b$, where a is the slope of the line, and b is the intersection of the y-axis. Figure 2.3 shows a series of data points that seems to indicate that the actual function is a linear function. By training a model, we can update the values of these two parameters to improve the accuracy of our testing data.

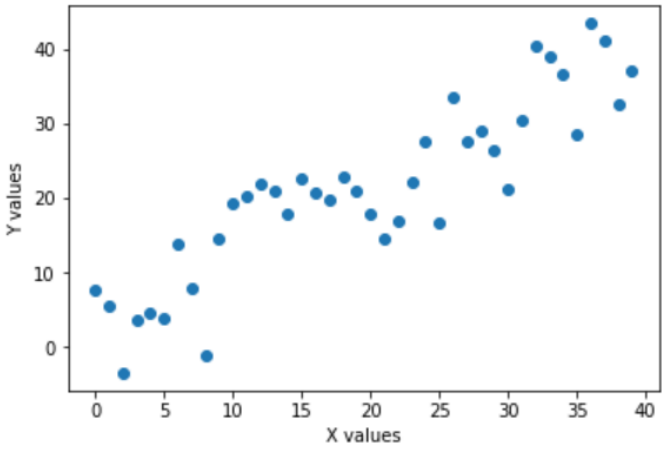


Figure 2.3: A series of points on a graph that seems to indicate a linear function

Because the performance of the model created in feature-based learning algorithms is directly affected by the quality of the features, they need to be chosen carefully. A good way to find features that will perform well is to explore the training data to see if the attributes contain a lot or little to no useful information. For instance, if the goal is to classify whether a dog is of a specific breed, an attribute like the height of the dog can give a good amount of helpful information. On the other hand, an attribute like the eye colour might not help at all assuming that all breeds of dogs have the same distributions of eye colours. Sometimes, a feature can perform better if it is not used in its raw form, but rather defined by combining two or more attributes in the data or by modifying the representation of the values. One technique is to binarise data, for instance giving the numerical value of 1 to all vehicles with four or more wheels and 0 otherwise. Similarly, consider a dataset of credit card transactions like the one in table 2.1. Here, we can see that in all cases where the *Amount* attribute is one or below, the transaction is fraudulent. This allows us to create buckets of values where one of the buckets contain all data points where the amount is one or below.

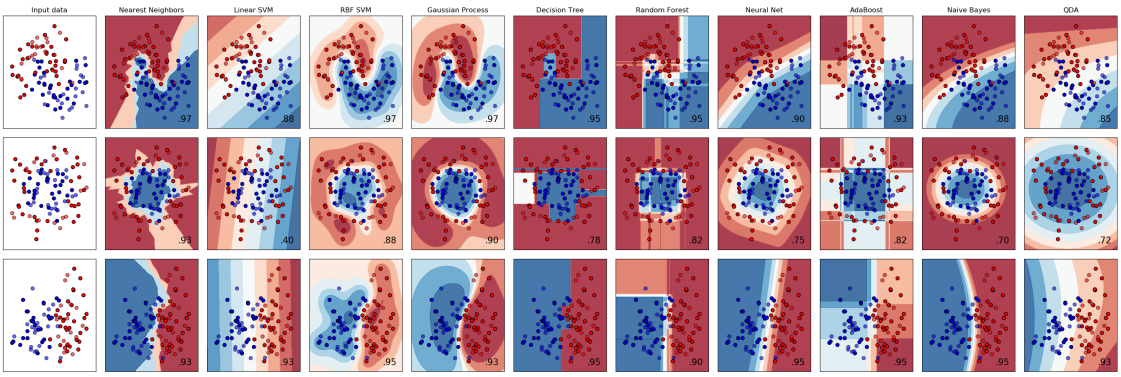


Figure 2.4: Classifier comparison, retrieved from scikit-learn's (SKLearn) classifier comparison

There are several types of classifiers that use different algorithms. One classifier may perform better than another depending on the input data. Figure 2.4[4] illustrates the performance of 10 different classifiers on three different synthetic data sets. The developers specify that the results should be taken with a grain of salt as the examples may not carry over to real data sets. Despite this, the plot can still prove useful for ruling out certain classifiers before starting with any development.

Table 2.1: Credit card transactions

Time	V1	V2	V3	Amount	Class
7.0	-0.894286082202820	0.286157196276544	-0.113192212729871	93.2	Legitimate
2.0	-1.15823309349523	0.877736754848451	1.548717846511	69.99	Legitimate
6986.0	-4.39797444171999	1.35836702839758	-2.5928442182573	59.0	Fraudulent
7.0	-0.644269442348146	1.41796354547385	1.0743803763556	40.8	Legitimate
4.0	1.22965763450793	0.141003507049326	0.0453707735899449	4.99	Legitimate
9.0	-0.33826175242575	1.11959337641566	1.04436655157316	3.68	Legitimate
2.0	-0.425965884412454	0.960523044882985	1.14110934232219	3.67	Legitimate
0.0	1.19185711131486	0.26615071205963	0.16648011335321	2.69	Legitimate
7519.0	1.23423504613468	3.0197404207034	-4.30459688479665	1.0	Fraudulent
7526.0	0.00843036489558254	4.13783683497998	-6.24069657194744	1.0	Fraudulent
7535.0	0.0267792264491516	4.13246389713003	-6.56059996809658	1.0	Fraudulent
7543.0	0.329594333318222	3.71288929524103	-5.77593510831666	1.0	Fraudulent
7551.0	0.316459000444982	3.80907594667829	-5.61515901119457	1.0	Fraudulent
7610.0	0.725645739819857	2.30089443776603	-5.32997618300917	1.0	Fraudulent
406.0	-2.3122265423263	1.95199201064158	-1.60985073229769	0.0	Fraudulent

[4] http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Naive Bayes

Naive Bayes bases itself on an "naive" assumption of independence between every pair of features. With class variable y, and dependent feature vector x_1 through x_n, Bayes' theorem states that

$$P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n}.$$

With the independence assumption,

$$P(x_i|y, x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = P(x_i|y)$$

simplifies to

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_1|y)}{P(x_1, ..., x_n)}.$$

With $P(x_1, ..., x_n)$ as a constant given the input the classification rule becomes:

$$P(y|x_1, ..., x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y) \rightarrow \hat{y} = \arg\max_y P(y)\prod_{i=1}^{n} P(x_1|y)$$

The main difference between naive Bayes classifiers is mainly the assumption made regarding the distribution of $P(x_i|y)$. Despite of the over-simplified assumptions, naive Bayes classifiers works well in many real-word situations, such as spam filtering. Requiring small amounts of data, naive Bayes learners and classifiers can be very fast in comparison to other methods. An explanation of the classification performance naive Bayes gives can be found in the paper written by Harry Zhang [16].

Decision trees

Decision trees are the results of an algorithm that builds a tree of rules. All branches on the tree ends in a *leaf* node, which has a single output value for one of the labels that the decision tree should predict. In the training process of a decision tree, a node considers each of the attributes in the dataset. For each of the attributes, it will split the dataset into new subsets of the original dataset, where each subset corresponds to the value that the data points have for that specific attribute. This process is continued recursively until all the data points in the subset have the same output label, or until the information gain does not improve by further splitting the dataset. This splitting process, by using each attribute once in each initial branch from the root node, creates a tree structure with the root node being the base of the tree.

Table 2.2: Titanic

Survived	Pclass	Sex	Age	Ticket	Fare	Cabin	Embarked
0	3	male	22	A/5 21171	7.25		S
1	1	female	38	PC 17599	71.2833	C85	C
1	3	female	26	STON/O2. 3101282	7.925		S
1	1	female	35	113803	53.1	C123	S
0	3	male	35	373450	8.05		S
0	3	male		330877	8.4583		Q
0	1	male	54	17463	51.8625	E46	S
0	3	male	2	349909	21.075		S
1	3	female	27	347742	11.1333		S
1	2	female	14	237736	30.0708		C

Consider the popular Titanic dataset on Kaggle. As shown in Table 2.2, the output label is *survived*, but one could also predict the other attributes. In the case of a decision tree, one could for instance start by splitting the dataset into the sex attribute. Assuming the attribute can be converted into a binary attribute, splitting by this attribute will split the dataset into two subsets. These two new subsets can now be split further by another attribute, e.g. the age attribute. As the age attribute is not as simple to split, an option is to create buckets of age pools, like five buckets with ages [0, 20), [20, 40), [40, 60), [60, 80) and [80, inf). Splitting in this way would mean that each of the two subsets would create five new subsets, creating a total of 10 subsets in the outer nodes of the tree. This process will continue until all nodes have the same target label. An example result of a built tree can be shown in Figure 2.5.

Decision trees are relatively cost efficient, with the cost of predictions being logarithmic to the number of data points used to train the tree. As decision trees may creates biases if some classes dominate the others, small variations in the data may result in a completely different tree. One way to ensure more stability is to use a collection of decision trees, with the results being aggregated into a final result, a *random forest*. Random forests may reduce training overhead by for example training on different subsets of the data, or using random subset of features.

Pruning is a technique used to address overfitting, which is when a decision tree potentially fit the training data perfectly, but does not produce good prediction results on data it has not seen before. There are two kinds of pruning, pre-pruning and post-pruning, which correspond to reducing the size of a decision tree before and after the construction has completed. This generalises the decision tree, so that the splits in the tree are not too strict.

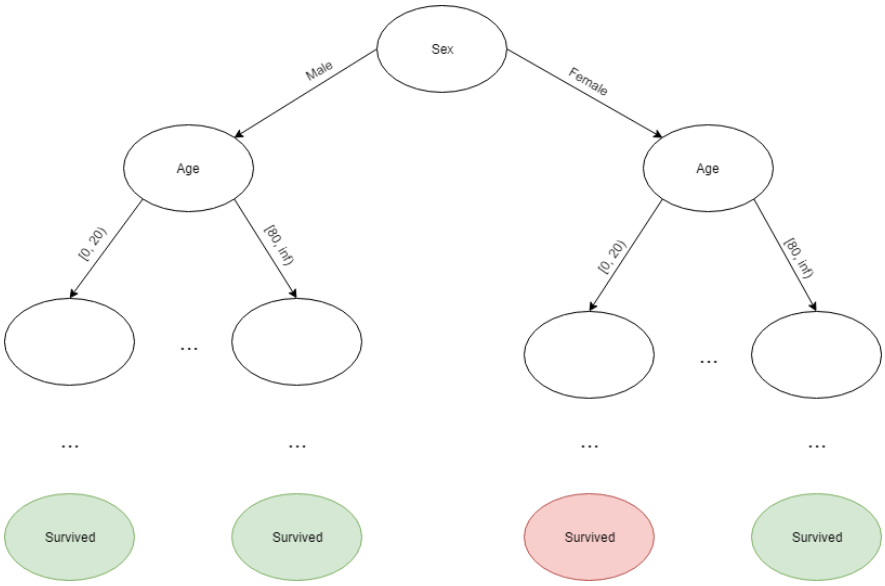


Figure 2.5: Decision tree

Pre-pruning, also known as the early stopping rule, introduces some conditions in the building process that will prevent the tree from growing to its full extent. Some typical early stopping conditions are cases where all the data points in a subset belong to the same output label, or if all attribute values are the same. Other, more restrictive stopping conditions include [14]:

- A specified maximum depth has been reached
- All attributes for the examples are the same
- No improvements in the information gain

Post-pruning on the other hand allows a tree to grow completely. Then, for each leaf node in the tree, compare the generalisation error of the node and its siblings with their common parent. If the generalisation error improves, replace the parent with a leaf node. The label of this new leaf node is the most common label in the subset [14].

Random Forest

A random forest classifier is a type of classifier known as an *ensemble*. It is in fact a collection of classifiers, which inspired the name *forest* as it is an collection of decision trees. A random forest seeks to increase the stability of a regular tree classifier. If we consider a decision tree as asking a person for his/her opinions on whether or not to watch a movie, with random forest we ask multiple different people and consider all of their inputs when finally deciding for ourselves [4].

The SKLearn library for random forests also supports feature importances. The feature importance attribute is a list of numerical values that sum up to one, that indicates how important each attribute in the dataset is. This means that the forest can be trained on a full dataset before the resulting importances are examined in order to see if any attributes can be removed entirely.

Nearest neighbours

Nearest neighbours, or k-nearest neighbours, is a simple classifier that uses the coordinates of points in a n-dimensional graph to calculate the distance between any new point, and all the existing points. There are multiple ways of calculating the distance, but one popular way is the *euclidean distance*. Euclidean distance between two points is calculated as the square root of the squared distance between the different coordinates of both points in a graph. E.g., the distance between two points in a three-dimensional graph with coordinates x, y and z will be calculated as

$$d(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2 + (p_z - q_z)^2}.$$

Nearest neighbours uses this distance to decide the label of a new point by choosing the label of the nearest neighbour. If a new point is an equal distance between its two nearest neighbours, there are a few ways to decide. One way is simply by randomly choosing one of the points, and another way is to introduce the k. The only difference is that instead of choosing the nearest neighbour, the classifier chooses the k-nearest neighbours. The value k is usually an odd number so one ensures there will always be a majority in one of the two labels.

2.1.5 Neural Methods

In representation learning, like neural networks, the model learns the representation of the data itself. This means that it is not needed to specify features and the raw data can be fed straight into the model for training. In [9], Goodfellow et al. take a look at the history of deep learning, the way neural networks are employed today, and research of deep learning. In the book's introduction they write about the three waves of artificial neural network research, with deep learning being the most recent. In this paper the use of the term "neural networks" will be referring to the fields of deep learning and artificial neural networks in computer science.

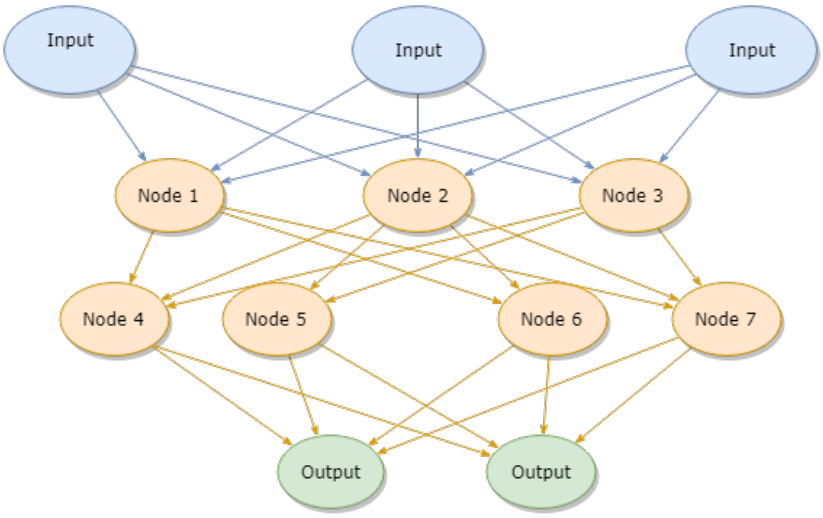


Figure 2.6: Example of a neural network with two hidden layers

Architecture

Choosing the right architecture for a neural network can be challenging. Over the years since the research into neural networks began, there have been many architectures proposed for different types of problems. Some of these include:

- Feedforward neural networks
- Recurrent neural networks
- Recursive neural networks
- Convolutional neural networks

One of the simpler neural networks is the fully connected feedforward neural network, also called a multilayer perceptron (MLP). In Figure 2.6 an example of a simple feedforward neural network is shown, but it is missing some connections between the first and the second hidden layers to be called fully connected. By fully connected, we mean that in each layer in the network, the units have a connection to every unit in the next layer in the network. The layers in such a network are also referred to as dense layers. The MLP utilises forward propagation, which means that the output of each unit directly influences the value of the connected units in the next layer. The opposite, backpropagation, feeds the output of a layer back into the model for it to influence itself when computing the gradient [9].

The different types of neural networks have different use-cases in which they shine. This is especially true when it comes to convolutional neural networks and how they have improved image

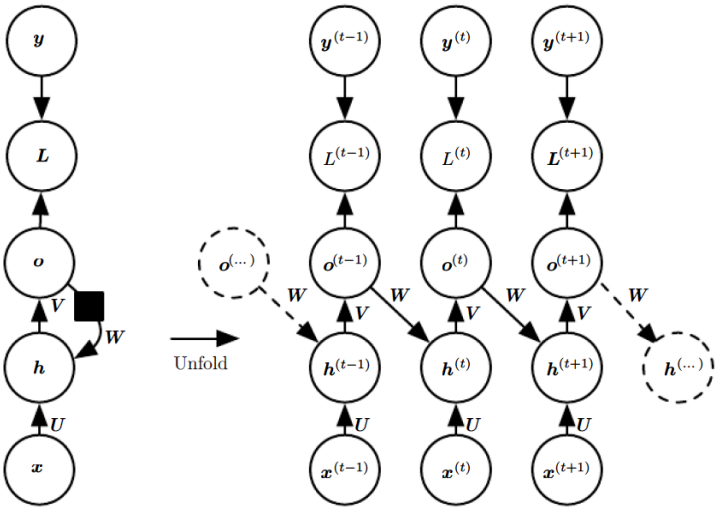


Figure 2.7: Recurrence through output

processing. The convolutional network is built with convolutional layers, which utilises the convolution operation as shown in Equation 2.1, instead of a regular matrix multiplication. In [9], Goodfellow et al. explains the convolution as using multiple readings and averaging them to locate a spaceship, as it is not realistic to get completely continuous readings. However, this average will prefer more recent readings and gives a penalty to older ones.

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \tag{2.1}$$

Recurrent neural networks are often used in language processing, as its recurrent nature have many similarities in how sentences are built, where each next word in a sentence is influenced by the chain of words in the past. One popular recurrent model is the Long Short-Term Memory (LSTM) model which has been proved to be extremely successful in applications like recognising handwriting, speech recognition, handwriting generation, machine translation, image captioning and parsing [9]. An example of a recurrent neural network is shown in Figure 2.7, where the output of a hidden unit is fed back into the model to influence the next hidden unit in the network. The figure is retrieved from [9, Section 10.2].

Pooling and dropout

In between layers in a neural network, there are additional computations that one can add to alter the output. Two of these are pooling and dropout. A pooling operation in neural networks is an operation that helps the neural network become more invariant to small changes on the input. A pooling operation works by traversing its way through a matrix input with another matrix that is

48	192	224	254	254	57	7	221
269	4	88	91	10	164	66	176
92	225	71	72	191	216	12	159
133	242	172	146	133	245	182	41
97	127	111	47	187	101	174	217
101	45	47	212	169	197	45	166
5	161	209	51	223	92	111	125
188	68	244	240	140	14	124	120

(a) Initial values

269	269	224	254	254	57	7	221
269	269	88	91	10	164	66	176
92	225	71	72	191	216	12	159
133	242	172	146	133	245	182	41
97	127	111	47	187	101	174	217
101	45	47	212	169	197	45	166
5	161	209	51	223	92	111	125
188	68	244	240	140	14	124	120

(b) Max pooling

128,3	128,3	224	254	254	57	7	221
128,3	128,3	88	91	10	164	66	176
92	225	71	72	191	216	12	159
133	242	172	146	133	245	182	41
97	127	111	47	187	101	174	217
101	45	47	212	169	197	45	166
5	161	209	51	223	92	111	125
188	68	244	240	140	14	124	120

(c) Average pooling

Figure 2.8: The pooling process

smaller than the matrix it is working on. Two types of pooling are max pooling and average pooling operations. Figure 2.8 shows the basics of both the max pooling and average pooling operations, where a part of the original matrix is either equalised to the max or average value of the the selected values. When this computation is completed, the pooling operation moves a specified number of steps to the right (also called strides) and continues doing this, moving along the entire matrix to pool values together.

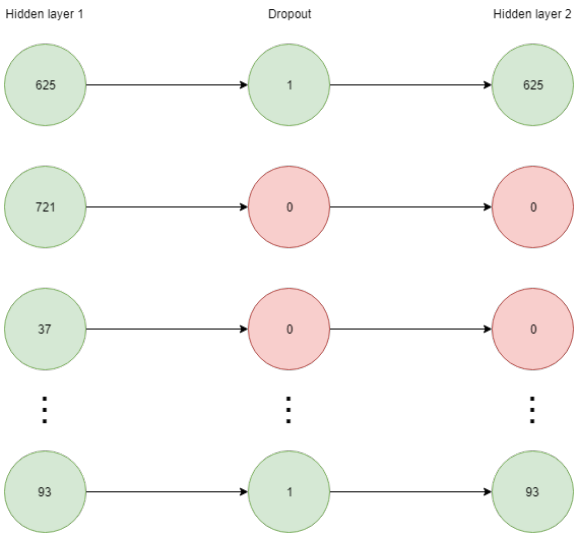


Figure 2.9: Dropout

Dropout is an operation that excludes some hidden units. A dropout layer is initialised with a keep probability specified by the developer, that decides whether or not the value in the hidden unit should be kept or not. If it is decided not to keep the value, it is simply set to zero and passed on to the input unit in the next layer. Figure 2.9 shows a representation of a dropout layer in between two other layers in a neural network. Out of the four visible units, two are kept and two are dropped. A typical probability to keep the value of a unit is either 0.8 or 0.5. Another typical usage of dropout is in *ensemble* networks. Ensemble networks is similar to what a random forest is to the decision tree. An ensemble network is a collection of neural networks that decide together. In cases like

this, the usage of dropout would be whether or not to keep the output of the entire network in the final prediction [9].

Data augmentation

A strategy especially useful in neural networks is *data augmentation*. Data augmentation is a way to generate additional data by modifying the training data already present in the data set. This is useful in cases where there is not enough data for the model to find patterns. A technique in data augmentation for e.g. images are to change the hue or saturation of the images. This leaves the same patterns in the image, but with different colours. Another technique in data augmentation is flipping or distorting the image. With techniques like these, the patterns are still present in the image, but the data itself is different. Figure 2.10 shows examples of how the image of a cup can be altered to create multiple copies containing the same patterns.

Figure 2.10: Multiple representations of the same image

Loss and Cost Functions

A loss function is the main evaluation method for the training process of a machine learning algorithm [8]. It is used to measure the current performance by for instance measuring how many mistakes the algorithm makes in its classifications. The more mistakes it makes, the higher the loss. There are a lot of different loss and cost functions, and while *cross entropy* is the one we ended up relying most on, we explain a lot of the other possible functions here.

