Improving rule-based classification systems by using arguments

Master's Thesis / Internship at Cognicor Technologies

Utrecht University

Author:
S. van Driel

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

Daily supervisor:
Dr. N.D. Rotstein

August 25, 2014

Abstract

Improving the predictive accuracy of rule-based classifiers, specifically the Clark Niblett 2 algorithm, can be done by using an argumentation-based approach. Previous research on this topic focuses on using expert feedback to improve the predictive accuracy. This research has concentrated on providing a working algorithm and limits itself to a few domains which require expert feedback. In combination with writing this thesis an application has been built which makes the practical use of argumentation-based classification a possibility. Experiments are included to show the validity of the techniques in the application and to research the viability of the use of argumentation-based classifiers in a number of other domains. The results show that the argumentation-based approach is solid, but possible future research into using ordered rule-based classifiers as opposed to unordered ones can provide greater benefits. The application and experiments strengthen the validity of argumentation-based classification and enhance its practical usage.

Preface

Work on this thesis started in September 2013, with an internship at Cognicor Technologies in Barcelona. Despite some problems along the way the whole project, including the writing of this thesis, was finished within a year (but barely). The main research question was changed along the way, as were the supervisors associated with the project.

A great number of people, professors, colleagues, friends and family have helped me during the last year. I would like to thank everyone who helped me, in all possible ways, to finish my thesis.

Special thanks go out to my former supervisor, Henry Prakken, who suggested the internship at CogniCor and introduced me to the company. His help in writing the project proposal was invaluable as well.

My current supervisors, Ad Feelders and Arno Siebes, have reviewed endless iterations of this thesis, providing suggestions, and feedback. Their contributions have vastly improved the quality of the research and of my writing, for which I am grateful.

Nico Rotstein, as my daily supervisor at CogniCor, provided feedback on the application and helped guard the scientific validity of the algorithms. His remarks and encouragement and those of our colleagues Menahem Lisei, Alexandra Ligouri and Volker Vorwerk have helped me bring this project to a fruitful conclusion. I heartily thank all of them for their support.

I am also grateful to Sindhu Joseph and Rosh Cherian, the owners of CogniCor, for providing me with the opportunity to built a real world and practical application during my internship.

My sister, Ellinore, proofread this thesis, providing me with insight into the fallibility of my English writings. Together with my other sisters Simone and Judith, she has put up with my complaints and endless stories during the last year. I thank them all for providing me with moral support and entertaining visits to Barcelona during my internship.

My parents, Govert and Liliane, whose visits provided some much needed relaxation during the internship and who both encouraged me to really finish my thesis have been instrumental in getting me here. I want to thank them for all the wise words and helpful comments over the years.

Finally I would like to thank Jessica, for putting up with me during the last year. She helped me keep perspective and positive about everything.

To everyone I have forgotten to mention, and again to all of those named above, thank you for your help, kind words, encouragement and everything else!

Contents

Chapter 1

Introduction

Classification is the task of assigning objects to one of several predefined categories (Tan et al. [2006]) and is regarded as a subtopic of machine learning. Classification problems occur in many different domains, with the respective solutions having equally diverse applications.

Classification uses knowledge contained in previous cases to make a prediction for a new case, similar to other machine learning methods. It is a supervised learning method, as it works on labeled data to produce a prediction for a known variable. There are several methods of classification, among others support vector machines, decision trees, rule-based classifiers and neural networks.

Both support vector machines and neural networks have shown great promise in recent years, but suffer from the fact that they are black box methods, which makes it rather difficult to check the correctness of a prediction. Decision trees and rule-based classifiers both provide a clear and deliberate structure which is used to classify new cases. This structure (a tree or set of rules) can be checked by experts with domain knowledge to verify the consistency and validity.

Although an expert can check the validity of the structure, influencing said structure is a different matter. For the comments of an expert to have any influence an adjustment mechanism is needed. Možina et al. [2007] provides such a mechanism for a rule-based classification system. The rule-based classifier used is an adjusted version of the Clark Niblett 2 algorithm (Clark and Niblett [1989] and Clark and Boswell [1991]).

These adjustments are done in the form of arguments, which allow the expert to differentiate the properties of a specific case. An expert can annotate a case by choosing which properties of the case form positive arguments and which form negative arguments. Positive arguments indicate a good reason for the given solution class according to the expert, while negative arguments indicate properties that are considered contrary to the reasons an expert would use to classify the case with the given solution class.

The real world application of argumentation to classification has only been studied in a small number of experiments (Možina et al. [2007] and Možina et al. [2005]). These papers have looked at the applicability of argumentation-based learning in hindsight, as the results on existing datasets were compared to those of other algorithms. This research was done with a limited number of datasets within a small sample of domains.

The goal of this thesis is to build a usable system for a real life application of argumentation based learning. To achieve this an internship at a company in Barcelona, CogniCor Technologies, was arranged. CogniCor Technologies is active in the automated complaint resolution business. For their solution they needed a classification system that is capable of being influenced by feedback from experts.

During the internship at CogniCor such a system, based on the argument-based Clark Niblett 2 algorithm from Možina et al. [2005], was implemented. This implementation takes the form of an application that has a website and a web service as interfaces. The web service allows the other CogniCor systems to access the classification system with the minimum amount of interaction needed. The system uses a slightly adjusted version of the algorithm and implements several other versions and variants of the Clark Niblett 2 algorithm for comparison purposes.

The algorithms implemented are the LID algorithm (Lazy Induction of Description), a type of case based reasoning classifier, and several variants of the CN2 algorithm (Clark Niblett 2). The variants used are the original CN2 algorithm from Clark and Niblett [1989] and an updated version from Clark and Boswell [1991], both producing a sorted ruleset, and the unsorted version of CN2 algorithm introduced in Clark and Boswell [1991] and the argument-based version described in Možina et al. [2005], both creating unordered rulesets.

After the internship ended the application was used to test the performance of the classification system on several datasets. These results were examined in detail and compared to those of Možina et al. [2005] and Možina et al. [2007]. Some of the datasets from the experiments in those papers were used, with the addition of several new datasets. To compare the performance of the new datasets the results have been compared with the performance of the relevant non-argument based versions of the algorithms. This comparison was done both on predictive accuracy and runtime. Predictive accuracy tests how well a new (previously unknown) case is classified, with the runtime being a reflection of how fast the algorithm does this.

The algorithm and experimental results presented in Možina et al. [2005] already prove that classification performance (defined as the predictive accuracy) can be improved by the use of arguments. This thesis aims to extend the knowledge on this subject by building an application usable in practice and supplementing the experimental results with comparative experiments on datasets from other domains. Thus it looks at whether the classification performance of the CN2 algorithm can be improved by including argument-based expert feedback.

Chapter 2

Theory and literature

The implementation of the classification system rests on previous research done by various sources. Therefore an examination of the existing work on this topic is necessary and enhances the understanding of the choices made in the implementation and experimentation.

The main sources for the implementation of the rule-based classifier are the work of Clark (Clark and Niblett [1989] and Clark and Boswell [1991]) and Možina et al. [2005 and 2007]. These last publications concern the Clark Niblett 2 algorithm and its variations.

This chapter starts with a definition of classification, followed by an exposition on the classification methods used in this thesis. Case based reasoning and rule-based classifiers will be described in more detail, including the specific implementations used in the application built for this thesis.

2.1 Classification

Classification is defined by Tan et al. [2006] as the task of assigning objects to one of several predefined categories. There are a number of algorithms and methods for classification, divided into a couple of categories. These categories include support vector machines, tree-based methods and rule-based methods. This list is not exhaustive, but for the sake of briefness this thesis refrains from giving that full list and the explanation of all those methods.

Support vector machines are one of the black box methods mentioned in the introduction. A support vector machine represents all cases as points in multidimensional space and then searches for boundaries in the same multidimensional space that divide all cases into their respective solution classes as well as possible.

Tree-based classifiers generate a choice tree, which is a tree consisting of choice nodes (i.e., male or female?) with the appropriate attribute on the case that is being classified used to make a decision. This process continues until a leaf node is found, which gives the prediction the classifier makes for that case.

A rule-based classifier is similar, although it uses simple rules in a list to generate a prediction. A distinction can be made between unordered and ordered rule-based classifiers. In an ordered classifier the rules are looked up in order and the first rule that matches is used to make a prediction. An unordered classifier tries to match all rules and makes a prediction based on all rules that match.

Given the research done during and after the internship the most interesting and relevant method of classification are the rule-based classifiers, specifically the Clark Niblett 2 algorithm and its variants. For a complete understanding of the research done and design decisions made in building the classification system, a proper understanding of these classification methods is useful.

2.1.1 Defining classification

A formal definition of classification is given in Tan et al. [2006] as follows: "Classification is the task of learning a target function f that maps each attribute set x to one of the predefined class labels y." This definition makes clear that the basis for the classification decision are the attributes of a case. The attributes are the only information from the case used for classification.

Broadly speaking the classification methods can be divided into two categories, those with a separate phase for building a mechanism for classification and those without. Rule-based classifiers are a good example of the first category, as the method consists of two phases. First a ruleset is built based on the previous cases known to the system. Afterwards new cases are classified based on this ruleset. Support vector machines use a similar technique, where a model of points in multidimensional space is build that encompasses the decision boundaries used in the classification phase.

Classification systems of the other category use all available knowledge from the previous cases at the moment of classification to make a prediction. These methods are used less often as they are slow and often lack several of the features found in other classifiers, such as an explicit model that can be checked by a domain expert.

2.1.2 Classification methods and techniques

As mentioned earlier there are a large number of algorithms and methods used in classification. Providing a full list of these is not a concern of this thesis, thus only the most relevant methods will be named and discussed. The rule-based classification methods, specifically the Clark Niblett 2 algorithm, are the main topic of this research. Another method implemented in the classification system that was developed during the internship is case-based reasoning, which is a relatively simple method to explain, although the speed with which decisions are made is slow compared to more modern and comprehensive methods. This method was implemented as a baseline method, as it is easy to explain and use while still providing good results.

There are several modern machine learning techniques of which the inner workings are hard to explain and which do not necessarily have any correspondence to the real world logic inherent in the domain in which they are used. These methods include support vector machines and neural networks. This thesis focuses on the usefulness of arguments provided by a domain expert to enhance classification accuracy, therefore these methods are not used in this research. As these are not mentioned anywhere else in this thesis they will not be discussed any further. More information on these methods can be found in Tan et al. [2006] or other introductory works on machine learning or data mining.

All classification methods require a so called 'case base', a set of previous cases of which the correct prediction (or solution class) is known. The case base is used, directly or indirectly, to predict the solution class of a new case. The cases in the case base should ideally contain all necessary information and have a correct solution class. In practice however, it is possible that certain values are missing and that some of the solution classes for cases in the case base are wrong. This is called noisy data and some classification methods work better on noisy data than others. The effect of noisy data are shown in Možina et al. [2007], as is the ability of the Clark Niblett 2 algorithm to deal with this type of data.

When talking about classification (and machine learning in general) the cases used to train the model and make a prediction are called the training cases, whilst the case that is to be classified is called the test case. This terminology is also used in cross validation, a technique discussed later in this thesis. This terminology is kept when discussing the usage of the system in a real life environment in which prediction is the goal. Here the cases for which the solution is already known are part of the case base and thus seen as the training cases, while the incoming case, for which a prediction has to be made, is the test case.

2.1.3 Example

To give insight into the methods used in this paper an example is used. This is an example from Možina et al. [2007] and concerns credit approval. A bank needs to decide whether or not someone should be allowed a loan. To do this the bank uses a number of properties of the person requesting the credit. The example contains some logical properties and a completely irrelevant one. The usefulness of this will be proven later on. The example case base can be found in table 2.1.

Name	PaysRegularly	Rich	HairColor	CreditApproved
Mrs. Brown	no	yes	blond	yes
Mr. Grey	no	no	grey	no
Miss White	yes	no	blond	yes

Table 2.1: Case base used for the examples in this thesis, taken from Možina et al. [2007], concerning credit approval.

2.2 Case Based Reasoning

Case based reasoning (CBR) is a classification technique without an explicit model. All knowledge of previous cases is used to generate a prediction at the moment a new case must be classified. It is a relatively simple technique and provides a good baseline to measure the performance of the other classification methods against.

The technique originates in the work of Schank [1983], with the terminology stemming from Aamodt and Plaza [1994]. A recent in depth discussion of the technique can be found in Riesbeck and Schank [2013].

2.2.1 Technique

Case based reasoning works by comparing the properties of a test case to the cases in the case base. When a new case comes in the algorithm starts looking at the different properties of the case. A comparison is made with previous cases to check the similarities between cases. In the end the algorithm ends up with a subset of cases in the case base which have a high similarity with the new case. These cases are then used to make a prediction about the solution class of the current case.

Take for example a new person coming to the bank for a loan. Miss Orange pays her bills regularly, is not rich and has brown hair. Considering the example case base from table 2.1, should the classifier predict that she should receive credit or not? The fact that she has brown hair cannot help the classifier, as there are no previous cases with that hair colour. Thus the classifier will look at the other properties. She is not rich, but this does not provide a clear prediction as we have two poor people in the case base, with one receiving credit and the other not. Finally the fact that she pays regularly is studied. Only one other person in the case base does this, Mrs. Brown. Thus on this property only on other case exists, whose credit was approved. A CBR classifier would thus predict that Miss Orange should receive credit, because of the similarity between her case and that of Mrs. Brown (both pay their bills regularly).

The advantage of this method is twofold. The simplicity of the algorithm makes it easy to see why a certain prediction is reached as the similarity between the test case and the cases in the case base can be clearly shown. It thus allows for a clear base line when comparing the performance of other algorithms. The biggest disadvantage is speed, because the algorithm iterates over all properties and cases in the case base at least once, but often multiple times. Therefore it is a slow algorithm and often not useful in a practical situation where a fast response is needed or would be preferable.

When calculating the similarity between the test case and cases in the case base a measure indicating this similarity is needed. This measure is referred to as the similitude term. The technique used for determining the similitude score can be varied, with corresponding differences in results. Most often a measure will be used that prefers the most lopsided class distributions, i.e., those class distributions which have the highest percentage of a single solution class. It is possible to use systems that focus on large numbers as well, although this is done less often.

2.2.2 Implementations

It is possible to build a CBR classifier in several different ways. One of the more efficient (and often used) is the implementation described in Armengol and Plaza [2001], which is the so called LID-algorithm, which stands for Lazy Induction of Descriptions. Current research into the LID algorithm can be found in Armengol and García-Cerdaña [2010].

The algorithm introduced in Armengol and Plaza [2001] is one of the first to make use of a similarity comparison between a test case and the case base for case based learning. The paper refers to this similarity comparison as the similitude term. For the LID algorithm an adapted version of the López de Mántaras (RLM) distance is used for this comparison.

For each property the algorithm looks at the subset of training cases which have the same value for the attribute as the test case. These subsets are then assigned a value based on their distinctiveness. This distinctiveness encompasses the proportion of those cases belonging to a single solution class. These distinctiveness values are compared and the most distinctive selector (attribute and value combination) is chosen.

The chosen selector is then used on the case base to select a subset of the training cases which have the same value for the attribute as the selector (and the test case), these cases are said to be subsumed by the selector. The algorithm then continues by looking at all properties again (except the ones already used to subsume a part of the case base) and calculating the distinctiveness values on the new subset of the case base.

The algorithm continues until a stopping condition is met. There are several stopping conditions possible, of which three are used in the algorithm implemented later on. The first is when the selected subset of the case base contains cases with only one class. In this case the class that should be predicted for the test case is clear and further specification of the selector does not give any advantages.

Another stopping conditions is when none of the selectors which are tested subsume any cases. If this happens the algorithm would not be able to make a prediction after the next selection and therefore stops.

The last stopping condition is when there are no more properties to add to the selector. This happens when there are training cases which are identical to the test case. Because of the first stopping condition however, this only occurs when there are at least two training cases identical to the test case and these must have different solution classes. This does mean that there is either a wrong prediction in the case base or the information contained in the attributes is not sufficient to classify a case.

In the latter two stopping conditions the algorithm terminates whilst the current selector subsumes a set of the training cases in which at least two solution classes occur. There are several possibilities for determining the prediction for the test case, but the most often used one is also the simplest, majority voting. In this case the algorithm predicts the class most often occurring in the set of subsumed cases. In case of a tie the solution class that most often occurs within the complete case base is chosen.

