MASTER'S THESIS | LUND UNIVERSITY 2015

Machine-Learning Techniques for Customer Recommendations

Felix Glas

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2015-17

Machine-Learning Techniques for Customer Recommendations

(A Practical Study in Data-Driven Customer Prediction for Customer Relationship Management)

Felix Glas
felix.glas@gmail.com

June 9, 2015

Master's thesis work carried out at Lundalogik AB.

Supervisors: Pierre Nugues, Pierre.Nugues@cs.lth.se
Peter Wilhelmsson, peter.wilhelmsson@lundalogik.se

Examiner: Thore Husfeldt, Thore.Husfeldt@cs.lth.se

Abstract

Today, there is a demand for automated procedures for predicting future customers using recommendation engines in the customer relationship management market. There are already functions commonly available for finding "twins", *i.e.*, possible customers that are similar to existing customers, and for browsing through lists of customers partitioned into categories such as locations or lines of business.

Current recommendation engines are typically built using machine-learning algorithms. Thus, it is of interest to determine which machine-learning algorithms that are best suited for making a recommendation engine aimed at customer prediction possible. This thesis investigates the prerequisites for determining suitability, and perform an evaluation of various off-the-shelf machine-learning algorithms.

The supervised learner models are shown to have promise, as a direct method of identifying new potential customers. A classifier algorithm can be trained using a set that contains existing customers, and be applied on a large set of various companies, to classify suitable prospects, provided there is a sufficiently large number of existing customers.

Keywords: CRM, Recommendation Engine, Customer Prediction, Machine Learning, Classification, Clustering, Apriori, k-Nearest Neighbors, C4.5, Decision Tree, k-Means Clustering

Acknowledgements

To my supervisor at LTH, Pierre Nugues, for his solid knowledge, valued feedback, and support.

To my supervisor at Lundalogik, Peter Wilhelmsson, for his helpful suggestions and support.

To examiner Thore Husfeldt, for the helpful suggestions.

To all the employees at Lundalogik, for their friendly reception and help.

To my family and friends, for all their support.

Thank you all!

Contents

Chapter 1
Introduction

Today, there is a demand for automated procedures for predicting future customers using recommendation engines in the customer relationship management (CRM) market. Current prediction techniques are limited to single attribute filtering, and simple observations of equivalence. The quickly evolving clientele of sales-oriented businesses desire more advanced recommendation techniques for identifying new prospects.

In other industries, complex recommendation systems are already put to use, *e.g.*, by well-known providers of music and movie entertainment. The prevalence of *big data* in combination with *machine-learning* techniques are behind these high quality recommendations.

As more data becomes available in the CRM industry, new possibilities arise for deriving deep insights from the accumulated amount of information. Such learning techniques can benefit most users of CRM systems, enabling them to make accurate predictions on future customers.

This master's thesis concerns the realization of a recommendation engine for CRM. Current recommendation engines are typically built using machine-learning algorithms, hence, it is of interest to determine which machine-learning algorithms that are best suited for making possible a recommendation engine aimed at customer prediction. In this thesis we will investigate the prerequisites for determining suitability, and perform an evaluation of various off-the-shelf machine-learning algorithms.

The objective is to find a suitable algorithm with the following aspects in mind:

- Which criteria should be used to determine the suitability of an algorithm?
- Which algorithms are suitable for this type of problem?
- Which suitable algorithm have optimal performance?

This master's thesis project was carried out at Lundalogik AB.

1.1 CRM

Customer Relationship Management (CRM) is a system for managing the interaction between current and future customers to a company.

In a business world that is growing more and more competitive and where customer experience is becoming increasingly important, businesses need their products and services to be better aligned with their customer's needs. To address this, businesses have increased focus on their customers by examining the customer's perspective and deliberately managing customer information and relationships more thoughtfully (Kostojohn et al., 2011). CRM can be seen as the vendor's reaction to a more demanding and less loyal customer base by collecting and refining information about individual customers and using it for finding new and more effective ways of communication (Peel and Gancarz, 2002).

In order to support this new focus on customers and customer management there has been an emergence of a new class of computerized tools and software. This software is aimed at utilizing the power and capacity of modern technology for complex tasks, such as statistical analysis and machine learning. Computers and software allow us to work with large amounts of customer data in real-time. This allows for the discovery of new valuable information, which would otherwise not have been available, and this information can be used for further improving customer relations.

1.2 Recommendation Engine

In today's expanding CRM market, there is a demand for automated procedures that can be used for customer prospecting. There are already functions commonly available for finding "twins", *i.e.*, possible customers that are similar to existing customers, and for browsing through lists of customers partitioned into categories such as locations or lines of business. In the near future, computer algorithms will enable systems to automatically suggest prospects that have a high potential for becoming profitable customers. Such systems need a recommendation engine for making predictions on which companies are relevant.

A recommendation engine can use data associated with *existing* customers to automatically produce new customer suggestions. To find new suggestions, data can be analyzed for similarities that characterize the existing customers. These similarities can then be used to find new customers that are similar to existing customers. The resemblance need not necessarily be precise. It can in fact sometimes be desirable to get a broad range of matches that extend somewhat outside the characteristic domain of existing customers.

The recommendation engine can potentially be based on a self-learning algorithm that uses existing customers as a training set. Data associated with the customers can then be used to identify new prospects in a large company database.

1.3 Machine Learning

Machine learning is a discipline that describe techniques used to make observations and predictions about data algorithmically.

Machine-learning algorithms can be divided in two main groups: *supervised learners*, used to train predictive models for classification tasks, and *unsupervised learners*, used to

train descriptive models for clustering tasks (see James et al., 2013, chap. 2).

1.3.1 Supervised learning

Supervised learning is the process of training a predictive model that can learn to determine a plausible value from a set of *known* target values. A **predictive model** learns to predict one value by using other values in the data set to model the relationship among the target feature (the feature being predicted) and the other features. The model is given clear instruction on what to learn and how to learn it and therefore the training of a predictive model is called *supervised* (Lantz, 2013).

One of the most common uses of supervised machine-learning tasks is predicting which category an example belongs to. This is known as **classification** and the model trained for this task is called a **classifier**.

Supervised learning can be summarized in the following steps:

1. Training - train the model with the labeled training set.
2. Validation (optional) - tune the parameters of the model.
3. Testing - test the performance of the model against the test set.
4. Application - apply the model to real-world data.

1.3.2 Unsupervised learning

As opposed to predictive models that predict a target feature, **descriptive models** give no special importance to any single feature. Because there is no target to learn, the process of training a descriptive model is called *unsupervised* (Lantz, 2013).

A descriptive model tries to summarize data by dividing it into homogeneous groups. This is known as **clustering** or **cluster analysis**. This can be used for segmentation discovery where groups that are generally similar in some way can be identified in the data set.

1.4 Data Requirements

All machine learning is based on analysis of existing data, and often, a learner will have improved performance when it has access to large quantities of data. More samples will make it easier for a model to make statistical assumptions about general characteristics in the data.

The data used in this project consists of business data originating from a large database of companies located in Sweden. This business data includes information about company locations, lines of business, number of employees, and various financial properties. See Table A.1 in Appendix A for a list of all attributes available in the company database. Some of the attributes have discrete values, such as: location and line of business, while most of the financial attributes have continuous ranges of values.

Only a selection of the available attributes were used when training the models. While most of the attributes are relevant for use, many of the discrete attributes exists as both

a “code” variant, and a plain text variant, where the coded attribute conveys the same information as the corresponding text variant, but translated into codes. Thus, it is only necessary to use one of the variants. Furthermore, a few attributes contain no values whatsoever, and are omitted.

Additionally a second database was kept, containing lists of existing customers for the individual companies in the company database. Using this information, it was possible to create a directed graph of all the companies, where the edges represent company-to-customer relationships.

All data used during this project was provided by Lundalogik AB.

1.5 Previous Work

Pazzani and Billsus (2007, chap. 10) describe different methods for recommending items using a *content-based filtering* approach. Common algorithms, such as k-nearest neighbors and decision trees are reviewed with respect to suitability for classification tasks. Results show that good recommendations can be given if the data contain enough information to distinguish desirable items from undesirable items.

Ungar and Foster (1998) explore the possibility of using common algorithms, such as k-means clustering, for *collaborative filtering*. Cluster analysis is performed on a data set consisting of movie and music history for individual users. It is indicated that clustering using k-means is somewhat problematic as the data is too sparse for creating relevant clusters with distinct characteristics, which suggests a dependence on suitable data for producing good recommendations.

A previous master's thesis project, carried out by Buö and Kjellander (2014), investigated the possibility of utilizing **data mining** for predicting *churn*[1], by exploiting the same set of data used in this thesis. The data mining procedure was performed using the well-known machine-learning algorithm C4.5. The conclusion from this project was that the churn could in fact be predicted from the data by some degree, which implies that the data is of sufficient quality for the extraction of new information using machine learning.

1.6 Summary of Contributions

This master's thesis begins with an introductory explanation of the essential concepts, such as CRM, recommendation engines, machine learning, data requirements, and other matters of relevance, in Chapter 1 (Introduction).

Chapter 2 (Approach), gives a detailed description of the evaluation methods used to analyze the properties of machine-learning models, such as various measures of performance, and methods for visualizing differences between results. Chapter 3 (Evaluation of Algorithms), contains evaluation results of the individual algorithms that were analyzed. In Chapter 4 (Discussion) the evaluation results are discussed.

Finally, in Chapter 5 (Conclusion), the thesis results are presented, and future work suggested.

[1] *Churn* rate refers to the rate by which customers or subscribers leave a supplier during a given time period.

Chapter 2
Approach

2.1 Finding Potential New Customers

The main goal of the evaluation is to determine a successful method for automatically identifying new prospects among the company data. As new prospects will be based on existing customers, the effort should be focused on finding a predictive model that can identify characteristic properties among the data trained upon, and that can use this information to predict new customers with similar properties.

2.2 Analysis of Existing Algorithms

As there alrady exist a large array of off-the-shelf machine-learning algorithms that are well documented, the evaluation will consist of measuring the performance of a selection of existing algorithms. The choice of algorithms must be based upon research of suitability for the problem at hand. The chosen algorithms must also be common and readily available from well-known providers of machine-learning services, such as PredictionIO or Apache Mahout.

2.3 Evaluation Method

The first requirement for comparing performance between different learner models is to determine a method of evaluation. This section will present a number of common statistics for measuring the performance of machine learners. These statistics will range from simple measures of model *accuracy* to measures of specific characteristics of the models as, for example, the proportion of relevant instances when dealing with information retrieval.

Performance in the terms of computational speed or memory consumption are considered second-rate as computational power is often not an issue with modern hardware.

More important is how well a learner is able to learn and successfully identify relevant instances. However, the running time must not become impractically long during execution of an algorithm when it is applied on large sets of data.

2.3.1 Confusion matrix

A confusion matrix is a table used to categorize predictions according to how they match the actual value in the data. One of the table's dimensions represents the possible categories of predicted values and the other dimension represent the actual values. A confusion matrix can be used to categorize the predictions of a model predicting any number of target values, but it is mostly used for binary predictions represented by the 2×2 confusion matrix.

The prediction outcomes of interest are known as the *positive* classes, while the other outcomes are known as *negative*. The relationship between positive class and negative class predictions can be represented by the 2×2 confusion matrix (Figure 2.1) depicting the four categories (see Lantz, 2013, chap. 10):

True positive (TP): Correctly classified as outcome of interest

True negative (TN): Correctly classified as outcome of *no* interest

False positive (FP): Incorrectly classified as outcome of interest

False negative (FN): Incorrectly classified as outcome of *no* interest

		True Class	
		Positive	Negative
Hypothesized Class	Positive	TP	FP
	Negative	FN	TN

Figure 2.1: Confusion Matrix.

2.3.2 Accuracy and error rate

There are many measures of performance in machine learning that have been developed for specific purposes. **Accuracy** is perhaps the simplest one, describing the success rate of a prediction. It is defined as the proportion of correct classifications in relation to all classifications made (see Lantz, 2013, chap. 10).

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

The terms TP, TN, FP and FN refer to the number of times the predictions fell into each of these categories.

The opposite of the success rate is the **error rate**, describing the proportion of incorrect classifications. It is defined as the number of incorrect classifications divided by the total number of classifications, or simply: $1 - accuracy$ (see Lantz, 2013, chap. 10).

$$error\ rate = \frac{FP + FN}{TP + TN + FP + FN} = 1 - accuracy.$$

Accuracy and error rate are simple measures of how well a model performs in general terms. However, it can be misleading when used alone. If the number of positive target values in the test set is small in relation to other values, say 10%, then a model that, *e.g.*, classifies all instances as negative will still have an accuracy of 0.9 as 90% of the instances are correctly classified as negatives.

2.3.3 Precision and recall

Precision and recall are two other measures of performance that are used primarily in the domain of information retrieval. Both measures are meant to give an indication of how interesting and relevant a model's results are.

Precision is defined as the proportion of positive classification that are truly positive (see Lantz, 2013, chap. 10). In other words, when a positive prediction is given, how often is it correct? A high precision might suggest that a model is trustworthy. To take an example of a *precise* model in the context of information retrieval, this would correspond to, *e.g.*, a search engine returning a high degree of related results. A search engine using an *imprecise* model would return mostly unrelated results.

$$precision = \frac{TP}{TP + FP}.$$

Recall is instead a measure of how complete the results are. It measures the proportion of positive examples that were correctly classified (see Lantz, 2013, chap. 10). A model with high recall will correctly classify a high portion of the positive instances as positive. That being said, there is no guarantee that there will not also be a lot of incorrectly classified positives. For example, a search engine using a model with *high* recall will return a large number of results. A search engine with *low* recall will return a low number of results.

$$recall = \frac{TP}{TP + FN}.$$

The two measures precision and recall are closely related and there exists an inherent trade-off between having a high value of either. It is easy to be precise by only classifying the most obvious instances, and conversely, it is easy for a model to achieve a high recall by being overly optimistic when classifying instances. However, it is difficult to build a model that has both high precision and high recall. It is often a balance between being conservative and overly aggressive in decision making (see Lantz, 2013, chap. 10).

Precision and recall can be combined into a single number known as the **F-measure**. The F-measure combines precision and recall using the harmonic mean. This measure can be valuable to determine an overall performance from the perspective of information retrieval. It is defined according to the following formula:

$$F\text{-}measure = \frac{2 \times precision \times recall}{recall + precision} = \frac{2 \times TP}{2 \times TP + FP + FN}.$$

2.3.4 Bias-Variance trade-off

The trade-off between precision and recall is a symptom of the general trade-off dilemma called the *bias-variance trade-off*. The bias-variance trade-off applies to all supervised learning tasks and represents two sources of error that prevent a learner to generalize beyond its training set (Geurts, 2002).

If a model is *too simple* with respect to the complexity of the Bayes model[1], there will always be an error due to the fact that the model is too simple to cover all instances of the training set. This lack of complexity of the model is called **bias** (Domingos, 2000). To illustrate, the regression problem described in Figure 2.2 shows that an overly simple model will inevitably fail to account for small variations in the training set. A model with high bias will make false assumptions about the data and can cause the learner to miss relevant relations between features and target features. High bias will generally tend to cause underfitting.