Gradient Descent uses the derivative of a function at a specific point on a graph to decide the direction one should move, in order to move closer to the lowest point in the graph, as shown in Figure 2.11. Here, if the $f'(x) < 0$, moving to the right will decrease the value of $f(x)$ and if

$f'(x) > 0$, moving to the left will decrease the value of $f(x)$. A challenge for the gradient descent algorithm is to decide whether or not a current minimum on the graph is the optimal solution. Even though the derivative of a function might be zero at a certain point on the graph, it might be a *local minimum*. A local minimum is where the values on both sides of the point will be higher, but eventually, the graph can start to descend again. A problem with a local minimum is that the graph could end up in another local minimum whose value is lower than the previous one. The optimal solution is referred to as the *global minimum*, meaning no value in the entire graph is lower than that specific point. In cases where there are more than one input, say the three inputs $f(x, y, z)$, the gradient is computed with the partial derivative for each input [9].

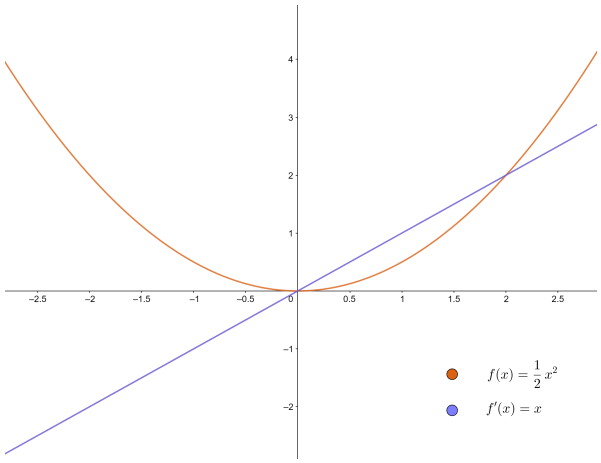


Figure 2.11: A function and its derivative indicating the direction to the minimum

Stochastic Gradient Descent (SGD) is an extension of gradient descent. In [9], the authors explain that the difference between stochastic and regular gradient descent is that SGD uses an estimate of the gradient. This estimate is calculated by a small fraction or *minibatch* of one to 100 or 1000 samples from the training set. The reason for doing this is that when the size of the training set increases, one can expect the computation of the gradient to become more complicated. With a larger training set the computations of the gradient would also takes longer to converge. The runtime is $O(N)$, but the problem rises when the size of the training set is a billion examples, so one would want to speed up this process.

Mean Squared Error (MSE) is a cost function that increases as the euclidean distance between the predictions and labels of a test set increases [9].

$$MSE_{test} = \frac{1}{m}||\hat{\boldsymbol{y}}^{(test)} - \boldsymbol{y}^{(test)}||_2^2 \tag{2.2}$$

This measurement can also be used for the training set by multiplying the MSE with the gradient of the weights, and solving the equation for where the gradient is zero [9]:

$$\nabla_{\boldsymbol{w}} MSE_{train} = 0 \tag{2.3}$$

$$\Rightarrow \nabla_{\boldsymbol{w}} \frac{1}{m} ||\hat{\boldsymbol{y}}^{(train)} - \boldsymbol{y}^{(train)}||_2^2 = 0 \tag{2.4}$$

$$\boldsymbol{w} = (\boldsymbol{X}^{(train)\top} \boldsymbol{X}^{(train)})^{-1} \boldsymbol{X}^{(train)\top} \boldsymbol{y}^{(train)} \tag{2.5}$$

In [9], the final solution for the weights is as shown in Equation 2.5.

Negative log likelihood, also referred to as the *cross entropy*, is a very common cost function. The reason for this is due to the fact that, in comparison to MSE, small changes in the output from the neurons make a very large difference because of the logarithmic part of the function. Equation 2.6 shows the cross entropy function for discrete variables of t and y. This logarithm also makes the gradient very large when the target value is one and the output from the neurons is almost zero, i.e. when the output is wrong.

$$C = -\sum_j t_j log(y_j) \tag{2.6}$$

Activation functions

An activation function in a neural network is a fixed, non-linear transformation from the raw output of each neuron. In [9], the activation function is defined as a function that is applied element-wise to an affine transformation, from a vector $\boldsymbol{x}$ to another vector $\boldsymbol{h}$, where $h_i = g(x^\top \boldsymbol{W}_{:,i} + c_i)$. Here, we will list a few different activations used in neural networks today.

Sigmoid, also known as the *logistic sigmoid*, is a function that saturates at each end and is defined by

$$\sigma(x) = \frac{1}{1 + exp(-x)}. \tag{2.7}$$

The sigmoid function will, at each end of its value range, be insensitive to small changes to the input as shown in Figure 2.12 [9]. It is also easily recognised by its *s*-shape.

Tanh, also known as the *hyperbolic tangent* activation function, is similar to the sigmoid function with a similar *s*-shape, but instead of only existing in $[0, 1]$, it exists in $[-1, 1]$.

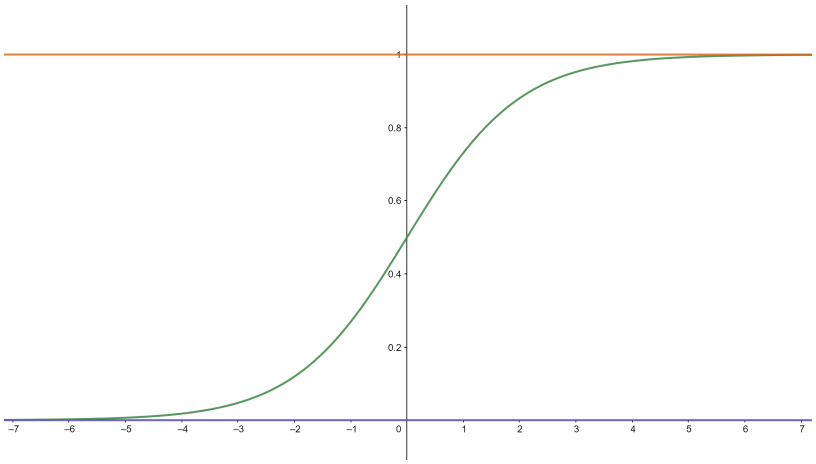


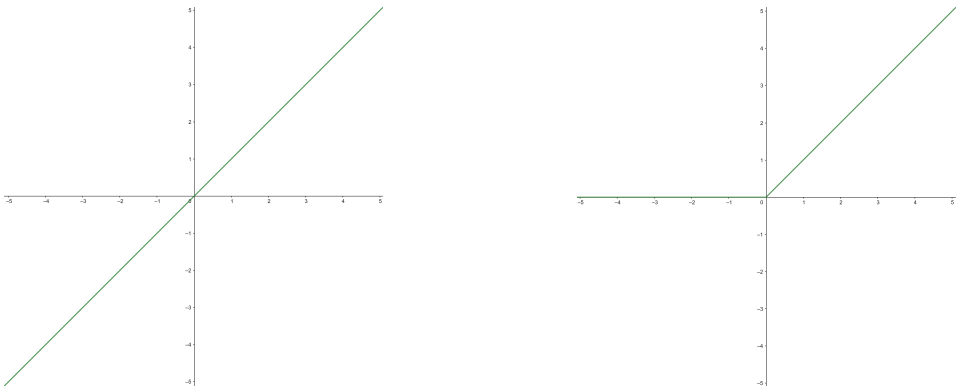
Figure 2.12: The sigmoid activation function

Softmax is a popular activation function often used in the output layer of a neural network. It is defined by

$$softmax(\boldsymbol{x}_i) = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(x_j)}, \tag{2.8}$$

and is used in multi-label classification (although it also works for binary classification). The softmax can output probabilities for each possible label in the dataset that sum to one.

Rectified Linear Unit, or ReLU, is a popular activation function often used in between layers. Instead of a regular linear unit, where $g(z) = z$, ReLU is defined as $g(z) = max\{0, z\}$, so all values below zero are discarded. The difference between the linear function and the ReLU function is shown in Figure 2.13.



(a) $g(z) = z$ (b) $g(z) = max\{0, z\}$

Figure 2.13: The difference between the linear function and the ReLU function

2.1.6 Overfitting and Underfitting

In machine learning, it is important that when one attempts to learn a function, the function must perform well on predicting not only data it has seen before, but more importantly, data it has never seen before. After all, what we want is to use prior experiences to perform well in the future, like using previously drawn handwritten digits from the MNIST dataset to find digits in other images. To perform well on unseen data is called *generalisation* [9].

When training an algorithm, we use some sort of measurement, like a loss function that we want to reduce to perform well. This is called the *training error*, but even though this training error is low, the *test error* can be high. The test error is in fact what we want to reduce in order for the algorithm to perform well on new unseen data. Hence the test error is also called *generalisation error* [9].

The two main focuses in the training process is to keep the training error as low as possible, while at the same time keeping the gap between the training error and the test error small. Failing to keep the training error small results in *underfitting*, which means that the algorithm fails to find the patterns in the data and will not perform well. If the training error is small, but the test error is high, the algorithm suffers from *overfitting*, which means that it cannot find a suitable spot for the new data to fit in [9]. Figure 2.14 illustrates the balance between underfitting and overfitting.

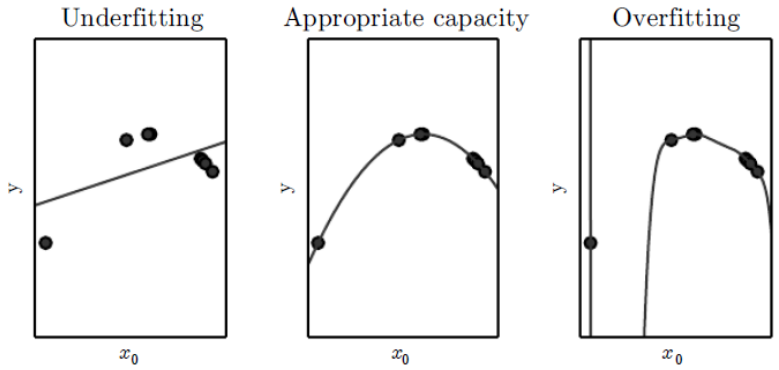


Figure 2.14: The balance between underfitting and overfitting, retrieved from [9].

As we can see, the optimal solution is the function in the middle. In that case, the function seems to fit the data well as opposed to the function on the right. The right function actually fits the data really well, but the function is all over the place. Even though the cost function could be very close to zero, the result could be problematic. Especially when the function is exposed to data it has not seen before, it would have a high error. The function in the middle have a lower probability of obtaining a high error when exposed to unknown data, due to it having a much lower variance as a function.

Regularisation

Regularisations is used to reduce the magnitude, or value, of parameters. Works well when one have a lot of features. In Andrew Ng's online course about machine learning[5], he uses an example where the optimal solution is a function close to a quadratic function, like in Equation 2.9, to predict the price of a house.

$$\theta_0 + \theta_1 x + \theta_2 x^2 \tag{2.9}$$

In another case, a quartic function like Equation 2.10 would also fit the data well, but it would overfit. In this case, a good way to generalise the function would be to penalise θ_3 and θ_4 so they are close to zero. Then, one would be left with approximately a quadratic function.

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 \tag{2.10}$$

To add regularisation one could, in general and given a cost function, add a *regularisation term*. This term would be added at the end of the function to penalise all the parameters like shown below, with λ as the regularisation parameter.

$$\begin{aligned} J(\theta) &= \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \\ \Rightarrow J(\theta) &= \frac{1}{2m} \Big[\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2 \Big] \end{aligned}$$

One have to be cautious with this parameter. Andrew explains that this parameter, if too high, would penalise all the parameters $\theta_1, \ ... \theta_n$ and as a result, the cost function could wind up as $J(\theta) = \theta_0$, e.g. a constant.

[5] `https://www.coursera.org/learn/machine-learning`

2.1.7 Evaluation

K-fold cross validation

Cross validation is an evaluation method often used in machine learning. It consists of splitting the training portion of the dataset into k disjoint subsets and train and validate the specified model on these new datasets. Using this approach, each run will give different results, and these can be combined in some way to produce a single output. The final, fully trained model can then be used on the original test set. As shown in Figure 2.15, the original training data is split into four folds. For each iteration, one fold is kept as validation data and the rest is used as training data.

Figure 2.15: 4-fold cross validation

Performance metrics

When measuring the performance of a machine learning algorithm, we need to use a performance metric that is able to measure how well the algorithm did. A popular metric is *accuracy*. Accuracy simply measures how many of the given examples the classifier labelled correctly.

$$Accuracy = \frac{Correct\ predictions}{Total\ number\ of\ predictions}$$

Accuracy works well when the distribution between class labels is close to equal. E.g. if a classifier only predicts one out of two labels in a dataset with a 50/50 distribution of class labels, the accuracy would be 50%, but if the class label distribution in the dataset is 99/1, the accuracy will be 99%. [6]

[6] `https://bit.ly/2x2e4D4`

Table 2.3: Performance measures.

Measure	Description	Formula
True positive rate	Also called sensitivity or recall. The fraction of positive examples predicted correctly	$TPR = \frac{TP}{TP+FN}$
True negative rate	Also called specificity. The fraction of negative examples predicted correctly	$TNR = \frac{TN}{TN+FP}$
False positive rate	The fraction of negative examples predicted as positive	$FPR = \frac{FP}{TN+FP}$
False negative rate	The fraction of positive examples predicted as negative	$FNR = \frac{FN}{TP+FN}$
Precision	The fraction of positive records among those that are classified as positive	$P = \frac{TP}{TP+FP}$
F1-measure	Precision (P) and recall (R) in a single number. Harmonic mean.	$F1 = \frac{2RP}{R+P}$

In the cases where the distribution between class labels in binary classifiers is uneven, the accuracy might not be sufficient to determine whether a classifier is performing well or not. In [14], the authors mention a set of alternative measurements that work well and in Table 2.3 we can see their explanations.

Classification errors

Even if the overall performance of a classifier is good in terms of the accuracy on the test set, the accuracy on its own is not sufficient to determine whether or not the classifier performs well. If all of the mistakes are what is known as *type II errors*, the performance of the classifier can still be considered bad. Type I and type II errors are the two types of classification errors in machine learning. Type I errors are referred to as false positives and type II errors are referred to as false negatives. In our case, we want to set our focus on correctly rejecting TECs. Our positive classification will then be a correct rejection and failing to do so will result in a type II error. Rejecting a TEC that should be approved will result in a type I error and should be sent to manual processing. This means that a false negative in our case would be failing to reject a TEC and returning money to someone that should not have received it. Figure 2.16 shows the combinations of truth and decisions and their outcomes (also called a confusion matrix). We need to find a model where we can, hopefully, eliminate type II errors and minimise the amount of type I errors. Another example of a type I error can be raising an alarm, even though there is no emergency whereas the type II

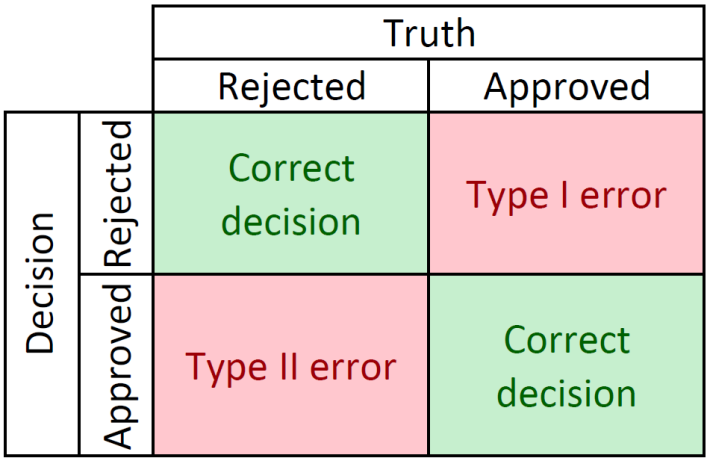


Figure 2.16: Classification errors

error equivalent would be failing to raise an alarm in the case of an emergency. A well performing classifier will have a good balance between the type of classification errors combined with a high accuracy on the test set, although the fewer classification errors, the better.

2.2 Frameworks

Deep learning is a field in growth, and there are several frameworks created to support the creation and training of models for deep learning. We hope to give a short introduction to some of them in this section.

2.2.1 TensorFlow

TensorFlow[7] is an open-source framework that was originally developed by the Google Brain Team. It is designed to work in multiple programming languages and comes in two different distributions, one for calculations on Central Processing Units (CPUs) and one for calculations on Graphical Processing Units (GPUs). Currently, TensorFlow provides Application Programming Interfaces (APIs) for Python, C++, Java and Go, although Python is the only language covered by their API stability promises. In the Python language, there are multiple different ways of using the library. At its core, it is possible to build a network using custom variable and constant objects together with mathematical operations, or use their `tf.layers` API. This API have implemented the most popular neural network layers that can easily be created to construct a network. Some good examples of this is convolutional layers, LSTM-cells and dense layers as well as operations like

[7] `https://www.tensorflow.org/`

dropout, flatten and pooling operations like max pooling and average pooling.

TensorFlow also support the usage of TensorBoard. TensorBoard is the visualisation tool that is part of TensorFlow, and used to serialise and visualise data. TensorBoard can also provide an idea of the structure of a neural network and the flow of data. If the correct data is provided TensorBoard can also provide debugging to a network. An example graph visualised in TensorBoard is shown in Figure 2.17. Here we can see the flow of the data going into the input layer and moving through two hidden layers, a dropout layer and a loss function.

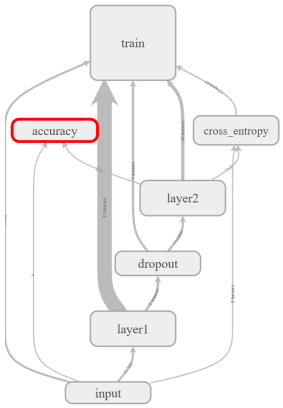


Figure 2.17: Graph visualised in TensorBoard.

Testing

As we had little to no experience in this field when starting our thesis, we used a set of data from the Internet page Kaggle [12] to train and test a simple neural network. First, we used the high level API to train and test the data, and afterwards, we used the low level API and obtained almost as good results with only a 0.1% accuracy difference. As we believed there was no reason to expect the high level API to support whatever we may need, the experience with the lower level API may be useful. Neural networks use what is called tensors as inputs and outputs. A tensor, in relation to neural networks, is an array of numbers arranged on a regular grid with a variable number of axes. By utilising large amounts of data for weighing tensors and testing predictions, neural networks are proving themselves to be a powerful tool for pattern recognition and certain classification tasks. There are multiple examples of neural networks outperforming humans. A good example is the computer program AlphaGo Zero that after training against itself for 40 days surpassed any previous version of AlphaGo[8], and thereby becoming the best computer program in the world at the game of Go.

[8]Alpha Go have been out in 4 version, 5 if AlphaZero is counted

2.2.2 MxNet

In January 2017 MxNet[9] was taken into the Apache Incubator program. The development is led by the Apache Software Foundation and it is open-source. The framework, used to train and deploy deep neural networks, supports fast model training, good scalability and is flexible in the programming model. Several programming languages is supported, where some of the more widely used ones are Python and C++. MxNet is also the deep learning frameworks favoured by Amazon Web Services (AWS), and is as of now supported by several other computer related companies. The frontend is supported by a backend in C++ for optimising the usage of GPUs and CPUs. Their webpage not only include APIs for Python, Scala, R, Julia, C++ and Perl, but also links to a model zoo with off-the-shelf pre-trained models, several example projects and tutorials. MXNet at the time of writing uses the NVidia cuDNN 6 and NVidia CUDA 8 libraries for deep learning and GPU development, and also supports Python, that it is written in, Scala, R, Julia, Perl and C++. The MxNet framework have been shown to scale close to linearly with multiple GPUs or CPUs. It is built to be flexible, as both imperative and symbolic programming may be used. MxNet also supports early stopping.

2.2.3 PyTorch

PyTorch[10] is a Python implementation of the widely popular Torch[11] framework written in Lua. Torch supports a wide variety of machine learning algorithms that are implemented with GPUs in mind. One of the few advantages with PyTorch is its ability to use imperative programming. This means that every line of code actually executes a computation, instead of the alternative which consists of performing a setup process and finally executing the entire program in a single go. This makes it easy to debug, as the developer can know exactly what line of code a potential error occurs on.

2.2.4 Keras

Some low-level and default APIs for machine learning frameworks can be difficult to understand. Keras[12] is an open-source framework capable of running on top of TensorFlow, CNTK[13] and

[9] https://mxnet.incubator.apache.org/
[10] http://pytorch.org/
[11] http://torch.ch/
[12] https://www.keras.io
[13] https://github.com/Microsoft/CNTK

Theano[14]. Keras is built in Python and seeks to add more user friendliness by introducing modularity to the creation of a neural network model. The author describes Keras as an API designed for human beings, not machines and have built Keras by putting the user experience in front and center. Keras introduces a `Sequential` model, which can be used to seamlessly add more layers to the network by simply adding a new instance of a layer object into the model. In Listing 2.1, we can see a comparison in how the layer of a neural network is built in TensorFlow and Keras. TensorFlow has its own low-level API as well as a high-level API in the `tf.layers` module that is built to make TensorFlow easier to use.

Listing 2.1: Creating a neural network layer in Keras and TensorFlow

```
import tensorflow as tf
def nn_layer_tf(input):
    weights = tf.Variable(tf.truncated_normal(SOME_SHAPE, stddev=0.05))
    biases = tf.Variable(tf.constant(0.05, SOME_LENGTH)
    preactivate = tf.matmul(input, weights) + biases
    return(tf.nn.relu(preactivate))

def nn_layer_tf_layers(input):
    conv1 = tf.layers.conv2d(inputs=input, ..., activation=tf.nn.relu)
    pool1 = tf.layers.max_pooling2d(inputs=conv1, ...)

from keras.models import Sequential
from keras.layers import Dense
def nn_layer_keras(input):
    model = Sequential()
    model.add(Dense(SOME_LENGTH, input_shape=SOME_SHAPE, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
```

When it comes to training the model, the difference between Keras and TensorFlow gets even more apparent as we can see in Listing 2.2. With the low-level TensorFlow API, one needs to use an open session and run specific layers and provide a dictionary with tf.Variable objects and set the values of these variables. The optimiser needs to be created by creating an instance of the optimiser one wants and use that instance to minimise the cost function that specified. TensorFlow also provides an optional `Estimator` class that can be used. This class implements functions like `train`, `predict` and `evaluate`. With Keras, one simply runs a `compile` function where the optimiser and loss functions are sent as inputs, either as object instances or as string values. Finally, Keras implements a `fit` function where the data is provided.

