Automatic classification of orders lines in joint Dealer Management Systems

Master thesis
ICA-5574781

Author
Martijn Lentink

Supervisors
Dr. A.J. Feelders
Prof. Dr. A.P.J.M. Siebes

August 29, 2016

Universiteit Utrecht

Faculty of Science
Department of Information and Computing Sciences
University Utrecht

RDC

RDC inMotiv Nederland B.V.

Contents

List of Figures

List of Tables

Abstract

In this thesis we looked at different data originating from several Dealer Management Systems. By comparing the different data we tried to find a field set that can be used as features for our classifier models of receipts. We found that this data can be uniformed well by taking the few fields an intersection of the fields of all Dealer Management Systems yield. When we add some extra fields with slight manipulations we created a data set that has high potential for machine learning classifications. Different set ups showed F_1 scores for classification well above 90% through three data sets with four learning models. Further we introduce new options in attempt to improve the classification rate further. We used our domain knowledge for the construction of smart *token detectors* and construct a unique *compound word splitting* algorithm for splitting Dutch compound words.

Chapter 1

Introduction

RDC manages nationals largest data set on cars and their owners, their information is gathered throughout several systems which are maintained within the organization. The numerous systems they maintain are mainly targeted at dealers and car retailers, covering huge amounts of data concerning the dealers and their customers. One of these systems, 'CaRe-Mail', is used by Dealer Management System's and car retailers for informing their customers and contacting them at certain moments. Typical messages sent to the dealers customers are upcoming maintenance checkups and tire swaps. The data of dealer is processed on a daily basis, and RDC is constantly looking for ways to improve this process. With the evergrowing data RDC interest rises on smart data mining solutions to use within their systems. This thesis serves as a first look into this field for one of their products; CaRe-Mail.

1.1 Motivation

RDC has been through many changes in the last couple of years. Since their portfolio mainly targeted the automotive branch, the credit crunch was inevitably felt by RDC too. During this period the organisation decided to part from other products that did not fall within their automotive activities to be able to fully focus on what they really excel in; everything concerning vehicles. The CaRe-Mail system originates from the rising question among car dealers on an efficient system for contacting their clients via different channels. The proliferation of different contact media like e-mail, text messages, online dashboards et cetera, created the need for new smart ways to contact their clients. Where contact was earlier on mainly through two channels (letter correspondence and telephone) the new ways of contacting became complex fast; all clients have different preference regarding the contact with their dealer and because people tend to visit them less and less with the rise of the internet, client information gets outdated quickly. CaRe-Mail enables dealers to automatically contact their clients personally through the channels that the user prefers, this takes away a lot of manual labour and frees time for the dealer to do other businesses. The messages that are sent are generated by means of the information that is collected within the dealers' Dealer Management System. In the current version of CaRe-Mail a rule based information extraction is used which checks if certain regular expressions match the entries in the Dealer Management System and classifies receipt accordingly to predict future messages. In essence the value of the product lies within the accuracy of the receipt classification. Incorrect classification results in propagating the wrong evidence through the system which may lead to messages being sent at the wrong time, or sometimes not at all.

A better classification rate will not only affect RDC by improving their product and their competitive position, it may save the dealers a lot of time by doing bulk operations for example; if multiple employees of a company need tire swaps one could consider doing bulk tire swaps on location, easing the experience for the employees. Or a dealer can instruct their workshops to update their stock for certain parts to match the messages sent.

The CaRe-Mail product has a history of over a decade and through the past years it was just able to keep up with the rapid changes in the dealer industry. The system now supports all major Dealer Management System's but its code base is heavily outdated. As a result changes and maintenance work are usually labour expensive. For this reason RDC has decided to rebuild the system from ground up with a complete new architectural design, a different programming language and data model design. Apart from maintenance reasons (and for providing the user a more attractive/modern interface) they are also interested in improving the accuracy of their data classification. RDC is excited in using statistical approaches for this purpose, and requested my help in finding models that are suited for predicting the data streams. Although CaRe-Mail already performs as one of the best in the business, RDC aims to stay ahead of the competition by trying out new models for data classifications. This strive, to stay ahead of the rest, brings this thesis, to further explore the possibilities in data classification for the improvement of their CaRe-Mail system.

1.2 Definitions

Dealer Management System A dealer management system, abbreviated as DMS, is a management system that is used by car dealers to manage all information concerning their customers. Typical information included in these databases are customer information, vehicle data, workshop plannings and declared receipts. Throughout this document the term dealer management system and its abbreviation will be used interchangeably.

CaRe-Mail One of the most important systems within RDC that works as a service for numerous vehicle dealers. The system contacts the dealers customers on their behalf by sending them messages on almost everything; informing them for upcoming mechanical checks, tire swaps, routine maintenance or may be even birthday congratulations.

CSV A Comma Separated Values file format. Data records are stored in a plain text file and fields are separated by a set delimiter such as a comma (hence comma separated values).

(Class) label A discrete value known (or derived) for data that represents its true state. Typical class labels for the current domain is a receipt being that of a routine maintenance, tire swap or technical examination. The labels are used to guide the models in their training phase to assign the right features to the right class.

Features Also known as *feature vectors* and *descriptors* are selected (and possibly manipulated) parts of the data which are created to be able to rapidly compare different input for training and evaluation purposes.

Classifier A classification model constructed to predict class labels of previously unseen and unlabeled data. For training of a classifier, features are used to generate a predictive model.

Overfitting The event of training a classifier to fit the training data too precisely, resulting in a model that does not generalize to new data well. Best practice is to avoid this from happening altogether to get the best predictive results.

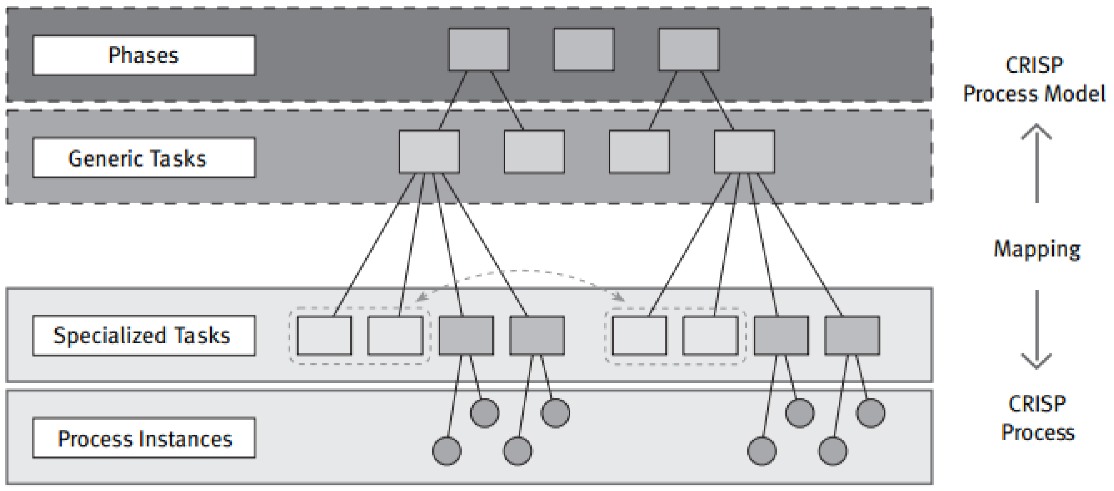


Figure 1.1: The hierarchical breakdown of the CRISP-DM model, taken from [1]

Ground truth For the training of statistical models using supervised learning, examples will need to be provided in order for the model to distinguish one class from another. The labels that are given to these example cases are assumed to be correct, this is called the ground truth. A classification is said to be correct if it matches the state of the data's ground truth value.

Impurity reduction The gain that a certain split has in comparison with the previous state. The impurity reduction is used in decision tree learning.

***n*-grams** Technique for splitting text corpora in smaller sets of size n, where each word counts for one. The special case $n = 1$ (unigrams), where the corpus is a bag of all its words is called the *bag of words* approach. n-grams may also be called shingles.

1.3 Methodology

Since this thesis project has practical character and has defined a clear goal we decided to adopt a proven research methodology that support the projects and gains insight at every point within the project. In this section, first the CRISP-DM methodology is described that is used for this project, then the results of the first phase is summarized. Followed by a planning for the final part of this project. Concluded by techniques that are to be used in the second phase and decision that has to be taken regarding the data and the models that will be constructed.

1.3.1 Cross-Industry Standard Process for Data Mining

For this thesis project the CRISP-DM process methodology will be used. CRISP-DM which stands for *Cross-Industry Standard Process for Data Mining* is, as its name states an industry standard, proven, process methodology for Data Mining projects. The process was founded around 1996 by three big players in the data mining community and it quickly became widely adopted. In [1] step-by-step the methodology is explained in detail. The most recent survey (2014) on the use of data mining methodologies yielded that CRISP-DM is the most used process in the data mining field [2]. Throughout the project a half a dozen stages are iterated, the different phases don't have strict boundaries itself, meaning that phase switches can occur in the project when needed. The process model itself is built up in an hierarchical order (see Figure 1.1). CRISP-DM distinguishes the following stages;

- Business Understanding
- Data Understanding
- Data Preparation
- Modeling
- Evaluation
- Deployment

Each of these stages will be explained in more detail in the following sections.

Business Understanding

In the first phase the focus in mainly on the understanding of the business. While creating an understanding of the business objectives, the problem definition can be defined on the basis of this information.

The following tasks are part of this phase; determining the Business Objectives, an assessment of the situation, a determination of the data mining goals and the production of a Project Plan.

Data Understanding

This phase starts off with the collection of the required data from all data sources. What follows is a thorough analysis of the data, to increase familiarity with it and find potential issues on the data quality.

In this stage the following steps exists; the initial collection of the data, a description of the data, and the exploration of the data, finally the data quality is verified.

Data Preparation

After the previous step gained insight in the data itself, in this step one looks at the possibilities in improving the quality of the data for processing purposes. The first arrangements are performed to transform the data so it can be used for generating models.

First a data selection is done, followed by a cleaning step, constructing data together with integration and lastly a formatting of the data.

Modeling

In this step different modeling approaches are considered, for each model different parameters are calibrated to find optimal values. Different mining techniques require the data to be formatted differently, for this reason it may be that a transition is made to the previous steps to preparing the data accordingly.

The tasks for this step are the selection of a modeling technique, generation of test design, the actual creation of the models and the assessment of the models yielded.

Evaluation

During the evaluation phase the built models are evaluated given the previously defined business goals. This phase basically is a decision moment for the project, where the decision is whatever happens to the resulted model.

The steps involved in this phase are the comparison of the evaluation results, a review process of the solution, determine the next steps for the final phase.

Deployment

The final phase involves the integration of the product that was generated in the previous stages. Here the model gets it final shape such that it can be effectively used by the client. This phase often also includes the sharing of the knowledge about the model and documenting the process.

The defined subtasks are plan deployment, plan monitoring and maintenance, the production of the final report and a project review.[3]

1.4 Research questions

When new receipts arrive at RDC it is of the utmost importance that they are interpreted in the correct way and that the most valuable information is extracted. The accuracy of the CaRe-Mail system relies wholly on how well this system performs. CaRe-Mail is completely rebuilt from ground up to be able to maintain the system more easily and make it able to integrate other systems as well. With the CaRe-Mail service a lot of manual labour is taken off the hands of the car dealers. Saving time in informing their customers for maintenance, the storage of their customers preferences, keeping track of maintenance plans etc.

To work towards the goals defined in Section 1.1 a couple of research goals are defined which all contribute to the correctness and accuracy of the classification. Since the data not only contains categorical and numerical values but also full text description fields, text mining is also part of the problem we face in this thesis. We pose the following research questions;

- How can the deviations in data fields be uniformed in such a way that a classifier can be constructed that is able to classify the greatest set of Dealer Management Systems correctly?

  The DMS' have a great variety in fields and therefore the information one sends is not comparable with that of other Dealer Management Systems. This also means that cross-DMS classification is also a challenge. A classifier may be trained on only the intersection of the fields of every DMS, in Section 3.1 we see that this would leave us with merely a car identifier and the workshop visit date, which are useless for classification. Different approaches are thinkable to tackle this problem; one is of course training a classifier per DMS, an other is taking some sort of data fusion approach and training just one (or a few) that generalized the classification of multiple Dealer Management Systems. For the latter case a research has to be done in which fields should be taken using normalisation and how these would be manipulated/amended in such a way that a classifier can be trained with ease.

- Which fields of the records are suitable to train and classify the records and which feature extraction approach can be used?

  Since every DMS serves different fields for their receipts a research has to be conducted in to which fields contribute to the correct classification of the receipt. One might think that a *description field is of high importance for the classification, while the license plate number may gain less insight. While this might be true in a simple case it may also be that the plate number gives us a service plan, which could be used as reference for classifications.*

- How accurate do(es) the selected technique(s)/algorithm(s) perform?

  We are then to implement one/some of the methods/techniques that we found in our earlier research and evaluate it to test its accuracy. A variety of training sessions will be performed with different data sets. These will vary in contents and size for training and evaluation sets. The results are then compared to the current system, since this system merely does a classification on maintenance this has to be taken into account. On the basis of the evaluation results conclusions are drawn accordingly and an action plan is set up for the deployment phase.

The evaluation time of the algorithms are considered during this thesis. There isn't however a set criterion that the system needs to meet, the current system receives its information at night and takes the rest of the night for processing this data. The information received on a certain day is often only relevant for messages that need to be sent in the distant future; a maintenance today is another maintenance years later. This means that in theory run time may as well be the time from the moment receiving the information up to some weeks before the next maintenance, for this reason no priority is given on the performance.

1.5 Overview

The rest of the thesis is structured as follows; in Chapter 2 we discuss the previous work relevant to the thesis project. In Chapter 3 we tell something about the structure of the data that the Dealer Management Systems have and perform a feasibility analysis to verify if the data is suited for classification. In Chapter 4 we prepare the data such that our classification models can be trained. In Chapter 5 we discuss the models that are trained and how the feature generation works. The models are then evaluated in Chapter 6 by some experiments we set up. And lastly in Chapter 7 we conclude this thesis with providing a summary on the work we did and we look at what future work might be done.

Chapter 2

Previous work

No previous work was found on tackling the problem of classifying receipt order lines. This is not too surprising since it is a quite specific topic. Classification, however, has been discussed at length in the literature. Throughout the years different models have been constructed for classifying different types of data into categories, arguably the first paper on this topic was that of Fix and Hodges Jr in [4] where classification of an unknown value was done on the basis of statistical comparison of known values. This method is closely related to the method we today call the k-nearest neighbour method. This term was coined by Cover and Hart in [5] where they formally defined the k-nearest neighbour approach, and they show that its performance is at least as good as an squared-errors approach.

An other approach is that of decision trees. These trees, that originate from decision analysis[6], are widely adopted in various areas and a lot of research has been done in efficient construction algorithms from data[7][8]. After the construction the evaluation is done by traversing it from root to leaf with binary splits in every node (given it's a binary decision tree) which makes it evaluate rapidly. An interesting paper is that of Toole that used decision trees to classify unknown words to find misspelled words and names/abbreviations [9]. Ho used decision trees together with random seeds for constructing multiple independent trees which all are evaluated during the classification process[10]. Candillier, Tellier, and Torre classified XML-structured documents with a *bag of tags*-approach using decision trees, although they assumed that the documents were semantically different for each class, their approach is interesting[11].

A more recent method is that of the Support Vector Machines defined by Boser, Guyon, and Vapnik in [12][13]. They proposed the idea of kernel functions for mapping data to hyperplanes to separate the classes linearly given the observations using a maximum margin separator. In [14] category names are used to classify documents using support vector machines. The method that Barak, Dagan, and Shnarch use can be helpful, since the description fields in the receipts that we are considering are somewhat similar to the category names they used.

A Bayesian network is a probabilistic model in contrast to the deterministic models mentioned earlier. The network is a mathematical model that is built around Bayes' theorem[15]. The term Bayesian network, coined by Pearl[16] in [17] for propagating evidence through the network, for updating beliefs. A computationally more interesting way in expressing a network is one with a 'naive Bayes assumption'; Friedman, Geiger, and Goldszmidt compare different Bayesian classifiers. One which simply neglects every mutual dependency relationship between variables. And also a tree based approach which relaxes the independence assumption by employing a tree structure [18].

A completely different approach is that of neural networks, this technique is based upon the workings of the brain, consisting of several neurons. Although these models been around for decades[19], due to computational challenges at the time of development, they were not widely used. Classification using these models became feasible at the end of last century, various algorithms have been proposed for training the networks[20][21]. One application that comes near to what is to achieved in this thesis is that of Ghosh and Reilly, in [22] they try to track down cases of credit card fraud using a neural network by looking at the credit card transactions. In 2006 Huang, Zhu, and Siew introduced the notion of Extreme Learning Machines (ELM) being feed forward neural networks with a single hidden layer[23]. Zhao et al. used these ELM's for creating a classifier for structured XML documents, in their evaluation section they show good accuracy classifying the documents [24]. They also state that their training time is manifold faster than other algorithms like Support Vector Machines.

The classifiers mentioned above need to be trained against certain data. Since the complete data itself is too rich and in itself has no way to compare the data to one another, a function has to be chosen to be able to do this comparison. One way of doing this comparison is by the means of *features*, a feature selection method extracts certain descriptors within the media and returns these descriptors in a vector; the feature vector. During evaluation and training a medium generates one feature vector which makes bilateral comparison possible. Liu and Motoda stress the importance of meaningful features in their book '*Feature selection for knowledge discovery and data mining*', they state that the feature selection is arguably the most important step in the design of a learning algorithm [25]. For evaluating the quality of a typical feature set often Shannon's Entropy measurement is used, this measurement tries to describe the data in terms of how uncertain we are about the true class of the data, the lower the entropy (chaos) score the more certain we are. In [26] a lengthy comparison is done on all kinds of feature evaluation techniques. Liu and Motoda also point out the importance of a quality distance measure to compare feature vectors. Of course this relies on the type of feature we are considering. In the book they distinguish between the following types of features; complex, continuous and discrete, with discrete values having an ordering; ordinal values or without order; nominal features. For the receipt classification of Dealer Management Systems all of these types of data are considered, just a few papers could be found that concentrate on how to take on mixed feature-types, the most research on this topic is done in the context of symbolic clustering using unsupervised learning methods, the best set of features is sought to maximize the clustering performance[27][28], such best feature set selection might also be useful when selecting which fields should be used when normalising the data.

Major parts of the data that will be processed will contain textual descriptions, since these fields are of the utmost importance for the manual labeling, correct processing of these fields is crucial to achieve accurate results. Text mining and classification is therefore an important part of this thesis, thankfully information retrieval and text mining are fields on their own and a lot of research has been done already. Sebastiani addresses the text classification and mining tasks development throughout the years in [29]. Often co-occurrence of words, so-called n-grams, are used to classify text collections. Such algorithm will go through a document and register the co-occurence of a number of n words, Shannon showed that this (semantically incorrect) assumption performs statistically well. The *bag-of-words* approach can be considered a special case of the n-gram algorithm for $n = 1$ (unigrams), this way the features are solely based on single words within the text and word order and surrounding words are neglected [30] altogether. Other popular techniques are based on the n-grams like the term frequency (*tf*) or the model that takes into account the inverse document frequency (*tf-idf*)[31].

One of the most challenging parts of this thesis is the fact that the messages that are to be processed are relatively short. Classification of short messages are, on the other hand, not novel at all. An example is the sentiment analysis on the basis of micro blogging service Twitter;

Go, Bhayani, and Huang proposed a sentiment classification technique on the basis of distant supervision. Their approach is quite unique and mainly focused on sentiment using emoticons, but they show that small corpora still qualify to be classified statistically [32]. Bobicev and Sokolova use an approach based on the compression technique PPM; it uses character context models to build a probability distribution for predicting upcoming characters in the text [33]. Patel and Bhatnagar propose a system for classifying text messages (short messaging service; *SMS*), they show a method that involves five steps using term frequencies with preprocessing using stop-word removal and stemming. The features that they select are on the basis of term weighing and Principal Component Analysis, a neural network is used as classifier. Unfortunately they didn't implement the system themselves and therefore no evaluation results are shown in the paper [34]. In the paper "Short text classification using very few words" short corpora are estimated in categories by shortening the brief texts even further, the key is to keep just the words that represent the content the best. Sun makes use of both *tf*, *tf-idf* and suggest a custom scoring function; *clarity*. Despite their results, which are just up to par, they advocate that their method has great potential [35]. Then there is the approach of Duan, Li, and Huang, they make use of rough set theory. Which is based on a mathematical approach on data vagueness, described by their boundaries. A nearest neighbour approach is used to predict the true class of unseen data, their results section show good results with F_1 around eighty-six percent [36].

The data that is processed every day is coming from a variety of Dealer Management Systems, this means that data that comes from different sources contain other types of information and the ways that it is represented varies per DMS. If a classifier is to be trained to cover multiple DMS' the selection of the features is a challenge. The main issue is the fact that uniform descriptors are expected by classifiers. To cope with this, research has to be done in the most effective way to fuse the data from the different sources into valuable features. This problem is related to data cleansing[37][38], where a solution is sought for finding near duplicate entries in the database. Further there is the data fusion problem where data from multiple origins has to be determined which parts of the data should be used from which source to be able to find the best combined state [39], an adequate answer for this problem will help in creating a good generalized data set. An other problem that is more closely related is the schema integration or data integration problem, Batini, Lenzerini, and Navathe show a spectrum of different approaches and compare schema integration methods. Thieme and Siebes describe a schema integration method in relational databases, and this approach is quite interesting and can partially be translated to the problem we are to tackle; the different DMS' data entities can be considered as different objects they describe, the hierarchy can be seen as an inheritance over the DMS entities. They formally propose a method for factorization and normalisation of subclasses into a single schema [41]. As for every field; domain knowledge is important for the selection and normalisation of the data during this thesis.

Chapter 3

Data Understanding

3.1 Feasibility analysis

RDC serves numerous dealers with their CaRe-Mail notification system. To provide the customers of a dealer with useful notifications and doing this at the right moments, RDC uses the DMS' of their clients as main source for input. CaRe-Mail sets up the right information at every configuration and this is pushed through numerous channels towards RDC, ready to be parsed and evaluated. Since all the Dealer Management Systems differ in format and fields the parsing of the data is quite complex and contains a lot of special cases. The acquisition of the data is just as complex. For example; one dealer might provide direct access to their database, while an other sends a structured file containing the information. One DMS configuration sends their information with a history of one month, while an other uses retention of a whole year, this data is passed in typical formats like XML and CSV. In this section I will attempt to shed some light into the different information streams that orbit around CaRe-Mail and how this data is processed. In the end I will try to summarize the data that is received and couple a decision, whether we deem that the data suits the problem we are trying to solve in this thesis, or not. Next follows a complete list of the Dealer Management Systems that are used with CaRe-Mail;

- Autoline
- Carfac*
- CarIT
- CARLO DMS
- CarSys
- Darts**
- D'ieteren*
- Driver
- EVA DMS
- Gids
- Grand Prix
- I'Car
- iDAS
- Light
- MegaCar**
- SternRent
- WinCar
- XPower*

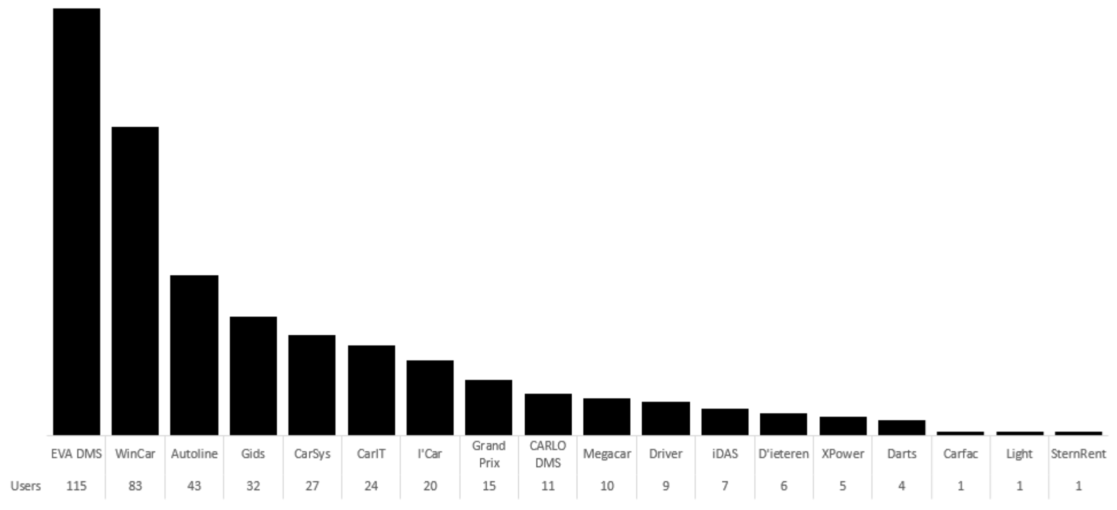


Figure 3.1: Usage of Dealer Management Systems

* - *These are the Dealer Management Systems that are solely used in Belgium.*

** - *Dealer Management Systems that are end-of-life, and are phasing out.*

Major, commonly used and remarkable Dealer Management Systems are mentioned here, for the feasibility analysis data samples are added. Because of privacy concerns all of this data is anonymized; license plates are randomized, identifiers modified and names are changed, et cetera. As shown in the list there a lot of Dealer Management Systems that are supported by CaRe-Mail, but not every DMS is just as popular as an other. In Figure 3.1 we see the distribution over all the supported Dealer Management Systems.