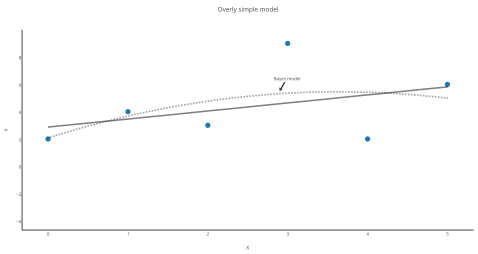

Figure 2.2: Overly simple model.

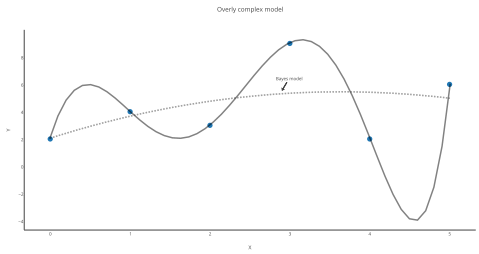

Figure 2.3: Overly complex model.

On the other hand, if a model is *overly complex* it might achieve a near perfect match on the training set. However, it will also learn the noise in the training set and will therefore not generalize well on different sets of data. If the model will achieve a perfect match on the training set, it will generally tend to overfit. Even if there is no noise, the model will have errors due to being overly complex. This exaggerated complexity in respect to the complexity of the training set is called **variance** (Domingos, 2000). The analogy in the regression problem can be seen in Figure 2.3. A model with high variance will be overly sensitive to small fluctuations in the training data and can cause modeling of random noise. High variance will generally tend to cause overfitting.

As both bias and variance depend on the complexity of the model, but in opposite direction, there must exist a trade-off effect between these sources of error. Due to bias, care must be taken not to use an overly simple model. And vice versa, care must be taken not

[1]The *Bayes model* is an ideal optimal classifier with perfect accuracy (Bayes and Price, 1763).

to use an overly complex model with respect to the complexity of the problem due to variance. The bias-variance trade-off applies to all supervised learning such as classification and regression. Apart from the generalization errors produced by bias and variance there is also an unavoidable error that is always present called the *irreducible error* caused by noise in the data.

2.3.5 Visualizing performance trade-offs

Visualizations are often helpful for human comprehension. They also provide a method for comparing machine-learning models side-by-side in a single diagram.

The **ROC** graph (Receiver Operating Characteristic) can be used to visualize the trade-off between the detection of true positives and false positives. This can be a good measure of the general efficiency of machine-learning models. The ROC graph is a two-dimensional space that is defined by showing the proportion of true positives (true positive rate) on the vertical axis, and showing the proportion of false positives (false positive rate) on the horizontal axis (see Lantz, 2013, chap. 10).

The **True Positive Rate** (TPR) is estimated by dividing the number of *true* positives (TP) by the total number of positives (T). TPR is also called *hit rate* or *recall* (see Fawcett, 2006, Classifier performance section).

$$TPR = \frac{TP}{TP + FN} = \frac{TP}{T}.$$

The **False Positive Rate** (FPR) is estimated by dividing the number of *false* positives (FP) by the total number of negatives (N). FPR is also called *fall-out* or *cost* (see Fawcett, 2006, Classifier performance section).

$$FPR = \frac{FP}{FP + TN} = \frac{FP}{N}.$$

A *discrete* classifier model, that outputs a target feature label (as opposed to a probabilistic classifier that outputs a probability), will produce a single measure of TPR and FPR corresponding to a point in the ROC graph. Depending on where a classifier's corresponding point is positioned in the graph, some conclusions can be drawn. Classifiers appearing on the left-hand side of the ROC graph, may be considered as *conservative* as they will only make positive classifications when there is strong evidence, hence they will generate few false positives. The downside with a conservative classifier is that it will also have a low true positive rate. Classifiers appearing on the upper right-hand side of the ROC graph may inversely be considered as *liberal* as they will make positive classifications with weak evidence and will classify a high proportion of the positives correctly. However, a liberal classifier will likely also have a high false positive rate. Inductively, a classifier appearing to the northwest of another classifier is better as its TPR is higher, its FPR is lower or both. See Figure 2.4 depicting the ROC graph. Performance on the left-hand side of the ROC graph is often more interesting as real-world problems are likely dominated by large numbers of negative instances. The point positioned at the uppermost left-hand side of the ROC graph at coordinates $(0, 1)$ represents a perfect classifier with perfect performance (see Fawcett, 2006, ROC space section).

Classifiers appearing along the diagonal $y = x$ represent models that randomly guesses a target feature. To move away from the diagonal, a classifier must be able to exploit some information in the data. Points positioned in the lower right triangle of the ROC graph will perform worse than random guessing and therefore this region is usually empty. However, if a classifier performs worse than random guessing it will exhibit deterministic behavior and its output can simply be negated to produce a point in the upper left triangle (see Fawcett, 2006, ROC space section).

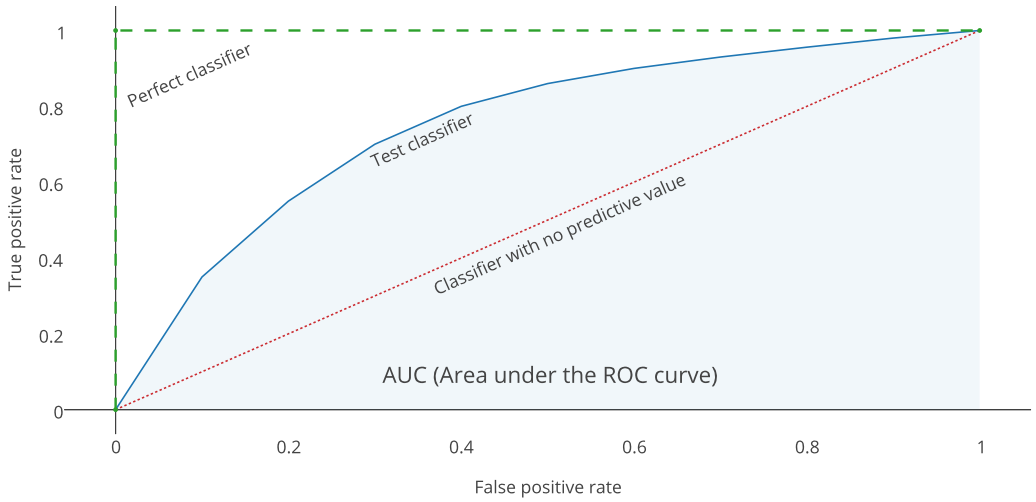

Figure 2.4: ROC curve and AUC.

A classifier that outputs a probability or a score representing the likeliness that an instance is considered positive is called a *probabilistic classifier*. Such a classifier can easily be used as a discrete classifier by comparing the output to a threshold. If the output is above the threshold the instance is considered positive. By varying the threshold, a varying degree of the instances in the test set will be classified as positives. It is therefore possible to produce multiple TPR and FPR value pairs on the same classifier model by using multiple threshold values. This can be used to create a **ROC curve**, by connecting the points produced. A discrete classifier can even be made into a probabilistic one by applying *ensemble learning* methods[2] and averaging the results. The ROC curve is useful as it can visualize if a model performs better in the conservative or liberal region of the graph. Further it can be revealed at which threshold the classifier has optimal accuracy (the point on the curve that has the smallest distance to the upper left corner has the best accuracy) (Fawcett, 2006).

[2] *Ensemble learning* is a method for using multiple classifiers concurrently on the same instance to be classified. The final classification can be achieved by averaging or voting on the results of the classifiers.

The **AUC** (Area Under the ROC Curve) is a statistic that can be used to depict general performance from a ROC curve. It treats the ROC graph as a two-dimensional square and measures the total area under the ROC curve. AUC will range from 0.5 (random guessing) to 1.0 (perfect classifier). There is also a convention using classes similar to academic letter grades for interpreting the AUC scores (see Lantz, 2013, chap. 10).

0.9–1.0 - A (outstanding)

0.8–0.9 - B (excellent/good)

0.7–0.8 - C (acceptable/fair)

0.6–0.7 - D (poor)

0.5–0.6 - F (no discrimination)

2.4 Implementation

Implementing code did inevitable cover a large portion of the algorithm evaluation process. To help reduce the effort put into implementation, existing software and libraries have been used, such as Weka, SciPy, NumPy and Armadillo. Weka is a desktop application that comes packaged with a collection of machine-learning algorithms that can be applied on custom data. SciPy, NumPu, and Armadillo are libraries oriented at statistical, mathematic and scientific computing for Python and C++, respectively.

Some of the algorithms were tested using different implementations, however, as all evaluated algorithms were simple to implement, custom implementations were created in C++ for reasons of flexibility, speed, control, freedom of parallelization, and last but not least --- for fun.

Languages used during implementation: Matlab, R, Python, Java and C++. Most of the data sets used during evaluation were structured as comma-separated-value (CSV) text files.

Chapter 3
Evaluation of Algorithms

The chosen algorithms for this evaluation are: **frequent set counting**, ***k*-nearest neighbors**, **C4.5**, and ***k*-means clustering**. These algorithms are all well-known off-the-shelf algorithms with readily available implementations in most programming languages. The first three algorithms are designed to solve classification problems, while the fourth is intended for performing cluster analysis.

The selected algorithms were chosen as they are well documented, easy to understand, and seem fitting for solving the problem at hand. They are also known for having performance that is competitive with more advanced algorithms, such as *Neural networks*, given suitable data.

This section contains descriptions of the implementations tested and evaluation measures for each algorithm, respectively. The evaluations were performed using a test machine equipped with a 2.6 GHz quad-core CPU using 8 GB DDR3 SDRAM.

3.1 Data Preparation

Before the algorithms were evaluated, the data needed some preparation. The goal of the preparation was to produce a diverse collection of high quality training sets and test sets for evaluation of the algorithms.

3.1.1 Labeling

All training sets used to train the classifiers need a way of distinguishing between desirable and undesirable outcomes. In order to define a notion of positiveness and negativeness among the instances in a training set, a distinction was made between customers and non-customers. The purpose of this distinction is to be able to train a classifier to identify all customers in a set of data. Hence, the notion of a positive instance was related to customers, and the notion of a negative instance was related to non-customers.

As the company database stores information about existing customers for individual records, it is possible to label all instances in a training set as positive, or negative, depending on the customer states of the instances, respectively. Hence, an additional attribute was added to all training sets and test sets, containing the value "positive" or "negative" which denotes each instance as either a customer, or a non-customer.

3.1.2 Continuous attributes

Continuous attributes contain numeric values which potentially range from negative infinity to positive infinity. This presents a problem as the number of attribute values that need to be accounted for by the models is possibly infinite, and most machine-learning models require a small number of attribute values (Chmielewski and Grzymala-Busse, 1996).

Another problem with attributes containing continuous ranges of values is that it is hard to make generalizations about the values, as many models generalize using equality. For example, given a range of numbers 1–100,000, it can be reasoned that the numbers 5,000 and 5,001 are similar, but to a machine-learning model using symbolic equality as the measure of similarity, this is not true. To the model, the numbers 5,000 and 5,001 are equally dissimilar as 1 and 100,000. In order for the model to be able to make generalization about the data, the range of numbers would need to be partitioned into discrete intervals: $\{[a_1,\ b_1],\ [a_2,\ b_2],\ \ldots,\ [a_n,\ b_n]\}$, where $b_i - a_i >= 1$, and $i = 1,\ 2,\ \ldots,\ n$. This process of partitioning continuous ranges into discrete values is called **discretization**.

In this project, two different methods for discretization was evaluated. The first method: *entropy-based partitioning*, is part of the C4.5. decision tree algorithm, as an extension, and the second method was invented during the course of this project as an improvement over the first method.

Entropy-based partitioning works by partitioning the range of continuous values in two partitions, by splitting the sorted range at a threshold value. The threshold value is found by performing an exhaustive search for the maximum *gain* score over the attribute values (**gain** is a measure of gained information, as described in section 3.4.1). In other words, the maximum threshold is found by iterating over the set of sorted values: $\{v_1,\ v_2,\ \ldots,\ v_n\}$, using v_i as the threshold in the ith iteration, and computing the summed gain for the the two partitions $\{v_1,\ v_2,\ \ldots,\ v_i\}$ and $\{v_{i+1},\ v_{i+2},\ \ldots,\ v_n\}$. The threshold that maximizes the summed gain values of the partitions is chosen as the final value.

The improved discretization method analyses the distribution of attribute values, and tries to identify multiple partitions of attribute values that are frequently occurring in the set. The method works by first iterating over all attribute values and counting individual occurrences, creating a distribution of values. As depicted in Figure 3.1, the attribute value distribution will form a jagged curve (most likely). The local maxima of the curve represent the most frequently occurring attribute values, which a classifier should be able to generalize from. When *e.g.*, two local maxima of near equal value happen to be located far from each other, it becomes hard to create a well-performing discretization by using a binary split, as only one of the maxima will be included. It would be better if the continuous range could be partitioned into multiple parts somehow.

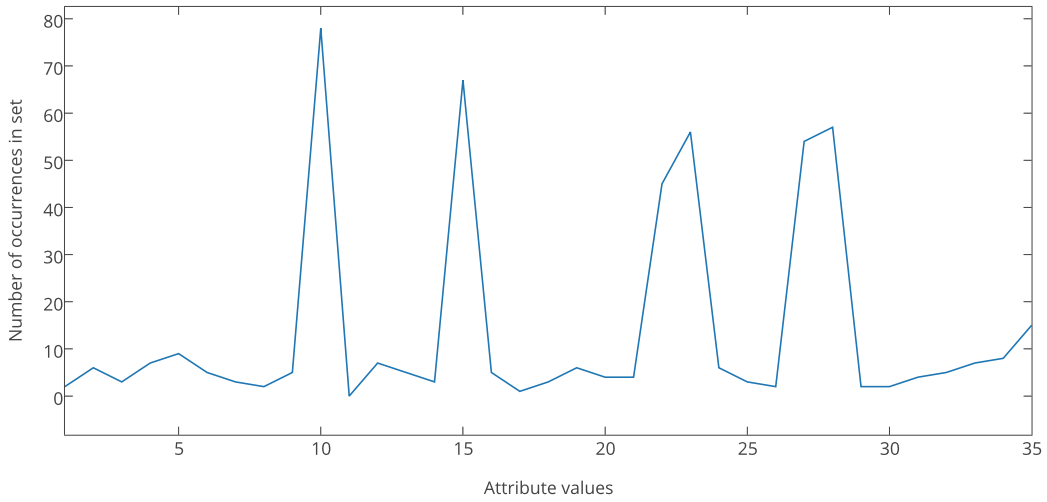

Figure 3.1: The distribution of attribute values among existing customers.

Our method partitions the distribution by first approximating a polynomial fit on the distribution curve using **Least squares** (Legendre, 1805), then forms new segments by splitting the distribution at the approximation's minima. Figure 3.2 shows the Least squares approximation of the distribution and Figure 3.3 shows the new segments, partitioned at the approximation's minima. The number of segments created by the approximation fit can be controlled by varying the degree of the polynomial fit.

