Automatic Filtering of Abuse Reports

Niklas Udd

Department of Computer and Systems Sciences
Stockholm University / Royal Institute of Technology

June 2008

This thesis corresponds to 20 weeks of full-time work. Version: 1.0.0

Abstract

This master thesis investigates how abuse reports can be automatically filtered in order to save time, lower costs and increase safety. Abuse reports are reports that users on a website file when they encounter content they find inappropriate. These reports are then generally handled by the Customer Service who decides if the content should be removed from the website. The reports that potentially can be automatically removed are the ones that do not result in deletion of content.

The study that is presented in this report took place at the community website Stardoll. A decision tree was built to classify reports as either good or bad. The over 200 attributes that were used to train the tree contained information about the user who filed the report, the user who the report was filed against and the report itself. Unfortunately no data could be extracted from the reported content.

On unseen data the decision tree correctly removed 22% of the reports that should be removed while incorrectly removing 8% of the reports that should not. These numbers are not good enough to make it feasible to start using the system without further refinements. Such refinements are outlined in this report together with suggestions for how other automated methods can be used at Stardoll and similar communities.

This report contains examples of abuse reports. The nicknames used in those examples have been changed in order to protect the privacy of the users. Any similarities between nicknames used by users on Stardoll and the nicknames used in this report are purely coincidental.

Acknowledgments

I would like to take this opportunity to thank the people who have helped me write this report and done the work presented in it. First of all I thank my supervisors Lars Askers (Stockholm University) and Mikael Krantz (Stardoll) for supporting me and giving me good advices. I would also like to thank everybody at Stardoll, especially Margareta Petersson and Johannes Schildt at the Customer Service, for all the help and suggestions they have given me. Finally I thank my good friend Elin Nilsson for providing valuable feedback on various topics throughout the work.

Table of Contents

1. Introduction....1
   1.1. Objective....2
   1.2. Study Performed....2
   1.3. Method....3
   1.4. Limitations....3
   1.5. Target Group....3
   1.6. Outline of Report....4
2. Background....5
   2.1. Automated Methods....5
   2.2. Related Work....7
   2.3. Case Studies....8
   2.4. Machine Learning....10
      2.4.1. Decision Trees....13
3. Stardoll....15
   3.1. General Information....15
   3.2. Abuse Reports at Stardoll....17
      3.2.1. Handling of Abuse Reports....19
      3.2.2. Overview of an Abuse Report....20
      3.2.3. Example of Abuse Reports....20
   3.3. Possible Uses of Automated Methods....23
      3.3.1. Ordering Reports by Priority....24
      3.3.2. Profiling Users....25
      3.3.3. Finding Specific Behavior....26
      3.3.4. Filtering out Reports....27
   3.4. Potential Benefits and Moral Implications....28
4. Preprocessing the Data....30
   4.1. Theoretical Aspects....30
      4.1.1. Preparation....30
      4.1.2. Collecting Attributes....32
      4.1.3. Recovery of Data....35
   4.2. Preparation....37
   4.3. Attributes in the Report....38
   4.4. Attributes Suggested by the Customer Service....40
   4.5. Attributes Used....42
   4.6. Recovery of Data....43
   4.7. Lack of Data....45
5. Training the Classifier....47
   5.1. Theoretical Aspects....47
      5.1.1. Large Data Volumes....49
      5.1.2. Ensemble Methods....50
   5.2. Weka....51
   5.3. Choosing Classification Algorithm....52
   5.4. Choosing Parameters....55
   5.5. Finding the Most Suitable Subset....58
   5.6. Favoring Memory Instead of Time....61
   5.7. Increasing Memory....65
   5.8. Creating a Tree Based on Fewer Attributes....65
   5.9. Bagging and Boosting....67
   5.10. Building a Custom Ensemble System....69

5.11. Ensemble with Veto....73
5.12. Description of the Classifier....74
6. Using the Classifier....76
6.1. Theoretical Aspects....76
6.2. StarClassifier....78
6.3. Handling of Rejected Reports....79
6.4. Protected Reports....80
7. Evaluating the Classifier....82
7.1. Evaluation on New Data....82
7.2. External Factors Affecting the Result....85
7.3. Interviews....87
7.3.1. Customer Service....87
7.3.2. Management....88
7.3.3. Development....89
8. Result and Analysis....91
8.1. Conclusions....91
8.2. Contributions....92
8.3. Lessons Learned....93
8.4. Future Research....94
8.4.1. Improved Automatic Filtering....94
8.4.2. Other Automated Methods....95
8.4.3. Other Approaches....96
9. References....98
Appendix A: StarClassifier....101
Appendix B: Effects of Individual Attributes....109

1. Introduction

We live in an information rich society where we all, corporations as well as people, have to handle and process large amounts of information. One contributing factor is the Internet which has made it possible for everyone to create and share data in a way that has not been easily done before. This is in itself something positive; proper information makes it possible to invest in the right stocks at the right time, reviews written by owners of a specific product makes it possible to buy a product that fulfills our needs, and having social networking sites informing us about where our friends are traveling makes it easier to schedule a meeting.

The problem is that the amount of information that is presented to us is so vast that it is difficult and time-consuming to process it all. A solution to this is to employ an automated method. Such methods can take many shapes, but examples include systems that filter out information, assign priorities to it, or handle all information on its own and make a decision based on the information it has processed.

Many of us are using automated methods in our everyday life. It is likely that a spam filtering system is going through your emails [Gra02], that the search engine you are using is sorting the results by relevance or priority [PNM+98], and that your credit card company automatically monitors your card and puts a hold on it if the usage pattern indicates fraudulent behavior [FP97].

The increase in amount of data created might be most visible on the Internet. This should especially be true for websites letting users create a large part of the content themselves, something that is often true for Web 2.0 sites. If everyone can contribute to the content on the site it is likely that the amount of information will increase as the number of sources increase. In addition to the increased amount of content that is explicitly created, the amount of automatically gathered, or generated, information is also likely to rise. A shop owner on the Internet is for example able to track customers in a way that is harder to do in a physical store. It is possible to collect information about where people come from, who referred them and how much time they spend looking at items before they buy them.

1.1. Objective

The purpose of this thesis was to investigate the benefits, drawbacks and feasibility of employing an automated method on a community website, and examine potential efficiency improvements, in terms of time, resources and funds, made by using such a method.

Community websites and social networking sites are getting more and more attention and they are gaining in popularity. The popular social networking site Facebook currently has more active users (67 million [Fac08a]) than many countries, including United Kingdom, Canada and France, have citizens. Many sites, even if they are not originally thought of as community websites, are today at least partly similar in nature to traditional community sites. Video, link sharing and gaming sites are just examples of sites that commonly have a community function on top of some other primary function.

1.2. Study Performed

There are many data flows on a community website. Some of these are created by the users directly, for example by writing posts in a forum, and others are generated automatically by logging and similar systems. Most of these flows are not regularly reviewed by the staff of the website and does not give much opportunity for improving efficiency. One reviewed flow that is often present on these sites is a flow of abuse reports. It is often possible to file reports against other users when they behave inappropriately. These reports are generally reviewed by the staff manually and the task can be very resource-intensive.

The study that was performed aimed at building a system that should improve the process of handling abuse reports on a community site. Abuse reports are written by users when they discover an abuse, which for example could be an offending post. The reports are read by the Customer Service which will decide what action to take.

Several automated methods were considered but the one that this study was focused on dealt with filtering. A significant part of all reports that are filed on the investigated site are in one or another way bad. These reports are, once their quality has been determined, discarded. The system built aimed at automatically finding and removing these bad reports. If successful the Customer Service's workload would decrease most notably.

The system was based on machine learning, which is a method for letting computers “learn” without being explicitly taught. The exact technique used was a decision tree algorithm. These concepts are explained in Section 2.4 on page 10.

The study was performed at Stardoll, a community site with over seventeen million users. More about Stardoll and their abuse reports can be read in Section 3 on page 15.

1.3. Method

The work was divided into three phases. The first phase consisted of doing initial investigation about the subject and deciding the details about the study. The second phase was the technical phase where the automated method was built. The third and last phase analyzed the results.

During the first phase the details of the work were set. Different automated methods for improving the handling of abuse reports were considered and evaluated. Input was taken from previous work in the area, case studies and discussions with Stardoll employees.

Once the details were decided upon the technical phase could be started. During that phase the automated method was built with the help of machine learning techniques. Existing algorithms were used to test and compare different approaches. The whole technical phase was in itself divided into a few logical steps. The first step performed was preprocessing the data which includes gathering and transforming data, which was needed before any further work could be done. The second step consisted of using the data in order to build a model that would later be the heart of the automated method. This step was iterative; an initial approach was tested after which other approaches and refinements were tested in order to improve the initial result. The third and last step included creating the tools needed in order to practically employ the automated method.

When an automated method had been built it was evaluated both quantitatively and qualitatively. The former was done by using the standard method for measuring an automated method's estimated performance. The latter was done by interviewing people who in one or another way would be using the system.

After the evaluation of this particular automated method was done the result of the study was analyzed on a more general level. How this particular result and automated methods in general could fit into a bigger picture were discussed. The objective set up above was also reviewed.

1.4. Limitations

The focus of this study has been on investigating how an automated method could be employed and different aspects that need to be thought of when doing so. The goal has not been to get an industrial-strength system up and running, but a prototype has been created in order to show the feasibility. Some non-technical aspects, such as moral implications, were also considered.

1.5. Target Group

The contents of this report can be interesting from several points of view. Potential readers include researchers who are doing related work, employees of large websites who are interested in learning if and how automated methods can

improve efficiency, and people how are generally interested in this increasingly important field. These different readers are likely to have different background knowledge; some might be experts in machine learning but novices when it comes to websites and vice versa. Because of this much background information is available in this report to make it rather self-contained. Some details might be difficult to understand without any additional background information, but the general picture should still be graspable.

1.6. Outline of Report

This report is divided into sections based on the different phases outlined in the previous section describing the used method. The different sections will be shortly described below. It is possible for readers well familiar with certain fields to skip the subsequent sections.

The first section, *Background*, provides general information about automated methods and what has previously been done in the field. An introduction to machine learning in general and specifically the decision tree technique is also provided.

The next section, *Stardoll*, describes the community Stardoll, how abuse reports are handled today and possible automated methods that could be put in place.

The following three sections all deal with building the automated method. These sections start with a theoretical introduction to the task to be performed. After this general introduction a description of what has been done and the results from it are provided.

Before the data can be processed by a machine learning algorithm it has to be preprocessed. How this was done is described in the section *Preprocessing the Data*. Once the data was ready a machine learning algorithm was used to train a classifier, which is described in *Training the Classifier*. Finally, in *Using the Classifier*, the trained classifier was made practically useful.

The following section, *Evaluating the Classifier*, tries to evaluate the classifier by measuring its performance and interviewing people about the result.

The last section, *Result and Analysis*, evaluates the result on a more general level. Conclusions, contributions and some ideas for future research are listed.

In addition to the sections mentioned above the report has two appendixes. The first, *StarClassifier*, contains elaborate instructions for using the tool StarClassifier which has been developed as a part of this work. The second, *Effects of Individual Attributes*, presents the most important groups of factors that affect the quality of an abuse report.

2. Background

Some background information is needed in order to fully understand the study that has been performed. This section aims to give the necessary information to those not previously acquainted with automated methods. The first subsection explains and discusses automated methods in general and stresses the importance of such methods. The following two subsections provide the context of this work by stating was has been done previously. The last subsection focuses on the technical methods; machine learning in general and the specific decision tree technique will be described.

2.1. Automated Methods

Automated methods are becoming increasingly important in our society as the amount of information raises. It is in many domains difficult and time-consuming, or even impossible, to manually manage all data. In those cases an automated method can help by organizing and processing the information. One example of this type of application is a search engine which aims at finding specific web pages. In this situation it is easy to see that an automated method can help since the number of web pages available is so vast that it is not feasible to manually go through them all.

In other domains the amount of data available might be more than what is initially apparent and in these situations the potential benefits of employing an automated method might not be obvious. Finding interesting news articles is one example belonging to this category. There is probably only a few people who see the need to use an automated method in this domain, but the number of articles published each day is so large that it is infeasible to skim trough them all in the hunt for what is interesting. Most of us probably stick to one source and skim through their articles. By using this method it is easy to miss good articles that would have been appreciated, simply because they did not appear in the news source we chose. One possible solution is to let an automated system go through articles we like and try to find patterns based on characteristics of the article. Once trained the system could go through huge amounts of articles and notify us about articles that we are likely to appreciate, even if they were written in a paper we have never heard about.

Automated methods can take on many shapes. It was shown above that they are applicable in a wide variety of domains and it is also true that they can be used for many different applications. Below is a list of four categories of automated methods together with examples of systems that would belong in that category. It should be noted that these categories are listed the way they are simply to show that automated methods can be applied in different ways. This is not an attempt to provide a formal classification scheme since the list is neither complete nor disjoint. These issues will be discussed more in details after the list has been presented.

Automatic filtering. Two classes of data, where one is typically interesting and one is not, are often mixed together in the same data flow. An automated filtering system can help separate these two classes by looking at different properties of the available entities. One example where this is utilized is emails. Legitimate email messages are mixed together with spam messages. The recipient is only interesting in the first of these two classes and it is desirable to filter away all spam messages.

Automatic priority assignment. Sometimes the different classes within a data flow is not very important, but the priority might be. Different entities can have different priorities and handling the most important entity first can be beneficial. In the unfortunate event of a large accident it is likely that many people will almost simultaneously call the emergency line (e.g., 112 or 911). In such situations it is possible that the number of operators available can not answer to all calls instantly. A system could then automatically assign priorities to calls and forward the calls to operators according to priority. One factor that might be interesting to look at is the geographical distribution of callers. It might be reasonable to assume that the accident has taken place in the center of this distribution and that the caller closest to this point is the one that has the most accurate and elaborate information.

Automatic detection. A flow of data can occasionally contain a pattern that is desirable to find once it appears. One example of this is an Intrusion Detection Systems (IDS) which monitors computer systems in order to find malicious behavior. Automated detection can be used to find these patterns. It should be noted that automatic detection is closely related to automatic filtering and the difference appears sometimes only on a conceptual level, which will be further discussed below.

Automatic handling. In some situations a system that could by itself make decisions based on the data collected is preferable. One example of such a system is the type of system mentioned earlier that goes through all credit card transactions and immediately puts a hold on cards that shows fraudulent behavior. Acting quickly makes it possible to protect money and make fraud less appealing. There is no reason to get into fraud if the cards are blocked so that no money can be earned.

It was mentioned earlier that the list is not a very good categorization of automated methods, but it should clearly show the wide variety of applications for automated methods. One problem with the categorization is that one category can be seen as a special case of another. Automatic filtering can for example be seen as automatic handling where the action in question is to remove or, depending on perspective, save entities. Automatic handling can, in turn, be seen as automatic detection followed by some actions taken once an item has been detected. The action in itself might be completely separated from the rest of the automated method and this addition should not change any of the underlying techniques. In fact, all of the above automated methods can technically look very similar. In the same way it is also possible that different systems within the same category look very different technically.