3.1.1 General

The data RDC receives contains at least three sections *Klant* (customer), *Auto* (car) and *Werkplaatsbezoek* (workshop visit, the invoice lines). Some Dealer Management Systems additionally serve more sections like; *Plan* (scheduled maintenance), *Bedrijf* (company), *Verkoop* (sells) and *Lease* contracts. Since all entities in the files are relationally connected to one another each entity has a unique identifier. As mentioned earlier both XML and CSV formats are used, the actual format is not determined by the DMS itself, but this is configured during installation. The last few years a transition is made from CSV to XML and this means that dealers that connected to CaRe-Mail the last few years will use an XML format.

Klant

Every record contains personal details including name, surname, sex, address, city, zip code, date of birth, telephone number and their e-mail address. Typical fields that are custom per DMS are salutations in a letter[1], fax number and possibly a chain store identifier that the customer is associated with. Some Dealer Management System configurations even contain a flag for

[1]In Dutch we call this *Aanhef.* The words to be put in front of someones name ex.: *Dear mr.* Lentink or *Dear mrs.* Jansen

identifying whether a customer is passed away. A perfectly suitable unique identifier would of course be the nationally unique *burgerservicenummer*, the Dutch equivalent of the social security number, it is however prohibited to store this number in digital systems outside of the medical domain or government. For this reason identifiers are usually auto-incremented numbers.

Auto

The *Auto* section lists all the vehicles that are associated with the dealer. While the vehicles often are passenger cars, they don't necessary need to be. In some systems the license plate serves as the identifier, others use an auto-increment, or the Vehicle Identification Number (VIN) which is a worldwide unique identifier. The brand of the car and the specific type are usually stored, along with the fuel type (diesel/benzine/electric/hybrid). A few Dealer Management Systems also store information about *deel 1* and *deel 2*, the so-called first and second part of the license plate and transfer certification that couples an owner to the car. Some DMS' have fields for the next planned maintenance or the next routine checkup date, these can be either planned or expected.

Werkplaatsbezoek

The *Werkplaatsbezoek* in the documents, often abbreviated as *WPL*, contains the receipt of the finished maintenance done to the cars associated to the dealers. Most systems choose to store receipts as a single entity, with a single description and price summarizing all labour and parts. However other DMS' store the invoice line-based. In the latter case every line describes labour or parts separately. Together all the lines with the same identifier form a single bill. Generally the descriptions in this type of storage are more elaborate than the former way of storing receipts.

3.1.2 The different Dealer Management Systems

Although a typical DMS essentially delivers the same fields, also within different configurations of the same DMS others fields can be used. Fields depending on the configuration, will not be explained in this section. With the exception of fields that may contain extremely useful information for the data mining task.

iDAS

The iDAS Dealer Management System serves a file containing three sections; *Klant* (customer), *Auto* (car) and *WPL* (workshop visit). In Table 8.4 we see a sample of the *WPL* section of the file. As can be seen the *Omschrijving* (description) field is quite elaborate about the type of service that is performed on the car, some description fields even start with the work code number, as can be seen in line 5 of Table 8.4. There are also description fields which are left empty altogether, in the current system these are skipped and never used. Classifying these fields won't be an option since no ground truth is known for these.

dmsId The id of the Dealer Management System that generated the output.

auto The license plate of the corresponding vehicle. This field is a foreign key from the *Auto* table.

klant Is the foreign key value for the customer identifier from table *Klant*.

filiaal The chain store identifier that generated the receipt for the car.

type A categorical value that should distinguish labour from parts.

soort The type of invoice.

datum The date that the vehicle was checked in.

kmstand The mileage of the vehicle.

bedrag The total amount that was charged for the maintenance.

factuur The unique identifier of the invoice.

factuurdatum The date that the invoice was generated.

order The id of the associated order.

afdeling The division identifier.

credit Factorial value which flags if it either contains a credit note or an original invoice.

opdracht The short description of the work that was done.

I'Car

All the samples inspected from I'Car Dealer Management System were in CSV format, a default configuration contains only the standard sections. For such construction the fields of the *WPL* are explained in the description list below.

AUTOID The unique identifier of the car.

FACTUURDATUM The date when the receipt is generated.

FACTUURNUMMER The unique identifier of the invoice.

WERKPLAATSDATUM The date that the car entered the workshop.

FILIAAL The workshop establishment that the car entered.

BEDRAG The total cost for the maintenance.

KMSTAND The mileage of the car upon leaving the workshop.

INTERNOFEXTERN Factorial whether the maintenance is internal or external.

ONDERHOUD The maintenance description done to the vehicle.

One of the custom setups exposes an extra section; *WPLD*. In this section (see Table 8.3) the invoice is split into a sequence of invoice lines. A line contains a portion of the labour with costs, description and car parts used for that maintenance. Apart from these receipt lines, there also are section lines (*ISSECTIE* = JA) which seem to only be there for markup purposes, a section has multiple lines that are part of that section. A typical section could be a tire swap, the lines that fall within this section could be for example; the actual tires with their part code, the labour hours for the swap and the bolts that were used. Labour is distinguished from parts by the *arbeid* (labour) field. It either has the value *O* for *onderdeel*, a part, or *A* for *arbeid*. After inspection of the data we see that the *A* flag is used much more than *O* and it seems that *A* is used for charging both parts as labour descriptions. In fact the total percentage of *O* lines within the database is just under one percent.

There exists also are lines that have a negative amount, an example can be seen in Table 8.3, second row. Negatively signed numbers seem to be used to round off the total price or to meet an earlier agreed amount in the case the costs exceeds the price that was arranged on forehand. The third row actually isn't a replacement description it is an advice for the customer to replace their timing belt. This means that these invoice lines sometimes are used as notes, which need to be filtered out carefully otherwise it might pollute the data. When the customer decides to follow the advice separate lines are added for the labour and parts.

Autoline

Autoline serves a XML document as can be seen in Figure 8.1, it distinguishes the sections; *Bedrijf* (company), *Klant* (customer), *Auto* (car), *Plan* and *WPL* (workshop visit). In this document every *dmsdata*-element in *WPL*-section is a single invoice.

dms_id The identifier of the database that it is generated in.

autonr The car identifier which is connected to the *sleutel* of the database.

factuur_bedrag The total price of the invoice.

km_stand The mileage that the car has at the moment of the workshop visit.

bezoek_datum The date when the customer visited the workshop.

vestiging_code The identifier of workshop that was visited.

menu_code The code that is associated with the type of maintenance in the input menu.

factuur_nummer The unique invoice identifier.

details Open details field for the maintenance description.

tarmagic Identifier for a specific link, in this case coupled to the customer.

The field *menu_code* quite interesting and serves as a sort of categorization of the work done to the car, typically this field contains an abbreviations of the type of work. An example is tires, shown as *BAN* (abbreviation of *Banden*) and *LIC* for lights, classification based on these codes solely would be wrong since not only replacements but also repairs fall within these categories. At the end of this section the classes are listed, one of them is tire swaps. It might be that the *menu_code* equals *BAN* which indicates tires, but that a rim change has been done, in that case we do not want it to be flagged as tire swap, in that case we will need the *details* field for clarification.

WinCar

WinCar is actually the first Dealer Management System that is being used during the transition of the CaRe-Mail rebuild. The Windows oriented DMS WinCar has adopted the XML file format. It has no additional sections apart from the fields that are default.

dms_id The identifier of the database that it is generated in.

faktuurnummer The invoice identifier which is connected to the *sleutel* of the database

kenteken The licence plate of the car that visited the workshop. For this DMS the license plate is the unique identifier of the car.

klantnr Klantnr is a abbreviation of *Klantnummer* which stands for customer number. The unique identifier of the customer, coupled with an entity in *Klant*.

omzetsoort This field indicates the *type of revenue*. It is used to distinguish lease maintenance from private maintenance, work for car exports and sales for example. This is an factorial field having around thirty levels.

p_totaal The total price of the invoice. Including taxes.

p_btw The tax costs of the invoice.

kmstand The mileage that the car has at the moment of the workshop visit.

werkplaatsdatum The date when the customer visited the workshop.

door The name of the mechanic that worked on the vehicle.

vestiging The identifier of workshop that was visited.

omschrijving A description field. Categorical

pakketten Work packages involved. Typical packages are maintenance that is done often like *APK*, *annual checkups* and *replacing breaks.*

artikelen The parts that were associated with the workshop visit.

werk Open details field for the labour done to the car

codes Identifier for the work done to the car.

The fields *pakketten*, *codes* and *artikelen* contain comma separated lists of categorical fields. These fields indicate the work done to the car and their parts and since they are categorical they are quite easier to interpret then full text fields. As can be seen in the Figure 8.2, the sample, the DMS data can be both original invoice lines but also credit notes, in case of a credit note the original invoice exists also within the file. Although the codes give an easy way of interpreting the contents of the workshop visit, these codes are not always set. The last record in the sample shows the invoice of a *Ford Fiesta car manual*, this record is a example of a record that does not contain the *codes* element. Admitting, the majority of the records do have a non-empty *codes* field, but there do exists records with this field left empty.

Carlo

Carlo DMS is the product of *ThinkRIT*, a Greek company founded in 2009. Their Dealer Management System serves as one of the largest amongst dealers for Opel cars. For CaRe-Mail they deliver the standard *Klant*, *Auto* and *WPL*. There exists however configurations that also provide custom sections like *Vest*, which includes information about different corporation establishments. These configuration fields are ignored by CaRe-Mail.

vin The chassis number of the car. This is the unique identifier for the car in the database.

bedrag_inc The total costs including taxes.

km_stand The mileage of the car on entry of the workshop.

header Short description of the maintenance that was done.

description Elaborate description of the maintenance, including the parts used and labour.

locatiecode The unique identifier of the workshop that was visited.

documentsoort The type of document.

ordernummer Unique identifier of the order.

factuurnummer The invoice unique identifier.

factuurdatum Date that the vehicle entered the workshop.

intern Boolean indicating if the workshop visit was internal or external.

As can be seen in the sample every receipt line is stored in the description field in a XML-like structure surrounding every line with a *Description* tag. This field is required and is never left empty, the header field on the other hand can be empty. In fact just over 40 percent of the values this section were left empty.

EVA DMS

The EVA DMS has a lot of custom fields in comparison to other Dealer Management Systems, if we look at the *WPL* section we see 24 fields. To describe the work on the invoice it both contains descriptions, multi-line, but also codes both for the type of line on the receipt (*soortlijn*) but also for the work that has been done *bruto*.

dms_id The identifier of the database that it is generated in.

dms_db The database used for the storage.

auto_nr Unique identifier for the vehicle associated with the workshop visit.

factuur_bedrag The total price of the invoice.

km_stand The mileage the vehicle has during the visit.

bezoek_datum The date of the workshop visit.

nummerplaat The license plate of the vehicle.

qwaarde A field that associates the type of location, workshops for certain type of car.

factuurnummer The unique identifier of the invoice.

intern Value indicating whether the visit was internal.

factuurdatum The date that the invoice was generated.

locatie The unique identifier of the workshop that was visited.

omschrijving A delimited string containing a description of the work done to the vehicle.

bruto A delimited list of work codes of the description list.

aantal A comma separated list of numbers containing the amounts of units of parts used.

soortlijn Comma separated list of types of description in *omschrijving* field.

lijngeencorrectie Comma separated list, indicates when a *lijnsoort* at that index is empty or undefined.

interventiecode Comma separated list of variables used to fill the *soortlijn* field. Typically this is an array of alphabetical values, fields that don't need filling have undef.

soortuurcodelijn This field has a comma separated list that indicates a type of labour hours.

kch_oms Factorial field used for warranties, work for make a vehicle ready for delivery etc.

fd_oms This is a similar field to *kch_oms* often containing the same value, sometimes a different one.

bedrag_intern The costs made internal.

merk The brand of the vehicle.

klantnummer The unique identifier of the associated customer.

Gids

The *Gids* Dealer Management Systems is the fourth most used DMS that is connected with CaRe-Mail. Gids have the sections *DMS*, which stores information about establishments contact information. Then they have a customer section, *KLA*, a vehicle section, *ATO*, a section that holds the salespeople *VKO* and a section with cars that are ordered/bought *VVO*. For the invoices there is the *WOK* section that holds complete invoices as single entities. There are, however, configurations seen where there also was specified a *HIST* section which contains separate invoice lines that refer to the *WOK* via the invoice identifier. After checking the code of the current system we discovered that this section never was used, missing actually interesting data. The field definition of the *HIST* section is shown below.

car_id The license plate of the car.

datum The date of the workshop visit.

factuurnummer The unique identifier of the invoice, refers to the WOK section.

code The code associated with the work.

ks The mileage that the car had driven.

oh Boolean indicator whether this visit was due to maintenance.

klant_soort Categorical field (P = Person, B = Company or L = Lease company).

type Categorical field containing one of the following values; (ON,R,O,K,D).

omschrijving Description of the work done during the visit.

aantal The amount of units, one when not applicable.

stk_prijs Unit price of this line.

CarIT

From CarIT different configurations have been seen, with information both formatted in XML as CSV. The example provided is one of the CSV format, see Table 8.1. The following field definitions exists for this Dealer Management System;

car_id The unique identifier of the car.

sec_inv_amnt_exc The total costs of the workshop visit.

woh_mileage The cars mileage after the workshop visit.

woh_ord_date The date that the car entered the workshop.

factuurdatum The date that the invoice was drawn up.

release_date The date that the car exited the workshop.

userid The customer associated with the car.

whs_id Chain store identifier.

onderhoud Boolean indicating whether the visit was for a maintenance checkup.

rob Boolean indicating whether ROB Net transactional system was used.

As can be seen in the definition and the provided example this DMS does not contain a field for work description or codes. This simply makes it impossible to determine the type of work that has been done to the car. The *onderhoud* field does specify if a maintenance was done, but apart from maintenance it the *WPL* section does not provide useful information for predicting other classes. CarIT does provide an extra *Plan* section with the planned workshop visits. This section unfortunately does not provide any extra information on the required data. Since the current system only targets on recognising maintenance, it contains exactly the needed information, for the classification for this thesis however the data is lacking valuable information.

3.1.3 Normalized data

For the current system all data from the various databases are normalized into a single database. This database stores the receipt information as separate lines and classifies them on the basis of every line and not on the basis of the complete receipt. In this database only the fields were kept which could contribute towards successful classification. The schema has the column *Oh* (onderhoud - maintenance), which is the field that indicates if a maintenance is detected in the line.

Id The unique identifier within the database.

Auto dms id The unique identifier within the original DMS.

Bezoek datum The date of the workshop visit.

No The line number of the invoice.

Oh Boolean indicating whether the current line is classified as maintenance service.

Soort Type of revenue.

Code The code that was associated with the maintenance.

Tekst The textual description of the maintenance.

Rule The rule that classified whether it is a maintenance or not.

The first column, id, is the unique identifier of the receipt line in the normalized database not the database where the record originates from. As mentioned earlier a receipt is split up by its lines, but the original receipt can be reconstructed by querying the database for *Auto dms id* and the corresponding *Bezoek datum* ordering by the *No*, the receipt line number.

In Table 8.5 we see a sample of the normalized data, the records shown in this example have the normalized fields of a WinCar Dealer Management System. That the current system still isn't flawless can be seen in Table 8.5. WinCar has both work codes and description fields (see Section 3.1.2), both could be used for classification and the system shows on which basis it classified a certain line. The fourth row in the sample clearly states that a maintenance has been done, but since the code of the receipt line 000001 is associated with *Miscellaneous* it ended up as a false negative, because the code was the first criteria of the system. As can be seen in the column *Rule*, all the classifications were on the basis of the *code*-field, if a classification was done on the basis on the *Tekst*-field *Rule* would have referred to *Tekst*. The normalized data that the current system emits may also be used to train a classifier, but as can be seen some potential useful information is filtered out this view. For example workshop visit costs is missing, which may be useful in determining the class. For this reason one of the research questions focuses on effectively normalising data from multiple sources.

3.1.4 Multilingual

The CaRe-Mail product widespread under dealers and workshops, the most of their clients are situated in the Netherlands, but also a few in Belgium. Belgium, that consists out of two parts; Flanders and Wallonia, is bilingual. This addresses a new challenge in regard to the processing of the free description fields. The Dealer Management Systems used in Wallonia, the French-speaking part of Belgium, will certainly be filled with French also used in Flanders and the

Netherlands. Since there are proportionally less French clients the data provided will also be less. In order to let the model also recognize the data from French fields correctly sufficient data from these dealers need also be labeled and added to the training set. In the end it might also be very well feasible to train separate models for the different languages.

3.1.5 Judgement on feasibility

The extent of richness of the data seems to be very dependent of the Dealer Management System we are looking at. CarIT can be considered as a DMS that delivers few data, whereas WinCar is one of the most rich data. The former does not contain any textual description or indication what was done to the car, merely a field that indicates if the work that was done to the car was a routine maintenance. And the latter containing multiple fields including codes, packages codes and textual descriptions. This directly addresses the problem that some DMS' or configurations might not be suitable for recognition of the newly suggested labellings. The lack of descriptive information makes it simply impossible to recognize some classes, the DMS example of iDAS (see Table 8.4) is a configuration for which the description field is only used for maintenance descriptions.

Also of some classes more examples will be available then others simply because they occur less often; the prior probability of it occurring is much smaller, for training this must be taken into account such that the truth data provided will not be skewed because this will hurt the classification accuracy. The major part of the ground truth data was labeled by queries and checked by hand, this means that there still may be some small amount of noise in the labeling, but we are quite confident that this is not problematic during this thesis research.

As seen in Figure 3.1 there a total of 18 different Dealer Management Systems that are supported by the system. But looking at the distribution we see that the first three Dealer Management Systems together are used more often than the others. To make sure that the new system will perform at its best for the most customers we need to take the distribution into account when creating the normalized data and training the models.

Typical language processing models take parts of the corpus and use proportional frequencies to create classifiers. These models however often omit numerical data and special characters like hyphens, quotation marks etc. Such characters are however often used within the description fields for work codes, types and serial numbers of parts. To be able to create the best classifier possible these descriptions may be very useful, during the creation of textual models this has to be taken into account.

In conclusion the data seems to be of the right quality to be used for classification purposes. Some Dealer Management Systems simply don't provide the right information to detect certain classes in the data, which is no problem for now, since CaRe-Mail doesn't detect those classes at this moment in the first place. Further the data from all the Dealer Management Systems do not directly fit well in training a model, for this a normalisation step has to be done.

3.2 Current system

Since CaRe-Mail is an existing product within RDC, they already have a current solution for their product. This solution has been built throughout several years in the language Pearl. Every Dealer Management System has a custom implementation for the classification of the receipts, this means that variations in input from the Dealer Management Systems will immediately be felt in the change of recognition rate. The classification of the data takes place in a few steps.

At the side of the dealer the software first is set up to send data toward RDC every night. A dealer has the so-called CaRe-Mail ROC hub (read-only connector) in place that is responsible for sending the data. This hub directly connects to the Dealer Management System that they have installed. It often is a physical single board computer that checks periodically if information was set staged for it to be sent toward RDC. The connector has a manual implementation by RDC and has no association with the Dealer Management Systems they support. Most companies behind the Dealer Management Systems aren't co-operative in providing the right data, the information that is sent by the ROC hub is gathered manually and selected earlier by RDC when they started the CaRe-Mail system.

Now that the data push is set up on the dealers side, we need to interpret that data. Since data that is pushed towards CaRe-Mail contain no delta information, and in fact could already be processed already a delta is done upon receiving the files. This difference check happens file-wise, and only changed and new entities are then read in by the system for processing. All connected dealer have a custom integration for CaRe-Mail. This means that a lot of maintenance is done for many implementations. All these customers have their own database with their invoice lines, cars and other sections of the files that their Dealer Management System delivers. All of dealers that have the same DMS share the same parser, but within these parsers there are still deviations. One might use a code for one type of work while an other uses the exact same code for other labour or parts, others do share codes or descriptions. During a classification pass different regular expressions are evaluated against the invoice line, if they match the lines are marked as potentially maintenance. For these potential maintenance lines a second check is done where the line is ran against a set of exceptions regular expressions (like: 'last' or 'advice
s?:'). If none of the exceptions' regular expressions match, an invoice line is finally marked as maintenance. Regularly dealers contact RDC for refining their recognition system, they request if certain terms or expressions could be added to the expression list.

The classification in the current system is based on the recognition of maintenance and whether a maintenance was marked as internal or external, other classes are not recognized. The recognition of maintenance and internal visits are unrelated, the internal flags simply marks if the workshop visit was not charged to the customer, this could be maintenance that is done to make a car ready for delivery. Or may be work that has to be done to the car because of damage that was caused by the mechanic. Because we typically do not want to bother the customer with these kind of repairs CaRe-Mail distinguishes internal and external repairs. Various Dealer Management Systems have a boolean field indicating whether the workshop visit was internal (see Section 3.1). As for the maintenance recognition, we saw earlier in the *Normalized data* section (see Section 3.1.3) a bit is added serving as a boolean indicating maintenance.

Chapter 4

Data Preparation

4.1 Ground truth

RDC possesses a few hundred receipt lines for which the true state is known. This information will be used for the training procedure of the statistical models, but it is not directly useful since the ground truth is only known for the classes used in the current system; maintenance or not. Further there exist deviations in the number of available labeled data per DMS. For some of the Dealer Management Systems more truth data is available than others and few don't have examples available at all.

The product owner of CaRe-Mail put out a list of the most valuable classes that are to be recognized in the invoices. The list exists of vital parts for the car's functioning, parts that people are anxious about when they malfunction and services and checkups that are of a periodic nature. These criteria resulted in the following list;

1. Routine maintenance
2. V-belt
3. Car battery
4. Timing belt
5. Tire swaps
6. Air conditioning
7. Brakes

Figure 4.1: The class labels defined that should be used during classification

As mentioned earlier the current system just divides the data into two classes; maintenance and non maintenance. For the new system the list in Figure 4.1 is used, of course this is an incomplete list of all classes that could be distinguished in the data, for this reason an additional class is introduced; *other* which is collective class for everything that does not fall within the classes in Figure 4.1.

4.1.1 Manual labeling

For the all Dealer Management Systems we have labeled all the data manually. An example for DMS I'Car, a file received in December 2015, containing almost 330,000 receipt lines. When we filtered out all the section lines (see Section 3.1.2) we were left with just under 250,000 lines. First we ran the expression list that is used in the current system (see Section 3.2) leaving out the lines containing words from the exceptions list for the routine maintenance classes. For the other classes we composed our own lists of words that identified these classes, typical words would be the translations of the class names, abbreviations and synonyms. But since context for the receipts often is extremely important for the class label, everything has to be checked manually afterwards.

Then the automatic labeling a manual pass was done over the labeled data to verify the labeling and correcting them when errors were found. Over one third of all records were given a class label using the automatic labeling pass. Around 10,000 of these needed manual adjustment in the next pass. For a small amount of the data the ground truth could not be determined, these records were discarded, together with multilabel receipts. Because labeling by hand is quite labour intensive we used grouping queries for our database to label more efficiently. Grouping works pretty well because of the large number of the total order lines share the same description and price. In the end all data was labeled using the expression lists and by hand with the help of grouping queries.

Class	#	**Class**	#
Maintenance	1522	V-belt	3
Timing belt	11	Brakes	6
Tires	1770	Airconditioning	19
Battery	13	Other	6493
Mixed	55	Total	**10020**

Table 4.1: Manual labeling of Autoline invoice lines

In Table 4.1 we see an example of the distribution of classes among a manually labeled data set. As can be seen some types of labour happens more often than others. Which makes sense since some parts of course have a shorter life span than others and maintenance and tire swaps are of a periodic nature and therefore appear more often. An other reason might also be that dealers often charge more for workshop visits, and that customers prefer cheaper garages that are less expensive.

An other challenge arose while labeling; the ambiguity of some words. While labeling the samples from an I'Car DMS I came across some invoice lines (in a *WPLD* section) with the *OMSCHRIJVING* having; *RIEM* which is the Dutch translation for belt which is ambiguous. It could be the V-belt just as well as the timing belt. In fact there are more parts within the car that can be indicated with belt. In this DMS the *CODE* field gave the definite answer on what kind of belt we were looking at; it contained a code number of the car part. When I did an online search on the car part type number, in combination with the car brand I quickly found that some of these lines were in fact V-belt replacements and some timing belts. But I also discovered that some of these lines referred to the replacement of a fuel tank strap, which is a type of replacement we don't want to classify at all.

4.2 Invoice entity

In the previous sections we have seen that data is stored quite differently in the Dealer Management Systems. To cope with these dissimilar information streams a step is proposed for making the data uniform, this covers a great portion of this thesis. But a decision that is just as important choice as the normalisation of the data is what we want to assign as main entity for classification. In this section the options are listed and for each of these their advantages and disadvantages are mentioned. An assessment at the end concludes this chapter and is considered fixed throughout the rest of this project.