2.3 Intelligent process automation

In later years intelligent process automation have become more relevant, due to the decrease in effort needed for results, the improved cost efficiency, the enhanced productivity and freeing human

[14] `http://www.deeplearning.net/software/theano/`

Listing 2.2: Training a model in Keras and TensorFlow

```
def tf_model():
    x = tf.placeholder(tf.float64, [None, input_dim], name='x-input')
    y_ = tf.placeholder(tf.int64, [None], name='y-input')
    hidden1 = nn_layer(x, input_dim, hidden_dim, 'layer1')
    dropped = tf.nn.dropout(hidden1, dropout)
    y = nn_layer(dropped, hidden_dim, 2, 'layer2', act=tf.identity)
    loss = tf.losses.sparse_softmax_cross_entropy(labels=y_, logits=weighted_y)
    optimizer = tf.train.AdamOptimizer(options.get('learning_rate', 1e-4)).minimize(loss)
    return loss, optimizer

def tf_train_1():
    _, optimizer = tf_model()
    summary, _ = sess.run(optimizer, feed_dict={
        clf.x: train_x, clf.y_: train_y, clf.dropout: 0.9
    })

def tf_train_2():
    loss, optimizer = tf_model()
    estimator_spec = tf.estimator.EstimatorSpec(loss = loss, train_op=optimizer)
    estimator = tf.estimator.Estimator(model_fn=estimator_spec, ...)
    dataset = tf.data.Dataset.from_tensor_slices((dict(train_x), train_y))
    input_fn = dataset.shuffle(1000).repeat().batch(batch_size)
    estimator.train(input_fn=input_fn)

def train_keras():
    model = Sequential()
    model.add(Dense(dense_length, activation = dense_activation))
    model.add(Dropout(0.9))
    model.add(Dense(num_classes, activation = out_activation))
    model.compile(optimizer=Adam(lr=1e-4), loss='sparse_categorical_crossentropy', metrics=['↩
        accuracy'])
    model.fit(x = train_x, y = train_y, epochs = epochs, batch_size = batch_size)
```

resources. Berruti, Nixon, Taglioni and Whiteman [2] from McKinsey[15], explains how to utilise key computer science technologies to "enhance productivity and efficiency, reduce operational risks, and improve customer experience." Berruti et al divides the intelligent process automation into five core technologies: robotic process automation (RPA), smart workflow, machine learning/advanced analytics, natural language generation (NLG) and cognitive agents. An RPA utilises a user interface to perform repetitive, menial, and often administrative, rule-based tasks. One big upside with an RPA is that since it utilises already existing software and functions, there are no need to change backend or processes, so once the robot is trained it will perform its task. In some ways RPA is a low tier artificial intelligence, as it only learns from watching a task being performed repeatedly, making it limited to only being able to perform tasks it have already "seen". NLG modules are used to produce prose from data, making it easier for humans to understand the context of the data. In 2002 Evans, Piwak and Cahill [7] called NGL "the linguistic part" of a communicative output system.

[15] https://www.mckinsey.com/

Chapter 3

Problem statement and Overview

The process of manually approving or rejecting travel expense claims (TECs) is time consuming and it is performed by either a TEC professional or by a superior in an organisation. We want to explore and find out if there is a possibility of automating this process and potentially save time and money that can be spent elsewhere.

3.1 SAP Workflows

With 600,000 TECs in 2017, the approval process is very time consuming as HR have to go through each and every one. If one or multiple parts of this process can be automated with machine learning algorithms of different kinds, one could potentially reduce the amount of time spent drastically. Techniques like deep learning and neural networks have shown great potential in solving automation tasks in the past, so maybe they could help us too?

The purpose of this thesis will be to explore the different parts of the approval process of TECs and see if any of them can be automated. We will have to see if a single model can do it by itself, or if it will be required to combine multiple different techniques to develop a solution that will work in parallel.

SAP is the world's largest business software company[1] and based in Germany. It was started by five former IBM employees in 1972 and delivers many different IT solutions to the enterprise market. These solutions include complete computer systems along with many other initiatives including some open source products like the OpenUI5 JavaScript frontend framework.

[1] `https://www.sap.com/corporate/en/company.html`

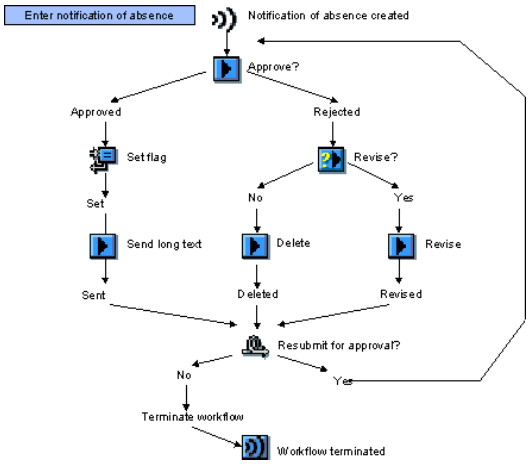


Figure 3.1: Example SAP workflow retrieved from the SAP documentation

SAP provides an interface to visually design and develop workflows that can be used to execute a series of steps in a user defined process, like the one shown in Figure 3.1. Each step can be multiple different types of *work items*, and depending on the type it is executed in different ways. The types of work items are shown in Table 3.1. The work item type *W*, the dialog work item, is the work item that is manually executed by a user utilising a graphical user interface (GUI) and the work item type *F* is the workflow itself.

Table 3.1: SAP Work item types

Type	Short description
P	Work item that represents a remote work item (Proxy WI)
E	Work item that waits for an event (wait step)
A	Work item that represents a work queue
W	Dialog work item; represents a single-step task
D	Deadline work item; notification upon missed deadline
N	Notification item (no longer used)
B	Work item for background step
F	Workflow (also sub-workflow)
C	Work item that represents a container linkage
R	Work item that represents a remote work item (http)
X	Work item that represents a block
Q	Work item that represents a dialog block

Each step in a workflow represents a *task* in the workflow. Tasks can either be performed in the foreground or background. The background tasks can be designed to do anything that does not require a human input. Tasks like these can for instance be changing states in database tables, reading memos or getting additional data. Foreground tasks include decisions to approve or reject a TEC. These tasks are executed with a dialog work item and requires an *agent* which, in case of TECs, are either HR professionals or superiors.

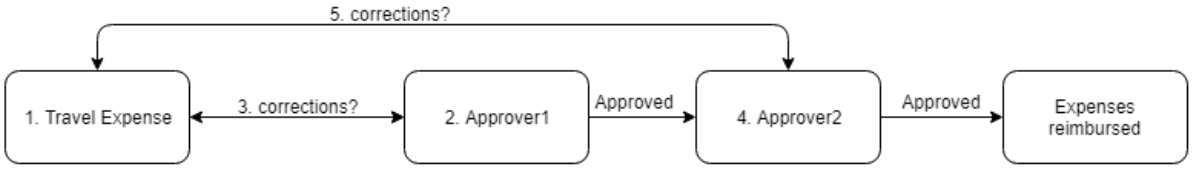


Figure 3.2: TEC workflow

3.2 Travel Expenses

3.2.1 The TEC Workflow

TECs are approved in two phases by DFØ, where phase one is performed by a human resource employee to check for errors in filling out the TEC, and phase two is a person from management approving the spending of these resources. Figure 3.2 gives an idea of the workflow a TEC goes trough to be approved.

1. An employee have completed a work related travel, fills out the required information in the travel expense and sends it for reimbursement.
2. Approver number 1 goes through the TEC checking for errors or missing information.
3. If corrections are to be made, the TEC is sent back to the employee
4. Approver number 2 goes through the TEC.
5. If corrections are to be made, the TEC is sent back to the employee.
6. Once approved by two people, the expenses are reimbursed.

The first approval step is equivalent to a checklist. The information provided in the TEC is compared to the current rules and regulations considering TECs. If everything is correct, the TEC is forwarded to the next approval step. If something is missing, the TEC is returned to the sender for corrections. When corrections have been made, the TEC will then have to be checked again, meaning any TEC that is filled out correctly will be able to get accepted by the first approver. The first approver will also be responsible for checking numbers and dates on receipts sent with the TEC. Once the first approver is satisfied, the request is then forwarded to the second approver. This is a person related to finances and management in the employee's organisation, like the employee's supervisor. After approver two have approved the TEC, the employee will be reimbursed for the expenses from the travel. Due to these steps a TEC may take very long time to be reimbursed.

After taking a closer look at the TEC workflow, it seems that the step of the first approver is the one that is easiest to automate, as it is quite rule-based compared to the second approver. At first glance, it seems like there will be a need for either an algorithm that can take a closer look at the raw data of the TEC, or maybe even some static rules. The reason that static rules might work is that there are regulations in TECs that decide whether or not certain TECs will need receipts or not. To read these receipts, we will need an Optical Character Recogniser (OCR) that will be able to extract the text for further processing. This way, the receipts can be compared to the expenses they represent, to see if there are any discrepancies.

Imagine a scenario where the employee sends a TEC in, and when the data is received at the servers it is forwarded to an automated service. This service would then determine whether the the data is sufficient to approve it, instead of approver one. If the service is sufficient certain the first phase of approval can be completed at once without waiting for a person to perform any checks on it. If not, the service sends feedback describing the needed corrections for an approval. This means that not only will the step previously done by approver one be close to instant, one may also circumvent any time a person would have to spend on performing this task. Instead of waiting for a person to manually come to a conclusion, the only time needed will be the communication between the device and the server as well as the time needed by the server to come to a decision. If the TEC is approved there will be no idle time caused by the approval step. If something is wrong, a manual approver will have to check the TEC and give the employee feedback on what corrections to make.

This thesis will take different models and methods to create this service, and test to see what will perform best in terms of:

- Accuracy
- Speed
- Adaptability (if the model is wrong or uncertain, how difficult will it be for the it to take this new information into consideration)
- Constraints (the requirements such a service will have for hardware and software).

3.2.2 TEC Regulations

The requirements for what a TEC should contain is controlled by a set of rules and regulations that are maintained by DFØ and The Norwegian Tax Administration (Skatteetaten). The challenge with these regulations is that they might change, which complicates the use of machine learning algorithms as they are used to find existing patterns in data. If the patterns change, the model will

need to be retrained to fit the new regulations better. In 2018, Skatteetaten changed some of these regulations, like a reduced reimbursement for accommodations with and without cooking options. Handling changes like these will be a challenge, as effectively, old data will be invalidated. In our thesis, we will be using data from the years 2015 to 2017 to see if there is any automation potential in the data, but to avoid inconsistent decisions, we will be excluding 2018 from our dataset.

Some of the regulations today include whether or not expense types need a paper receipt or not. In general, they all do except for the following:

- Miscellaneous paid by business
- Toll
- Toll paid by business
- Flight paid by business
- Public transport paid by business
- Public transport with 12% VAT
- Rental car paid by business
- Parking
- Parking paid by business
- Commuting expenses paid by business
- Taxi paid by business

On Skatteetaten's own website[2], the advance rates for all tax free allowances are listed as well as descriptions for certain rates. Some of these include trips with and without accommodation, extraordinary rates when using dorms, boarding houses or private accommodations as there are different rates depending on the type of accommodation that is used.

In Appendix B.3, we show a web interface where one can fill out a TEC and then send it in. The elements a TEC will have is:

- A travel period consisting of both a start- and end-date.

[2] `https://www.skatteetaten.no/satser/forskuddssatser-for-trekkfri-kostgodtgjorelse/`

- A travel reason, giving an idea of the main purpose for travelling. This can be pretty much anything, like a project meeting or a seminar on resource management.
- Travel destination(s): What is the destination? This needs to include both a country and a region. If there are several destinations, one must also include the time one leaves the previous destination.
- Travel regulative(s): Used to decide the amount of tax one has to pay and whether or not one must include an attachment, e.g. for a hotel.
- Cost distribution regarding who is paying for the trip. Under here one can also put different projects the trip was made in connection with or other accounting facts.
- Compensations to be repaid. This includes meals, compensation for driving, allowance from night shift and allowance from travelling outside of Norway added together.
- Expenses one had in connection with the trip. E.g. the hotel receipt, the taxi from or to the airport, expenses paid for hosting an event, etc.
- Attachments for expenses one had, plane tickets or hotel reservations.
- Comments to the TEC. This is an open section where one may write additional information, such at the name of the hotel or other information relevant to the trip.

Of these elements, every TEC will always have a travel period, reason, destination and regulative. The comment element is mostly optional[3], but the rest is dependant on the trip type, because the information one have to fill out in each element changes depending on the trip type. This is natural, as a four week long trip to Brazil for leading the start-up phase of a project should demand more information than a regular one day seminar in Oslo.

3.3 Data overview

The original idea for the solution is a machine learning model trained on data extracted from the SAP backend of DFØ. The amount of training data that can be extracted from there is essential for a model to classify well. The more data, the higher chances are that the model will find patterns and make good decisions on new data. Once trained on the data, the model could potentially be deployed in a solution. In Figure 3.3, a standard SAP system architecture is shown, and our model could be connected to any of the three layers shown, with advantages and disadvantages on two of

[3]If one is travelling under specific regulative types, one have to fill out the name of a hotel

them. The third layer should, for safety reasons, not be considered, as this could potentially prove to be a way into the database layer, where security is of the utmost importance[4].

Optionally, an external service could be created, and this service could be called in any of the three layers. This approach could be useful if one wanted to put the model in a cloud computing system, such as Microsoft Azure[5] or Amazon Web Services[6]. It is important to remember that, by using cloud services, there will be additional security involved as the model could be exploited or tampered with if the cloud service is compromised. E.g., if the service is exposed, one could brute force the model to find a certain TEC that would be approved without actually submitting it. One should also consider the safety of the communication channels, and ensure that the communication between the SAP system and the model service is encrypted.

Putting the model in the presentation layer would make it receive the needed data immediately, and before anything is sent deeper into the architecture, one could check if the TEC needs any changes to be accepted. It would also mean adding the model to the GUI data presented to users, adding unnecessary data to the front end, maybe even affecting responsiveness of the GUI, depending on the size of the model. Another possible problem with using the presentation layer is the higher chance of tampering with the model, due to the possibility of direct access if the design is flawed.

With the model in the application layer, one would not have to worry about the model affecting the performance of the front end, nor tampering of the model, or the data sent by the model. The data could still be processed before being sent to the database, and one could potentially utilise the most of transportation and security. The model would not have direct access to the database, compromising its security, nor would users have direct access to the model.

If the database was setup to automatically forward the data to the model upon receiving it, the model would be directly communicating with the database (or the database management system). This direct communications is a risk and an advantage at the same time, as there are fewer layers between model and database. This makes communication easier between model and database, but also means that any successful tampering with the model could directly effect the storage system. The database is something that is of the highest importance not only in the ERP system delivered by SAP, but in fact most systems with large amounts of data. The answer delivered by Noam Ben-Ami to the question "what is the importance of databases?"[7], does in fact give a good idea of this importance. The importance of keeping the database safe should be enough to discourage any thought of adding the model to the database layer.

[4] https://technet.microsoft.com/en-ca/security/gg483744.aspx
[5] https://azure.microsoft.com/en-in/
[6] https://aws.amazon.com/?nc2=h_lg
[7] https://www.quora.com/What-is-the-importance-of-databases

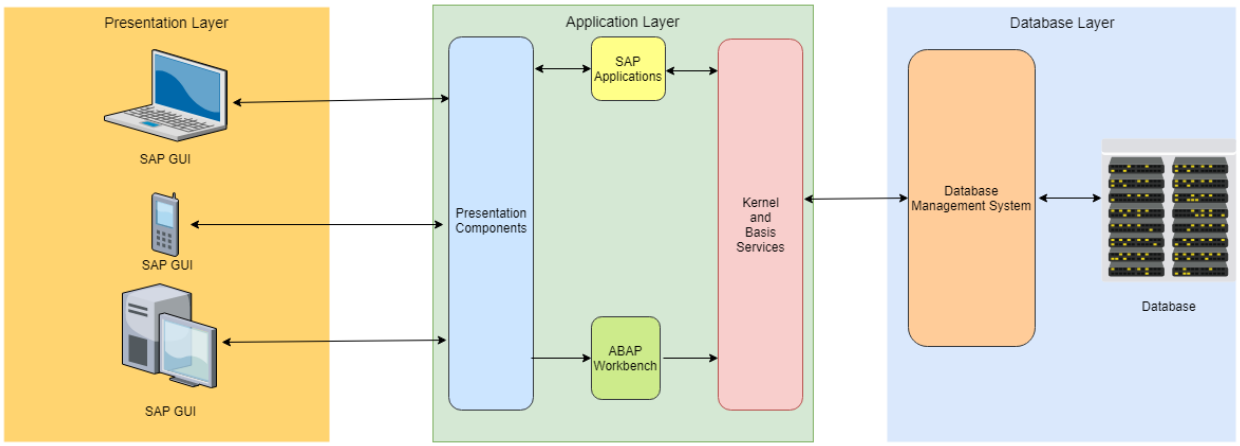


Figure 3.3: SAP System Architecture

The two options one should consider in the end is either using the application layer, or creating the model as an external service. By using the application layer, the model would be directly joined to the system and would benefit from all the safety features already created for the system. If an external service is used, the model will not be affected if the SAP system is compromised, and vice versa.

3.4 Proposed solution

By taking a closer look at the TEC workflow in SAP, we want to find a part of the process that could potentially be automated in some way. By using records of previously submitted TECs from the SAP system along with their corresponding receipts, we will have access to a large amount of data that we can use to train classifiers. The classifiers can use either feature-based machine learning algorithms or neural networks. As there are many different algorithms and architectures available, we will need to test out several algorithms to see what performs best, and if the training process takes a long time to finish, we can split the dataset into smaller subsets to speed up the process. Using smaller subsets, we should be able to gain an insight into the different models to see what would perform best in a full-scale model. After we have discovered the model that best fits our purposes, we can use all our available data in that specific model.

In addition to using classifiers, we want to explore Optical Character Recognition (OCR). An OCR can be used to scan both old receipts and new ones and extract the any text located in the image. This text can be used to see if there are any patterns in the receipts and why a TEC might have been rejected.

After finishing the final model, a suggestion for a production solution would be to implement

network communication between the application layer and a web service that utilises the model. To prevent eavesdropping, the web service should be located internally behind the same firewall as the backend system itself. Furthermore, should the web service be compromised, or if someone managed to force it to perform tasks outside of its specified area, some sort of IP filtering could be used. Finally, the database tables should not be accessed directly by the model.

To see how effective the proposed solution is, we will have to employ an evaluation method that allows us to measure the classification speed, the prediction correctness and the adaptability of the model to see if the model performs within acceptable levels. The objective we are trying to complete is partly achieved by an RPA already created by DFØ that, in January 2018, was capable of approving TECs as long as they had no attachments. However, this is still not comparable to our objective, as we want to be able to process any TEC, with or without attachments. Model adaptability will also be a concern because approval grounds of TECs change over time, as mentioned earlier.

Chapter 4

Classifiers

In this chapter, we have extracted raw data from the SAP backend database that will be used to construct a numerical dataset. This dataset will be used to explore several types of classifiers, both feature-based and neural networks.

4.1 Exploring the data

Exploring data and constructing the ideal dataset will be a large part of our thesis. As we do not know much about the data except for the sections it is divided into, we will need to figure out what attributes we will need. As each table has many columns, we expect that a few of them will be irrelevant to us. After a meeting with an HR professional employed by DFØ, we established some ground ideas. To be able to classify examples, we will need the following:

- The header table
- The cluster table
- The workflow tables
- The company code of the employee

The header table contains the core data for the TEC, including some crucial attributes that we will need, like the travel regulative and some that we can filter out. This regulative specifies the type of TEC, like a day trip or a travel with accommodation. The value of the regulative can also specify that the entry in the table is an expense reimbursement or a travel request, so we will need to filter

these entries out, as they are not a part of our thesis. Even though they are in the same table and are similar, they are approved on different grounds, so they could introduce unnecessary noise to our dataset. Like we explained in Section 3.2.1, the reason for the regulative being so important is that it specifies the requirement of additional information, like paper receipts.

Techopedia describes cluster tables as follows[1]:

> Cluster tables are special types of tables present in the SAP data dictionary. They are logical tables maintained as records of the normal SAP tables, which are commonly known as transparent tables. A key advantage of using cluster tables is that data is stored in a compressed format, reducing memory space and the landscape network load for retrieving information from these tables.

I.e., we will need the cluster table to retrieve the rest of the details for each TEC.

The workflow tables will be required to filter out any TECs that have been rejected by approver two. This is because a TEC can be rejected if it was sent to the wrong person for approval, or if the employee has requested refunds outside an agreement with his/her superior. This means that even though the TEC was correctly entered, it was still rejected, and this will introduce inconsistent approval grounds to our dataset.

Finally, we will need the company code of the employee. This is because some customers of DFØ approve TECs on different grounds than others.

4.1.1 Extracting the data

The raw data for TECs are located in database tables with a `PTRV` prefix, but these tables are not always easy to understand. The cluster table simplifies the process of retrieving data for each TEC, but each table in the cluster is not designed to show information for multiple TECs. To counter this, we will need to extend each table type with a travel identifier that can be used as a key when we are going to merge the data. The PTRV tables includes the travel id, so when extracting, we will add this key to each entry of the cluster tables. An example of a local extension of a type is shown in Listing 4.1. We define a new type with a travel id, then include the entire type that it is supposed to extend. We use the naming convention `ty_s_{name}` and `tty_s_{name}` for structure types and table types respectively.