2.2. Related Work

A good starting point when conducting a study such as this is to investigate what work has previously been done in the field. Unfortunately it was hard to find any published material about work related to the use of automatic methods on large websites. The domain is quite narrow and the number of large websites with much user generated material is not unlimited, even though many sites matching that description exist. On the other hand automated methods are likely to be usable and perhaps even necessary in this precise domain. It is likely that work has been done in this field without information about it has been published. One possible explanation for this is that the research might have taken place at the site in question and been considered normal development rather than research. A case study, which will be described in the following section, was performed in order to investigate if any unpublished work had been performed.

If the focus is turned to employing automated methods in the field of customer service reports some related work has been done. Lenz, Hübner and Kunze [LHK98] discuss how a Customer Service can be helped by having the right documents presented to them automatically. The idea is that the system should be fed with a description of the problem at hand and then analyze the problem and determine which documents the Customer Service Representative should get in return. The documents can be troubleshooting guides, instructions about how to fix a certain problem or other documents that the system finds relevant. Brüninghaus and Ashley [BA97] takes on a somewhat more general approach. Their idea is to extract important features from the description which later can be fed into another system than can reason with the case in some way. Both these approaches have focused on the textual description in itself, meaning all relevant information about the case has been believed to be in the textual description. This is not a preferable approach in the case at hand since it only focuses on one small part of all the data available. In many situations, and especially on websites, the amount of

information that is, or could be, available is much larger than the information that is explicitly made available. In the case of a Customer Service for a website factors such as the user's environment (e.g., browser and operating system) and the user's usage pattern (e.g., how often the user has visited the website) can be important when troubleshooting a problem the user is experiencing on the website.

Much work has been and is still being done in the field of automated methods in general. These studies are performed in many different domains, some very far from the one seen here. Despite this the underlying techniques and considerations are sometimes very similar. It appears that the method itself and factors such as amount and properties of available data are more important than the domain. In a study perform by Burl et al. [BAS+98] images showing the surface of Venus were automatically analyzed in order to find volcanoes. At a first glance this study seems totally different from the study that is presented in this report, but the fact is that these two have many similarities. These are complicated to explain without referring to details of the work that has not yet been described. The similarities will be brought up again later in the report where they can be more easily explained.

2.3. Case Studies

Since the search for finding published material about similar work turned up empty the focus was turned to other communities. Other large communities, and especially the largest, are likely to have encountered problems with the large amount of data in the same way Stardoll has. An inquiry was sent to some of them asking if they had considered automated methods and if they, in that case, had implemented a system utilizing one.

A list of communities was found on Wikipedia [Wik07] and the ten largest ones were contacted. It is not safe to say that this list is perfectly correct, but the exact member count is not important. The point was that big communities should be contacted and the list should be accurate enough for that purpose. It is, furthermore, a difficult task to make a perfect list since some communities do not release exact numbers. One aspect that makes Stardoll different from many other communities is the age of the users. It is possible that other communities which also have a large amount of abuse reports to handle have solutions that will not work for Stardoll, and vice versa. The characteristics of abuse reports at Stardoll appear to be heavily influenced by the young user group (see Section 3 on page 15). One example of this is when two friends on Stardoll know each other in real life and fall out with each other there. Sometimes this is handled by filing false reports against each other on Stardoll. This behavior is probably not as common on sites with an older user group, such as LinkedIn. In order to get responses from communities with a user group similar to Stardoll's a few communities which

focus on young people were added to the list. In total the list of communities to contact consisted of 14 names.

Only three of the 14 contacted communities answered, namely Bebo, Classmates.com and Facebook. Unfortunately none of them could provide any information in the matter: Bebo referred to their privacy policy, Classmates.com to the proprietary nature of the information and Facebook claimed to be unable to answer any research questions.

Unfortunately not much information could be extracted from the answers received, but some conclusions can be drawn anyway. The best example of this is when the inquiry was sent to Faceparty and an automatic reply was received shortly afterwards. The reply contained information stating that the question had been understood and answered by a machine and that it afterwards had been deleted, hence no further action would be taken on their part. This is a type of automatic handling. Unfortunately the automatic reply system misunderstood the questions completely and answered to something else.[1] Another message was sent asking more about the system but no reply has been received. Worth mentioning about Faceparty is that paying members, users which have purchased the Cool Tools package, are not filtered through this system. They are instead guaranteed a personal answer within 48 hours. [Fac08b] Friendster also uses an automated, but less offensive, system. After they have been contacted a confirmation letter is sent. That letter contains links to information pages that are believed to be relevant, but the original message is still kept.

Some information can also be extracted by reading the information available on the communities. Xanga states that "we don't have the resources to fully investigate every abuse report"[2] [Xan08]. It can not be concluded if this means that they are investigating every report, but not fully, or if they simply do not investigate some of the reports. This quote might imply that they are filtering out some reports. Bebo on the other hand states that "a member of staff will personally review all reported violations" [Beb08]. This means that they can not filter out any reports, but they could still be using other automated method, such as automatic assignment of priority, to support the Customer Service.

In summary it can be concluded that automated methods are used by a few communities and also that communities in general are unwilling to answer questions about this. If the unwillingness is due to general issues, such as communities looking at each other as potential competitors, or if automated methods is extra sensitive can not be concluded. Automating parts of the Customer Service might be a sensitive issue. It has moral implications and it might also be

1 To be fair it should be stated that the type of questions sent must be rather atypical.

2 It should be mentioned that this quote is taken out of context. It can be found on a page dedicated to police officers. The full quote reads "While we don't have the resources to fully investigate every abuse report, we are always happy to cooperate with police investigations".

the case that some measures can be countered if people know that they are in place.

2.4. Machine Learning

Machine learning is a field in computer science, closely related to the perhaps more well-known field of data mining, that focuses on developing ways for computers to "learn" without being taught explicitly. The idea is that a computer, or rather a computer system, should be able to observe reality and learn from observations the same way humans can. This contrasts the traditional model where the computer is instructed explicitly what to do. Somewhat simplified the type of machine learning used here consists of two phases: the training phase and the classification phase. During the first phase the computer is observing the world in order to find patterns and learn how it works. Once the computer has acquired the necessary skills it can move on to the next phase in which it makes decisions based on the things it has learned and the information it has acquired. One example might be a computer that is being trained to understand the stock market. In the beginning this computer will be fed with information about stock quotes, company acquisitions, annual reports and the like. The computer might after a while predict patterns in this data and a new phase can begin where the computer is fed with similar data, but this time it makes decision whether to buy or sell stocks based on its prediction for what will happen in the future.

Machine learning comes in different shapes and the type being considered here makes classifications, which means that the computer tries to predict one class out of a set of possible ones. In order to facilitate this a so-called classifier has to be built, which is done during the training phase. During this phase the computer is shown items, called examples. These examples are labeled with one of the available classes. The computer will try to extract patterns from this data in order to understand the connection between (specific properties of) the examples and the classes. As an example let us say that the task at hand is for the computer to learn to distinguish between good and bad pizzas. In this case the computer is shown pizzas and told whether that specific pizza is tasty or not. The classes here are "TASTY" and "NOT TASTY". After a while the computer has hopefully been able to spot patterns and been able to create a generalization, a model, from this. The model can be simple (e.g., "all pizzas are good except the ones with curry on") or more complex. Once a model is generated the next phase, the classification phase, can be started. During this phase the computer can be shown pizzas and use its knowledge, the model, to decide whether it is tasty or not. If the simple model above turned out to be correct the computer would simply look to see if the pizza had curry on it and make a decision based on that.

The reference to "the computer" above is made in order to make the reasoning more straight-forward. In reality there is no mystical computer that can do all this.

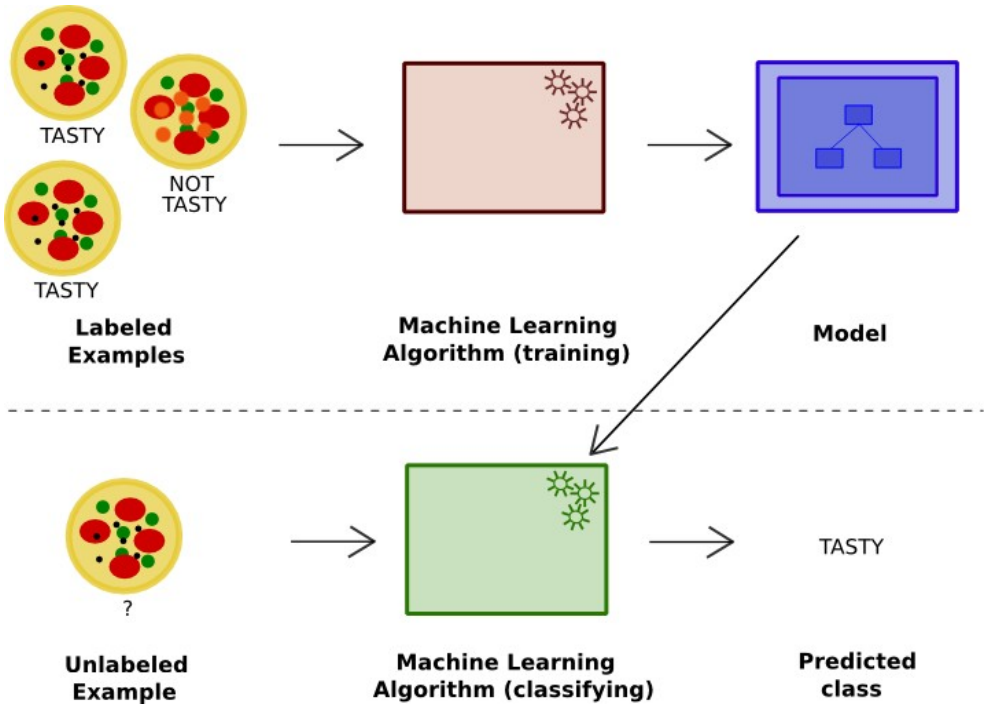

Figure 1: Overview of machine learning.

The actual learning, which is a set of computations, is conducted with the help of an algorithm. The labeled training examples are fed into the algorithm which then returns some result, the model. This result together with an unlabeled example is then fed into another algorithm which will output an estimated class. This process is illustrated in Figure 1. Many machine learning algorithms exist and one of them will be explained later. Before going into details about any specific algorithm some more attention will be put on general machine learning. In order not to make it too cumbersome and abstract the explanation will not be fully general and techniques exist that do not fit into the pattern described here, but that type of algorithms are not considered in this thesis.

No matter what the examples are supposed to represent the form of the representation is important. In the pizza example things were simplified by saying that the computer looked at pizzas. From a conceptual standpoint this is easy to grasp, but it is hard in reality to understand how this will be done. The computer is not able to interact and investigate the pizza in the same way as a human being faced with the same problem would. The actual pizza has to be represented in some way before it can be fed into an algorithm. This representation can take many different forms and choosing which one to apply is important. Among other things it is possible to represent it with the dimensions of the pizza, the number of ingredients or the type of ingredients.

To complicate things further the best representation depends on the goal of the whole exercise. If the goal is to make a computer distinguish between American

and Italian pizzas the height is probably important. If the goal is to figure out how much a pizza will cost the number of ingredients might be an important factor. If, on the other side, the tastefulness is to be decided as in the example above the type of ingredients must be of high importance. One might ask oneself if the simplest thing would not be to use all possible attributes. The algorithm that is being used should be intelligent enough to find the important attributes and ignore the rest, and this is indeed true. The problem is that even in uncomplicated examples like this the amount of possible attributes, or properties, is enormous. Calculating all is simply infeasible. The pizza can, for example, be represented by the day it was made, how long time it took to make it, its height above sea level, in which country it was made, its temperature, in what type of oven it was made, if the ingredients were above or below the dough, the name of the pizza baker's dog, the velocity of the pizza, and so on. Another problem that can arise if all attributes are used is that coincidental patterns can emerge. This is likely to happen if attributes are used that have many values. Say for example that one attribute is the pizza baker's age in milliseconds. It is unlikely that several pizzas will have the exact same age; hence pizzas can be perfectly divided into tasty and not tasty by writing down a list of ages that produce these two types of pizzas. These types of patterns are only coincidental and not actually useful. This type of problem is likely to occur if an excessive number of attributes are used.

As seen above the best representation to use is hard to find but important to get right. A great help in finding an appropriate representation is to ask people well acquainted with the field, so-called domain experts, about their ideas. Someone well-familiar with pizzas can tell us that the velocity of a pizza does not have anything to do with the pizza being American or Italian. In a similar way such an expert can point in a direction where extra attention should be focused. It might in this case very well be that the ingredients should get as much attention as possible. Potential domain experts in this example are pizza eaters, pizza bakers, professors in physiology, and so on. The choice of appropriate expert may also depend on the goal.

As illustrated above the type of machine learning algorithm needed must be able to handle different types of data. The number of ingredients is represented by a numeric value where there is an internal order. Rules concerning these values usually split them at some specific number in order to make two classes. One example is the rule “pizzas with fewer than three ingredients are cheap”. On the other hand there are also attributes, called categorical or nominal, which have a set of possible values. One example of such an attribute is the country in which a pizza is made. A rule such as “pizzas made in a country less than Norway taste bad” does not make sense. Instead specific rules must be made for each of the possible values. The issue with different data types is present concerning the output too. In some cases the result should be a numeric value rather than a

categorical. This might for example be true when determining the estimated price of a pizza.

Except for delivering a result, like a class, some algorithms can also determine a confidence level. This level is an indication of how certain the result is, in other words how much confidence one should have in it. This can be very useful when incorrect classifications are more expensive in one direction than the other. A spam filter for example tries to remove unsolicited and unwanted email messages based on a set of criteria. If the filter is sure that the email is either legit, often called ham, or spam it is clear what action to take. If the filter on the other hand is unsure it is much better to keep the potential spam than to delete it risking the loss of a real, possibly important, email.

It is important to avoid overfitting during machine learning. Overfitting means that the model that has been trained so intensely on one set of example that it is not likely to generalize well. Recall once again the pizza example. In situations like this it is not unusual to assign id numbers to all items in order to easier keep track of them. One possibility is that the machine learning algorithm would use these id numbers and find a "pattern" among them (e.g., "pizzas with id numbers 2, 3, 5 and 7 are tasty, while 1, 4, 6, 8 and 9 are not tasty"). This would not generalize well at all and the model would be unable to handle a new unlabeled pizza with id 10. To avoid this overfitting it is important to test the model on new data that was not available during training. If performance is satisfactory on unseen data the classifier is believed not to be overfitted.

2.4.1. Decision Trees

One widely used algorithm in machine learning is the decision tree algorithm. Strictly speaking there is not just one decision tree algorithm; instead there are several slightly different algorithms available. The general idea is nonetheless the same and the focus will be on the principles so the exact internal workings are not important here. What a decision tree algorithm does is build a decision tree during training. The tree in itself is then traversed when a classification is done.