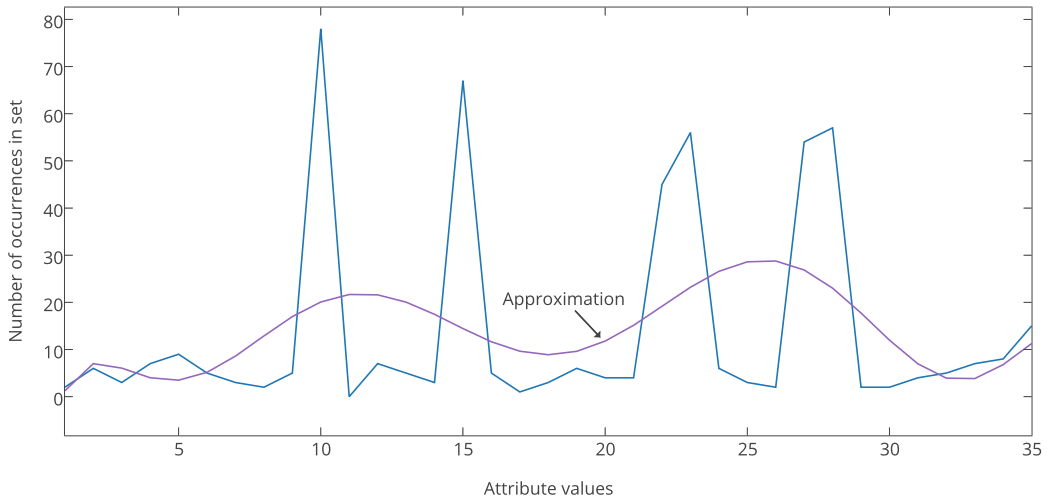

Figure 3.2: The Least squares approximation for the distribution.

Next, if the new bell-shaped segments are treated as normal distributions, it is possible to compute the expected values μ_i and the standard deviations σ_i for the individual segments. The attribute values can now be partitioned into discrete groups by creating

intervals of size $2\sigma_i$ around the expected values of the individual bell-curve segments:

$$\{[\mu_1 - \sigma_1,\ \mu_1 + \sigma_1],\ [\mu_2 - \sigma_2,\ \mu_2 + \sigma_2],\ \ldots,\ [\mu_n - \sigma_n,\ \mu_n + \sigma_n]\},$$

where n i the number of segments created from the approximation. Finally, all values in the continuous range are partitioned into two groups: the *representative range* consisting of the values that lie within any of the intervals created from the approximation-curve segments, and the *non-representative range* consisting of the rest of the values.

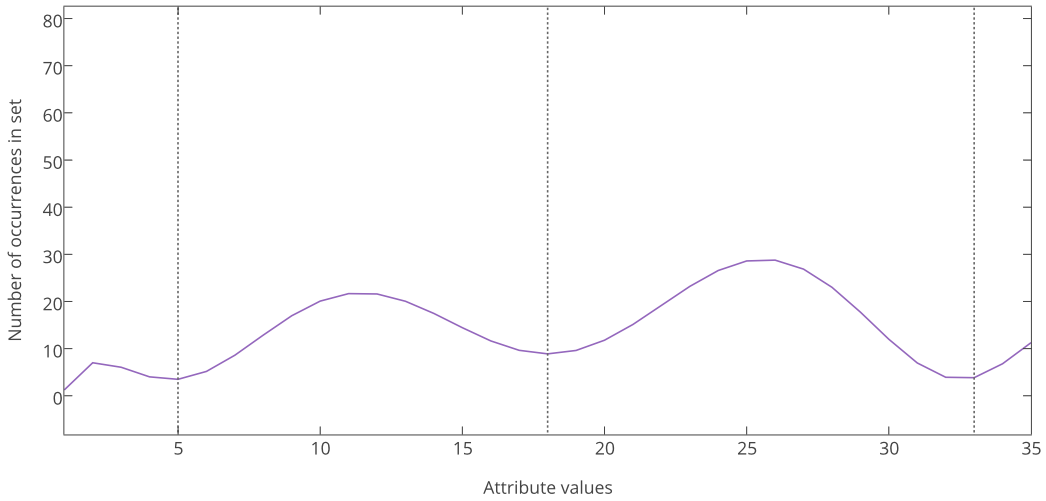

Figure 3.3: Four segments are created by splitting the distribution at the approximation's local minima. The dotted lines mark the split points.

The second method was tested to perform marginally better than Entropy-based partitioning, when used in combination with all three classifier algorithms evaluated in this project. Additionally, the second method was much faster, generally completing the discretization process in seconds, compared to many minutes using Entropy-based partitioning, which is computationally expensive due to the amount of iterations needed for its exhaustive search for maximum gain.

3.2 Frequent Set Counting using Apriori

Frequent Set Counting (FSC) is often used as an unsupervised learning method for finding frequently occurring subsets of values in a data set (Orlando et al., 2001). The method is similar to n-*gram* extraction used in natural language processing. n-gram extraction works by counting the frequency of n consecutive words in a corpus [1] for determining the statistical probability of a specific order of n words (see Nugues, 2010, chap. 4). The same principle can be applied on data sets with unordered and unrelated attribute values. All possible combinations of n attribute values are created using every available pair in the

[1] In linguistics, a *corpus* is a large structured set of texts.

data, *i.e.*, the **power set**[2] is created, and then all corresponding occurrences are counted. By applying FSC on *labeled* data, the method can be used for supervised learning by counting frequently occurring ***n*-itemsets** (an n-itemset consists of n attribute values or *items*) that are labeled as positive. Consequently, n-itemsets with a high probability of representing a positive instance can be identified and used to classify unlabeled data.

By varying the size of n, the complexity of the model can be increased or decreased. Using a small n will yield a model with *high bias* and *low variance*. Conversely, a large n will yield a model with low bias and high variance, however, finding frequent occurrences of large n-itemsets will become increasingly difficult as the size of n increases.

When the method is applied for classification, a *threshold* can be used for determining the minimum number of occurrences of an n-itemset that is needed for it to be used for positive classification. In other words, an n-itemset needs to be sufficiently frequent for it to be representative of the set of data labeled as positive. It is even possible to use combinations of frequently occurring n-itemsets that have different sizes of n for classification.

Company	Location	Revenue (SEK)	# of employees	Label
A	Stockholm	2 mn	300	Customer
B	Copenhagen	2 mn	100	Non-Customer
C	Helsinki	1 mn	100	Non-Customer
D	Stockholm	2 mn	300	Customer
E	Copenhagen	1 mn	200	Non-Customer
F	Stockholm	1 mn	300	Customer

Table 3.1: Training set of labeled companies.

Table 3.1 shows a training set of companies that are labeled as customers or non-customers of a hypothetical company aspiring to use frequent set counting for identifying new customers. Companies labeled as customers are considered as positive instances and companies labeled as non-customers are considered as negative instances. Table 3.2 shows the most frequently occurring n-itemsets among the 3 positive instances. Here, the threshold is set to 0.5, *i.e.*, the table only shows the n-itemsets present in at least 50% of the positive instances. Observe that the 2-itemset **{Stockholm, 300}** is present in all of the positive instances ($n = 2$ as two items are present), as well as the 1-itemsets **{Stockholm}** and **{300}** (1-itemsets consist of a single item), which are also present in all instances. The 3-itemset **{Stockholm, 2 mn, 300}**, the 2-itemsets **{2 mn, 300}** and **{Stockholm, 2 mn}** and the 1-itemset **{2 mn}** are present in two thirds of the positive instances.

Using this information it can be hypothesized that companies located in Stockholm which have 300 employees are likely to fit the current customer profile and should therefore be classified as positive instances. Companies located in Stockholm, having 300 employees and that additionally have a revenue of SEK 2 million could also be hypothesized to be positive instances if the threshold is set to a lower value. Note that using the 1-itemsets will yield a model with very high bias as, for example, if a classifier would classify all companies located in Stockholm as positive instances, this would likely result in a large number of irrelevant positives (underfitting). On the other hand, using only the biggest

[2]The *power set* of a set S is the set whose members are all possible subsets of S, *e.g.*, the power set of $\{1, 2\}$ is $\{\{\ \}, \{1\}, \{2\}, \{1, 2\}\}$

n-itemset	n	Occurrence
{Stockholm, 300}	2	3 of 3
{Stockholm}	1	3 of 3
{300}	1	3 of 3
{Stockholm, 2 mn, 300}	3	2 of 3
{Stockholm, 2 mn}	2	2 of 3
{2 mn, 300}	2	2 of 3
{2 mn}	1	2 of 3

Table 3.2: List of most frequently occurring n-itemsets among companies labeled as customers.

n-itemset available will likely result in very few positive classifications, although most, if not all, will be relevant. Moreover, if the intent is to find new customers then finding the exact same companies labeled as existing customers in the training set is of no use. Therefore it can be concluded that using a too high value for n will not generalize well to different sets of data and will lead to overfitting.

3.2.1 Implementation

The FSC learner model consists of the process of identifying the frequently occurring n-itemsets and storing these for future classification use. The algorithm used for identifying the frequent itemsets was implemented by first creating a *sparse matrix* representation of all the *features* in the training set. A **feature** is an attribute-value pair used to describe a value related to an attribute. This is needed as values need to be distinct among the attributes *e.g.*, the value "3" will have different meanings when observed within either of the two different attributes *employees* or *revenue*. The training set depicted in Table 3.1 have 8 distinct features as can bee seen in Table 3.3

Feature #	Attribute	Value
1	Location	Stockholm
2	Location	Copenhagen
3	Location	Helsinki
4	Revenue	1 mn
5	Revenue	2 mn
6	Employees	100
7	Employees	200
8	Employees	300

Table 3.3: All features present among the positive instances in the training set.

The purpose of using a sparse matrix representation is to get an ordered set of all the features, to save storage space and also to produce a compact representation of the data to utilize *locality of reference*[3] for performance gains. The sparse matrix saves storage

[3] *Locality of reference* is a principle that states that working with data located in close proximity, both in

space as all features can be stored in a *binary matrix* as opposed to storing the features themselves. An example of a sparse matrix representation can be seen in Table 3.4.

Company	Feature 1	Feature 4	Feature 5	Feature 8
A	x		x	x
D	x		x	x
F	x	x		x

Table 3.4: Sparse matrix representation of the positive instances in the training set where "x" indicates that the feature is present. Columns for features 2, 3, 6 and 7 are not displayed as those features are not present in any of the positive instances.

The next step consists of creating the power set of the set of features by producing **supersets** from all the combinations of all features in the training set. Creating the supersets is performed in a *bottom up* manner, *i.e.*, first, 1-itemsets are created from all features, then 2-itemsets are created from all possible combinations of the 1-itemsets, and so on until all combinations of n-itemsets have been created. However, identifying all itemsets from all combinations of features using a brute force approach will become very expensive both in time and space when the number of features is large (Orlando et al., 2001). The number of supersets that can be generated from a data set containing k different features is $2^k - 1$ (Tan et al., 2006). Thus, the search space of itemsets is exponentially large. See Figure 3.4 for a graph depicting all itemsets produced from a data set containing only 5 features.

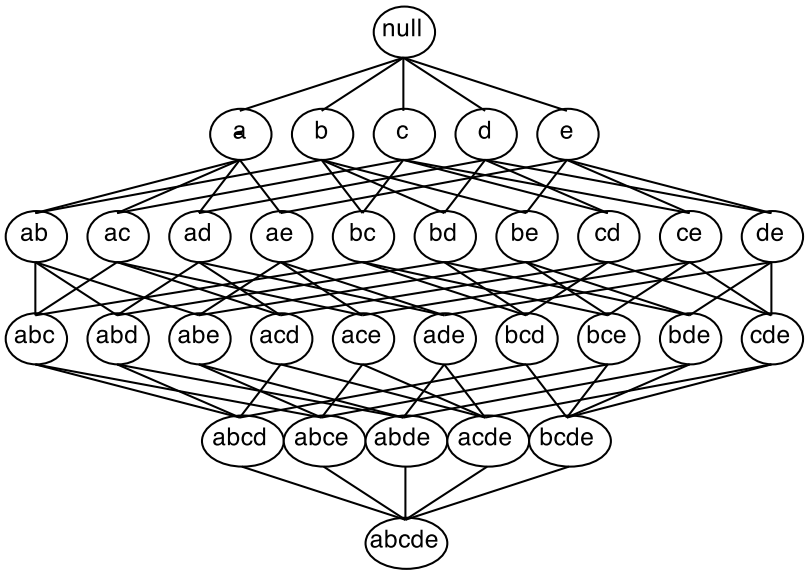

Figure 3.4: The lattice structure shows the 31 possible itemsets produced from a data set containing 5 features.

To remedy the problem of exponential growth, the *Apriori principle* (Agrawal and Srikant, 1994) can be applied to the problem.

time and space, can make better use of the built-in CPU cache.

The **Apriori** principle states:

"If an itemset is frequent, then all of its subsets must also be frequent." (Tan et al., 2006)

i.e., suppose that the set **{b, c, d}** is frequent, then all of its *subsets* **{b, c}**, **{b, d}**, **{c, d}**, **{b}**, **{c}** and **{d}** must also be frequent. Conversely, if a subset **{a, b}** is *infrequent*, then all of its *supersets* must also be infrequent (Agrawal and Srikant, 1994). Using this strategy, it is possible to immediately prune the entire subtree of supersets that contains the subset **{a, b}**, *i.e.*, the subsets **{a, b}**, **{a, b, c}**, **{a, b, d}**, **{a, b, e}**, **{a, b, c, d}**, **{a, b, c, e}** and **{a, b, d, e}** can be removed from the search space. By trimming the search space using the Apriori principle, the number of possible itemsets that need to be identified by the algorithm is reduced drastically.

To take advantage of the conclusions drawn above, the frequent set generation can be implemented by using a threshold value indicating the minimum support required for an itemset to continue generating supersets. Starting at the 1-itemsets, the frequency of each itemset is counted among all occurrences in the actual data set (*e.g.*, the training set). If the frequency is found to be less than the threshold value, the algorithm will discard the itemset from further use. Take, for example, the set of positive instances represented in the sparse matrix in Table 3.4, if an algorithm were to identify all itemsets using the Apriori principle it would start with counting the frequency of all 1-itemsets. Given a threshold value of 0.5, a single feature itemset must be present in at least 2 of the 3 instance rows to qualify for further itemset generation. Itemsets **{1}**, **{5}** and **{8}** is present in at least 2 instances, but itemset **{4}** is only present in 1 instance, hence it is removed from further use. Next, 2-itemsets are created by merging the available 1-itemsets resulting in the itemsets **{1, 5}**, **{1, 8}** and **{5, 8}**, of which all are present in 2 or more instances. Finally, a 3-itemset **{1, 5, 8}** can be created by merging the frequent 2-itemsets. The 3-itemset itself is present in 2 instances and will thus also qualify as frequent. No more itemsets can be identified as the full search space has been explored. When comparing these results with the frequently occurring itemsets in Table 3.2, it can be seen that the same itemsets have been identified (use Table 3.3 to translate feature numbers into features, respectively).

Pseudocode for the FSC algorithm using the Apriori principle, is showed in Algorithm 1.

Algorithm 1 Frequent set counting using the Apriori principle.

```
1: procedure FrequentSetCounting(L, t)
2:     F_1 ← {frequent 1-itemsets}
3:     k ← 2
4:     while F_{k-1} ≠ ∅ do
5:         C ← Generate all possible k-itemsets from F_{k-1}
6:         F_k ← Filter C, keeping only itemsets that are frequent above threshold.
7:         k ← k + 1
8:     return ∪F_k
```

3.2.2 Performance evaluation

Our tests show that the model have high accuracy when performing classification using large itemsets. However, identifying larger itemsets during training requires using a low threshold value, resulting in increased running time. Measured running time for varying threshold values is showed in Table 3.5. As can be seen by the figures, the running time seems to be inversely exponential in relation to the threshold value. The relation between running time and threshold value is also depicted in Figure 3.5.