[1] `https://www.techopedia.com/definition/28648/cluster-table-sap`

Listing 4.1: Local extension of a type

```
TYPES:
  BEGIN OF ty_s_deductions,
    tripno TYPE reinr.
        INCLUDE TYPE bapitrvded.
TYPES: END OF ty_s_deductions.
```

A cluster table cannot be accessed directly like regular database tables, so we will have to use an SAP function module (FM). The FM we will use is `BAPI_TRIP_GET_DETAILS` (shown in Listing 4.2), and it uses an employee number together with a travel id as an input to return all the travel details that we will be using to build our dataset. The travel id is the same as the one located in the header table. We will execute a call to the FM once for every unique entry in the header table to retrieve the details for all TECs. The retrieved details will have to be casted to the newly extended table types, so the id can be added to each line. This casting is shown in Appendix apx:typeCasting. The tables we will export are:

- Framedata (contains core data like departure dates, purpose, location and country)
- Addinfo (additional information about registered receipts)
- Amounts
- Deductions
- Mileage
- Receipts
- Stopover
- Emp_info (contains the company code of the employee)

When the details are converted, the details from each table will be appended to a corresponding output table that will contain all details for all TECs, but before we can do this, we will need to check if the TEC has been approved. If it has not, we will do a look-up in the workflow tables to find the status of the workflow item. This status is found by looking for certain workflow tasks. We want to look for active correction work items. If there are any, we can look for completed rejection tasks for approver one and two. We only want to keep the ones that are rejected by approver one and skip the rest. We need to look for completed tasks from both approvers as a TEC with a completed task from approver two will always have a completed decision from approver one. By performing these look-ups, we are filtering out all drafts and TECs that are rejected by approver two. When all

of the data has been retrieved we can start the final process of the extraction. First, we will convert the tables to comma-separated values (CSV), which can then be downloaded. To convert the data to CSV, we will use a custom implementation of the FM `SAP_CONVERT_TO_TEX_FORMAT`. This custom implementation allows us to use a multi-character delimiter, as the standard implementation only accepts a single character. To download, we will use the FM `GUI_DOWNLOAD`. When the extraction script is finished, we are left with nine CSV files with data from each of the tables.

Listing 4.2: Calling the trip details FM

```
CALL FUNCTION 'BAPI_TRIP_GET_DETAILS'
    EXPORTING
      employeenumber = ls_ptrv_head-pernr
      tripnumber     = ls_ptrv_head-reinr
    IMPORTING
      framedata      = ls_framedata_tmp
      status         = ls_status_tmp
    TABLES
      addinfo        = lt_additional_receipt_info
      amounts        = lt_amounts
      deductions     = lt_deductions
      mileage        = lt_mileages
      receipts       = lt_expenses
      stopover       = lt_stopovers
      emp_info       = lt_emp_info.
```

4.1.2 Creating the data set

Creating the ideal data set is difficult. Every TEC is different and have different amounts of the different tables in the cluster. For instance, a TEC can include 50 or more expenses, or it could include one. Other tables can be completely empty as not all of them are required. Because of this, we will need to find a way to merge all of this information into a single entry without losing out on any of the information that each table can provide. Our idea is to give all information tables a fixed length (e.g. 100). I.e., all TECs will always have the same number of rows in each table. This means that, should we compress a TEC into a single list of values, every TEC will have the same number of attributes. On the other hand, this would mean that the number of attributes for each TEC in the training set would be very high, over 19,000 in fact. This is way too many so we want to reduce this somehow. To reduce this length, we can calculate the standard deviation or a similar statistic, to find all the attributes that are equal and remove them entirely. An alternative is to compress the data of each table we want to include, like shown in Equation 4.1 for an $m * n$ matrix.

$$f(x) = \sum_{i=1}^{n}(\sum_{j=1}^{m} x_{i,j}) \tag{4.1}$$

Consider the table for additional receipt info, which has 60 columns. With 100 rows, this would result in 6,000 attributes for each TEC, but by using Equation 4.1, the number of attributes would be reduced to one. An approach like this will reduce the size of the data set, as well as improve the efficiency of the training process, meaning we will spend less time training our classifiers. For an approach like this to work, we will need to convert all of the values in the data set into numerical values. Currently, there are many other types of attributes in a TEC like strings and dates.

While we were waiting for the required permissions to access the database systems where the TEC data is located, we tried out some pre-processing approaches on data sets from University of California, Irvines' machine learning repository [3]. As one of the most important parts of creating our data set will be to find a numerical representation of each attribute in a data point, we will hash each value using Equation 4.2. The function takes a string x with length n as input, and uses a Python function called `ord`. The ord function returns the unicode value of any character, which can be used by our function to sum each of the values together after multiplying the value by its position in the string.

$$f(x) = \sum_{i=1}^{n} (i * ord(x[i])) \tag{4.2}$$

On a dataset consisting of weighed census data from a 1994/1995 survey this resulted in no loss of testing accuracy, but it yielded a noticeable improvement in the training time, even with each epoch of the training process and the pre-processing combined.

After receiving the permissions that we needed to access the correct backend systems, we started our extraction process. The processing script was written in Python using a Jupyter Notebook, so we could continuously look at our data and perform the computations we needed. After the processing was completed, we dumped the resulting data into sets of JavaScript Object Notation (JSON) files. By doing this, we could simply import the dataset for later use, instead of doing all of the pre-processing every time we wanted to use it. Some example data is shown in Listing 4.3. Here, there are a total of nine attributes, where seven of them are values from the raw files and the two extra are the regulative and company code of the employee. The dataset is split into three sets of features and labels. One set for training, one set for validation and one set for testing. When splitting the dataset like this, a common distribution of data points is 80/10/10 in the training, validation and testing sets respectively. Another option is that instead of doing a three way split is to split the dataset into two sets and run k-fold cross validation on the training set.

Listing 4.3: Example training data output

```
/* Features */
[
    [54, 0, 2553, 7, 94, 133, 684, 23, 60],
    [88, 54, 139, 7033, 12, 8, 33, 23, 111],
    [443, 77, 323, 79, 3, 9, 2, 799, 30]
]

/* Labels */
[
    1, 0, 1
]
```

4.1.3 Historic Data

The problem we discovered with the dataset, is that the labels for the TECs are highly unevenly distributed. Out of a dataset with 82,200 TECs, only 646 of them are rejected. A mere 0.78%. This means that we will need to implement some class weights in our classifiers, to attempt to prioritise the correct classification of the rejected TECs. For the data in the neural networks, we can also implement some data augmentation to increase the number of rejected TECs.

In an ideal situation, we would have the opportunity to extract a previous version of each TEC in the SAP client system. In the workflow tables, we can see a large number of TECs that have been rejected at some point. Our problem is that there is no way to properly retrieve this historic data. It is in these older versions that the key differences will be located, but the closest we can get is a PDF of an older version. This PDF does not include any information about whether or not a receipt has been included and it is not structured in a way that allows us to properly compare the data to the current version. If we had the ability to extract this old data in the same format as we extract the latest version, we would have access to a vast number of rejected TECs.

4.2 Feature-based method

In Section 2.1.2, we discussed the different machine learning tasks and a few different methods that we can use for classifying, both feature-based classifiers and neural networks. In this section, we will craft features for the feature-based classifiers as well as test them out with different models.

4.2.1 Classification

Feature selection

As explained in Section 2.1.4, selecting the right features is crucial to achieve good performance of a model. To know what features to look for, we will need to explore our data set. We explore our data to see if we can discover any clear and obvious patterns between our attributes and the output label, as well as discovering attributes that have no helpful information. Examples of this is an attribute whose values are evenly distributed across all classes, so in our case that will be a 50/50 distribution between approved and rejected, as we will implement a binary classifier.

To speed up the process of extracting the data, we will only extract data from 2016 and continue to build the data set later on. This way, we can start by getting an idea of what the data set will look like. We want to find patterns in the data, to see if there are any attributes we can leave out or if the distribution between approved and rejected TECs is insufficient. As the backend system consists of multiple separate client systems, we will start by extracting data from a single client and later combine data from multiple clients, in order to increase the size of our data set. In the end, our goal is to combine as much data from as many client systems as possible. In our first client system, we retrieved 30,830 TECs in 2016, but when we examined the labels we discovered that only 59 of them were in fact rejected, which suggests that we will need a lot more data from other time periods and client systems to be able to create a model that will perform well.

In Figure 4.1 we can see the importances for each feature using the `feature_importances_` attribute of SKLearn's random forest classifier. Most of the results are as expected with the with amounts scoring high and stopovers not being too important. The only deviation from the expectations that we have had is the low importance of the regulative. Previously, we have emphasised the importance of the regulative as it controls many rules that must be upheld in a real life situation when requesting a refund, but that specific information may have disappeared either during the pre-processing or it may have disappeared because we are not looking into paper receipts together with this data.

Model types

As we have discussed previously, we would like to test multiple different machine learning algorithms to take a closer look at the results of each one and compare them. Using the data set we created in the previous section, we imported the training and test sets to start building our model. To create a classifier, we will create a simple Python class that as input takes a name and another

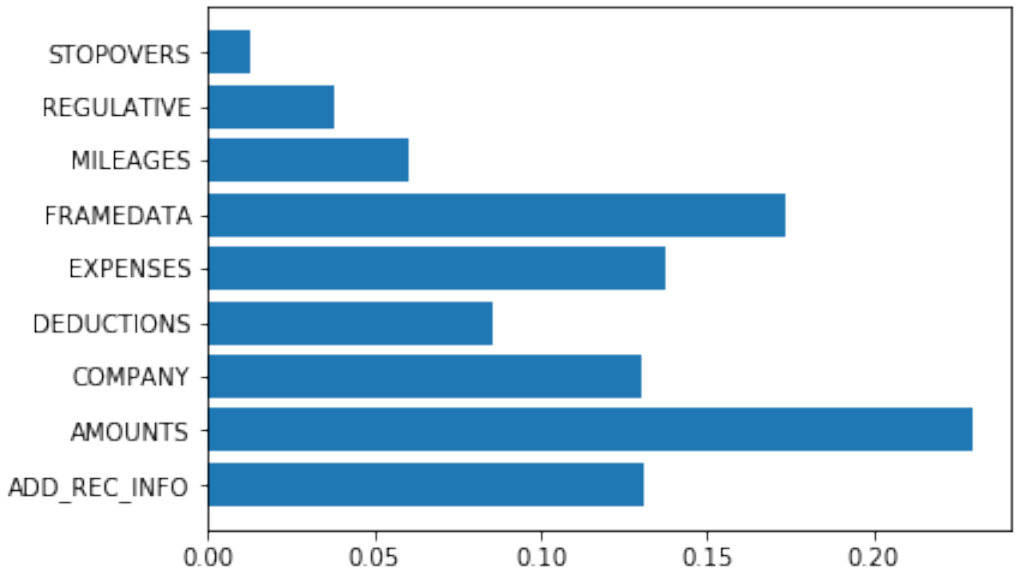


Figure 4.1: Random Forest Feature Importances

classifier from SKLearn or a similar library. This way we can keep easy track of what classifier is being used. In a similar fashion, we will create functions that use functions from the same library as the classifier itself as shown in Listing 4.4. The reason for creating functions this way instead of making them a part of the Classifier-class itself, is that we are using Jupyter Notebooks which allows us to run sections of code at a time. So if we have to correct an issue, or add some functionality in e.g. the evaluation-function, we do not have to create an entirely new instance of the class and retrain the model before testing it. Instead, we can simply re-run the segment of code housing the functions in order to apply the changes, and then execute them with the same classifier as the first input.

Listing 4.4: Embedding classifier

```
class Classifier(object):
    def __init__(self, name, model):
        self.name = name
        self.model = model

def fit(clf, X, y, log=True):
    if log: print('Training with {}'.format(clf.name))
    start = time()
    clf.model.fit(X, y)
    end = time()
    if log: print('Training took {} seconds'.format(end-start))

def score(clf, X, y, log=True):
    result = clf.model.score(X, y)
    if log: print('Accuracy: {}'.format(result))
    return result

def evaluate(clf, X, y):
    return clf.model.predict(X)
```

Training the model

To start off, we will use one of the classifiers we discussed in Section 2.1.4, the random forest classifier, a collection of decision trees. In Table 4.1, we have listed the features that we will be using.

Table 4.1: Chosen features for classification.

Feature	Description
Additional receipt info	Contains all additional information about a registered expense, e.g. if a paper receipt is added or not.
Amounts	Total reimbursement amount
Deductions	Contains day-wise information about deductions for food and accommodations
Expenses	Registered expense types with corresponding amounts.
Framedata	Contains core information like when the travel begins and ends, the reason for traveling, location, country etc.
Mileages	Information about physical travel like driving from one destination to another, fuel prices, etc.
Company code	SAP company code of the employee.
Travel regulative	SAP travel regulative.
Stopovers	Information about additional stops during the travel.

We initialised the classifier itself with all the default values, except for the `n_jobs`-parameter, which is set to -1 in order to use all available cores on the CPU. The training process takes three seconds with data from the first client system. We used data from the years 2015 to 2017, consisting of 76,532 training examples. In Table 4.2, we see the results we achieved. Although the decision

Table 4.2: Classification results with default values

Classifier	Accuracy	Type II	Type I	Type II Rate	Type I Rate
Decision Tree	0.98577	49	68	0.89091	0.00833
Random Forest	0.99367	50	2	0.90909	0.00024
K neighbors	0.99331	52	3	0.94545	0.00037
Ada Boost	0.99343	53	1	0.96364	0.00012
Gaussian Naive Bayes	0.99331	55	0	1.0	0.0

tree achieved the lowest rate of type II errors, it also had a considerably more type I errors compared to the other classifiers, which had similar results with the random forest coming out on top.

As the most common type of error is the type II error, we calculated the probability of each output label with the function `predict_proba`, to see if there were any uncertainties in the output labels. We used these probabilities to override the output label to rejected if the model was even slightly unsure about its decision, and the amount of type II errors was reduced from 51 to 32. This means that the model was 100% sure that 32 TECs in the test set should be approved, when in fact, they were rejected. On the other hand, by overriding the output label, the amount of type I errors increased from one to 900.

Altering the training data

As we were not satisfied with the results we were getting from the different models, we wanted to take a closer look at the data set and possibly alter the data. We suspect that either some information have been lost in our pre-processing or that there were simply too many approved TECs in our training set compared to rejected ones. This results in the model not finding the patterns it needs to find to be able to correctly classify the outcome of the test set. We have tested some of the following approaches:

- Omit a percentage of the approved TECs
- Find an alternative pre-processing
- Filter out some of the attributes in the existing and alternative data set

For some time, we have suspected that the distribution between approved and rejected TECs in the data set have caused problems in the machine learning algorithms. To try and compensate for this, we tried to even out the distribution by omitting a portion of the approved TECs. The omitting process was be done in multiple steps, to see if we could find a sweet spot. We expected that there was a trade-off in this process. Our expectations was that the more approved TECs we omitted from the data set, the more the type I error rate would increase, but the type II error rate would decrease. It is this type II error reduction we want, because as we have discussed earlier, the type II error is more important to our thesis. Using a simple loop, we recreated the data set 20 times and trained a random forest model with the same parameters. The results are shown in Table 4.3.

As we expected, there is a trade-off in the amount of approved TECs and type I errors. The less approved TECs we used, the less type II errors the model had until it seemed to stop in the mid 40s.

Table 4.3: Reduction results

Modulo	Entries	Accuracy	Type I errors	Type II errors
1	65760	0.99355	2	51
2	33156	0.99367	3	49
3	22261	0.99343	4	50
4	16847	0.99294	9	49
5	13571	0.99331	8	47
6	11402	0.99258	15	46
7	9847	0.99234	16	47
8	8672	0.99185	20	47
9	7770	0.99088	29	46
10	7051	0.99075	30	46
11	6443	0.99063	30	47
12	5959	0.99075	31	45
13	5530	0.98759	56	46
14	5185	0.98929	40	48
15	4871	0.98844	49	46
16	4602	0.98674	64	45
17	4359	0.98297	97	43
18	4147	0.98589	73	43
19	3950	0.98321	94	44
20	3784	0.97956	123	45

The lowest amount of type I and type II errors was at a modulo of 18. However, if we combined this reduction with the previously discussed uncertainty override, we were able to reduce the amount of type II errors to four, at the cost of 5,670 type I errors in a test set of 8,220 examples. To see if we could find any reason to as of why the TECs were predicted as approved, we worked our way backwards to find the trip numbers for each one, so that we could examine the corresponding workflow. What we discovered was that in most cases, the lack of a receipt for an expense was the cause for a rejection. This means that the TEC had everything entered correctly, except for the lack of receipts. One of the tables in the dataset, the table for additional receipt information, has an attribute called `IMAGE_LINK`, which is empty if there is no paper receipt attached, but it might be that this information was lost in the pre-processing.

Secondly, we attempted to increase the performance of our feature based models, by trying to find a new way to represent a single TEC. We suspected that some of the valuable information has been lost when reducing the entire data set into 9 attributes. The first option we tested was that, instead of having a single attribute per file type (not including additional attributes), one TEC will have 100

mean values per file, where one mean value is the combined information of one entry per file type. This approach will result in 702 attributes for each TEC. Increasing the amount of attributes also increased the time spent training the models from approximately 2.5 seconds to 6.13 seconds, but it did not improve the accuracy of the models, which makes us believe that the compression of all the attributes from each file down to a single number did not have an impact on the information in the data set. We also attempted to re-run the algorithms with the complete, uncompressed dataset of 19,302 attributes. This resulted in a single type I error and 52 type II errors for the random forest classifier, so it seems that the full compression does not affect the accuracy of the classifiers. On the other hand, training the model took a full hour to complete. The results we achieved with 702 attributes are shown in Table 4.4. These results also correspond to what we achieved with the full, uncompressed dataset.

Table 4.4: Classifier results with less compression.

Classifier	Attributes	Accuracy	Type II	Type I	Type II Rate	Type I Rate
Gaussian Naive Bayes	702	0.02019	0	8054	0.0	0.98641
Decision Tree	702	0.98662	48	62	0.87273	0.00759
Random Forest	702	0.99355	51	2	0.92727	0.00024
K neighbors	702	0.99331	52	3	0.94545	0.00037
Ada Boost	702	0.99343	53	1	0.96364	0.00012

Finally, we wanted to test filtering out some attributes in both the original data set and the new dataset with 702 attributes. To filter out attributes, we created a function that ignored certain indices in the data set, but to find these indices we had get creative. First off, we calculated the standard deviation of all the attributes across each TEC, to see if any of the attributes had the same constant value. These attributes would have a standard deviation of zero and could safely be removed from the data set without any impact on the accuracy. The standard deviations can also be used further to filter out attributes if the value is below a certain threshold. To find such a threshold, we can simply create a dictionary that maps an index to a standard deviation and sort the dictionary by its values. To test the filtering, we used the new dataset with 702 attributes first. We filtered out an increasing amount of attributes to find a point where the accuracy started to change in either direction. When we had filtered out all but 40 of the most important attributes, the accuracy of the dataset had not increased at any point, but it had started to decrease. In the original data set, we were not able to see any improvements either, but this was expected as we figured the few attributes we had all played a role in the decision-making. It turned out that we were able to filter out the two least important attributes, reducing the number of attributes to six without any loss of accuracy.

Parameter sweeping

To find the most effective hyperparameters for machine learning algorithms, a commonly used technique is parameter sweeping. This process consists of simply testing different combinations of hyperparameters available to the classifier. Because it requires training a classifier once for each combination, the process can be time consuming depending on how long it takes to train the classifier once.

For the random forest classifier, our options are the loss function, the number of decision trees used and the max depth. To perform a sweep, we will simply perform a training process inside nested for-loops for each of the parameter types, like shown in Listing 4.5. Here, we initialise a list that will hold the hyperparameters used, and the results that we want for each combination of hyperparameters, namely the accuracy and the amounts of type I and type II errors. The random state of the classifier is also set to a static value to to keep the results from each combination consistent.

Listing 4.5: Parameter sweeping

```
sweep_results = []

for loss in ['gini', 'entropy']:
    for n_estimators in [25,50,75,100]:
        for max_depth in [None, 5, 10, 15, 20, 25, 30, 35, 40]:
            model = RandomForestClassifier(
                        n_jobs=-1,
                        criterion=loss,
                        n_estimators=n_estimators,
                        max_depth=max_depth,
                        random_state=11) #Need some constant random state
            clf = Classifier('Random Forest', model)

            fit(clf, X_train, y_train)
            accuracy = score(clf, X_test, y_test)
            predictions = evaluate(clf, X_test, y_test)
            type_1_e = failed.get(1, {}).get('count', float('inf'))
            type_2_e = failed.get(0, {}).get('count', float('inf'))

            sweep_results.append([clf.name, loss, n_estimators, max_depth,
                round(accuracy, 5), type_1_e, type_2_e])
```

Table 4.5 shows the result of our parameter sweep with the random forest classifier sorted by type II errors, then type I errors. The parameter sweep was executed after the optimal modulo was found when reducing the amount of approved training examples. The results indicate that in general the entropy loss function works better in most cases except when the max depth is limited beyond a threshold.