A decision tree is a tree where the nodes are conditions, the edges are results to those conditions and the leaves are decisions. This may sound more complex than it is, hence an example might be helpful. Figure 2 shows a decision tree from the pizza example used earlier. This tree is a model that an algorithm has produced. The idea is that anyone who wants to know if a certain pizza is tasty or not can consult this decision tree. The first step to take in such a process is to look at the root of the tree. This is a node and should therefore have a condition. In this case it says "curry?" which refers to if the pizza has curry on it or not. Whether the pizza has this or not defines which of the two possible edges to follow. If the pizza does not have curry on it another node will have to be processed in the same way. Sooner or later a leaf will be reached. The leaf contains the estimated class. The

tree presented here is very small in order to keep the example simple. Real decision trees are generally much larger.

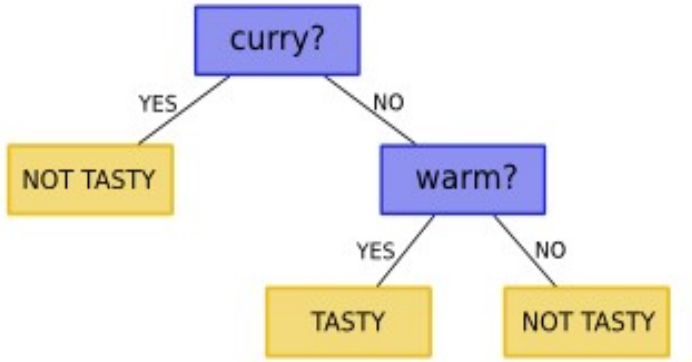

Figure 2: A simple decision tree with two nodes representing conditions and three leaves.

One characteristic of decision trees is that they are very readable. A decision tree can quite easily be read and understood by someone who is not acquainted with machine learning. Even if the tree in itself is not understood it is, without much effort, possible to translate the tree into rules. The rules produced can be quite formal, but except for that they resemble familiar structures such as instructions or simple laws. The example tree above can be translated to the following list of rules. It can afterwards easily be translated to less formal rules that should be easily understood by everyone.

If the pizza contains curry **then** it is not tasty
If the pizza does not contain curry **and** is not warm **then** it is not tasty
If the pizza does not contain curry **and** is warm **then** it is tasty

Having a readable model is especially helpful when talking to domain experts. The transparency provided by this solution is also good for "sanity checks" on the model. Actually seeing the model is good to avoid overfitting, avoid trusting coincidental patterns, complying with applicable laws[3] and so on. A model that anyone can look at and understand might also have the benefit of easier being trusted than an opaque model looking like a black box.

Another appealing aspect of decision trees is that only used attributes need to be calculated. In order to do a classification all attributes from the root down to the leaf in question needs to be calculated, but there is no need to consider other properties. In some cases this is not so important but if the properties in question are time-consuming to calculate this might be appealing.

3 It is in some countries illegal to discriminate people based on for example race [WF05].

3. Stardoll

This section aims at providing all information about the community at which this work has taken place, Stardoll, needed in order to understand the rest of this report. Some of the information presented is not strictly needed, but it is provided for those who want to fully understand the context. Some terms used later on, such as stardollars and starpoints, are explained in this section. This section also explains in details how abuse reports are structured and how they are currently handled. Finally some different approaches for automated methods are listed together with a discussion about the effects of those methods.

The description about Stardoll given here is rather detailed. It should be stressed that this does not imply that the work performed is only relevant under these exact circumstances. Many community sites have features in common, but they are sometimes presented under different names. The work presented should be fairly generalizable.

3.1. General Information

Stardoll is a community site which lets its users play with and dress up paperdolls. Along with hundreds of celebrity dolls all users can create their own doll representing themselves, a so-called MeDoll. The focus on the site is on fashion and celebrities, where the dress-up game is a very central part. In addition to this the site has all the features that can be expected from a social networking site including profile pages, instant messaging, chat, friend lists, guestbooks, blogs and clubs.

The part of the site where celebrities are dressed up is open to everybody and there is no need to register in order to access those games. The user is given the opportunity to choose between over 400 celebrity dolls to dress up. When the preferred doll is selected the user will be taken to the Flash-based dress-up game were the user can dress the doll with the clothes available in the game. On top of the celebrity dress-up a few other games are available for everybody to play. One example of such a game is a game where the user should find clothes that have been hidden among other objects.

A user who decides to sign-up for a free membership on the site gets, in addition to what was mentioned in the previous paragraph, access to the

community section of the site. This means that the user can create their own space including a MeDoll, a profile page, a guestbook, a blog and other similar features. The MeDoll, which can be created so that it looks like the user does in real life, has only underwear on from the beginning. Additional clothes, and other accessories for the doll, can be bought in the virtual shop Starplaza.

In order to buy items on Starplaza, or use a service that costs money, a virtual currency, stardollars, are used. All new members are originally rewarded 25 stardollars and active members will occasionally receive more by earning starpoints. Starpoints are each night handed out to users that have been active during the day. Writing guestbook entries, blog entries, comments about dolls and performing other activities that enrich Stardoll gives starpoints to the author according to a specific scheme. When a user has certain amounts of starpoints they will receive a bonus in some form, one of which is more stardollars. If this is not enough for a user it is possible to buy extra by using real-life money. To protect children from spending more than they should an upper limit is set for how much stardollars that can be bought in a certain amount of time.

Being a member makes almost all features available, but there are a few which only can be accessed by superstars. Superstar is a time-limited V.I.P. membership which gives the user access some additional features and also lets the user access some functions and dolls before they are made available to everyone. Users who buy stardollars automatically become superstars for a certain period of time.

Celebrities are a central part of Stardoll. The dress-up game is centered on them and many users are passionate fans of one or more celebrities. In order to make the celebrities more alive a concept called RealCeleb has been founded. The idea is that real celebrities register on Stardoll in order to get access to the same features as a regular user would have. In this way it is possible for a user to have these celebrities as their Stardoll friends, sign their guestbooks and read more about them on their profile pages. This also gives celebrities opportunities to reach out to their fans. Current RealCelebs include Hillary Duff, Avril Lavigne and Heidi Klum.

The main target group for Stardoll is girls in the ages seven to seventeen who are interested in fashion. The site currently has more than seventeen million registered members, out of which 91% is female and 67% between seven and seventeen. The site is available in 15 languages and its members come from over 250 countries around the globe. During 2007 Stardoll won an award in the AlwaysOn 100 Top Companies awards [Kel07]. The site was also a winner in the Cnet sponsored Webware 100 awards 2007, which ranked the best web 2.0 sites [Web07].

3.2. Abuse Reports at Stardoll

In order to provide a friendly environment, Stardoll has rules that all users must follow but unfortunately these are sometimes violated. With such a big user community it is impossible for the staff to keep track of all offenses on their own, so they have to rely the users. If someone sees something that is against the rules they are supposed to report it to the Customer Service, which will investigate it and take the appropriate actions. Notifying the Customer Service about a potential abuse is done by filing an abuse report, which is easily done by clicking on an exclamation mark (!) icon. These signs are available almost everywhere on the site and are associated with an object. In a guestbook for example all posts have an exclamation mark after them. If one of those posts would violate the rules the user is supposed to click the mark that is associated with that particular post. Figure 3 shows an example of a post with an exclamation mark associated with it.

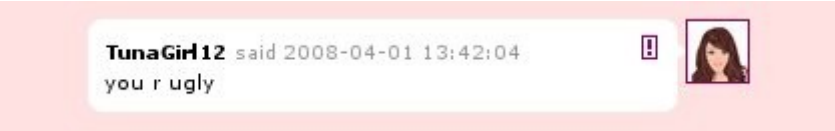

Figure 3: An example post containing inappropriate content.

A form for filing the abuse report is shown to the user after the exclamation mark has been clicked. The user is asked to provide information about the offense in order to simplify the handling of the report. The first thing the user will be asked about is the nature of the offense. The answer should be one of the following:

- bad language,
- threats,
- shows email, phone or address,
- asks for email, phone or address,
- asks for password,
- other.

After the category has been selected the user is offered to write a description of the offense. This is done in an ordinary text field and the user can choose between ignoring the field altogether or writing an elaborate description of the offense in question.

Except for the two explicit fields (category and description) mentioned above, other information is associated with the report automatically. This automatically attached data contains a reference to the reported object and id numbers for both the reporting and the reported user. All users on Stardoll have a unique id number from which it is possible to find the user in question and get additional information if that is necessary. In addition to this information some metadata, such as date, is saved.

All the reported objects have a textual representation. This might seem trivial, but it is worth pointing out. Some objects, such as guestbook entries, are already textual by nature so in such a case the difference between the object itself and a textual representation of the object is only philosophical. For some other objects this distinction can be more important. A user's profile page can for example be formatted with different text colors, sizes, fonts and so on. It is worth to consider how this should be handled and decide whether to include formatting information in the textual representations or not.[4] Some object that are present on Stardoll do not have an obvious textual representation. One such object type is images that the users can upload. These object can, however, not be reported. Images are checked by the Customer Service after they have been uploaded but before they are shown on the site. Because of this all images that are seen on Stardoll have already been approved hence there is no need to report them.

The users of Stardoll are filing approximately 20000 reports each week. The aim is to handle all reports within three days, but sometimes this period can be substantially longer. At the time of writing the oldest report that has not been handled was filed eleven days ago, but normally delays are shorter. Problems can occur that causes tops as high as this or even higher. During the process users can not track the progress of their reports and they will not be notified about how it was handled.

About 39% of all reports that are being filed are about actual offenses. The rest are not and these reports are, once it has been established that they are false reports, ignored. Incorrect use of the reporting system can take many forms, but four of the most common ones are listed below.

Reported behavior does not violate the rules. The users of Stardoll do not always know what is, and what is not, an accepted behavior, which result in reports being filed about behavior the individual user finds inappropriate but are allowed on Stardoll. An example of a report falling into this category is a report filed against a user who states that they are atheists. One user may find it inappropriate to say that God does not exist, but this is not against any rules.

Offense done somewhere else. Sometimes reports are filed about actions taken on another site or network, such as MSN Messenger. The Customer Service at Stardoll can not investigate this further and must ignore the report.

Pure abuse of reporting system. It is quite common that friends suddenly fall out with each other in real life and sometimes this result in reports on Stardoll. These reports are generally accusations about the former friend now selling drugs, threatens to kill everyone or doing other extreme activities. Reports can also be filed out of jealousy against people who get more attention.

4 It has been found while doing work on spam filters that ff0000, a representation for bright red, is a good indicator that a message is spam. [Gra02]

Report sent instead of instant message. Stardoll tries to be very clear about what filing a report means, but the system is at times misunderstood anyway. The most common mistake is to use the system in order to send an instant message to the reported users.

3.2.1. Handling of Abuse Reports

As mentioned above the reports are handled by the Customer Service, which reviews the potential offense and checks it against Stardoll's rules. If the reported potential offense was a real offense and a violation of the rules the appropriate actions are taken. The possible actions to take are to send a warning to the user or to delete the user right away. There are two different levels of warnings. The first is called a soft warning and the second a final warning. The latter type is more severe than the first and it will be followed up by the Customer Service after a certain period of time. If the user has not changed either their general behavior or some specific object, such as an offensive wording, within that time the account will be deleted. Three soft warnings can also cause the deletion of the account. Accounts can also, as stated above, be deleted without any prior warning but this is seldom done. It might be done if it is likely that the main reason, or only reason, the user is on Stardoll violates the rules. One example of such behavior is when users sign up for an account only to find someone to have cyber sex with.

All reports that have not yet been handled are listed in the administrative interface in the same order they were written. The main workflow is for the Customer Service Representative to take the first report in the queue, analyze it, take the appropriate actions, set the report quality, and then move on to the next. Setting the report quality means classifying the report as being good, bad or nonsense. The first category, good, means that the report was about an actual offense and that actions were taken. The second and third category means the opposite and that no action was taken. Originally a distinction was made between a bad report and a nonsense report. The first was when the reporting system was used correctly but the reporting user and the staff did not agree on whether it violated the rules or not. The report mentioned above about reporting someone because they were an atheist is an example on a bad report. A nonsense report on the other hand is a result of the reporting system being used incorrectly. The previously mentioned example user who tried to send instant messages via the reporting system would file nonsense reports. A rule of thumb is that bad reports are filed when the rules are misunderstood and nonsense reports are filed when the system is misunderstood. This distinction has not always been honored and some different rules for distinguishing between bad and nonsense exist so the classifications among old reports are not consistent. From here on the distinction will therefore be ignore and in the rest of this report abuse reports will be said to be either good or bad.

In addition to the standard first-in, first-out queue a view listing the most reported users are available. This lists the users who have most reports filed against them. This makes it possible for the Customer Service to spot if some user is causing a lot of trouble on the site. A user signing-up for a membership only to violate the rules, a so-called troll, will generally cause many other users to file reports shortly after one another. If only the queue view were used it would take a few days before the first report about this specific user reached the top and was handled. In the meantime this user could have caused lot of trouble and upset many users resulting in unsatisfied users and many reports for the Customer Service to handle. With the help of the additional view this type of behavior can be stopped earlier than it would have otherwise, since the troll quickly would end up being one of the most reported users.

3.2.2. Overview of an Abuse Report

Figure 4 shows an overview of an abuse report. The user PrettyFayed has written a message in a guestbook promising to give the owner of the guestbook 100 stardollars if they reveal their password. Asking a user for their password is a violation of Stardoll's rules so users who see this message are supposed to report it by filing an abuse report. The user chloe92 sees this entry and files an abuse report. The corresponding category is chosen and an informative description is written.

Associated with the abuse report are the two involved users and the reported object. This means that all data associated with these entities are available. It is for example possible to see how many friends the user who filed the abuse report has, which favorite color the reported user has and if the two users are friends.

Please recall that assumed names are used in all examples throughout this report.

3.2.3. Example of Abuse Reports

A few examples of abuse reports are listed below. The first three reports, showed in Table 1, Table 2 and Table 3, are examples of good reports. The users who have filed these reports have understood both the rules and the reporting system correctly. The reported users have in fact done what they are accused for and those actions violated the rules.

In the forth example, showed in Table 4, the reporting user has understood how the system should be used but used it incorrectly by filing a false report. Abuse report should be used to report threats but the reported user has not threatened the reporting user and the report in question is therefore a false report.

The fifth and last report, showed in Table 5, is an example of when a reporting user has misunderstood the system. The reporting user is trying to get in touch with another user by filing an abuse report, believing that these reports are sent to the respective user instead of the Customer Service.