Threshold	Running time (s)	Max set size
0.85	1	2
0.8	1	4
0.77	2	6
0.76	7	6
0.75	35	7
0.74	202	7

Table 3.5: Time spent training for varying threshold values.

The size of the itemsets identified during training is also related to the choice of threshold value, as can be seen by our measures showed in Table 3.5. Using lower threshold values will enable the model to identify larger frequent itemsets, but at the cost of longer running times. Thus, it becomes impractical to identify itemsets consisting of more than approximately 7 items, as it simply will take too long.

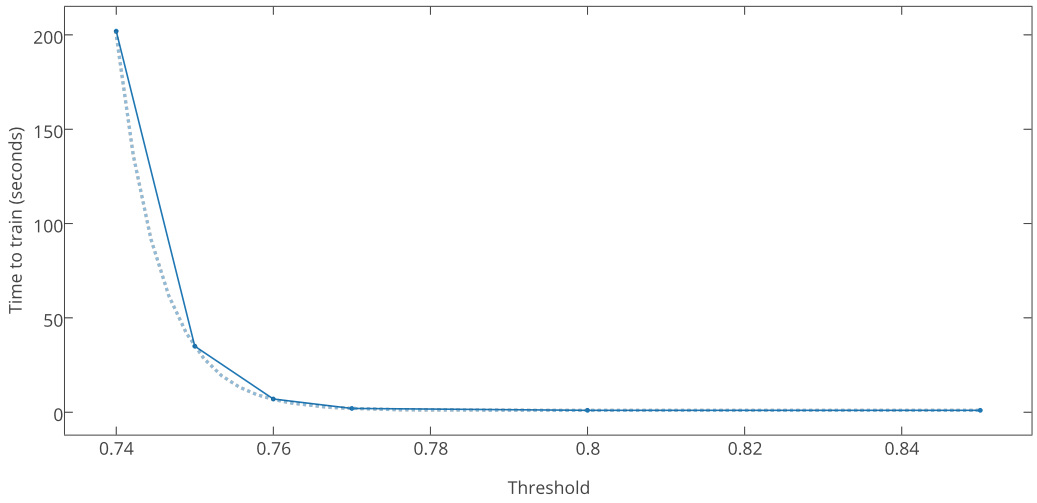

Figure 3.5: Time spent training for varying threshold values.

During classification, different intervals of itemset sizes can be chosen for different performance results. By using a broader range of itemset sizes, a higher bias is achieved. For example, using an interval of 1 to 7 means that all itemsets with sizes 1 through 7 are used for classification. Performance results are showed in Table 3.6. The tests was performed using a threshold value of 0.75. The results show that higher accuracy is achieved

when using only larger itemset sizes (see Figure 3.6). This however, comes at the cost of reduced recall scores. Also, the precision score is constantly low for all set intervals, never exceeding 0.01 (see Figure 3.7).

Set interval	Accuracy	Precision	Recall	F-measure
1–7	0.004	0.002	1.000	0.004
2–7	0.012	0.002	0.999	0.004
3–7	0.105	0.002	0.995	0.004
4–7	0.562	0.004	0.949	0.008
5–7	0.832	0.010	0.867	0.019
6–7	0.838	0.010	0.823	0.019
7–7	0.841	0.010	0.809	0.019

Table 3.6: FSC performance results using $threshold = 0.75$.

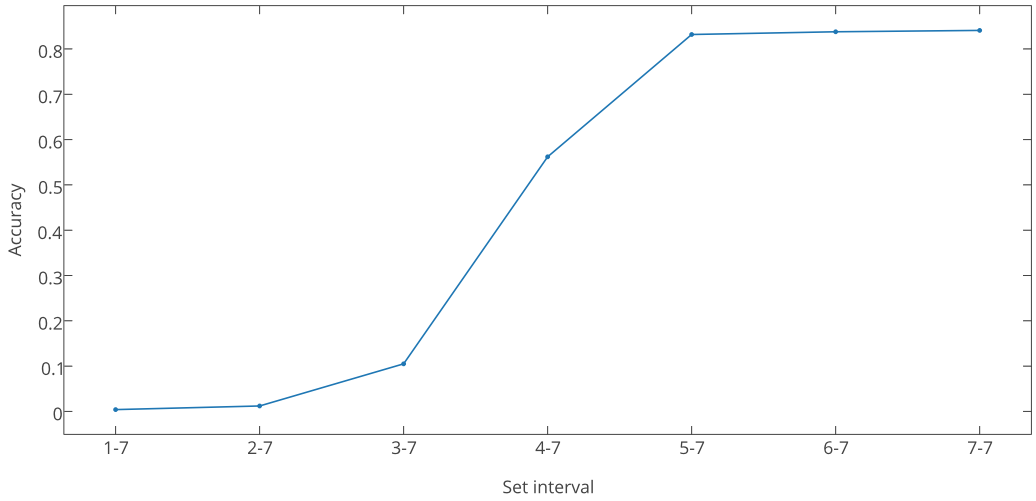

Figure 3.6: Accuracy increases as the choice of itemset interval becomes more conservative.

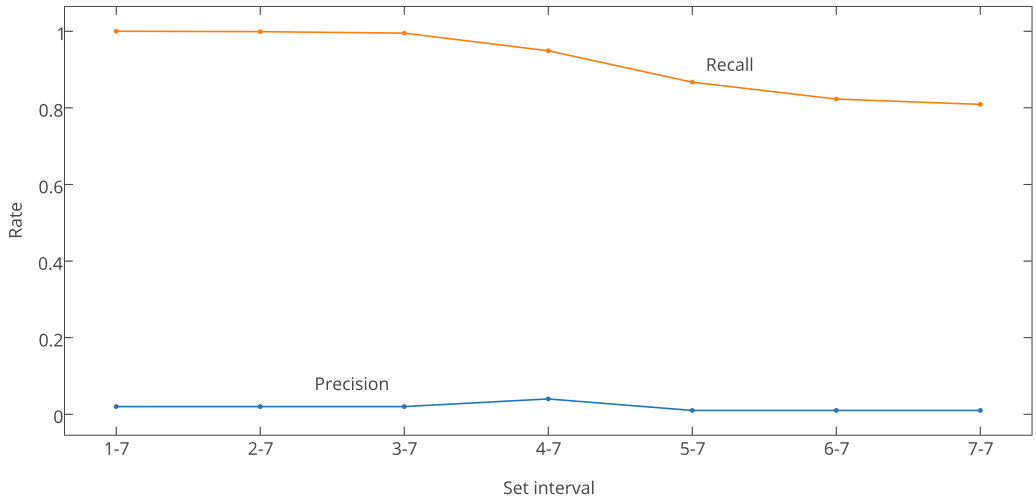

Figure 3.7: Recall decreases when the set interval is limited to only the larger sizes. Precision is at a constant low.

3.3 *k*-Nearest Neighbors

k-nearest neighbors (k-NN) is one of the oldest and simplest algorithms for pattern classification (Cover and Hart, 1967). Despite this it often yields good results in many domains compared to other algorithms. k-NN is considered to be a *lazy learner*, as it delays generalization on the training data until the model is applied for classification (Lantz, 2013). This is opposed to an *eager learner*, which performs generalization on the data during the training phase. The algorithm usually has good accuracy, but suffers from being computationally expensive both in time and space.

The algorithm works by classifying unlabeled instances by the majority label of its k-nearest neighbors in the training set. Neighboring instances are determined to be near by means of some distance metric that can be calculated and compared among all instances. See Figure 3.8 for a depiction of a two dimensional k-NN classification model. In the figure, an unlabeled instance is classified by measuring the distances between all labeled instances in the training set, here depicted as black and white dots, and determining the majority vote among the k-nearest instances. For example, if $k = 3$, the unlabeled instance will be classified as being associated with the **white class** as two out of three of the 3-nearest instances are white. A different choice of k might produce different results, *e.g.*, the unlabeled instance in the figure will belong to the **black class** when $k = 15$. Note that using odd numbers for k is preferable when performing binary classification as it avoids a tied vote (see the example in the figure where $k = 8$).

When training the model, every instance in the training set is represented by a vector consisting of all the features present in the respective instance. If the vectors carries n features, respectively, then all vectors are modeled in an n-dimensional space, where the features represent the vector coordinates (see, *e.g.*, Lantz 2013 for a more detailed description). The model creates the space of neighboring instances from a labeled training set, and then classifies each unlabeled test-set instance by inserting them one by one into the

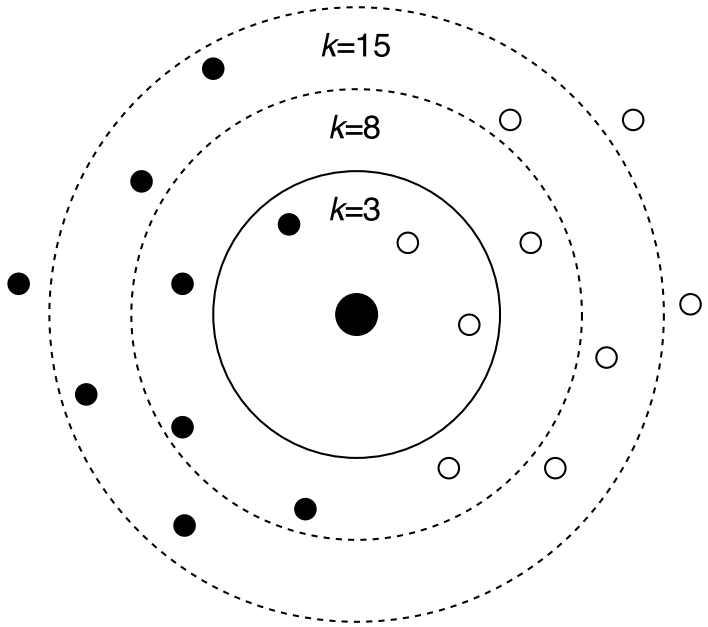

Figure 3.8: k-nearest neighbors classification visualized in two dimensions. The big black centerpiece is the unlabeled instance to classify. Black and white dots represent instances with different labels in the training set.

space and finding the k-nearest neighbors, respectively.

The choice of k is user-defined. Choosing a good value for k is reliant on the characteristics of the data. Choosing a small k may yield accurate results as closer neighbors will likely be more similar, but it will also cause the learner to be more sensitive to noise in the training data. A larger k will cause the learner to be less dependent on noise, but as k approaches the total number of neighbors in the training set, the result will have less and less predictive value (a majority vote among all instances will be entirely dependent upon the balance of class membership in the training set).

3.3.1 Implementation

There are many different methods for calculating the distance between instances, but it is common to use *Euclidean distance* for **continuous attributes**, and *Hamming distance* or *Levenshtein distance* (also called edit distance) for **discrete attributes**. These methods are usually good enough, but sometimes the model accuracy can be increased significantly by using more advanced distance metrics (Weinberger and Saul, 2009). The **Euclidean distance** between two points P and Q in an n-dimensional space can be calculated using:

$$d(P,Q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}.$$

The **Hamming distance** can be calculated for two strings of equal length by counting the number of characters for which the strings differ (Hamming, 1950). For example, the Hamming distance between "paper" and "vapor" is 2. For strings of unequal length a generalization of the Hamming distance metric, called the **Levenshtein distance** (or edit distance), can be used (Levenshtein, 1966). The Levenshtein distance between two strings can be determined as the minimum number of single-character insertions, deletions, or substitutions needed for the strings to become identical. See Algorithm 2 showing pseudocode for an implementation of a recursive Levenshtein-distance function. A *dynamic programming* approach can be used to avoid the inefficiency of recomputing the distances of the same substrings multiple times.

Algorithm 2 Computing Levenshtein distance.

```
1:  procedure LevenshteinDist(s, t)
2:      if Length of s is zero then                          ▷ Base case: empty string s
3:          return Length of t
4:      if Length of t is zero then                          ▷ Base case: empty string t
5:          return Length of s
6:      if s and t have same last character then
7:          cost ← 0
8:      else
9:          cost ← 1
10:     return The minimum of:
            LevenshteinDist(s − 1, t) + 1                    ▷ Remove last letter from s
            LevenshteinDist(s, t − 1) + 1                    ▷ Remove last letter from t
            LevenshteinDist(s − 1, t − 1) + cost             ▷ Remove last letter from s and t
```

In this project, all continuous attributes was discretized prior to the model being trained, hence, only the Levenshtein metric was needed (if desired, a lookup table containing type mappings for all attributes in the data can be used to keep the attribute types apart).

3.3.2 Performance evaluation

Tests performed by us show that the k-nearest neighbors algorithm is computationally expensive when applied on large sets of data. Because of the time required, only **small fractions** of the training set data could be used for testing. Table 3.7 shows the measured performance of the evaluation implementation on the test machine for varying training set sizes (the number of instances in the training set). The training sets used was balanced using 30% positive instances and 70% negative instances. In this measurement, classification was performed on the full set of test data containing 2,227,831 unlabeled instances.

The data suggests a linear increase in running time as the number of instances in the training set increases. By approximating a *linear fit* on the data, it is possible to produce a model that can be used for estimating the running time required when using the reference training set in its entirety (it consists of 13,933 labeled instances). This approximation model suggests a running time of approximately 11 hours, which is clearly too much for the existing application (see Figure 3.9 for a linear model of the running time).

#	Running time (s)	Accuracy	Precision	Recall	F-measure
22	72	0.83	0.01	0.86	0.019
83	241	0.88	0.009	0.54	0.017
167	478	0.90	0.012	0.68	0.024
250	712	0.86	0.012	0.75	0.024
333	946	0.88	0.012	0.75	0.023

Table 3.7: Measured performance results for varying training set sizes. $k = 3$ was used for all measurements.

The results in Table 3.7 show that accuracy is above 80% even for the smallest training set, but the precision is very low at around 1%. This is reflected by the small training set sizes used, as the model have a hard time determining the relevancy of information with such a low amount of training data.

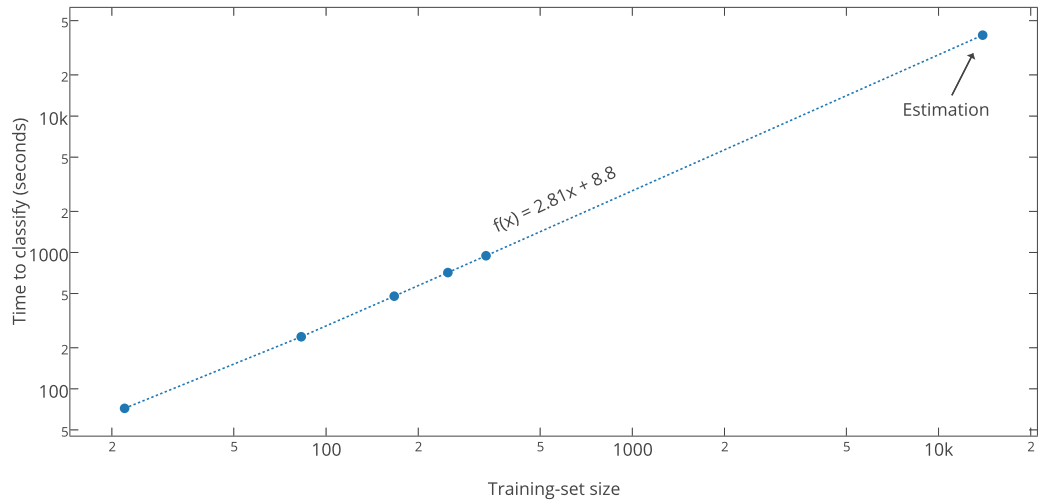

Figure 3.9: Measured running time on the test machine with varying training-set sizes (logarithmic scales used).