While looking at the data feasibility study that was done earlier (see Section 3.1) we see that there are essentially three different types of invoices; *complete invoices*, *invoice lines*, *combination of complete invoices with invoice lines.* The difference in these types of invoices are in the way they are built up; a complete invoice contains everything that was done during the workshop visit in a single entity, there is no separation between different tasks and parts recognizable on the receipts. Then there are the invoice lines, these consist of merely a single line of work. When all the lines are put together that share the same invoice number a complete invoice can be recreated, often this type of invoice storage has some degree of data redundancy often in the form of storing the date, invoice number and customer identifier. And lastly there is the hybrid solution between these two former invoice storage mechanisms. These Dealer Management Systems store the actual invoice in a separate table of the lines that it is built up of. The actual invoice will, for example, contain all the global information like the customer, date, car identifier and mileage, whereas the lines will contain the detailed information with work descriptions and car parts.

4.2.1 Complete invoices

The complete invoices have as advantage that they store all the available information for a single receipt at one place, this makes sure that the training procedure has sufficient data to work with. The main disadvantage is that within a single invoice multiple types of repair may be done. It is not unusual that the timing belt replacement is done along with the V-belt, when this occurs these receipts can not be used for training. When normalising invoices that are stored in other formats, this type of storage has as advantage that it does not need any heuristic to build up the invoices, the other approaches will need some sort of splitting method for separating a single invoice to multiple lines. While some Dealer Management Systems do have line endings, comma's or periods that may be used for this but if a wrong splitting method is chosen it will affect the performance.

4.2.2 Invoice lines

The other option are invoices that are separated, these have the advantage that they less often describe multiple classes. Distinct invoice lines, however will make a proliferation on the amount of entities that has to be processed; an invoice has an average of 13 lines. Because multiple entities per invoice will be classified the results of these classifications may contradict each other. Some fields indicate routine maintenance, while others might indicate that it was actually a winter checkup (a optional inspection that can be done before a winter sports trip).

4.2.3 'Hybrid' invoices

The hybrid solution has as biggest advantage that it no redundant information and per line stores the price of certain part of the invoice, but also has the complete invoice entity which then stores the complete costs. This approach has however two disadvantages; this first being that it is extremely hard to convert other DMS to this format and the latter that the data is essentially stored in two different entities which is not supported for generating models.

Now we have seen the different (dis)advantages of the different invoice entities we will start off by referring back to the statement made in the beginning of this section; we are looking for a single entity to be used for classification. And because the third way of storing essentially deals with two entities (that are strictly coupled) we do not consider this type of invoice storage as desired. This leaves us with two possible candidates; the complete invoice entity or the line based entity. The CaRe-Mail team has the ambition to recognize every work that is done to the car and make it possible for the customer to select customers that had recent maintenance concerning some parts. Entities on the basis of complete invoices seems for this reason less attractive since during a repair often multiple parts are replaced at once to save labour costs, in practice workshop visits are generally multiclass. Different binary classifiers for each class might be able to recognize these multiclass invoices, but these classifiers might be hard to generate when we are using the same entities to train them; it might find characteristics of an other, unassociated, class. Although multilabel classification is not considered in this thesis, possibilities to extend the support will be kept in mind for possible future research. A big disadvantage of the line based entities are the conversion from complete invoices to the separate lines. During the data understanding and preparation phase it became clear that the most invoices had some sort of punctuation that could be splitted on to divide the invoice in to small portions that semantically still made sense. For these reason the invoice line based entities will be used; this means that the normalized data structure that is researched will be one on the invoice line level and Dealer Management Systems that serve invoices that are laid out differently will be mapped to lines.

4.2.4 Acquiring uniform data

The first research question mentioned was concerning variety in data fields and how this data could be made uniform to tackle the incompatibility between different Dealer Management Systems. As posed in Section 1.4 we want a uniform schema that covers the largest subset of DMS' and their respective fields which maximizes the evaluation performance of the algorithm. This brings us a maximisation function in which we want to maximize the sum of these two variables. The difficult part of formulating this function is the fact that the weights that should be attached to these variables are unknown and are hard to get by at all. The problem is due to two problems. The first being that there is no clear scoring function for largest set coverage of the DMS' in the uniform schema's and secondly how important is this coverage considering the evaluation performance. To illustrate these questions I will provide two extremes as examples, these were briefly mentioned in the Research Questions explanation.

The first extreme is that of maximising the largest subset; we essentially want two complete set to me formulated in a single uniform schema. For this simple example we exclude all possible transformation. In this case we will take the intersection of all the Dealer Management Systems; and there are just two fields that exist in every DMS the invoice identifier and the date are kept. Intuitively one could already guess that these fields will not help towards correct classifications. Now of course with some transformations (merging/cleaning/splitting fields) other fields could be added/created to keep interesting fields.

The second extreme is towards the maximisation of the classification accuracy. For this we want to make sure that every field is used, obviously in this case we want to chose to use a different model for every DMS. But custom implementations of a DMS might have deviations in fields and sections and we could chose to narrow the classification scope even further and generate a model for every implementation.

Now we will try to find the best accuracy/DMS coverage ratio. The fields of the different Dealer Management Systems will be mentioned and then explained how they could contribute towards towards better classification results. When a field is not present in a certain subset of DMS' or might have an other format an explanation will be added how the information is transformed and used ultimately.

4.2.5 Invoice details

We start with arguably the most obvious field, the invoice description field. This field is one of the most important one for the recognition of the data; as mentioned in the Judgement section, at the end of the Feasibility Analysis (see Section 3.1.5), it is extremely hard to determine the true class of an invoice without this field. Hence, this field has to be included in the normalized schema. The only DMS which misses such a field is CarIT, for this DMS the field will be left empty.

If we take a look at the WinCar we see that this Dealer Management System also exposes separate fields for packages (*pakketten* and parts *artikelen*). While these fields contain interesting information, no other DMS has such fields, so we chose to map these fields to the invoice details and create separate invoice lines from the other fields.

Further; the usage in the training model will not be the entire field but will be of the n-gram approach. Every n words will be converted to a field which will contain a boolean or the number of occurrences in the description.

4.2.6 Part/labour

Some Dealer Management Systems have been seen that distinguish invoice lines that contains parts from ones that represent labour. During data labeling in the data preparation phase I came across some invoices lines for which I could not directly determine the class label. An example is a line which had in its details field merely 'distributie', which translates to timing (as in timing belt). Now these types of lines are hard since these parts do not only get replaced but also tightened and may be other maintenance. But since this line was marked as 'part' it is obvious that this is a replacement, since otherwise no new timing belt was needed.

As mentioned in the previous *Invoice details* section we mentioned the WinCar fields *pakketten* field and the *artikelen*. To keep this information we will make the part/labour field categorical with the values *labour*, *part* or *package*. In our normalized model pakketten will be mapped to package and artikelen to part.

4.2.7 Unique car identifier

Every invoice is coupled to a car using a unique key in the DMS. Particular Dealer Management Systems use either the license plate of the car as identifier or the vehicle identification number (VIN). Although these identifiers are of a random, generated nature, they are important for the CaRe-Mail system after the classification was done by the model. For this reason this field will be added to the normalized model and will be omitted during training and classification.

4.2.8 Customer/invoice identifier

Self-evidently an invoice is coupled to a customer, this is done with a customer identifier that refers to a person entity in the customer section (see Section 3.1). Just as the car identifiers these are often incrementally generated and do not add useful information, the same holds for the invoice identifier. For the same reason as the previous case these fields will be added to the schema but will not be used during training.

4.2.9 Chain store identifiers

This field is quite interesting. We'll start off mentioning that not all Dealer Management Systems have fields on chain stores, some for the simple reason that no chain stores are connected and

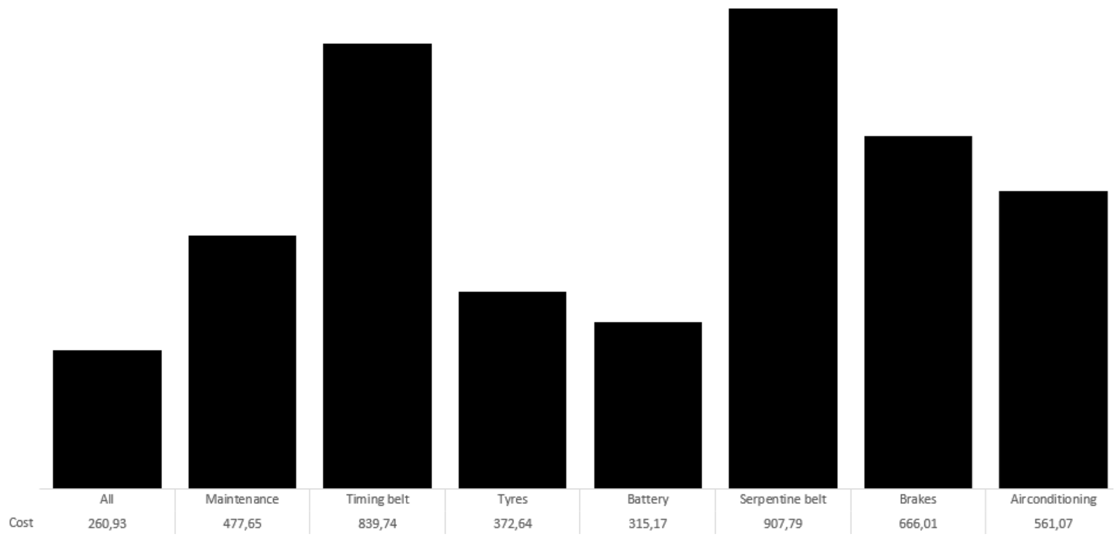


Figure 4.2: Average invoice costs in euro based on a WinCar export

others just do not store the information. Now at first sight this information might not seem to useful, because one might think it should not make a difference if a car was checked in at one workshop or an other. But during the data understanding phase I became familiarized with the data of most DMS' and correlations could be seen between car brands and workshops. The data tells that it is likely that some chain stores are specialized in certain brands. Although this is interesting information, during labeling no correlation was to be found in the class label and the chain store association.

4.2.10 Date

Every invoice has a date on which the invoice was drawn up. Since some of the classes we distinguish are planned for certain seasons, think of snow tire swaps which happen around October/November, we could use this field during class prediction. Since dates are hard to work with for classifiers we might want to convert this date field to a categorical field contains the month of the year. The year itself is not really interesting in this case since we do not provide any information on the maintenance plan.

4.2.11 Mileage

Most of the Dealer Management Systems also register the car mileage when they enter the workshop. Dutch garages are obligated to register and report the mileage of the cars during annual MOT test and repairs for which the cost are above a certain amount. The mileage gives a good indicator for types of maintenance, periodic ones always happen on the basis of the driven car distance. Invoices on which no mileage are provided we might want to fill with *NULL*.

4.2.12 Costs

As mentioned earlier in this thesis the potential significance of the workshop costs were explained. For completeness I gathered some statistical information to back this claim, now that we have labeled the data this is easily possible. As can be seen in Figure 4.2 the average costs of the classes we want to identify are multiples of the average workshop visit invoice charges. This proves that the maintenance we want to identify are indeed costly and that classification algorithms may benefit from having this field. For this reason the price will be added when available.

Now the statistics shown were that of complete invoices, meaning that the prices were the total of the workshop visits, but for the line based storage we often also have the price of a certain part (or labour) available. With these unit prices no other parts/labour cause the price to go up. Unit prices will be added to the schema and nullified when not available, further when total prices are not available but unit prices are we will use the sum of the unit prices as the total costs.

4.2.13 Amount

The amount field often refers to the amount of materials used during the repair. This field is interesting for invoices that, for example, mention the refill of brake fluid where often the amount in millilitres. This field will be useful for telling the difference between the replenishment of fluid or the replacement of the actual brakes; an amount of 200 millilitres brake fluid makes sense whereas replacement of 200 brakes does not. The model may get to the same conclusion using this field. Empty *NULL* values will be used when this field is unavailable.

4.2.14 Invoice header

A few of the invoices also mention some sort of header information. Sometimes these headers are coupled to a few invoice lines and others have a single header assigned to complete invoices. These fields can be either categorical fields (usually containing text like *onderhoud* or *diversen*) or free text fields that leave room for the mechanic to fill. The I'Car DMS has header information stored in so-called section lines, these are the lines that have the value *JA* for the field *ISSECTIE*. For I'Car it is however not directly recognizable which invoice lines are associated with which section.

4.2.15 Codes

The usage of codes are extremely helpful in the recognition of the invoices. Autoline provides *menu_code* information which are predefined codes which can be used to create receipts for all types of workshop visits. These codes are a three character identifier, usually an abbreviation of the work, although most configurations of the Dealer Management System share the same meaning for the codes in the current system there are quite a few custom implementations for CaRe-Mail customers. Apart from the *menu_codes* in the Autoline DMS there are also code numbers in the invoice lines descriptions. The WinCar Dealer Management System has codes associated with work and EVA DMS has the field *bruto* which is an optional field containing the type of work similar to *menu_codes* in Autoline.

4.2.16 Vehicle brand

Although few Dealer Management Systems really provide a field that contains the brand name of the car for every vehicle in the database it is possible to trace back the brand using their license plate number or the *Auto* entity that it contains. Apart from the fact that some car brands need replacement of certain parts earlier than others the data in our database does not show real differences across the various brands. For this reason we will not include the car brand to the normalized schema.

4.2.17 Maintenance

CarIT, as shown in Section 3.1, is the only DMS that contains no free text field containing information about the work done to the car during the workshop visit. This Dealer Management System has a boolean to indicate whether the visit was a maintenance. This field is of course very interesting for the recognition of routine maintenance (class label 1) and should not be omitted since this is the only way for CarIT to recognize this class in their data files. In our normalized schema this field will be represented by a nullable bit which will only be filled if the source contains a value (1 or 0). Otherwise it will be set to *NULL*.

4.2.18 Dealer identifier

Although the dealer identifier itself is nothing more than a auto-incremented number that associates the dealer with the rest of its data, it might come in helpful when it is used together with other fields. Imagine that there are two different dealers which both use the code *OND*. The first one dealer will use the code to identify onderhoud (maintenance) and the latter will use it for onderzoek (investigation). Now for the generation of for example a statistical decision tree splitting on the code *OND* will not result in a high impurity reduction so we may want to add this field to make it possible for the model to split on that field after a split on the dealer. While, traditional classification trees will not decide to split on the dealer id because of the initial impurity reduction, more advanced algorithms are available which look ahead. Also, random forests generate a whole set of random classification trees on random decisions (see Section 2).

4.2.19 Internal visit

A few Dealer Management Systems have a field that indicates whether the car was brought in, or that a repair was done internally (more on this see Section 3.2). Because the field is available in just a few of the systems including this field will result in an enormous amount of empty values, which is not what we are looking for in a data set. Further the field does not appear to have any correlation with one of the class labels we distinguish. For these reasons the *internal visit* field will not be included in the normalized model.

4.3 Approaching the data

Now that we defined the data entities there are still some considerations to make to approaching the data.

4.3.1 Relational entity integration

In many DMS data a relation is defined between, for example, the workshop visit and the car associated. Or a relation between the workshop visit and the customer. These relation may have interesting data e.g. some Dealer Management Systems have been seen where information about planned routine maintenance are included which can be used as an indicator whether a receipt that is considered actually is a routine maintenance. This information, however, will not be included in the current model. The reason for this is that millions of data needs to be processed on a daily basis and these operations result in data joins which press heavily on the evaluation time. Further, the current system also does not join this information to determine whether maintenance took place.

4.3.2 Misspellings

As mentioned earlier one of the challenges is that of the identification of receipt lines even when misspellings occur in its body. This is problem closely related to the approximate (or fuzzy) string matching problem. Often a distance measure is used to distinguish faulty spelled words from correct ones [42], a typical algorithm looks at a corpus of text and checks the word frequencies. Since misspelled words occur less often than correct ones, the frequencies are used to retrieve the words. Next, they are compared to other words using the edit distance. Meaning; what is the minimum of operations that has to happen to transform string a into string b, the operations include the deletion, insertion and substitution of a character.

4.3.3 Verb conjugation

Apart from misspellings verbs can be conjugated in many different ways depending on the tense and context, these conjugations are solely meant for human interpretation but do not contribute to the meaning of the word itself. To make sure that the different forms of the verbs are interpreted as the same word by the algorithm a stemming algorithm can be used. We will look for suitable algorithms to do this efficiently during feature generation. In the end we will evaluate whether the use of this type of algorithm will contribute to the classification score.

4.3.4 Compound words

The Dutch language knows the notion of *samenstellingen*, compound words, this enables an author to write different words that form a unity as one[43]. An example would be the word; *voertuigregistratiesysteem* which means vehicle registration system. In the end there can be infinitely many compounds built up from all words, in informal written Dutch compound words are sometimes written as their separate elements. When these types of writing styles are both used they may affect the accuracy of the classification. In [44] some compound splitting algorithms are considered. In this thesis we will be looking at the possibility in integrating such algorithm in our experiment and take a look how the method changes the models classification performance.

Chapter 5

Modeling

5.1 Models

For the classification of the invoice lines different models will be trained and compared against each other. In this section the main models are briefly introduced and discussed. For in depth overview of the models we refer to the citations of this section.

5.1.1 Decision tree

A decision tree is a hierarchical model that has a finite number of recursive splits on provided features. The trees we will be using are binary and thus has exactly two child nodes or 0 for the special case that it is a leaf. A more formal definition is the following; a decision tree T is a set of $n \geq 1$ nodes; $T = \{1, \ldots, n\}$. Every node contains a decision function which has a binary outcome; either 0 or 1. The possible values of the function are that of its branches. Given an input the tree is traversed through the different nodes evaluating each decision function, when a leaf is reached the output of the local decision function is used as output of the model given the input.

A typical tree is constructed using a data set and an impurity function. Such impurity function is a function that calculates the distance to a 'pure' state, where all data is successfully split into its appropriate labels. A split is generated based upon the *impurity reduction*, which calculates the gain in purity for a chosen split. At each node the best possible split can be calculated and this way a decision tree can be constructed.

In Figure 5.1 we see an example a decision tree model for the classification of Titanic survivors. In each node we see a query that answers either true or false. Following the left child would be if it is true and right when false. Traversing the tree leads to leafs where we predict if a certain passenger either survives or dies. In these leaf nodes the probability of the survival can be seen, together with the percentage of the observations in that leaf.

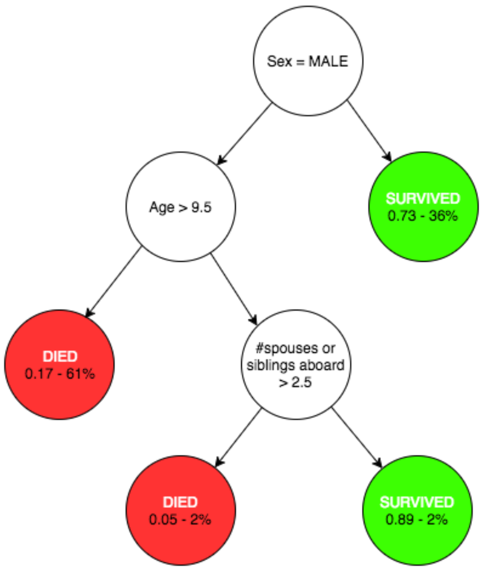


Figure 5.1: A sample of a decision tree showing the survival of the passengers of the Titanic

Decision forest

Decision forests are basically an extension of the decision tree model. There are a lot of variants of decision forests, though the basis of their training and evaluation are largely the same. As the name *forest* indicates it works with a set of b decision trees where the output of the different trees are averaged (or a voting scheme). A typical forest has a large set of trees $100 < b < 10^4$, dependent on the data set. The trees $1 \dots b$ are constructed by sampling a subset of the data to train on. The random forests have a slight variation on this: at each decision node a subset of the available features are randomly selected which can be chosen. Note that the maximized impurity reduction still is used to select the optimal split from the feature subset.

5.1.2 Bayesian network

A Bayesian network is an acyclic directed graphical model that represents events as nodes and edges between the events are conditional dependencies between them. The name Bayesian network comes from Bayes theorem which is where the network is based on. Bayes theorem is on that of conditional probabilities; $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$. Here we have $P(A)$ and $P(B)$ as probabilities for event A and B without regard to each other. $P(A|B)$ is the probability of event A given that event B is true. And $P(B|A)$ is the probability of event B given event A is true, these are conditional probabilities. The Bayesian network has a great advantage; its design makes it easy to interpret and to understand domains by studying the network. One could learn a lot by just observing the events and dependencies within the network about the domain. Also, the algorithm for generating probabilities from the Bayesian network take into account unobserved information very well. A disadvantage is however the computational complexity of calculating a probability of a event happening. An example of a Bayesian network can be seen in Figure 5.2.

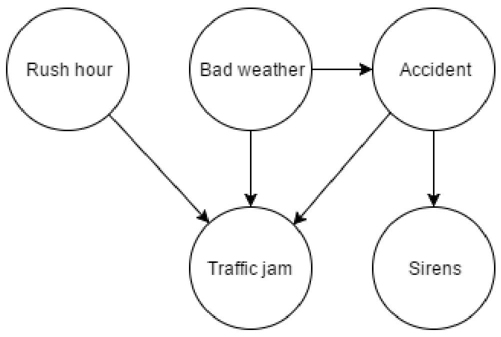


Figure 5.2: An example of a simple Bayesian network

As can be seen it maps the events and causalities concerning a traffic jam. The relation $\textcircled{A} \rightarrow \textcircled{B}$ can be read as event A causes event B, this can be observed with a given probability. Further all the probabilities in the graph are dependent upon the evidence. One might imagine that the probability that there was an accident is greater if we have observed that we are in a traffic jam. And we might be more sure if we also observe that it's pouring rain.

Naive Bayes assumption

As explained the traditional Bayesian networks are quite computationally expensive. Since this is not practical in situations where speed is preferred over accuracy there also exist the Bayesian networks that have a Naive Bayes assumption. For this network we only model the variable we are interested in for prediction and then drop all the edges between other mutual relations with the other variables. This means that every variable has just one parent relation, and this way it is computationally very fast. If we take the previous example we simply specify the class we are interested in as parent, in this case we want to predict if there is a traffic jam, and all other variables we set as its children. Now this graph can be seen in Figure 5.3 and can be read as traffic jam causes rush hour, bad weather, accident and sirens. While this is strictly incorrect the model will performs quite well in most of the cases [18], with the advantage that it is computationally multifold faster.

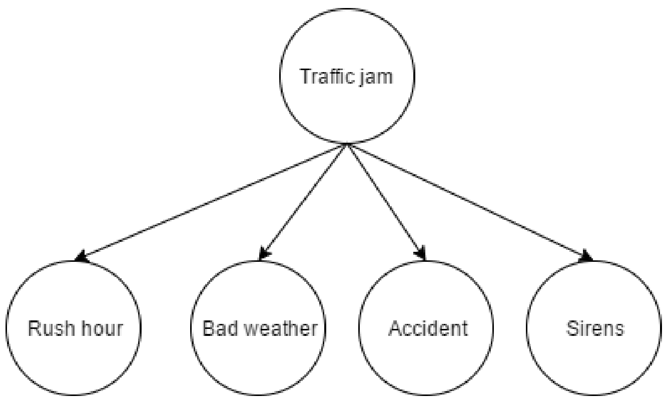


Figure 5.3: An example of a Naive Bayesian network

5.1.3 Support Vector Machine

A Support Vector Machine classifier is a non-probabilistic binary linear classifier. A Support Vector Machine is constructed by sampling all the data in a space, associated with its class label and will look for the maximal margin hyperplane that separates the two classes the best. The margin is the distance between the decision boundary and the closest point. The classifier works using the function $f(\vec{x}) = Sgn(\vec{w}^\top \vec{x} + b)$ where $f(\vec{x}) \in \{-1, 1\}$. The separator hyperplane is defined by $\vec{w}^\top \vec{x} - b = 0$. When looking closely we can see that the position of the hyperplane is determined only by the samples closest to the edge of the cluster of both classes. Taking only these vectors is computationally more attractive, when determined these vectors will be the *support vectors* of the hyperplane. When evaluating the data is sampled in the vector space and based on its position relative to the boundary. The predicted class is the sign of the relative distance to the separation hyperplane.

Now since not all the data is linearly separable the algorithm would not create a suitable decision boundary to separate the data in these cases. To cope with non-linear data Support Vector Machines may use kernel functions where it will try to map the samples to a higher dimension where the sets of samples of the two labels are separable. In Figure 5.4 we see a visual representation of a Support Vector Machine. We have crosses and circles, the crosses represent one class and the circles the other. Now these two classes are separated by a margin. In the figure the lines are the decision boundaries. As you can see both lines separate the two classes perfectly, but the red one does so while maintaining the largest distance between the two classes. So this means that the red line is preferred for our example.

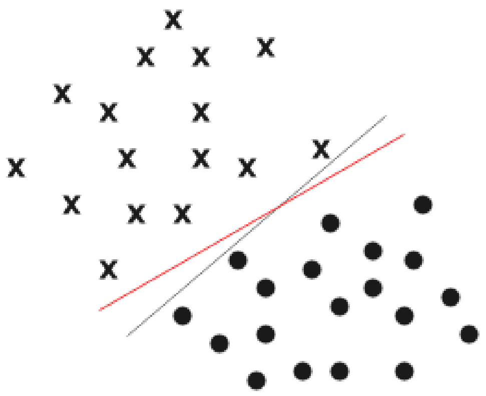


Figure 5.4: Visual representation of a Support Vector Machine

5.1.4 Neural Networks

An artificial Neural Network is a model that is inspired by the workings of the brain, where sets of neurons react to impulses. A network consists of neurons, that are either input, output or hidden. The first nodes are input nodes where data is provided to the network, the input is processed through the hidden neurons, which are divided into a number of layers. A traditional neural network passes the information from input towards the output layers passing connected neurons in each subsequent layer. In each neuron there is a transformation function that transforms its input and propagates it further through the network. When training the network with backward propagation we consider the output of the network and compute the error in comparison with the desired output. We do this for every neuron from the output backward to the input to find the local error in every neuron. After the training, the network is ready to evaluate against new input, all the output neurons will receive a value. For classification we need to examine all these outputs of a neural network and return the class that corresponds to the output with the largest value.