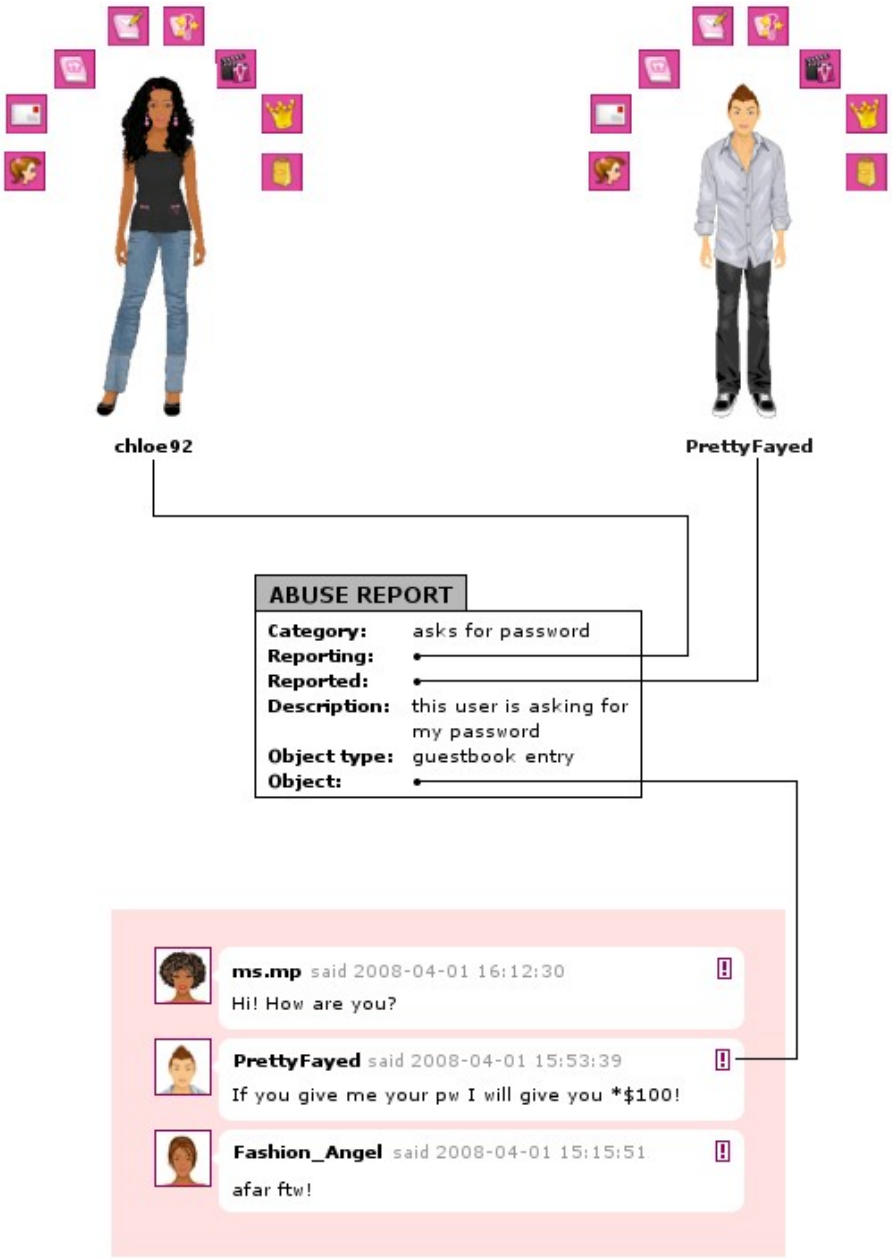

Figure 4: Overview of an abuse report.

The first three of these reports would result in warnings being filed against the reported users (i.e., yna_413, -xhot-lynnx- and Cherry_pop). The latter two on the other hand would just be deleted.

Table 1: Example of an abuse report (1/5).

Attribute	Value
Category	asks for password
Reporting user	Avril-lover
Reported user	yna_413
Description	Shes realy nice but she askd 4 my password!!!!
Object type	private message
Object	hi can i borow ya password

Table 2: Example of an abuse report (2/5).

Attribute	Value
Category	bad language
Reporting user	jammcc
Reported user	-xhot-lynnx-
Description	she is only sevan and she's using bad words!
Object type	guestbook entry
Object	kiss my azz u dumb bit2h

Table 3: Example of an abuse report (3/5).

Attribute	Value
Category	other
Reporting user	Laura1787
Reported user	Cherry_pop
Description	she told everybody to 1) copy and past this into 7 peoples album 2)press f6 3)log out 4)log in then you will aoutomatically have 25 stardollars
Object type	album comment
Object	1) copy and past this into 7 peoples album 2)press f6 3)log out 4)log in then you will aoutomatically have 25 stardollars

Table 4: Example of an abuse report (4/5).

Attribute	Value
Category	threats
Reporting user	tIkKa
Reported user	queenashley
Description	she has threatined me by saying that she was going to beat my butt
Object type	broadcast message
Object	im so bored so i need friends

Table 5: Example of an abuse report (5/5).

Attribute	Value
Category	shows email, phone or address
Reporting user	emopower
Reported user	Garcia10101
Description	do u want to be my frend
Object type	doll comment
Object	hi evry one come check out my page !!! l8r

3.3. Possible Uses of Automated Methods

Automated filtering of abuse reports is not the only way automated methods can help to deal with large amounts of data. Four methods, including the filtering option, will be presented below together with some considerations regarding them. All methods are focused on improving the handling of abuse reports since that was the aim of this work. It is worth pointing out that this is by no means the only area an automated method can help a community site. If users can buy items or services, which is true in Stardoll's case, an automated method can be used to optimize advertisement parameters for those items and services in order to maximize the conversion rate (i.e., the number of users who buys the item or service thanks to the advertisement). For websites mainly depending on high volumes of traffic, such as sites financed by advertisements, a helpful automated method might identify patterns that lead to users returning and take the appropriate actions to make sure that those patterns occur more often.

Before dealing with possible ways an automated method can help the Customer Service handle abuse reports it is worth describing the problematic parts in the current processes. As mentioned earlier all abuse reports are handled by hand on a one-by-one basis. Normally the report first written is the one first handled. There

are two major drawbacks of this system. One is that the workload on the customer service is large. Dealing with more than 20000 reports each week requires a lot of work. The other is that reports, especially extra important such, are not handled as quickly as they should. The last problem is somewhat mitigated by the separate view that lists reports based on how many other reports are filed against the same user. Even though this list captures high-profile abusive users, the average report still takes a long time, usually a few days, to handle. It is desirable to shorten this time and act faster. If a user for example asks other people for their passwords, an activity often referred to as phishing, it is important to act as soon as possible in order to prevent users from giving away sensitive account information.

3.3.1. Ordering Reports by Priority

One lead on how the process of handling abuse reports could be improved was to prioritize the reports so that serious abuses could be handled first and less serious abuses afterwards. One benefit such a system would provide is that important and severe abuses could be dealt with right away instead of laying in a queue for a few days. The implication of this is that less prioritized abuses would take longer time than today to handle, but this might be an acceptable side effect. In the long run it is likely that the amount of reports to handle would reduce since abusive users could be spotted earlier and actions could be taken against them sooner, stopping them before they could cause further concern.

Prioritizing reports as outlined causes problems that must be dealt with. One problem is that of starvation [Tan01]. There is a risk that reports with low priority never will be handled because they would never reach the top of the queue since new reports, most of which will have a higher priority, always will get filed. Solutions to this problem exist and one approach is to increase the priority of reports over time. A report that has been in the system for a certain amount of time will automatically have its priority increased. This will guarantee that even reports concerning less prioritized abuses will get handled at some point. Unfortunately this solution also reduces the win of the system. Even with this system in use, reports about less prioritized abuses might get handled before higher prioritized abuses due to this procedure.

Another, perhaps more important, problem is that reports are not assigned with priorities today. Reports are flagged as either good or bad but they are not prioritized at the same time. These two properties might seem very closely related but there is in fact an important difference. It is possible for a report to be good but not prioritized and vice versa. Consider for example a user who is reporting another user because they used a somewhat inappropriate language. These issues are important to handle, but few would probably argue that it is a pressing issue that must be dealt with right away, hence the report has a low priority. It is still possible that the report is good in the sense that the right object has been reported,

a good text has been written about the abuse and the abuse has actually taken place and it violated the rules.

Without manually assigning priorities at the time of handling the priority is hard to determine. Some basic rules can be made based on simple criteria, such as a report against someone asking for passwords should have a higher priority than the report outlined in the previous paragraph about the use of bad language. These rules based on the correlation between priority and type of abuse can be helpful but they can also be misleading. A threat for example sounds very serious and should probably be assigned the highest priority, but at the same time these reports are often false and written in ill will. The most common reason for reports like this is that two people fall out with each other and one of them wants revenge and reports the other for the worst thing they can possible think of, such as selling drugs and threatening to kill people. Another issue worth taking into consideration is that there are also variations in each group. Bad language might in its simplest form not be highly prioritized, but it might very well be if the reported user is using very offensive language in places that are easily accessible to others and often viewed.

3.3.2. Profiling Users

It is possible that a small subset of the users is responsible for the majority of the reports. This assumption seems to hold when looking in the database; only 3% of all users are ever reported. If it would be possible to find a common pattern among these users it would be possible to take actions against that type of user in time. Assume for example that a pattern is found saying that users in a certain group start abusing the system if they get bored, which they are likely to do if they do not have any friends to interact with. An appropriate action to take in such a situation might be to make it easier for this type of user to find friends. This might for example be done by making a link to friend finding functions more visible for these users.

A more straight-forward approach would be to try to find users that are likely to violate the rules and list these in a separate view in the administrative interface. The Customer Service could then browse through all these users and take appropriate actions. This would make it possible to spot violations of the rules even before any (ordinary) user saw them and filed abuse reports.

Profiling has moral implications that must be considered. It is for example possible that a system like this would be too general and target innocent people, possibly causing them problems, because of properties such as nationality, sex, age and other factors which are hard or impossible to control. It might also be the case that people are formed after other people's expectations. If a group of people are always believed to do the wrong thing and never act rightfully, it might be the case

that those people might actually start behaving that way.[5] This could mean that the amount of rule violations would increase within the targeted group.

3.3.3. Finding Specific Behavior

A wide variety of different types of abuses are found in the vast amount of reports available and automate handling of all these in a satisfying way is probably an infeasible task. It might, however, be possible to automate handling of a few common types of abuses. One such abuse is chain mail. It is quite common, yet against the rules, to send chain mails to other users. These mails often include some treat or threat encouraging the user to keep resending the mails. The user might be told that they will receive stardollars if they comply or that their pet will die if they do not. If these chain mails could be identified automatically it would be possible to automate the handling of these. One approach to the problem would be to have a separate view listing all occurrences of chain mailing on the site, regardless if that particular chain mail has been reported or not. It would in this case be possible for the Customer Service to act immediately and send a warning to the offending user hoping that further resending will be limited. Another possibility is that all filed reports about chain mails are automatically forwarded to a system that automatically could decide if a warning should be sent or not.

Identifying chain mails can be done in several ways. One approach is to have the Customer Service manually flag messages as chain mails. If a similar message surfaces at any later time the system can be confident that the message in question is a chain mail. Another approach is to study how messages spread around the site. Clear indicators of chain mails are if a user is sending the same message to multiple users and if the exact same message has been sent many times, possibly from different users. Some sort of precaution must be taken here in order to not have the system trigger on messages that are common, but still legit. It would be very unfortunate if “hi” would be considered a chain mail. A third approach is to try to analyze the content in the messages and try to interpret (part of) the semantic meaning of the message. A possible starting point for investigation would be to look at spam filters.

Chain mails are by no means the only abuse that could be automatically handled. Another possibility is to look at phishing. A simple indicator might be the existence of the word “password”. It is probably quite hard to ask someone for their password without using the word itself, but at the same time the word is rarely used in other situations. Yet another possibility is to look for actual account theft. If a user is filing a report saying that someone has stolen their account it might be interesting to see if the user is now logging in from another IP address (or

5 Several studies have shown that people perform worse if they are told that the group they belong to are by nature inferior. Women perform worse on math tests if their gender is highlighted, since women traditionally are not believed to be good at math. Negative stereotypes can, in other words, be self-fulfilling. [DH06]

ISP) than before or if usage patterns have changed. These pieces of information can then either be used to act automatically or give the Customer Service additional information when they are handling the report manually.

3.3.4. Filtering out Reports

One solution that would be beneficial from several points of view is to reduce the total amount of abuse reports. This would reduce the workload put on the Customer Service and it is also likely to reduce the waiting times. Reducing the amount of reports might seem as an infeasible problem and it is indeed not acceptable to just randomly delete reports. Instead only reports that do not contain any important information could be deleted while all other reports should be kept. This might be possible if consideration is given to the quality of reports. Recall that 61% of all reports are bad and no actions are taken due to them. When the Customer Service finds such a report they will investigate the matter in order to see that it is indeed a bad report and then set the appropriate report quality and move on to the next report. No information would be lost if this process would be automated. From the Customer Service's perspective it will appear as if the amount of reports has been reduced significantly, but without any side effects.

If the amount of abuse reports could be reduced to less than half the workload would decrease significantly which in turn would reduce the time it takes to handle reports. If all reports would be handled quicker the need to further prioritize reports would probably disappear. One question that might arise is if the workload would really decrease enough for these positive effects to be seen. It is easy to assume that the time it takes to handle a bad report is significantly less than the time it takes to handle a good report. Some actions, such as writing a warning, must be done in the latter case, which they do not in the former. This is indeed true but bad reports can sometimes take longer to investigate. Take as an example a user filing a report saying that they received a message from someone who offered to sell drugs, but the reported object does not contain such information. Since this is a very serious accusation it is reasonable for the Customer Service to look into this issue further by, for example, skimming through all messages the user received the days prior to the filing of the report. It is possible that the user writing the report did receive such a message but accidentally clicked the wrong exclamation mark when reporting it. This process can take time and in this case it would have been faster to handle a good report where the actual offense was seen already in the object actually reported.

The first paragraph in this section states that filtering out bad reports have no side effects since no action is taken due to these reports anyway. The drawback is that making a perfect filter is hard, especially if the data is noisy. Any filter of this type is likely to make two sorts of mistakes. The first mistake is that some bad reports are likely to be missed. This will reduce the reduction of workload since

the Customer Service still have to investigate these reports manually. The second mistake is probably far worse, namely the removal of good reports. It is possible that the filter will trigger on some of the good reports and remove them, which is serious since valuable information might be lost. This is an important drawback worth taking into consideration.

3.4. Potential Benefits and Moral Implications

Some of the benefits and implications of an automatic filtering system for abuse reports have already been mentioned, but it is worth summarizing them in a separate section. The potential benefits of the system are large. If the system would be perfect it would remove all bad reports, which is 61% of all reports. The time saved thanks to such filtering would be most noticeable; in average it takes 40 seconds to handle a report and given that approximately 20000 reports are written each week the time saved does make a difference. The most obvious change might be seen from the economical point of view. If less time needs to be put on this issue, time worked and hence salaries paid, can be reduced, or the staff involved in the process can focus their attention on other tasks. Except for the monetary issue the reduction of workload could result in shorter response times, which is highly desirable. Shorter response times have several benefits. It would mean that violations against the rules would be noticed faster and offending material could be removed before it has been seen by many users. This is a clear benefit since the environment would be safer if potential abuses could be investigated promptly.