To find the best selector to use a score is assigned to each possible selector, the earlier mentioned similitude term. In the case of the LID algorithm this is the RLM distance. The RLM distance is calculated as the similarity between the correct partition and the current partition. The correct partition is the partition in which all the cases with the majority class are subsumed and none other. Thus a higher RLM distance indicates that a partition (selection of cases) is more lopsided and thus more confident when used as a prediction criterion.

Formally, given two partitions P_i and P_c of the cases base B, where P_i is the current partition to calculate the similitude for and P_c is the correct partition, the RLM distance is computed as follows:

$$RLM(P_i, P_c) = 2 - \frac{I(P_i) + I(P_c)}{I(P_i \cap P_c)}$$

where

$$I(P_i) = -\sum_{j=1}^{n} p_j \log_2 p_j; \qquad p_j = \frac{|B \cap S_{ij}|}{|B|}$$

$$I(P_c) = -\sum_{k=1}^{m} p_k \log_2 p_k; \qquad p_k = \frac{|B \cap C_k|}{|B|}$$

$$I(P_i \cap P_c) = -\sum_{j=1}^{n}\sum_{k=1}^{m} p_{jk} \log_2 p_{jk}; \quad p_{jk} = \frac{|B \cap C_k \cap S_{ij}|}{|B|}$$

The terms $I(P_i)$ and $I(P_c)$ denote the information contained in the current partition and the correct partition respectively.

The LID algorithm used the stopping conditions described under section 2.2.1 virtually unaltered, leaving the algorithm itself relatively simple. The following pseudocode describes the algorithm used:

Algorithm 2.1 LID algorithm

```
function LID($S_D, p, D, C$)
    if stopping-condition($S_d$)
        then return $class(S_D)$
    else
        $f_d =$ select-leaf($p, S_D, C$)
        $D' =$ add-path($\pi(root(p), f_d), D$)
        $S_{d'} =$ discriminatory-set($D, S_D$)
        LID($S_{D'}, p, D', C$)
end
```

The select-leaf procedure determines the best attribute to add to the selector at this point in the algorithm. It uses the RLM distance to compute similitude terms for each of the attributes that have not yet been used for the current selector. The add-path procedure adds this new attribute-value combination to the current selector. The discriminatory set finds the part of the case base that is subsumed by the new selector.

The stopping-condition is used to check if any of the stopping conditions mentioned before hold at this point in the program. If this is the case a result needs to be returned. The $class(S_D)$ is the majority class of the cases currently subsumed by the selector.

Given all this information an example is in order. Miss Orange, our previous example, is used again. When the algorithm starts all cases in the case base are subsumed. A score is assigned to each attribute-value combination of the Miss Orange case. The fact that her hair is brown gives zero other cases with this attribute-value combination. This selector is thus not evaluated any further. The second possible selector, PaysRegularly, gives us a better score than the third, Rich. PaysRegularly gives us 1 subsumed case, which means that the set of subsumed case has a perfect score (all cases have the same solution class), while Rich gives us 2 subsumed cases, with a 1 to 1 split on the correct prediction. Thus the LID algorithm would use the PaysRegularly property to predict that Miss Orange should receive credit.

In a more complex example image Mrs. Blue, who does not pay regularly, is not rich and has white hair. Both PaysRegularly and Rich would provide us with two subsumed cases (with different solution classes). The algorithm would choose one of the two at random (as both get the same RLM distance) and then continue with that subset of cases. In the next step the other of the two attributes would be used to select Mr. Grey as the most similar case, resulting in the LID algorithm predicting that Miss Blue should not receive credit.

2.3 Clark Niblett 2 and variants

The Clark Niblett 2 algorithm (hereafter referred to as CN2) is a rule-based classifier that is capable of creating both ordered (Clark and Niblett [1989]) and unordered rulesets(Clark and Boswell [1991]). It has been used by Možina et al. [2005] to demonstrate the possibility of using argumentation in the classification process.

CN2 is a rule induction algorithm designed for the efficient induction of simple and comprehensible production rules in domains where noise might be present. This is also the case for most environments where argument-based classification is likely to be used, as the necessity of using an argument-based system is generally determined by the ambiguity or noisiness of the data. Thus CN2 is well suited for this type of environment.

There are two main versions of the CN2 algorithm. The version from Clark and Niblett [1989], in which the CN2 algorithm was introduced, produces an ordered list of rules. This results in rules consisting of few attribute-value combinations which makes it easier for a human to understand those rules. The disadvantage in this approach is that to understand a classification not only is the rule used to classify the case necessary for understanding the system, but also all of the rules above the current one. These rules have not fired, thus the negation of all of these rules holds true. Without this information the rules often seem counterintuitive and oversimplified.

To address this shortcoming Clark and Boswell [1991] introduced a CN2 variant which produces unordered rulesets. With this rule generation method longer rules are produced, as more information needs to be contained in the rule to properly select a subset of the case base. Given that only the firing rules are necessary, in general the set of rules used to make a prediction are more understandable for the expert than the ordered rulesets. Using unordered rules means that it is possible for multiple rules to fire when given a case. Thus a mechanism is needed to deal with this when it occurs.

2.3.1 Technique

The original algorithm was designed with three main goals mind: *a*) accurate classification; *b*) simple rules; and *c*) efficient rule generation. The ID3 algorithm of Quinlan [1986] and the AR algorithm of Michalski [1969] have served as an inspiration and parts of both algorithms are used for the CN2 algorithm. The algorithm produces an ordered ruleset, with the process of generating this ruleset described below. Further on a look will be taken at unordered rulesets and the process to create these.

To understand the technique behind the Clark Niblett 2 algorithm a few definitions are in order. In the previous section the selection of one or more cases based on an attribute-value combination was introduced. Such a combination is called a condition. A condition consists of an attribute and an equality on a value. Other algorithms sometimes use greater than and other relations between attribute and value, but for the CN2 algorithm only equalities are used. A conjunction of conditions is called a complex or selector and is used to subsume (select only those cases conforming to it) cases from the case base. As a complex is a conjunction of conditions the empty complex subsumes all cases. Extending a complex with a new condition is referred to as specialisation, a technique used to find candidate complexes.

Putting the above into an example makes for a clearer explanation. Given the example introduced in table 2.1 the aim is to generate a complex that only contains Mrs. Brown. Starting with the empty complex, which covers all three cases, possible specialisations are all attribute-value combinations in the case base. The condition ($\texttt{PaysRegularly} = \texttt{no}$) is used to specialise the complex, which now only subsumes Mrs. Brown and Mr. Grey. Specialising the complex again, with the condition ($\texttt{Rich} = \texttt{yes}$), results in a complex that only subsumes Mrs. Brown. The final complex is ($\texttt{PaysRegularly} = \texttt{no} \cap \texttt{Rich} = \texttt{yes}$).

The CN2 algorithm works in an iterative fashion, each iteration searching for a complex covering a large number of examples of a single class C and few of other classes. An evaluation function is used to determine if the found complex is both predictive and reliable.

Having found a good complex, all covered examples are removed from the case base for the rest of the ruleset generation and the complex is added to the ruleset in the form 'if complex then predict class C'. The removal of all cases only applies to the ordered rulesets, the process for unordered rulesets is slightly different and is explained later on. This process iterates until no more good complexes can be found or the case base is empty. The system searches for complexes (selectors) in a general to specific search, as the evaluation functions value bigger sample sizes higher then smaller, if the class distribution in both is the same.

The difference with the process for CBR is that not only the best complex is used but the algorithm keeps track of a number of best complexes. Each of these complexes is crossbred with all potential selectors. Using this approach means that complexes that would be valuable but consist of multiple selectors can still be used as a potential complex. The CBR algorithm would miss these combinations, as it directly chooses the best selector according to its evaluation function. The CN2 algorithm keeps track of these best candidates in a size-limited set or *star* S of best complexes found so far.

Besides the current star, which only contains complexes with the same number of conditions (determined by the current iteration), the algorithm also keeps track of the single best solution up to that point. This tracking is separate from the star and used to select the best complex when the algorithm for finding the best complex terminates.

In the algorithm two evaluation functions are used. A function is used to evaluate the quality of a complex, both for inclusion in the star and to determine if it is better than the current best complex. In Clark and Niblett [1989] the information-theoretic entropy measure is used, while Clark and Boswell [1991] uses the Laplace expected error estimate. Both functions and their difference will be explained in the next section.

The second evaluation function concerns the significance of the found complex. The significance of a complex expresses the likelihood with which the complex could exist by chance. Thus a high significance signifies that the complex locates a regularity unlikely to have occurred by chance and that it reflects a genuine correlation between the attribute-value combination and the solution class. The algorithm of Clark and Niblett [1989] uses the likelihood ratio statistic to test significance.

Because of differences between the quality evaluation function used in the publications the role of significance testing is slightly different in the separate versions of the algorithm. In the original algorithm significance testing is used to select more general rules. The second version (of Clark and Boswell [1991]) uses a quality measuring function that is already predisposed towards more general rules. Therefore significance testing has more influence on the stopping point of the improved version than it has on the complexes selected.

The type of rules produced in the CN2 algorithm is slightly different from that in the UCN2 version. As the rules of the first are ordered, it is important to store their ordering. For the second this point is moot of course. However, the unordered rules store not the majority class for the found rule but the class distribution found. Doing this allows the prediction process to produce more accurate results and to keep track of the accuracy of the generated rule.

2.3.2 Implementations

The paper from Clark and Boswell [1991] introduces several changes to the original CN2 algorithm, resulting in several different possible implementations of the algorithm. The pseudocode for generating the ordered rulesets is the same, but the quality evaluation function is different. The paper also introduces a slightly different version of the algorithm for generating unordered rulesets, for which a different generation procedure is needed. The pseudocode given in Clark and Niblett [1989] for generating the ordered ruleset can be found in algorithm 2.2.

The pseudocode in algorithm 2.2 is written to explain the inner working of the algorithm. Most readers will immediately spot several improvements that should be implemented when this code is actually used, such as first ordering the complexes and then comparing only the best of the new star with the best complex. The details on the actual implementation of the algorithm in the application will be left to chapter 3.

As mentioned in the technique section on CN2 we see several user supplied parameters and functions that influence the efficiency of the algorithm, each of which will be discussed below. We have the evaluation function for the quality of complexes, the statistical significance function for complexes, the star size to keep between iterations and the minimum statistical significance. All these will be discussed for the original CN2, afterwards we will focus on the differences with the other versions of the algorithm.

The evaluation of quality function for complexes determines which measure is used to determine the best complexes in the current iteration. In the original CN2 algorithm this is the information-theoretic entropy measure, given by the formula:

Algorithm 2.2 CN2 algorithm

```
function CN2(E)
    rule_list = empty list;
    repeat
        best_cpx = find_best_complex(E);
        if (best_cpx != nil)
            E' = all examples covered by best_cpx;
            remove from E all cases in E';
            C = most common class in E';
            add "if best_cpx then class = C" to rule_list;
    until (best_cpx == nil || E == empty)
    return rule_list;

function find_best_complex(E)
    star = new set containing only the empty complex;
    best_cpx = nil;
    selectors = set of all possible selectors;
    while (star != empty set)
        new_star = {x ∧ y | x ∈ star, y ∈ selectors};
        remove all complexes from new_star that are
            in star (complex ∈ star) or
            are null (i.e. big = e ∧ big = y);
        for (complex in newStar)
            if (statistical_sig(complex) > user_def_min &&
                quality(complex) > quality(best_cpx));
                best_cpx = complex
        order new_star (in desc order of quality);
        keep best n complexes from new_star;
        star = new_star;
    return best_cpx;
```

$$\text{Entropy} = -\sum_i p_i \log_2(p_i)$$

With p_i indicating the part of the case subsumed by the complex belonging to that class and thus a lower entropy is indicative of a higher quality of the complex. Note that this function does not solely focus on the best class distribution at the moment but takes into account the possibilities when specialising the complex.

Take the following two distributions, P = (0.7, 0.1, 0.1, 0.1) and Q = (0.7, 0.3, 0, 0). The above function will prefer the second to the first, while there is no difference in actual usage. One could even argue that the first is indicative of noise in the data, while the second is not specialised enough. The reasoning behind choosing this evaluation function can be seen when we specialise the complex to exclude all cases of the first class, resulting in the new distributions P = (0, 0.33, 0.33, 0.33) and Q = (0, 1, 0, 0). The reasoning for making the second class distribution preferred can now be clearly seen.

The second evaluation function involves significance testing. According to Clark and Niblett [1989]: "By [significance] we refer to a complex that locates a regularity unlikely to have occurred by chance, and thus reflects a genuine correlation between attribute values and classes." The choice of function and the user defined minimum are closely related, as the choice of minimum depends on the function used.

In the original CN2 function the chosen function is the likelihood ratio statistic, which is given with the following formula:

$$\text{Significance} = 2\sum_{i=1}^{n} f_i \log(f_i/e_i)$$

With F being the observer frequency distribution of the cases subsumed by a complex and E being the expected frequency based on random sampling on the whole case base. A higher significance is indicative of a lower chance of the complex being a random occurrence, thus necessitating a minimum to be defined by the user.

This choice of minimum is somewhat difficult as there are no clear guidelines. The probability of a certain pattern occurring also differs based on the number of cases in the case base (thus duplicating every case in the case base means that the minimum significance needs to be adjusted; it needs to be raised to produce an identical ruleset).

The final parameter to the algorithm is the star size. This is the number of complexes kept between iterations. A bigger star size leads to a higher number of potential complexes, thus taking more time to compute. It can also lead to a higher accuracy as it allows the algorithm to find complexes of which supersets are not that good but the complex itself is highly specialised and of good quality. Clark and Niblett [1989] state that in general a star size of 15 is sufficient. Taking a star size lower than this threshold results in most instances of the algorithm losing accuracy, whilst a higher star size does not benefit results, but raises the runtime.

Augmented Clark Niblett 2

In the augmented version of the algorithm, proposed in Clark and Boswell [1991], slight changes to the algorithm have been made. These include using a different version of the function for determining the quality of complexes. This changes the role of significance testing, although it is kept in the algorithm.

The original function used, information-theoretic entropy measure, has a problem that occurs in more algorithms (such as AQ15 and ID3). The problem is that these quality measures have a perfect score on a complex that is highly specialised, covering just one case. This behaviour is unwanted, as it leads to highly specific rules covering few cases. These cases could be an effect of noise in the data, thus leading to an accidental rule which has no resemblance to the logic in the real world application.

The way in which this problem is mitigated (but not solved) in the original CN2 is by using a significance testing method, as described above. An example given in Clark and Boswell [1991] to illustrate the fact that this problem remains is the following:

Consider a domain with two equally likely classes, C_1 and C_2. Also consider three possible rules R_1, R_2 and R_3 with the following class distributions:

R_1 covers 1000 examples of class C_1 and 1 of C_2 [1000,1]

R_2 covers 5 examples of class C_1 and 0 of C_2 [5,0]

R_3 covers 1 example of class C_1 and 0 of C_2 [1,0]

We would want the algorithm to prefer R_1 as the accuracy on new data is likely to be best, given that the other two rules only cover a very small part of the case base. However, a 99% significance test would eliminate R_3 but not R_2, thus the algorithm would prefer R_2. We could raise the significance level further, but we can also think of a new example rule (say $R_{1.5}$) which is just above this threshold but has the same problem.

Generally speaking, we state that the measure used (information-theoretic entropy measure) and others like it have a downward bias; they prefer rules that are more specific, as they are often (slightly) more accurate.

We can use a different quality measure to circumvent this problem, specifically one that prefers more general rules. The measure used by Clark and Boswell [1991] is the Laplace expected error estimate. This quality measure is described by the following function:

$$\text{Laplace Accuracy} = \frac{n_c + 1}{n_{tot} + k}$$

where

k is the number of classes in the domain

n_c is the number of examples in the predicted class c covered by the rule

n_{tot} is the total number of examples covered by the rule

The predicted class when generating and evaluating the rules is simply the class which contains the majority of the covered cases. This means that the role of significance testing changes. In the original algorithm significance testing prunes out the more specialised rules, leading to less complexity in the rule list at the expense of a slight reduction in accuracy. This behaviour changes when the Laplace expected error estimate is used as the quality measure, as more general rules are already preferred. Thus raising the significance threshold alters the point at which the algorithm stops searching for further rules. This means that significance testing becomes solely a stopping criterion in this version of the algorithm.