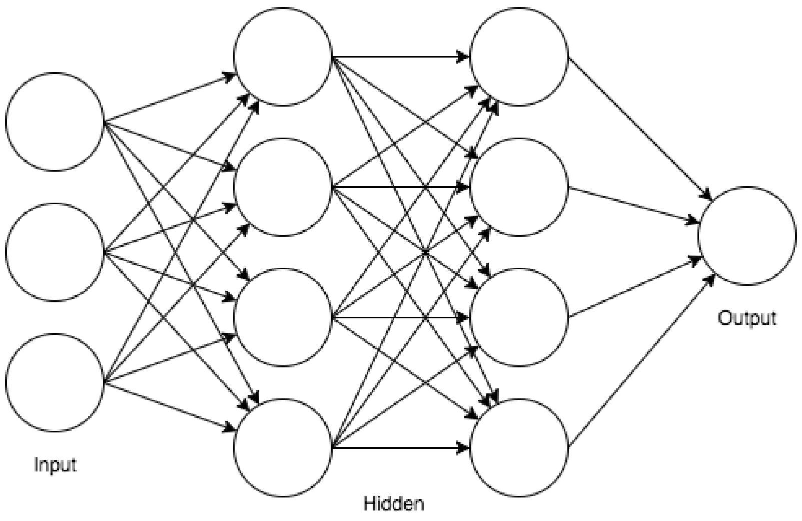


Figure 5.5: A sample of Neural Network

5.2 Feature generation

As earlier mentioned in this thesis we opted to use the edit distance for the comparison of the different description fields of the invoices, mainly to be able to find misspelled words. As basis for our algorithm we used distance measure proposed by Levenshtein [45] and conclusions made by Damerau in his paper "A technique for computer detection and correction of spelling errors" [46]. This algorithm is often referred to as the *Damerau-Levenhstein distance*, the algorithm gives the distance between strings based on edit operations. Levenshtein proposed insertion, deletion and substitution operations to transform one string to the other. Each of these operations has a cost assigned to performing the operation. Usually insertion and deletion have a cost of 1 and depending on the implementation substitution has 1 or 2. Damerau also introduced the transposition operation; where two adjacent characters may be swapped as a single operation whereas in the original algorithm two operations were needed; a deletion and an insertion. Further he states that approximately 80 percent of all misspellings can be corrected by using these simple operations[46].

For this project I developed a custom implementation of this distance measure and introduced a weighting schema that is based on the key distances of the *QWERTY*-keyboard. The greater the distance is between an expected character (that of the original string) the greater the error and the penalty for that character. Now apart from misspellings due to hitting the wrong key on the keyboard it may also happen that keys are pressed that look similar. Examples are the lowercase version *l* and the uppercase *I* or the letter *o* and the number *0*. People that aren't able to type blindly may confuse these characters with each other when typing, but we will not take this into account since, luckily, QWERTY has these characters mapped quite close to each other on the keyboard and thus our distance algorithm will be able to cope with these faults. We chose for using the keyboard as distance weighing since this is the most common mistake to be made for known words, other techniques for weighing may include phonetics but errors because of phonetics usually happen for unknown words or names, which is not really the case in our situation. We started off mapping the keyboard keys to a two dimensional matrix where every cell contains a character with an X and Y value. The scoring function for the distance between two keys is described in the following equation:

$$\textsc{Scoring} = 1 - \frac{1}{2\delta - \epsilon}$$

Here δ is the *Euclidean* distance between two keys on the keyboard, using the indices of the keys mapped in the matrix as dimensions. ϵ is the weighing of the distance between the keys. The higher the value the less strict the scoring will be on the distance between two characters. Obviously not all characters are mapped in the keyboard mapping, so in this case we return ∞ which will result in an equal scoring to every other operation of the Damerau-Levenshtein distance. Because we are only interested in strings with relatively small amount of misspellings we introduce also a stopping criteria that stops the calculation of the edit distance after the distance is already greater than a certain provided threshold.

Now when generating features for the different models we use our implementation of the Damerau-Levenshtein distance to weigh in misspelled words from in the unseen corpus; we do this using an optimized version of the algorithm below.

```
input  : N-grams $W$ extracted from corpus. And input word $I$ that is to be evaluated.
output: A feature vector for the corresponding input.
Variables
    $\epsilon$: The maximum distance
Function GenerateFeatureVector(I, W)
    $features := \mathsf{array}[|W|]$
    if $W \cap \{I\} \neq \emptyset$ then
        $idx :=$IndexLookup$(W, I)$
        $features[idx] = 1$
    else
        for $i := 1$ to $|W|$ do
            $w := W[i]$
            $d :=$DamLev$(I, w)$
            if $d \leq \epsilon$ then
                $features[i] = {}^{d}/_{\epsilon}$
            end
        end
    end
    return features
```
Algorithm 1: Damerau-Levenshtein misspelling correction for feature extraction.

The algorithm first checks if the new word is known by the trained features, if so it sets the feature at its corresponding index to 1 (the index is retrieved in the IndexLookup function). If the input n-gram is unknown to the bag of words it will use the DamLev (Damerau-Levenshtein) to check if the input n-gram qualifies as a misspelling with ϵ as maximum distance to consider. For the matches a value is set to the feature vector and in the end the entire vector is returned, ready to be compared with the other feature vectors.

5.2.1 Stemming

Stemming algorithms are developed for the purpose of stripping verb conjugations. These algorithms are used for natural language processing and will most likely not generate grammatically correct stems of the verbs, but will uniform the same verb that is conjugated differently to the same stem. Typical algorithm will strip these suffices from verbs without prior knowledge of the original verb. We will be using Porter algorithm to test if stemming the description texts will increase accuracy with classification [47]. More specifically the Dutch implementation of Kraaij and Pohlmann will be used [48], although the more sophisticated of Gaustad and Bouma[49] performed a more accurate stemming, the requirement of an dictionary for mapping makes it unattractive to consider for this thesis.

5.2.2 Compound words splitting

Earlier we mentioned about the already existing algorithms for breaking down compound words into their separate words. We want to use one of these methods to test if it may improve the accuracy for our models. After some search we did not find an algorithm which could easily be integrated in to the .NET application we are looking for. For this reason I have formulated my own compound word splitting algorithm (Algorithm 2), that returns possible split words based on an earlier generated dictionary (unigrams).

```
input  : A suspected compound word $x$ and a list of all words $W$ (candidates).
output: The words that $x$ was built up off, otherwise an empty set.
struct {
    Set result := $\emptyset$
    Float score := $-\infty$
    Integer numLeftChars := $\infty$
    Integer numCharPos := $\infty$
} CompoundCoverageResult;
Function SplitCompound(x, W)
    $y := \emptyset$
    if $|x| \geq \alpha$ then
        foreach $w$ in $W$ do
            if $x \cap w \geq \beta$ then
                $y = y \cup \{w\}$
            end
        end
        if $|y| \geq 2$ then
            $largestCoverage$ := LargestWordCoverage($x$, $y$, $\emptyset$, $x$)
            if $largestCoverage.numLeftChars \leq \lceil(|x| \cdot \gamma)\rceil$ then
                return $largestCoverage.result$
            end
        end
    end
    return $\emptyset$
Function LargestWordCoverage(compound, candidates, result, original)
    $best$ := CompoundCoverageResult()
    foreach $w$ in $candidates$ do
        $current := result \cup \{w\}$
        $remains$ := Split($compound$, $w$)
        $nextCandidates := \{c \in candidates | \exists r \in remains \ni c \cap r \neq \emptyset\}$
        if $|remains| \geq 1$ and $nextCandidates \neq \emptyset$ then
            foreach $w_{remaining}$ in $remains$ do
                $result$ := LargestWordCoverage($w_{remaining}$, $nextCandidates$,
                  $current$, $compound$)
                if $result.score > best.score$ then
                    $best = result$
                end
            end
        else
            return  struct {
                $result := current$
                $score := -numCharsleft - numCharPos - |current|$
                $numLeftChars := |original| - \sum_{e \in remains} |e|$
                $numCharPos := |remains|$
            } CompoundCoverageResult;
        end
        return $best$
    end
    return $x$
```
Algorithm 2: Compound word splitting algorithm

In the algorithm a string is represented as a bag of characters. The union $\cup$ of two strings can be seen as a concatenation of the two. The intersection $\cap$ of two strings can be translated to a strings Contains function. The pipes $|s|$ represents the length of string s, meaning the amount of characters that it contains. There is also a reference to a method named Split, this is a simple string split algorithm that given the first argument splits the second arguments in to zero or multiple strings. The algorithms makes use of the idea that the a compound word is built up out of two or more separate words. Through the candidate dictionary all words that are contained within the compound word are used. Then it does a recursive search for a set of words that;

1. Minimize the amount of separate words, while;
2. Minimize the amount of letters that are left after the compound construction, and;
3. Minimize the amount of places where the letters are left.

The first rule ensure that the word *onderhoudservice* prefers the combination of words $\{onderhoud, service\}$ over $\{onder, houd, service\}$. The second rule makes sure that $\{rest, je\}$ is an invalid construction of the word *restaurantje.* The third rule is an extension of the second which further penalises the construction of the set $\{class, ion, cat\}$ for the word *classifications*; it leaves two parts *ifi* and *s* which is 'worse'.

5.2.3 Custom helpers

All texts that is processed first is normalized by converting all characters to lowercase and stripping the diacritics from the characters.

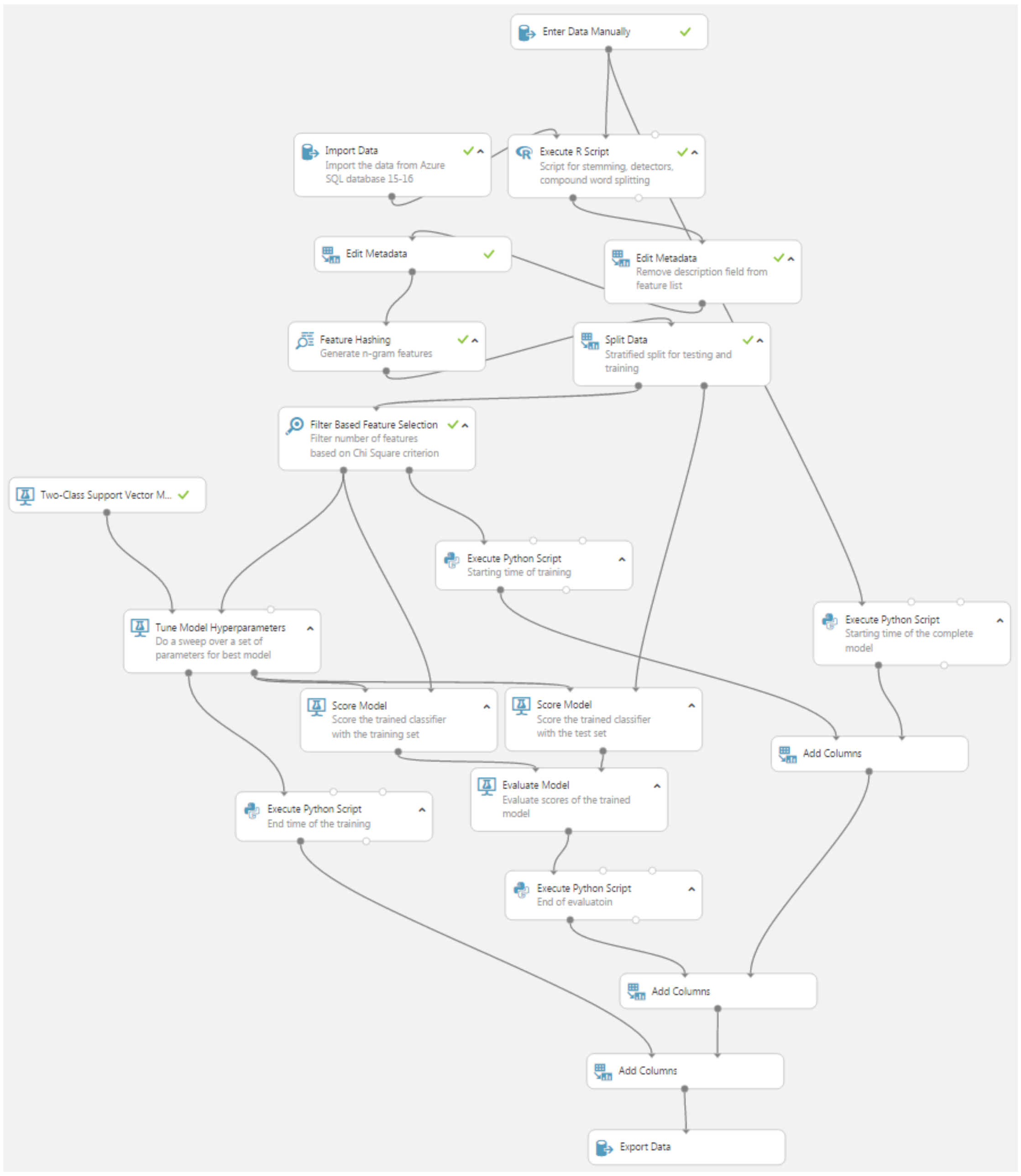
When looking closely at the data there seem to be some noticeable patterns in some classes. As an example there are the tyre swaps which we want to identify in the invoices. Often the labour descriptions indicate assemblage or aligning of the wheels, but the parts itself are often also added to the invoice. Now tyres have a typical type number; *255/45-R18, Michelin Premacy 4, Conti. 17560R12*, since a lot of permutations are possible in these type numbers it is hard to register all possibilities in a simple *n*-gram model. Because of this we introduced so-called *token detectors* which look through the invoice descriptions before they are sliced and work on the basis of *regular expressions* if it matches 1 is counted toward the feature and 0 otherwise.

A similar technique is used for matching dates. As explained earlier invoice lines may refer to future maintenance visits (e.g. *20/5/2016, 7-2, febr. 2017, next dec.*). It looks for all the months names in the given language (including abbreviations) or tries to create a .NET DateTime object using the DateTime.Parse function which effectively can translate strings to valid dates. When matched it is counted towards the *date* feature if the valid date is not too far in the past or future.

5.2.4 Machine learning with Azure

Recently Microsoft introduced the Azure Machine Learning platform for easily creating machine learning projects with a user friendly graphical interface. Azure Machine learning Studio offers a great set of data manipulation modules as well as models and parameter sweeps. In Figure 5.6 we see an example of a model that we designed using the graphical interface. With different building blocks you can build a flow from importing the data, to the manipulation of data and generation of models.

The algorithms described in the previous sections, we first implemented in C#. Unfortunately Azure does not support running .NET modules in their Azure Machine Learning portal, for this reason all the custom implementations given are ported to R-script (see Section 8). If Azure



Enter Data Manually
Import Data
Import the data from Azure SQL database 15-16
Execute R Script
Script for stemming, detectors, compound word splitting
Edit Metadata
Edit Metadata
Remove description field from feature list
Feature Hashing
Generate n-gram features
Split Data
Stratified split for testing and training
Filter Based Feature Selection
Filter number of features based on Chi Square criterion
Two-Class Support Vector M...
Execute Python Script
Starting time of training
Execute Python Script
Starting time of the complete model
Tune Model Hyperparameters
Do a sweep over a set of parameters for best model
Score Model
Score the trained classifier with the training set
Score Model
Score the trained classifier with the test set
Add Columns
Evaluate Model
Evaluate scores of the trained model
Execute Python Script
End time of the training
Execute Python Script
End of evaluatoin
Add Columns
Add Columns
Export Data

Figure 5.6: An example of an experiment created in Azure Machine Learning Studio

provided a module for a certain script these were preferred over a custom R implementation. In Figure 5.6 an Azure experiment is shown that was used during this thesis.

For the generation of features from the description fields we used the *Feature hashing*-module on Azure. This module has implemented the Vowpal-Wabbit hashing algorithm[50]. Vowpal-Wabbit starts off with generating n-grams for all $1 \ldots n$, to speed up comparison of these they apply an hash on the shingles. Azure Machine Learning offers a variety of models, multiclass as well as two-class classifiers. The ones that we will be using for this thesis are the *Decision Forest, Bayes point machine[51], Support Vector Machine, Neural Network*. For each of these modules both a binary as multiclass (if available) will be trained.

On Azure we decided to use the *feature filtering* module which only keeps the top 500 features based on its χ^2 score. Not only does this speed up the model creation significantly, it also does not seem to generate worse models. Further Azure uses a model where one gets charged for the evaluation time and the amount of computation power used, cutting down the evaluation time can save lots of money in the end for RDC.

5.3 Parameter tweaking

Earlier in this chapter we discussed the different models that we use for our classification problem. In this section we elaborate on how the parameters, that are used during the evaluation phase, are generated. This section is divided into two; the parameter tweaking for the Azure Machine Learning experiments and the tweaking of the custom C# models.

5.3.1 Custom C# parameters

We began with our tree based decision models; the decision forests. We trained these using the C4.5 algorithm. During evaluation against large data sets we came across some memory issues for the generation of the random forest model. The Accord.NET machine learning library does not provide a memory efficient algorithm for the construction of decision trees. At each candidate split it stores a copy of the subset of all available splits, which results in huge arrays with largely duplicate data. I did try to amend the original algorithm that it would work with sparse matrices (matrices where only the non-zero values are stored), while it did save a lot of memory it was still not sufficient to train multiple trees storing a lot of redundant data across them. Because of this we were limited to just 2 decision trees for the smaller data sets and just a single for larger sets. Limiting tree depth didn't make a lot of difference, setting it to 12 seemed to make decent trees that classified pretty well. No pruning algorithms were concerned.

As for our Neural Network, we used a network with just a single hidden layer, containing 10 neurons. Changing the number of neurons didn't provide any real difference in accuracy to our model. Introducing an extra layer did gave us some improvements for small sets, but this caused the evaluation time to go up rapidly for small data sets and made it infeasible to generate a model for larger sets. We trained it with 500 iterations using backward propagation with a fixed learning rate of 0.1. We tried different number of iterations, both more and less. But the result was with less iterations that the F_1 score would drop and with more iterations we saw that the error rate declined quickly which was an indicating that the model was probably overfitting. We tried different momenta to speed up the search procedure, resulting in a momentum of $\frac{1}{2}$. We were careful in setting the momentum not too high and risk overshooting the minimum.

The Support Vector Machine models are trained using a Sequential Minimal Optimisation algorithm. We started off with a tolerance of 0.1, but this gave room for too much error, especially for the *serpentine belt* class. With a tolerance of 0.01 the algorithm performed well and going lower would raise exceptions because of the algorithm could not converge. For the complexity parameter C we made use of the built-in complexity heuristic that exists in the Accord.NET framework. For the Support Vector Machine only a linear kernel was used.

Because the feature vectors exceeded the memory bounds for the larger data sets, we tried to implement a Discriminative Analysis with a χ^2 kernel for feature filtering. But unfortunately the algorithm took too long to incorporate in our evaluation. As fallback we used the Pearson correlation coefficient to find the best features. With this we managed to reduced the number of features from over 10,000 to just 500 features per sample. The features that were used were the ones that had the greatest absolute Pearson correlation value.

5.3.2 Azure ML parameters

Azure Machine Learning has a module called '*Tune Model Hyperparameters*' this module performs a parameter sweep on a model to determine the optimum parameter settings. For the Azure Machine Learning models this module is used to find the best performing parameters. We chose a *Random grid sweep* to find the best parameters. While an entire grid sweep is more thorough it requires much more time. Also, Bergstra and Bengio claim in their paper that random search of parameter space is by far more effective than grid search [52]. We started off using 10 random sweeps, but this took quite a lot time. After trying a bit around we discovered that 5 sweeps resulted in the same score for the best scoring models. As performance measurement we chose that the F_1 score should be optimized for this model.

For the Support Vector Machines we used the *Support Vector Machine* module. Microsoft nowhere specifically specifies what kernel is used in this model. But since they also offer a non-linear *Locally Deep Support Vector Machine* we are convinced that this module uses a linear kernel. For the regularisation parameter λ we had the value range $[10^{-1}, 10^{-5}]$. And we set the amount of iterations between $[10, 150]$. We used no normalisations. In Table 5.1 we see the resulting scores of such parameter sweep, Azure returns the best performing parameters which can be stored for later use.

Iterations	**Lambda**	**Accuracy**	**Precision**	**Recall**	**F-Score**	**AUC**
98	0.003046	0.996625	0.995837	0.997134	0.996485	0.999332
68	0.031466	0.989125	0.995247	0.982022	0.98859	0.999458
29	0.046737	0.988375	0.995239	0.980459	0.987794	0.999554
63	0.046956	0.988375	0.995239	0.980459	0.987794	0.999549
86	0.099535	0.988125	0.994452	0.980719	0.987538	0.999395

Table 5.1: Example of parameter sweep for Support Vector Machine

Our Neural Network was a fully connected Neural Network where every node has an edge to the other nodes in other layers. The amount of layers was not adjustable by settings, but only by providing a custom Net#[53] neural network specification. Since we were not familiar with this type of neural network definition we chose to stick with the default number of hidden layers, which was just a single layer. For the tuning of the neural network we set the number of hidden nodes to range $[50, 300]$. We made use of a momentum which could speed up our search process for a minimum, we set this value to 10^{-1}. Further we set the learning rate of the model to the range of $[10^{-3}, 10^{-1}]$. Again we set the number of iterations between $[10, 150]$.

The decision forest parameters were set to; number of trees between $[8, 150]$. With maximum depth of the tree set between $[10, 64]$. The number of random splits was between $[64, 512]$. Further we fixed the number of minimal number of samples per leaf node to 15, this means that there can be a leaf if there are at least 15 samples reaching that leaf node to avoid overfitting.

Now for every experiment we do we use the *Tune Model Hyperparameters* module to find the best parameters for every case and data set. The parameter ranges, however, are kept the same across all the experiments.

Chapter 6

Evaluation

6.1 Experiment setup

The experiments will be run as on the models and configurations described in the previous section. The custom C# implementation experiments were run on a HP ProDesk 600 G1 SFF with a Intel Core i5 running at 3.2 GHz with 16 gigabytes of DDR3 memory. The system runs on a 64-bits version of Windows 7 and has a 500GBs - 7200 RPM disk drive.

For the custom implementation we used a timer to record the training and evaluation time. Our experiment will begin by reading setting up an connection from the database. First the training data is read in to generate our feature vectors. Mind that the timer starts after the connection is set up and the n-grams are constructed. The data is than streamed from our Microsoft SQL database to our training algorithm. After a model has been trained the timer is reset and our system will feed the evaluation set to the model. The outcomes are tallied and registered in a confusion matrix, from which scores are calculated (e.g. accuracy, precision, recall, F_1) and exported to disk.

The Azure Machine Learning experiments will be run on the online Azure platform. Microsoft does not specify the hardware that is used but they built it such that is automatically scales to the needs of the experiment. In Figure 5.6 we see the complete set up of our experiments. We start with the import of the data from a Azure database, the evaluation and training data is split. After which feature generating is done using R-script and feature hashing. The start and end dates of training and evaluation are recorded by Python scripts. Then training with parameter tuning using a random grid sweep, and evaluation with the best trained model. The results of evaluation are then automatically exported to an Azure database.

6.1.1 Training and evaluation data sets

Now that we have set a baseline for data normalisation we will define the different data sets that are used during training and evaluation. We choose different sizes of sets to compare the performance on the various sizes. A total of three differently sized sets will be used, their sizes;

Small Total set size of just $5 \cdot 10^3$ records.

Medium Total set size of 10^4 records.

Large Set size of 10^5 records.

The total size is around $2.5 \cdot 10^5$ *lines.*

With every test we will use a split percentage of 80/20 meaning that 80% of the data will be used for training and 20% for evaluation. Examples are carefully taken and made sure that they are sufficient examples of every class such that the trained model will perform at its best. Further the data split between training and evaluation will be stratified such that the distributions between the training and the evaluation are roughly the same.

6.2 Experimental results

The complete results are added as tables to the appendix. These are the tables Table 8.6 through Table 8.20. If we look at the evaluation results we can see that the scores are quite similar across the different models. The training times are however quite divers. The neural networks are by far the most complex classifiers to train, the small data sets take around 2 minutes to train, whereas the large may take up to four hours. The results are quite remarkable with F-scores around 0.99 through the different data sets. Unfortunately our Support Vector Machine caused memory issues which could not be resolved by our dimensionality reduction. For this reason the results of the large data sets are missing from the results table in the appendix.