This type of work has large potential benefits, but it might still be morally debatable. The users on Stardoll write reports when they see what they think is a violation of the rules, perhaps because they feel uncomfortable and even threatened. Letting a system automatically go through these reports and delete some of them is by many people perceived very differently from having humans read through all reports. It is nevertheless important to keep in mind that a manual process does not guarantee correctness. The current manual process is discussed from this perspective in Section 7.2 on page 85. Furthermore, it is also important to keep in mind that a successful filtering system could shorten response times and by that, as seen in the previous paragraph, make the site safer. The trade-off between these aspects is not trivial to handle.

In addition to the immediate effects it is wise to consider the long-time effects of implementing this type of system. Since predicting the future is hard, especially in volatile environments such as Internet communities, it is difficult to anticipate how a system like this would behave in the future, but it is possible to make some educated guesses. One prediction that is likely to hold is that the number of members on Stardoll will keep increasing. When this thesis work was initiated Stardoll had seven million members and when this is being written the number is up to seventeen million. When the number of users is increasing the number of

reports is also increasing and the more reports there are the more effect a system like this will have. Figure 5 shows how the number of reports received per week has increased when the number of users has increased. The correlation seems to be fairly linear. The most eye-catching artifact might be how the number of reports was exceptionally high when the site had 7 million users and exceptionally low when the site had 8 million users, at least in comparison with the trend line. The reason for this artifact is unknown but no major changes to the system should have been done during this period.

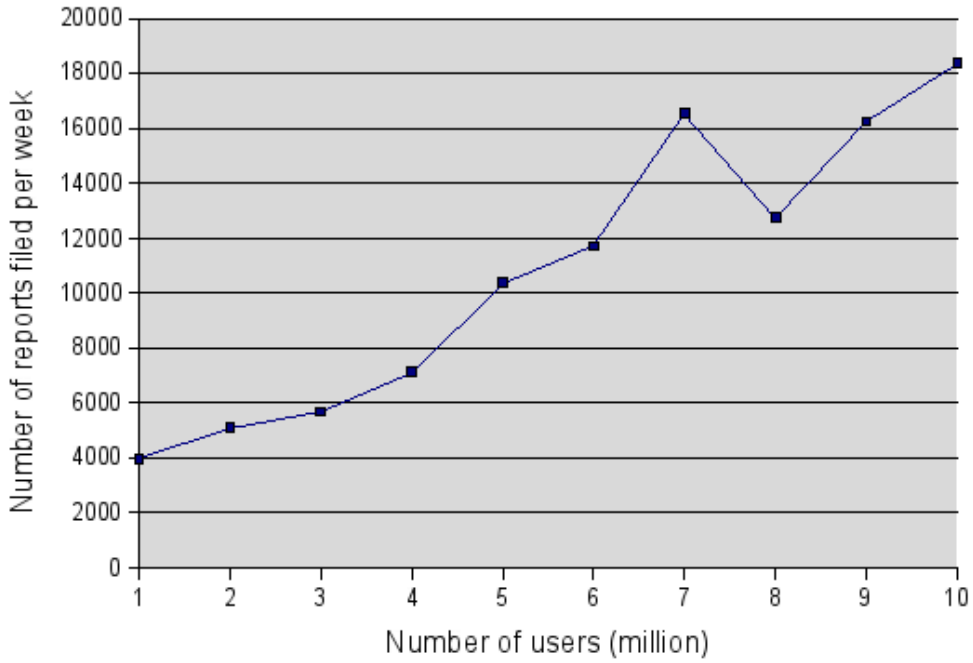

Figure 5: Diagram showing the correlation between number of reports filed and number of users.

4. Preprocessing the Data

Before the data can be fed into the machine learning algorithm it must be preprocessed. This processing consists of many steps and is a large part of the overall work. This section will first explain preprocessing in general and then discuss the steps taken during the work preformed.

4.1. Theoretical Aspects

Preprocessing data is an important and time-consuming part of any machine learning process. Mostly, the training phase is seen as the heart of a machine learning process. It is during that phase the learning algorithm is used and it is at that point the system is making the generalized model that is most likely the output the whole process aimed at producing. The problem with this view is that it does not reflect how time is usually spent in real projects. Training is usually only a small part of the total amount of work done and one large part of the rest is the preprocessing phase. [LS95,BAS+98]

The aim of the preprocessing phase is to gather the data that should be used and to make it available in a form that is usable in the learning phase. During the learning phase each example should be presented in the form of a set of attribute-value pairs, where one pair contains the class. Data is seldom available in this format originally for several reasons. It is first of all possible that the data is spread out. If the data is saved in a database it is likely that several tables are used and then data must be fetched from all relevant tables and collected in this flat format. Furthermore the data is likely to be presented in another form than the desired.

4.1.1. Preparation

Some practical details that can potentially cause concern must be dealt with during the preprocessing phase. These practicalities are not necessarily related to machine learning in any way, but they must be overcome in order to continue the work. Getting access to the right data is one thing that must be done. This can be as simple as getting a specific file or login information to a database but it can also be more complex depending on where the data is stored. If the data is currently accessed by other processes it might be required that the data is copied to another environment. Accessing such an environment or setting one up can be more complex than first imagined.

It is not necessarily the case that all data gathered can be used; hence some initial filtering might be required. It is possible that some examples in the database do not already have a class or that the class is likely to be incorrect, or at least not helpful. Reasons for having such examples may vary. One explanation for the presence of invalid data is that testing has been performed and that fragments of it are still in the database.

The data available at this point is the total amount of data available and it should be used both for training the classifier and to test its performance. It is not appropriate to use the same set of data for both these tasks due to the risk of overfitting mentioned earlier. To avoid this the total amount of data is preferably split into two subsets before any further work is done. One subset will be used to train the classifier and the other will not be looked at before a classifier has been created. The latter data, called testing data, can only be used once. Assume that several classifiers were tested on the same testing data and that the best one was selected as the final classifier. The selected classifier would in this case not be independent of the testing data any longer. The result is likely to be too optimistic. The drawback of this approach is that a classifier can not be optimized. Once its performance is known, when it has been tested on the testing data that is, it is too late to improve it. The solution to this problem is to further divide the training data. One part of this data can be put aside before a classifier is trained and that data can later be used to test the classifier. This scenario does not violate any dependency constraints. The part of the training data that is put away before the actual training is called validation data. An overview of these three datasets is seen in Figure 6.

Figure 6: Overview of data sets.

One question that might arise is how the dataset should be split. It is not possible to give an exact percentage for how big the training and testing datasets should be and the exact number may also differ between different applications. It has in many real-life situations been shown that putting away 25% for testing often give good results [WF05]. Extracting the validation set can be made similar. Validation and testing data serves the same purpose so it is reasonable to assume that the same distribution between sets should work satisfactorily.

Stratified extraction is a technique that can be used to improve the quality of the different datasets. Stratified extraction means that the distribution between

different classes is preserved even after the split. [WF05] This means that if 25% of all examples in the original dataset belonged to class A, then 25% of all examples should belong to class A in the training, validation and testing datasets too.

An example might help explaining both the different datasets and stratified extraction. Assume there are 160 pizzas available which all have been classified by domain experts as either tasty or not tasty. Assume further more that 90% of all pizzas, in this example 144 examples, taste good. Extracting 25% for testing means putting away 40 pizzas. If this is done in a stratified way 90% of the extracted pizzas, which would translate to 36 pieces, should by tasty. The remaining 120 pizzas, which should be used for testing and validation, are split in a similar way. The exact result of the splitting can be seen in Table 6.

Table 6: Example of division into different dataset and stratified extraction.

Dataset	Number of pizzas	Number of tasty pizzas	Number of not tasty pizzas
All data	160	144	16
Testing	40	36	4
All training	120	108	12
Validation	30	27	3
Actual training	90	81	9

The exact proportions might sometimes be hard to maintain, but the result should be as close as possible. It is easy to see why stratification might be important by looking at an extreme case. Recall the example above where 25% of all examples were classified as A, and assume that a testing dataset of 25% should be extracted. It is in this situation possible for all exampled belonging to class A to end up in the testing dataset and none of them in the training dataset. Since the machine learning algorithm would not see a single example belonging to class A in this case it would be hard for the algorithm to learn the characteristics of that class. The algorithm would probably conclude that no examples belong to class A since that is true for all examples it has seen.

4.1.2. Collecting Attributes

Collecting attributes is the main task during the preprocessing phase. It is important to catch all needed attributes and also present them in the most appropriate form. When this is done each example should contain a set of attribute-value pairs which, in addition to containing the correct attributes in the correct form, should be self-contained. It must be possible to take out each example and look at it independently.

Making examples self-contained is done by explicitly stating all connections, if any, that are believed to be of importance. Assume for example that an automated system should learn to recognize signs of intrusion in a computer system by looking at failed login attempts. One such login attempt is probably not enough to determine if the system is under attack or not. Even legitimate users can mistype their password or accidentally mix it up with another password. The system will likely need to know about how a particular example fits into a bigger picture. It might be helpful to specify attributes that indicates how many previous login attempts that have failed, if the specific host (if applicable) has successfully logged in before and so on. All these attributes have to be specified within each example. Making examples self-contained also means that data from several sources or tables should be put together. In the pizza example from above all information that was thought to be interesting was associated with the pizza itself. A pizza can for example be represented with the list of ingredients that were used to make the pizza. In some cases this is not enough and then focus must be shifted to metadata. One example might be if we are trying to find out if a pizza is made purely out of ingredients that are produced nearby. For this to work we need to know where the pizza was made and where all the ingredients come from. If saved in a database this information would probably be scattered around in many different tables. The pizza would be linked to a pizza parlor which in turn would be linked to a physical location. A similar chain would probably be found to trace the origin of the ingredients. In order for the machine learning algorithm to work all metadata that is thought to be useful must first be extracted and combined with the original example.

Knowing which attributes to use is hard. Using them all is not feasible since the set of all potential attributes can get enormously long even for seemingly simple examples such as the pizza example. This implies that focus must be restricted to some parts of the attribute space that appears to be extra interesting. Domain experts, which are people who are well-familiar with the domain in question and possess knowledge about it, can help determine where this attention should be focused.

Determining which attributes to use is not always enough. Sometimes the attributes are presented in a form that is not immediately understandable to the machine learning algorithm. Many such algorithms can only deal with numeric and categorical attributes, the former being numbers and the latter a set of predefined values. It is possible that some attributes can not be directly translated to either of these types. An example of such an attribute is a text attribute. Most algorithms do not have a built-in way of handling text but it is still possible to handle it by using an appropriate representation. The representation is responsible for transforming the text into a set of attributes that can be handled by the algorithm. Several representations exist and one widely used is the bag of words representation

[WF05]. Another type of attribute that can cause concern is timestamp attributes. These attributes contain much information and perhaps important patterns but it is not trivial to access it. The timestamps 1194368400, 1194956100 and 1195602300 all belong to the same day of the week, but a general purpose machine learning algorithm would not realize that. If the weekday is thought to be important this information must be extracted beforehand and saved as a separate attribute.

The attributes collected do not always contain data for all examples. It is often the case that a certain attribute is not present for a certain example and this must be handled in an appropriate way. In order to decide how to do this the reason for the missing data must be considered. The reasons can be that the data is either unknown or undisclosed.

Unknown. The value might be missing because it is simply not known. Say for example that some entries in a database have been accidentally lost. After such event no knowledge of what those entries were is available; the data is missing. This does not say anything about the examples that were affected and the missing values do not provide any information.

Undisclosed. The value might be missing because someone intentionally chose not to provide it. Take a person's birthday as an example. If this field is left blank in some database the birthday is unknown to the holder of that database, but it is not completely unknown to everybody. The field was intentionally left blank and that is actually a piece of information in itself. The subgroup of people that do not want to supply their birthday may differ from the subgroup of people that do. The important part is that this potential difference is not known and therefore should not be ignored.

The difference between these two cases might not be crystal-clear in the beginning but the subtle difference is very important. In the first case the missing values should just be kept as missing and handled appropriately by the machine learning algorithm. Exactly how this is done differ between algorithms and also between training and testing, but most algorithms have some sort of mechanism for handling this. In the latter case it is not appropriate to keep the missing values as missing since this would force the algorithm to ignore the attribute, hence loosing possibly important information. Instead these cases are handled by replacing the missing value flag with an actual, but unique, value. As an example of this method let us assume that one attribute is the country in which a certain event took place. The country can, for example, be represented by letter combinations such as “se”, “us” and “ca”. If this value was undisclosed this filed would initially contain a value representing a missing value, such as NULL. During preprocessing this flag would be exchanged with an actual value representing people who have chosen not to enter a country. Care must be taken to ensure that this value does not already exist in the data with another meaning. Consider for example the use of “no” in the

country example. This would be inappropriate and violates the rule of uniqueness since the values in question is most likely already used to represent Norway.

4.1.3. Recovery of Data

The time factor can sometimes complicate the process of collecting attributes. For the machine learning algorithm to work properly the data presented in the two phases must be comparable. This means that the data that is shown to the algorithm during training must be processed so it resembles the data that will be presented to it during actual classification. In some situations this is trivial and causes no concern. The image analysis software mentioned earlier that aimed at identifying geological features in an image is an example of one such situation. When the images were taken does not effect how attributes are collected. In another of the example presented above, the credit card fraud detection system, the time causes more concern. During training of such a system it is likely that the algorithm will be presented with historical data but when actually deployed it will be presented with live data. Assume for example that one attribute that is assumed to be of importance is the balance of the account. In this situation it is important that the examples presented to the algorithm have the balance as it was then. The current balance of the account, which is the account balance some time after the possible fraud, is not to be used since that value would not be known during live classification.

The reason data must be recovered varies from situation to situation. In general it can be said that data can be changed in three ways: added, modified or removed. The way data has been changed does not change recovery much in the general case but sometimes it can help, as we shall see in the next paragraph. It is also worth keeping the three different modifying operations in mind during the preprocessing phase so that no changes will be forgotten.

In real-life situations modifying operations can sometimes be reverted. The most common situation might be when data is added. Timestamps are often associated with all entries produced for one or another reason. Sometimes the timestamp is part of the entry itself and something that is desirable to know. This can be the reason that timestamps are associated with blog posts and news articles. In some other cases the timestamp is needed from a business perspective. It is necessary to know when bank transactions were made in order to calculate the correct interest. No matter why it is present, a timestamp can help recover data. By ignoring all entries that were added after the example in question was created interference from new entries is avoided. A similar situation can occur when it comes to removed data. It is often the case that data is not actually removed but instead flagged in such a way that it will not be viewed under normal circumstances. There are several reasons for using this method instead of simply removing data. One is traceability since it in many domains is important to be able

to look back in time. Another is to be able to undo the operation. It is possible that the removal was accidental and in that case it is desirable to be able to get the data back. There may also be other reasons, such as performance, why data is flagged as removed rather than actually removed.