Unordered Clark Niblett 2

The goal of the UCN2 algorithm is to produce, instead of ordered, unordered rules. This necessitates some changes to the algorithm itself. The research in Clark and Boswell [1991] has indicated that using the Laplace expected error estimate improves the algorithm. This version of the algorithm already takes those changes into account.

The changes to the algorithm focus on the control procedure, whilst the actual search procedure stays very similar (with the exception of the quality measure). For a full overview both parts of the algorithm are produced in algorithm 2.3.

The adjustment to the quality function concerns a change in the class that is used as the predicted class in the Laplace measurement. In ACN2 this is the majority, but in UCN2 this should be the class for which we are generating rules.

Concerning the control procedure there are several changes. First of all the search for the rules is now done separately for each class. There is also a change in the cases removed from the case base. In the ordered algorithm all cases covered are removed, including the cases that would be wrongly predicted by the rule. The unordered algorithm only removes those cases for which the solution class was predicted correctly, so the other cases still influence the rules found further on.

The rules produced by the algorithm should not just save the majority class of the applicable cases. As the rules are unordered it is possible that multiple rules fire when a new case is tested. To deal with this the class distribution of all applicable cases is noted when creating the rule.

When a new case enters the system, all rules that are applicable fire. For each rule the class distribution is returned. These class distributions are then summed and from that total the majority class is taken as the class to predict for the new case. Thus more general rules carry a bigger weight when predicting the solution class for a new case than specialised ones.

2.4 Argument Based Clark Niblett 2

The argument-based version of the Clark Niblett 2 algorithm is based on the unordered version. The original algorithm is taken from Možina et al. [2005], with Žabkar et al. [2006] and Možina et al. [2007] providing adjustments and further experiments. In essence the algorithm should function exactly the same on cases that have not been argumented, whilst using the arguments on cases that have been annotated to improve its prediction accuracy. To do this several changes to the algorithm need to be made.

Algorithm 2.3 UCN2 algorithm

```
function UCN2(examples, classes)
    rule_set = empty set;
    for each class in classes
        rules = UCN2_one_class(examples, class)
        rule_set = rule_set + rules;
    return rule_set;

function UCN2_one_class(examples, class)
    rules = empty set;
    repeat
        best_complex = find_best_complex(examples, class);
        if (best_complex != null)
            rules = rules + "if best_complex then class";
            remove all examples covered with the predicted
                class;
    until (best_cond == null)
    return rules;

function find_best_complex(examples, class)
    star = new set containing only the empty complex;
    best_cpx = nil;
    selectors = set of all possible selectors;
    while (star != empty set)
        new_star = {x ∧ y | x ∈ star, y ∈ selectors};
        remove all complexes from new_star that are
            in star (complex ∈ star) or
            are null (i.e. big = e ∧ big = y);
        for (complex in newStar)
            if (statistical_sig(complex) > user_def_min &&
                quality(complex) > quality(best_cpx));
                best_cpx = complex
        order new_star (in desc order of quality);
        keep best n complexes from new_star;
        star = new_star;
    return best_cpx;
```

To adjust the algorithm into taking argumented cases into account the definition of covering as used in the other versions needs to be adjusted as well. This guarantees that arguments are used and not just ignored when another rule would already classify that particular case.

Having experts provide feedback on a system can be useful in general, as the connection between the model produced and the real world situation can be checked. How we use this feedback to improve the model is not immediately obvious. One possible line of thinking would be to have the experts make adjustments to the model until they are satisfied with it. Unfortunately this turns out not to be very useful, as a primary reason to use a classification system is that it can be rather difficult for a human operator to keep track of all possible situations and their associated predictions.

If the experts need to produce their knowledge into rules or a model that is then used as the basis of a classification system another problem rears its head. In this case the major problem is that humans in general have difficulty in properly generalising their knowledge.

To avoid the above two problems when soliciting feedback from experts on a classification system Možina et al. [2007] have devised a method in which experts are asked to provide feedback on one specific case at a time. This circumvents the generalisation problem by only asking for a specific situation. Secondly the feedback from the expert is then incorporated into the system in such a way that the algorithm has the possibility to generate rules that contradict or ignore this information if necessary.

The feedback experts provide on a case is given in the form of arguments. An expert is presented with a complete case, encompassing the case, its decision and all relevant information and attribute-value combinations. The expert then analyses the case and provides arguments on the specific attributes. These arguments can be positive or negative. An expert can also leave an attribute unannotated. Arguments are used here to enhance the prediction accuracy of a classifier, as suggested in Modgil et al. [2013].

A positive argument indicates that, according to the expert's knowledge, this attribute-value combination is a good reason for deciding the case has the current solution class. It is possible for an expert to provide multiple positive arguments, effectively indicating that one or more of these reasons could be used to predict the specific solution class. Positive arguments can be seen as having a "because" relation with respect to the case and the solution class.

Negative arguments are the exact opposite of positive arguments. When an expert annotates an attribute with the information that it is a negative argument the expert indicates that this attribute-value combinations should not be used to predict the current solution class. Effectively the expert says that the information given is contradictory or at least completely irrelevant to the solution class. Thus negative arguments can be seen as having a "despite" relation with respect to the case and the solution class.

Some arguments may not be either according to the expert. These attributes could provide information in making a decision, but the expert does not immediately see the connection or does not think the connection is strong enough.

The case base from table 2.1 is used to provide an example. An expert is asked to annotate a case, specifically that of Miss White. Miss White pays her bills regularly, is not rich and has blond hair. She has received credit, which is considered the correct solution class. When annotating this case

`PaysRegularly` = `yes` can be annotated with a positive argument, as it is information that would justify providing credit to Miss White. The attribute-value combination `Rich` = `no` can be annotated with a negative argument, as the fact that Miss White is not rich should not be a reason for providing her with credit. The last attribute, `HairColor` = `blond` can be left unannotated, as it has no bearing on the decision to provide Miss White with credit. On the other hand an expert could annotate this attribute with a negative argument as it should not be used as a reason for providing credit. This ambiguity can make annotating cases somewhat difficult, although most experts would classify the last attribute as a negative argument.

When an expert has annotated cases the algorithm is run again, which provides the expert with a new set of cases to annotate. If the annotations are correct and there is some improvement to be made to the predications made by the current model a new iteration should provide a better fitting model and improved predictive accuracy.

Of course the algorithm also needs to decide which cases an expert should annotate. Having the expert annotate all cases in the case base would suffice, although this is highly unpractical and inefficient. In the current version of the algorithm a number of cross validation runs is done, with the results being used to find the cases that are predicted wrong most often by the algorithm. These cases are presumed to be a weak point in the model and thus the expert is asked to annotate these.

Thus, when executing ABCN2 there are three parameters above and beyond those used in UCN2 (star size and minimum significance). First of all there are the settings for the cross validations to find cases to annotate. The number of k-fold cross validations and the size of k can be set. Finally a user can set the amount of cases to annotate per run of the algorithm. In general we will use five 5-fold cross validations to decide which single case is performing the worst. This case is then given to an expert to annotate.

2.4.1 Technique

As described above ABCN2 needs to deal with annotated cases separately from non-annotated cases, while still producing rules in the same manner as UCN2. To cope with this several changes are necessary. First of all the definition of covering should be adjusted to take into account annotated cases and what to do with them. Secondly the procedure for generating rules should be adjusted to look at argumented cases first.

The first adaption, extending the definition of covering, makes sure that argumented cases are only covered when their arguments have been used properly as well. Remember from the section on the CN2 algorithm that a complex or selector is a set of attribute-value combinations. The complex covers or subsumes a case when the case has the same value as the complex for each attribute-value combination in the complex. Thus a case is covered when all of the information described in the complex matches the case.

When we look at argumented cases it is clear that some adjustments need to be made. First of all a complex is used as the set of conditions for a rule to fire. In describing what a negative argument entails we have stated that a negative argument is an attribute-value combination that is information-wise contradictory to the solution class. Thus it is logical that none of the negative arguments can be used in a complex to subsume the case that has such a negative argument.

Besides negative arguments we also have positive arguments, which are a good reason, according to the expert, for assigning the case its solution class. Given that a case can have multiple positive arguments we should require at least one of the positive arguments to be present in the complex which is to subsume said case.

To formally define the above two conditions we extend the original concept of covering to that of AB-covering. We define AB-covering as a combination of the following requirements:

1. All the conditions in the complex are true for the case under consideration (same as in CN2).
2. The complex is consistent with (contains) at least one positive argument of the case.
3. The complex is not consistent with any of the negative arguments of the case.

The adjustment made to the covering mechanism helps in classifying the case an expert has annotated, but it does not influence the overall outcome significantly. To this purpose we also adjust the mechanism for finding the rules. When looking for rules applicable to a solution class we will start by considering the argumented cases one by one. For each argumented case we need to find a rule covering that case first, before moving on. Based on the rules we find we already remove positively covered cases from the case base, keeping in mind that we use AB-covering.

With this last adjustment we make sure that annotating cases has enough influence on the process to adjust the rules generated by the algorithm. The quality scores of complexes are not adjusted, thus annotated cases do not weigh heavier in evaluation than their non-annotated counterparts.

2.4.2 Implementation

With the above adjustments to the technique in mind we can now produce a new algorithm in pseudocode that brings this algorithm in practice. The algorithm is similar to that of UCN2 [Clark and Boswell, 1991]. Only the last two procedures need to be adjusted, the `UCN2` procedure can be kept the same. The pseudocode in algorithm 2.4 is taken from Možina et al. [2007].

At the end of the per class procedure we see a call to the same procedure from the UCN2 algorithm to find rules for the remaining cases. We use the code from the UCN2 section above, with one modification. When we talk about covering we replace this with AB-covering. Thus to cover an annotated case the complex has to comply with the restrictions on positive and negative arguments.

From the code we can see that the algorithm focusses on annotated cases first. After iterating over these cases and finding rules that classify them correctly the old UCN2 algorithm is used to find appropriate rules for the remaining cases. Thus considerable value is given to the expert's opinion, because the rules that comply with that opinion are generated first and afterwards the algorithms creates rules to classify the other cases.

Algorithm 2.4 ABCN2 algorithm

```
function ABCN2(examples, classes)
    rule_set = empty set;
    for each class in classes
        rules = UCN2_one_class(examples, class)
        rule_set = rule_set + rules;
    return rule_set;

function ABCN2_one_class(examples, class)
    rules = empty set;
    arg_cases = list of annotated cases;
    for each case in arg_cases
        evaluate all positive arguments in case
            with the quality evaluation function;
    sort arg_cases by their best arguments;
    while (arg_cases != empty)
        ae1 = arg_cases[0];
        rule = ab_find_best_rule(examples, ae1, class)
        add rule to rules;
        remove all AB-covered cases from arg_cases;
    for each rule in rules
        remove all AB-covered cases from examples ;
    non_arg_rules = UCN2_one_class(examples, class);
    add non_arg_rules to rules;
    return rules;

function ABCN2_find_best_complex(examples, arg_case class)
    star = new set containing all positive arguments of
        arg_case as complexes;
    best_cpx = best complex in star;
    selectors = set of all selectors from arg_case;
    arg_reasons = set of all positive arguments of arg_case;
    while (star != empty set)
        new_star = {x ∧ y | x ∈ star, y ∈ selectors};
        remove all complexes from new_star that are
            consistent with negative arguments of arg_case;
        for (complex in newStar)
            if (statistical_sig(complex) > user_def_min &&
                quality(complex) > quality(best_cpx));
                best_cpx = complex
        order new_star (in desc order of quality);
        ab_new_star = all complexes from new_star containing
            only selectors from arg_reasons;
        keep best n complexes from new_star;
        keep best n complexes from ab_new_star;
        star = new_star + ab_new_star;
    return best_cpx;
```

The time complexity of finding rules covering the annotated cases is significantly less than that for non-annotated rules. As we can see above, by focussing on one case at a time we only have one value per attribute to consider. Combining this with the fact that at least one positive argument has to be in the complex means that a great number of potential complexes are never checked, which saves time. Therefore we do not have to check if a complex covers at least one case as it always covers the annotated case.

Finally we see a slight adjustment in the algorithm compared to the description above. Besides the normal star of best complexes found up until that point the algorithm also keeps track of a star of complexes containing only positive arguments. This is done to make sure that a potential combination of positive arguments is always checked, ensuring that the viability of the expert's opinion is taken into account.

Chapter 3

Architecture

As mentioned in the introduction a part of the thesis research was an internship at CogniCor Technologies. During the internship an application was built that implemented a classification system based on the algorithms mentioned in chapter 2. In this chapter the architecture used to implement the system, including the choice of language and framework, is discussed.

The application was built as a part of a larger system which is concerned with automated complaint resolution. Currently calling a help desk can be an arduous experience which often frustrates customers. The idea behind the CogniCor product is to largely dispense with the human element in complaint resolution, thereby making the complaint process easier and more customer friendly. At the same time it is an opportunity for companies to reduce their expenses in this department by doing away with most of the manual work.

The full suite of software works by having the users fill their complaint into an online form. This form is then parsed by a natural language processor that was built specifically for the domain and the language in use. The information gained from the complaint is combined with information already available within the company for that customer, such as the type of contract, remaining duration and monthly billing details. Relevant parts of the data are then sent to the classification application, which uses the classification system to decide what the best solution is for the given case. Afterwards the outcome is communicated to the customer in natural language by way of natural language generation software developed for this purpose. Thus the CogniCor classification application should be able to produce the correct solution class as often as possible when used on the data that normally enters the system.

When discussing the architecture of the system several aspects need to be highlighted. First, the choice of programming language and web development framework will be shortly discussed. The setup of the system in terms of the data structures follows, after which the implementation of the different algorithms will be discussed. Then an overview of the unrelated parts of the application that were implemented, such as a user verification system, is given. The chapter finishes with set of screenshots to provide the reader with a clearer picture of the user experience of the application.

3.1 Ruby

At the start of the internship several programming languages and technologies were already in use at CogniCor, amongst others Java and Python. During the explorational phase in which the goals and requirements of the project were discussed several possible choices of programming language and web framework were compared. Python and Django were a possibility, whilst Java and Spring were considered too heavy duty given the current status of the company and the software. During the discussions the possibility of using Ruby on Rails was mentioned.

Ruby is a programming language which focusses on ease of use for the programmer. One of the cornerstones of Ruby programming is the mantra "by convention, not by configuration". In practice this means that whilst things can often be done in several ways the language is built with one particular way in mind. This means that the Ruby community sees this way as the right way. At the same time this makes programming in Ruby, when adhering to these conventions easy and hassle-free.

The syntax is similar to Python, with some changes to make the language more readable and more intuitive to write. The concepts used in designing Ruby are taken from a number of other programming languages such as Smalltalk, Perl, Ada, Lisp and many others [Matsumoto, 2014].

According to the Ruby language website: "Ruby is a dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write." The idea behind Ruby is to create a language which has the simplicity and power of Python while being completely object-oriented. Several other concepts also have their place in Ruby language, although a full discussion of its merits and hinderances would be out of place here.

3.1.1 Ruby on Rails

Ruby on Rails is the web development framework for use with Ruby. Its function is similar to other web development frameworks such as Django and Spring. The idea behind all these frameworks is to make it easier for the developer to create web based applications. Thus many things are preordained as these would be required to run a web service anyway.

Given the choice for Ruby as a programming language the choice for Ruby on Rails is the only logical one, considering the broad support and general maturity of the framework.

The architecture of Ruby on Rails is based on the model-view-controller (MVC) paradigm. Thus the code implementing the classification algorithms is all implemented within the models created within the framework. The controllers are only responsible for getting the correct information from the users to the models and from the models to the views. The views themselves are created twice, firstly for the website and secondly for the web service, which makes use of JSON (Javascript Serialized Object Notation) for communicating with other parts of the system.

Within the CogniCor classification app the controllers are mostly concerned with permissions and providing reasonable defaults to the user. For the purposes of this thesis this is uninteresting code, and thus will not be described any further. We will shortly return to them in discussing the access system (user login and API keys).

The views are an important part of any website, as this concerns the actual webpages viewed by the user. As mentioned the views in the CogniCor classification app exist in two varieties. The normal webpages are created to allow users to access the system. Administrators are able to change more settings and create new classifiers, cross validation runs and rulesets. Users of the system can only access results and generate new rulesets. Experts are only allowed access to a page containing the cases to annotate.

The second type of views are JSON views, which are used to return information to automated systems. Only a subset of all pages is reachable via the JSON interface, as creating new classifiers and importing data should always be done with the website interface. The JSON views contain solely the necessary information returned from each request, with messages being kept to their minimum size.