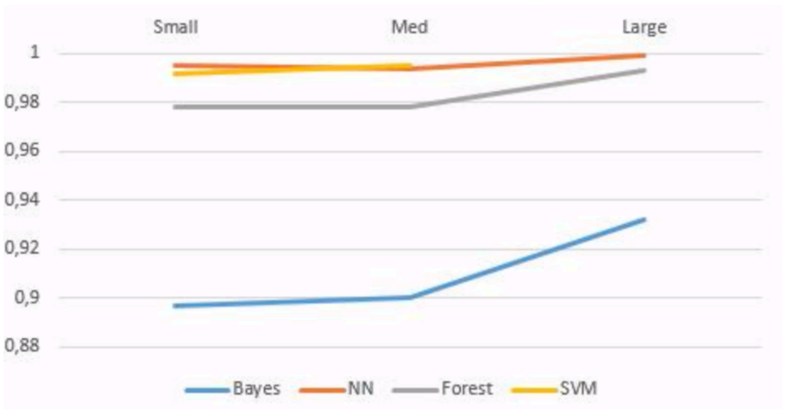


Figure 6.1: Results C# - None options

We see that the classifiers overall performed very well, further can be seen that for every evaluation results improve with the size of the data set. Of course this is expected since more training data means more data to generate a classifier with. By looking closely at the evaluation and training time some strange deviations can be seen for the custom implemented models. We think this might be the case because of background processes running on our Windows machine that might consume memory and cause our program to switch to (magnitudes slower) swap memory on our disk. Overall the serpentine and V-belts were the classes that were the hardest to classify for our models. But in the end even these classes had good scores. In Figure 6.1 and Figure 6.2 we see the F_1 scores of the different implementations. We observe that the scores go up with the size of the data sets increasing, which is exactly what we would expect. Only the Neural Network for the Azure model show a small drop when comparing the small and the medium set. But also a steep increase for the large set.

Since the results of the Neural Network and Support Vector Machine showed the best performance, and other models showed similar results we decided to evaluate only these model for the multi-class results, just to get an idea how a multi-class classifier would perform. Also, we used

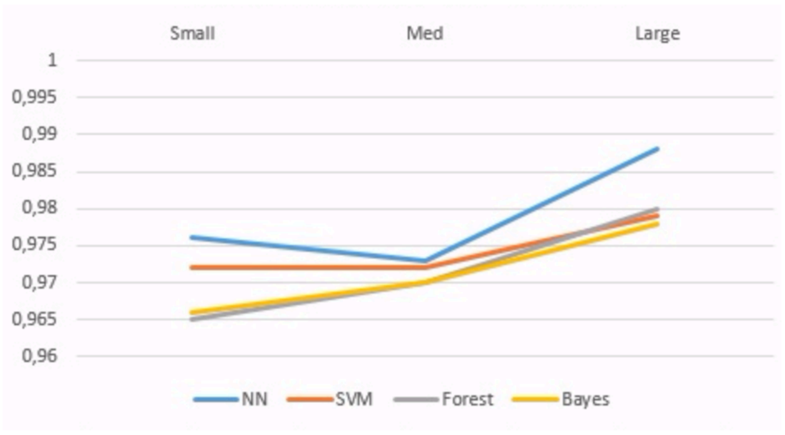


Figure 6.2: Results Azure - None options

none of the evaluation options because these do not show significant improvement in classification accuracy. For the training and evaluation set of this classifier we used completely random sampling so that distributions will be kept untouched.

Also the results of these algorithms show high classification accuracy, this single classifier unfortunately is not as good as the sum of the separate classifiers. But we still see *overall agreements* of well above 95 percent. In Table 8.20 the results can be seen in detail. We were quite surprised by the accuracy these models showed we more misclassifications, for some classes a very low amount ended up in the training set but still the models were able to distinguish these classes pretty well. What was quite interesting was the fact that overall, the Naive Bayes performed as one of the worst models in comparison with the rest. But for the multi-class example, we save the opposite. Here the Naive Bayes models comes out on top with an overall agreement of 0.977.

6.2.1 Evaluation versus the current system

It was quite a challenge to find the output of the current system with the corresponding input. In the end we manage to find a small data set of thousand receipts with the result that the Perl implementation gave. After a manual classification of the data into maintenance and no maintenance we calculated the errors of the current system. Then we ran our earlier generated 1 vs all model (without options) with the this data. In the tables below the confusion matrices are shown of the classification of maintenance;

	T	**F**
T	156	10
F	4	830

	T	**F**
T	127	33
F	11	829

Table 6.1: Confusion matrices of our system versus the current system. Left our Neural network, right the current implementation

We see that our system made 14 errors whereas the current system made 44 total errors. If we look at the errors which our neural network made the most errors were made for the description: *Ford Motorcraft Service*. This is a new type of maintenance, which slipped through our manual labeling during the data preparation phase. Other errors were made on *tussen beurt*, which is intermediate maintenance (which strictly does not classify as maintenance).

All in all we see an improvement of the F_1 more than ten percent from 0.852 to 0.957. With new accuracy of 0.986, precision 0.975 and recall of 0.939, which are clear improvements over the old system. Further we still see room for improvement. We have identified missing samples which can be added to the training set for next cycle for possibly even better results.

Chapter 7

Conclusion

The trained classifiers showed great results for the classification of almost all examples, while the performance varies per classifier and class label the overall score was better than hoped. In this end this shows that the data is very well suited to be trained as a classification problem. With the promising results RDC decided to immediately integrate the classifiers in their CaRe-Mail product. We generated a lot of different models for classification with very different options (e.g. typo correction, expression detectors) although some difference is seen when using these detectors no huge gain can be seen in evaluation results. In the end the custom C# implementation performed a bit better than the solution that was created with the Azure Machine Learning platform this is mainly due to the fact that the custom implementation made it possible to change the algorithms to work perfectly with each other, whereas the Azure solution only has *black-boxed* models where just a few parameters could be set. With regard to the training and evaluation time it seems heavily in favour of Azure Machine Learning, where almost every custom implementation took longer. Unfortunately we can't measure the real difference since we can't control the hardware setup for the Azure Machine Learning solution; it automatically scales to its needs. And we ran our custom implementation to the same fixed setup with limited memory and processing power.

Our algorithms show that for the correct classification of the receipts, the description fields, together with the codes, costs and (sometimes) dealer id are the most important for classification. No intensive tweaking was needed to find the best set of features to train on. Further we show that a set of binary classifiers perform better than one multiclass classifier. Another argument for using multiple binary classifiers is that adding new class labels does not affect the existing classifiers. Instead only new classifiers have to be generated and can be used alongside the others.

In the end RDC decided to go for the Azure Machine Learning platform because of multiple advantages that the platform offers over the custom C# application. The most obvious reason is that is requires less knowledge of machine learning techniques to work with it. Furthermore it also is hosted within Azure, where the rest of the solutions also live.

7.1 Discussion and future work

RDC is very satisfied with what the results show, and is incorporating the solutions into their product. Further, this project peeked the interest of RDC in machine learning solutions and they are actively in search of opportunities where similar techniques can be used for their products.

They are currently looking into pattern set mining techniques to be used on their car owner and purchases data set.

While the Dealer Management Systems give the mechanics the complete freedom when creating invoices, the data in these fields were quite singular (Dealer Management Systems carry a lot of receipt lines with the same data). Though they did differ with DMS and dealer a lot of the same descriptions and codes were used in these invoices. This may be the reason for the high classification accuracy we saw in the evaluation section.

As we have seen we can conclude that the Dealer Management System data can be classified with great results. Still there is some work to be done. For example; RDC is really interested in creating models which are able to learn from misclassifications while running in production. This would enable the model to adapt to faults and improve using feedback from their customers without manually retraining the model.

During labeling it became quickly clear that the classes V-belt and serpentine belt are almost the same. In fact the data we received from some dealers used the terms interchangeably. In future research we might want to merge these labels for better results and, may be, also less confusion when labeling the data.

An other research area is that of multi-label receipts. For the thesis project we decided to only work with single class receipts for training and evaluation such that in production the 1 vs all classifiers would be able to classify multi-label this way. There exist however multi-label classifiers that might be interesting to look at for future research.

7.2 Acknowledgements

In this section I would like to thank a few people for their support and help during my master thesis. First off I would like to personally thank my primary supervisor dr. A.J. Feelders for his swift and thorough feedback throughout this thesis. Second, my second examiner prof. dr. A.P.J.M Siebes. And RDC, which gave me the opportunity for doing my thesis project on-site and providing me with the necessities for completing this thesis. Further I would like to thank my principal supervisor Joris Luder who gave me the opportunity to help his team in creating a better experience for the CaRe-Mail customers. And lastly the whole CaRe-Mail team that helped me have a great time at RDC; thank you all.

Bibliography

[1] Pete Chapman et al. “CRISP-DM 1.0 Step-by-step data mining guide”. In: (2000).

[2] Gregory Piatetsky. “What main methodology are you using for your analytics, data mining, or data science projects? Poll”. In: (2014). URL: `http://www.kdnuggets.com/polls/2014/analytics-data-mining-data-science-methodology.html`.

[3] Colin Shearer. “The CRISP-DM model: the new blueprint for data mining”. In: *Journal of data warehousing* 5.4 (2000), pp. 13–22.

[4] Evelyn Fix and Joseph L Hodges Jr. *Discriminatory analysis-nonparametric discrimination: consistency properties.* Tech. rep. DTIC Document, 1951.

[5] Thomas M Cover and Peter E Hart. “Nearest neighbor pattern classification”. In: *Information Theory, IEEE Transactions on* 13.1 (1967), pp. 21–27.

[6] John F Magee. *Decision trees for decision making.* Harvard Business Review, 1964.

[7] Sreerama K Murthy. “Automatic construction of decision trees from data: A multi-disciplinary survey”. In: *Data mining and knowledge discovery* 2.4 (1998), pp. 345–389.

[8] David Bryant and Vincent Berry. “A structured family of clustering and tree construction methods”. In: *Advances in Applied Mathematics* 27.4 (2001), pp. 705–732.

[9] Janine Toole. “Categorizing unknown words: Using decision trees to identify names and misspellings”. In: *Proceedings of the sixth conference on Applied natural language processing.* Association for Computational Linguistics. 2000, pp. 173–179.

[10] Tin Kam Ho. “Random decision forests”. In: *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on.* Vol. 1. IEEE. 1995, pp. 278–282.

[11] Laurent Candillier, Isabelle Tellier, and Fabien Torre. *Transforming XML trees for efficient classification and clustering.* Springer, 2006.

[12] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. “A training algorithm for optimal margin classifiers”. In: *Proceedings of the fifth annual workshop on Computational learning theory.* ACM. 1992, pp. 144–152.

[13] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.

[14] Libby Barak, Ido Dagan, and Eyal Shnarch. “Text categorization from category name via lexical reference”. In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers.* Association for Computational Linguistics. 2009, pp. 33–36.

[15] Mr. Bayes and Mr Price. “An Essay towards solving a Problem in the Doctrine of Chances. By the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFRS”. In: *Philosophical Transactions (1683-1775)* (1763), pp. 370–418.

[16] Judea Pearl. *Bayesian networks: A model of self-activated memory for evidential reasoning.* University of California (Los Angeles). Computer Science Department, 1985.

[17] Judea Pearl. “Fusion, propagation, and structuring in belief networks”. In: *Artificial intelligence* 29.3 (1986), pp. 241–288.

[18] Nir Friedman, Dan Geiger, and Moises Goldszmidt. “Bayesian network classifiers”. In: *Machine learning* 29.2-3 (1997), pp. 131–163.

[19] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386.

[20] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural networks* 4.2 (1991), pp. 251–257.

[21] Stephen Grossberg. “Nonlinear neural networks: Principles, mechanisms, and architectures”. In: *Neural networks* 1.1 (1988), pp. 17–61.

[22] Sushmito Ghosh and Douglas L Reilly. “Credit card fraud detection with a neural-network”. In: *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on.* Vol. 3. IEEE. 1994, pp. 621–630.

[23] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. “Extreme learning machine: theory and applications”. In: *Neurocomputing* 70.1 (2006), pp. 489–501.

[24] Xiang-guo Zhao et al. “XML document classification based on ELM”. In: *Neurocomputing* 74.16 (2011), pp. 2444–2451.

[25] Huan Liu and Hiroshi Motoda. *Feature selection for knowledge discovery and data mining.* Vol. 454. Springer Science & Business Media, 2012.

[26] Moshe Ben-Bassat. “35 Use of distance measures, information measures and error bounds in feature evaluation”. In: *Handbook of statistics* 2 (1982), pp. 773–791.

[27] Zhexue Huang. “Extensions to the k-means algorithm for clustering large data sets with categorical values”. In: *Data mining and knowledge discovery* 2.3 (1998), pp. 283–304.

[28] Francisco de AT de Carvalho and Renata MCR de Souza. “Unsupervised pattern recognition models for mixed feature-type symbolic data”. In: *Pattern Recognition Letters* 31.5 (2010), pp. 430–443.

[29] Fabrizio Sebastiani. “Machine learning in automated text categorization”. In: *ACM computing surveys (CSUR)* 34.1 (2002), pp. 1–47.

[30] C.E. Shannon. “A mathematical theory of communication”. In: *Bell System Technical Journal* 27.4 (1948), pp. 623–656.

[31] Kenneth Church and William Gale. “Inverse document frequency (idf): A measure of deviations from poisson”. In: *Natural language processing using very large corpora.* Springer, 1999, pp. 283–295.

[32] Alec Go, Richa Bhayani, and Lei Huang. “Twitter sentiment classification using distant supervision”. In: *CS224N Project Report, Stanford* 1 (2009), p. 12.

[33] Victoria Bobicev and Marina Sokolova. “An Effective and Robust Method for Short Text Classification.” In: *AAAI.* 2008, pp. 1444–1445.

[34] Deepshikha Patel and Monika Bhatnagar. "Mobile SMS Classification". In: *International Journal of Soft Computing and Engineering (IJSCE) ISSN* (2011), pp. 2231–2307.

[35] Aixin Sun. "Short text classification using very few words". In: *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval.* ACM. 2012, pp. 1145–1146.

[36] Longzhen Duan, Nan Li, and Longjun Huang. "A new spam short message classification". In: *2009 First International Workshop on Education Technology and Computer Science.* IEEE. 2009, pp. 168–171.

[37] Mauricio A Hernndez and Salvatore J Stolfo. "Real-world data is dirty: Data cleansing and the merge/purge problem". In: *Data mining and knowledge discovery* 2.1 (1998), pp. 9–37.

[38] Heiko Mller and Johann-Christph Freytag. *Problems, methods, and challenges in comprehensive data cleansing.* Professoren des Inst. Fr Informatik, 2005.

[39] Jeffrey G Brown. "Using a multiple imputation technique to merge data sets". In: *Applied Economics Letters* 9.5 (2002), pp. 311–314.

[40] Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. "A comparative analysis of methodologies for database schema integration". In: *ACM computing surveys (CSUR)* 18.4 (1986), pp. 323–364.

[41] Christiaan Thieme and Arno Siebes. "Schema integration in object-oriented databases". In: *Advanced Information Systems Engineering.* Springer. 1993, pp. 54–70.

[42] Esko Ukkonen. "Algorithms for approximate string matching". In: *Information and control* 64.1 (1985), pp. 100–118.

[43] Genootschap Onze Taal. *Samenstellingen.* Oct. 2015. URL: `https://onzetaal.nl/taaladvies/advies/samenstelling` (visited on 03/15/2016).

[44] Carla Parra Escartn. "Chasing the Perfect Splitter: A Comparison of Different Compound Splitting Tools". In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14).* Ed. by Nicoletta Calzolari (Conference Chair) et al. Reykjavik, Iceland: European Language Resources Association (ELRA), May 2014. ISBN: 978-2-9517408-8-4.

[45] V. I. Levenshtein. "Binary codes capable of correcting deletions, insertions and reversals". In: *Soviet Physics Doklady* 10 (1966), pp. 707–710. URL: `http://ci.nii.ac.jp/naid/10020212767/en/`.

[46] Fred J Damerau. "A technique for computer detection and correction of spelling errors". In: *Communications of the ACM* 7.3 (1964), pp. 171–176.

[47] Martin F Porter. "An algorithm for suffix stripping". In: *Program* 14.3 (1980), pp. 130–137.

[48] Wessel Kraaij and Rene Pohlmann. "Porters stemming algorithm for Dutch". In: *Informatiewetenschap* (1994), pp. 167–180.

[49] Tanja Gaustad and Gosse Bouma. "Accurate stemming of Dutch for text classification". In: *Language and Computers* 45.1 (2002), pp. 104–117.

[50] Lihong Li and Alex Strehl. *Vowpal Wabbit.* Dec. 2007. URL: `http://hunch.net/?p=309` (visited on 05/04/2016).

[51] Ralf Herbrich, Thore Graepel, and Colin Campbell. "Bayes point machines". In: *Journal of Machine Learning Research* 1.Aug (2001), pp. 245–279.

[52] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.

[53] Jeannine Takaki. *Guide to Net# neural network specification language for Azure Machine Learning*. May 2016. URL: `https://azure.microsoft.com/en-us/documentation/articles/machine-learning-azure-ml-netsharp-reference-guide/` (visited on 08/11/2016).

Chapter 8

Appendices

XML data from Dealer Management Systems

```xml
<?xml version="1.0" ?>
<section>
 <dmsdata dms_id="0" sleutel="1345">
  <auto_nr>1345</auto_nr>
  <factuur_bedrag>214.29</factuur_bedrag>
  <km_stand>151976</km_stand>
  <bezoek_datum>2015-12-01</bezoek_datum>
  <vestiging_code>12</vestiging_code>
  <menu_code>S00</menu_code>
  <factuur_nummer>17212289</factuur_nummer>
  <details>Service-beurt</details>
  <tarmagic>75169</tarmagic>
 </dmsdata>
 <dmsdata dms_id="0" sleutel="10878">
  <auto_nr>10878</auto_nr>
  <factuur_bedrag>108.44</factuur_bedrag>
  <km_stand>103502</km_stand>
  <bezoek_datum>2015-12-11</bezoek_datum>
  <vestiging_code>11</vestiging_code>
  <menu_code>BAN</menu_code>
  <factuur_nummer>1679407</factuur_nummer>
  <details>Bandenmontage</details>
  <tarmagic>20661</tarmagic>
 </dmsdata>
 <dmsdata dms_id="0" sleutel="125064">
  <auto_nr>125064</auto_nr>
  <factuur_bedrag>26.92</factuur_bedrag>
  <km_stand>72760</km_stand>
  <bezoek_datum>2015-12-02</bezoek_datum>
  <vestiging_code>12</vestiging_code>
  <menu_code>LIC</menu_code>
  <factuur_nummer>21940542</factuur_nummer>
  <details>Verlichting</details>
  <tarmagic>0</tarmagic>
 </dmsdata>
</section>
```

Figure 8.1: Autoline Dealer Management System data sample

```xml
<?xml version="1.0" ?>
<section name="WPL">
  <dmsdata dms_id="0" sleutel="8192">
   <faktuurnummer>8192</faktuurnummer>
   <kenteken>16-LSE-2</kenteken>
   <klantnr>8427</klantnr>
   <omzetsoort>1 WEX  Wpl Ext</omzetsoort>
   <p_totaal>39.95</p_totaal>
   <p_btw>6.93</p_btw>
   <kmstand>187644</kmstand>
   <werkplaatsdatum>2015-01-19 00:00:00.0</werkplaatsdatum>
   <door>Dirk Jansen</door>
   <vestiging>V007</vestiging>
   <omschrijving>Diversen</omschrijving>
   <pakketten>APK,Vervangen gloeilamp koplamp H7 ,</pakketten>
   <artikelen>APK Afmelding,GLOEILAMP,</artikelen>
   <werk>APK keuring,Vervangen gloeilamp koplamp H7,</werk>
   <codes>APK001  ,000065  ,</codes>
  </dmsdata>
  <dmsdata dms_id="0" sleutel="8492">
   <faktuurnummer>8492</faktuurnummer>
   <kenteken>17-NPR-5</kenteken>
   <klantnr>9383</klantnr>
   <omzetsoort>1 WEX  Wpl Ext</omzetsoort>
   <p_totaal>-229.0</p_totaal>
   <p_btw>-39.74</p_btw>
   <kmstand>51200</kmstand>
   <werkplaatsdatum>2015-12-01 00:00:00.0</werkplaatsdatum>
   <door>Dirk Jansen</door>
   <vestiging>V007</vestiging>
   <omschrijving>CREDITNOTA van 32001312</omschrijving>
   <pakketten>05 Jaar/100.000Km Ford Onderhoud,</pakketten>
   <artikelen>GEURFILTER,OLIEFILTER,Motorolie 5W-30,</artikelen>
   <werk>Beurt na 5 jaar/100.000 km,</werk>
   <codes>4500,</codes>
  </dmsdata>
  <dmsdata dms_id="0" sleutel="65981">
   <faktuurnummer>65981</faktuurnummer>
   <kenteken>84-POV-3</kenteken>
   <klantnr>2276</klantnr>
   <omzetsoort>1 WEX  Wpl Ext</omzetsoort>
   <p_totaal>39.0</p_totaal>
   <p_btw>6.77</p_btw>
   <kmstand>20000</kmstand>
   <werkplaatsdatum>2015-12-28 00:00:00.0</werkplaatsdatum>
   <door>Justin Myjer</door>
   <vestiging>V001</vestiging>
   <omschrijving>Diversen</omschrijving>
   <artikelen>Instructie boek Fiesta,</artikelen>
  </dmsdata>
</section>
```

Figure 8.2: WinCar Dealer Management System data sample

```
<?xml version="1.0" ?>
<dmsdata dms_id="0" sleutel="322841">
   <dms_id>0</dms_id>
   <dms_db>N_R</dms_db>
   <auto_nr>322841</auto_nr>
   <factuur_bedrag>33.02</factuur_bedrag>
   <km_stand>48825</km_stand>
   <bezoek_datum>2015-02-04</bezoek_datum>
   <nummerplaat>PS-SV-06</nummerplaat>
   <qwaarde>RO</qwaarde>
   <factuurnummer>1175334187</factuurnummer>
   <factuurdatum>2015-02-09</factuurdatum>
   <locatie>17</locatie>
   <omschrijving>&apos;OCT 1) WINTERBANDENWISSEL&apos;,
   &apos;CREDITNOTA OP FACTUURNUMMER 1175303817&apos;,
   &apos;WINTERWISSEL &apos;,&apos;KLANT HEEFT ZELF DE
   BANDEN&apos;,&apos;WINTER-/ZOMERBANDENSET WISSELEN &apos;</omschrijving>
   <bruto>&apos;@PLANNING&apos;,&apos;undef&apos;,&apos;undef&apos;
   ,&apos;undef&apos;,&apos;WBV &apos;</bruto>
   <aantal>0,0,0,0,-1</aantal>
   <soortlijn>Y16,AUT,   ,Y16,Y14</soortlijn>
   <lijngeencorrectie>undef,undef,1,undef,undef</lijngeencorrectie>
   <interventiecode>undef,undef,B,undef,undef</interventiecode>
   <soortuurcodelijn>undef,undef,undef,undef,undef</soortuurcodelijn>
   <bedrag_intern>0</bedrag_intern>
   <merk>RENAULT</merk>
   <klantnummer>194831</klantnummer>
  </dmsdata>
  <dmsdata dms_id="0" sleutel="156517">
   <dms_id>0</dms_id>
   <dms_db>N_R</dms_db>
   <auto_nr>156517</auto_nr>
   <factuur_bedrag>23.75</factuur_bedrag>
   <km_stand>77131</km_stand>
   <bezoek_datum>2015-12-17</bezoek_datum>
   <nummerplaat>03-PW-LE</nummerplaat>
   <qwaarde>YA</qwaarde>
   <factuurnummer>5132306700</factuurnummer>
   <intern>4</intern>
   <factuurdatum>2015-12-17</factuurdatum>
   <locatie>3</locatie>
   <omschrijving>&apos;1. V-SNAREN NAZIEN IVM REGELMATIG PIEPEN.&apos;,
   &apos;V-SNAREN&apos;,&apos;9999 V-SNAREN GESPANNEN&apos;</omschrijving>
   <bruto>&apos;@PLANNING&apos;,&apos;undef&apos;,&apos;undef&apos;</bruto>
   <aantal>0,0,.25</aantal>
   <soortlijn>Y16,   ,Y20</soortlijn>
   <lijngeencorrectie>undef,1,undef</lijngeencorrectie>
   <interventiecode>undef,A,undef</interventiecode>
   <soortuurcodelijn>undef,undef,1</soortuurcodelijn>
   <bedrag_intern>0</bedrag_intern>
   <merk>HYUNDAI</merk>
   <klantnummer>71751</klantnummer>
  </dmsdata>
```

Figure 8.3: EVA DMS Dealer Management System data sample

Samples of Dealer Management System data

car_id	sec_inv_amnt_exc	woh_mileage	woh_ord_date	factuurdatum	release_date	userid	whs_id	onderhoud	rob
47094	109,02	183213	2457465	2457465	2457465	305	11	0.0	0.0
47121	105,88	68118	2457328	2457328	2457328	310	2	1.0	0.0
47251	370,78	136527	2257366	2257866	2257866	490	9	0.0	0.0
47399	71,79	120730	2455242	2465242	2465242	38	9	0.0	0.0
46441	84,54	77360	2032780	2033780	2033780	95	5	0.0	1.0