Table 4.5: Random Forest parameter sweep results

Entries	Rejected	Loss	Trees	Depth	Accuracy	Precision	Recall	F1	Type II	Type I	Type II rate	Type I rate	TNR
4147	517	entropy	50	30	0.98662	0.995	0.99162	0.99331	41	69	0.42708	0.00838	0.57292
4147	517	entropy	50	30	0.98662	0.995	0.99162	0.99331	41	69	0.42708	0.00838	0.57292
4147	517	gini	50	25	0.98382	0.995	0.98886	0.99192	41	92	0.42708	0.01114	0.57292
4147	517	gini	50	25	0.98382	0.995	0.98886	0.99192	41	92	0.42708	0.01114	0.57292
4147	517	entropy	50	25	0.98698	0.99488	0.9921	0.99349	42	65	0.43299	0.0079	0.56701
4147	517	entropy	50	25	0.98698	0.99488	0.9921	0.99349	42	65	0.43299	0.0079	0.56701
4147	517	entropy	50	None	0.98662	0.99488	0.99174	0.99331	42	68	0.43299	0.00826	0.56701
4147	517	entropy	50	None	0.98662	0.99488	0.99174	0.99331	42	68	0.43299	0.00826	0.56701
4147	517	entropy	50	40	0.98662	0.99488	0.99174	0.99331	42	68	0.43299	0.00826	0.56701
4147	517	entropy	50	40	0.98662	0.99488	0.99174	0.99331	42	68	0.43299	0.00826	0.56701
4147	517	entropy	50	35	0.9865	0.99488	0.99162	0.99325	42	69	0.43299	0.00838	0.56701
4147	517	entropy	50	35	0.9865	0.99488	0.99162	0.99325	42	69	0.43299	0.00838	0.56701
4147	517	gini	25	15	0.98516	0.99488	0.9903	0.99258	42	80	0.43299	0.0097	0.56701
4147	517	gini	25	15	0.98516	0.99488	0.9903	0.99258	42	80	0.43299	0.0097	0.56701
4147	517	gini	50	20	0.98516	0.99488	0.9903	0.99258	42	80	0.43299	0.0097	0.56701
4147	517	gini	50	20	0.98516	0.99488	0.9903	0.99258	42	80	0.43299	0.0097	0.56701
4147	517	gini	50	None	0.9837	0.99488	0.98886	0.99186	42	92	0.43299	0.01114	0.56701
4147	517	gini	50	None	0.9837	0.99488	0.98886	0.99186	42	92	0.43299	0.01114	0.56701
4147	517	gini	50	30	0.9837	0.99488	0.98886	0.99186	42	92	0.43299	0.01114	0.56701
4147	517	gini	50	30	0.9837	0.99488	0.98886	0.99186	42	92	0.43299	0.01114	0.56701

Combining multiple algorithms

Instead of just relying on a single algorithm for our classification, we can combine multiple different models. We want to explore a few different approaches. This means that we will need an odd number of classifiers that will vote for an output label, and the final decision will be the label with the most votes. As different classifiers can find different patterns in the data, they will output different predictions for the test set. This can be used to our advantage. Some patterns are harder to find and therefore, a single model can get it wrong, but we hope that with multiple models, a majority of the models will get the right label.

As a starting point, we created five random forest classifiers with different loss functions and random states in order to implement a simple voting process. This approach will utilise a consensus model with similar classifiers that might have found different patterns in the data, so they might output different results. The idea is that the output label with the highest number of votes will be chosen as the final output label. The classifiers can either be trained on the same data set, or the data set can be shuffled for each classifier. To get the prediction labels of each classifier, we appended them all to a list, so they could be retrieved later. Finally, we counted the amount of times each label was predicted for the different test entries. The majority label was appended to the output result. When the final results were calculated for the classifiers with the same unshuffled dataset, we calculated the amount of type I and type II errors as before got the same amount of type I errors and one less type II error. The combined result of all the classifiers had a single type I error and 51 type II errors. An interesting result on the other hand is that one of the classifiers performs better than the combined result. This is not uncommon when using voting classifiers, as one of the

Listing 4.6: K-fold cross validation using Python and SKLearn

```
from sklearn.model_selection import KFold

def kfold_cross_validate(clf, train_x, train_y, k=5, options={}):
    results = []

    kf = KFold(n_splits=k)
    for train_idx, val_idx in kf.split(train_x, train_y):
        features = train_x[train_idx]
        labels = train_y[train_idx]
        val_x = train_x[val_idx]
        val_y = train_y[val_idx]
        fit(clf, features, labels)
        results.append(score(clf, val_x, val_y))
    return results
```

algorithms might find a better pattern than the rest.

As a second attempt, we will utilise a classifier that SKLearn has integrated in their library, the `VotingClassifier`. The VotingClassifier can be initialised in the same way as we initialised our own version earlier, with a set of other classifiers. The difference is that we only have to train the model once and the classifier will automatically train every model that has been supplied. As expected, we got the same result with this approach.

Instead of using the same data set, a different approach is to train a classifier on different segments of the dataset, e.g. using one classifier for just the expenses in the dataset and another for the entire dataset. To do this, we will have to redo some of the pre-processing on the original dataset as all the data from the expenses for each TEC has been compressed into either a single number for all expenses or a single number for each expense.

K-Fold cross validation

For our final attempt to increase the accuracy of our model, we will implement k-fold cross validation. To do this, we will use SKLearn's `KFold` class, which is initialised with the training set to be used. This training set will be split into k folds, which will be used to train and validate the dataset. We will use 10 folds to train our classifiers. In Listing 4.6 we have used the KFold class to create a function that splits a training set into k folds before it trains and validates a classifier. Here, the `fit` and `score` functions can be any set of functions that train and score a classifier.

With the feature-based configurations shown in Table 4.6, we can take a closer look at Table 4.7 that shows the results with different classifiers. We ran all the configurations with and without k-fold cross validation and surprisingly, we achieved better results across the board without it. A reason for this can be that the rejected samples are unevenly distributed across each fold, causing

Table 4.6: Feature-based configurations

#	Entries	Rejected	Attributes	Classifier
1	65760	572	9	Random Forest
2	4147	572	9	Random Forest
3	65760	572	6	Random Forest
4	4147	572	9	Voting
5	4147	572	9	RF/Ada voting
6	65760	572	9	Decision Tree
7	4147	572	9	Decision Tree
8	65760	572	9	Gaussian Naive Bayes
9	4147	572	9	Gaussian Naive Bayes
10	65760	572	9	K neighbours
11	4147	572	9	K neighbours
12	65760	572	9	Ada Boost
13	4147	572	9	Ada Boost

the classifiers to overfit. The results we achieved show that by using the data found in the database-tables alone, it is difficult to achieve a low type I and type II error rate at the same time. The only way we managed to reduce the amount of type II errors was to sacrifice accuracy by overriding the label if the classifier had uncertainty in its classifications. This indicates that to get accurate predictions, we will need a combination of multiple models including looking into the actual paper receipt of each TEC as it is essential to approving a TEC.

Table 4.7: Feature-based results

#	KFold	Accuracy	F1	Type IIs	Type Is	Type II rate	Type I rate	Precision	Recall	TNR
7	Yes	0.89891	0.94662	35	796	0.63636	0.09749	0.99527	0.90251	0.36364
2	Yes	0.98358	0.99171	43	92	0.78182	0.01127	0.9947	0.98873	0.21818
4	Yes	0.98516	0.99251	44	78	0.8	0.00955	0.99459	0.99045	0.2
6	Yes	0.98479	0.99233	47	78	0.85455	0.00955	0.99422	0.99045	0.14545
13	Yes	0.98382	0.99184	47	86	0.85455	0.01053	0.99422	0.98947	0.14545
11	Yes	0.96083	0.98	48	274	0.87273	0.03356	0.99395	0.96644	0.12727
1	Yes	0.99355	0.99676	51	2	0.92727	0.00024	0.99379	0.99976	0.07273
5	Yes	0.99355	0.99676	51	2	0.92727	0.00024	0.99379	0.99976	0.07273
9	Yes	0.98771	0.99382	51	50	0.92727	0.00612	0.99375	0.99388	0.07273
10	Yes	0.99343	0.9967	52	2	0.94545	0.00024	0.99367	0.99976	0.05455
12	Yes	0.99343	0.9967	53	1	0.96364	0.00012	0.99355	0.99988	0.03636
3	Yes	0.99234	0.99615	53	10	0.96364	0.00122	0.99354	0.99878	0.03636
8	Yes	0.99319	0.99658	55	1	1.0	0.00012	0.99331	0.99988	0.0
7	No	0.9017	0.94817	34	774	0.61818	0.09479	0.99542	0.90521	0.38182
4	No	0.98662	0.99325	41	69	0.74545	0.00845	0.99496	0.99155	0.25455
2	No	0.98589	0.99288	43	73	0.78182	0.00894	0.99471	0.99106	0.21818
13	No	0.98479	0.99233	45	80	0.81818	0.0098	0.99446	0.9902	0.18182
6	No	0.98504	0.99246	48	75	0.87273	0.00919	0.9941	0.99081	0.12727
11	No	0.96046	0.97981	48	277	0.87273	0.03393	0.99395	0.96607	0.12727
1	No	0.99367	0.99683	50	2	0.90909	0.00024	0.99391	0.99976	0.09091
5	No	0.99367	0.99683	50	2	0.90909	0.00024	0.99391	0.99976	0.09091
9	No	0.98406	0.99196	51	80	0.92727	0.0098	0.99373	0.9902	0.07273
10	No	0.99331	0.99664	52	3	0.94545	0.00037	0.99367	0.99963	0.05455
12	No	0.99343	0.9967	53	1	0.96364	0.00012	0.99355	0.99988	0.03636
3	No	0.99234	0.99615	53	10	0.96364	0.00122	0.99354	0.99878	0.03636
8	No	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0

Listing 4.7: Model building with Keras

```
# Model building
model = Sequential()
model.add(InputLayer(input_shape=(size_flat,)))
model.add(Reshape(shape_full))

## Convolutional layers
model.add(Conv2D(
    kernel_size=5, strides=1, filters=16,
    padding='same', activation = conv_activation, name='layer_conv1'))
model.add(Conv2D(
    kernel_size=5, strides=1, filters=36,
    padding='same', activation = conv_activation, name='layer_conv2'))

model.add(MaxPooling2D(pool_size=2, strides=2))
model.add(Flatten())
model.add(Dense(128, activation = dense_activation))

# Output layer
model.add(Dense(num_classes, activation = out_activation))
```

4.3 Neural network method

We want to test multiple different types of neural network models to be able to compare them against each other. We want to look at the time it takes to train the network and how good they perform.

4.3.1 Model types

Convolutional Neural Network

The first model we will test is a Convolutional Neural Network (CNN) with two convolutional layers. The network is constructed using the Keras API for TensorFlow and we used the Sequential model, which allows us to dynamically add more layers of different types. An example of the model building is shown in Listing 4.7.

Deep Feed Forward Network

The next model we will test is a traditional feedforward network, also known as *multilayer perceptrons*. A deep feedforward network (DFFN) consists of multiple fully connected layers. These layers contain neurons that sends an output into every neuron in the next layer. This forwarding continues until the values reach the output layer. It's the addition of multiple layers – which can

also be referred to as a chain – between the input and the output that gives this model its depth, so it becomes a deep feedforward network instead of just a feedforward network [9].

Instead of using Keras as we have done with the convolutional network, we will use a regular set of mathematical operations within tensorflow to build the network. This way we can easily use TensorBoard to visualise the progression of all the values from start to finish in the training process. We will discuss TensorBoard in more detail later in this chapter.

To build each hidden layer in the network, we will create a function that initialises weights and biases to use in a matrix multiplication, which will create the input to an activation function like the rectified linear unit (ReLU). Listing 4.8 shows a function that defines and returns a single layer to be used in a neural network, as well as the creation of two layers. This code is retrieved from a tutorial that the TensorFlow team wrote on how to use TensorBoard[2], which is why the function includes the `tf.summary` module. The hidden layers built by this function can be sent as inputs into the other layers in the TensorFlow API in order to build the complete network.

Listing 4.8: Neural Network Layer

```
def nn_layer(input_tensor, input_dim, output_dim, layer_name, activation=tf.nn.relu):
    with tf.name_scope(layer_name):
        with tf.name_scope('weights'):
            weights = weight_variable([input_dim, output_dim])
            variable_summaries(weights)
        with tf.name_scope('biases'):
            biases = bias_variable([output_dim])
            variable_summaries(biases)
        with tf.name_scope('Wx_plus_b'):
            preactivate = tf.matmul(input_tensor, weights) + biases
            tf.summary.histogram('pre_activation', preactivate)
        activations = activation(preactivate, name='activation')
        tf.summary.histogram('activations', activations)
        return activations

hidden_layer_1 = nn_layer(x, 784, 10, 'layer1')
hidden_layer_2 = nn_layer(hidden_layer_1, 500, 10, 'layer2', activation=tf.identity)
```

4.3.2 Training the models

Although our training examples are not exactly images, we will still attempt to generate some additional examples, as one of the major issues we have with out dataset is not having enough rejected TECs. We will start off by using numpy to reshape our rejected training examples into the same shape as the neural network expects, e.g. an $m * n$ matrix for the CNN. We can then be apply operations to flip, mirror or something similar, that would still render the ”image” understandable. For each of these operations, we double the initial amount of rejected TECs.

[2] `https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard`

Table 4.8: Runtime results

GPUs	Batch Size	Time/Step	Time Total
0	64	$3ms$	$169s$
0	64	$3ms$	$168s$
1	64	$233\mu s$	$15.8s$
1	64	$227\mu s$	$15.42s$
1	64	$229\mu s$	$15.29s$
1	64	$226\mu s$	$15.35s$
2	64	$262\mu s$	$17.70s$
2	64	$262\mu s$	$17.74s$
2	64	$260\mu s$	$17.58s$
2	128	$180\mu s$	$12.33s$
2	128	$176\mu s$	$12.08s$
2	128	$176\mu s$	$12.08s$
2	160	$161\mu s$	$11.06s$
2	160	$152\mu s$	$10.49s$
2	160	$162\mu s$	$11.28s$
2	160	$157\mu s$	$10.52s$

For the CNN, we see significant improvements in time spent on training the model because of the convolution operation, and the fact that all the values are numbers. We started off by training the model using the CPU, an Intel Core i7 7700K. We configured the training process to run two epochs on the training data, a batch size of 64 and an `Adam` optimiser with a learning rate of 0.001. The training process used on average $3ms/step$, which resulted in a total training time of 168 seconds. To compare, we have decided to test out an Nvidia GeForce GTX970 GPU that we have on hand to see how much of an improvement we can get. This time, the training process used on average $233\mu s/step$, which resulted in a training time of just 15.8 seconds, meaning training on a GPU was over 10 times faster. By adding another GPU, we achieved worse results than running on a single GPU, but it allowed us to increase the batch size from 64 to 128 and 160. These increased batch sizes further improved the runtimes to $157\mu s/step$ and 10.8 seconds in total, a $15,5$ times improvement. Table 4.8 shows the runtime of each configuration.

The trained model achieved an accuracy of 98.65%, which is slightly lower than the feature based classifier used in the previous chapter. We decided to take a closer look at the results of some predictions with the test set. What we noticed was that every single prediction resulted in an approved TEC. As we discovered in Section 4.2.1, only a mere 2.6% of the examples were rejected, so the model is not able to find the pattern for the rejected TECs. In an attempt to decrease these

Table 4.9: Configurations for Neural Networks

#	Entries	Rejected	Attributes	Classifier	Optimizer	Class Weight	Neurons
1	65760	517	9	DFF Network	Adam	[1.0, 1.0]	15
2	65760	517	9	DFF Network	Adam	[1.1, 1.0]	15
3	65760	517	9	DFF Network	GradientDescent	[1.0, 1.0]	15
4	65760	517	9	DFF Network	GradientDescent	[1.1, 1.0]	15
5	65760	517	9	DFF Network	RMSProp	[1.0, 1.0]	15
6	65760	517	9	DFF Network	RMSProp	[1.1, 1.0]	15
7	65760	517	9	Convolutional NN	Adam	[1.0, 1.0]	3x3
8	65760	517	9	Convolutional NN	Adam	[1.1, 1.0]	3x3
9	65760	517	9	Convolutional NN	GradientDescent	[1.0, 1.0]	3x3
10	65760	517	9	Convolutional NN	GradientDescent	[1.1, 1.0]	3x3
11	65760	517	9	Convolutional NN	RMSProp	[1.0, 1.0]	3x3
12	65760	517	9	Convolutional NN	RMSProp	[1.1, 1.0]	3x3

type II errors, we tried different pooling methods, as well as adding and removing convolutional layers, but nothing worked so it is possible that the problem lies elsewhere. One possibility is that the data set simply does not include enough rejected TECs to find the patterns, but another possibility is that the pre-processing that we have done have introduced too much noise, but as we discussed in the previous section, running with all of the 19,302 attributes did not improve the performance of the classifiers. However, if this is the case, we will have to try and find other options for pre-processing.

The trained DFFN achieved very similar results to the CNN in all measurements, meaning it approved all the TECs in the test set. When we implemented class weights on the other hand, the network sometimes panicked and ended up rejecting all TECs in the test set. This means that a classifier on the raw data alone is not enough to automate a process in the TEC workflow.

In Table 4.9, we can see the different configurations that were used in the neural networks and in Table 4.10, we can see the results from the different configurations we used in our neural networks. As we can see, the neural networks struggled to find the patterns in the data and to adjust its weights properly. The result was that the neural networks sometimes approved every single TEC in the test set. When we implemented class weights to try and prioritise the rejected TECs, the validation panicked and suddenly rejected the entire test set. The results we got with the neural networks further support the conclusion from the feature-based results, that we will need a combination of the raw data and paper receipts to get accurate results.

Table 4.10: Neural network results

#	KFold	Accuracy	F1	Type IIs	Type Is	Type II rate	Type I rate	Precision	Recall	TNR
12	No	0.00815	0.00294	0	8153	0.0	0.99853	1.0	0.00147	1.0
12	Yes	0.00718	0.00098	0	8161	0.0	0.99951	1.0	0.00049	1.0
7	Yes	0.00669	0.0	0	8165	0.0	1.0	0.0	0.0	1.0
9	Yes	0.00669	0.0	0	8165	0.0	1.0	0.0	0.0	1.0
10	Yes	0.00669	0.0	0	8165	0.0	1.0	0.0	0.0	1.0
11	Yes	0.00669	0.0	0	8165	0.0	1.0	0.0	0.0	1.0
12	Yes	0.00669	0.0	0	8165	0.0	1.0	0.0	0.0	1.0
8	No	0.00669	0.0	0	8165	0.0	1.0	0.0	0.0	1.0
9	No	0.00669	0.0	0	8165	0.0	1.0	0.0	0.0	1.0
10	No	0.00669	0.0	0	8165	0.0	1.0	0.0	0.0	1.0
12	No	0.00669	0.0	0	8165	0.0	1.0	0.0	0.0	1.0
11	No	0.07628	0.13272	9	7584	0.16364	0.92884	0.98475	0.07116	0.83636
8	Yes	0.19161	0.31587	14	6631	0.25455	0.81212	0.99096	0.18788	0.74545
10	No	0.66642	0.79897	26	2716	0.47273	0.33264	0.99525	0.66736	0.52727
7	Yes	0.49732	0.66275	27	4105	0.49091	0.50276	0.99339	0.49724	0.50909
7	No	0.58662	0.73862	34	3364	0.61818	0.412	0.99297	0.588	0.38182
8	No	0.60012	0.74933	35	3252	0.63636	0.39829	0.99293	0.60171	0.36364
6	Yes	0.76934	0.86935	39	1857	0.70909	0.22743	0.99386	0.77257	0.29091
9	Yes	0.92287	0.95985	47	587	0.85455	0.07189	0.99384	0.92811	0.14545
10	Yes	0.96971	0.98461	50	199	0.90909	0.02437	0.99376	0.97563	0.09091
2	Yes	0.96448	0.98191	52	240	0.94545	0.02939	0.99348	0.97061	0.05455
6	No	0.99075	0.99535	53	23	0.96364	0.00282	0.99353	0.99718	0.03636
4	Yes	0.98127	0.99054	54	100	0.98182	0.01225	0.99335	0.98775	0.01818
2	Yes	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
3	Yes	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
3	Yes	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
4	Yes	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
5	Yes	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
5	Yes	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
8	Yes	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
11	Yes	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
7	No	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
9	No	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
11	No	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
1	No	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
3	No	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
4	No	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
6	No	0.99331	0.99664	55	0	1.0	0.0	0.99331	1.0	0.0
1	Yes	0.99319	0.99658	55	1	1.0	0.00012	0.99331	0.99988	0.0
1	Yes	0.99319	0.99658	55	1	1.0	0.00012	0.99331	0.99988	0.0
1	No	0.99319	0.99658	55	1	1.0	0.00012	0.99331	0.99988	0.0
3	No	0.99319	0.99658	55	1	1.0	0.00012	0.99331	0.99988	0.0
5	No	0.99307	0.99652	55	2	1.0	0.00024	0.99331	0.99976	0.0
5	No	0.99307	0.99652	55	2	1.0	0.00024	0.99331	0.99976	0.0
4	No	0.99282	0.9964	55	4	1.0	0.00049	0.99331	0.99951	0.0
6	Yes	0.99246	0.99621	55	7	1.0	0.00086	0.9933	0.99914	0.0
2	No	0.99124	0.9956	55	17	1.0	0.00208	0.9933	0.99792	0.0
2	No	0.99027	0.99511	55	25	1.0	0.00306	0.99329	0.99694	0.0

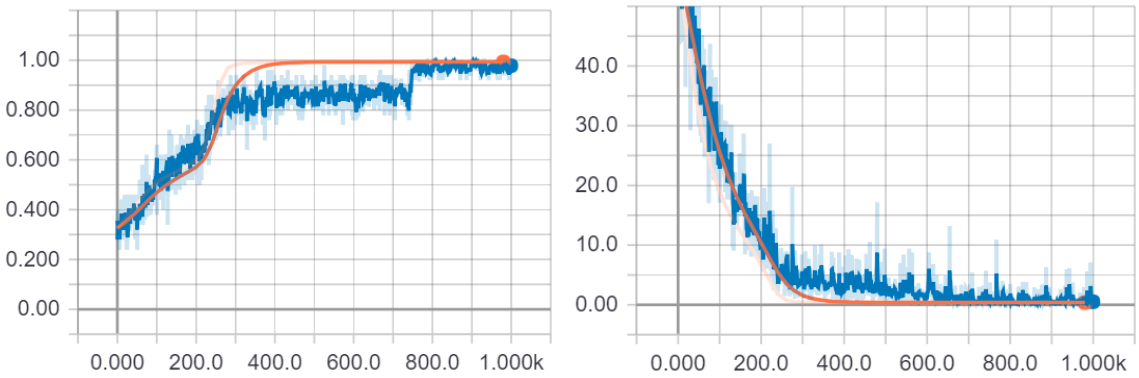
4.4 Data and Flow Visualisation

In this section we will show the structure of some networks we created, follow the flow of data and provide graphs for different training of networks for comparison using TensorBoard.

TensorBoard is configured using a `FileWriter` that logs values in the neural network to a file. It is up to the developer to choose what kind of values to log. We will start by logging some statistical values for the weights and biases as they are updated in the training process. Additionally, we will log the output from each step in each layer in the network using the TensorFlow summaries library.