The models of the Ruby on Rails app contain the actual code needed to run the classification system. As there is rather a lot of code concerning the classification model the decision was made to split this into multiple files. To keep all code organised logically the code for all models is divided into the following files:

classifier.rb
Contains all methods which are general for all types of classifiers. Differentiates between CBR and other classifiers.

submodules of the classifier model

cbr.rb
All code to execute classification with the CBR algorithm.

database.rb
The methods for retrieving and storing information in the database, separated from the other files as these methods are agnostic to the purpose for which they are used.

import.rb
Code to parse and import existing case bases. The already existing information can be imported into the application in a CSV format. This module contains all code to parse and check these inputs.

cn2_ruleset.rb
Methods to generate rulesets for all Clark Niblett varieties and classify new cases with these rulesets. Also includes code to find new cases to annotate, which invokes the cross_validation_run model to execute cross validations to this purpose. The code to handle annotations (creating, modifying and removing them) also resides in this file. As the ABCN2 ruleset generation code is in this file as well, this file also handles the use of annotated cases in generating rulesets.

cross_validation_run.rb
Contains code to create and execute a cross validation run. Generates a number of cross validation folds based on the settings when a new run is created.

cross_validation_fold.rb
: Cross validation folds pertain to a specific division of the case base into training and testing data. The cases that are in each part are registered, so results can be checked later on. Each of the cases in the testing set will be entered into the database as a cross validation case.

cross_validation_case.rb
: Only governs the storage of the results for each testing case in a cross validation fold. The information stored is used to determine which cases performed worst and should be annotated for ABCN2.

field.rb
: Each classifier works on a case base, with the case having predefined attributes. To use these attributes correctly their type is input when an existing case base is imported and stored as a field object.

This gives a short overview of the architecture of the application. The way in which the algorithms are implemented is described below in the appropriate sections. For a copy of the code the author can be contacted.

3.2 Data structures and database usage

When building an application the design of the data structures is important, especially for something as data-dependent as a classification application. Ruby on Rails provides the programmer with an easy to use way to define and use persistent data structures, which are stored in the database. For this application this standard way of creating persistent structures was not sufficient, so some manual interaction with the database was necessary.

3.2.1 Database

The database engine used for the application is MySQL. This choice was made because of the integration with Ruby on Rails and open source nature of the engine. SQLite, which is the standard database engine in use for Ruby on Rails, was ruled out as its possibilities regarding indexes are insufficient.

Given the nature of the Ruby on Rails framework it should require little effort to switch to a different database engine as long as the migration of the data is done properly. Given the manual procedures implemented to deal with the cases in the case bases this is unfortunately not the case for the classification app. A migration to PostgreSQL should not be too difficult, although at least some work is necessary to check the syntax of the manual queries.

The indexes available within MySQL are essential to the application, as these help speed up the selection mechanisms in the rule-generation and classification methods. The time-complexity of the various algorithms is mostly dependent on database passes to check whether cases are subsumed. By using indexes on the appropriate attributes the time needed has been significantly reduced.

3.2.2 Data structures

There are a number of models within the classification application that need persistent storage. Most of the peripheral models, which are relevant to the application but not the classification system itself use the standard mechanism provided by Ruby on Rails. As these parts are both relatively standard and do not influence the research described in this thesis we will refrain from describing these in more detail.

The data structures used to store the cases in the case bases and those used to store the rules and cross validation runs are more interesting as these provide the underpinnings on which the algorithms run. Each newly created classifier is stored in a general table for all classifiers, including the name, date of creation and several default settings.

When a new classifier is created several tables are made specifically for that classifier. A table which is going to be used to store all cases in the case base, a table to store the rules produced by the Clark Niblett algorithm and variants and two extra tables to use for cross validation.

The table used to store the cases is constructed based on the attributes of the cases. Every attribute is given a type (i.e., string, boolean, integer) which is used to define the type of the database column used to store the information. The treatment of the different types is similar in the algorithms later on, as all attributes are treated as nominal. Every case is assigned an id automatically by the system, which is used to keep track of references to that case in the `cross_validation_case` table.

The rules table constructed for each classifier has the same columns as the case base table, as each rule can contain attribute-value combinations for a number of attributes. Rules are part of a ruleset that is generated, thus a column is added indicating which ruleset the rule belongs to. Columns are added for the ordering and the predicted class (in case of ordered rules) and for the class distribution (in case of unordered rules) as well. Finally columns for confidence (accuracy on the training set) and significance are added, which allow for better analysis of the rules produced.

The tables that are used for cross validation are identical to the case base table. Two tables are used, one to store all cases at the moment a cross validation run is started and one to store the current training set. The first table is used so the addition of cases to the case base during the running of the cross validation does not influence the results. The second is used to simplify ruleset generation, as it becomes unnecessary to keep track of all cases in the training set if those are all stored in a separate table.

3.3 Algorithms

Aside from the data structures used to store all information relating to the classifiers in a persistent manner, the implementation of the algorithms themselves is of interest with regard to this thesis. Below we discuss the implementation of the various algorithms and the optimisations used to produce faster results.

The mechanism for classifying cases can be divided into two parts, production of rules (not applicable for CBR) and classification of new cases. For all CN2 classifiers the code to produce rules and the code to classify cases are both the responsibility of the rulesets. When a new ruleset is created the algorithm that creates the rules is run from the new object with rules being added to the ruleset when they are found. The code classifies a new case based on those rules and thus belongs in the model for the ruleset as well. For the CBR classifier the classification mechanism is encapsulated as part of the classifier model.

All methods related to the classification algorithms take an argument which is the name of the case base table. This is done so cross validation runs can use the same methods whilst temporarily retrieving the information from their own specialised tables.

3.3.1 Case Based Reasoning

The implementation used is based on the work of Armengol and Plaza [2001], as described in section 2.2.2. The actual code differs little from the pseudocode reproduced there. As the algorithm described in the paper is recursive the actual implementation has been made this way as well. The RLM calculations are handled in a separate method which is optimised to cache partial results as to speed up the computation time.

All three stopping conditions mentioned in section 2.2.2 are implemented, with two of those in a separate method. The case where there are no more attributes to use in classifying a case is hardcoded into the algorithm as it would end up in an endless loop without this check. The other two conditions are implemented in a method called `stopping_condition?` which checks if one or more of the stopping conditions hold. The question mark at the end of the method name is a Ruby convention indicating a boolean return value. Extra stopping conditions can be added or existing ones changed in this method without affecting the rest of the implementation.

Comparison between possible selectors is done based on the RLM distance. The separate method call which calculates this value is useful as it allows changing this metric without affecting the rest of the algorithm. For efficient computation of the RLM distance the indexes created on the case base table described in section 3.2 are necessary. These indexes maintain a $O(1)$ lookup time for cases based on their values. Thus retrieval of cases based on the value of their attributes is done relatively fast.

When the LID algorithm finishes some extra information is computed and returned to the caller. Thus the user of the system can see which attributes were used in what order to select the solution class. The possible solution classes and their respective confidences are shown as well.

3.3.2 Clark Niblett 2 & Augmented Clark Niblett 2

The Clark Niblett 2 algorithm and its variants are implemented in a separate model, `cn2_ruleset`. There are a number of class methods on the model which are used to generate the cases to annotate, which is of interest later on when discussing the argument-based variant of the algorithm.

The code to generate a ruleset has been split into multiple methods, similar to how this is done in the pseudocode from section 2.3.2. The only difference between the standard CN2 algorithm and its augmented variant is the calculation used to define the quality of a complex. To this end an `if-then-else` construct is used to determine which calculations should be used based on the properties of the ruleset.

Within the calculating methods calls are made to retrieve the information from the database. The database methods contain caching mechanisms to deal with similar requests. This allows the algorithm to run much faster. The indexes defined on the columns in the table are still necessary, as new calls would take orders of magnitude more time without those.

Crossbreeding the star (best n complexes) with the potential selectors is done in a method which immediately filters out entries that should not be considered. To simplify this process a hash is used, a data structure with a key-value pair notation. When crossbreeding is done with an attribute that is already in the complex the data structure does not grow, because of the fact that a hash only stores unique keys. Thus the complex has either not changed at all or would have two attribute-value pairs for the same attribute. In both cases the new complex can be immediately discarded.

All further details concerning the implementation are identical to the pseudocode produced in Clark and Niblett [1989] and Clark and Boswell [1991] or concern standard solution. These will not be discussed in any further detail, although the source code of the application should provide ample opportunity for the reader to dissect the way in which these were implemented.

3.3.3 Unordered Clark Niblett 2 & Argument Based Clark Niblett 2

The construction of the unordered CN2 algorithm is done in a manner similar to that of the ordered ruleset. Because of the many subtle differences between both algorithms the algorithm has been implemented in a separate set of methods.

When the call to generate a new ruleset is done a distinction is made between the argument-based version of the algorithm and its unordered brother. In case of the argument-based version a call is made to process all annotated case. After this is done the algorithm continues with a call to the unsorted version of the algorithm. By making the distinction between the two types of rule-generation algorithms it is still possible to generate an unordered ruleset when several cases have already been annotated. In that case the annotations on the cases are not taken into account.

The implementation of the algorithm to generate the unordered algorithm is very similar to the pseudocode from section 2.3.2. As opposed to the CBR implementation the stopping conditions are hardcoded into the algorithm as they are an integral part of the design of the classification method.

The calls to the methods are repetitive in that after each iteration they are called again if there are unclassified cases remaining in the training data. Because removing parts of the table containing the case is not an option the calls to these functions include a list of ids which contain the unclassified cases. Retrieval of only these cases is relatively fast because of the caching and indexing on ids in the database table and the database retrieval functions.

The predicted class is stored differently for unordered rules than it is for ordered rules. The first needs a full class distribution when classifying a new case, while the second only needs a majority class as mentioned in section 2.3.2. To this purpose when generating a UCN2 or ABCN2 ruleset the class distribution is serialized and stored in the database. A confidence measure is still inserted, which indicates the part of the class distribution that belongs to the majority class. This information can be inferred from the stored distribution as well but storing this in a separate field allows for an easier overview when looking at all rulesets generated.

3.4 Other parts of the system

Several other parts of the system are worthy of note, although they do not tangibly relate to the research. Their implementation is of interest for anyone with interest in software engineering, however. The system used to implement access for users is shortly discussed as is the role based authorisation system. We follow this with a short overview of the specific changes necessary to accommodate the use of the application as a web service, which includes introducing an API key verification mechanism and a JSON interface.

3.4.1 User login

Implementing a user login system is often a problem point in developing a web application. Ruby on Rails makes this problem easier by providing plugins (something done by most web development frameworks). For user management the most often used plugin within the Ruby on Rails environment is Devise, which has a highly configurable user system.

For the classification application that was built as part of this thesis Devise was implemented with a subset of its available functionality. Given that users needed to be created by the CogniCor administrators there would be no use for giving visitors the ability to sign up for an account themselves or allow them to login with OAuth.

Some small changes were made to the standard implementation based on the specifics needed in the application. The username is the main authentication token, thus allowing the creation of users without an email address, although this removes the possibility of password retrieval via an email message. Another addition was the API key authentication token which is used to make authentication of automated systems possible. This choice is discussed further in section 3.4.2.

User roles

Once a user system is in place an access control system is required as well. As mentioned earlier three main types of users will use the system *a*) administrators; *b*) users; and *c*) experts. The automated components of the CogniCor ecosystem have the same rights as normal users and can thus be given the same role.

Ruby on Rails provides several plugins for access permission as well, in this case CanCan was chosen. This is a simple system in which user roles are provided with certain named rights. These rights will be checked in the controller based on a separate model that defines which actions are available to each user group.

Administrators have full access to all parts of the application. Users have access to most actions, although most objects cannot be removed/deleted by them. Their access to cross validation runs is limited to viewing only and cannot change anything with regard to users except their own. The experts are only allowed access to the overview of classifiers and the cases that need annotations for each classifier. Initiating a new run to generate cases to annotated is reserved for users and administrators.

3.4.2 Webservice architecture

To use the system in its intended role other systems within the CogniCor ecosystem should be able to interact with it. To this purpose parts of the application were made accessible as a web service. To keep the access system intact a valid API key is necessary. These API keys are generated for each user when the account is created. API keys can be reset with the click of a button so unauthorised access can be stopped immediately.

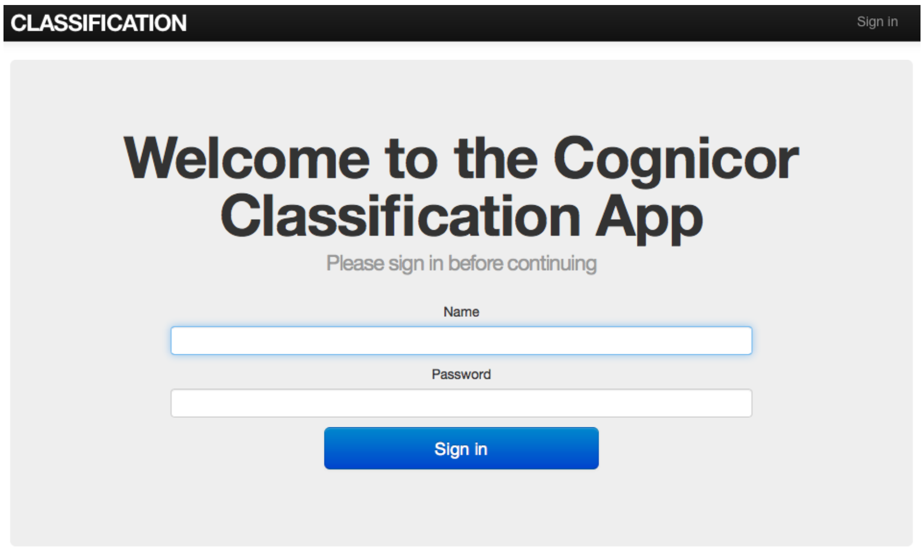
Using an API key that is sent as part of the request for a certain action has advantages compared to a login system. The connection system between the systems is transient and thus a login would not persist between requests. Providing login credentials and initiating a new session for each request is ungainly and would demand more computing power than necessary. With an API key, which is appended at the end of the request URL sent, these problems are avoided.

The communication with the parts of the web service that are available to the other system is done via JSON, both incoming and outgoing. Certain requests, such as a request for the home page, provide the system with an overview of possible actions. This is not very useful for the system itself but can be a great help to a programmer implementing the communication in another system.

3.5 Screenshots

To give a clearer picture of the application several screenshots are included. These screenshots provide some insight in the user experience when using the application.

The screenshot in figure 3.1 shows the login screen presented to the user when first visiting the site. A new user is not allowed to create an account himself, an administrator has to create the account for a new user.



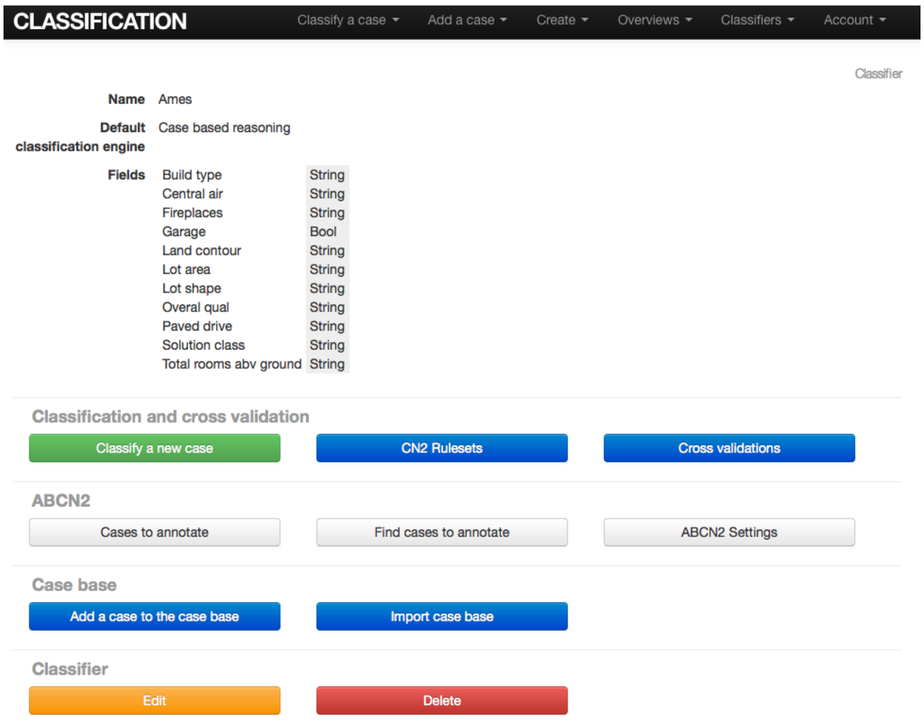
CLASSIFICATION

Welcome to the Cognicor Classification App

Please sign in before continuing

Figure 3.1: Screenshot of the login screen.