Sometimes the data needed in order to collect all wanted attributes can be recovered even if no timestamps and removal flags are present. If the information is taken from a database being backed up it might be possible to use old backups in order to get information about what values certain data had in the past. The same might be true if logs are kept about changes in data. In that case it might be possible to start from the current value in the database and reverse all actions up to a desired point in time. If none of these possibilities are applicable one last possibility is to gather live data instead of historical data. In some situation it is possible to set up a system that records all needed attributes and then afterwards assigns the appropriate class. Consider again the credit card fraud detection system. An agent could be put in place that records all attributes, including the current balance, when transactions are made. Once a fraud has taken place the transactions that were carried out as a part of the fraud are marked. This procedure will guarantee that data during training will look the same way as data during classification, simply because it is gathered under similar circumstances.

Unfortunately it is not always possible to recover data completely with any of the above described methods. It is possible that data sometimes are lost and in that case it is not possible to gather attributes perfectly. There are several methods for handling this and which to choose depends on the situation. Possibilities include ignoring either the attribute or the possible change. Ignoring the attribute means that attempts to recover the attribute will not be made and that the attribute will not be used in the machine learning process. This might be feasible if it is not possible to recover the attribute and if the attribute is believed to be changed frequently and rather randomly. A person's mood might be one such example. If a person is happy or sad today does not say much about how a person felt a specific day five years ago. At the opposite end of the spectrum it is possible to ignore the change instead. Doing so can be reasonable with attributes that are seldom changed. The country a person lives in is not changed very often and it can be acceptable to assume that the present country of residence is the same as it was six month ago for most people.

There are other possibilities too but they mostly depend on the individual attributes. In some situation it might be possible to acquire the exact value or estimate it in another way. Other sources that apply to the domain can be helpful. Government statistics can for example help determine certain patterns that can be applied. It might not be possible to know exactly how much savings a certain person had at a certain time but it might be possible to estimate it using knowledge

about the current value, changes that have taken place in the person's life and data on how taxes and rates have changed.

Related data can also be difficult to recover. The problem is basically handled in the same way but since it occurs in a slightly different way it is worth stressing it. The fact that data is related does not always appear at a first glance but it is important to avoid relying on data due to dependencies that will not be available during classification. At the same time it is important to capture as much data as possible in order to build a system with good performance. General guidelines for how to handle this trade-off are hard to find but it is important to think carefully about this issue during preprocessing. As an example of when the problem is very visible, consider a system that aims to determine where bank robberies are likely to take place. One might think that the locations of police units are important since robbers perhaps choose banks far away from those units, in other words in areas with little police presence. The problem with this is that if the timing is slightly off it is likely that the system will come to the conclusion that bank robberies are taking place in areas with an extreme high police density. The reason for this is that police units will be called to the robbed bank when the robbery is known; hence the density will be high afterwards. This correlation can not be used in a system that should anticipate which banks will be robbed in the future. A further complicating factor is that it is not always easy to know if the timing is off. Clearly no one who aimed to determine the locations of police units at a certain time would deliberately add half an hour to the time. It is, however, possible that the data, in this case the location of police units, is only collected at certain intervals.

4.2. Preparation

The first step taken in the preprocessing phase was to take a snapshot of the live database and save it in a local database. There were several reasons for doing so, one of which was the load. Preprocessing the data, as well as doing the actual processing, was very time-consuming and processor intense, which made it inappropriate to do on a live database. Even worse was that the live database was already having problem coping with the normal load. Another reason for using a local copy instead of the live database was that a local copy will remain in its current state while the live database will be constantly updated. In the live database, a report that is not classified today can be so tomorrow.

The local database was an exact copy of the live database on all but two points. Some sensitive information, namely email addresses and passwords, were removed in order to protect the users' privacy. Neither of these fields was believed to be important for the task at hand so this deviation was ignorable. More important was that some of the tables in the live database were not copied. This issue will be described in more details in Section 4.7 on page 45.

Before any other processing could take place some reports had to be removed, some of which were disregarded entirely and some that were removed from the working set but safely kept in a different place. The reports that were not further considered at all were the ones that at the time of creating the database copy were not yet handled. These reports did not have any class assigned to them and could therefore not be helpful during either training or testing. There was also a category of handled reports that had the class "INDIFFERENT" assigned to them. This class does not provide any information and reports belonging to it are not useful to keep. This special classification can be assigned to reports that for one or another reason are not relevant. When a misbehaving user is deleted all reports related to that user can be safely removed, and those reports could be assigned the indifferent status. After these non useful reports were filtered out the database contained 633784 reports classified as either good or bad.[6] Before any other manipulation of the data took place a subset of the data was put away for testing. Recall the importance of having separate datasets for training and testing in order to avoid overfitted and biased results. The size of the testing set was chosen as 25% of the total amount of data.

4.3. Attributes in the Report

The reports in themselves do not contain much information. Table 7 shows all attributes available in an abuse report before it is linked together with other information. The reports have the id numbers for the reported as well as the reporting user, the category selected by the reporting user and a description of the potential violation also written by the reporting user. The time the report was filed, from which top domain it was filed and what language it was written in are also kept in the report. In order to keep track of the reports each one of them is assigned a unique id number. The report also contains the type of the reported object. The actual links to the specific objects are contained in separate tables, one for each object type. This is not much information to use in the machine learning process. The id and timestamps are not useful attributes; instead they can actually harm the process. The problem with id attributes has been discussed when talking about overfitting. Timestamps are suffering the same problem since it is unlikely that two reports are getting the exact same timestamp. Unlike the id attribute, however, the timestamp attribute actually contains real information that could be important. Someone might have a hypothesis that reports written on Tuesdays always turn out to be good. In reality this was not the case and the timestamp attribute was also ignored.

6 In reality reports could also be classified as nonsense but these were considered as bad reports.

Table 7: Attributes available in abuse reports.

Attribute	Description
Id	A unique id number.
Reporting user	The id number of the user filing the report.
Reported user	The id number of the user the report regards.
Object type	The type of the object that is reported (e.g., guestbook entry).
Timestamp	A timestamp for when the report was filed.
Category	The category the reporting user chose (e.g., threats).
Description	The free-text description the reporting user wrote.
Top domain	The top domain the report was filed from (e.g., com).
Language	The language the report is assumed to be written in. This attribute is used to redirect the report to a Customer Service Representative that speaks the language the report is written in.

The fields containing the id numbers for the involved users are not so important in themselves. That a user has id 7798126 does not say much about the expected behavior of that user, or how that user relates to other users. Luckily these id numbers can be linked to users and all information available about them. Very much information can be extracted this way. A few of them are the age of the user, how long since the user registered at Stardoll and the user's favorite food. In some situation this is fairly straight-forward, but it can be more complex. One such example is when it comes to guestbook entries. Each user has written and received a number of guestbook entries and one suggestion is to simply associate the number of written and received guestbook entries for the user with the report. Doing so ignores the content of the entries completely which is unfortunate. Another suggestion is to include the length of the entries. Yet another suggestion is to also analyze the messages and indicate if they contain bad language. Selecting which attributes to use will be discussed in the following section.

When it comes to the description filed the exact content of it was not believed to be of relevance. The reason for this assumption was that the field is used in several different ways and sometimes not at all. It was decided that it was unlikely that using an advanced representation would be worth the time invested. Instead a simpler representation was used. A simple tool for extracting a few metrics about the text was used. Table 8 lists all metrics that were outputted by the text analyzing tool. The table also contains the result of running the tool on the example description “This person started calling me names when I refused to send a gift” found in the abuse report showed in Table 9.

Table 8: Attributes produces by text analyzing tool.

Name	Description	Example result
Length	Length of the comment.	66
Punctuation	Number of periods, question marks and exclamation marks.	0
Words	Number of words.	13
Long words	Number of words longer than six characters.	3
Sentences	Number of sentences.	1
Lowercase	True if the comment contains a lowercase letter, false otherwise.	true
Uppercase	True if the comment contains an uppercase letter, false otherwise.	true
Both cases	True if both the above properties are true, false otherwise.	true
Lix	Readability calculated by Lix.	36

Lix is a way of calculating how easily read a text is. It was originally developed for Swedish, but it has been tested on English with successful result. The lower the number, the easier the text is. [And81] This value can therefore be used to get an approximate value for how advanced the description is; whether it is written as a children's book or a technical report. However, it is worth noting that Lix is usually applied to longer and more properly written texts, and not to short comments like this. The nonsense description “asdfasdfasdfasdfasdf” has a Lix value 101, which would indicate that it is a very advance, technical and bureaucratic text.

Table 9: Example of an abuse report.

Attribute	Value
Category	bad language
Reporting user	annannanna
Reported user	Digbu7
Description	This person started calling me names when I refused to send a gift
Object type	private message
Object	your a fugly whore

4.4. Attributes Suggested by the Customer Service

Using all attributes is seldom feasible since the list of attributes tend to be enormously long. This implies that a real application must use a subset of all

attributes and that was true in Stardoll's case too. In order to focus the attention on the most promising attributes from the start, domain experts were consulted. Those were people working in the domain in question, thus likely having knowledge about what affects the result. It is likely that a person working with classifying reports, which would be a Customer Service Representative, has noticed a certain connection and if that is the case it is wise to pay some extra attention to the attributes concerned.

The Customer Service came up with four attributes which they thought deserved extra attention. The first of these was the monetary aspect. A user with a large economical investment has more to loose if being deleted from the site so it is possible that these users are more careful about how they behave. It is also possible to look at it from a wider perspective. The users investing time and money in Stardoll are the users who like the site the most and care about what happens. These users are not interesting in harming the community since this is something they like. The second attribute was if a user has been a cover girl lately. Each day one of Stardoll's users is rewarded with the cover girl title. This user will, in addition to getting an amount of stardollars, be featured on the front page of Stardoll's virtual magazine The Show. There is much prestige in winning this title and users are fighting hard over it. Unfortunately, there is also a great deal of jealousy involved. It is not uncommon that the winning user is accused of being an unworthy winner and that other users feel that they earned the title more. It is likely that these users abuse the reporting system and file reports against the actual cover girl because of jealousy. The third attribute was if a user is kidlocked. This is a lock that automatically gets applied to all users who are below the age of thirteen when they sign up for an account. Users from the USA need, according to federal law [Uni98], their parents' consent for removing the lock. Users from other countries can unlock the lock themselves. When this lock is in place the users are not allowed access to all features on the site and they can therefore not produce as much reportable content as regular users. Because of this it can be assumed that the kidlock is an important attribute to consider. The forth and final suggestion was to consider if a user is a RealCeleb or not. RealCelebs are real celebrities who are specially invited to the site in order to interact with their fans. These users seldom browse the site like regular users. Instead they are interacting with other users through special channels, such as celebrity chats. It is rather unlikely that a RealCeleb would write a questionable guestbook entry for example. It has even been suggested that it should not be possible to report RealCelebs in the first place. It is worth noticing that all these four attributes pointed out by the Customer Service have focused on the reported user rather than the report itself (e.g., the category selected) or the reporting user (e.g., if the user filing the report has filed reports before).

4.5. Attributes Used

The four attributes mentioned by the Customer Service were just a small fraction of all the attributes that were used in the process. When all data had been encapsulated in the reports each report had over 200 attributes. These attributes were divided into four different categories, which will be explained in turn below.

Meta. Some metadata was still associated with each report, mainly for traceability. The meta attributes were not used in the actual machine learning algorithm. Typical attributes in this category were an id number for the report and a timestamp for when it was created.

Defendant. The amount of information that could be associated with each user was huge, as described earlier. The defendant category contained all attributes that were associated with the reported user (the defendant). A typical attribute in this category was the number of days the reported user had been on Stardoll.

Plaintiff. The plaintiff category contained exactly the same attributes as the defendant category, but the considered user here was the user who wrote the report (the plaintiff).

Report. Some attributes were related to the report itself rather than one of the users. One example is the number of reports written in the reverse direction, meaning the number of reports where the current plaintiff was the defendant and the current defendant was the plaintiff.

All attributes are not specified in detail in this report since the list would be long and it would be difficult to explain all attributes without going into details about how Stardoll works and how the database is organized. The list would also expose sensitive data. It is at the same time hard to specify, at a general level, exactly how these attributes were chosen but some things can be said about the creation of the list. All tables in the database were looked at in order to get an idea of the type of information saved there. If each user only had one row, information was extracted from that row. If instead each user had more than one row, the number of rows was used. A user's favorite food, which can only be one, was stored as one attribute. A user's favorite dolls on the other hand were saved only as the number of such dolls.

Exceptions were made to the above description when information was not believed to be fully captured. This was a subjective process and the information provided by the domain experts in the previous section was taken into consideration. The economical investment involved had, for example, been thought to be of importance; hence several attributes looked at this aspect. The most straight-forward approach was to simply save the amount of stardollars a user had as one attribute, but this does not capture all aspects of this matter so three additional attributes were used. The first of them was the total number of stardollars invested, the second was the total number of stardollars bought and the third was the number of stardollar purchases. All these were capturing slightly

different aspects of a user's behavior. The total investment is basically the value of an account as it specifies how many stardollars has been spent on making it the way it is. A user can either get stardollars by being an active user on the site and winning different contests, or by purchasing. The second attribute specified how much “real” money has been invested. The third attribute specified how many times stardollars has been bought for “real” money.

Another exception to the simple rules outlined above was the number of album comments. According to the rules above one attribute, the number of album comments, would be associated with the example. However, album comments can either be sent or received which gives two attributes: one for the number of sent comments and one for the number of received comments. Another aspect not captured by this simple representation is the number of unique users who have received the sent comments. It may be important whether a user is talking to many or to few other users.

4.6. Recovery of Data

In order to create the full list of attributes a lot of information needed to be collected about the involved users. In some cases this was trivial, but in most cases time complicated the process. The attributes calculated should match the value they had when the report was written rather than what they were when this work was done. When it comes to the amount of money for example the attribute used in the examples should be the balance the user had when the report was written. This can, and is likely to, be different from the user's current balance since the user might have refilled the balance, earned money or bought items. In this particular case the reconstruction is simple since all transactions are properly logged with a timestamp attached to it. These logs made it easy to know exactly which transactions had been performed at any given point in time. It was, in other words, possible to know how much funds a user had when a certain abuse report was written.

All changes to the database are not done as described above. Sometimes old information is simply overwritten when a new value is available. Unfortunately the old information gets lost and makes it impossible to make a perfect reconstruction, but two remedies were used to cope with this problem. The first and simplest method was to ignore it. This approach was for example used when it came to the quick facts, which are short and simple questions with a number of predefined answers that users fill in about themselves. The user answers these questions and the answers will be showed on the user's profile page. Possible questions include favorite food and favorite celebrity. These facts are not believed to be changed too often and it was reasonable to ignore possible changes.

The second approach used for handling attributes that could not be perfectly reconstructed was to try to interpolate [GWF02] an old value given the new value.