To keep things tidy, we will utilise name-scopes. This way, we can separate values into different plots. For instance, for each layer in the neural network, we define a name scope for the layer itself which contains three name scopes; one for the weights, one for the biases and one for the matrix multiplications. The weights and biases in return contain name scopes for the summary of the statistics. When all the logging points have been configured, they are all merged into a single operation that can be executed at the same time as a step in the training process. When running the training process, the training step is executed together with the merged summary operation. This way, the TensorFlow session will return a result from both operations and we can use the logger to add a new summary for that specific step. Finally, the loggers must be closed.

In Figure 4.2, we see the progression of the accuracy and cross entropy loss function from the training process. Every time a batch is completed in the training process, an evaluation with the test set is executed as well, which allows us to see results at each time step for both the training set and the test set. The behaviour of the accuracy and cross entropy are as expected, with the accuracy gradually increasing and the loss decreasing as the model performs better and better the more it is trained.



(a) Accuracy over time

(b) Cross entropy over time

Figure 4.2: Visualisation output from TensorBoard. The blue graphs represents the training values and the orange graphs represent the test values.

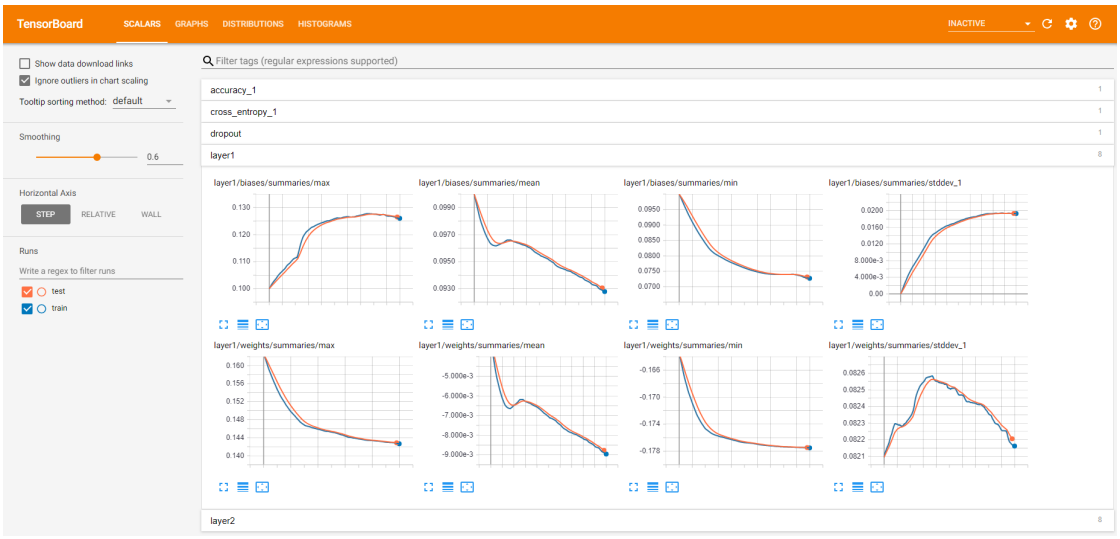


Figure 4.3: Visualisation of different scalar statistics from a hidden layer in TensorBoard

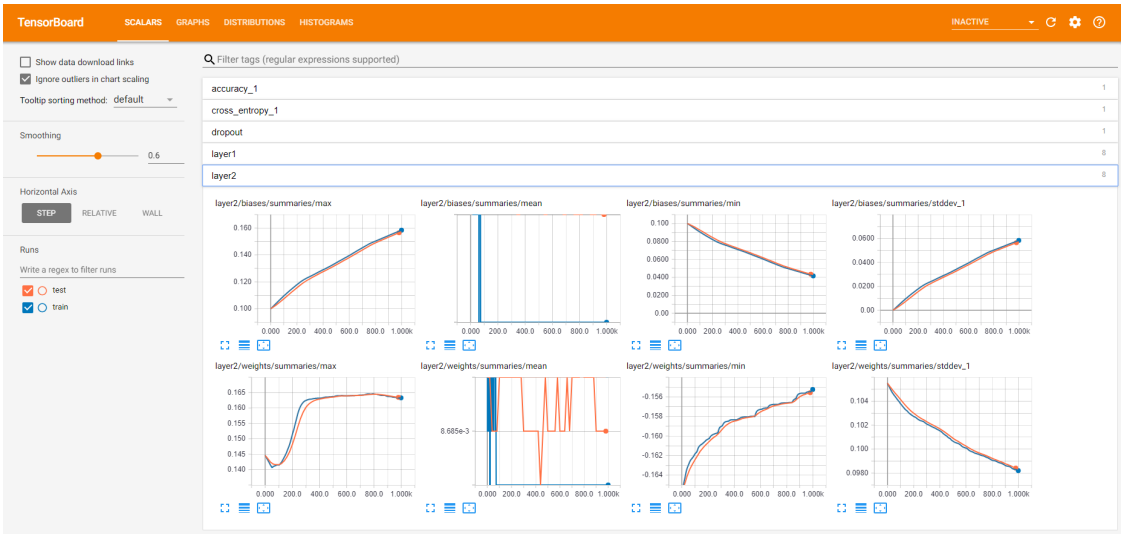


Figure 4.4: Visualisation of different scalar statistics from another hidden layer in TensorBoard

In Figures 4.3 and 4.4, we can see the values over time for the different statistics that we configured in TensorBoard for two different deep feedforward networks. In 4.4, we can see that the mean values for both the weights and biases of the second hidden layer does not change over time.

Chapter 5

Optical Character Recognition

In this chapter, we will explain Optical Character Recognition (OCR). Firstly, we will start with a short introduction to OCR and why we need it. Then, we will explain how we measured the efficiency potential of OCRs in our thesis, as well as how we will extract the data needed to use one. In Section 5.3, we will also show some code examples of how we are using Tesseract[1], an OCR maintained by Google, and some ABAP scripts for extracting data from SAP. When referring to OCR in this chapter it will be either the program used to extract text from images, such as Tesseract or OneNote 2016 (OneNote)[2], or the methodology behind them, depending on the context.

5.1 What is Optical Character Recognition?

Sometimes, it could come in handy for a machine to replicate or simulate the human senses. Examples include speech-recognition, where one could try to make a machine perform tasks based on voice commands, or sorting images using vision based systems. In later years, this replication has often been done by neural networks, but there are other ways of performing this too. OCR is used when one want to have some information readable by both humans and machines. As opposed to "on-line" character recognition, where the system recognises characters as they are written, an OCR tries to perform an "off-line" character recognition. This can for example be after the character have been printed on a piece of paper or recognising handwritten notes. In this chapter we will be using the term "reading" to refer to the task of extracting text from an image.

A TEC can have many expenses, and most of them can not be approved unless the correct receipts

[1] https://github.com/tesseract-ocr/tesseract
[2] https://bit.ly/2Jz9kts

are added to the registered expenses. This means that without any means of checking the values on the printed (or digital) receipts, one would be unable to verify the correctness of an added expense. This is where an OCR comes into play. By using an OCR, one can extract text from any image, provided that the image quality is good enough. To use an OCR we have two options: create and train our own, or use an already existing one developed by others. As this thesis does not focus on extracting text from images in the best possible way, we will utilise a pre-trained OCR. Developing our own OCR would be time consuming and using an OCR to test the feasibility of checking receipts is only a sub-task for this thesis.

There are some pre-trained OCRs that are free to use, such as SimpleOCR, FreeOCR and Microsoft OneNote's integrated OCR. We will be utilising the OCR currently maintained by Google, Tesseract. In the next section we will also be using OneNote, in order to compare the quality of image readings to Tesseracts. OneNote can not be used to perform the actual task, as it depends on a cloud service. This means sending attachments with possible personal data out, and restrictions from the GDPR would not allow us to do this. As Tesseract does not support images in the form of a PDF, ImageMagick was used to create images from PDF-files. The program `command_line.py` takes in an image and a TEC, or its expenses, before it returns a value. If the returned value is 0, the attachment corresponds to the values in the TEC or its expenses. If the attachment does not correspond to the values, it returns a value of an increasingly larger number depending on the amount of errors.

Both Tesseract and ImageMagick are open-source, free to use programs. In Appendix C there are explanations of how to install them and any dependencies they have, while a README is provided for our implementations.

5.2 Measuring OCR efficiency

To ensure that it is feasible to utilise an OCR to read receipts, we will need a way to test a measurement of success. We will be using two OCRs, Tesseract version 3.14 and the OCR employed by OneNote 2016 (will be referred to as OneNote). These two OCRs will be tested and compared in how well they are able to find dates, amounts and other relevant data in the receipts. To compare the efficiency, we manually labelled 150 attachment images. These images contained receipts from previous travels, which we can use to test the accuracy of the OCRs. Some images contained only one receipt, others several. The labelled receipts were bus tickets, flight tickets, hotel receipts, taxi fares, dinner receipts and from other similar situations. This resulted in a total of 210 receipts from the 150 attachments we could check and compare against. Three measurements were created: one for date, one for amount and one for readability and general errors occurring when reading the im-

age. The first two measurements give an idea of how many receipts that can be "verified" as correct without performing any image editing or enhancements. The last one gives an idea of errors found after "reading" an image.

Document readability and error rate were used to create an idea of how the OCRs would handle blurry images, special characters, language specific characters, and file formatting. While labelling and testing, we quickly noticed some things the OCRs cannot handle at all. Firstly, the OCRs struggled with a receipt format often used by Norwegian airport shuttles. The issue is that they contain straight lines that span the entire receipt which introduces large amounts of noise, effectively drowning the rest of the text. This will be discussed in detail later in this section, and the receipts of this type will be referred to as airport shuttle bus receipts. Secondly, the OCRs struggled when an image was rotated, usually with a tipping point of 45 degrees. This can be solved easily by simply rotating the image back to compensate. The problem is that we do not necessarily know if a rotation is the reason that the OCR failed, and even if it is, we do not know by how much. By blindly rotating an image until the OCR succeeds, we will introduce a lot of extra processing, as it becomes a matter of trial and error. Potentially, this could mean that some images will be rotated three times and processed four times without finding a date and an amount because, it was in fact a matter of image quality, and not orientation. Unfortunately we found no Python modules or any other programs that were able to perform this in an very efficient manner. One way to potentially perform less rotations is to perform a Hough transform[3] to detect if the text is horizontal or vertical in orientation. By knowing the orientation of the picture, one could potentially exclude the two rotations where the text would be vertical. The Hough transform is still computationally expensive, but this could potentially be more efficient than trying every rotation of an image.

As the OCRs fail on approximately every third receipt, one should consider an optimisation of the failure handling, because of the high potential of unnecessary processing. E.g., if the OCR finds a date in a receipt, but not the expected amount, there will be no need for rotations as there is readable text in the image already. To quantify the errors and make it easier to evaluate, we have used six different categories:

- All: The OCR failed to read the image. The receipts airport shuttle buses and rotated images dominate this category. The extracted text is either gibberish, non-existent or a series of lines.
- Mostly: The OCR barely manages to extract any useful information. The readability is low, but occasionally, words and numbers appear. This category is the smallest and is comprised of blurry images or rotations between 20 to 45 degrees.

[3] `https://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm`

- Lots: The OCR manages to read a lot of the text correctly, but it is still missing cohesion and there is a noticeable amount of errors. The images here are sometimes written weakly and one can perceive that the ink has faded slightly. Other images include cases where several receipts are bundled too closely together in a single image.
- Some: The OCR manages to extract a mostly coherent text from the image, but there are still too many errors for the text to be considered a great extraction. This category includes images where the font has a certain character overlap that makes it difficult to find spaces between them.
- Few: The OCR has managed an almost perfect recreation of the text located in the image. This category includes images that contain receipts with a good font choice as well as a lot of computer generated PDFs. Tesseract struggled a lot with special characters like the Norwegian Æ, Ø and Å, which resulted in extractions falling into this category instead of being perfectly recreated.
- None: The OCR managed a perfect recreation of all the text located in the image. In this category, all the images are screenshots of a computer or computer generated without being printed.

Below, we can see two plots. On the left, we can see the frequencies for each error category for both Tesseract and OneNote. On the right, we can see the failure percentages one can expect depending on the error category.

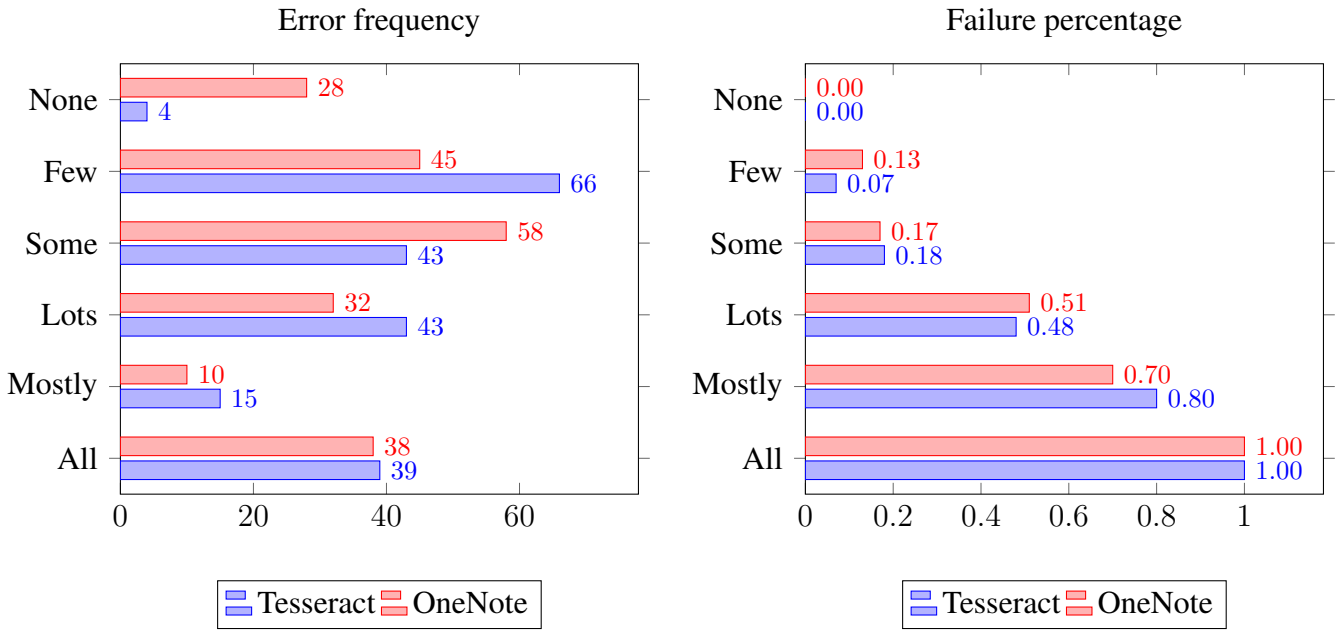


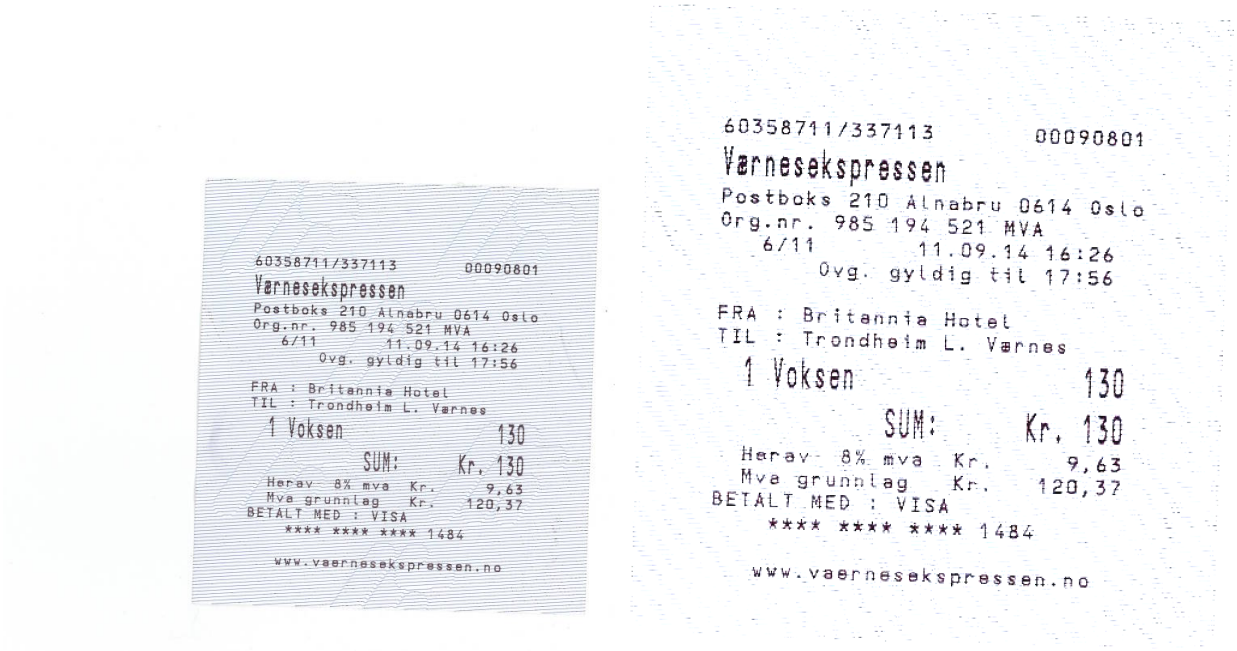
Figure 5.1: Error frequency and failure percentage of the different classifications

As one would expect, images where the OCR fails will never provide any dates or amounts. This is followed by a sinking failure probability going from the mostly category, down to the none category where there are no failures. If we compare the number of images in each category for OneNote and Tesseract, we can see that they have similar performance. OneNote excels at reading screen-shots and computer-generated files, whereas Tesseract struggles to perform a perfect extraction, but will often read images well. For many of the images, the reading quality will be similar for both OneNote and Tesseract, but Tesseract tends to have a slightly higher error rate compared to OneNote.

Figure 5.2 shows a receipt one typically receives on the airport shuttle buses and the like. Neither OCR will be able to read these images in any form or way, and the output will simply be blank. If the image is rotated, the OCR may return text consisting of a lot of capital letters (capital i to be exact) and pipes. This could pose a problem as a lot of the TECs naturally involves transportation to and from airports, hence there is a need for some solution. Luckily there are several potential ways to solve this specific problem in the image processing department, and we will explain three of them. Firstly, one could utilise some morphological operations to detect and remove the straight lines. Morphological image processing is presented in the digital image processing book by Nick Efford [5, chapter 11]. Another possibility is to increase the image brightness. By doing this correctly, one could remove most of the horizontal and vertical noise lines before the text itself starts to disappear, as seen in Figure 5.2. The last method would be to utilise filters. There are several filters that could potentially perform well in this task combined with a convolutional operation. Some potential filters could be Prewitt(horizontal and vertical), median or Gaussian. If one is using a Prewitt edge detection one would remove any elongated edges one finds.

Failures in a receipt will be when an OCR fails to obtain either the date or the amount from a receipt, as both are needed to accept a receipt as correct. This means that even if the only thing missing in the text received from the OCR is the last number in 2014, or if the text writes ‘3EP’ instead of ‘SEP’, or a comma is missing from the amount making ‘100,50’ to ‘10050’, the OCR have failed to properly read the image good enough to accept it. To obtain more solid results, some specific rules were created with regards to dates and numbers when checking image texts. This includes, but is not limited to:

- If a date is written as 20/09/2012 or 09/20/2012, both will be accepted as 20th of September 2012. This entails that 05/06/2012 can be either 6th of May 2012 or 5. of June 2012. This is to ensure the correct date is not lost due to different date-formatting on receipts.



(a) A receipt from an airport shuttle bus

(b) The same receipt with added brightness

Figure 5.2: Two snapshots of a receipt, where (a) is the original and (b) a tried improvement

- The separator in the date is ignored. This means that even if a backslash is read as the number one or a pipe, both will be accepted as correct, as long as the date numbers are correct and in correct positions.
- Months can be written in numbers only format (05/11/2018), in short format (5.Nov 2018) or in long format (5.November 2018). Both Norwegian and English names on months are accepted.
- Amounts must support different formats as well. 234 NOK could be formatted “234”, “234.00”, “234,-” or “234,00”.
- Amounts need to be solitary. This means that the text “The quick brown fox owed 235 NOK, but accidentally paid 1234 NOK” would not be recognised as having 234, but 235 and 1234 would both be accepted.

All of these rules were created to better find the dates and amounts in receipt texts, so they will also be used when measuring the efficiency. By finding what usually causes a reading to fail, e.g. by finding the incorrect dates or amounts, one could anticipate the errors and thereby find ways of counteracting them. The failure is more often due to not finding the amount, than not finding the date. In OneNote, this difference is larger than in Tesseract, as seen in Table 5.1. The best possible way of ensuring the quality of a read would be to utilise both OneNote and Tesseract, as there is a notable chance that Tesseract succeeds where OneNote fails and vice versa. When

accounting for rotating failed readings combined with a read from both OneNote and Tesseract, the failure percent becomes slightly under 22 for our test set. Unreadable receipts stand for nine out of these 22 percent. In these cases, the OCRs return either empty values or gibberish text, even after rotating and brightening the images. Table 5.1 visualises this. A solution aiming for the lowest possible error rate would utilise both Tesseract and OneNote, as well as perform rotations and image enhancements previously explained. On the labelled data, this solution would only fail on 25 receipts, or approximately 12 percent of the total dataset.