When a user has logged in an overview of all classifiers is given. Clicking on a classifier leads to a page containing an overview of the classifier with all possible actions and general information. A screenshot of this is shown in figure 3.2. The information shown is the most basic information for the classifier; the name of the classifier, the current default classification engine and the attributes of the classifier with their type.



Name	Ames	
Default classification engine	Case based reasoning	
Fields	Build type	String
	Central air	String
	Fireplaces	String
	Garage	Bool
	Land contour	String
	Lot area	String
	Lot shape	String
	Overal qual	String
	Paved drive	String
	Solution class	String
	Total rooms abv ground	String

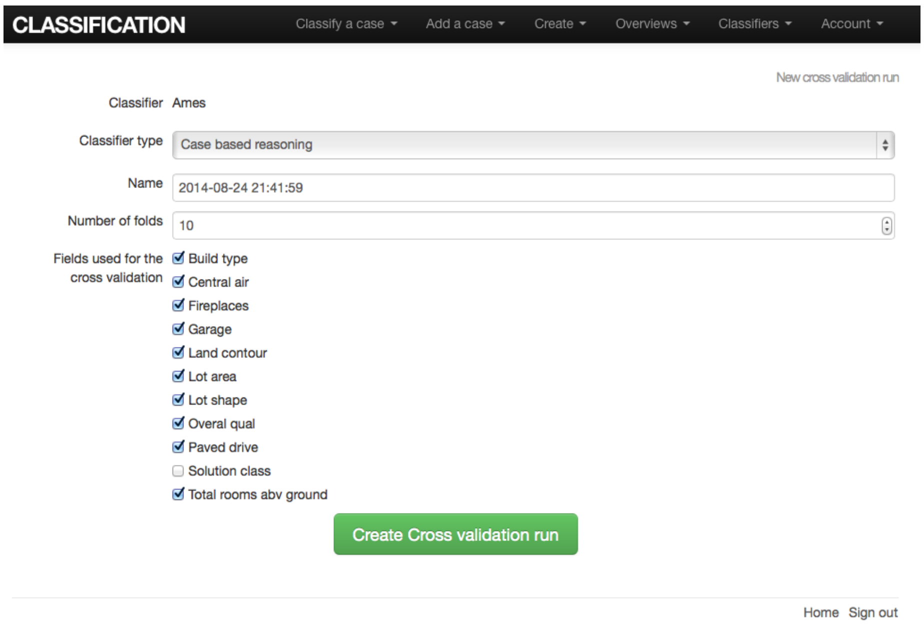
Figure 3.2: Screenshot of classifier page, showing all options and general information.

To create a new cross validation run a user goes to the page shown in figure 3.3. Here a user can select which classification engine is used for the cross validation, how many folds the cross validation run should have, the name of the run and which fields should be used to predict the solution class. This last option allows a user to see whether the omission or addition of a certain field leads to an increase or decrease in classification accuracy.

The result of a cross validation run are shown in figure 3.4. The settings used to create the run are shown, together with the results of each of the folds. The average of those results is shown, together with the time taken to execute the cross validation run.

When the argument-based CN2 algorithm is used it is necessary to find cases to annotate. Figure 3.5 shows the screen used to start looking for annotatable cases. The process of executing the cross validations necessary for finding the annotatable cases starts when the form is submitted.

This process results in one or more annotatable cases. These cases are shown to the user in the screen depicted in figure 3.6. To annotate a case the user would



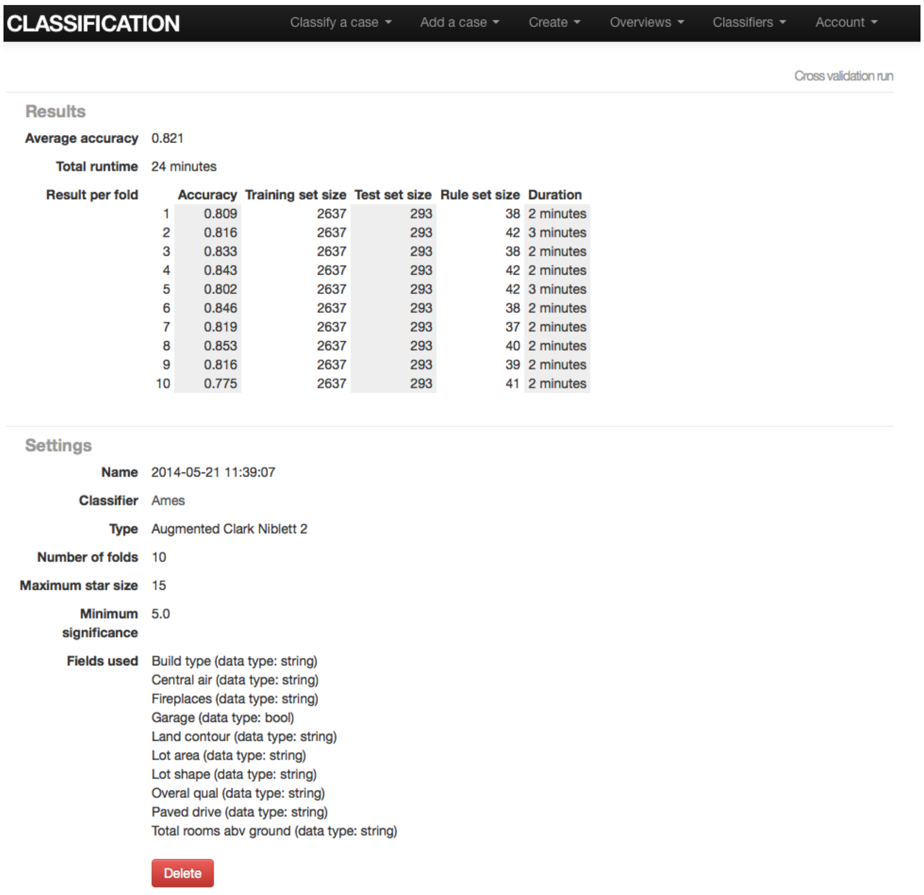
Classifier	Ames
Classifier type	Case based reasoning
Name	2014-08-24 21:41:59
Number of folds	10
Fields used for the cross validation	☑ Build type ☑ Central air ☑ Fireplaces ☑ Garage ☑ Land contour ☑ Lot area ☑ Lot shape ☑ Overal qual ☑ Paved drive ☐ Solution class ☑ Total rooms abv ground

Figure 3.3: Screenshot of the creation of a cross validation run.

click on the case he wanted to annotate. He is then presented with the screen shown in figure 3.7. Here a user can adjust the solution class (if necessary) and provide for each attribute whether it is a positive or a negative argument. An attribute can also be left as neither. If a user tries to annotate an attribute as both positive and negative an error will be raised, as shown in figure 3.8.

Classifying a new case is normally done through the web service, as this is done by automated system. For completeness an interface was created for the manual entry of new cases to classify. This interface can be seen in figure 3.9. When the form is filled out and submitted a result is returned. A screenshot of the result page is shown in figure 3.10.

All styling of the application is done with the Twitter Bootstrap theme, providing a simple and clear user interface. The usage of a standard theme reduced the time spent coding the visual parts of the application. For a more detailed look at the application access can be requested from the author. The source code is available on request as well.



Results

Average accuracy	0.821
Total runtime	24 minutes

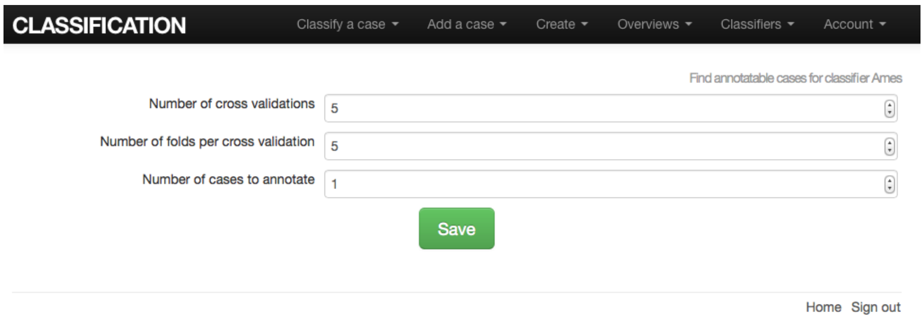
Result per fold

	Accuracy	Training set size	Test set size	Rule set size	Duration
1	0.809	2637	293	38	2 minutes
2	0.816	2637	293	42	3 minutes
3	0.833	2637	293	38	2 minutes
4	0.843	2637	293	42	2 minutes
5	0.802	2637	293	42	3 minutes
6	0.846	2637	293	38	2 minutes
7	0.819	2637	293	37	2 minutes
8	0.853	2637	293	40	2 minutes
9	0.816	2637	293	39	2 minutes
10	0.775	2637	293	41	2 minutes

Settings

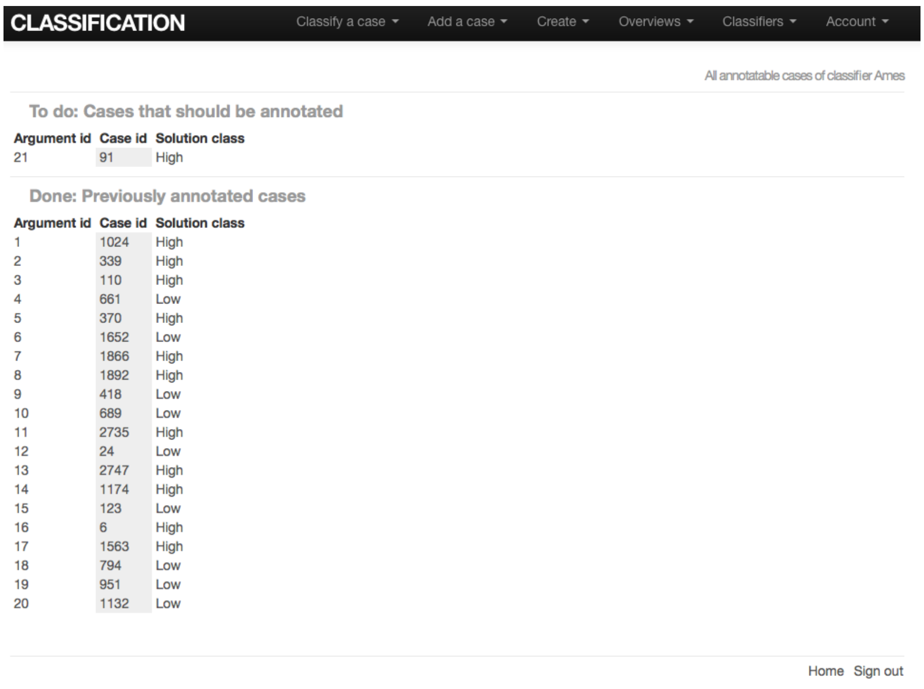
Name	2014-05-21 11:39:07
Classifier	Ames
Type	Augmented Clark Niblett 2
Number of folds	10
Maximum star size	15
Minimum significance	5.0
Fields used	Build type (data type: string) Central air (data type: string) Fireplaces (data type: string) Garage (data type: bool) Land contour (data type: string) Lot area (data type: string) Lot shape (data type: string) Overal qual (data type: string) Paved drive (data type: string) Total rooms abv ground (data type: string)

Figure 3.4: Screenshot of the results of a cross validation run.



Number of cross validations	5
Number of folds per cross validation	5
Number of cases to annotate	1

Figure 3.5: Screenshot of the page to find annotatable cases.



All annotatable cases of classifier Ames

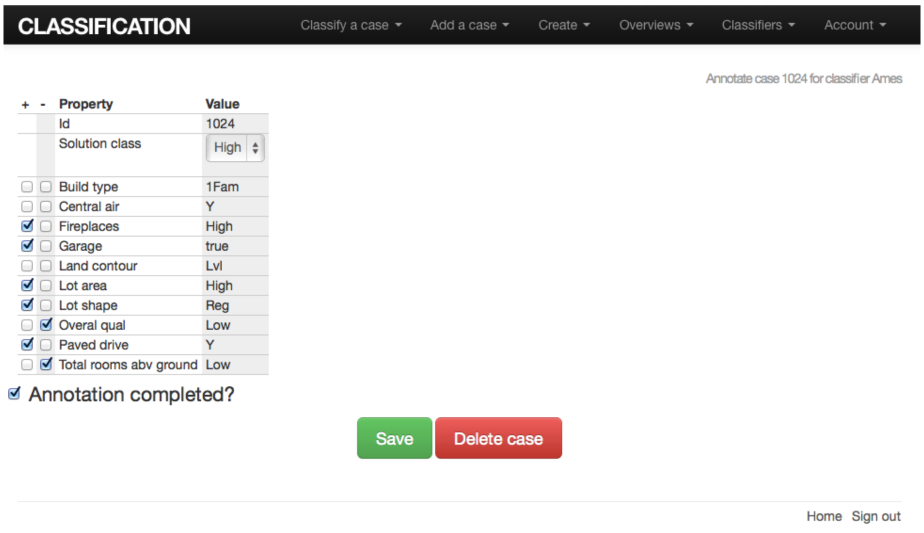
To do: Cases that should be annotated

Argument id	Case id	Solution class
21	91	High

Done: Previously annotated cases

Argument id	Case id	Solution class
1	1024	High
2	339	High
3	110	High
4	661	Low
5	370	High
6	1652	Low
7	1866	High
8	1892	High
9	418	Low
10	689	Low
11	2735	High
12	24	Low
13	2747	High
14	1174	High
15	123	Low
16	6	High
17	1563	High
18	794	Low
19	951	Low
20	1132	Low

Figure 3.6: Screenshot of overview of cases to annotate.

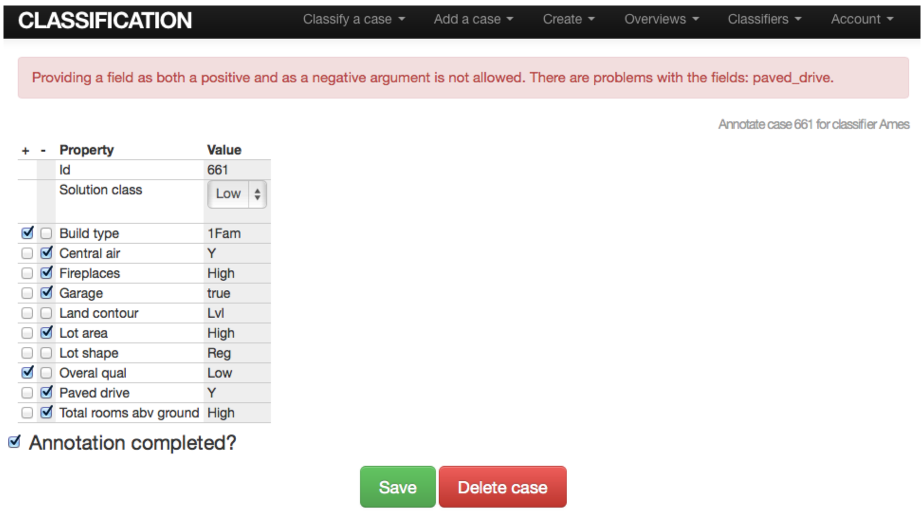


Annotate case 1024 for classifier Ames

+	-	Property	Value
		Id	1024
		Solution class	High
☐	☐	Build type	1Fam
☐	☐	Central air	Y
☑	☐	Fireplaces	High
☑	☐	Garage	true
☐	☐	Land contour	Lvl
☑	☐	Lot area	High
☑	☐	Lot shape	Reg
☐	☑	Overal qual	Low
☑	☐	Paved drive	Y
☐	☑	Total rooms abv ground	Low

☑ Annotation completed?

Figure 3.7: Screenshot of page for annotating a specific case.