When using a **smaller test set**, however, the full training set could be used for performance evaluation. Classification-performance results using this method are showed in Table 3.8. The reference training set used consists of 13,933 labeled instances (30% positive instances), and the test set used contained about 14,000 unlabeled instances. The performance metrics are presented for different values of k. The results show that the model is accurate with both precision and recall reaching almost 90% for small values of k. The running time using this method was around 4 minutes for all values of k. The graph showed in Figure 3.10 depicts a trend of decreasing accuracy as larger values of k are chosen.

k	Accuracy	Precision	Recall	F-measure
3	0.92	0.86	0.88	0.87
5	0.91	0.84	0.85	0.85
9	0.90	0.82	0.83	0.83
15	0.89	0.82	0.82	0.82
19	0.89	0.82	0.81	0.81
25	0.89	0.82	0.80	0.81
33	0.88	0.81	0.80	0.81
45	0.88	0.81	0.79	0.80

Table 3.8: k-nearest neighbors performance results.

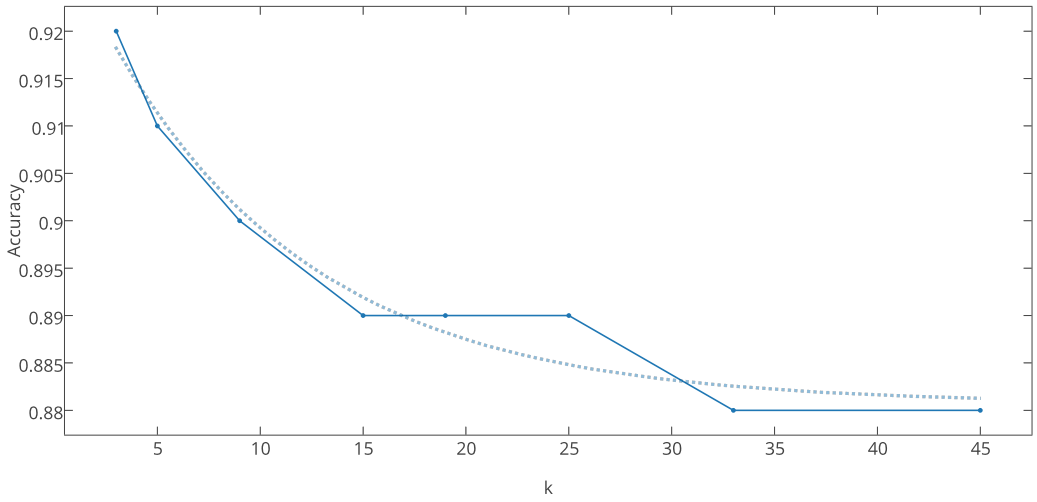

Figure 3.10: Measures show that accuracy decreases as k is increased.

Figure 3.11 shows that both precision and recall are subject to the same falling trend as larger values of k are chosen.

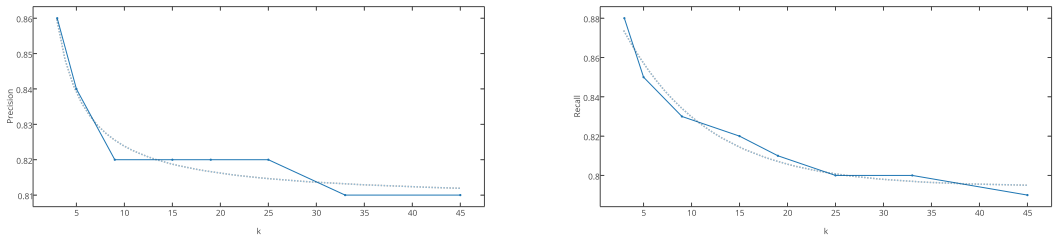

Figure 3.11: Both precision (left) and recall (right) decreases as k is increased (the same horizontal scale as for accuracy is used).

Further, our results show that the bias-variance trade-off can be controlled somewhat by varying the degree of positives in the training set. Training sets with the proportion of instances labeled as positive was varying from 30% to 70% was used for testing and the results can be seen in Figure 3.12.

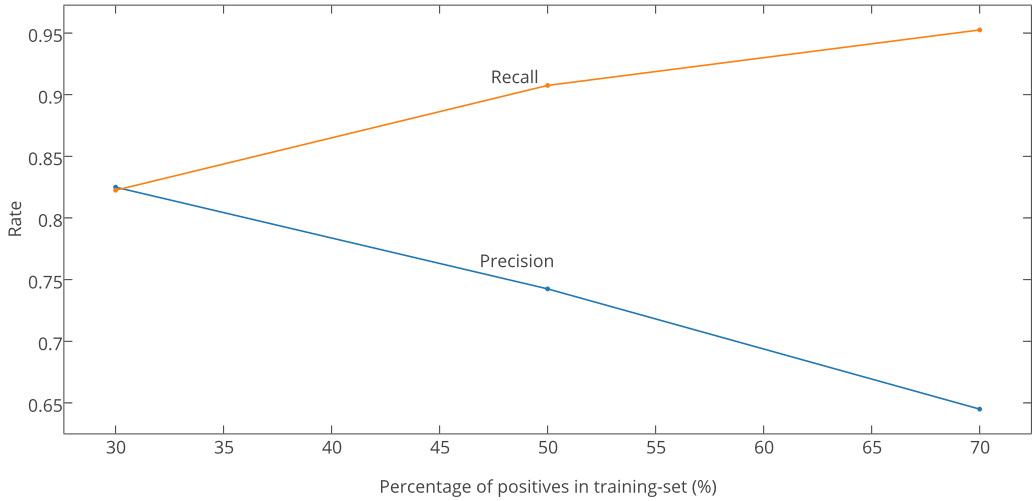

Figure 3.12: Precision and recall varies as the proportion of positives in the training set is changed.

The **learning curve** of k-NN (see Figure 3.13) shows that the algorithm has an accuracy above 80% even when used with smaller training set sizes (as have been previously shown).

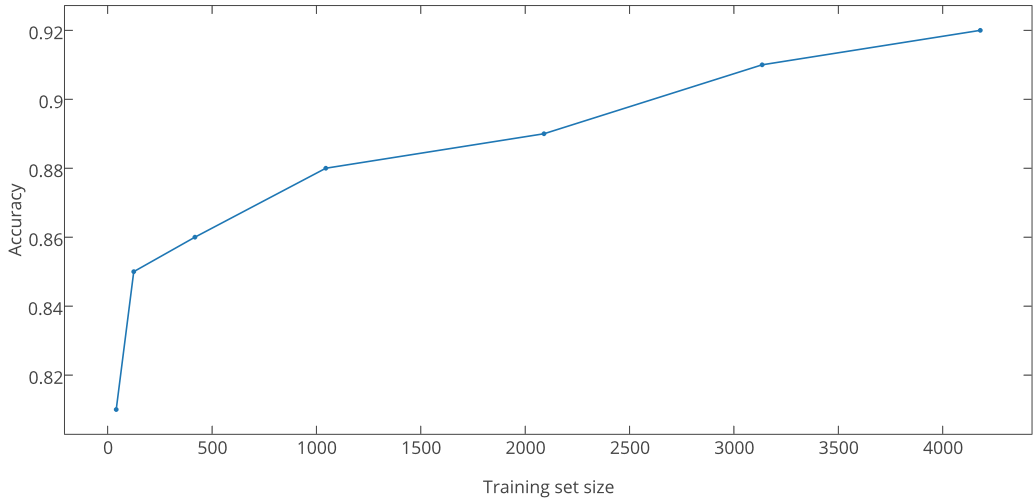

Figure 3.13: k-nearest neighbors learning curve.

3.4 Decision Tree Induction using C4.5

Decision tree induction is a machine-learning method used for supervised learning tasks. The method focuses on the modeling of the relationships between inputs and outputs in the form of if-then rules (originally described by Hunt et al. 1966). Decision tree learning is at the top of the list of various classification learning methods for meeting the requirements of being an off-the-shelf method for data mining. It is quite fast, produces models interpretable to humans, is resistant to irrelevant variables, immune to outliers[4] and can be easily extended to be used with many data types such as: discrete classes, numerical and time series (Hastie et al., 2013). Decision tree models often have low bias and high variance and might suffer from bad generalization when used on different sets of data (Geurts, 2002). There are, however, techniques for addressing this *e.g.*, by simplifying the model using discretization, tree pruning and ensemble learning.

Decision tree induction works by creating a tree structure from a *labeled* training set representing the multiple decisions required to reach a leaf of the tree. Each leaf in the tree determine a target-feature or class while the nodes represent attribute based tests with a branch for each possible outcome. The task of the induction is to create a classification model that can decide the class of any instance from its attributes and values (Quinlan, 1986).

Classification of an instance is performed by starting at the root of the tree, evaluating its test and continuing on the branch with the most appropriate outcome. If the selected branch leads to another node, its test is evaluated and the classification algorithm will

[4]In statistics, an *outlier* is an observation point that is distant from other observations.

continue onto another branch. The process will continue until a leaf has been reached, at which point the instance can be determined to be a member of the class named by the leaf.

Table 3.9 shows a training set depicting some weather data over 14 days (Quinlan, 1986). The set has been labeled with the coincidence of a person playing golf or not on that day. By using decision tree induction, it is possible to create a classification model that can be used for determining the likelihood of a person playing golf on any future day, provided that enough weather data is supplied.

Day #	Outlook	Temperature	Humidity	Windy	Play golf
1	sunny	hot	high	false	no
2	sunny	hot	high	true	no
3	overcast	hot	high	false	yes
4	rain	mild	high	false	yes
5	rain	cool	normal	false	yes
6	rain	cool	normal	true	no
7	overcast	cool	normal	true	yes
8	sunny	mild	high	false	no
9	sunny	cool	normal	false	yes
10	rain	mild	normal	false	yes
11	sunny	mild	normal	true	yes
12	overcast	mild	high	true	yes
13	overcast	hot	normal	false	yes
14	rain	mild	high	true	no

Table 3.9: A labeled training set with weather data.

An example of a decision tree created from the weather data is showed in Figure 3.14. Notice how the “overcast”-branch is connected directly to a leaf node as all instances in the training set that have the value **overcast** for attribute **Outlook** have the label **yes**

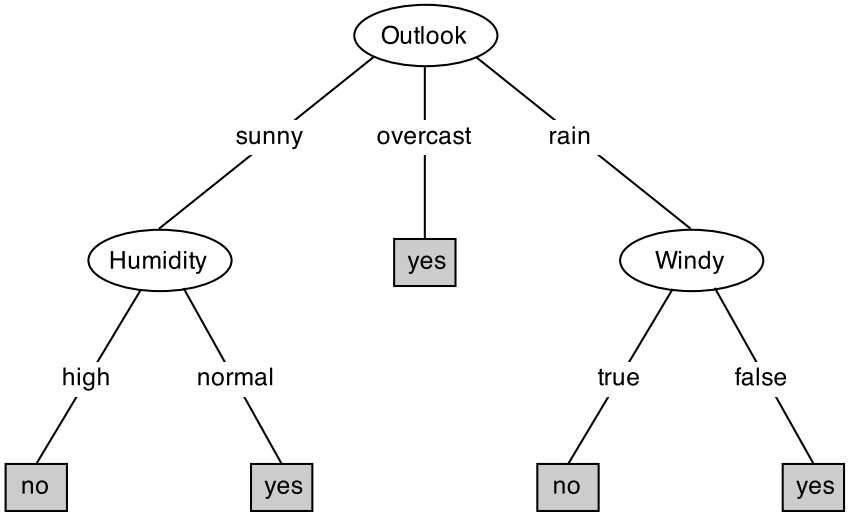

Figure 3.14: Decision tree for the weather data.

Take, for example, a particular Sunday morning, which is not included in the training set, that has the weather characteristics as follows:

Outlook: overcast

Temperature: cool

Humidity: normal

Windy: false

Using the aforementioned decision tree, it can be concluded that the Sunday morning can be classified as a member of the class **yes**, and is thus a probable candidate day for playing golf.

It is always possible to construct a decision tree that can correctly classify all instances in a training set given that there is enough attribute data available. However, there are usually many such correct decision trees as there is no predetermined order in which the attribute tests should be evaluated. Which attribute should be tested at the root of the tree and which attributes should be tested at the nodes? In the example given in Figure 3.14, the attribute **Outlook** is tested at the root, but it could just as well be replaced with the attribute **Humidity** and the tree would look very different. A different tree might perform equally well as a classification rule on the training set, but generally a smaller tree will generalize better to different sets of data as it will suffer less from having high variability being overly complex (Quinlan, 1986).

3.4.1 Implementation

To create a good decision tree during the induction process the algorithm **C4.5** was used (Quinlan, 1993). C4.5 is an extension to the original algorithm **ID3**, invented by Ross Quinlan. It provides some improvements over ID3 by being able to handle both continuous and discrete attributes, working with missing attribute values and by providing some built-in pruning capabilities. There is also another extension **C5.0**, but it is limited to commercial use and was not explored in this project. The C4.5 algorithm has generally been found to construct simple enough decision trees with good performance, but it can not guarantee that better trees have not been overlooked (Quinlan, 1986). This was, however, not a problem with the particular task of classifying companies as customers, or non-customers, because the models never showed the characteristics of being excessively complex. Some of the extensions provided by C4.5 over ID3 was not of interest in this project and was thus not implemented. Also, the discretization extension providing the algorithm the ability to handle continuous numeric attributes was implemented separately and executed as a separate step in the classification process. This was done, in order to enable evaluation of different types of discretization algorithms, independently of the classifier algorithm.

As C4.5 works in the same way as its predecessor, aside from providing some extended capabilities, the core implementation steps are identical to ID3. The steps of the ID3 algorithm are showed in Algorithm 3 (Quinlan, 1993). Here, R is the set of attributes to evaluate at the different nodes of the tree and S is the set of instances (the training set).
The algorithm works by using a divide-and-conquer strategy for partitioning the set of

Algorithm 3 ID3 decision tree induction algorithm.

1: **procedure** ID3(R, S)
2: **if** S contains one or more instances, all belonging to the same class **then**
3: **return** A leaf identifying the class
4: **if** S is empty **then**
5: **return** A leaf identifying the most frequent class in the parent node
6: **if** R is empty **then**
7: **return** A leaf identifying the most frequent class in S
8: $A \leftarrow largestGain(R, S)$ ▷ Get attribute with largest gain in S among R.
9: $\{v_i \mid i = 1,\ 2,\ \ldots,\ n\} \leftarrow$ the values of attribute A.
10: $\{S_i \mid i = 1,\ 2,\ \ldots,\ n\} \leftarrow$ the subsets of S consisting respectively of instances with value v_i for attribute A.
11: **return** A tree with a root node evaluating attribute A and branches labeled $v_1,\ v_2,\ \ldots,\ v_n$ going respectively to trees: $ID3(R - \{A\},\ S_1),\ ID3(R - \{A\},\ S_2),\ \ldots,\ ID3(R - \{A\},\ S_n)$

instances according to the values they contain in respect to the attribute being evaluated. This will continue until all instances in the current subset belong to the same class or until the subset is empty (Quinlan, 1993). The problem is to determine in which order the attributes should be evaluated. Our implementation used the standard method of measuring the *information gained* by branching on a particular attribute. Given a probability distribution $T = \{t_1,\ t_2,\ \ldots,\ t_m\}$, the information conveyed by this distribution (also called the **entropy** of T) is (Quinlan, 1986):

$$I(T) = -\sum_{i=1}^{m}(t_i \log_2(t_i)).$$

For example, if $T = \{0.5,\ 0.5\}$ then $I(T) = 1$, if $T = \{0.8,\ 0.2\}$ then $I(T) = 0.72$, and if $T = \{1,\ 0\}$ then $I(T) = 0$ (note that a more uniform distribution conveys more information). When performing binary classification, the information needed to identify the class of an instance in the training set S is the information conveyed by the probability distribution of the class affiliations in the set. If p is the number of instances that belong to class *Positive*, and n is the number of instances that belong to class *Negative*, then the probability distribution is:

$$T = \left\{\frac{p}{p+n}, \frac{n}{p+n}\right\}.$$

In the case of the weather training set showed in Table 3.9, the information needed to identify the class of an instance is $I(T) = I(\{\frac{9}{14}, \frac{5}{14}\}) = 0.94$.