Table 5.1: OCR results in amounts and percent (in parenthesis)

	Tesseract			OneNote		
	Total	Amount	Date	Total	Amount	Date
Base score	84 (40%)	70(33%)	65 (31%)	77 (37%)	66 (31%)	55 (26%)
With rotations	68 (32%)	54 (26%)	49 (23%)	61 (29%)	50 (24%)	39 (19%)
Handling stripes	65 (31%)	51 (24%)	46 (22%)	58 (28%)	47 (22%)	36 (17%)
Both	49 (23%)	35 (17%)	30 (14%)	42 (20%)	41 (20%)	20 (10%)

Looking at the example from Figure 5.3, one can see that the text extracted is for the most part readable. Most of the hotel receipt is computer generated, but the image have been printed out to add the parking ticket in the lower right corner. Due to the printing, and subsequent scanning, there is some loss of image quality. One can notice that several parts of the receipt is being read wrong by the OCR due to this lower image quality. Such an example is the "Billing date" (becoming "Bi{/ing date"). Other examples of this is the reservation number going from "Reserveringsnr" in the image to "ReseNerjngsnc" in the text. One can also notice a line of zeroes missing in the four columns above the BAX, and the BAX number missing as well. Another mistake includes "0,00" becoming "o.oo". However, if one considers what is needed to approve this receipt, both the amounts (1040 and 110) and the dates (17. and 18.September 2014) are clear and readable in both extracted texts. The extracted text have been formatted to look like the image it is taken from, but is otherwise unchanged. The real extracted text span over several pages as the OCRs do not differentiate between a tab and a newline.

5.3 Implementation

To approve TECs where attachments are required on specific expenses or regulative, one needs to verify the content of specific attachments, meaning the ability to extract text from images is an important aspect. To read and verify attachments, one first need to extract both the data of a TEC and the image itself from the SAP backend. After extraction, the data needs to be processed by creating images from the base64 encoded strings. Once the images have been created, one can read them, extracting text (if any) from the image or PDFs, and run a comparative check against image

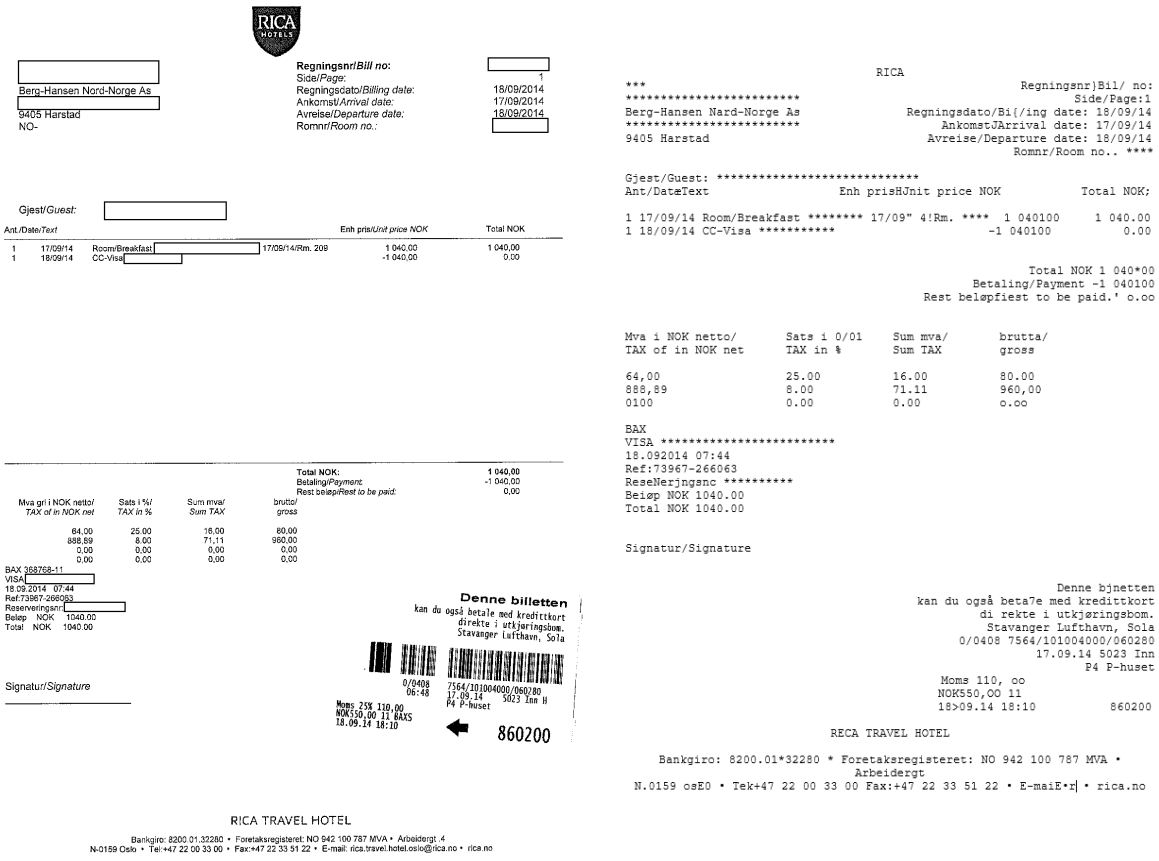


Figure 5.3: Example of an attachment and the text OneNote extracted

text and TEC amounts. Lastly one returns a score. The returned score can be used to evaluated the quality of an image and give an idea of success. Some of the code described here can be found in Appendix C, while everything can be found in the attached zipfile.

5.3.1 Extracting and preparing the data

Listing 5.1: Appending a field to a CSV string

```
CONCATENATE
  l_max_field
 <f_source>
  i_field_separator
  INTO
  l_max_field.
```

To use the OCR, the receipts have to be extracted from the SAP system. The attachments to SAP TECs are extracted with an ABAP script that iterates trough all the travels and then extracts the travel data first. After the travel data have been extracted the script checks if any travels have

Listing 5.2: Creating an images from a csv file

```
def read_and_proc_csv(FILE, folder):
    FILE = "/".join((folder, FILE))
    images = []
    with open(FILE, 'r') as file:
        data = file.read().split(DELIMITER)
        image = []
        for i, attr in enumerate(data):
            if i > 0 and i % len(HEADER.keys()) == 0:
                images.append(image)
                image = []
            if attr.find(base64tag) > -1:
                idx = attr.find(base64tag)
                image.append(attr[idx+len(base64tag):])
            else:
                image.append(attr.strip())
    filenames = []
    for j, image in enumerate(images):
        fname = image[HEADER["filename"]].replace("/", "").replace("\n", "").replace("*", "").↩
            replace(":","").replace("\\", "").strip().replace('\"', "").replace("\'", "")
        filename = '{}_{}.{}'.format(image[HEADER["reinr"]], fname, image[HEADER["file_ending"↩
            ]])
        out_file = '/'.join((folder+"_out", filename))
        with open(out_file, 'wb') as fout:
            fout.write(b64decode(image[HEADER["image"]]))
            filenames.append(out_file)
            if image[HEADER["file_ending"]] is "pdf" or image[HEADER["file_ending"]] is ".pdf":
                fout.write("\n%%EOF")
    if filenames:
        return image[HEADER["reinr"]], filenames
    return False
```

attachments in the tables from the SAP backend. If there are any attachments, they are all saved into a `binary large object` (BLOB) that is separated by "|;|". The script saves one CSV for each travel. Due to extracting BLOBs of various sizes this script took a long time to run. Once the BLOBs have been obtained, we ran a the script shown in Listing 5.2 to recreate the attachment files. Once completed, the only part left of data preparation was to perform a text extraction. This extraction runs three checks before it returns text. Firstly it checks if the TEC identifier already have been checked or prepared. If it have there will be a text file already extracted for the data. The first step was not present initially, but added due to the time it took to process over 250 gigabytes worth of images were longer than we were able to perform in one day on our laptops. Then it will take a look at the type of the attachment, and perform special tasks if the filetype is PDF or MSG(an outlook message file). If an attachment is a MSG file the text can be extracted much easier with a read of the email body instead of trying to run a text-extraction from the image. The same can be said for PDFs if they are computer generated. In Listings C.2 and C.3 this task is performed by PyPDF2 and Msg Extractor respectively. PyPDF2 and Msg Extractor are python modules recognised for performing well in their tasks of reading PDFs and MSG files. IF the file is an image file, or PyPDF2 did not return text, then the image is sent to the OCR for reading.

Listing 5.3: Calling the document read FM

```
CALL FUNCTION 'SO_DOCUMENT_READ_API1'
  EXPORTING
    document_id                = lv_doc_id
  IMPORTING
    document_data              = lv_doc_data
  TABLES
    object_header              = lt_header
    object_content             = lt_content
    contents_hex               = lt_contents_hex
  EXCEPTIONS
    document_id_not_exist      = 1
    operation_no_authorization = 2
    x_error                    = 3
    OTHERS                     = 4.
```

In Listing 5.1 the key change made to the `csv convert` function module is shown. The l_max_field is of type string, instead of type character. This change was made so that the arbitrary large numbers a BLOB contain could still be written to file, and were necessary as the character type with an upper limit of 262,143 characters could not contain the larger BLOBs, and especially not several BLOBs. Several other function modules were created. They were used to perform things such as support the extraction of the BLOBs from the SAP tables (Listing 5.3), obtain the necessary keys for attachments (Listing 5.4) and find the expenses with attachments(Listing 5.5).

Listing 5.4: The FM called to obtain the global unique identiyfers for the attachments

```
    CALL FUNCTION 'ZNAD_GET_ATTACHMENTS_FOR_TRIP'
      EXPORTING
        i_pernr      = i_pernr
        i_reinr      = i_reinr
      IMPORTING
        es_srgbtbrel = ls_srgbtbrel_tmp
      TABLES
        et_srgbtbrel = lt_srgbtbrel.
```

As there are a lot of attachments to process, we found it prudent to perform an intermediate save of the attachments if we used OCR to process an image. This way if the program encountered a problem it would be unnecessary to use Tesseract to reread an image it had already read, instead just jumping straight to reading the ".txt" files. Tesseract was run through command line and Listing 5.6 shows the conversion to tiff file if the attachment have the PDF format, the running of the image reading via Tesseract and the reading of the text file created by Tesseract. In the process of checking the attachments, we utilised four of the same CSV files as in Chapter 4. Some more listings from our python script can be found in Appendix C. After this process is complete we are left with a score for each TEC checked where 0 is the receipt returning no errors, and higher numbers means more errors in checking attachments on a TEC. The score 100 is used for TECs where they have no attachments but their expense type or travel regulative demands it.

Listing 5.5: The function module created to get attachments belonging to expenses

```
FUNCTION ZKVIST_HUSEBO_GET_ATTACH_EX.
    DATA:
          lt_attachments TYPE TABLE OF toahr .
    TYPES
      toahr_t TYPE TABLE OF toahr.
    CALL FUNCTION 'ZNAD_GET_ATTACHMENTS_EXPENSE'
      EXPORTING
        i_image_link    = i_image_link
      TABLES
        et_attachments = lt_attachments.
    et_attachments = lt_attachments.
ENDFUNCTION.
```

Listing 5.6: The construction and handling of interacting with both Tesseract and ImageMagick

```
def read_image(file, in_dir, out_dir):
    old_name = file
    file = in_dir + "/" + file
    if ".pdf" in file:
        new_file =  file.replace(".pdf", ".tiff", -1)
        command = ["convert", "-density" , "300", file ,  "-depth", "8",
            "-strip", "-background",  "white", "-alpha", "off", new_file]
        subprocess.run(command)
        file = new_file
    out_file = out_dir + os.path.splitext(old_name)[0] + "out"
    command = ["tesseract", file, out_file, "-l", "eng"]
    subprocess.run(command)
    with open(out_file+".txt") as f:
        data = f.read()
    return data
```

5.4 Results

After processing 6626 TECs, the results were used to attempt a classification of approve or reject solely by using the data from reading and checking attachments. The python machine learning library SKLearn was used for this, and the classifiers used were Random Forest, AdaBoost, k-Nearest Neighbours, Multilayer Perceptron and Decision Tree. Of the dataset used there are 410 rejected TECs, while the rest of them are accepted. Of the rejected data, 380 is placed under the no attachments class with the weight of a 100. In Table 5.2 the result from our testing shows. All of our errors is of the unwanted type II kind shown in Figure 2.16, but considering the amount total amount of TECs managed in this problem, this can still be considered very good. The results may however be faulty since it does not contain any approved or rejected TECs that should not have any attachments, nor any TECs having computer generated PDFs as attachments, and this may create lower amount of variance in the dataset. The computer generated PDFs would all, however, most probably be read by the PyPDF2 module. The text in them would thus be returned perfectly and the data hence be independent of OCR usage.

Table 5.2: Results for testing classification by attachments result after OCR

Failures	Classifier	Additional Settings	Additional Values
5	Random Forest	criterion, estimators	entropy, 25
5	Decision Tree	No	-
5	Multilayer Perceptron	learning rate, hidden layer	e^{-4}, 100
5	AdaBoost	random state	20
5	k-Nearest Neighbours	neighbours	3

The results Table 5.2 shows the average failure amount of five different classifiers over 20 runs with 6626 TECs being classified and 410 of those TECs being rejected. The data is split into a train- and a test set, where the test set is of size 1326 and there are 68 rejected TECs in it. They are being run on two attributes, not including the target attribute. The attributes is the discrepancies score and a list of the type of failures. As previously mentioned the failures consist solely of the undesirable type II errors. The five failures all classifiers have problems predicting is all rejected TECs that have attachments. Successfully rejecting 63 TECs that demand attachments, but do not have them, shows that the script properly demands attachments, and that any classifier would be able to predict rejection without it. This leads to the further conclusion that the 5 false positives in our test set have been rejected due to other reasons.

Chapter 6

Conclusions and future work

In this chapter, we will draw some conclusions to our work as well as define some future work.

6.1 Summary of Findings

To assess our work, we will in this section perform several forms of evaluation. We will analyse both the performance of our OCR, the performance of our machine learning algorithms as well as the data and methods behind them.

6.1.1 Manual Rules

As there are strict rules and regulations concerning TECs, simply using a series of rules defined in a centralised configuration might work really well in a real-life situation. One of the concerns by using machine learning is that every year, regulations might change and effectively rendering an already trained classifier useless[1]. If this happens, the classifier must be retrained on data that does not yet exist as the new regulations invalidate the old TECs as training data. Using a centralised configuration instead will enable us to do simple adjustments every time something changes in the regulations. A classifier might still be used in parts of the solution, but by using machine learning to find patterns in the old TEC data, we are effectively attempting to find some patterns that we already know because of statically defined rules and regulations concerning TECs. Perhaps the correct solution would be not to use machine learning on all the data, but rather a specific portion of the data that might benefit from it.

[1] This is why we have not used data from 2018

6.1.2 Classifiers

As it turned out, it was difficult to use the raw data alone to automate the TEC workflow. One major reason for this was the lack of historic data in a raw format, like the one we used for the current version. As the database tables are overwritten when a change is saved to a TEC, we have missed out on a lot of information about why a TEC has been rejected at some point in time.

Out of the classifiers we used, both in feature-based learning and in neural networks, the winner in our opinion was the random forest classifier. The random forest showed the best potential and it outperformed all the other feature-based classifiers as well as the neural networks. The neural networks failed to find any patterns in the data and seemed to either approve all TECs, or reject them.

We believe that, even though it was hard to find patterns on the entire dataset to successfully use machine learning algorithms, machine learning can still be used on fragments of the data. The performance of our classifiers can be looked at in both its actual performance when it comes to the evaluation metrics, but also with time spent training the algorithms.

6.1.3 Dataset

We have a few concerns about the dataset. Even though we have attempted multiple different ways to represent the dataset, we have seen no differences in the actual performance metrics of the classifiers. The only improvement we have seen was to the training process as the main difference in the representations is the number of features that the dataset has.

In our pre-processing we had to hash the data, as it is personal information and we could not take the data out of DFØs servers otherwise. This hashing function was tested on a different dataset, and with the hashing function applied to every attribute the performance of the classifier did not decrease, which made us conclude that the hashing function did not destroy any information in the dataset. However, our dataset has a lot of values that are strings in the first place and we fear that it may have caused some loss after all, but we have not been able to prove this as of now.

6.1.4 Optical Character Analysis

In Chapter 5, we explain our usage of OCRs and the performance one could expect from two different OCRs. Even as our results were fairly bad, the usage and potential of OCRs to approve TECs can still very good. One could expect the OCRs to perform much better once one handles

rotated images and receipt with lines in them. Once finding a sufficiently good, and efficient, way to handle this the results from the OCRs should be able to reflect this improvement.

One thing to note is that the receipts used to measure the potential of the OCRs, 210 receipts in 150 images, where all from a different set than the ones used to perform the actual classification. There is a reason for this, that is the confidentiality of the attachments demanded we use a different set to analyse on. We do, however, feel like there is no need to expect different images in the actual attachments, as the images used to test is a subset of previously used attachments, albeit a bit older. Another thing to note is that as the amount of receipt used to measure the potential is rather small, at least compared to the amount of attachments we handled while performing the classification. The measuring set may, due to being small, also be wrongly distributed over the expense types if compared to the real data.

The efficiency results of using OCR are still very promising and with proper combination of other attributes in a TEC could provide a good solution to automatically approve a TEC. One could expect the OCR to be able to detect receipts for dates outside of the travel period or difference in amounts from the receipt to amounts filled out in the TEC.

Finally, a thing to consider is countermeasures against harmful tampering of attachments images. If someone, on purpose, change the numbers on a receipt to become larger, or add an old receipt and change the date to get a larger reimbursement, the receipt should be invalidated. This problem have currently not been considered in our solution, but as there are often someone trying to exploit computer systems, one should expect this to be a possible high priority problem.

6.2 Conclusion

In this thesis, we have looked into the possibility of automating a part of the TEC workflow in an attempt to save both human and financial resources. Our main focuses has been data exploration of historic TECs, creating a dataset that we have used to train different machine learning and deep learning algorithms, as well as OCR to process paper receipts.

By using the OCR to read attachments, we were able to use the processed text to compare against TECs and their expenses to check for errors. Our tests shows that Tesseract can obtain text from 23 percent of our test set well enough to accept them as long as solutions are used for rotated and airport shuttle bus receipts. We were however unable to prove the usefulness of OCR in the dataset we had. This was due to the attachments used being correct. In the process of using OCR we successfully reject a lot of TECs due to lack of attachments as part of the processing done here.

Our scripts for using OCR and attachments to approve or reject rejects all of the TECs without receipts, but fails to reject TECs with correct attachments. This is, as previously mentioned, to be expected as attachments is not the sole reason for rejecting TECs.

The raw data from the SAP database table on its own are not sufficient to perform classification on TECs with neither neural networks, nor classic machine learning methods. One reason for this result is that there are also the attachment data to consider. Another, probably smaller reason, is the effect that human bias may have on the dataset, causing there to be rejected data with similar attributes as approved data. In machine learning this should not necessarily cause problems, but the low amount of rejected data, combined with the fact that a lot of our rejected data came from lack of attachments, further strengthened this problem.

However, there is still potential in automating the approval process, especially in combination with OCR usage. As the possibility of reading images is a key factor for approving any TEC demanding attachments there has to be some type of OCR usage. An optimal solution for the approval process would be dependant on all the data involved in a TEC. The best potential of such a solution would be reached by a module based solution, allowing for different techniques to be used on different segments of data, and changing of a single module in cases of new regulations or rules.

6.3 Future work

In any project of this magnitude, there will always be certain things one would wish to have time for, but when one is on a schedule that is easier said than done. In this section, we will discuss some of the things that we want to take a closer look at in the future.

6.3.1 Combining OCR With Classifiers

As the results for both feature-based classifiers and neural networks have shown us, it is apparent that a machine learning algorithm on the entirety of the raw data will not work well enough without some data from the paper receipts. In the future, we wish to explore the possibilities of combining a machine learning algorithm like the ones that we have used in Chapter 4 with an OCR like the ones used in Chapter 5. As the OCRs look promising alone, a combination with a classifier might work even better.

6.3.2 Real Life Solution

In the future, we wish to deploy what we have discovered and developed into a real life solution. To do this, we will need the following:

- A web service for the classifier
- A web service for the OCR (can be merged with classifier)
- A connection to existing applications

There are many ways to implement a solution like this, but a solution that would work well for our purposes is placing the web services for the classifier and OCR behind existing firewalls. This would prevent users from using the model directly. As Bouvet has already created a web and a mobile application for submitting TECs, its backend would provide a secure endpoint, and that allows us to use the SAP backend to check for the correct permissions on a per-user basis. With signing into the applications requiring a strict two-factor authentication to get access to the SAP user account, the model would be provided with a lot of extra security.

Once the employee has signed in and has the correct permissions, we have a few ideas on how the solution could be implemented. One way is to use a trained classifier or another set of rules to verify the data in the TEC and prevent a submission if there are any errors in the data. This way, if an employee is allowed to submit the TEC, it will already have bypassed the first approver in the TEC workflow and it can be sent straight to the superior of the employee for approval.

Another possibility is to use the OCR to automatically generate an expense using the data that has been extracted from a receipt to speed up the process and making it more convenient and less of a hassle for employees to submit TECs. The OCR can also be used the moment a receipt has been uploaded to verify that the image is good enough and that the information corresponds to other information provided in the TEC.

6.3.3 OCR Improvements

A lot of potential improvement in a solution would come from making the OCR perform better at reading receipts. The two main reasons for not being able to properly read an attachment is `a` the image is rotated or `b` the image is a receipt from an airport shuttle bus or another receipt with similar noise generating lines. Solving these two problems is therefore a good way to improve the solution as a whole. Previously we mentioned that b causes the OCRs to be unable to extract any

text from the image, but one may filter the noise or remove it to support the OCRs. The potential problem with a is the added overhead, especially when considering solving both problem a and b. If one want to rotate the image and also brighten it to obtain some form of removal of both one suddenly have eighth potential correct ways to handle an attachment, adding a lot of extra overhead to the overall runtime of the solution.

There are several other steps one can utilise to improve the performance of an OCR. By creating high contrast images one makes the extraction of text simpler as there is less uncertainty. One should also consider slicing images to the bare minimum so that any additional noise generated from background will not disturb the reading. These are both the first two steps suggested by Tesseract if one fails to obtain the desired results.

One could furthermore generate a set from an existing set of TECs data where the attachments are wrong, making it so there are errors in the dates or amounts for the OCR to detect. Once this set is made it can be used to measure the efficiency classifiers are able to provide in rejecting TECs where attachments are incorrect. This measurement would give an idea of our implementations proficiency of detecting faulty receipts. One thing to note here is that the weight of a 100 we set to any rejected TEC where there are no attachments, even as they are required, could disrupt this. The weight may have to be reduced after further testing, but tested reducing this weight to both 5 and 10 without any changes in our results. This lack of change may be due to the lack of tampered and faulty attachments in our dataset.