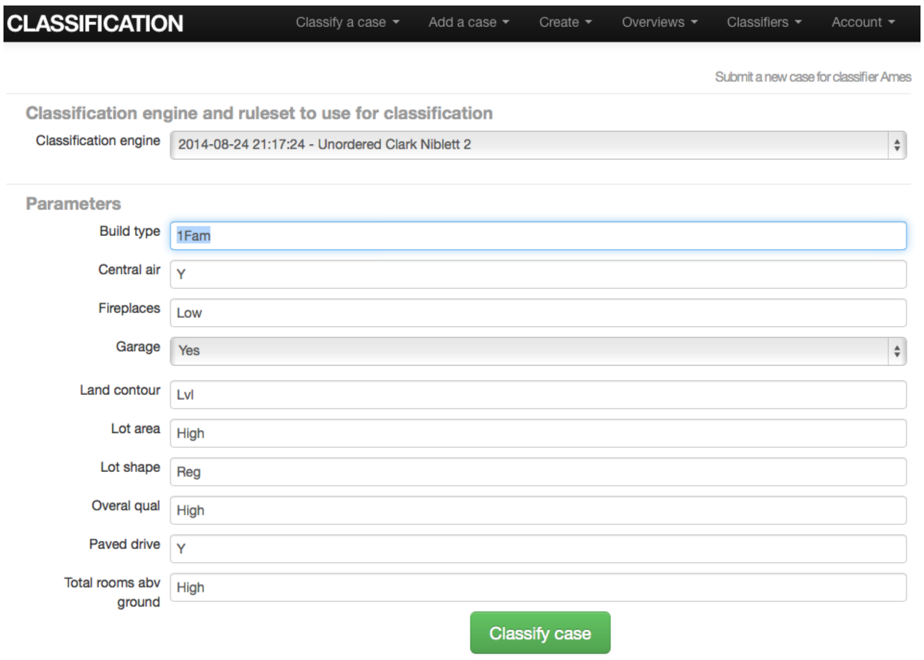
Providing a field as both a positive and as a negative argument is not allowed. There are problems with the fields: paved_drive.

Annotate case 661 for classifier Ames

+	-	Property	Value
		Id	661
		Solution class	Low
✓	☐	Build type	1Fam
☐	✓	Central air	Y
☐	✓	Fireplaces	High
☐	✓	Garage	true
☐	☐	Land contour	Lvl
☐	✓	Lot area	High
☐	☐	Lot shape	Reg
✓	☐	Overal qual	Low
☐	✓	Paved drive	Y
☐	✓	Total rooms abv ground	High

✓ Annotation completed?

Figure 3.8: Screenshot of the error message displayed when an attribute is selected as both a positive and a negative argument.



Submit a new case for classifier Ames

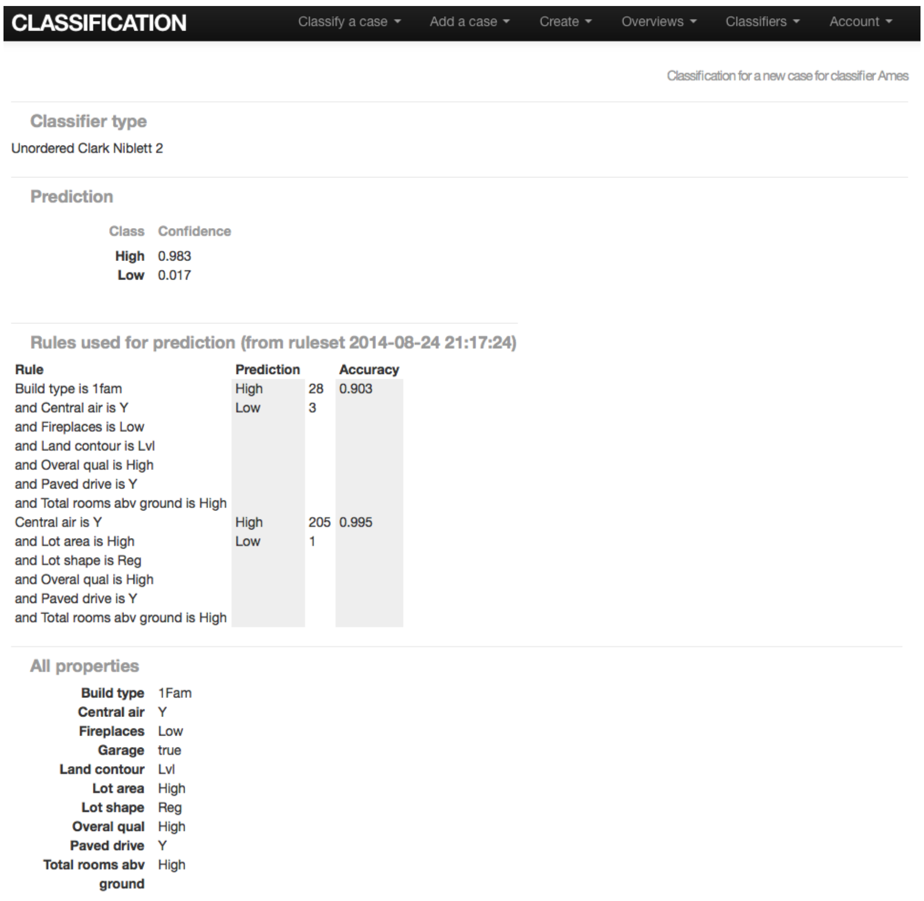
Classification engine and ruleset to use for classification

Classification engine	2014-08-24 21:17:24 - Unordered Clark Niblett 2

Parameters

Build type	1Fam
Central air	Y
Fireplaces	Low
Garage	Yes
Land contour	Lvl
Lot area	High
Lot shape	Reg
Overal qual	High
Paved drive	Y
Total rooms abv ground	High

Figure 3.9: Screenshot of the entry of a new case to classify.



Classification for a new case for classifier Ames

Classifier type

Unordered Clark Niblett 2

Prediction

Class	Confidence
High	0.983
Low	0.017

Rules used for prediction (from ruleset 2014-08-24 21:17:24)

Rule	Prediction		Accuracy
Build type is 1fam and Central air is Y and Fireplaces is Low and Land contour is Lvl and Overal qual is High and Paved drive is Y and Total rooms abv ground is High	High Low	28 3	0.903
Central air is Y and Lot area is High and Lot shape is Reg and Overal qual is High and Paved drive is Y and Total rooms abv ground is High	High Low	205 1	0.995

All properties

Build type	1Fam
Central air	Y
Fireplaces	Low
Garage	true
Land contour	Lvl
Lot area	High
Lot shape	Reg
Overal qual	High
Paved drive	Y
Total rooms abv ground	High

Figure 3.10: Screenshot of the result when classifying a case.

Chapter 4

Experiments

To test the performance of the implemented algorithms extensive experiments with a range of datasets have been performed. All datasets have been tested in combination with all algorithms, with the exception of the Mushroom dataset. Some of the datasets were taken from the UCI machine learning repository [University of California, Irvine, 2014] and others come from the datasets provided by Greene [2003] on the website accompanying the 7th edition of the book. Where possible both the repository from the dataset was obtained and the original source of the data will be provided.

Unfortunately the data from CogniCor could not be used to test the system for two reasons. The data itself was not allowed to leave the CogniCor servers and could thus not be used after the end of the internship. Secondly, which compounded the first problem, the complete system was not operative at the finish of the internship. Thus no testing was done with experts before the internship ended. Given that the complaints were in Spanish with the author having a poor grasp of this language, it was not possible to do testing with this dataset without the help of an expert.

To simulate the opinion of domain experts datasets have been sought that allow a layman to stand in for the expert when annotating cases. For all datasets presented here this proved reasonably straight-forward, one main exception being the Mushrooms dataset from the UCI repository. Both for this reason and the astonishing performance of the UCN2 algorithm no testing has been done with the argument-based algorithms for this dataset.

We will start by introducing the data preparation techniques used to convert the data into an acceptable format. Then we will briefly discuss each dataset, its origin and application. A short overview of all datasets is provided for comparative purposes. We finish the chapter with a description of the experiments performed. The results for the experiments can be found in the next chapter.

4.1 Data preparation techniques

To get the datasets compliant with the structure necessary to use the implemented algorithms, several changes had to be made. The algorithms assume that all values are nominal in nature and treat them this way, thus a discretisation step is necessary. In the papers of Clark and Boswell [1991] and Možina et al. [2007] this discretisation is included, but it is suggested as an automated process. For the system that was built during the internship this was deemed an unnecessary feature and not implemented. For the experiments that followed the discretisation was done by hand. In case of ordinal values the attributes were used in a nominal fashion.

Discretising continuous data has been discussed in literature extensively. In the case of the datasets used in the experiments a relatively simple method of discretising the data was used. This method is called equal frequency bins and a formal description is given in Dougherty et al. [1995]. The cases were ordered based on that particular attribute and boundaries were determined that provided the same amount of cases in each category. These categories were then labelled and used as nominal data values when running the algorithms.

Given the nature of the attributes the number of nominal categories to divide the data in has also been varied. In several instances a clear division was immediately apparent, whilst in some others several possibilities were tried. The specifics of these choices are documented with the respective datasets.

The prototypical nature of the application makes that the parser of the CSV files can be considered quite tricky. Thus it was necessary to convert the data into the exact format that was expected by the application. The steps necessary (string and special character escaping, defining fields, etc.) are not very interesting nor relevant for the experiments and will thus be omitted.

A final preparatory step was to omit some attributes of the datasets. The Ames housing dataset, as an example, contains 82 attributes. Running the algorithms on the full dataset would take too much time, thus some of the attributes had to be omitted. In other case the choice to drop one or more attributes was based on the problems expected when annotating this attribute or the knowledge that there is no apparent correlation with the solution class.

4.2 Datasets

Seven datasets were used in the experiments to test the usability of the algorithms. Each of these datasets is discussed briefly, including any particular choices with regard to discretisation. The size of the datasets, as well as the number of attributes can all be found in the overview in table 4.1.

Name	# attributes	# attributes (after prep)	Number of cases
Auction	11	7	1225
PC	10	10	6259
Ames	11	11	2930
Credit	11	11	13444
Housing	12	12	546
F181	9	8	601
F182	15	15	210

Table 4.1: Overview of datasets used for experimentation.

4.2.1 Auction

The auction dataset comes from the research in Rezende [2008]. It pertains to data about auction of iPods on eBay.

An overview of the attributes of the dataset can be found in table 4.2. In the table the name of the attribute, the type of data and the possible values or minimum and maximum are given. This information is followed by a column indicating whether some form of conversion or editing was done before the dataset was used in the experiments. The final column shows the new possible values if the attribute was converted or edited. When an attribute was omitted it will be stated in this column as well, with the reason for this decision given.

In discretising the attributes a different number of categories has been chosen. This is done to give meaningful boundaries to the categories.

4.2.2 PC

This dataset, taken from Stengos and Zacharias [2006], concerns the sales price of personal computers between January 1993 and November 1995. Several properties of the sold PCs are included. The data was originally collected for a temporal analysis but can also be used to test the algorithms implemented in this thesis. The temporal component is kept, but no special attention is given to it. Thus it shows up in some rules, where appropriate.

Discretisation for this dataset was taken somewhat further than that of the auction dataset, as only two categories have been used when discretisation was applied, high and low.

4.2.3 Ames

The Ames dataset concerns the sales price of houses in Ames, Iowa, in the United States. The data was provided by the Iowa Assessor's Office and was first used in De Cock [2011]. The number of attributes available is too much to handle for the algorithms, thus a slimmed down version has to be used. Previous analyses of the dataset have provided a good indication of which properties have a good correlation with the sales price and are provided in a separate, slimmed down, dataset. This adjusted dataset was used as the basis for the experiments. The attributes present in the dataset and their adjustments can be found in table 4.4.

As can be seen in the table the same mechanism was used for the Ames dataset as for the PC dataset, attributes which needed discretisation were divided into two categories of approximately the same size. Many of the attributes could be kept as is, given their nominal nature. The number of fireplaces has been replaced by a boolean indicating whether one or more fireplace are available to provide a clearer distinction between the two groups.

4.2.4 Credit

The source of this dataset is Greene [2003], where the data can be found in table 7.3. The attributes of the dataset and the adjustments made can be found in table 4.5.

Three of the attributes in the original table have been left out of the dataset. These properties concern the ratio between earning and spending, average spending and the log of the average spending. All three are difficult to discretise and more useful for linear regression than for classification purposes.

The discretisation within this dataset was done based on the intuitiveness of the new values for the expert annotating the cases. Thus age has been converted into brackets that make sense for a human and a cutoff point for dependents is given (3 or more).

4.2.5 Housing

The housing dataset (from Windsor, Ontario, Canada) comes from Anglin and Gencay [1996] which tests the effectivity of using semiparametric estimation functions. The data is similar to both the Ames housing data and the Boston housing data.

The nature of the attributes made discretisation unnecessary in most cases. Only the sale price in dollars (the class to predict) and the lot size had to be discretised before they could be used in the application.

4.2.6 F181

F181 and F182 are datasets from Greene [2003], named after the example in which they were used. This dataset on extramarital affairs come from Fair [1978] originally.

For the F181 dataset few adjustments were needed. The occupation attribute was removed as it would be difficult to annotate and the number of affairs (used as the solution class) was discretised into two categories, those who had an affair and those who did not.

4.2.7 F182

The F182 dataset comes from Greene [2003], with the original data being published in Greene and Hensher [1997]. The dataset contains information about the choice of transport used in 210 cases. In the dataset as found on the website accompanying Greene [2003] the choices have been split and thus it contains 840 entries. Before using this dataset every four rows were combined into one, with each row corresponding to a journey undertaken. To get meaningful attributes the time for each journey was calculated (in the original dataset this is split into terminal waiting and in vehicle time). The same was done for the total cost of the journeys. A combined measure (cost multiplied by time) was introduced to directly compare methods of transport (with the assumption that both are appreciated equally).

To use these attributes in annotations a meaningful concept for a human operator has to be created. In this case the choice was made to use the relative ranking between the modes of transport. Thus for each of the three attributes described above their relative rank was calculated and entered into the dataset.

The chosen type of transport, travel party size and income were taken directly from the original dataset, as they do not differ for any of the four entries for each journey. The income was discretised into three categories; high, medium and low with the boundaries determined to create equal frequency bins.

4.3 Executed experiments

To test the predictive accuracy of the algorithms the datasets were used in a series of experiments. Both the comparative performance of the types of algorithm (CBR, CN2 and its variants) and the performance gains of adding annotations (in case of ABCN2) were tested.

Determining the predictive accuracy of the algorithms was done by using 10-fold cross validations. As described earlier this mechanism is used by the ABCN2 algorithm to determine which cases need to be annotated. The standard setting in that case is to use five 5-fold cross validations. Using 10-fold cross validations provides the algorithms with larger training sets and thus slightly improved accuracy compared to a lower number of folds. It slows down the procedure however, which is not a major concern for a one-off event such as these experiments.

The experiments consist of doing cross validations on each dataset with all of the none-annotated algorithms to see their relative performance. After that the annotation mechanism was used to determine which case needed annotations. Then a new series of cross validations was done to see the performance gains of these annotations. This process was repeated until each dataset had 20 annotations.

The runtimes for each of the cross validations were noted as well, with the purpose of comparing runtimes and seeing whether the remark made by Možina et al. [2007] was confirmed, that adding arguments improved (i.e., reduced) the runtime of the rule-generation process.