Table 8.1: Sample of CarIT Dealer Management System

vin	bedrag_inc	km_stand	header	description	locatie	soort	order	factuur	datum	intern
W0L0SDL6874307456	16,16	161017	apk	<Description>APK all-in myOpel leden</Description> <Description>Philips autolamp 5w steek</Description>	1	1	01-130529	01-130529	20160105	0
DA2GBC33S02342408	0	60846		<Description>Uilaat nakijken</Description>	11	1	11-04561	11-04561	20160107	0
W0L04HD3325334642	377,02	285132	Distributieriem	<Description>Distributieriem set</Description> <Description>Distributieriem vervangen</Description> <Description>Koelvloeistof, Dex-Cool long life coolant</Description> <Description>Pakking</Description> <Description>Plugje</Description> <Description>Waterpomp</Description> <Description>Waterpomp uit- en inbouwen</Description>	1	1	01-129775	01-129775	20160206	0
W0P0XDF6260534232	665,88	173512	ondehoud/apk/diversen	<Description>Aandrijfas voor, afdichting hoes repareren/reinige</Description> <Description>Afvoer Klein Chemisch Afval</Description> <Description>APK i.c.m. beurt + afmeld-en administratiekosten</Description> <Description>Batterij CR2032</Description> <Description>Certificaat</Description> <Description>Diagnoseapparatuur</Description> <Description>Draagarmrubber a.</Description> <Description>Draagarmrubber vervangen - Rechterkant</Description> <Description>Keerring, aftapplug</Description> <Description>Motorolie10w-40 sj/cf</Description> <Description>Motorsteun vervangen - bovenzijde</Description> <Description>Motorsteundemper</Description> <Description>Oliefilter</Description> <Description>Onderhoudsbeurt</Description> <Description>Opel Assistance Plus</Description> <Description>Opmerking monteur bijgeluid hoorbaar in bochten</Description> <Description>Pechhulp</Description> <Description>Pollenfilter</Description> <Description>Reiniger</Description> <Description>Ruitensproeiervloeistof 500 ml</Description> <Description>Ruitenwissers vooraan vervangen</Description> <Description>Ruitewissers set</Description> <Description>Wiellager achteraan vervangen - links</Description> <Description>Wiellager achteraan vervangen - rechts</Description> <Description>Wielrollager, achteraan</Description>	2	1	02-40252	02-40252	20160115	0
W0LDD9E91410345273	70	19	WIBA	<Description>Wissel + opslag banden seizoen ZOMR&gt; WINTR</Description>	11	1	11-01954	11-01954	20160122	1

Table 8.2: Sample of Carlo Dealer Management System

AUTO	FACTUUR	ARBEID	CODE	OMSCHRIJVING	AANTAL	BEDRAG	FACTUUR	ISSECTIE
106	2131654	A	APK	APK KEURING UITGEVOERD	1.00	79.75	244011	NEE
106	2131654	A	SUBTOTAAL	SUBTOTAAL INCL. BTW : 19.95 EUR	1.00	-67.06	244011	NEE
291	1501683	A		ADVIES GEGEVEN OM DIST. RIEM TE VERVANGEN	0.00	0.00	350315	NEE
320	9414864	O	1609525280	DISTRIBUTIESET + K.V.POMP	1.00	212.90	3249979	NEE
9051	3340121	A		RUITRUBBERS VV			881519	JA

Table 8.3: Sample of I'Car Dealer Management System (WPLD section)

dms	autoId	klant	filiaal	type	soort	datum	kmstand	bedrag	factuur	factuurdatum	order	afdeling	credit	klant	opdracht
0	BF 403 X	305870	5	H	W	09-12-15	4	0.00	746126	04-05-11	746911	0	F	1	/ Afleveringsbeurt
0	18 XTD 0	318954	5	H	W	09-12-15	117458	0.00	746122	02-08-12	747499	0	F	1	/ Totale onderhoudsbeurt bij 150.000 km
0	91PVV8	313622	10	H	W	09-12-15	170901	740.02	1013707	29-12-09	1014176	0	F	1	/ APK keuring (gratis tijdens beurt)
0	0PDD91	463914	10	H	W	09-12-15	56385	423.98	1013680	09-12-12	1014050	0	F	1	
0	25 MR DP	133058	2	H	W	10-12-15	187934	474.16	300274	10-03-13	312031	0	F	1	17001 / Volvo jaarbeurt
5	AF 290 S	138324	1	H	W	18-06-15	5	0.00	739997	12-06-13	740967	0	F	51	/ 0- beurt
5	XN 086 E	923817	1	H	W	03-08-15	5	0.00	740226	01-08-14	740932	0	F	51	/ 0-beurt uitgevoerd
5	ML 823 V	163829	1	H	W	31-08-15	5	0.00	740346	03-11-10	740928	0	F	51	/ 0 km Onderhoudsbeurt uitgevoer
5	PK TY 16	395739	1	H	W	20-01-15	488294	445.79	739136	20-04-14	739989	0	F	1	/ Winter controle-beurt

Table 8.4: Sample of iDAS Dealer Management System

Id	Auto dms id	Bezoek datum	No	Oh	Soort	Code	Tekst	Rule
18847	01-EGF-2	30-04-2015	1	0	PK	1490	Gloeilamp koplamp vervangen (1)	code
15921	45-ODS-G	22-11-2015	1	0	PK	APK001	APK keuring	code
18210	38-POP-9	11-03-2015	10	0	PK	2110	Band (1x) monteren en balanceren	code
100275	18-LO-IE	17-01-2015	4	0	PK	000001	onderhoudsbeurt uitgevoerd	code
236193	82-SF-PD	01-02-2015	1	1	PK	4500	Beurt na 5 jaar/100.000 km	code
129110	13-LIX-1	16-11-2015	3	1	PK	9954	Comfortpakket tussentijds onderhoud	code

Table 8.5: Sample of normalized data Dealer Management System

In this sections some names are abbreviated. Here you can find the different abbreviations;

Columns descriptions

Model The name of the model.

Size The data set used for generation.

Train The time to train the model.

Eval The time the model took to classify the evaluation set.

Acc The accuracy of the model.

Prec The precision of the model.

Rec The recall of the model.

$\mathbf{F_1}$ The F_1 score of the model.

Option Types

None No additional options.

Detectors The token detectors which we introduced as option.

Stemming A stemming operation is used as pre-processing step.

Typo The typographical error correction option.

Compound The compound word splitting option.

Data set sizes

Small The small data set having $5 \cdot 10^3$ rows.

Med The small data set having 10^4 rows.

Large The small data set having 10^5 rows.

Model types

NN Neural Network.

SVM Support Vector Machine.

Forest Random Forest.

Bayes Naive Bayes.

Results of C# implementation

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
Bayes	Small	00:00:00.60	00:00:00.11	None	0,886	0,817	0,994	0,897
Forest		00:00:01.58	00:00:00.05		0,978	0,998	0,959	0,978
NN		00:01:53.26	00:00:00.06		0,995	0,994	0,996	0,995
SVM		00:00:17.04	00:00:01.85		0,992	0,992	0,992	0,992
Bayes		00:00:00.31	00:00:00.17	Detectors	0,886	0,817	0,994	0,897
Forest		00:00:02.46	00:00:00.14		0,978	0,998	0,959	0,978
NN		00:01:54.88	00:00:00.16		0,994	0,992	0,996	0,994
SVM		00:00:15.37	00:00:01.77		0,992	0,992	0,992	0,992
Bayes		00:00:00.35	00:00:00.37	Stemming	0,891	0,824	0,994	0,901
Forest		00:00:01.51	00:00:00.15		0,980	0,998	0,962	0,980
NN		00:01:49.90	00:00:00.14		0,994	0,992	0,996	0,994
SVM		00:00:14.97	00:00:01.75		0,992	0,992	0,992	0,992
Bayes		00:00:00.40	00:00:00.59	Typo	0,891	0,824	0,994	0,901
Forest		00:00:01.76	00:00:00.72		0,913	0,998	0,831	0,907
NN		00:01:51.80	00:00:00.56		0,993	0,990	0,996	0,993
SVM		00:00:16.48	00:00:02.21		0,988	0,992	0,984	0,988
Bayes		00:00:00.61	00:00:01.36	Compound	0,891	0,824	0,994	0,901
Forest		00:00:01.33	00:00:00.69		0,913	0,998	0,831	0,907
NN		00:01:52.27	00:00:00.56		0,994	0,994	0,994	0,994
SVM		00:00:15.87	00:00:02.24		0,988	0,992	0,984	0,988
Bayes	Med	00:00:00.88	00:00:00.29	None	0,892	0,826	0,989	0,900
Forest		00:00:02.80	00:00:00.14		0,979	0,995	0,962	0,978
NN		00:05:26.78	00:00:00.17		0,995	0,993	0,996	0,994
SVM		00:01:05.94	00:00:44.14		0,995	0,994	0,996	0,995
Bayes		00:00:00.90	00:00:00.46	Detectors	0,892	0,827	0,989	0,901
Forest		00:00:02.56	00:00:00.32		0,979	0,995	0,962	0,978
NN		00:05:28.61	00:00:00.34		0,995	0,993	0,996	0,994
SVM		00:01:02.15	00:00:07.54		0,995	0,994	0,996	0,995
Bayes		00:00:00.89	00:00:00.46	Stemming	0,893	0,828	0,989	0,902
Forest		00:00:03.29	00:00:00.36		0,979	0,995	0,963	0,979
NN		00:05:18.84	00:00:00.34		0,996	0,995	0,996	0,995
SVM		00:01:09.26	00:00:07.96		0,995	0,994	0,996	0,995
Bayes		00:00:01.01	00:00:01.65	Typo	0,897	0,834	0,988	0,904
Forest		00:00:03.01	00:00:01.93		0,883	0,986	0,779	0,870
NN		00:05:25.03	00:00:01.48		0,996	0,994	0,998	0,996
SVM		00:01:05.22	00:00:08.63		0,995	0,992	0,998	0,995
Bayes		00:00:02.06	00:00:04.32	Compound	0,897	0,834	0,988	0,904
Forest		00:00:02.60	00:00:01.86		0,883	0,986	0,779	0,870
NN		00:05:25.05	00:00:01.49		0,996	0,993	0,998	0,995
SVM		00:01:04.32	00:00:08.53		0,995	0,992	0,998	0,995
Bayes	Large	00:01:13.07	00:00:13.61	None	0,928	0,874	0,999	0,932
Forest		00:00:23.54	00:00:05.17		0,993	0,998	0,989	0,993
NN		03:35:33.24	00:00:07.43		0,999	0,999	1,000	0,999
Bayes		00:01:08.04	00:00:21.28	Detectors	0,928	0,874	0,999	0,932
Forest		00:00:23.59	00:00:07.95		0,993	0,998	0,988	0,993
NN		03:34:12.11	00:00:09.07		0,999	0,999	1,000	0,999
Bayes		00:01:07.50	00:00:24.14	Stemming	0,928	0,874	0,999	0,932
Forest		00:00:22.50	00:00:07.04		0,993	0,998	0,988	0,993
NN		03:24:33.87	00:00:08.59		0,999	0,998	1,000	0,999
Bayes		00:01:11.62	00:00:50.16	Typo	0,928	0,875	0,999	0,933
Forest		00:00:40.49	00:00:25.75		0,928	0,977	0,884	0,928
NN		03:29:23.20	00:00:21.73		0,999	0,998	1,000	0,999
Bayes		00:01:19.52	00:00:42.94	Compound	0,928	0,875	0,999	0,933
Forest		00:00:42.14	00:00:25.91		0,928	0,977	0,884	0,928
NN		03:29:55.13	00:00:21.82		0,999	0,998	1,000	0,999

Table 8.6: Results of custom implementations for class Maintenance

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
Bayes	Small	00:00:00.40	00:00:00.11	None	0,889	0,818	1,000	0,900
Forest		00:00:01.37	00:00:00.05		0,975	0,996	0,955	0,975
NN		00:01:58.34	00:00:00.11		0,996	0,992	1,000	0,996
SVM		00:00:17.82	00:00:02.33		0,994	0,994	0,994	0,994
Bayes		00:00:00.46	00:00:00.20	Detectors	0,889	0,818	1,000	0,900
Forest		00:00:01.83	00:00:00.15		0,976	0,996	0,957	0,976
NN		00:01:58.63	00:00:00.17		0,997	0,994	1,000	0,997
SVM		00:00:20.21	00:00:02.70		0,994	0,994	0,994	0,994
Bayes		00:00:00.47	00:00:00.19	Stemming	0,894	0,825	1,000	0,904
Forest		00:00:01.27	00:00:00.14		0,970	0,982	0,959	0,970
NN		00:01:56.24	00:00:00.15		0,997	0,994	1,000	0,997
SVM		00:00:19.78	00:00:02.50		0,994	0,994	0,994	0,994
Bayes		00:00:00.63	00:00:01.39	Typo	0,898	0,831	1,000	0,907
Forest		00:00:01.40	00:00:00.62		0,788	0,990	0,588	0,738
NN		00:01:58.78	00:00:00.63		0,996	0,992	1,000	0,996
SVM		00:00:14.92	00:00:02.28		0,997	0,996	0,998	0,997
Bayes		00:00:00.66	00:00:01.33	Compound	0,898	0,831	1,000	0,907
Forest		00:00:01.47	00:00:00.63		0,788	0,990	0,588	0,738
NN		00:01:58.52	00:00:00.63		0,996	0,992	1,000	0,996
SVM		00:00:16.46	00:00:02.35		0,997	0,996	0,998	0,997
Bayes	Med	00:00:01.22	00:00:00.37	None	0,987	0,974	1,000	0,987
Forest		00:00:01.68	00:00:00.19		0,985	0,992	0,977	0,985
NN		00:06:40.39	00:00:00.20		0,996	0,992	1,000	0,996
SVM		00:01:34.67	00:00:11.16		0,993	0,989	0,997	0,993
Bayes		00:00:01.14	00:00:00.54	Detectors	0,987	0,974	1,000	0,987
Forest		00:00:01.91	00:00:00.35		0,984	0,992	0,976	0,984
NN		00:06:42.40	00:00:00.38		0,997	0,993	1,000	0,997
SVM		00:01:38.07	00:00:12.88		0,993	0,989	0,997	0,993
Bayes		00:00:01.19	00:00:00.52	Stemming	0,988	0,976	1,000	0,988
Forest		00:00:01.65	00:00:00.41		0,983	0,992	0,974	0,983
NN		00:06:30.73	00:00:00.38		0,996	0,992	0,999	0,996
SVM		00:01:34.00	00:00:10.98		0,993	0,989	0,997	0,993
Bayes		00:00:02.12	00:00:03.27	Typo	0,988	0,976	1,000	0,988
Forest		00:00:01.78	00:00:01.32		0,881	0,957	0,818	0,882
NN		00:06:40.27	00:00:01.37		0,996	0,992	1,000	0,996
SVM		00:01:24.35	00:00:09.30		0,994	0,989	0,998	0,994
Bayes		00:00:02.30	00:00:03.26	Compound	0,988	0,976	1,000	0,988
Forest		00:00:01.98	00:00:01.34		0,881	0,957	0,818	0,882
NN		00:06:39.89	00:00:01.38		0,996	0,992	1,000	0,996
SVM		00:01:23.50	00:00:09.14		0,994	0,989	0,998	0,994
Bayes	Large	00:10:12.90	00:10:17.90	None	0,997	0,985	0,998	0,992
Forest		00:05:28.40	00:02:13.67		0,998	0,998	0,990	0,994
NN		03:44:35.64	00:00:15.46		0,998	0,998	0,992	0,995
Bayes		00:11:42.72	00:09:32.74	Detectors	0,997	0,984	0,998	0,991
Forest		00:00:46.93	00:00:09.87		0,998	0,998	0,989	0,994
NN		03:45:11.23	00:00:23.18		0,999	0,998	0,995	0,997
Bayes		00:01:09.57	00:00:28.40	Stemming	0,997	0,984	0,998	0,991
Forest		00:00:31.40	00:00:07.59		0,998	0,998	0,990	0,994
NN		03:34:47.68	00:00:09.03		0,999	0,998	0,994	0,996
Bayes		00:01:10.35	00:00:44.72	Typo	0,997	0,984	0,998	0,991
Forest		00:05:30.99	00:03:04.21		0,888	0,960	0,543	0,693
NN		03:39:38.67	00:00:21.58		0,998	0,998	0,992	0,995
Bayes		00:01:09.87	00:00:47.78	Compound	0,997	0,984	0,998	0,991
Forest		00:07:40.48	00:01:18.47		0,888	0,960	0,543	0,693
NN		03:39:41.89	00:00:21.57		0,998	0,999	0,991	0,995

Table 8.7: Results of custom implementations for class Serpentine belt

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
Bayes	Small	00:00:00.75	00:00:00.28	None	0,985	0,986	0,984	0,985
Forest		00:00:00.00	00:00:00.04		0,500	0,500	1,000	0,667
NN		00:02:12.09	00:00:00.06		0,983	0,988	0,978	0,983
SVM		00:00:27.12	00:00:02.47		0,981	0,984	0,978	0,981
Bayes		00:00:00.42	00:00:00.19	Detectors	0,985	0,984	0,986	0,985
Forest		00:00:01.38	00:00:00.13		0,972	0,992	0,953	0,972
NN		00:02:12.35	00:00:00.40		0,991	0,988	0,994	0,991
SVM		00:00:21.57	00:00:02.19		0,987	0,982	0,992	0,987
Bayes		00:00:00.45	00:00:00.20	Stemming	0,985	0,984	0,986	0,985
Forest		00:00:01.05	00:00:00.14		0,977	0,994	0,961	0,977
NN		00:02:09.86	00:00:00.16		0,992	0,990	0,994	0,992
SVM		00:00:22.37	00:00:02.33		0,988	0,984	0,992	0,988
Bayes		00:00:00.39	00:00:01.65	Typo	0,983	0,982	0,984	0,983
Forest		00:00:01.13	00:00:01.00		0,834	0,992	0,685	0,811
NN		00:02:12.48	00:00:00.98		0,991	0,988	0,994	0,991
SVM		00:00:23.96	00:00:02.57		0,982	0,976	0,988	0,982
Bayes		00:00:01.13	00:00:00.92	Compound	0,983	0,982	0,984	0,983
Forest		00:00:01.14	00:00:00.91		0,834	0,992	0,685	0,811
NN		00:02:12.23	00:00:00.83		0,989	0,984	0,994	0,989
SVM		00:00:21.80	00:00:02.52		0,982	0,976	0,988	0,982
Bayes	Med	00:00:02.01	00:00:00.96	None	0,985	0,979	0,990	0,985
Forest		00:00:00.00	00:00:00.13		0,500	0,500	1,000	0,667
NN		00:06:30.04	00:00:00.20		0,993	0,995	0,990	0,992
SVM		00:01:47.98	00:00:11.49		0,990	0,991	0,989	0,990
Bayes		00:00:01.15	00:00:00.50	Detectors	0,985	0,980	0,989	0,985
Forest		00:00:02.61	00:00:00.34		0,985	0,991	0,978	0,985
NN		00:06:31.57	00:00:00.37		0,993	0,991	0,995	0,993
SVM		00:01:28.23	00:00:11.11		0,993	0,991	0,994	0,993
Bayes		00:00:01.01	00:00:00.50	Stemming	0,985	0,982	0,988	0,985
Forest		00:00:02.09	00:00:00.33		0,988	0,992	0,983	0,987
NN		00:06:21.90	00:00:00.38		0,994	0,992	0,996	0,994
SVM		00:01:27.38	00:00:11.67		0,993	0,990	0,995	0,993
Bayes		00:00:01.07	00:00:01.88	Typo	0,985	0,982	0,988	0,985
Forest		00:00:02.49	00:00:01.75		0,715	0,996	0,446	0,616
NN		00:06:32.62	00:00:01.78		0,996	0,994	0,997	0,996
SVM		00:01:26.24	00:00:10.80		0,993	0,989	0,996	0,993
Bayes		00:00:01.04	00:00:01.90	Compound	0,985	0,982	0,988	0,985
Forest		00:00:02.21	00:00:01.71		0,715	0,996	0,446	0,616
NN		00:06:32.62	00:00:01.78		0,994	0,992	0,996	0,994
SVM		00:01:24.41	00:00:11.24		0,993	0,989	0,996	0,993
Bayes	Large	00:17:51.42	00:04:48.54	None	0,994	0,990	0,998	0,994
Forest		00:00:41.72	00:00:08.26		0,986	0,994	0,978	0,986
NN		03:56:12.73	00:00:08.38		0,998	0,996	1,000	0,998
Bayes		00:13:21.20	00:13:07.85	Detectors	0,994	0,991	0,998	0,994
Forest		00:00:42.08	00:00:09.43		0,990	0,994	0,987	0,990
NN		03:58:16.63	00:01:38.56		0,998	0,995	1,000	0,998
Bayes		00:07:00.89	00:09:25.84	Stemming	0,994	0,991	0,998	0,994
Forest		00:00:30.86	00:00:09.05		0,990	0,992	0,987	0,990
NN		03:49:00.52	00:01:33.58		0,998	0,995	1,000	0,998
Bayes		00:29:02.56	00:15:44.47	Typo	0,994	0,991	0,998	0,994
Forest		00:01:12.50	00:00:22.72		0,767	0,989	0,559	0,714
NN		03:53:13.81	00:00:23.22		0,997	0,995	0,999	0,997
Bayes		00:04:35.91	00:05:21.24	Compound	0,994	0,991	0,998	0,994
Forest		00:01:17.42	00:00:22.87		0,767	0,989	0,559	0,714
NN		03:53:07.26	00:00:23.72		0,997	0,994	1,000	0,997

Table 8.8: Results of custom implementations for class Tyres

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
Bayes	Small	00:00:00.70	00:00:00.25	None	0,990	0,990	0,990	0,990
Forest		00:00:00.00	00:00:00.05		0,989	0,996	0,982	0,989
NN		00:01:45.66	00:00:00.05		0,996	0,996	0,996	0,996
SVM		00:00:10.63	00:00:01.34		0,996	0,996	0,996	0,996
Bayes		00:00:00.32	00:00:00.16	Detectors	0,990	0,990	0,990	0,990
Forest		00:00:01.17	00:00:00.12		0,963	0,986	0,943	0,964
NN		00:01:45.79	00:00:00.26		0,997	0,996	0,998	0,997
SVM		00:00:09.56	00:00:01.40		0,996	0,996	0,996	0,996
Bayes		00:00:00.42	00:00:00.55	Stemming	0,990	0,990	0,990	0,990
Forest		00:00:01.09	00:00:00.12		0,959	0,986	0,936	0,960
NN		00:01:43.23	00:00:00.28		0,997	0,998	0,996	0,997
SVM		00:00:09.01	00:00:01.36		0,996	0,996	0,996	0,996
Bayes		00:00:00.44	00:00:00.59	Typo	0,989	0,990	0,988	0,989
Forest		00:00:00.80	00:00:00.64		0,974	0,996	0,954	0,974
NN		00:01:45.19	00:00:01.13		0,999	0,998	1,000	0,999
SVM		00:00:08.05	00:00:01.57		0,997	0,996	0,998	0,997
Bayes		00:00:00.34	00:00:00.58	Compound	0,989	0,990	0,988	0,989
Forest		00:00:01.25	00:00:00.53		0,974	0,996	0,954	0,974
NN		00:01:45.26	00:00:00.55		0,999	0,998	1,000	0,999
SVM		00:00:07.87	00:00:01.58		0,997	0,996	0,998	0,997
Bayes	Med	00:00:01.35	00:00:00.32	None	0,998	0,996	1,000	0,998
Forest		00:00:00.00	00:00:00.12		0,991	0,999	0,983	0,991
NN		00:06:14.19	00:00:00.18		1,000	0,999	1,000	1,000
SVM		00:00:57.76	00:00:05.63		0,999	0,999	0,999	0,999
Bayes		00:00:01.13	00:00:00.46	Detectors	0,998	0,996	1,000	0,998
Forest		00:00:01.69	00:00:00.31		0,995	0,997	0,992	0,995
NN		00:06:14.67	00:00:00.34		1,000	0,999	1,000	1,000
SVM		00:00:53.52	00:00:05.66		0,999	0,999	0,999	0,999
Bayes		00:00:01.01	00:00:00.45	Stemming	0,999	0,997	1,000	0,999
Forest		00:00:01.40	00:00:00.60		0,994	0,997	0,990	0,994
NN		00:06:05.79	00:00:00.33		1,000	0,999	1,000	1,000
SVM		00:00:53.57	00:00:05.32		0,999	0,999	0,999	0,999
Bayes		00:00:01.24	00:00:01.60	Typo	0,999	0,997	1,000	0,999
Forest		00:00:01.71	00:00:01.44		0,906	1,000	0,821	0,902
NN		00:06:14.37	00:00:01.48		0,999	0,999	0,999	0,999
SVM		00:00:39.83	00:00:04.47		0,998	0,999	0,997	0,998
Bayes		00:00:01.06	00:00:01.60	Compound	0,999	0,997	1,000	0,999
Forest		00:00:01.67	00:00:01.50		0,906	1,000	0,821	0,902
NN		00:06:16.17	00:00:17.98		1,000	0,999	1,000	1,000
SVM		00:00:40.29	00:00:04.49		0,998	0,999	0,997	0,998
Bayes	Large	00:03:46.35	00:02:58.21	None	0,999	0,993	0,999	0,996
Forest		00:01:01.33	00:00:07.48		0,998	0,991	0,997	0,994
NN		03:45:55.88	00:00:07.48		0,999	0,997	0,999	0,998
Bayes		00:02:54.49	00:02:53.62	Detectors	0,999	0,993	0,999	0,996
Forest		00:00:14.28	00:00:08.91		0,998	0,991	0,997	0,994
NN		03:46:07.57	00:00:09.25		0,999	0,997	0,996	0,996
Bayes		00:14:47.87	00:04:30.65	Stemming	0,999	0,993	0,999	0,996
Forest		00:00:11.68	00:00:08.00		0,998	0,991	0,997	0,994
NN		03:36:54.91	00:00:09.19		0,999	0,998	0,995	0,996
Bayes		00:13:11.52	00:08:16.38	Typo	0,999	0,993	0,999	0,996
Forest		00:00:28.98	00:00:21.51		0,885	0,993	0,487	0,653
NN		03:42:27.21	00:00:24.04		0,999	0,998	0,997	0,998
Bayes		00:18:03.48	00:08:16.49	Compound	0,999	0,993	0,999	0,996
Forest		00:00:19.80	00:00:21.26		0,885	0,993	0,487	0,653
NN		03:42:19.58	00:00:22.28		0,999	0,998	0,997	0,998