If the training set S is partitioned on the basis of the value of an attribute A, with possible values $\{a_1,\ a_2,\ \ldots,\ a_v\}$, into subsets $\{S_1,\ S_2,\ \ldots,\ S_v\}$, then S_i will contain the instances in S that have value a_i of A. The information needed to identify the class of an instance of S_i is $I(\{\frac{p_i}{p_i+n_i},\ \frac{n_i}{p_i+n_i}\})$, where p_i is the number of instances in S_i that belong to class *Positive*, and n_i is the number of instances in S_i that belong to class *Negative*. Now,

the information needed to identify an element in S after the attribute A has been selected for testing is the weighted average (Quinlan, 1986):

$$E(A) = \sum_{i=1}^{v} \left(\frac{p_i + n_i}{p + n} I \left(\left\{ \frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i} \right\} \right) \right).$$

Finally, the **information gain** obtained by branching on A is defined as (Quinlan, 1986):

$$gain(A) = I \left(\left\{ \frac{p}{p + n}, \frac{n}{p + n} \right\} \right) - E(A).$$

This can be used to determine the evaluation order of the attributes when performing the induction task, by always choosing the attribute to branch on which has the highest information gain (Quinlan, 1986).

3.4.2 Performance evaluation

The performance evaluation of C4.5 shows that classification can be performed on large datasets with high accuracy. It is also possible to get high *precision* scores or *recall* scores, but as there is an inherent trade-off between the two, it becomes hard to achieve a high value of both measures at the same time. See Table 3.10 for a detailed summary of the performance results. These results where produced from the reference training set with varying proportion of positive instances. The lowest accuracy of 77% was achieved when training the model using a training set consisting of 90% positive instances. The best accuracy score of 99.9% was achieved when the model was trained on the entire data set consisting of 2,227,831 labeled instances giving a positive ratio of only 0.19%.

Positive ratio	Accuracy	Precision	Recall	F-measure
0.0019	0.999	0.980	0.35	0.52
0.005	0.998	0.560	0.44	0.50
0.01	0.997	0.360	0.53	0.43
0.025	0.99	0.185	0.65	0.29
0.05	0.99	0.116	0.74	0.20
0.1	0.98	0.071	0.82	0.13
0.2	0.96	0.044	0.89	0.083
0.3	0.95	0.032	0.96	0.063
0.5	0.91	0.020	0.95	0.038
0.7	0.87	0.014	0.97	0.027
0.9	0.77	0.008	0.99	0.016

Table 3.10: C4.5 classification performance results with respect to varying proportions of positives in the training set.

The measures of accuracy depicts a falling trend as the number of positive instances increases. Figure 3.15 shows the relationship between the proportion of positives and the accuracy. The horizontal scale in the figure is logarithmic to better show the variations of accuracy at lower positive ratios, as there are more measures in the lower range.

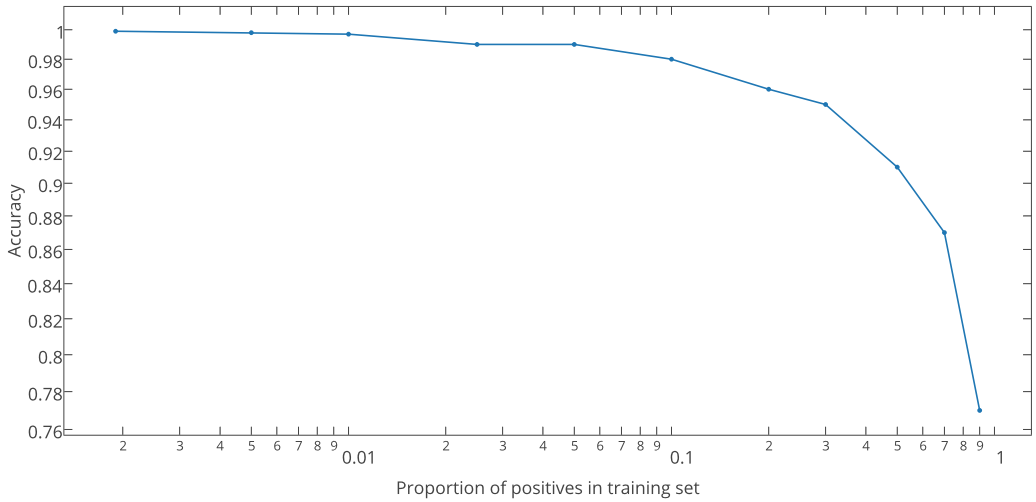

Figure 3.15: Measures show that high accuracy is achieved when the proportion of positives in the training set is low.

Our tests show that the precision score increases as the complexity of the model is increased by using a larger training set. Figure 3.16 depicts how precision increases as the proportion of positives increase. There also seems to be a steep rise of precision at positive rates below 10%, as can be viewed in the same figure.

Recall score seems to be subject to the inverse trend, as predicted by the trade-off effect. Figure 3.17 visualizes the rising trend as the proportion of positive instances increases. As with precision, there seems to exist a threshold at around 10%, causing a fast decline at rates below this threshold.

In practice, the apparent inverse relationship between precision and recall means that there is a trade-off between getting a high degree of relevancy among the results, and aggressively rounding up a large number of positive instances, which consequently will create a larger number of peripheral hits. Tests show, however, that this trade-off can be controlled by varying the proportion of positive instances in the training set. By using a low positive rate, a high degree of relevancy can be achieved among the results. The relationship between precision and recall is depicted in Figure 3.18 using logarithmic scales. This shows how the metrics coincide at a rate of around 0.5 and at a proportion of positives of approximately 0.65%.

The learning curve, displayed in Figure 3.19, shows that a high accuracy score persists as the training set size decreases. A small increase in accuracy can be observed as the training set size is increased. Precision, also depicted in the figure, is also subject to a small rise as the training set size is increased. The factor that seems most effected by training set size is recall, which increases as the training set size is increased. The overall low precision scores presented in the learning curve can be explained by the relatively high proportion of positive instances in the training set used during these measures. The learning curve was produced using training sets with varying sizes consisting of 10% positively labeled instances.

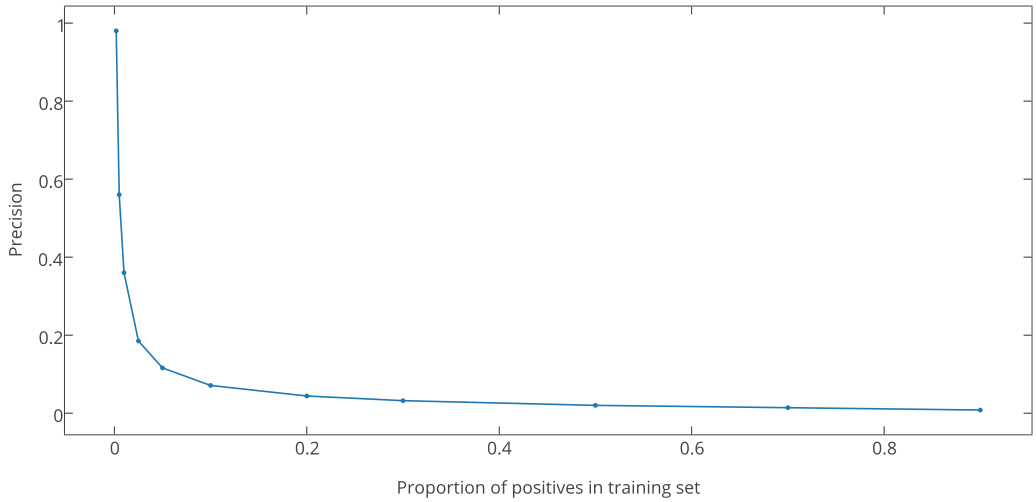

Figure 3.16: High precision is achieved when the proportion of positives in the training set is low.

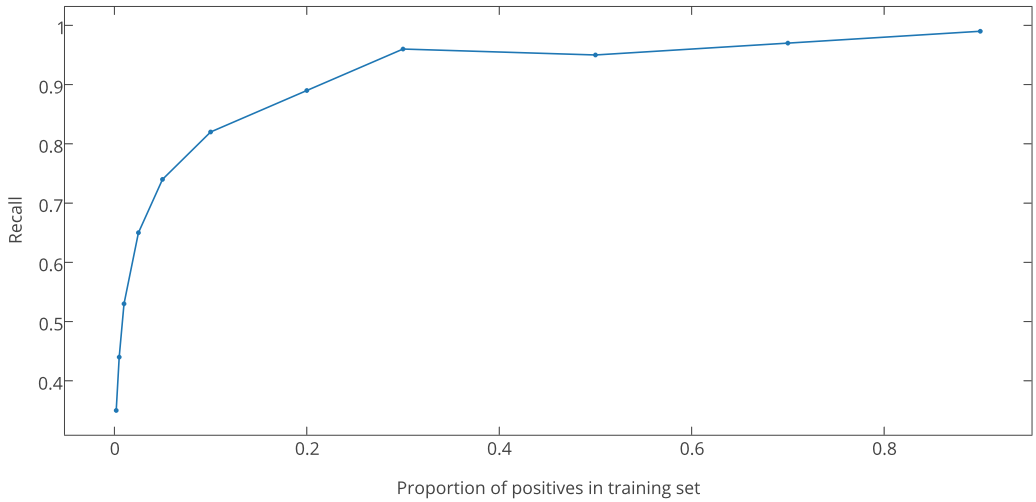

Figure 3.17: High recall is achieved with a high proportion of positives in the training set.

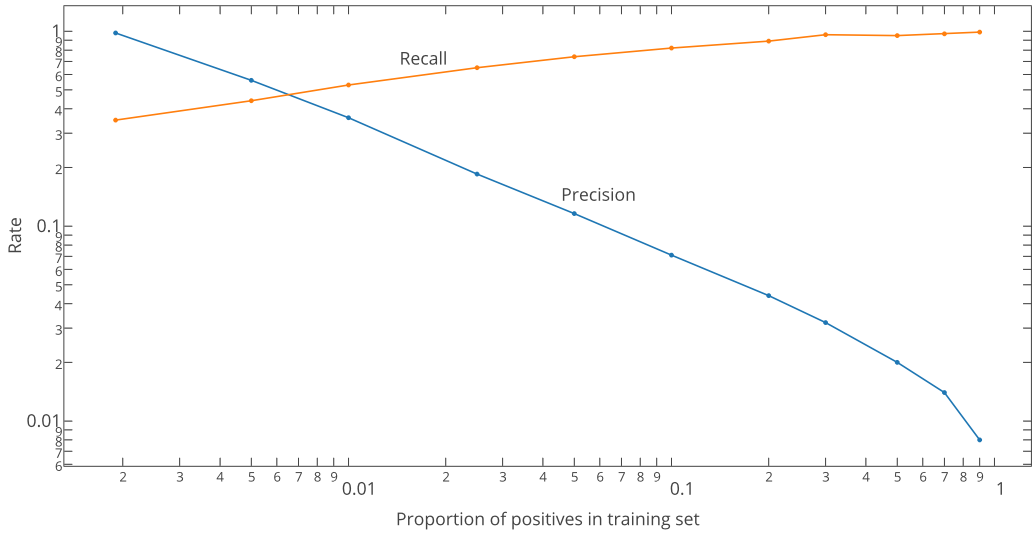

Figure 3.18: There seems to be a trade-off between precision and recall related to the proportion of positives.

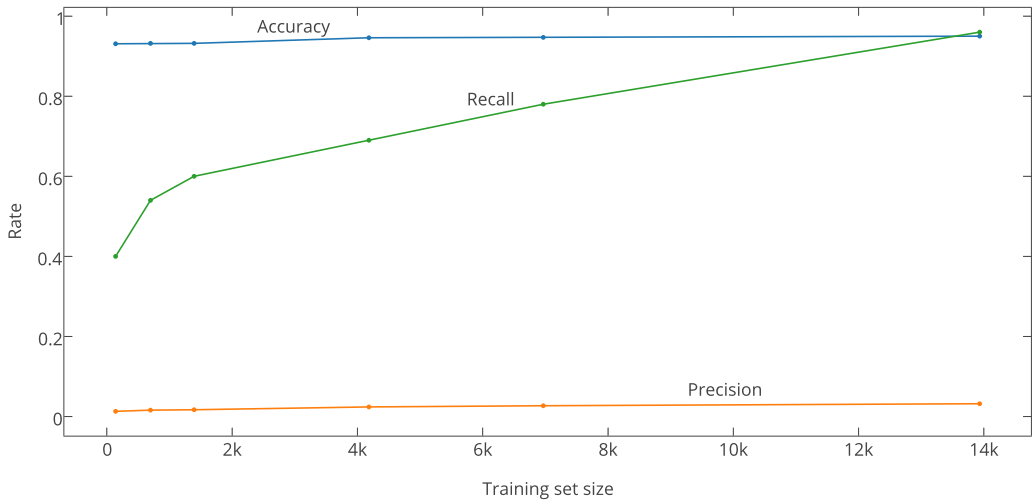

Figure 3.19: The learning curve of C4.5.

Overall running time during classification using the evaluation implementation of C4.5 never exceeded 1.5 minutes, including time spent training the model. The training phase make up for most of the time spent, with running times ranging from a few seconds up to approximately 1 minute when using the entire test set as training set. Time spent during classification was almost linear with respect to the size of the test set, and never exceeded 15 seconds.

3.5 *k*-Means Clustering

k-means clustering is an old, but popular algorithm that is known for its simplicity and speed (MacQueen, 1967). The algorithm belongs in the domain of *unsupervised learning*, and is used for *cluster analysis* of data. There are some variations of the basic algorithm, but the most common algorithm uses a technique that iteratively refines the results using geometric clustering (Arthur and Vassilvitskii, 2006). The common algorithm is based on work by Lloyd (1982).

This algorithm works by partitioning a set of unlabeled data into k clusters so that the local homogeneity of each cluster is maximized. It does this by representing the data points (the instances) as vectors in a *feature space* (similar to how k-NN represents its data). Then, the algorithm iteratively assigns the data points to the respective cluster whose *mean* has the least distance from the point. This continues until the means have converged to k points in the feature space and no longer changes position.

The algorithm can be summarized in **three steps:**

1. Initialize
2. Assign
3. Update

The initialization starts by creating the k **means** at arbitrary positions in the feature space. At this point the means are not necessarily the means or *centroids* of the respective clusters. Next, the assignment step proceeds to assign all data points to the cluster whose mean has the least calculated distance measure. Finally, the update step moves all the mean points to the centroid of the respective cluster.