Name	Type	Values	Edit?	New values
The unique eBay code	Nominal		Yes	Removed, not relevant
The number of bidders	Ratio	1 - 20	Yes	High (> 9), Medium ($6 < x \leq 8$), Low ($2 \leq x < 5$), One (1)
The final price	Ratio	1 - 270	Yes	High (> 150), Medium ($100 < x \leq 150$), Low (≤ 100)
The seller's positive feedback percentage	Ratio	0.0 - 100.0	Yes	Extreme (100%), High (99-99.9%), Medium (98-98.9%), Low (0-98%)
The seller's feedback score	Ratio	-3 - 158452	Yes	Removed, difficult to provide feedback on
A code for the reported colour of the iPod being sold	Nominal	Black, Blue, Gold, Grey, Green, Pink, Silver, SilverWhite, White, Custom, None, Other	No	
The reported condition	Ordinal	Used, Refurbished, New, None, Unknown	No	
The datetime of the final bid	Ratio		Yes	Removed, not relevant
Reported memory, in Gb	Ordinal	1, 2, 4, 5, 6, 15, 20, 30, 40	No	
A code for state of the item	Nominal	Non-operational or serious defect, Cosmetic defects, Good condition	No	
The auction reserve price	Ratio	0.01 - 250.00	Yes	Removed, difficult to provide feedback on

Table 4.2: Attributes of the Auction dataset

Name	Type	Values	Edit?	New values
Price in dollars	Ratio	949 - 5399	Yes	High ($>$ 2144), Low ($\leq$ 2144)
Clock speed in MHz	Ratio	25 - 100	Yes	High ($>$ 50), Low ($\leq$ 50)
Size of hard drive in MB	Ratio	80 - 2100	Yes	High ($>$ 340), Low ($\leq$ 340)
Size of RAM memory in MB	Ratio	2 - 32	Yes	High ($>$ 8), Low ($\leq$ 8)
Size of screen in inches	Ratio	14 - 17	Yes	High ($>$ 14), Low ($\leq$ 14)
Whether a CD-ROM drive is present	Nominal	Yes, No	No	
Whether a multimedia kit is present	Nominal	Yes, No	No	
Whether the PC is from a well known manufacturer	Nominal	Yes, No	No	
The number of price listings for the current month	Ratio	39 - 339	Yes	High ($>$ 246), Low ($\leq$ 246)
The month in which the price was noted	Nominal	January 1993 - November 1995	No	

Table 4.3: Attributes of the PC dataset

Name	Type	Values	Edit?	New values
Price of the house in dollars	Ratio	12789 - 755000	Yes	High (> 160000), Low (≤ 160000)
Lot area in square feet	Ratio	1300 - 215245	Yes	High (> 9437), Low (≤ 9437)
Overall Quality	Ordinal	1 - 10	Yes	High (> 6), Low (≤ 6)
Central air conditioning	Nominal	Yes, No	No	
Paved driveway	Nominal	Yes, No	No	
Fireplaces	Ratio	0 - 4	Yes	Yes, No
Building Type	Nominal	1Fam, 2FmCon, Duplx, TwnhsE, TwnhsI	No	
Lot shape	Ordinal	Regular, Irregular1, Irregular2, Irregular3	No	
Total Rooms Above Ground	Ratio	2 - 15	Yes	High (> 6), Low (≤ 6)
Garage	Nominal	Yes, No	No	
Land Contour	Nominal	Level, Banked, Hillside, Depression	No	

Table 4.4: Attributes of the Ames dataset

Name	Type	Values	Edit?	New values
Accepted	Nominal	Yes, No	No	
Defaulted	Nominal	Yes, No	Yes	Removed (irrelevant)
Age	Ratio	18,0 - 88,67	Yes	0 - 25, 25 - 35, 35 - 45, 45+
Number of dependents	Ratio	0 - 9	Yes	0, 1, 2, 3 or more
Months living at current address	Ratio	0 - 576	Yes	Short ($\leq$ 30), Long ($>$ 30)
Number of major derogatory reports	Ratio	0 - 22	Yes	Yes, No
Number of minor derogatory reports	Ratio	0 - 11	Yes	Yes, No
Owner of home	Nominal	Yes, No	No	
Monthly income	Ratio	50 - 8333,25	Yes	High ($>$ 2167), Low ($\leq$ 2167)
Self-employed	Nominal	Yes, No	No	
Yearly income per person in household	Ratio	362,5 - 150000	Yes	High ($>$ 19000), Low($\leq$ 19000)

Table 4.5: Attributes of the Credit dataset

Name	Type	Values	Edit?	New values
Sale price in dollars	Ratio	25000 - 1900000	Yes	High ($>$ 62000), Low ($\leq$ 62000)
Lot size in square feet	Ratio	1650 - 16200	Yes	High ($>$ 4600), Low ($\leq$ 4600)
Number of bed rooms	Ratio	1 - 6	No	
Number of bath rooms	Ratio	1 - 4	No	
Number of storeys	Ratio	1 - 4	No	
Driveway attached	Nominal	Yes, No	No	
Recreational room available	Nominal	Yes, No	No	
Basement available	Nominal	Yes, No	No	
Gas heating available	Nominal	Yes, No	No	
Air conditioning available	Nominal	Yes, No	No	
Number of garages	Ratio	0 - 3	No	
Desirable location	Nominal	Yes, No	No	

Table 4.6: Attributes of the Housing dataset

Name	Type	Values	Edit?	New values
Sex	Nominal	Male, Female	No	
Age	Ordinal	Under 20, 20 - 30, 30 - 40, 40 - 50, 50 - 60, 60 and over	No	
Years married	Ordinal	Under 3 months, 3 - 6 months, 6 months - 1 year, 1 - 2 years, 2 - 5 years, 5 - 8 years, 8 - 11 years, 12+ years	No	
Children	Nominal	Yes, No	No	
Religious	Ordinal	Atheistic, Not at all, Slightly, Somewhat, Very, No		
Education	Ordinal	Grade school, High school, Some college, Some graduate, College graduate, Masters, PhD MD or other advanced	No	
Occupation	Nominal	1 - 7	Yes	Removed, use of occupation in annotations unclear
Rating of marriage	Ordinal	Very unhappy, Unhappy, Average, Happier than average, Very happy	No	
Number of affairs	Ratio	0 - 12	Yes	No affairs, Affair(s)

Table 4.7: Attributes of the F181 dataset

Name	Type	Values
Mode of transport chosen	Nominal	Air, Train, Bus, Car
Income	Ratio	Low (≤ 22), Medium ($22 < x \leq 38$), High (> 38)
Travel party size	Ratio	1 - 6
Time bus (ranked)	Ordinal	1 - 4
Time car (ranked)	Ordinal	1 - 4
Time train (ranked)	Ordinal	1 - 4
Time plane (ranked)	Ordinal	1 - 4
Cost bus (ranked)	Ordinal	1 - 4
Cost car (ranked)	Ordinal	1 - 4
Cost train (ranked)	Ordinal	1 - 4
Cost plane (ranked)	Ordinal	1 - 4
Time x cost bus (ranked)	Ordinal	1 - 4
Time x cost car (ranked)	Ordinal	1 - 4
Time x cost train (ranked)	Ordinal	1 - 4
Time x cost plane (ranked)	Ordinal	1 - 4

Table 4.8: Attributes of the F182 dataset. The ranked attributes describe have each value between 1 and 4 occurring exactly once, thus those qualify as ordinal.

Chapter 5

Results

The setup of the experiments has been described in section 4.3, while the results of running those experiments will be described below. The experiments delivered two types of results, data on predictive accuracy and on runtimes. The predictive accuracy can be used to show whether the use of argument-based classification techniques actually improves the performance of those classifiers when used. The runtimes are of less importance but help to verify a hypothesis mentioned in Možina et al. [2007], that adding annotations to a case base decreases the time necessary for generating rulesets.

All datasets described in section 4.2 have produced results for all experiments described in section 4.3. These results concern predictive accuracy and runtimes for the different algorithms that do not use annotations and results with ABCN2 while increasing the number of arguments progressively. Thus case-based reasoning, the original Clark Niblett 2 algorithm, the augmented CN2 algorithm (both producing ordered rules) and the unsorted CN2 algorithm and argument-based version (both unsorted) can be compared on performance and runtimes.

Cases to annotate were found based on the procedure described in section 2.4.1. After annotating a case the classifier was run through a set of five cross validations. This procedure was repeated until 20 cases were annotated in each classifier. When reporting the results not all 20 steps will be reported. Only the situations with 1 to 5 arguments and 10, 15 and 20 arguments will be discussed.

5.1 Predictive accuracy

The predictive accuracy of the classifier is the average of five 10-fold cross validations. As explained each case base has been used in a series of experiments. First a look will be taken at the data produced by these experiments. Then a comparison between the different types of classifiers will be made, followed by a look at the effect of adding arguments to the argument based Clark Niblett 2 classifier.

Dataset	Majority class	CBR	CN2	ACN2	UCN2	ABCN2							
						1 arg	2 arg	3 arg	4 arg	5 arg	10 arg	15 arg	20 arg
Auction	39,7%	80,2%	48,3%	64,8%	37,1%	38,0%	39,6%	39,4%	45,9%	48,1%	50,5%	51,8%	52,1%
		0,7%	1,4%	1,2%	1,1%	1,3%	1,2%	1,0%	0,8%	0,9%	1,1%	0,7%	0,7%
PC	50,0%	86,0%	44,9%	77,3%	72,9%	69,4%	68,4%	66,8%	70,2%	67,2%	67,0%	68,2%	67,6%
		0,9%	0,8%	0,6%	0,5%	0,6%	0,6%	0,7%	0,5%	0,8%	0,4%	0,5%	0,6%
Ames	50,0%	81,2%	74,0%	82,1%	73,6%	76,6%	76,5%	77,7%	80,4%	81,4%	82,7%	83,6%	85,1%
		0,6%	1,2%	0,9%	1,1%	0,8%	1,3%	1,1%	0,8%	1,0%	0,7%	0,9%	1,0%
Credit	62,3%	79,2%	63,8%	71,1%	52,3%	51,9%	52,8%	53,4%	53,1%	55,6%	56,8%	56,1%	55,9%
		0,5%	0,9%	1,0%	1,3%	1,0%	0,9%	0,9%	1,2%	0,8%	1,1%	1,0%	1,0%
Housing	50,4%	92,5%	45,4%	69,4%	63,1%	59,3%	61,1%	61,7%	60,2%	61,5%	64,6%	67,2%	68,1%
		0,7%	1,4%	1,3%	0,9%	1,2%	1,4%	1,1%	1,0%	1,1%	0,9%	1,1%	1,2%
F181	52,4%	88,0%	74,7%	65,4%	41,6%	42,1%	44,2%	43,7%	45,6%	46,9%	53,6%	58,1%	60,4%
		0,9%	1,1%	0,8%	1,2%	1,0%	1,3%	1,4%	1,0%	1,2%	0,9%	1,0%	0,8%
F182	30,0%	80,6%	59,6%	62,7%	70,5%	70,9%	71,6%	72,1%	72,3%	74,7%	77,9%	80,1%	82,3%
		0,8%	0,7%	1,4%	1,1%	1,3%	0,9%	1,0%	0,8%	1,2%	0,9%	0,7%	1,0%

Table 5.1: Predictive accuracy results for all datasets. The reported predictive accuracy is the average accuracy of five 10−fold cross validations. The number below the accuracy is the standard error for that accuracy.

5.1.1 Raw data

Table 5.1 shows the performance of the different algorithms on all case bases in the experiments. Looking at the raw data several trends are immediately clear. First of all case based reasoning, our baseline method, performs exceptionally well. In all but one dataset it performs best of the five main types of classifier. Given that the CBR classifier does not have an explicit model but rather evaluates all cases when a new evaluation has to be made this is not wholly unexpected but still surprising. Part of the explanation is that the other methods, when generating rules, eliminate information from the dataset. This data is superfluous according to the algorithms, but it is an information loss nonetheless. The CBR classifier does not have this problem, potentially resulting in better predictive accuracy.

A second observation on the raw data is that the augmented Clark Niblett 2 classifier (ACN2) almost always outperforms the original Clark Niblett 2 algorithm. This indicates that the changes made to the algorithm have been beneficial for its predictive accuracy. Given that these improvements have also been taken into account when creating the ABCN2 classifier this leads to the conclusion that some potential problems have already been removed.

5.1.2 Comparison between classifier types

Figure 5.1 contains a comparison between the four main types of non-annotated classifiers and the end result of using the argument-based classifier (with 20 annotated cases). This is part of the data given in table 5.1, provided in this way to give a clear insight into the comparison between classifier types.

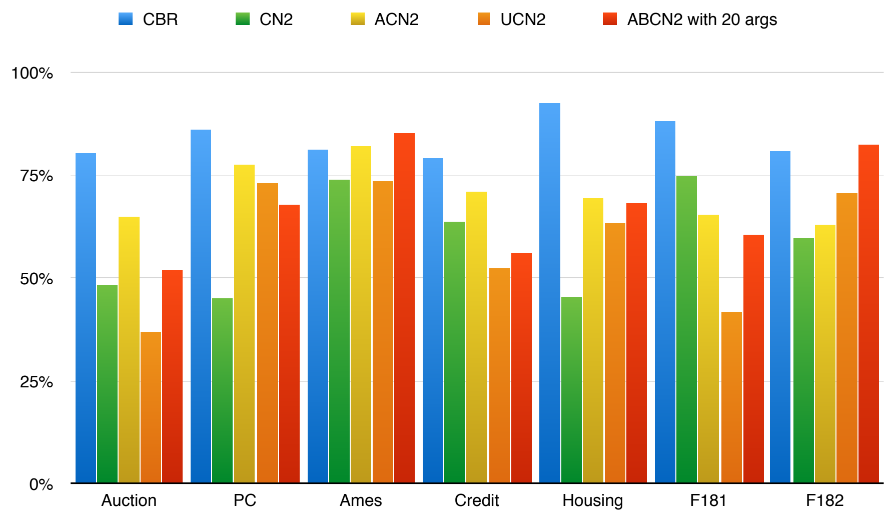


Figure 5.1: Overview of the accuracies for the different classifier types for all datasets. Taken as the average accuracy of five 10−fold cross validations.

Note that UCN2 and ABCN2 without any arguments are the same, thus producing identical results. Here the exceptional performance of the CBR classifier can be clearly seen. Only the Ames dataset has a rule-based classifier

(ACN2) that outperforms the CBR algorithm, with two argument-based classifiers outperforming CBR, on the Ames and F182 datasets.

In general the CN2 classifier performs worst, with the ACN2 classifier outperforming it on all but one dataset. Both make use of ordered rulesets, whilst the UCN2 classifier uses unordered rules. In most cases this seems to result in a worse performance compared to the ACN2 classifier. Only the F182 dataset improves in terms of classification accuracy when using unordered rules instead of ordered ones.

The F181 dataset seems to have some strange properties, as it is the only case where the original CN2 algorithm outperforms its augmented version. The difference between the ordered and unordered algorithms is also especially large for this dataset. No direct reason for these strange results can be found, although it is possible that the domain to which the data pertains lends itself unusually well to ordered rulesets.

5.1.3 Adding annotations

The effects of adding annotations to the datasets can be seen in figure 5.2 (which uses the data from table 5.1). In most cases annotating cases leads to a higher predictive accuracy. Several datasets defy this trend, with the PC actually worsening and both Credit and F182a datasets staying at the same level. The Housing dataset also provides interesting data, as it only starts improving after the addition of five annotations.

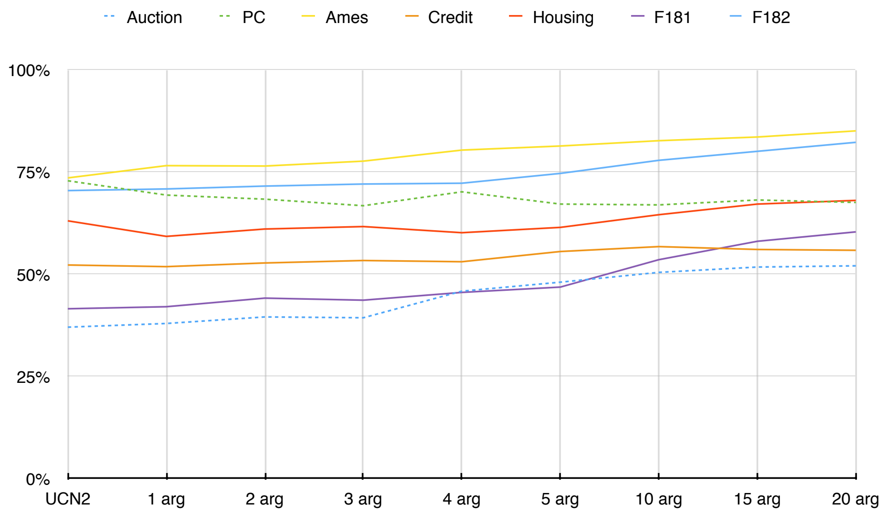


Figure 5.2: Overview of the accuracies when adding arguments for all datasets. Taken as the average accuracy of five 10−fold cross validations.

From studying the data it becomes apparent that the ABCN2 algorithm is not universally profitable. In some applications, a few of those we have used, no improvement seems to be made. The reasons for this can be found in a number of different directions. First of all the quality of the annotations could be worse on these domains than in others as no official experts have been used, thus limiting the usefulness of the annotations to the understanding of the domain knowledge by the author of this thesis. This might have resulted in incorrect or incomplete annotations, thus misguiding the algorithm in its search for proper rules.