Bibliography

[1] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, 3 edition, 2014.

[2] Federico Berruti, Graeme Nixon, Giambattista Taglioni, and Rob Whiteman. Intelligent process automation, 2017. URL https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/intelligent-process-automation-the-engine-at-the-core-of-the-next-gener

[3] Dua Dheeru and Efi Karra Taniskidou. Uci machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[4] Niklas Donges. The random forest algorithm, 2018. URL https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd.

[5] Nick Efford. *Digital Image Processing: A Practical Introduction Using Java*. Pearson Education, 2000.

[6] Line Eikvil. Ocr - optical character recognition, 1993. URL https://tinyurl.com/yax8ogqv.

[7] Roger Evans, Paul Piwek, and Lynne Cahill. What is nlg?, 2002. URL http://wing.comp.nus.edu.sg/~antho/W/W02/W02-2119.pdf.

[8] Justin Gage. Introduction to loss functions, 2018. URL https://blog.algorithmia.com/introduction-to-loss-functions/.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[10] Sirkka-Liisa Jämsä-Jounela. *Annual Reviews in Control: Future trends in process automation*. IFAC: the International Federation of Automation Control, 2007. vol. 39, issue 11, pp. 114-119.

[11] Patrick Laurent, Thibault Chollet, and Elsa Harzberg. Intelligent automation entering the business world, 2015. URL https://www2.

deloitte.com/content/dam/Deloitte/lu/Documents/operations/lu-intelligent-automation-business-world.pdf.

[12] Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification, 2015. https://www.kaggle.com/mlg-ulb/creditcardfraud.

[13] Pengfei Xu Xiaowen Chu Shaohuai Shi, Qiang Wang. Benchmarking state-of-the-art deep learning software tools, 2017. URL https://arxiv.org/pdf/1608.07249v7.pdf. Acessed: 2018-01-09, Version 7.

[14] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson Education, Limited, 2014. ISBN 9780273769224. URL https://books.google.no/books?id=T0dzMAEACAAJ.

[15] Utdanning.no. Personalrådgiver. https://utdanning.no/yrker/beskrivelse/personalradgiver, 2017. Accessed: 2018-01-04.

[16] Harry Zhang. The optimality of naive bayes, 2004. http://www.cs.unb.ca/~hzhang/publications/FLAIRS04ZhangH.pdf.

Appendices

Appendix A

Abbreviations

API Application Programming Interface
AWS Amazon Web Services
BLOB Binary large object
Bouvet Bouvet Norge AS
CNN Convolutional Neural Network
CPU Central Processing Unit
CSV Comma-separated Values
DFFN Deep Feedforward Network
DFØ The Norwegian Government Agency of Financial Management
ERP Enterprise Resource Planning
FM Function Module
GDPR General Data Protection Regulation
GPU Graphical Processing Unit
GUI Graphical User Interface
HR Human Resources
JSON JavaScript Object Notation
LSTM Long Short-Term Memory
MLP Multilayer Perceptron
MSE Mean Squared Error
NLG Natural Language Generation
OCR Optical Character Recognition
PDF Portable Document Format
RPA Robotic Process Automation
ReLU Rectified Linear Unit

SGD Stochastic Gradient Descent
SKLearn Scikit-learn
TEC Travel Expense Claim

Appendix B

Travel Expense Claim Examples

Figure B.3 shows example of a TEC in several snapshots from a web-client. There is some subpages to this web-client as well shown in figures B.1 and B.2

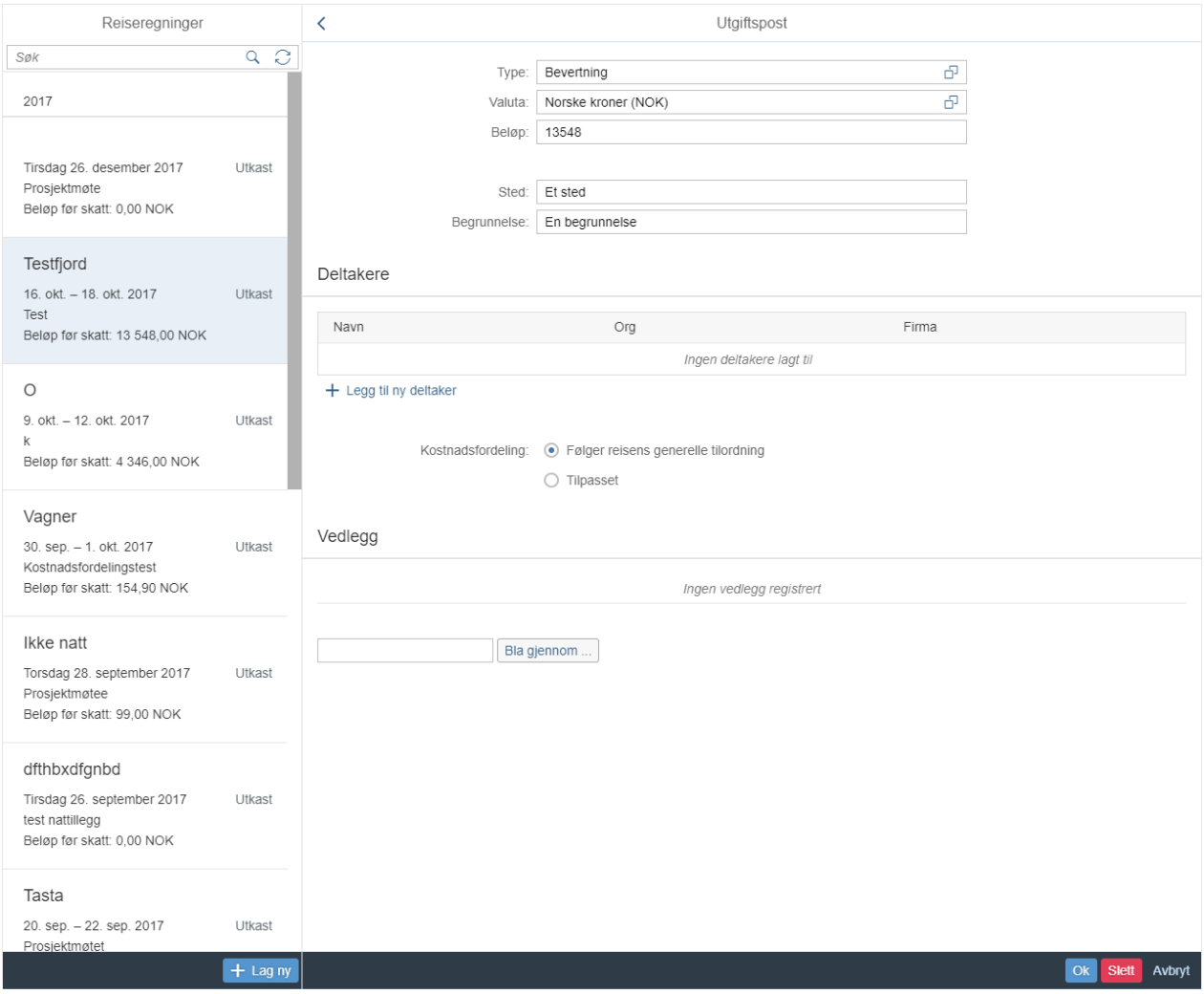


Figure B.1: The expenses subpage of the TEC web-client, here filling out fields for refreshments

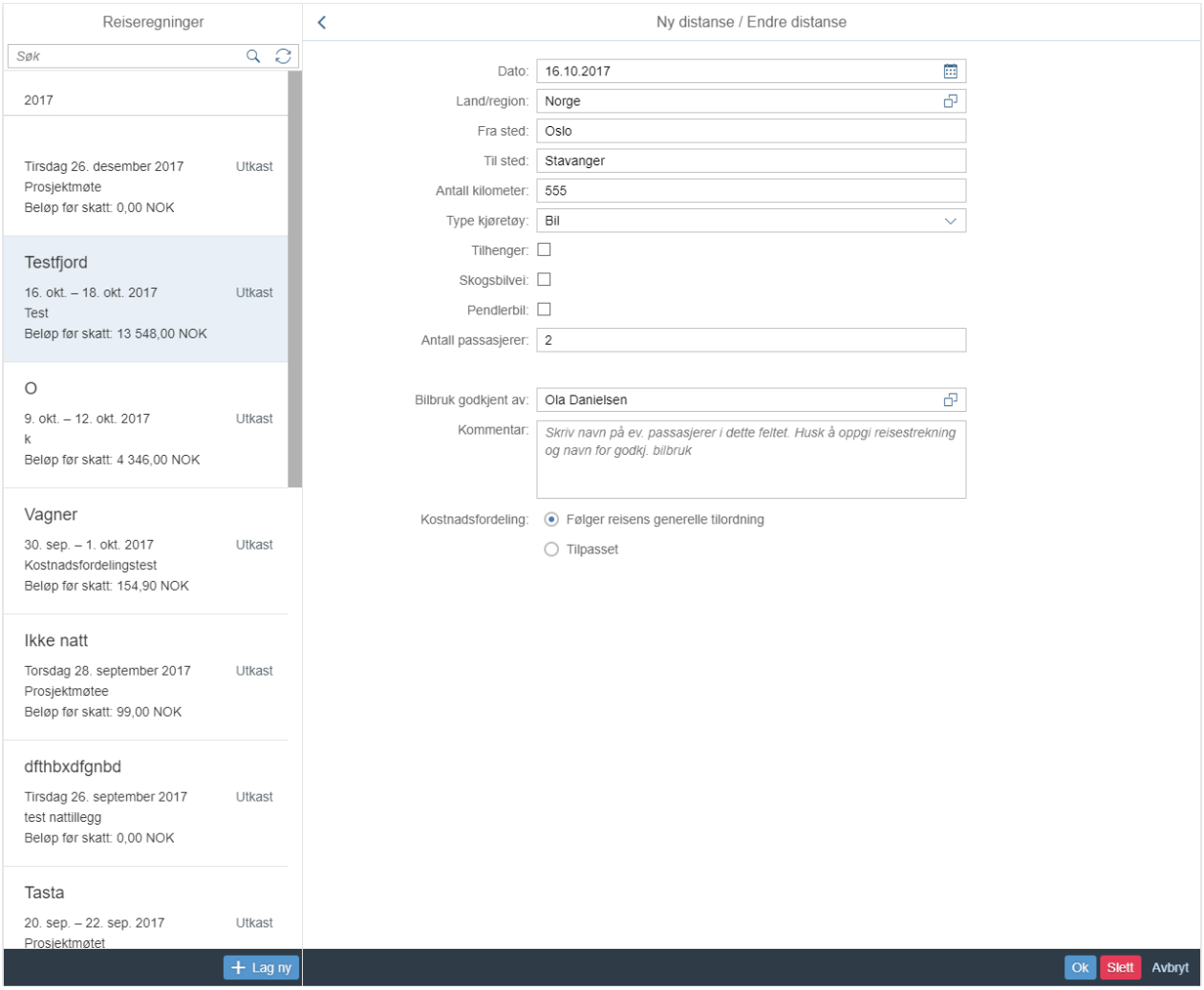


Figure B.2: The mileage subpage of the TEC web-client

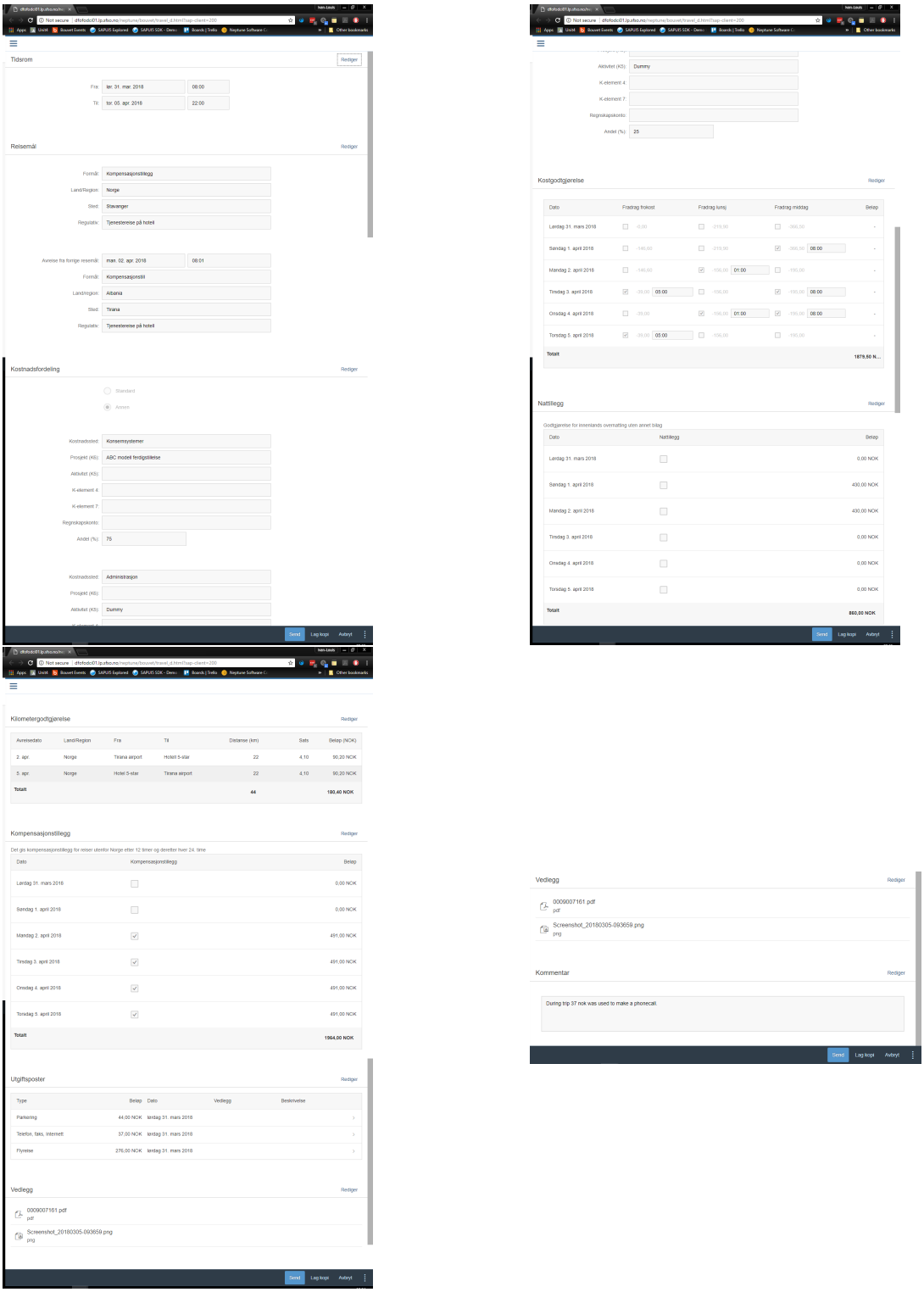


Figure B.3: An example of how a travel expense claim would look like in the web-client

Appendix C

OCR dependencies and code examples

C.1 Installation and licensing

Tesseract have two parts that may be installed, the engine itself and language training data. With bash it may be installed simply by "sudo apt install tesseract-ocr" and "sudo apt install libtesseract-dev". For this thesis only the engine itself was installed. For other installation guide see the Tesseract wiki[1]. Tesseract is licensed with an Appache License(v2).

Leptonica is used by Tesseract, and have an open source lisence[2].

ImageMagick is used to perform a command line change from PDF to tiff file. ImageMagick is free to use and have an license compativle with GNU General Public License(v3)[3]. Installation via executable or via bash "sudo apt-get install imagemagick".

PyPDF2 installs via pip, "pip install PyPDF2", and is a fork of Mathieu Fenniaks original pypdf sponsored by Phaseit, Inc. Utilises a standard open source license [4], courtesy of Mathieu Fenniak, Ashish Kulkarni and Steve Witham.

[1] https://github.com/tesseract-ocr/tesseract/wiki
[2] https://github.com/DanBloomberg/leptonica/blob/master/leptonica-license.txt
[3] https://www.imagemagick.org/script/license.php
[4] https://github.com/mstamy2/PyPDF2/blob/master/LICENSE

Msg extractor installs via pip, "pip install https://github.com/mattgwwalker/msg-extractor/zipball/master". Has a GNU General Public License [5], creator and maintainer is Matthew Walker.

C.2 Listings

Listing C.1 shows the switch-case used to check the attachment type and performed a check if it had been read by Tesseract before. Listings C.2 and C.3 both try to extract text from PDF and email, first one using PYPDF2 and the latter using .

Listing C.1: Function with a switch to return the text of an attachment

```
def get_attachment(file, directory):
    idx = file.rfind(".")
    t_file = file[:idx] + "out.txt"
    if t_file in os.listdir(TEXT_DIR):
        with open(TEXT_DIR+t_file, "r+") as file:
            data = file.read()
        data = data.replace("\t", " ").replace("\n", " ").replace("\r", " ")
        return data
    if ".pdf" in file: ## Check if the pdf is computer generated and contains text elements
        res = support_functions.try_read_pfd(file,directory)
        if res:
            return res
        else:
            print("No available text in the pdf, falling back to OCR")
    if ".msg" in file: ## Email file
        return support_functions.read_email(file, directory)
    return command_line.read_image(file, ATTACHMENT_DIR, TEXT_DIR)
```

Listing C.2: Function used to try to extract text from PDFs

```
from PyPDF2 import PdfFileReader
from PyPDF2.utils import PdfReadError
def try_read_pfd(path, folder):
    data = ""
    full_path = "{}/{}".format(folder,path)
    with open(full_path, "rb") as file:
        try:
            reader = PdfFileReader(file)
        except (EOFError, PdfReadError):
            print(PdfReadError)
            return False
        num_pages = reader.getNumPages()
        for i in range(0, num_pages):
            page = reader.getPage(i)
            data += page.extractText()
    if data == "": return False
    return data
```

[5] https://github.com/mattgwwalker/msg-extractor/blob/master/COPYING

Listing C.3: Function for extracting text from emails

```
import ExtractMsg
def read_email(path, folder):
    full_path = "{}/{}".format(folder,path)
    try:
        msg = ExtractMsg.Message(full_path)
    except OSError as e:
        return False
    if msg.body:
        print(type(msg.body), type("\n"), type("\t"))
        if type(msg.body) is bytes:
            try:
                ret = msg.body.decode("utf-8")
            except UnicodeDecodeError as e:
                ret = msg.body.decode("latin1")
            ret = ret.replace("\t", " ").replace("\n", " ").replace("\r", " ")
            return ret
        ret = msg.body.replace("\t", " ").replace("\n", " ").replace("\r", " ")
        return ret
    return False
```

Appendix D

Data scripts

D.1 ABAP Field Symbols

The difference between a regular variable and a field symbol is that it is referenced. Below, we can see an example of looping through a table and attempting to change the currency attribute of a structure. The attribute is not written to the entry in the table, but rather to a locally defined copy.

```
DATA:
    lt_amounts TYPE TABLE OF bapitrvsum,
    ls_amounts TYPE bapitrvsum.

READ TABLE lt_amounts INTO ls_amounts INDEX 1.
WRITE ls_amounts-currency. "Prints 'NOK'
CLEAR ls_amounts.

LOOP AT lt_amounts INTO ls_amounts.
    ls_amounts-currency = 'USD'.
ENDLOOP.

READ TABLE lt_amounts INTO ls_amounts INDEX 1.
WRITE ls_amounts-currency. "Prints 'NOK'
```

To fix this, we can utilise field symbols, which creates a reference to the actual entry in the table rather than copying it.

```
DATA:
    lt_amounts TYPE TABLE OF bapitrvsum.
FIELD-SYMBOLS:
    <fs_amounts> TYPE bapitrvsum.
```

```
READ TABLE lt_amounts ASSIGNING <fs_amounts> INDEX 1.
WRITE <fs_amounts>-currency. "Prints 'NOK'
UNASSIGN <fs_amounts>.
LOOP AT lt_amounts ASSIGNING <fs_amounts>.
    <fs_amounts>-currency = 'USD'.
ENDLOOP.

READ TABLE lt_amounts ASSIGNING <fs_amounts> INDEX 1.
WRITE <fs_amounts>-currency. "Prints 'USD'
```

Now, the currency attribute of the table entry is correctly changed from NOK to USD.

D.2 Casting to an extended type

```
FIELD-SYMBOLS:
  <fs_input_table>          TYPE STANDARD TABLE,
  <fs_output_table>         TYPE STANDARD TABLE,
  <fs_input_table_entry>    TYPE any,
  <fs_output_table_entry>   TYPE any,
  <fs_value>                TYPE any.

l_input_table_name = 'lt_amounts'.
APPEND l_input_table_name TO lt_input_table_names.

l_output_table_name = 'lt_all_amounts'.
APPEND l_output_table_name TO lt_output_table_names.

LOOP AT lt_input_table_names INTO l_input_table_name.
    ASSIGN (l_input_table_name) TO <fs_input_table>.
    IF <fs_input_table> IS ASSIGNED.
      READ TABLE lt_output_table_names INTO l_output_table_name INDEX sy-tabix.
      ASSIGN (l_output_table_name) TO <fs_output_table>.
      IF <fs_output_table> IS ASSIGNED.
        LOOP AT <fs_input_table> ASSIGNING <fs_input_table_entry>.
          CREATE DATA ls_output_table_entry LIKE LINE OF <fs_output_table>.
          ASSIGN ls_output_table_entry->* TO <fs_output_table_entry>.
          ASSIGN COMPONENT 'REINR' OF STRUCTURE <fs_output_table_entry> TO <fs_value>.
          IF <fs_value> IS ASSIGNED.
            <fs_value> = ls_ptrv_head-reinr.
            UNASSIGN <fs_value>.
          ENDIF.
          MOVE-CORRESPONDING <fs_input_table_entry> TO <fs_output_table_entry>.
          APPEND <fs_output_table_entry> TO <fs_output_table>.
        ENDLOOP.
      ENDIF.
    ENDIF.
    UNASSIGN <fs_input_table>.
    UNASSIGN <fs_output_table>.
  ENDLOOP.
```