The **centroid** or the geometric mean of an object in a d-dimensional space is *the mean position* of all the points in the space, *i.e.*, the arithmetic mean of all coordinates in all directions.

After the final step, the algorithm repeats steps 2-3 until no more points move between the clusters, in which case the means have converged into their final positions (see Lantz, 2013, chap. 9). The algorithm has an upper bound of the maximum number of iterations required to converge, determined by $O(k^n)$, where n is the number of points in the set (Arthur and Vassilvitskii, 2006). Thus, the algorithm is guaranteed to eventually converge on k clusters, but it is, however, not guaranteed to converge on the exact same clusters multiple times as the initial mean positions are randomized. Nevertheless, the clusters will still have a high probability of converging in the same way more than once when the data is sufficiently distinct by nature, and when the choice of k matches the number of natural segments that exist in the data.

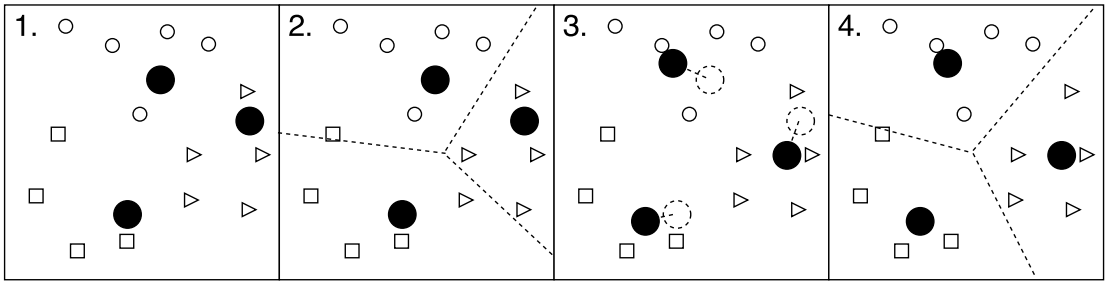

Figure 3.20: k-means clustering illustrated.

See Figure 3.20 for an example of the k-means clustering process, illustrated for $k = 3$. In **square 1**, the data points are depicted at their positions as white shapes (circles, triangles, and squares). Three mean points are placed at random positions, as depicted by the black circles (the initialization step). **Square 2** visualizes the clusters consisting of the data points assigned to their nearest mean point (the assignment step). Notice how the dotted lines denote the boundaries between the clusters. Next, in **square 3**, all of the means are moved to the mean position of all the data points in their clusters (the update step). This also moves the cluster boundaries, as showed in **square 4**. All data points can now be reassigned to the nearest mean, and new clusters have been produced.

3.5.1 Implementation

The implementation of k-means clustering that was evaluated in this project used data points represented as vectors denoting the presence of features in the point. In other words, each data point consists of a binary vector of 0s and 1s, where the number 1 denote the presence of a feature. This representation serves to allow for all attribute types, discrete and continuous, to be treated the same when computing the means. The feature space can be thought of as a d-dimensional hypercube, where all the data points are located in the corners of the hypercube. See Table 3.11 showing an example of a data set consisting of 3 instances and their coordinates, respectively.

Instance	Data point coordinates
{Location=A, Rating=1, Revenue=100}	[1, 0, 1, 0, 1, 0]
{Location=B, Rating=1, Revenue=200}	[0, 1, 1, 0, 0, 1]
{Location=A, Rating=2, Revenue=100}	[1, 0, 0, 1, 1, 0]

Table 3.11: Data set consisting of 3 instances with a total feature count of 6. Each point is represented by a vector with magnitude 6, denoting the presence of the features.

The choice of the initial means is very important in order to avoid empty clusters during the first iteration (Coates and Ng, 2012). There are two general methods for choosing the initial means: first, randomly selecting k points from the existing data points as means, also called the **Forgy method** (Forgy, 1965), and second, randomly assigning each point to a number ranging from 1 to k, denoting the cluster assignments, as described by James et al.

(2013, chap. 10). The implementation evaluated in this project used the Forgy method for the the initial means.

The assignment step consists of computing the distances between all data points and the means. Hence, it is necessary to decide on the method for computing the distance. In this project, we chose to calculate distance using Euclidean distance as given by:

$$distance(P, Q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_d - p_d)^2},$$

where P and Q are data points in the d-dimensional feature space.

To compute the mean vectors during the update step, the centroid of each cluster was calculated using the arithmetic mean of all coordinates in all directions, *i.e.*, for a physical object with uniform density, this would be the center of mass. It can be calculated using:

$$mean(C_u) = \left\{ \frac{\sum_{i=1}^{|C_u|} c_{i1}}{|C_u|}, \frac{\sum_{i=1}^{|C_u|} c_{i2}}{|C_u|}, \ldots, \frac{\sum_{i=1}^{|C_u|} c_{id}}{|C_u|} \right\},$$

where C_u is the uth cluster, $|C_u|$ is the number of data points or vectors in C_u, and c_{ij} is the jth element of vector c_i in cluster C_u. For example, if the data set from Table 3.11 would form a cluster, the mean of this cluster would be:

$$\left[\left\lfloor \frac{\Sigma \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix})}{3} \right\rceil, \left\lfloor \frac{\Sigma \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix})}{3} \right\rceil, \left\lfloor \frac{\Sigma \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix})}{3} \right\rceil, \left\lfloor \frac{\Sigma \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix})}{3} \right\rceil, \left\lfloor \frac{\Sigma \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix})}{3} \right\rceil, \left\lfloor \frac{\Sigma \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix})}{3} \right\rceil \right],$$

where $\lfloor x \rceil$ is the nearest integer of x.

Due to the randomized initial mean points, the algorithm will converge on different clusters when it is run multiple times using data that is not sufficiently distinct. It is obvious that there exists a problem in determining the best configuration of clusters. To address this, a method can be used to find the configuration of clusters for which the *within-cluster variation* is as small as possible. The **within-cluster variation** is a measure of how much the data points within a cluster differ from each other. The most common way of measuring this is by using the squared Euclidean distance (see James et al., 2013, chap. 10):

$$W(C_u) = \frac{1}{|C_u|} \sum_{i,i' \in C_u} \sum_{j \in F} (f_{ij} - f_{i'j})^2,$$

where C_u is the uth cluster, $|C_u|$ is the number of data points in the uth cluster, and F is the set of feature coordinates that represent the coordinates of each data point. In other words, the within-cluster variation for the uth cluster is the sum of all of the pairwise squared Euclidean distances between the points in the uth cluster, divided by the total number of points in the uth cluster.

It is now possible to determine the **total measure** of cluster variation for all clusters by summing the measures of the individual clusters (see James et al., 2013, chap. 10):

$$V(C) = \sum_{u=1}^{k}(W(C_u)),$$

where $C = \{C_1,\ C_2,\ \ldots,\ C_k\}$ is the set of all clusters.

The best cluster configuration can now be found by running the algorithm multiple times, and selecting the configuration that has the lowest total within-cluster variation, as given by $V(C)$.

3.5.2 Performance evaluation

k-means clustering is an algorithm used for cluster analysis of data, and as such, it can not be used for classification like the other algorithms evaluated in this project. However, we chose to include it as an example of how an unsupervised learner can be applied to gain insights into the test data.

Unsupervised learning algorithms can not be evaluated using the same approach as for supervised learners as there is no concept of a predetermined outcome. However, the algorithm was tested for its ability to detect the distinct classes of positive instances and negative instances in a labeled training set.

The testing procedure was carried out by conducting cluster analysis, using $k = 2$, on training sets with varying proportion of positives. The resulting clusters were compared to the true segments of positive and negative instances, and evaluated with respect to accuracy. For example, a hypothetical training set consisting of 1000 instances with 30% positives would optimally be clustered into two segments of 300 and 700 instances, respectively. The segment consisting of 300 instances would also exclusively contain true positives, in the case of a perfect model.

Accuracy can be measured if we presume the cluster containing the greatest number of *true positives*, to be the matches (the positives), and the cluster containing the greatest number of *true negatives* to be the negative matches (the negatives).

Positive rate	Size	Measured ratio	Variation	Accuracy
30%	13933	0.383	79110	0.844
	3777	0.352	23803	0.813
	3520	0.383	20980	0.842
50%	8360	0.524	50636	0.855
	2266	0.471	15378	0.804
	2112	0.522	13697	0.853
70%	5971	0.662	38082	0.867
	1619	0.589	11447	0.796
	1509	0.663	10489	0.851

Table 3.12: k-means clustering performance results using 3 separate training sets with varying proportion of positives.

The performance results from this evaluation is showed in Table 3.12. The column "Measured ratio" in the table depicts the proportion of instances found in the positive cluster after performing cluster analysis. This should be compared to the true proportion showed in the leftmost column. The column "Variation" shows the sum of the within-cluster variations for the two clusters. It is a relative measure of how homogeneous a cluster is, and should not be used for comparison between different training sets. In the last column, accuracy scores are presented, showing the proportion of true positives in the positive cluster combined with the proportion of true negatives in the negative cluster, with respect to the total number of instances.

The performance results can be viewed as a measure of how well the algorithm can distinguish between positive and negative instances when partitioning the data in two clusters. As is depicted in Figure 3.21, the cluster proportions matches quite well with the intended proportion of positives.

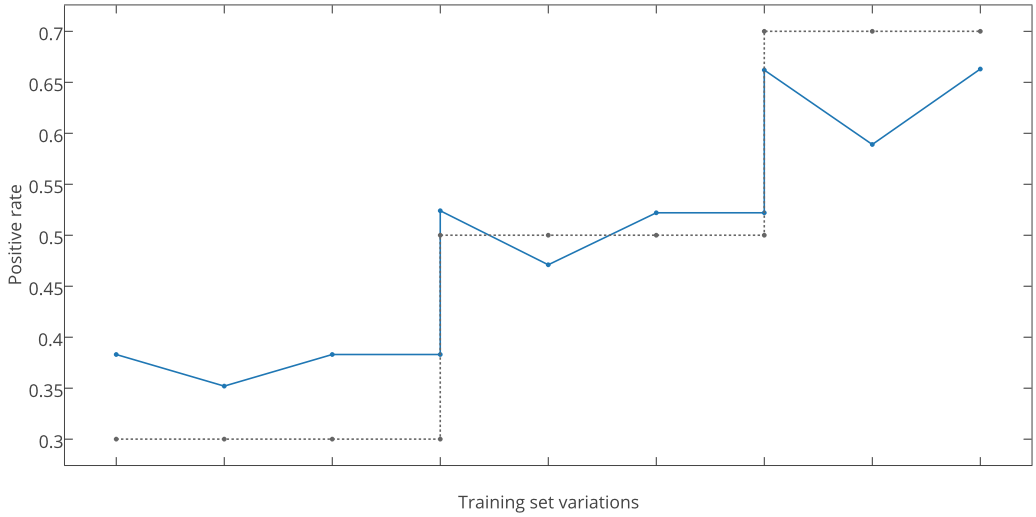

Figure 3.21: Cluster proportion viewed in relation to intended proportion.

Chapter 4
Discussion

4.1 Performance Comparison

4.1.1 Frequent set counting using Apriori

A consideration of using FSC for supervised learning is that the model has a small capacity for giving complex insight of the data. It can not make logical assumptions such as *e.g.*, if two frequent itemsets *intersect* each other (the itemsets have items that are common to both sets), but have subsets that are *disjoint*, then a model using FSC can not make the logic assumption that an itemset consisting of the union of the intersection and either of the disjoint subsets is frequent in the training set. Take, for example, the two 3-itemsets **{a, b, c}** and **{a, b, d}**, which both have a frequency of 50% in a training set. If the model would have been capable enough to understand that an itemset containing the items **a**, **b** and *either* of the items **c** or **d**, the itemset **{a, b, c|d}** would have a frequency of the combined occurrences, *i.e.*, 100% in the training set. However, this is not possible with FSC and a threshold value of 0.5 or lower would be needed to ensure the inclusion of the aforementioned itemsets. Using a low threshold value will make the model suffer from high bias, which should be avoided. This lack of logic insight into the data compared to other models can sometimes result in reduced accuracy and fewer relevant classification rules being identified.

4.1.2 *k*-Nearest neighbors

k-Nearest neighbors performs very well when applied on small sets of data. Accuracy is marginally lower compared to C4.5 when using small values of k. The model is also showed to suffer less from the bias-variance trade-off compared to C4.5, and maintains both high precision and recall throughout all tests. The model can thus be expected to provide accurate results of high relevancy. However, tests show that the model is compu-

tationally expensive with respect to time when larger sets of data are used. Running times of many hours can be expected when performing classification on the full data set used during evaluation.

An alternative method of using smaller training sets can be used to decrease the running time of the model, but our tests show that this will yield much lower precision scores, approximately around 1%.

Although the algorithm is computationally intensive, it is well suited for parallelization. The computationally hardest part of the algorithm is the computation of distances between the points in the training set. As all distances between all points must be computed, the algorithm has a time complexity of at least $O(n^2m)$, where n is the number of instances in the training set, and m is the number of distinct features. However, as all distance calculations can be performed independently, they can be *parallelized*. The evaluated implementation of the algorithm was parallelized prior to evaluation, using the multicore capabilities of the test machine, producing an almost linear increase of performance with respect to time. The test machine was, nevertheless, limited to 4 hardware cores providing, at maximum, a four-fold increase of performance.

4.1.3 C4.5 decision tree

Our tests show that a precision of 98% can be achieved by labeling the entire test set, and using this set to train the model. This will result in very high relevance, although few in numbers. It has also been shown that by varying the proportion of positive instances in the training set, precision can be traded for recall, which might be desirable in certain situations. Decision trees are often prone to achieving high variance, and because of this they might not generalize well to different sets of data (Geurts, 2002). As a consequence, classification on real data might not yield enough positive classification when the model is too complex. Yet, using the observed trade-off effect, the complexity of the model can be lowered, thus producing more positive classifications.

Accuracy score is observed to be sufficiently high in all tests, reaching just below 80% at its lowest when using a 90% positive rate in the training set. By keeping the proportion of positive instances below 50%, an accuracy of at least 90% can be achieved. The accuracy does seem to suffer some from smaller training set sizes, still the difference is marginal, according to the tests.

Run-time performance of C4.5 is acceptable using even the largest test set as training set, never exceeding 1.5 minutes.

4.1.4 *k*-Means clustering

This algorithm stands out as it is used for cluster analysis, as opposed to classification. As such it is hard to compare it against the other algorithms. k-Means clustering is, nevertheless a well performing algorithm in its own domain. Our tests show that the model can be used to successfully produce distinguishable clusters of data.

It would have been interesting to evaluate the algorithm for other values of k, but as it turns out, it is hard to define a desirable result for such an evaluation. Without a predetermined outcome, there is no way of knowing if the clusters produced are relevant or not. It is, however, possible to determine the *internal variation* within a cluster, but such

a measure does only tell how homogeneous a cluster is when compared to other clusters created at the same time. An external measure would require comparison of the results against a set of manually created clusters, which would be distinct in some way.

The run-time performance when performing cluster analysis is very good. Hundreds of iterations was completed in seconds when the model was applied using training sets containing thousands of instances. While the algorithm is guaranteed to converge eventually, it is sometimes practical to limit the number of iterations. During these tests the number of iterations were limited to 200, although the algorithm often managed to converge in 1-10 cycles.