Secondly some of the domains might not lend themselves to this type of annotations or the use of unsorted rules in general. This can be more clearly seen in figure 5.1, where the difference between ABCN2 without arguments (same as UCN2) and ABCN2 with 20 arguments shows no improvement for some domains.

In a number of cases the sorted rulesets outperform the unsorted rulesets. This might point at better applicability of these types of rulesets for those specific domains. It does not diminish the problem stated earlier on, that sorted rules can be much harder to check for experts and that it is more difficult to explain a decision taken based on these sorted rules.

Positive results occur in the Ames and F182 datasets. In the Ames dataset the argument-based version (given enough arguments) outperforms even the CBR method. The same applies for the F182 dataset, which is the one dataset where the unsorted classifiers outperform the ordered ones.

5.2 Runtimes

The runtime of an experiment is the average duration of five 10-fold cross validations. Each dataset has been used in a series of experiments. First a look will be taken at the data produced by these experiments, with a comparison between the different types of classifier following. Then a look will be taken at the effect of adding arguments to the argument based Clark Niblett 2 classifier.

The runtime data comes from the same experiments as the predictive accuracy data, with the average of five cross validation runs being taken.

5.2.1 Raw data

The raw data regarding the runtimes can be found in table 5.2. In looking at the data it is important to realise that a comparison between CBR and the other methods is not completely fair. The CN2 algorithm and its variants produce a ruleset for each fold, which is the most costly part of the whole process. This seems to give the impression that the CBR algorithm could produce decisions quicker than some CN2 variants. In practice this is not the case, as ruleset generation (the slowest part of these experiments) does not need to be done when a new case comes in but rather at fixed time intervals. Thus it is better to look at a comparison between the different CN2 variants for each dataset, though for completeness we have included CBR.

The changes made to the original CN2 algorithm, resulting in the ACN2 algorithm, have a negative side as well. This can be clearly seen in the data, as ACN2 takes on average 60% more time to complete a cross validation. Using the unsorted CN2 variants takes even longer, although there are some exceptions (PC, Credit).

A general trend can also be seen in the use of arguments, with more annotations resulting in a reduced runtime. This will be discussed in more depth in subsection 5.2.3.

Dataset	CBR	CN2	ACN2	UCN2	ABCN2							
					1 arg	2 arg	3 arg	4 arg	5 arg	10 arg	15 arg	20 arg
Auction	80	56	462	613	546	382	349	302	296	252	201	203
	3	2	9	11	12	8	9	7	7	9	6	8
PC	1672	554	3194	2219	1964	1923	1716	1575	1546	847	676	568
	27	16	39	34	28	26	27	24	20	14	15	13
Ames	535	327	1445	3549	2341	1884	1677	1409	1832	1440	1064	959
	16	18	34	53	49	42	45	51	43	46	48	43
Credit	6329	11483	13842	7044	6788	6613	6594	6631	6542	6213	6094	5972
	134	152	116	112	97	107	99	104	91	93	98	95
Housing	71	83	134	256	243	249	228	224	209	195	181	164
	4	7	9	17	15	23	21	16	19	17	19	18
F181	141	261	734	619	611	557	513	491	464	402	361	318
	9	23	46	38	37	42	44	41	37	39	42	38
F182	109	162	631	527	438	421	390	348	317	264	214	199
	8	13	33	29	31	24	26	27	21	19	16	19

Table 5.2: Runtime results in seconds for all datasets. The reported runtime is the average runtime of five 10−fold cross validations. The number below the runtime is the standard error for that runtime.

5.2.2 Comparison between classifier types

Figure 5.3 shows the average runtimes for the different types of classifier. The data for this chart is taken from table 5.2. The trend visible in the raw data concerning increasing runtimes between the classifiers in the order in which they were introduced can be seen here as well. There is a caveat though, in that this increase in runtime is always within the same order of magnitude, thus proving not that problematic as this process will be done relatively rarely in practice.

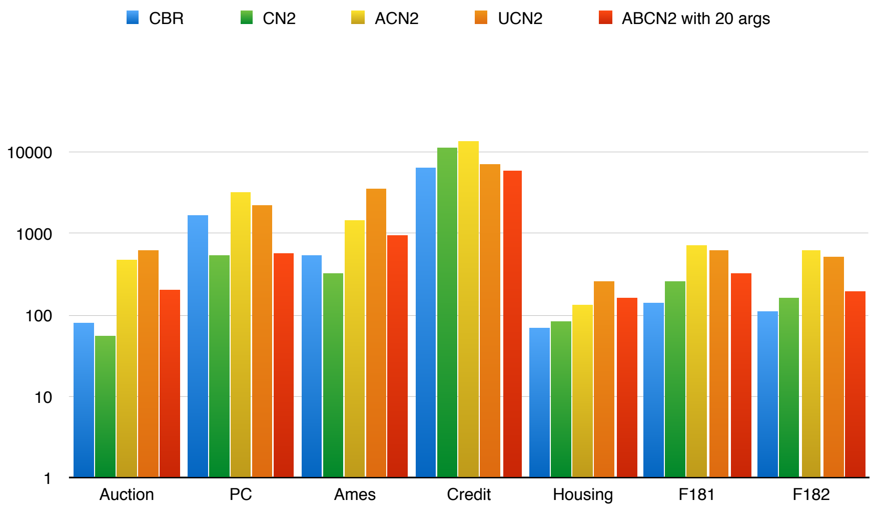


Figure 5.3: Overview of the runtimes for the different classifier types for all datasets. Taken as the average runtime of five 10−fold cross validations.

Interesting to note is that using a CBR classifier is considerably slower than using a rule-based classifier in practice. As mentioned the CBR data is reflective of the time taken when classifying all cases in the cases base. For the rule-based classifiers this is not the case, with the time in table 5.2 reflecting the generation of 10 new rulesets and classification of all cases in the case base.

The chart shows that the argument-based classifier is in all cases faster than the unordered CN2 algorithm. This shows that there are practical benefits to using a rule-based classifier, even if this does not improve predictive accuracy (although it should never reduce it).

It is interesting to see that both ACN2 (producing ordered rules) and UCN2 (producing unordered rules) have a similar runtime. CN2 is significantly faster than AC2, but often worse in terms of accuracy.

5.2.3 Adding arguments

The effects of adding arguments on the runtime of the cross validations can be seen in figure 5.4, which displays the data from table 5.2 in a different form. The trend within the different datasets is a reduction of the running time when annotations are added to a case base.

In some of the datasets this trend is somewhat stronger than in others, but in all cases the trend does exist. This can be seen more clearly in figure 5.3 with

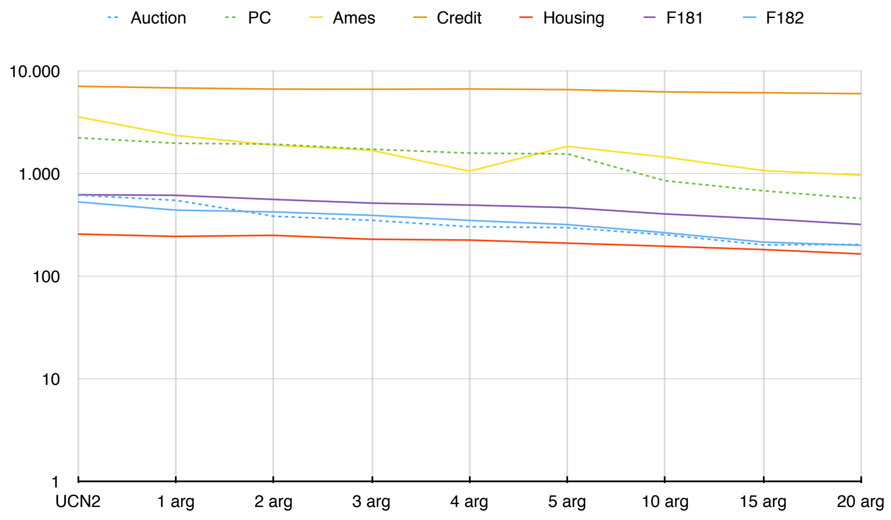


Figure 5.4: Overview of the runtimes when adding arguments for all datasets. Taken as the average runtime of five 10−fold cross validations.

a comparison of UCN2 and ABCN2 with 20 arguments.

The increase or decrease of the predictive accuracy and of the runtime can be found in table 5.3. No strong conclusion can be drawn from the data, although there seems to be a reasonable correlation between good performance of certain datasets on both statistics.

One oddity in this table is the PC dataset. As seen in the predictive accuracy data this dataset performed particularly badly, resulting in a decreased predictive accuracy when adding annotations. With regard to a reduction in the runtime necessary for the experiments it shows the strongest reduction however, although only with a small margin.

Dataset	Change in predictive accuracy	Change in runtime
Auction	40,4%	-66,9%
PC	-7,3%	-74,4%
Ames	15,6%	-73,0%
Credit	6,9%	-15,2%
Housing	7,9%	-35,9%
F181	45,2%	-48,6%
F182	16,7%	-62,2%

Table 5.3: Percent wise changes to predictive accuracy and runtimes as an effect of adding 20 annotations. Data from tables 5.1 and 5.2 was used to compile this table.

The benefits described above with regard to the reduced runtimes are not as encouraging as they might seem. The reduction in runtime only occurs during the rule generation process, while the classification process is unaffected. The runtime of the classification process is dependent on the number of rules produced, not on the availability of annotations.

It is also important to note that with the experimental setup described in chapter 4 the argument-based algorithm takes much more time to setup up than the unannotated versions. This setup will probably be used in practice as well, as the benefits in terms of predictive accuracy should be achieved before putting the system into production. The extra time taken is necessary as a number of cross validations has to be run after each annotation. Thus the time taken to produce a classifier with 20 annotated cases is much higher than that described for the individual runs in table 5.2, because a set of 5 cross validations has to be executed 20 times, with the annotations added in between.

Chapter 6

Conclusion

This thesis has tried to answer the question whether argument-based classification works in practice. To this purpose a system was built to do classification with several rule-based classification techniques and case-based reasoning as a baseline. This system will be used to help classification of complaints made at telecommunications companies in the future. The techniques and algorithms used have been developed by other authors, but are used with some minor improvements, mainly concerned with speed. The program itself is built in the most general way possible, thus allowing for future use in other applications.

Extensive testing has been done with several datasets from divergent domains. The results are generally encouraging. For some of the datasets tested adding annotations did not improve classification accuracy and in one case even reduced it. In most cases the results were positive, with the use of annotations resulting in enhanced classification accuracy.

On most datasets the unannotated classification was done with a higher accuracy by the ACN2 algorithm (using ordered rules) than it was by the UCN2 algorithm (using unordered rules). This would suggest that the ordered rulesets can be a better starting point for an accurate classifier. Unfortunately introducing annotations into the ordered classifier is much more difficult than for the unordered classifier. One of the problems is the question at which point in the process the annotated cases are used to generate a new rule. In the unordered classifier this is done first, but this does not influence whether unrelated rules discovered later in the process should fire or not. In the case of the ordered classifier starting out with rules based on the annotated cases could provide problems, because the first rule that fires determines classification.

A second problem is providing insight into the decisions made by the system. This is easier for unordered classifiers as only the firing rules are necessary for the explanation. For ordered rules all rules not firing above the one firing should be provided as well, making it much tougher to correctly understand the inherent structure for an expert. Some intelligent visualisation tools could provide improvement in this area. A potential line of future research would be the development of an argumentation-based classifier that uses ordered rulesets, thus providing a good alternative for domains where this classifier outperforms the unordered one.

Aside from the improved accuracy the use of annotations resulted in a reduction of the time necessary for the generation of rulesets. Given that this is a task that does not occur as often as classification but is still done frequently this shows a positive side to using argument-based classification methods. The use of annotations provides the algorithm with an initial direction in searching for rules, thus allowing the algorithm to ignore a part of the search space initially. This benefit mainly involves cases that are difficult to classify (as those are selected for annotation) but still has a significant impact on the overall performance of the classification methods.

It has to be noted that this benefit is negated in practice because of the time and computational complexity involved to get to the point where 20 cases are annotated. For each annotation several cross validations are necessary to calculate which case should be annotated, which is a task taking a significant amount of computing power and time.

Case based reasoning as introduced in chapter 2 has already been deemed reasonably unfit for use in practice. The results described above underscore this, providing evidence that proper use of argumentation- and rule-based systems results in similar predictive accuracies but vastly improved classification times.

These improved classification times are necessary for the application developed. The system should provide realtime feedback, as a user making a complaint is waiting for an answer. The CBR classifier does not provide this, with classification taken minutes to hours if a large case base is involved.

The algorithms used for classification make use of nominal attributes only, necessitating preprocessing to discretise other types of attributes. Including a mechanism for dealing with this automatically will improve the usefulness of these systems, although it could result in significantly higher runtimes.

The feasibility of using experts to provide improvements to classification systems has been strengthened by the research conducted. While only domains were used in which a layman could provide the annotations the algorithm to process these annotations is sound and makes use of the annotations to increase the classification accuracy. Research into using this system with actual experts in a specialistic domain needs to be conducted to validate this, but given the validity of the argument this seems straightforward.

Bibliography

Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1): 39–59, 1994.

Paul M Anglin and Ramazan Gencay. Semiparametric estimation of a hedonic price function. *Journal of Applied Econometrics*, 11(6):633–648, 1996.

Eva Armengol and Àngel García-Cerdaña. Lazy induction of descriptions using two fuzzy versions of the rand index. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems. Theory and Methods*, pages 396–405. Springer, 2010.

Eva Armengol and Enric Plaza. Lazy induction of descriptions for relational case-based learning. In *Machine Learning: ECML 2001*, pages 13–24. Springer, 2001.

Peter Clark and Robin Boswell. Rule induction with CN2: Some recent improvements. In *Machine learningEWSL-91*, pages 151–163. Springer, 1991.

Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine learning*, 3(4):261–283, 1989.

Dean De Cock. Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project. *Journal of Statistics Education*, 19(3), 2011.

James Dougherty, Ron Kohavi, Mehran Sahami, et al. Supervised and unsupervised discretization of continuous features. In *ICML*, pages 194–202, 1995.

Ray C Fair. A theory of extramarital affairs. *The Journal of Political Economy*, pages 45–61, 1978.

W Greene and D Hensher. Multinomial logit and discrete choice models. *Greene, W., LIMDEP, Version*, 7, 1997.

William H Greene. *Econometric analysis*. Pearson Education, 7th edition, 2003.

Yukihiro Matsumoto. About Ruby (retrieved June 24, 2014), 2014. URL `https://www.ruby-lang.org/en/about/`.

Ryszard S Michalski. On the quasi-minimal solution of the general covering problem. 1969.

Sanjay Modgil, Francesca Toni, Floris Bex, Ivan Bratko, Carlos I Chesñevar, Wolfgang Dvořák, Marcelo A Falappa, Xiuyi Fan, Sarah Alice Gaggl, Alejandro J García, et al. The added value of argumentation. In *Agreement Technologies*, pages 357–403. Springer, 2013.

Martin Možina, Jure Žabkar, Trevor Bench-Capon, and Ivan Bratko. Argument based machine learning applied to law. *Artificial Intelligence and Law*, 13(1): 53–73, 2005.

Martin Možina, Jure Žabkar, and Ivan Bratko. Argument based machine learning. *Artificial Intelligence*, 171(10):922–937, 2007.

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

Leonardo Rezende. Econometrics of auctions by least squares. *Journal of Applied Econometrics*, 23(7):925–948, 2008.

Christopher K Riesbeck and Roger C Schank. *Inside case-based reasoning*. Psychology Press, 2013.

Roger C Schank. *Dynamic memory: A theory of reminding and learning in computers and people*. Cambridge University Press, 1983.

Thanasis Stengos and Eleftherios Zacharias. Intertemporal pricing and price discrimination: a semiparametric hedonic analysis of the personal computer market. *Journal of Applied Econometrics*, 21(3):371–386, 2006.

Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson Education, 2006.

University of California, Irvine. Machine Learning Repository, 2014. URL `http://archive.ics.uci.edu/ml/`.

Jure Žabkar, Martin Možina, Jerneja Videčnik, and Ivan Bratko. Argument based machine learning in a medical domain. In Paul M. Dunne and Trevor J.M. Bench-Capon, editors, *Computational Models of Argument: Proceedings of COMMA 2006*, volume 144, page 59. IOS Press, 2006.