Table 8.9: Results of custom implementations for class Battery

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
Bayes	Small	00:00:00.50	00:00:00.12	None	0,995	0,990	1,000	0,995
Forest		00:00:00.00	00:00:00.04		0,761	0,996	0,524	0,687
NN		00:02:20.06	00:00:00.07		0,999	0,998	1,000	0,999
SVM		00:00:23.44	00:00:02.32		0,991	0,998	0,984	0,991
Bayes		00:00:00.43	00:00:00.20	Detectors	0,995	0,990	1,000	0,995
Forest		00:00:00.89	00:00:00.22		0,969	0,994	0,946	0,970
NN		00:02:20.42	00:00:00.14		0,999	0,998	1,000	0,999
SVM		00:00:22.04	00:00:02.43		0,991	0,998	0,984	0,991
Bayes		00:00:00.42	00:00:00.20	Stemming	0,995	0,990	1,000	0,995
Forest		00:00:00.82	00:00:00.13		0,968	0,996	0,943	0,969
NN		00:02:17.10	00:00:00.16		0,998	0,998	0,998	0,998
SVM		00:00:21.74	00:00:02.25		0,993	0,998	0,988	0,993
Bayes		00:00:00.78	00:00:01.89	Typo	0,995	0,990	1,000	0,995
Forest		00:00:00.96	00:00:00.73		0,933	0,988	0,887	0,935
NN		00:02:20.40	00:00:00.63		0,999	0,998	1,000	0,999
SVM		00:00:18.76	00:00:02.66		0,992	0,998	0,986	0,992
Bayes		00:00:00.82	00:00:01.45	Compound	0,995	0,990	1,000	0,995
Forest		00:00:03.11	00:00:00.61		0,933	0,988	0,887	0,935
NN		00:02:19.82	00:00:00.64		0,998	0,996	1,000	0,998
SVM		00:00:18.77	00:00:02.67		0,992	0,998	0,986	0,992
Bayes	Med	00:00:01.29	00:00:00.35	None	0,999	0,998	0,999	0,999
Forest		00:00:00.00	00:00:00.13		0,771	1,000	0,542	0,703
NN		00:06:51.02	00:00:00.20		1,000	1,000	1,000	1,000
SVM		00:01:29.78	00:00:09.23		0,992	1,000	0,983	0,991
Bayes		00:00:01.21	00:00:00.50	Detectors	0,999	0,998	0,999	0,999
Forest		00:00:03.11	00:00:00.32		0,988	0,998	0,978	0,988
NN		00:06:50.60	00:00:00.35		1,000	1,000	0,999	1,000
SVM		00:01:29.63	00:00:09.34		0,992	1,000	0,983	0,991
Bayes		00:00:01.32	00:00:00.50	Stemming	0,998	0,997	0,999	0,998
Forest		00:00:01.76	00:00:00.31		0,987	0,999	0,975	0,987
NN		00:06:41.90	00:00:00.36		1,000	1,000	1,000	1,000
SVM		00:01:29.08	00:00:08.85		0,993	1,000	0,985	0,992
Bayes		00:00:02.28	00:00:03.50	Typo	0,998	0,996	0,999	0,998
Forest		00:00:03.10	00:00:01.50		0,941	0,995	0,893	0,941
NN		00:06:51.30	00:00:01.50		0,996	0,999	0,993	0,996
SVM		00:01:24.49	00:00:09.43		0,991	1,000	0,982	0,991
Bayes		00:00:03.38	00:00:01.66	Compound	0,998	0,996	0,999	0,998
Forest		00:00:02.01	00:00:01.44		0,941	0,995	0,893	0,941
NN		00:06:50.77	00:00:01.51		0,996	0,999	0,993	0,996
SVM		00:01:23.31	00:00:09.32		0,991	1,000	0,982	0,991
Bayes	Large	00:10:54.18	00:07:37.58	None	0,995	0,947	0,987	0,966
Forest		00:03:14.60	00:06:22.18		0,998	0,995	0,971	0,982
NN		03:43:51.82	00:00:07.55		0,999	0,996	0,986	0,991
Bayes		00:03:08.55	00:03:25.78	Detectors	0,995	0,946	0,987	0,966
Forest		00:02:15.91	00:03:14.60		0,998	0,995	0,971	0,982
NN		03:44:08.58	00:00:09.49		0,999	0,997	0,987	0,992
Bayes		00:01:07.10	00:00:23.34	Stemming	0,996	0,947	0,994	0,970
Forest		00:01:37.47	00:03:34.70		0,998	0,995	0,970	0,982
NN		03:38:20.70	00:00:09.13		0,999	0,998	0,981	0,989
Bayes		00:01:08.87	00:00:47.49	Typo	0,996	0,947	0,994	0,970
Forest		00:02:08.76	00:00:27.91		0,964	0,853	0,688	0,762
NN		03:40:49.18	00:00:21.64		0,995	0,995	0,938	0,965
Bayes		00:01:06.38	00:00:44.43	Compound	0,996	0,947	0,994	0,970
Forest		00:05:29.32	00:02:32.02		0,964	0,853	0,688	0,762
NN		03:40:29.18	00:00:21.79		0,995	0,997	0,929	0,962

Table 8.10: Results of custom implementations for class V-belt

Model	Size	Train	Eval	Options	Acc	Prec	Rec	F_1
Bayes	Small	00:00:00.36	00:00:00.10	None	0,972	0,957	0,988	0,972
Forest		00:00:00.00	00:00:00.04		0,500	0,500	1,000	0,667
NN		00:02:01.85	00:00:00.06		0,991	0,992	0,990	0,991
SVM		00:00:22.92	00:00:01.84		0,988	0,996	0,980	0,988
Bayes		00:00:00.47	00:00:00.18	Detectors	0,972	0,957	0,988	0,972
Forest		00:00:01.56	00:00:00.27		0,962	0,990	0,935	0,962
NN		00:02:02.07	00:00:00.15		0,991	0,992	0,990	0,991
SVM		00:00:23.36	00:00:01.99		0,988	0,996	0,980	0,988
Bayes		00:00:01.17	00:00:00.23	Stemming	0,973	0,959	0,988	0,973
Forest		00:00:01.63	00:00:00.14		0,964	0,994	0,935	0,964
NN		00:01:59.26	00:00:00.36		0,989	0,990	0,988	0,989
SVM		00:00:21.16	00:00:01.98		0,988	0,996	0,980	0,988
Bayes		00:00:00.51	00:00:00.66	Typo	0,975	0,963	0,988	0,975
Forest		00:00:01.80	00:00:00.73		0,775	0,990	0,571	0,724
NN		00:02:00.99	00:00:00.60		0,990	0,990	0,990	0,990
SVM		00:00:17.95	00:00:01.81		0,986	0,996	0,976	0,986
Bayes		00:00:00.83	00:00:00.65	Compound	0,975	0,963	0,988	0,975
Forest		00:00:02.36	00:00:00.59		0,775	0,990	0,571	0,724
NN		00:02:00.94	00:00:00.61		0,992	0,992	0,992	0,992
SVM		00:00:17.73	00:00:01.85		0,986	0,996	0,976	0,986
Bayes	Med	00:00:00.92	00:00:00.31	None	0,984	0,975	0,993	0,984
Forest		00:00:00.00	00:00:00.11		0,500	0,500	1,000	0,667
NN		00:05:37.87	00:00:00.17		0,994	0,990	0,998	0,994
SVM		00:01:16.86	00:00:09.13		0,997	0,996	0,998	0,997
Bayes		00:00:01.95	00:00:00.46	Detectors	0,985	0,976	0,993	0,985
Forest		00:00:02.52	00:00:00.41		0,975	0,984	0,965	0,974
NN		00:05:38.39	00:00:00.35		0,993	0,988	0,997	0,993
SVM		00:01:13.62	00:00:09.29		0,997	0,996	0,998	0,997
Bayes		00:00:00.93	00:00:00.46	Stemming	0,984	0,975	0,993	0,984
Forest		00:00:02.51	00:00:00.42		0,977	0,987	0,966	0,976
NN		00:05:29.42	00:00:00.35		0,995	0,991	0,998	0,995
SVM		00:01:06.49	00:00:08.93		0,997	0,995	0,999	0,997
Bayes		00:00:00.92	00:00:01.49	Typo	0,987	0,980	0,993	0,987
Forest		00:00:02.41	00:00:01.61		0,766	0,989	0,544	0,702
NN		00:05:35.56	00:00:01.34		0,996	0,992	0,999	0,996
SVM		00:00:52.39	00:00:06.91		0,996	0,993	0,998	0,996
Bayes		00:00:00.92	00:00:01.41	Compound	0,987	0,980	0,993	0,987
Forest		00:00:04.38	00:00:01.40		0,766	0,989	0,544	0,702
NN		00:05:35.56	00:00:01.34		0,996	0,993	0,998	0,996
SVM		00:00:54.06	00:00:07.09		0,996	0,993	0,998	0,996
Bayes	Large	00:14:08.41	00:10:17.26	None	0,998	0,998	0,998	0,998
Forest		00:01:33.33	00:00:07.97		0,991	0,998	0,982	0,990
NN		03:45:04.40	00:00:07.57		0,999	0,998	0,999	0,999
Bayes		00:07:26.38	00:05:34.05	Detectors	0,998	0,998	0,998	0,998
Forest		00:01:01.63	00:00:15.70		0,991	0,998	0,982	0,990
NN		03:45:13.02	00:00:09.44		0,999	0,998	0,999	0,998
Bayes		00:11:49.88	00:02:07.25	Stemming	0,998	0,998	0,998	0,998
Forest		00:01:30.69	00:00:18.36		0,992	0,998	0,984	0,991
NN		03:36:21.88	00:00:09.32		0,999	0,999	0,999	0,999
Bayes		00:06:00.52	00:12:11.14	Typo	0,998	0,998	0,998	0,998
Forest		00:02:32.86	00:01:00.90		0,852	0,994	0,708	0,827
NN		03:42:31.35	00:00:22.82		0,997	0,997	0,997	0,997
Bayes		00:06:11.85	00:06:55.38	Compound	0,998	0,998	0,998	0,998
Forest		00:02:57.64	00:00:50.59		0,852	0,994	0,708	0,827
NN		03:42:56.66	00:02:37.69		0,997	0,997	0,996	0,997

Table 8.11: Results of custom implementations for class Brakes

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
Bayes	Small	00:00:00.27	00:00:00.09	None	0,960	0,928	0,998	0,961
Forest		00:00:00.00	00:00:00.03		0,888	0,995	0,780	0,874
NN		00:01:44.46	00:00:00.05		0,998	0,998	0,998	0,998
SVM		00:00:15.98	00:00:01.41		0,997	0,998	0,996	0,997
Bayes		00:00:00.48	00:00:00.25	Detectors	0,960	0,928	0,998	0,961
Forest		00:00:00.73	00:00:00.15		0,968	0,990	0,948	0,969
NN		00:01:45.58	00:00:00.30		0,998	0,998	0,998	0,998
SVM		00:00:14.10	00:00:01.39		0,997	0,998	0,996	0,997
Bayes		00:00:00.33	00:00:00.17	Stemming	0,960	0,928	0,998	0,961
Forest		00:00:00.63	00:00:00.12		0,977	0,990	0,965	0,977
NN		00:01:41.14	00:00:00.30		0,998	0,998	0,998	0,998
SVM		00:00:14.44	00:00:01.39		0,997	0,998	0,996	0,997
Bayes		00:00:00.55	00:00:01.35	Typo	0,959	0,926	0,998	0,961
Forest		00:00:00.94	00:00:01.29		0,948	0,990	0,913	0,950
NN		00:01:42.79	00:00:00.60		0,997	0,996	0,998	0,997
SVM		00:00:14.88	00:00:01.79		0,997	0,998	0,996	0,997
Bayes		00:00:00.36	00:00:00.89	Compound	0,959	0,926	0,998	0,961
Forest		00:00:01.83	00:00:00.62		0,948	0,990	0,913	0,950
NN		00:01:43.05	00:00:00.60		0,998	0,998	0,998	0,998
SVM		00:00:14.05	00:00:01.79		0,997	0,998	0,996	0,997
Bayes	Med	00:00:01.11	00:00:00.33	None	0,990	0,980	1,000	0,990
Forest		00:00:00.00	00:00:00.12		0,887	0,999	0,774	0,872
NN		00:06:12.58	00:00:00.18		0,998	0,997	0,999	0,998
SVM		00:01:21.66	00:00:08.31		0,998	0,998	0,997	0,997
Bayes		00:00:01.66	00:00:00.58	Detectors	0,989	0,978	1,000	0,989
Forest		00:00:01.33	00:00:00.40		0,989	0,998	0,980	0,989
NN		00:06:15.07	00:00:00.34		0,999	0,997	1,000	0,999
SVM		00:01:24.32	00:00:08.58		0,998	0,998	0,997	0,997
Bayes		00:00:01.08	00:00:00.49	Stemming	0,990	0,981	0,999	0,990
Forest		00:00:01.40	00:00:00.34		0,989	0,998	0,980	0,989
NN		00:06:01.70	00:00:00.36		0,999	0,998	1,000	0,999
SVM		00:01:18.89	00:00:08.40		0,997	0,998	0,996	0,997
Bayes		00:00:01.91	00:00:03.35	Typo	0,991	0,982	0,999	0,991
Forest		00:00:01.48	00:00:01.34		0,947	0,998	0,904	0,949
NN		00:06:10.54	00:00:01.41		0,999	0,998	1,000	0,999
SVM		00:01:13.89	00:00:08.61		0,997	0,998	0,996	0,997
Bayes		00:00:01.32	00:00:02.34	Compound	0,991	0,982	0,999	0,991
Forest		00:00:01.39	00:00:01.36		0,947	0,998	0,904	0,949
NN		00:06:10.76	00:00:01.43		0,999	0,998	0,999	0,999
SVM		00:01:12.39	00:00:08.51		0,997	0,998	0,996	0,997
Bayes	Large	00:10:20.12	00:08:15.46	None	0,998	0,990	1,000	0,995
Forest		00:01:07.43	00:00:06.89		0,999	0,996	0,995	0,996
NN		03:45:21.03	00:09:42.13		0,999	0,998	0,998	0,998
Bayes		00:03:20.94	00:13:10.15	Detectors	0,998	0,990	1,000	0,995
Forest		00:01:03.87	00:00:07.95		0,999	0,996	0,995	0,996
NN		03:44:06.08	00:00:13.29		1,000	0,998	0,999	0,998
Bayes		00:01:07.26	00:00:25.55	Stemming	0,998	0,990	0,999	0,995
Forest		00:00:19.74	00:00:07.61		0,999	0,996	0,995	0,996
NN		03:33:55.34	00:00:08.92		0,999	0,998	0,997	0,998
Bayes		00:01:06.10	00:00:40.13	Typo	0,998	0,990	0,999	0,995
Forest		00:00:27.57	00:00:20.59		0,957	0,995	0,791	0,881
NN		03:39:57.22	00:00:22.75		0,999	0,998	0,994	0,996
Bayes		00:01:06.99	00:00:47.57	Compound	0,998	0,990	0,999	0,995
Forest		00:00:29.41	00:00:20.93		0,957	0,995	0,791	0,881
NN		03:39:59.67	00:00:22.92		0,999	0,998	0,995	0,997

Table 8.12: Results of custom implementations for class Air conditioning

Results of Azure Machine Learning

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
NN	Small	00:02:08	00:00:18	None	0,969	0,974	0,978	0,976
SVM		00:00:31	00:00:24		0,963	0,962	0,981	0,972
Forest		00:00:32	00:00:26		0,955	0,982	0,948	0,965
Bayes		00:00:17	00:00:19		0,956	0,976	0,955	0,966
NN		00:02:10	00:00:17	Detectors	0,970	0,974	0,980	0,977
SVM		00:00:35	00:00:20		0,964	0,962	0,983	0,972
Forest		00:00:30	00:00:27		0,963	0,982	0,960	0,971
Bayes		00:00:20	00:00:17		0,957	0,976	0,957	0,966
NN		00:02:02	00:00:17	Stemming	0,963	0,979	0,963	0,971
SVM		00:00:32	00:00:30		0,963	0,972	0,971	0,971
Forest		00:00:26	00:00:28		0,959	0,982	0,954	0,968
Bayes		00:00:19	00:00:20		0,957	0,984	0,949	0,966
NN		00:02:19	00:00:19	Typo	0,966	0,977	0,971	0,974
SVM		00:00:40	00:00:20		0,967	0,972	0,977	0,974
Forest		00:00:34	00:00:22		0,964	0,971	0,974	0,972
Bayes		00:00:19	00:00:16		0,975	0,983	0,978	0,980
NN		00:02:17	00:00:23	Compound	0,983	0,986	0,987	0,987
SVM		00:00:43	00:00:26		0,981	0,983	0,987	0,985
Forest		00:00:32	00:00:21		0,983	0,981	0,992	0,987
Bayes		00:00:22	00:00:16		0,984	0,989	0,986	0,987
NN	Med	00:04:08	00:00:24	None	0,966	0,984	0,963	0,973
SVM		00:00:49	00:00:37		0,964	0,984	0,961	0,972
Forest		00:00:55	00:00:30		0,962	0,989	0,952	0,970
Bayes		00:00:21	00:00:21		0,961	0,985	0,955	0,970
NN		00:04:20	00:00:27	Detectors	0,970	0,985	0,968	0,976
SVM		00:00:57	00:00:30		0,961	0,983	0,957	0,970
Forest		00:00:55	00:00:28		0,958	0,987	0,948	0,967
Bayes		00:00:29	00:00:33		0,962	0,984	0,958	0,971
NN		00:04:12	00:00:24	Stemming	0,972	0,984	0,973	0,978
SVM		00:00:50	00:00:26		0,971	0,983	0,972	0,978
Forest		00:00:50	00:00:27		0,965	0,992	0,954	0,973
Bayes		00:00:24	00:00:32		0,970	0,986	0,968	0,977
NN		00:04:47	00:00:26	Typo	0,980	0,982	0,987	0,985
SVM		00:01:16	00:00:30		0,974	0,983	0,977	0,980
Forest		00:01:04	00:00:24		0,979	0,988	0,980	0,984
Bayes		00:00:27	00:00:28		0,977	0,983	0,981	0,982
NN		00:04:34	00:00:31	Compound	0,989	0,991	0,991	0,991
SVM		00:01:20	00:00:33		0,989	0,989	0,993	0,991
Forest		00:01:09	00:00:30		0,988	0,990	0,992	0,991
Bayes		00:00:29	00:00:31		0,986	0,990	0,988	0,989
NN	Large	00:42:41	00:01:00	None	0,987	0,993	0,983	0,988
SVM		00:07:40	00:00:37		0,977	0,985	0,973	0,979
Forest		00:11:29	00:00:36		0,979	0,987	0,974	0,980
Bayes		00:01:30	00:00:23		0,976	0,986	0,969	0,978
NN		00:41:54	00:00:57	Detectors	0,988	0,994	0,984	0,989
SVM		00:07:57	00:00:39		0,977	0,984	0,973	0,979
Forest		00:11:23	00:00:44		0,975	0,983	0,971	0,977
Bayes		00:01:39	00:00:27		0,975	0,986	0,969	0,978
NN		00:42:58	00:01:06	Stemming	0,987	0,992	0,985	0,989
SVM		00:07:32	00:00:33		0,977	0,985	0,974	0,979
Forest		00:11:17	00:00:45		0,978	0,984	0,976	0,980
Bayes		00:01:43	00:00:26		0,976	0,987	0,969	0,978
NN		01:01:27	00:01:20	Typo	0,994	0,996	0,992	0,994
SVM		00:22:04	00:00:43		0,989	0,992	0,988	0,990
Forest		00:14:58	00:00:46		0,991	0,997	0,987	0,992
Bayes		00:03:01	00:00:31		0,987	0,993	0,984	0,989
NN		01:05:06	00:05:24	Compound	0,995	0,997	0,994	0,996
SVM		00:23:47	00:02:13		0,991	0,993	0,991	0,992
Forest		00:17:45	00:01:41		0,994	0,996	0,992	0,994
Bayes		00:04:16	00:03:31		0,991	0,994	0,989	0,992

Table 8.13: Results of Azure ML for class Maintenance

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
NN	Small	00:02:10	00:00:20	None	0,950	0,977	0,947	0,962
SVM		00:00:33	00:00:22		0,948	0,972	0,948	0,960
Forest		00:00:34	00:00:26		0,948	0,951	0,971	0,961
Bayes		00:00:17	00:00:21		0,945	0,981	0,935	0,957
NN		00:02:06	00:00:28	Detectors	0,953	0,980	0,948	0,964
SVM		00:00:33	00:00:28		0,945	0,965	0,952	0,958
Forest		00:00:33	00:00:24		0,946	0,952	0,967	0,959
Bayes		00:00:17	00:00:25		0,945	0,981	0,935	0,957
NN		00:02:10	00:00:23	Stemming	0,952	0,975	0,952	0,963
SVM		00:00:28	00:00:29		0,956	0,977	0,956	0,966
Forest		00:00:29	00:00:29		0,950	0,968	0,956	0,962
Bayes		00:00:21	00:00:14		0,951	0,984	0,941	0,962
NN		00:02:15	00:00:19	Typo	0,967	0,971	0,979	0,975
SVM		00:00:50	00:00:28		0,968	0,974	0,977	0,976
Forest		00:00:37	00:00:23		0,969	0,980	0,973	0,976
Bayes		00:00:23	00:00:18		0,970	0,983	0,971	0,977
NN		00:02:32	00:00:19	Compound	0,978	0,991	0,976	0,983
SVM		00:00:40	00:00:25		0,974	0,988	0,973	0,980
Forest		00:00:41	00:00:29		0,971	0,982	0,974	0,978
Bayes		00:00:21	00:00:18		0,977	0,994	0,971	0,982
NN	Med	00:04:05	00:00:22	None	0,974	0,978	0,958	0,968
SVM		00:00:48	00:00:29		0,972	0,971	0,962	0,966
Forest		00:01:01	00:00:24		0,953	0,943	0,945	0,944
Bayes		00:00:24	00:00:23		0,972	0,976	0,956	0,966
NN		00:03:59	00:00:28	Detectors	0,974	0,975	0,962	0,968
SVM		00:00:54	00:00:25		0,972	0,970	0,962	0,966
Forest		00:01:01	00:00:25		0,958	0,954	0,944	0,949
Bayes		00:00:25	00:00:21		0,971	0,974	0,956	0,965
NN		00:04:22	00:00:27	Stemming	0,971	0,969	0,961	0,965
SVM		00:00:56	00:00:20		0,965	0,954	0,962	0,958
Forest		00:01:00	00:00:25		0,955	0,958	0,933	0,946
Bayes		00:00:21	00:00:23		0,972	0,979	0,954	0,966
NN		00:04:48	00:00:24	Typo	0,980	0,974	0,977	0,976
SVM		00:01:09	00:00:17		0,973	0,968	0,968	0,968
Forest		00:01:16	00:00:27		0,970	0,978	0,949	0,963
Bayes		00:00:24	00:00:24		0,981	0,985	0,969	0,977
NN		00:04:43	00:00:22	Compound	0,988	0,989	0,982	0,986
SVM		00:01:35	00:00:34		0,986	0,980	0,987	0,983
Forest		00:01:25	00:00:26		0,983	0,993	0,967	0,979
Bayes		00:00:30	00:00:26		0,987	0,990	0,977	0,984
NN	Large	00:43:29	00:00:57	None	0,996	0,964	0,970	0,967
SVM		00:07:04	00:00:35		0,991	0,939	0,910	0,924
Forest		00:14:07	00:00:28		0,991	0,973	0,883	0,926
Bayes		00:01:41	00:00:18		0,992	0,949	0,916	0,932
NN		00:43:55	00:00:59	Detectors	0,996	0,969	0,967	0,968
SVM		00:08:27	00:00:37		0,991	0,939	0,911	0,925
Forest		00:14:30	00:00:32		0,991	0,980	0,882	0,928
Bayes		00:01:33	00:00:23		0,992	0,952	0,916	0,934
NN		00:43:14	00:01:00	Stemming	0,997	0,972	0,973	0,973
SVM		00:08:15	00:00:32		0,993	0,955	0,930	0,942
Forest		00:12:49	00:00:36		0,993	0,978	0,919	0,947
Bayes		00:01:36	00:00:21		0,993	0,957	0,930	0,943
NN		01:13:57	00:01:51	Typo	0,996	0,957	0,982	0,969
SVM		00:23:57	00:00:43		0,991	0,956	0,894	0,924
Forest		00:18:16	00:00:41		0,993	0,986	0,907	0,945
Bayes		00:03:11	00:00:26		0,990	0,959	0,879	0,917
NN		01:11:57	00:06:09	Compound	0,999	0,984	0,998	0,991
SVM		00:26:29	00:02:46		0,996	0,979	0,962	0,970
Forest		00:18:40	00:01:17		0,996	0,989	0,944	0,966
Bayes		00:04:30	00:03:39		0,996	0,983	0,958	0,970