This was for example used with the starpoint rewarding system. The amount of starpoints a user has is only stored as a number. When a user is being rewarded with new points due to recent activity on Stardoll these new points are added to the previous number without any logging. In this case linear interpolation was used to estimate how many points a user had a certain day since the actual value can no be recreated. Assume for example that a user has 100 starpoints twenty days after registration. If these points were received at the same pace this will mean that the user was rewarded five points a day. It is therefore assumed that the user had 50 starpoints ten days after registration and 75 starpoints after fifteen days, and so on. This assumes that starpoints are rewarded at the same pace throughout time, which is not a realistic assumption. How active a user is on the site, hence how many starpoints that user will receive, is likely to change over time. It might be possible to get a better estimate than this by studying usage patterns, but this was a complicated process outside the scope of this work. Linear interpolation should be accurate enough.

The above paragraphs describe updates and how they cause problems. Another operation that also complicates reconstruction is when data is deleted entirely. Luckily such operations are quite seldom used. Most of the time when it appears to the user that content is deleted, for example when the user deletes a guestbook entry, the actual content is still left in the database but it is flagged as deleted and not shown to regular users. The Customer Service can still view this content which can be important when following up on a potential abuse. Unfortunately some content is not handled this way and is actually deleted. This is for example true for the high score lists on the games available on Stardoll. If a user makes it to the high score some other user is deleted from it.

A problem closely related to the one about reconstructing the data is reconstructing related data. One attribute which might be important is how many other abuse reports that have been filed against the same user. When looking at it in retrospect it is easy to know how many similar reports that were written, but it is not as trivial to know this when the classification would have been done. The most straight-forward approach is to simply count how many similar reports with a smaller timestamp are present. This will, however, not exactly emulate what will happen when a report is to be classified. The classification will, as mentioned earlier, not be made when the user is filing the report. Instead it will be run as a batch job at regular intervals, probably once a day. This means that there might be a delay between the filing of the report and the classification of it. In many cases, such as how many starpoints the user has, this is not so important. A user is not likely to change their behavior dramatically during just a few hours. However, when it comes to some attributes, most importantly the number of similar reports filed, this amount of time might be important. If the abuse is severe many reports are likely to be written in a small amount of time. As an example, let us assume

that five reports are being filed shortly after each other. During classification these reports will be handled during the same nightly batch and thus all have four as the number of similar reports filed. It is desirable that this is also the case during training. If the simplest approach described above would be used, the number of similar reports would be 0, 1, 2, 3 and 4 respectively. One way of simulating a more realistic amount for this value is to check how many reports that have been filed before, say, one day after the actual report is filed.

Another attribute might be based on the quality of these related reports, but relying on the status of reports written close in time to the report in question is dangerous. These are all known now, but they would have been unknown when a classification would have been made. There is a balance here between using as much historical data as possible, and avoiding using data that would not be available during actual classifications. One possible solution to the problem is to avoid looking at the status of reports written after, say, one week before the report in question was written. An exact time interval is hard to determine since this may vary over time, but it should be the approximate waiting time for a report or slightly longer.

4.7. Lack of Data

As shown above the preprocessing of the examples was a difficult and subjective task. A further complication was that all data has not been available during this process. Some of the tables in Stardoll's database are split into different shards[7]. This data has not been available in the local database copy created for this task, thus no attributes related to this data has been used. Unfortunately much information is sharded making a substantial amount of data unavailable. Generally, it can be said that the data available was metadata (e.g., the age of the user who wrote the guestbook entry, the amount of Stardollars the user who wrote the report has, etc) but not actual object (e.g., the guestbook entry in question).

The lack of reported objects was very noticeable. In the list above specifying the four categories of attribute there was no category focusing on the actual object that has been reported. One can easily see that this is a major drawback since the reported object is exactly what the Customer Service Representative would look at. The answer to the question whether a report is good or bad is the same as the answer to the question if the reported object is really violating the rules, and the answer to that is in the object itself. If a guestbook entry is said to contain bad language, the answer to whether it does or not is in that very guestbook entry.

Another problem, except for the lack of reported objects, was that a lot of useful data about users might exist in this missing data. One example is that the

7 Sharding is a way of splitting a large amount of data into smaller independent subsets. This makes is possible to store these different subsets, called shards, on different database servers. This can in turn improve performance. Sharding is also called database partitioning. [SKS05]

user associations are sharded, hence unavailable. From this data attributes such as the number of friends a user has and the number of times a user has been blocked can be extracted. More complicated patterns can also be extracted from the user association data. It might for example be helpful to consider the number of distinct friend networks that have reported a user. Independent users reporting the same user might suggest that the potential violation actually took place. This might of course be the case even if the reports are filed from within the same friend network but it is not uncommon that a user who is upset with another user tells all their friends to report that user too. They believe that a user will be deleted if many reports are filed against that specific user.

By not being able to look at the sharded data some measurements on how active a user is have not been calculated. It is, as stated above, a problem not having access to for example guestbook entries since this prevents examination of the reported object in itself. However, guestbook entries in general might also give away information about a user. The amount of guestbook entries sent, the amount of entries saved, the amount of unique guestbooks written in, and so on might all represent different parts of a user's behavior.

5. Training the Classifier

When the data is preprocessed and ready to be used the next step is to train the classifier and create a model. Before this can be done the type of classification algorithm to use, together with its parameters, must be decided. This section will investigate these issues in order to try to find the best algorithm to use for the task at hand.

5.1. Theoretical Aspects

Training a classifier is, once the preprocessing is done, in its simplest form very easy. A computer system implementing a certain machine learning algorithm will do the actual training once the data has been fed to it. The aspect that complicates things is that different algorithms perform differently. In order to get a classifier that is as accurate as possible several different algorithms have to be tested. Another complicating factor is that many of the algorithms have different parameters that can be tweaked. Comparing all possible algorithms with all possible parameter values are seldom feasible so manual intervention is needed in order to focus the search in the right direction. It is also possible that factors other than accuracy affect the decision. Different algorithms can behave differently when it comes to run time, memory consumption and the type of the produced model. The model created by some algorithms are easily read and understood by humans, while some are not.

The process is described here as something linear. The data is first preprocessed, then a classifier is trained and finally the classifier is deployed and used. In many situations this is not sufficient. It is possible that it during the training phase becomes apparent that preprocessing should have been done in another way. In that case it is necessary to go back to the previous step and redo some of the work preformed there.

It is important to be able to determine a classifier's performance in order to compare different classifiers and decide which is better. This might seem as a trivial task, but in real applications it is sometimes harder than expected. The first approach is to simply measure accuracy. To do this some data is put away before the classifier is trained. The data put aside, the validation data, is also labeled so it is possible to know the correct class for the examples in it. Once the classifier is

trained it is run on the validation data. After the run each of the examples in the validation data will have two classes, one actual class and one estimated class. By comparing these it is possible to see how big part of the examples that were classified correctly. This value, which will be between 0 (i.e., 0%) and 1 (i.e., 100%) is referred to as a classifier's estimated accuracy. The problem with this value is that it is sometimes misleading since it disregards the different types of mistakes a classifier can do. A classifier that has n classes, denoted $c_1...c_n$, to deal with can produce $n \bullet n$ possible results. The actual class, a, is one of the n classes available and the estimated class, e, is also one of them. A correct classification has been done if $a=e$, that is if the actual class is the same as the estimated class. The number of such possible correct classifications is n, hence the number of incorrect possibilities is $n(n-1)$. It is naturally the case that a correct classification is better than an incorrect classification, but it is not always the case that all incorrect classifications are equally bad. Given two classes, c_i and c_j where $i \neq j$, it is possible that incorrectly classifying an instance of c_i as c_j is worse than incorrectly classifying an instance of c_j as c_i. This difference is not honored by the simple accuracy metric and it can be valuable to also have another metrics. A concrete example of an application where the difference between mistakes is noticeable is spam filters for email. The two possible classes are "HAM", for legitimate messages, and "SPAM". The system can make two correct classifications and two incorrect classifications. The former are keeping ham and removing spam. The two incorrect actions a spam filter can take are to either remove ham or keep spam. Most people would probably argue that removing ham is much worse than keeping spam. A kept spam can simply be removed manually while a removed ham means loosing information that can be valuable.

The discovery that different errors have different effect can be useful in more areas than just when it comes to measuring performance. It might be the case that the classifier should be so that the total number of errors is increased if that makes a certain type of error less likely. Some algorithms, including the decision tree algorithm, can output a confidence level in addition to the actual class. The resulting decision tree would give not just the class (e.g., "SPAM"), but instead a class distribution (e.g., "60% SPAM, 40% HAM"). The interpretation of this should be that the tree is 60% confident that the class is in fact SPAM. If different types of errors were equally bad the natural thing to do would be to always use the class that got the highest percentage. These errors are not equally severe and it is reasonable to use another approach. The trick used here is to classify a report as class c if the probability that it does belong to that class is equal to or above p_c. This can be a bit difficult when dealing with more than two classes, but it is rather straight-forward in this particular case. In this case one class can be selected as the preferred class and all examples that are assigned a probability larger than or equal to p will be assigned that class. By assigning p the value 0.5 (i.e., 50%) the

standard behavior is mimicked.[8] By using a value lower than 0.5 that particular class will be favored. What value to use for p is sometimes difficult to determine. Domain experts or potential users of the system might have to tune the parameters so that the system produces the best possible trade-off. It is possible that this parameter is changed during a classifier's lifetime. It is for example possible that more faith is put in the classifier once it has been in place for a while and in that case it is reasonable to adjust the parameter.

5.1.1. Large Data Volumes

One problem that can cause concern in a machine learning process is that the available data is very large and difficult to handle. It is desirable to have much data since it is what the algorithm needs and leaving out data can potentially degrade the performance of the classifier. A certain pattern can not be found if the attributes that make up the pattern are not present. Too much data on the other hand does not pose any problem for most algorithms. The problems that do occur are instead seen on a more practical level. Handling large amounts of data is, in general, either time-consuming or resource-intensive.

It is possible that either time or resources limits the amount of data that can be processed. The solution that can be used then is to reduce the amount of data in one of two ways. The first way of reducing the amount of data that needs to be processes is to decrease the number of examples used. The other way is to reduce the size of each example by reducing the number of attributes per example. These two methods will be described below.

Reducing the number of examples is a simple method. The idea is to reduce the number of examples just so much that the time and resources needed reach an acceptable level. Not much consideration must be taken when employing it. The most important factor is that the data should be split in a stratified way so that the subset of data used has a similar class distribution to the total amount of data. A problem one can face when using this method is to know which subset to use. Say for example that the algorithm should be fed with 25% of the available data. If the data set is split in four there are four possible subsets to use. It is, furthermore, possible to do the split in numerous different ways. One way to mitigate this is to use a few different subsets and study the performance. If the performance is similar across the tested subsets it is most likely not worth spending too much time investigating different possibilities to create the subsets.

Reducing the number of attributes is a way of reducing the size of the examples. The reduction of attributes must be performed in such a way that a minimal amount of useful information is removed. A different way of putting it is

8 There is one difference between using the standard behavior and using the method described above with p set to 0.5 and that is how example with exactly 50% probability in both classes are handled. In the former case the behavior is undefined but in the latter it is not.

that the remaining attributes should be the most influential ones. One way of achieving this is to create a decision tree and look at the most important attributes in that tree. Those attributes could then be removed from the dataset and a new tree can be created. By iterating this process a few times it is possible to extract the most important and influential attributes from the dataset, and doing what is needed in order to reduce the number of attributes without loosing too much information. The most important attributes in a tree are found by looking at the attributes near the root of the tree. The root attribute is the most influential attribute since it was chosen first and it is used on all examples to decide which path to follow. The next level of attributes is also important since they were the second choice in their respective branch, but they are not quite as important as the root attribute. The attributes become less influential the further down in the tree they appear.

There is a trade-off between the number of attributes and the number of examples. Reducing one of them too aggressively is likely to harm the process more than it helps it. The machine learning algorithm must both have enough different examples to look at to be able to spot similarities and have enough attributes so that the similarities are actually in the data. It is not obvious how a balance should be found between these.

5.1.2. Ensemble Methods

A single classifier can in many cases produce an acceptable performance but in some cases it does not. One method for improving the result is to train multiple classifiers and let them work together. The idea is that different classifiers might be good at representing different aspects of the data and a group of them might as a whole be able to outperform each of them separately. This is not too different from when a group of human experts are consulted rather than just a single one of them. Methods that combine different classifiers in this way are referred to as ensemble, or committee, methods. [Oza06]

Perhaps the simplest method for combining different classifiers is to train a few of them, say 10, and then let each of them classify new instances independently. The final outcome will be the most common output for the classifiers. It would be like the classifiers voted on the outcome and the majority decides. One example might, for example, get classified as class A by seven classifiers and class B by the remaining three. The final verdict for this example would then be that it belongs to class A. Worth pointing out is that the exact same dataset can not be used for training all classifiers since this would generate the same classifier.

Bagging and boosting, which will be explained below, are two common ensemble methods.

Bagging. Bagging, or bootstrap aggregating, is based on the basics outline above. A few different classifiers will be created during training and a simple

majority vote will be conducted among them during classification. The key to the bagging algorithm is how the different datasets are used to create the different classifiers are assembled. It would not, as stated above, be fruitful to use the same set of data to create all classifiers since this would yield an ensemble where all members would be identical. Bagging solves this by creating different datasets with the help of sampling with replacement. This means that examples will be selected randomly from the original dataset when forming a training dataset. Since replacement is used the same element can be selected multiple times and some element may not be chosen at all. This makes it possible to use one dataset to create several different datasets. [Bre94]

Boosting. Boosting takes advantage of the fact that different classifiers can complement each other. One classifier might be very good at handling a certain portion of the data but not the rest, hence it is sensible to complement it with a classifier that can handle the rest, rather than just another one that can classify the same portion. Boosting uses iterations in order to accomplish this. A classifier is created in the first iteration. The examples that are incorrectly classified by this classifier are given extra weight in the second iteration. This process is continued for a fixed number of iterations, at the end of each weights are modified to favor examples that have been misclassified. [WF05]

Both bagging and boosting are so-called meta classifiers, meaning that neither of them are providing any actual classification logic. This means that bagging and boosting requires another algorithm to do the actual classification, which it takes as an input parameter. The type of algorithm used does not matter much. All types of machine learning algorithms that can output a class can be used.

Bagging and boosting are not the only available options for having multiple classifiers working together. Recall that the different classifiers all voted and the majority vote won. This is not the only way to utilize these classifiers and here, as well as in the example with a group of human experts, different protocols can be used. In some situations the decisions are so important that a consensus is needed in order to make the decision in question. A variant of this is that a decision in one direction is more fatal or unfortunate than a decision in the other direction. In situations like this it might be appropriate to only make such decision if all voters agree. Put differently this means that a positive decision will not be made if one expert, or in this case classifier, is against it. This means that all classifiers are given the right to veto the decision in question. Several other variants are also possible, including assigning weights to the different classifiers. [TKS06]

5.2. Weka

Weka[9] is a free, open source tool that can be used for a variety of machine learning and data mining tasks. Weka was therefore a good tool to use in this project. Weka

9 Found at <http://www.cs.waikato.ac.nz/ml/weka/>.

Explorer, which is part of the tool, is a program that lets the user load data from a file, an URL or a database, and then builds a model based on that data. Once the data is loaded some preprocessing can be done, but since the data at hand had already been preprocessed there was no need to use Weka's functionality for that. After this an algorithm can be chosen from the long list of algorithms available. Weka makes it easy to try different algorithms and compare the results. In addition to selecting the algorithm Weka also makes it possible to change parameters to those algorithms. This makes it easy to not only compare algorithms but also different parameter settings for those algorithms.