4.1.5 ROC graph

All three classification algorithms are depicted side by side in the ROC graph in Figure 4.1. However, the curves in the graph are not actual ROC curves, but are instead a series of connected points in ROC space, depicted for each algorithm, respectively. This is the case as all three classifiers output discrete classes instead of numeric scores or probabilities, which is required for producing a true ROC curve. Thus, the curves should not be viewed for comparison, instead they depict the relative performance between different measures of the individual algorithms. For instance, the curve that represents the k-NN algorithm shows that this model is very conservative for all measures, staying at a low false positive rate as the true positive rate increases. C4.5, on the other hand makes less of a sharp turn, which shows that there is more of a trade-off between the true positive rate and the false positive rate when the algorithm performs optimally.

Although the curves should not be viewed for comparison, the optimal *points* can be compared. The optimal points on the curves are the measures that lie closest to the upper left corner of the graph, hence they should have higher accuracy, according to the ROC. The optimal points can be compared between the models as they represent the best measurement of performance produced during the evaluations for the different models, respectively. The optimal points in ROC space are surrounded by circles in the graph.

As can be seen in the graph, the C4.5 model has its optimum closest to the upper-left corner. The distances between the upper-left corner and the optimum points, respectively, are listed as follows:

C4.5 distance: 0.093 ***k*-NN** distance: 0.125 **FSC** distance: 0.194

The curve for FSC was created using different measures of the algorithm produced by varying the itemset size used during classification. For k-NN, different measures were produced for varying values of k. It should be noted that classification results for k-NN were evaluated using only a fraction of the full test set, as performing classification on the full test set would have been overly expensive with respect to time. The other models were evaluated using the full test set, hence it might not be an entirely fair comparison between k-NN and the other models. The C4.5 curve was produced using training sets with varying proportion of positives.

Finally an AUC score is presented for each algorithm in Table 4.1. The AUC scores were computed by using the trapezoidal Riemann sum approximation.

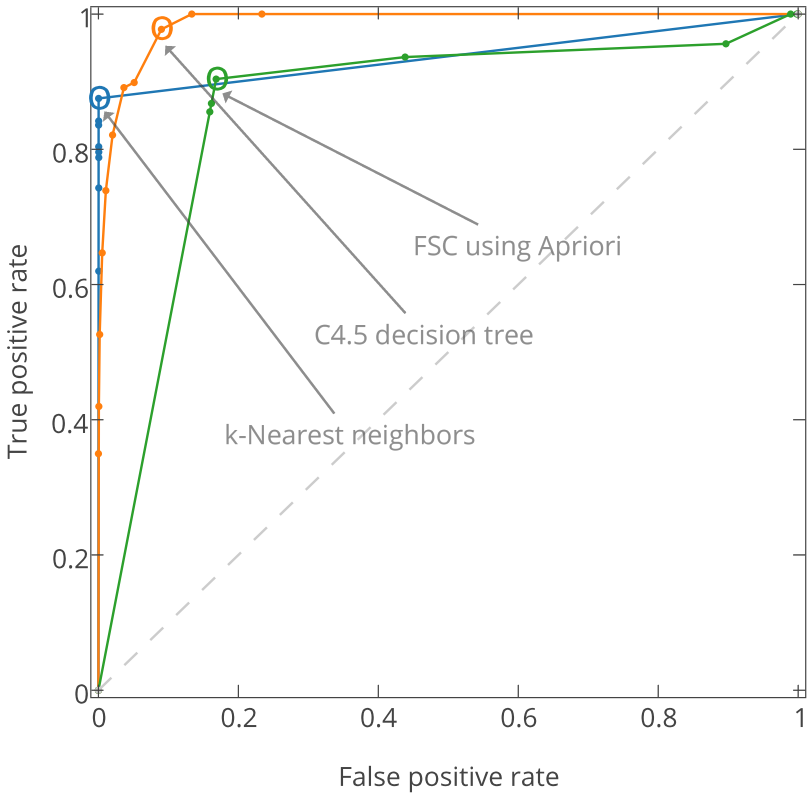

Figure 4.1: The supervised learning algorithms plotted in ROC space.

Algorithm	AUC	Grade
C4.5	0.987	A (outstanding)
k-NN	0.938	A (outstanding)
FSC	0.860	B (excellent/good)

Table 4.1: AUC scores.

Chapter 5
Conclusion

Current recommendation engines are typically built using machine-learning techniques and the results provided by this project show that relevant customer predictions can be made using common algorithms. In this thesis, four common off-the-shelf machine-learning algorithms have been evaluated. Three of the algorithms belong in the domain of supervised learning, and one in the domain of unsupervised learning. The algorithms that belong among the supervised learners have been measured with respect to performance of accuracy, relevancy of results, and run time using real-world data. The unsupervised learner was tested for its ability of producing distinguishable clusters.

5.1 Algorithm for recommendation engine

The supervised learner models show a lot of promise as a direct method of identifying new potential customers. A classifier algorithm can be trained using a set that contains existing customers, and be applied on a large set of various companies, to classify suitable prospects.

The FSC algorithm shows some promise with respect to accuracy, but the model suffers from lack of complexity, resulting in matches of low relevance. Tests show that the main drawback of FSC is its tendency of having very high bias, causing an evident underfit when applied on large sets of data. FSC is traditionally not counted as a full-fledged machine-learning algorithm, but its simplicity and apparent use in data mining makes it a suitable starting point when entering into the domain of machine learning.

k-Nearest neighbors is simple to implement, easy to understand, and shows great performance in most evaluated areas. However, the model suffers from being computationally expensive when applied on large sets of data. Hence, it becomes impractical in a real-world scenario where classification needs to be performed on a country-wide company database.

Among the evaluated classifiers, C4.5 decision trees shows the most promise. It is fast and accurate, as well as producing matches of high relevancy. The main issue with the

model is its tendency of high variance, which might cause overfitting of the data. Fortunately, our tests show that variance can be decreased by using a higher proportion of positives in the training set (at a small cost of precision). During the evaluation, it was found that there exists an apparent trade-off effect between achieving high precision and high recall. Nevertheless, this trade-off can be easily controlled by proper balancing of the training sets.

A possible use of k-means clustering is to apply the model on sets of existing customers for identifying homogeneous segments. These segments can then be examined with the purpose of identifying relationships to other segments of existing customers. Using these relationships, it is possible to predict members of the associated segments, as possible new customers.

5.2 Future work

The k-NN learner has the property of being easily parallelized. If this property could be properly exploited, the algorithm may prove to be useful in practice. Current techniques for massively parallel execution on GPUs or distributed cloud computing might emerge as possible solutions.

Regarding decision trees much can be done in order to improve performance. High variance can be countered using tree pruning methods, intended to simplify the structure of the tree. Other methods involve ensemble learning, where multiple decision trees can be trained in parallel and used for classification by means of tree majority voting. Ensemble-learning techniques such as, *e.g.*, random forest, and AdaBoost have been proven to increase accuracy (Freund and Schapire, 1999).

Finally, there are various other machine-learning algorithms available. More advanced models, such as *neural networks* (NN), and *support vector machines* (SVM) often have superior performance compared to simpler models. However, they are harder to implement and harder to understand. NN and SVM are often viewed as black-box algorithms using complex self learning structures which are hard to interpret from the outside. Nevertheless, they might be worth looking into when implementing an advanced recommendation engine.

Bibliography

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487--499, Santiago, Chile.

Arthur, D. and Vassilvitskii, S. (2006). How slow is the k-means method? In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 144--153, Sedona, Arizona, USA. ACM New York.

Bayes, T. and Price, R. (1763). An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, communicated by mr. price, in a letter to john canton, m. a. and f. r. s. *Philosophical Transactions of the Royal Society of London*, 53(0):370--418.

Buö, D. and Kjellander, M. (2014). Predicting customer churn at a swedish crm-system company. Master's thesis, Linköpings Universitet.

Chmielewski, M. R. and Grzymala-Busse, J. W. (1996). Global discretization of continuous attributes as preprocessing for machine learning. *International Journal of Approximate Reasoning*, 15(4):319--331.

Coates, A. and Ng, A. Y. (2012). Learning feature representations with k-means. In *Neural Networks: Tricks of the Trade*, volume 7700, pages 561--580. Springer Berlin Heidelberg.

Cover, T. M. and Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21--27.

Domingos, P. (2000). A unified bias-variance decomposition. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861--874.

Forgy, E. W. (1965). Cluster analysis of multivariate data: Efficiency versus interpretability of classification. *Biometrics*, 21(3):768--769.

Freund, Y. and Schapire, R. E. (1999). A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5):771--780.

Geurts, P. (2002). *Contributions to Decision Tree Induction: Bias/Variance Tradeoff and Time Series Classification*. Doctoral thesis, University of Liege Belgium.

Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147--160.

Hastie, T., Tibshirani, R., and Friedman, J. (2013). *The Elements of Statistical Learning*. Springer Science and Business Media.

Hunt, E. B., Marin, J., and Stone, P. J. (1966). Experiments in induction. Technical report, University of Michigan.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer Science & Business Media.

Kostojohn, S., Johnson, M., and Paulen, B. (2011). *CRM Fundamentals*. Apress.

Lantz, B. (2013). *Machine Learning with R*. Packt Publishing Limited.

Legendre, A.-M. (1805). Nouvelles méthodes pour la détermination des orbites des comètes [new methods for the determination of the orbits of comets]. Paris: F. Didot.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707--710.

Lloyd, S. P. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129--136.

Lundalogik AB (2015). www.lundalogik.se. `http://www.lundalogik.se`. Accessed: 2015-02-04.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281--297. University of California Press.

Nugues, P. (2010). *An Introduction to Language Processing with Perl and Prolog*. Springer Publishing.

Orlando, S., Palmerini, P., and Perego, R. (2001). Enhancing the apriori algorithm for frequent set counting. *Lecture Notes in Computer Science*, 2114(8):71--82.

Pazzani, M. J. and Billsus, D. (2007). *The Adaptive Web*. Springer Berlin Heidelberg.

Peel, J. and Gancarz, M. (2002). *CRM Redefining Customer Relationship Management*. Digital Press.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81--106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

Tan, P.-N., Steinbach, M., and Kumar, V. (2006). *Introduction to Data Mining*. Pearson Addison Wesley.

Ungar, L. H. and Foster, D. P. (1998). Clustering methods for collaborative filtering. In *AAAI Technical Report WS-98-08*, pages 114--129, Madison, Wisconsin, USA. AAAI Press.

Weinberger, K. Q. and Saul, L. K. (2009). Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(1):207--244.

Appendices

Appendix A
List of Attributes

#	Attribute	Type	Description
1	id	discrete	Internal identification number
2	postalzipcode	discrete	Postal code
3	postalcity	discrete	City name
4	visitzipcode	discrete	Postal code
5	visitcity	discrete	City name
6	municipalitycode	discrete	Municipal code
7	municipality	discrete	Municipality name
8	countycode	discrete	County code
9	county	discrete	County name
10	postalcountrycode	discrete	Country code
11	worksitestatuscode	discrete	Work site status code
12	worksitestatus	discrete	Work site status, *e.g.*, "active"
13	worksitestypecode	discrete	Work site type code.
14	worksitetype	discrete	Work site type, *e.g.*, "headquarters"
15	worksitecount	continuous	No of physical work site locations
16	lineofbusinesscode	discrete	Line-of-business code
17	lineofbusiness	discrete	Line-of-business name
18	legalformcode	discrete	Legal-form code
19	legalform	discrete	Legal-form, *e.g.*, "Joint-stock company"
20	employeecountworksitecode	discrete	No of employees at work sites code
21	employeecountworksite	discrete	No of employees at work sites (intervals)
22	employeecounttotalcode	discrete	No of employees code
23	employeecounttotal	discrete	No of employees (intervals)
24	sharecapital	continuous	Financial amount
25	turnoverrange	empty	Empty for all records in data
26	visitcountrycode	discrete	Country code
27	rating	discrete	Financial credit raiting
28	employeecount	empty	Empty for all records in data
29	turnover	continuous	Financial amount
30	turnoverperemployee	continuous	Financial amount
31	turnoverrangecode	empty	Empty for all records in data
32	financialinfodividends	continuous	Financial amount
33	financialinfoequityratio	continuous	Financial rate
34	financialinfoquickratio	continuous	Financial rate
35	financialinforesultbeforetax	continuous	Financial amount
36	financialinfodividends	continuous	Financial amount
37	financialinfosalariesboardmembers	continuous	Financial amount
38	financialinfosalariesothers	continuous	Financial amount
39	financialinfoturnovergrowth	continuous	Financial amount
40	financialinfonumberofemployees	continuous	Financial amount
41	financialinfonumberofsubsidaries	continuous	Financial amount
42	basicinfototalturnover	continuous	Financial amount

Table A.1: List of attributes in the test data.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTATIONSDAG 2015-06-04

EXAMENSARBETE Machine-Learning Techniques for Customer Recommendations
STUDENT Felix Glas
HANDLEDARE Pierre Nugues (LTH), Peter Wilhelmsson (Lundalogik)
EXAMINATOR Thore Husfeldt (LTH)

Recommendation Engines in Customer Relationship Management Systems

POPULÄRVETENSKAPLIG SAMMANFATTNING **Felix Glas**

Results show that software sales tools can predict potential new customers using well-known machine-learning algorithms such as: *k*-nearest neighbors, C4.5 decision tree induction, and *k*-means clustering, provided that there is enough data available.

Today, there is a demand for automated procedures for predicting future customers using recommendation engines in the customer relationship management (CRM) market. Imagine you are in sales and you are getting suggestions on possible new customers in the same way that Netflix recommends movies and Spotify helps you find new songs and genres. Machine-learning techniques are behind these high quality recommendations that derive deep insights from large amounts of past customer data. Such techniques can benefit most users of CRM systems, provided that they have gathered enough data to train models so that they can make accurate predictions on their customers. Our results show that relevant prospects can be extracted using off-the-shelf algorithms such as decision trees and *k*-means clustering.

The purpose of CRM is to help companies better manage their customers, present and future. It is an overall strategy for finding promising prospects and keeping existing customers by collecting and refining information about individual companies and learning their behaviors and needs. In today's expanding CRM market, there is a demand for automated procedures that can be used for customer prospecting. There are already functions commonly available for finding "twins", *i.e.*, possible customers that are similar to existing customers, and for browsing through lists of customers partitioned into categories such as locations or lines of business. In the near future, computer algorithms will enable systems to automatically suggest prospects that have a high potential for becoming profitable customers. Such systems need a recommendation engine for making predictions on which companies are relevant.

Current recommendation engines are typically built using machine-learning techniques and the results provided by this project show that relevant customer predictions can be made using common algorithms. During the project, an array of machine-learning algorithms were evaluated using measures of accuracy and relevancy of the data predicted. Using a decision tree classifier based on the C4.5 algorithm we showed that a manageable number of similar customers can be predicted, provided there is a sufficiently large number of existing customers to train on. However, observations show that there is a trade-off effect between getting high precision of relevancy and getting a large number of predictions. This trade-off can nevertheless be controlled by adjusting the complexity of the model. By decreasing the number of existing customers in relation to the proportion of non-customers in the training set, the model will obtain a higher bias, thus shifting the predictions toward being greater in number and of having less relevance. This can sometimes be desirable when targeting new customers more aggressively. Also, by performing *k*-means cluster analysis on the set of existing customers, homogeneous groups of customers can be identified. By storing the relationships between commonly occurring clusters of existing customers, new companies can be given recommendations based on these relationships.