Table 8.14: Results of Azure ML for class Serpentine belt

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
NN	Small	00:02:21	0:00:26	None	0,963	0,980	0,964	0,972
SVM		00:00:31	0:00:27		0,947	0,969	0,950	0,959
Forest		00:00:35	0:00:29		0,942	0,966	0,945	0,956
Bayes		00:00:25	0:00:21		0,941	0,960	0,950	0,955
NN		00:02:17	0:00:27	Detectors	0,970	0,979	0,976	0,977
SVM		00:00:37	0:00:24		0,967	0,982	0,968	0,975
Forest		00:00:38	0:00:29		0,956	0,972	0,961	0,966
Bayes		00:00:19	0:00:25		0,953	0,967	0,962	0,964
NN		00:02:15	0:00:21	Stemming	0,968	0,974	0,977	0,976
SVM		00:00:34	0:00:27		0,972	0,983	0,974	0,979
Forest		00:00:36	0:00:21		0,957	0,983	0,952	0,967
Bayes		00:00:19	0:00:15		0,955	0,971	0,961	0,966
NN		00:02:47	0:00:21	Typo	0,969	0,979	0,974	0,976
SVM		00:00:54	0:00:29		0,962	0,980	0,962	0,971
Forest		00:00:45	0:00:30		0,958	0,981	0,955	0,968
Bayes		00:00:25	0:00:19		0,966	0,982	0,967	0,974
NN		00:02:50	0:00:20	Compound	0,975	0,988	0,974	0,981
SVM		00:00:59	0:00:28		0,975	0,989	0,973	0,981
Forest		00:00:49	0:00:33		0,973	0,986	0,973	0,979
Bayes		00:00:23	0:00:21		0,972	0,988	0,970	0,979
NN	Med	00:04:14	0:00:31	None	0,950	0,956	0,968	0,962
SVM		00:00:56	0:00:32		0,952	0,962	0,966	0,964
Forest		00:01:09	0:00:24		0,966	0,971	0,977	0,974
Bayes		00:00:26	0:00:20		0,951	0,961	0,965	0,963
NN		00:04:20	0:00:29	Detectors	0,964	0,968	0,977	0,972
SVM		00:01:04	0:00:27		0,966	0,969	0,979	0,974
Forest		00:00:59	0:00:26		0,965	0,974	0,972	0,973
Bayes		00:00:27	0:00:27		0,961	0,968	0,973	0,971
NN		00:04:12	0:00:31	Stemming	0,963	0,970	0,974	0,972
SVM		00:01:00	0:00:27		0,964	0,970	0,975	0,972
Forest		00:01:00	0:00:29		0,967	0,974	0,975	0,975
Bayes		00:00:25	0:00:22		0,959	0,967	0,970	0,969
NN		00:05:58	0:00:30	Typo	0,970	0,976	0,979	0,977
SVM		00:02:03	0:00:22		0,969	0,977	0,976	0,976
Forest		00:01:25	0:00:28		0,971	0,979	0,977	0,978
Bayes		00:00:27	0:00:30		0,968	0,974	0,977	0,975
NN		00:05:32	0:00:26	Compound	0,978	0,989	0,977	0,983
SVM		00:02:05	0:00:32		0,979	0,983	0,985	0,984
Forest		00:01:23	0:00:29		0,977	0,982	0,983	0,983
Bayes		00:00:33	0:00:35		0,974	0,983	0,977	0,980
NN	Large	00:42:35	0:01:08	None	0,984	0,981	0,980	0,981
SVM		00:08:22	0:00:29		0,971	0,970	0,962	0,966
Forest		00:14:00	0:00:31		0,975	0,981	0,960	0,970
Bayes		00:01:57	0:00:21		0,969	0,973	0,955	0,964
NN		00:46:08	0:01:07	Detectors	0,987	0,984	0,985	0,985
SVM		00:08:39	0:00:32		0,979	0,980	0,972	0,976
Forest		00:13:29	0:00:36		0,981	0,984	0,971	0,977
Bayes		00:01:48	0:00:22		0,977	0,980	0,967	0,973
NN		00:44:44	0:01:05	Stemming	0,988	0,989	0,983	0,986
SVM		00:09:28	0:00:35		0,980	0,980	0,973	0,976
Forest		00:13:27	0:00:39		0,981	0,985	0,970	0,978
Bayes		00:01:48	0:00:23		0,977	0,980	0,966	0,973
NN		01:14:45	0:01:51	Typo	0,990	0,988	0,989	0,988
SVM		00:31:40	0:00:56		0,981	0,982	0,972	0,977
Forest		00:19:23	0:00:43		0,985	0,992	0,974	0,983
Bayes		00:03:58	0:00:44		0,978	0,981	0,967	0,974
NN		01:17:39	0:01:42	Compound	0,993	0,991	0,992	0,991
SVM		00:37:02	0:01:16		0,985	0,983	0,981	0,982
Forest		00:19:20	0:01:09		0,989	0,993	0,981	0,987
Bayes		00:04:37	0:03:10		0,982	0,983	0,975	0,979

Table 8.15: Results of Azure ML for class Tyres

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
NN	Small	0:02:04	0:00:20	None	0,993	0,992	0,997	0,995
SVM		0:00:32	0:00:25		0,990	0,992	0,992	0,992
Forest		0:00:30	0:00:25		0,988	0,989	0,992	0,991
Bayes		0:00:21	0:00:23		0,993	0,994	0,995	0,995
NN		0:02:04	0:00:26	Detectors	0,994	0,994	0,997	0,995
SVM		0:00:36	0:00:22		0,993	0,992	0,997	0,995
Forest		0:00:25	0:00:26		0,988	0,989	0,992	0,991
Bayes		0:00:27	0:00:20		0,993	0,994	0,995	0,995
NN		0:02:10	0:00:24	Stemming	0,991	0,992	0,994	0,993
SVM		0:00:30	0:00:31		0,987	0,991	0,989	0,990
Forest		0:00:21	0:00:33		0,990	0,989	0,995	0,992
Bayes		0:00:21	0:00:21		0,983	0,989	0,985	0,987
NN		0:02:30	0:00:22	Typo	0,996	0,998	0,995	0,997
SVM		0:00:48	0:00:28		0,995	0,997	0,995	0,996
Forest		0:00:30	0:00:26		0,991	0,994	0,992	0,993
Bayes		0:00:22	0:00:23		0,989	0,995	0,988	0,992
NN		0:02:33	0:00:27	Compound	0,997	1,000	0,995	0,998
SVM		0:00:46	0:00:20		0,995	0,997	0,995	0,996
Forest		0:00:37	0:00:27		0,994	0,997	0,994	0,995
Bayes		0:00:23	0:00:24		0,992	0,995	0,992	0,994
NN	Med	0:04:09	0:00:27	None	0,996	0,993	0,999	0,996
SVM		0:00:52	0:00:24		0,995	0,990	0,999	0,994
Forest		0:00:55	0:00:27		0,995	0,990	1,000	0,995
Bayes		0:00:27	0:00:19		0,994	0,995	0,992	0,993
NN		0:04:34	0:00:28	Detectors	0,997	0,994	0,999	0,996
SVM		0:00:56	0:00:23		0,996	0,993	0,999	0,996
Forest		0:00:51	0:00:25		0,995	0,990	1,000	0,995
Bayes		0:00:25	0:00:23		0,994	0,995	0,992	0,993
NN		0:04:09	0:00:27	Stemming	0,995	0,990	0,999	0,994
SVM		0:00:58	0:00:24		0,995	0,990	1,000	0,995
Forest		0:00:55	0:00:25		0,994	0,987	1,000	0,993
Bayes		0:00:25	0:00:18		0,993	0,992	0,993	0,992
NN		0:04:32	0:00:29	Typo	0,998	0,997	0,999	0,998
SVM		0:01:14	0:00:32		0,998	0,997	0,999	0,998
Forest		0:00:55	0:00:26		0,998	0,996	1,000	0,998
Bayes		0:00:25	0:00:24		0,995	0,995	0,995	0,995
NN		0:04:59	0:00:25	Compound	0,999	0,997	1,000	0,998
SVM		0:01:11	0:00:29		0,998	0,997	0,999	0,998
Forest		0:01:11	0:00:29		0,998	0,997	0,999	0,998
Bayes		0:01:03	0:00:30		0,998	0,997	0,999	0,998
NN	Large	0:00:26	0:00:30	None	0,996	0,997	0,994	0,995
SVM		0:42:37	0:01:06		0,998	0,978	0,991	0,985
Forest		0:07:39	0:00:34		0,997	0,978	0,987	0,983
Bayes		0:11:01	0:00:36		0,998	0,974	0,997	0,985
NN		0:01:38	0:00:22	Detectors	0,998	0,985	0,989	0,987
SVM		0:44:42	0:01:09		0,998	0,977	0,991	0,984
Forest		0:08:36	0:00:34		0,997	0,979	0,986	0,983
Bayes		0:11:21	0:00:26		0,998	0,974	0,997	0,985
NN		0:01:44	0:00:24	Stemming	0,998	0,984	0,989	0,987
SVM		0:43:38	0:01:04		0,997	0,977	0,987	0,982
Forest		0:09:14	0:00:31		0,997	0,977	0,985	0,981
Bayes		0:11:34	0:00:32		0,998	0,977	0,995	0,986
NN		0:56:37	0:01:24	Typo	0,999	0,990	0,995	0,992
SVM		0:21:40	0:00:40		0,998	0,985	0,995	0,990
Forest		0:10:47	0:00:30		0,999	0,988	0,998	0,993
Bayes		0:03:00	0:00:38		0,998	0,986	0,992	0,989
NN		1:01:32	0:02:04	Compound	0,999	0,991	0,996	0,993
SVM		0:23:32	0:02:02		0,999	0,986	0,995	0,991
Forest		0:10:33	0:01:42		0,999	0,990	0,998	0,994
Bayes		0:03:28	0:01:14		0,999	0,986	0,995	0,990

Table 8.16: Results of Azure ML for class Battery

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
NN	Small	00:02:11	00:00:24	None	0,984	0,981	0,976	0,978
SVM		00:00:38	00:00:28		0,981	0,976	0,973	0,974
Forest		00:00:36	00:00:35		0,959	0,963	0,924	0,943
Bayes		00:00:17	00:00:24		0,981	0,986	0,962	0,974
NN		00:02:14	00:00:23	Detectors	0,984	0,984	0,973	0,978
SVM		00:00:46	00:00:29		0,984	0,976	0,981	0,978
Forest		00:00:38	00:00:23		0,963	0,972	0,927	0,949
Bayes		00:00:21	00:00:22		0,981	0,986	0,962	0,974
NN		00:02:08	00:00:24	Stemming	0,978	0,965	0,976	0,970
SVM		00:00:35	00:00:23		0,979	0,970	0,973	0,972
Forest		00:00:34	00:00:25		0,970	0,986	0,932	0,958
Bayes		00:00:19	00:00:23		0,985	0,994	0,965	0,979
NN		00:02:20	00:00:24	Typo	0,982	0,981	0,970	0,975
SVM		00:00:42	00:00:23		0,979	0,965	0,978	0,972
Forest		00:00:30	00:00:36		0,973	0,991	0,935	0,962
Bayes		00:00:15	00:00:27		0,980	0,986	0,959	0,973
NN		00:02:18	00:00:25	Compound	0,982	0,983	0,967	0,975
SVM		00:00:44	00:00:29		0,983	0,978	0,976	0,977
Forest		00:00:43	00:00:21		0,985	0,992	0,967	0,979
Bayes		00:00:22	00:00:22		0,985	0,986	0,973	0,980
NN	Med	00:04:14	00:00:25	None	0,982	0,956	0,968	0,962
SVM		00:00:52	00:00:33		0,978	0,942	0,966	0,954
Forest		00:01:02	00:00:23		0,976	0,959	0,938	0,948
Bayes		00:00:26	00:00:24		0,986	0,985	0,953	0,969
NN		00:04:16	00:00:26	Detectors	0,983	0,960	0,968	0,964
SVM		00:00:55	00:00:30		0,980	0,948	0,966	0,957
Forest		00:01:02	00:00:32		0,971	0,972	0,902	0,936
Bayes		00:00:30	00:00:28		0,985	0,982	0,953	0,968
NN		00:04:18	00:00:25	Stemming	0,988	0,966	0,981	0,974
SVM		00:00:54	00:00:26		0,989	0,971	0,983	0,977
Forest		00:01:05	00:00:25		0,971	0,970	0,902	0,935
Bayes		00:00:25	00:00:19		0,984	0,982	0,947	0,964
NN		00:04:51	00:00:32	Typo	0,988	0,972	0,977	0,975
SVM		00:01:15	00:00:34		0,985	0,953	0,985	0,969
Forest		00:01:22	00:00:33		0,981	0,982	0,936	0,959
Bayes		00:00:27	00:00:24		0,983	0,974	0,951	0,962
NN		00:04:58	00:00:32	Compound	0,994	0,987	0,985	0,986
SVM		00:01:20	00:00:41		0,991	0,971	0,991	0,981
Forest		00:01:17	00:00:23		0,992	0,987	0,977	0,982
Bayes		00:00:27	00:00:25		0,994	0,991	0,981	0,986
NN	Large	00:41:52	00:01:01	None	0,998	0,949	0,960	0,954
SVM		00:07:28	00:00:33		0,996	0,942	0,892	0,916
Forest		00:12:51	00:00:27		0,996	0,976	0,883	0,927
Bayes		00:01:46	00:00:25		0,997	0,960	0,914	0,936
NN		00:44:15	00:01:01	Detectors	0,998	0,946	0,965	0,955
SVM		00:07:37	00:00:33		0,996	0,942	0,894	0,917
Forest		00:12:55	00:00:33		0,997	0,976	0,897	0,935
Bayes		00:01:35	00:00:24		0,997	0,956	0,916	0,935
NN		00:43:10	00:01:03	Stemming	0,996	0,916	0,958	0,936
SVM		00:07:40	00:00:34		0,996	0,938	0,908	0,923
Forest		00:12:58	00:00:32		0,996	0,962	0,872	0,914
Bayes		00:01:38	00:00:27		0,996	0,954	0,912	0,932
NN		01:07:31	00:00:45	Typo	0,998	0,961	0,960	0,961
SVM		00:23:09	00:00:47		0,996	0,939	0,899	0,918
Forest		00:17:04	00:00:38		0,997	0,978	0,912	0,944
Bayes		00:03:23	00:00:28		0,996	0,962	0,873	0,915
NN		01:12:33	00:01:05	Compound	0,998	0,960	0,971	0,965
SVM		00:25:03	00:02:07		0,997	0,961	0,941	0,951
Forest		00:17:22	00:01:20		0,998	0,977	0,954	0,966
Bayes		00:04:28	00:01:07		0,996	0,974	0,890	0,930

Table 8.17: Results of Azure ML for class V-belt

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
NN	Small	00:02:08	00:00:20	None	0,944	0,934	0,985	0,959
SVM		00:00:36	00:00:29		0,943	0,958	0,956	0,957
Forest		00:00:42	00:00:23		0,949	0,972	0,950	0,961
Bayes		00:00:20	00:00:19		0,947	0,966	0,953	0,960
NN		00:02:14	00:00:23	Detectors	0,946	0,937	0,985	0,960
SVM		00:00:33	00:00:25		0,944	0,958	0,958	0,958
Forest		00:00:33	00:00:29		0,948	0,966	0,955	0,960
Bayes		00:00:19	00:00:23		0,947	0,966	0,953	0,960
NN		00:02:01	00:00:29	Stemming	0,954	0,974	0,956	0,965
SVM		00:00:27	00:00:31		0,956	0,977	0,956	0,966
Forest		00:00:36	00:00:27		0,960	0,981	0,958	0,969
Bayes		00:00:22	00:00:17		0,963	0,986	0,958	0,972
NN		00:02:26	00:00:29	Typo	0,974	0,982	0,979	0,980
SVM		00:00:48	00:00:27		0,963	0,976	0,968	0,972
Forest		00:00:46	00:00:28		0,959	0,968	0,970	0,969
Bayes		00:00:23	00:00:23		0,973	0,986	0,973	0,979
NN		00:02:23	00:00:23	Compound	0,982	0,981	0,992	0,986
SVM		00:00:46	00:00:30		0,977	0,978	0,988	0,983
Forest		00:00:43	00:00:25		0,970	0,973	0,982	0,977
Bayes		00:00:25	00:00:25		0,982	0,989	0,983	0,986
NN	Med	00:04:05	00:00:27	None	0,961	0,979	0,961	0,970
SVM		00:00:48	00:00:37		0,958	0,970	0,966	0,968
Forest		00:00:59	00:00:32		0,954	0,978	0,951	0,964
Bayes		00:00:26	00:00:27		0,957	0,974	0,960	0,967
NN		00:04:15	00:00:35	Detectors	0,960	0,980	0,959	0,969
SVM		00:01:01	00:00:25		0,959	0,972	0,964	0,968
Forest		00:00:56	00:00:37		0,954	0,977	0,952	0,965
Bayes		00:00:30	00:00:24		0,957	0,975	0,960	0,967
NN		00:04:25	00:00:20	Stemming	0,968	0,987	0,964	0,975
SVM		00:01:02	00:00:27		0,962	0,981	0,961	0,971
Forest		00:00:59	00:00:31		0,963	0,981	0,961	0,971
Bayes		00:00:27	00:00:23		0,960	0,978	0,961	0,969
NN		00:04:35	00:00:22	Typo	0,972	0,987	0,970	0,979
SVM		00:01:23	00:00:33		0,968	0,977	0,973	0,975
Forest		00:01:07	00:00:30		0,965	0,973	0,973	0,973
Bayes		00:00:28	00:00:24		0,971	0,980	0,975	0,978
NN		00:04:41	00:00:27	Compound	0,981	0,986	0,985	0,985
SVM		00:01:25	00:00:25		0,981	0,984	0,986	0,985
Forest		00:01:17	00:00:30		0,973	0,977	0,981	0,979
Bayes		00:00:32	00:00:32		0,978	0,984	0,983	0,983
NN	Large	00:44:49	00:01:02	None	0,994	0,985	0,981	0,983
SVM		00:07:47	00:00:32		0,987	0,971	0,959	0,965
Forest		00:13:46	00:00:43		0,986	0,983	0,942	0,962
Bayes		00:01:39	00:00:23		0,987	0,971	0,957	0,964
NN		00:47:07	00:01:11	Detectors	0,993	0,983	0,981	0,982
SVM		00:08:32	00:00:33		0,987	0,972	0,957	0,964
Forest		00:13:39	00:00:45		0,986	0,982	0,944	0,963
Bayes		00:01:42	00:00:24		0,987	0,973	0,956	0,964
NN		00:45:18	00:01:06	Stemming	0,993	0,983	0,976	0,980
SVM		00:08:01	00:00:34		0,987	0,974	0,955	0,964
Forest		00:13:07	00:00:37		0,986	0,981	0,942	0,961
Bayes		00:01:43	00:00:25		0,986	0,977	0,949	0,962
NN		01:06:13	00:00:41	Typo	0,996	0,989	0,990	0,990
SVM		00:25:05	00:00:55		0,990	0,974	0,971	0,972
Forest		00:16:54	00:00:49		0,993	0,988	0,972	0,980
Bayes		00:03:17	00:00:28		0,990	0,977	0,968	0,973
NN		01:11:44	00:01:32	Compound	0,997	0,987	0,995	0,991
SVM		00:27:44	00:01:58		0,993	0,978	0,983	0,981
Forest		00:18:20	00:01:52		0,994	0,991	0,978	0,984
Bayes		00:03:47	00:00:56		0,992	0,980	0,979	0,979

Table 8.18: Results of Azure ML for class Brakes

Model	**Size**	**Train**	**Eval**	**Options**	**Acc**	**Prec**	**Rec**	F_1
NN	Small	00:02:05	00:00:18	None	0,976	0,989	0,974	0,982
SVM		00:00:34	00:00:24		0,971	0,980	0,976	0,978
Forest		00:00:33	00:00:32		0,967	0,985	0,965	0,975
Bayes		00:00:23	00:00:25		0,957	0,975	0,959	0,967
NN		00:02:04	00:00:26	Detectors	0,976	0,991	0,973	0,982
SVM		00:00:36	00:00:33		0,970	0,983	0,971	0,977
Forest		00:00:37	00:00:28		0,965	0,984	0,962	0,973
Bayes		00:00:19	00:00:30		0,955	0,974	0,958	0,966
NN		00:02:08	00:00:18	Stemming	0,971	0,980	0,976	0,978
SVM		00:00:38	00:00:23		0,965	0,971	0,976	0,974
Forest		00:00:33	00:00:24		0,965	0,983	0,964	0,973
Bayes		00:00:14	00:00:30		0,963	0,983	0,961	0,972
NN		00:02:20	00:00:27	Typo	0,973	0,979	0,980	0,980
SVM		00:00:35	00:00:20		0,973	0,983	0,976	0,979
Forest		00:00:30	00:00:29		0,972	0,992	0,965	0,978
Bayes		00:00:10	00:00:27		0,963	0,977	0,967	0,972
NN		00:02:14	00:00:24	Compound	0,985	0,991	0,986	0,989
SVM		00:00:41	00:00:29		0,985	0,989	0,988	0,989
Forest		00:00:36	00:00:27		0,983	0,988	0,986	0,987
Bayes		00:00:25	00:00:17		0,978	0,985	0,982	0,983
NN	Med	00:04:25	00:00:23	None	0,988	0,978	0,993	0,985
SVM		00:00:51	00:00:34		0,987	0,981	0,988	0,985
Forest		00:01:04	00:00:23		0,986	0,982	0,984	0,983
Bayes		00:00:26	00:00:21		0,985	0,977	0,987	0,982
NN		00:04:23	00:00:22	Detectors	0,987	0,977	0,993	0,985
SVM		00:00:55	00:00:22		0,988	0,983	0,988	0,985
Forest		00:01:03	00:00:28		0,985	0,981	0,984	0,982
Bayes		00:00:21	00:00:20		0,985	0,978	0,987	0,982
NN		00:03:58	00:00:18	Stemming	0,983	0,979	0,981	0,980
SVM		00:00:47	00:00:24		0,982	0,980	0,978	0,979
Forest		00:01:00	00:00:22		0,984	0,985	0,977	0,981
Bayes		00:00:22	00:00:25		0,981	0,973	0,981	0,977
NN		00:04:46	00:00:34	Typo	0,981	0,974	0,980	0,977
SVM		00:01:09	00:00:29		0,980	0,973	0,979	0,976
Forest		00:01:06	00:00:25		0,983	0,988	0,972	0,980
Bayes		00:00:29	00:00:22		0,981	0,982	0,972	0,977
NN		00:04:36	00:00:39	Compound	0,993	0,991	0,992	0,991
SVM		00:01:12	00:00:25		0,991	0,986	0,993	0,989
Forest		00:01:11	00:00:28		0,990	0,982	0,995	0,988
Bayes		00:00:28	00:00:26		0,992	0,991	0,989	0,990
NN	Large	00:42:33	00:00:51	None	0,997	0,972	0,982	0,977
SVM		00:08:04	00:00:36		0,995	0,968	0,956	0,962
Forest		00:13:51	00:00:28		0,996	0,971	0,961	0,966
Bayes		00:01:48	00:00:21		0,995	0,968	0,953	0,960
NN		00:44:24	00:01:05	Detectors	0,996	0,973	0,970	0,972
SVM		00:08:30	00:00:28		0,995	0,969	0,956	0,963
Forest		00:13:36	00:00:35		0,996	0,975	0,961	0,968
Bayes		00:01:41	00:00:22		0,995	0,970	0,952	0,961
NN		00:44:23	00:00:58	Stemming	0,997	0,976	0,974	0,975
SVM		00:08:45	00:00:32		0,995	0,973	0,957	0,965
Forest		00:13:43	00:00:34		0,996	0,980	0,954	0,967
Bayes		00:01:45	00:00:26		0,995	0,964	0,952	0,958
NN		01:00:51	00:00:58	Typo	0,997	0,987	0,969	0,978
SVM		00:24:38	00:00:48		0,996	0,983	0,953	0,968
Forest		00:15:16	00:00:36		0,997	0,995	0,956	0,975
Bayes		00:03:17	00:00:29		0,996	0,992	0,949	0,970
NN		01:01:47	00:06:21	Compound	0,998	0,982	0,989	0,986
SVM		00:25:51	00:03:28		0,997	0,985	0,975	0,980
Forest		00:17:38	00:03:53		0,999	0,992	0,987	0,989
Bayes		00:04:24	00:03:39		0,998	0,991	0,973	0,982

Table 8.19: Results of Azure ML for class Air conditioning

Results of non-binary models

Model	**Overall Agreement**	**Kappa κ**
Naive Bayes	0.977	0.886
Neural Network	0.966	0.818
Support Vector Machine	0.978	0.891
Random Forest	0.974	0.870

Table 8.20: The results of the multi-class models