The version of Weka used here was 3.4.11 which was downloaded 2007-09-18.

5.3. Choosing Classification Algorithm

Classification algorithms can take many forms and there is a wide range of them available. Weka, for example, comes with almost 50 algorithms ready to be used. Analyzing and comparing all those algorithms is a demanding task beyond the scope of this thesis work. Another issue complicating this further is that most of these algorithms can be fine tuned by changing certain parameters that affect the process. A comparison was nonetheless needed in order to find a good algorithm to work with. The approach used was to let a few selected algorithms work with their default parameters. The algorithm that had the best result under these circumstances was fine tuned in order to find the parameters to use.

Five different algorithms where used. *ZeroR* is an algorithm that outputs a model that always returns the same class. The class selected is the majority class, since this will be correct in most cases. This algorithm is not of any real use. In Stardoll's case it would simply remove all reports. The reason for having it in this list at all was because it is good to compare more sophisticated algorithms with it, since it in some sense represents the most unsophisticated way to act. *NaiveBayes* and *DecisionTable*[10] have informative names and implement Naïve Bayes and Decision Table respectively. These algorithms have not been described earlier and it is outside the scope of this thesis to do so. *J48*[11] is one implementation of a decision tree algorithm, namely the C4.5 algorithm. *Ridor*[12] uses rules and exceptions to find an appropriate model.

Since both training and testing are resource-intense tasks, requiring both time and memory, the different algorithms could not be given the entire amount of data. Instead 5% were randomly chosen, in a stratified way, among the entire set of test data available. Out of these, 80% were used for training the classifier while the remaining 20% were used to validate it. The same sets were used for all algorithms. The result of the experiment can be found in Table 10. Instead of

10 Default parameters are "-X 1 -S 5".
11 Default parameters are "-C 0.25 -M 2".
12 Default parameters are "-F 3 -S 1 -N 2.0".

showing the exact number of reports classified either way (i.e., a confusion matrix) four more general measures are shown. These measures will be explained below.

Table 10: Comparison of classification algorithms.

Algorithm	Accuracy	Percentage bad reports removed	Percentage good reports removed	Score
ZeroR	62%	100%	100%	1.13
NaiveBayes	65%	86%	68%	0.86
DecisionTable	68%	87%	62%	0.78
J48	67%	83%	59%	0.77
Ridor	66%	97%	86%	0.99

The first measure is *Accuracy* which simply shows the percentage correct classifications. It was shown earlier that this metric has drawbacks and an example can help show this once again. Consider for example a classifier that removes all reports without considering any attribute at all, such as the ZeroR classifier does. This classifier classifies 61% of all reports correctly since this is how many reports that are bad. Now consider instead a classifier that removes one third of all bad reports without removing one single good report. This classifier would correctly classify all good reports and one third of the bad reports, which would translate into 59% correctly classified reports.[13] If only the accuracy is considered the former of these two classifiers would look slightly better than the second, but in reality the latter would have an enormous value while the latter would be practically worthless.

In order to give a better view of how classifiers perform the two errors are separated and the two rows *Percentage bad reports removed* and *Percentage good reports removed* are added. The former indicates how much good and helpful work the classifier does by removed bad reports. The desired number here is 100%. The latter on the other hand indicated how much problem the algorithm causes by removing good reports. A perfect classifier would not make any mistakes and the desired number is 0%. These two numbers give a good picture about how the classifier performs. The downside of using these metrics is that it is hard to compare two classifiers if two metrics have to be considered at once. Say for example that one classifier has a higher value on both these metrics than another classifier. On one hand this would be preferable since more bad reports are removed, but on the other hand it would be undesirable since less good reports are removed. One way of making this easier is to use the metric *Score* which tries to incorporate all this in one single number making comparisons easier. This metric is not an established method for doing evaluations of this type, but it should give a

13 The bad reports consist of 61% of all reports so one third of them would mean approximately 20%. The good reports consist of 39%. If these two are added together the accuracy in this case equals 59%.

hint about how the algorithms work. It should be good enough to give a starting point but all decisions should consider the two more elaborate metrics too. *Score* builds on the function for calculating the error rate for a classifier but it weights errors differently. The error rate is simply calculated by dividing the number of errors by the number of attempts (called *n*), in this case examples. The number of errors is in turn calculated by adding together the number of bad reports being classified as good (called e_{bg}) and good reports being classified as bad (called e_{gb}). The extension here adds a weight (called *w*) to the latter of these. The aim of the weight is to make errors of that type worse (or better) than errors of the other type. The exact difference between them can be tuned by increasing or decreasing this number. It virtually says how much more an error of one type is compared to an error of the other type. The exact form of the algorithm is shown in Equation 1.

One issue that still remains is what value to choose for *w*. The Customer Service was consulted in order to get an estimate but no exact value could be given. It appears that a reasonable value is somewhere around three and there is no need to make an elaborate study in order to find a perfect value for this. The most important thing is that it is approximately right and that the other metrics are consulted before decisions are made.

$$\frac{e_{bg} + e_{gb} w}{n}$$

Equation 1: Formula for calculating the score metrics. e_{bg} *denotes the number of bad reports erroneous classified as good,* e_{gb} *the number of good reports erroneous classified as bad,* w *a custom weight and* n *the total number of classifications.*

The original class distribution, which says that 39% are good and 61% bad, is not necessary to keep in mind. The two most important figures presented in the table above are percentages based on the number of reports in each category. If the presented numbers would be absolute the situation would be different and the original class distribution would be relevant. The reason for this is that the two classes are not equally likely. Randomly removing reports would also remove more bad than good reports when it comes to absolute numbers, simply because bad reports are more common than good reports. This is not the case when considering percentage numbers based on the number of reports in each class.

As we can see in Table 10 the most promising algorithms were DecisionTable and J48. Except for the performance itself there were also other factors worth considering. One such factor was the readability. A decision tree is very readable as we have seen. This can be important for checking the model and understanding it. Because of this and the relatively good performance shown the J48 decision tree algorithm was chosen.[14] It is worth stating that this is not necessarily the algorithm

14 This probably was no surprise since that was the only type of machine learning algorithm described in the Background section.

that would come out as a winner from a more extensive comparison. The rule-based algorithms, for example, appeared to be performing well under tests, but they also appear to be more demanding when it came to time and memory. These algorithms could, on the equipment available, only run on very small subsets of the data, which was unfortunate since it did not give comparable results.

5.4. Choosing Parameters

The algorithm that was chosen, decision tree, has the ability to output a certainty level. This means that it is possible to favor one type of mistake. The exact value for this uncertainty level does not have to be decided at this point. The value should be kept as a setting that the Customer Service or the Management can tune so that the classifier is showing an acceptable trade-off. The reason this issue is brought up here at all is that it might affect the tuning of the algorithm. Small variations in this parameter are not likely to affect the behavior and performance drastically but large changes might. In other words it is good to establish an approximate value here. One reasonable value might be 0.25 (i.e., 25%), meaning that if there is one quarter of a chance that the report is good, it should be saved. This value will be used when testing how classifiers perform. In reality this value does not have to be exactly this. The important thing is, as stated above, that it is within the same range.

The parameters available for the J48 algorithm are seen in Table 11. Some of these parameters are boolean, meaning that they will either be “on” or “off”, while some others have numeric values. Yet others of them depend on some other parameter. It is for example possible that one parameter decides whether a certain method should be used or not and one or more other parameter specify specific behavior of that method. If that method is not used at all there is no reason to tune specific behavior of it.

Some of these parameters can be dealt with right away, most notably *debug* and *saveInstanceData* since they do not affect the actual process. Another parameter that will not be considered is *seed* since the way this particularly data is randomized should not be of importance when working with another set of data. The parameter *unpruned* will not be considered either since pruning, a way to make trees more general by removing spurious branches, is vital when data is noisy. The same goes for *subtreeRaising*, since subtree raising is a form of advanced pruning. The option *useLaplace* makes probabilities a little “safer”. When this is turned off it is possible that a certain report would be classified as either 100% bad or 100% good, which might be inappropriate in some situations. Dividing by zero is, for example, very different from dividing with a small nonzero value. This is not of importance here so *useLaplace* will be turned off. The last parameter to be left alone is *binarySplits*. By not using binary splits the tree might get wider, but that should not cause any concern in this case.

Table 11: Parameters available for the J48 algorithm. This table is a verbatim copy from the corresponding help section within Weka.

Parameter	Description
binarySplits	Whether to use binary splits on nominal attributes when building the trees.
confidenceFactor	The confidence factor used for pruning (smaller values incur more pruning).
debug	If set to true, classifier may output additional info to the console.
minNumObj	The minimum number of instances per leaf.
numFolds	Determines the amount of data used for reduced-error pruning. One fold is used for pruning, the rest for growing the tree.
reducedErrorPruning	Whether reduced-error pruning is used instead of C4.5 pruning.
saveInstanceData	Whether to save the training data for visualization.
seed	The seed used for randomizing the data when reduced-error pruning is used.
subtreeRaising	Whether to consider the subtree raising operation when pruning.
unpruned	Whether pruning is performed.
useLaplace	Whether counts at leaves are smoothed based on Laplace.

The remaining four attributes are *confidenceFactor*, *minNumObj*, *numFolds* and *reducedErrorPruning*. The first parameter that will be dealt with is *reducedErrorPruning* which specifies what type of pruning will be used. C4.5 pruning, which is used if this parameter is off, is the default pruning used, but it can be changed to a pruning technique called reduced-error pruning. These two pruning techniques were used one time each, with no other parameters altered, and the result can be seen in Table 12. They are performing approximately the same, but the tree produced by C4.5 pruning has fewer leaves. The number of leaves a tree has is a way of measuring the complexity of the tree. It is desirable to have simple and small trees since these are easier to read and understand. They will also be faster since less computation are needed and it is also likely that fewer attributes are used which might mean that less attributes have to be computed in the first place. A large tree also runs the risk of being overfitted. Since C4.5 pruning performed well and had few leaves it was chosen. This also means that the value for *numFolds* will be irrelevant.

Table 12: Comparison of pruning techniques.

Pruning technique	Number of leaves	Accuracy	Percentage bad reports removed	Percentage good reports removed	Score
Reduced-error pruning	12828	56%	45%	27%	0.65
C4.5	6764	59%	52%	31%	0.65

The two remaining parameters, *confidenceFactor* and *minNumObj*, are both numeric. The best values for these were found through an experiment. A set of possible values were listed for each of the parameters and then all possible combinations were tested. Knowing which parameters to include in the list of possible values was hard. If the list would become too large the number of experiments would be too large, and if the list would become too short it was possible that no a good value would be found. A compromise between these two issues was using 0.15, 0.20, **0.25**, 0.30 and 0.35 as possible values for *confidenceFactor* and **2**, 5, 10 and 15 as possible values for *minNumObj* (default values are written in bold). The result of these tests is seen in Table 13.

The most promising parameter values appear to be 0.30 for *confidenceFactor* and 15 for *minNumObj* (marked in bold). This combination has the lowest score among all the tested combinations, but it has been stated that the score should only be used as a guideline so all the combinations should be compared against each other. One reasonable method for doing so is to start with the assumed winner and compare that to all the other classifiers. If no other classifier is performing better, the assumed winner is in fact the winner. There is no point in considering the classifiers that only removed a few reports, or not any of them, since choosing one of those would defeat the whole purpose of filtering reports. The selected combination is performing better than the other remaining classifier with the same *minNumObj* setting since they are both performing equally much good work, but the latter makes more mistakes. The estimated winner is also performing better than the two remaining classifiers with *minNumObj* set to 10. The latter is performing one percentage point more good work, but it makes two percentage points more mistakes, which makes the improvements less than the increased downsides. Comparing all the classifiers in this way is a tedious task, but for all of them the same connection will hold. The increased good work is less than the increased mistakes.

The dataset used here is the same as the one used in the previous section, except that its size has been doubled. The data here consists of 10% of the entire training data available.

Table 13: Comparison of different values for numeric attributes.

confidence-Factor	min-Num-Obj	Number of leaves	Accuracy	Percentage bad reports removed	Percentage good reports removed	Score
0.35	2	11188	60%	58%	38%	0.71
0.30	2	10031	59%	56%	36%	0.69
0.25	2	6764	59%	52%	31%	0.65
0.20	2	2786	58%	47%	26%	0.63
0.15	2	28	39%	0%	0%	0.61
0.35	5	5548	59%	54%	33%	0.67
0.30	5	2881	58%	47%	25%	0.62
0.25	5	915	58%	43%	21%	0.59
0.20	5	36	39%	0%	0%	0.61
0.15	5	29	39%	0%	0%	0.61
0.35	10	914	57%	43%	20%	0.58
0.30	10	69	57%	43%	20%	0.58
0.25	10	69	39%	0%	0%	0.61
0.20	10	68	39%	0%	0%	0.61
0.15	10	51	39%	0%	0%	0.61
0.35	15	81	57%	42%	19%	0.58
0.30	**15**	**63**	**57%**	**42%**	**18%**	**0.57**
0.25	15	63	39%	0%	0%	0.61
0.20	15	62	39%	0%	0%	0.61
0.15	15	50	39%	0%	0%	0.61

5.5. Finding the Most Suitable Subset

Creating a classifier requires a lot of memory and the machine used was not able to process all of the available data at once, instead only about 10% of the data could be processed given the resources at hand. This was problematic since important information might be available in the remaining 90% causing the classifier to act worse than it could. Another issue was that the classifier that was created would be different depending of which 10% of the dataset that was used. In order to find the best one other experiments had to be done. The data was randomly, but in a stratified way, divided into ten subsets. These subsets were used to create ten classifiers and each of them classified the rest of the data. In other words one

classifier was trained using subset one and tested on subset two to ten. The next classifier was trained using subset two and tested on subset one and three to ten. The same pattern was followed for all ten classifiers. The result of the ten runs can be seen in Table 14.

Table 14: Comparison of classifiers built on different subsets of the data (uncertainty setting 25%).

Subset	Number of leaves	Accuracy	Percentage bad reports removed	Percentage good reports removed	Score
1	236	57%	42%	18%	0.57
2	1535	57%	43%	19%	0.58
3	38	57%	41%	18%	0.56
4	55	57%	41%	18%	0.56
5	845	57%	42%	19%	0.57
6	47	57%	41%	18%	0.56
7	1497	58%	43%	19%	0.58
8	1272	57%	42%	19%	0.57
9	723	57%	42%	18%	0.57
10	45	49%	22%	7%	0.56

The first nine classifiers produced similar results, where a little over 40% of all bad reports were removed and a little fewer than 20% of all good reports were removed. It is also interesting to note that the size of the trees varied a lot, while the result was approximately the same. One subset, the tenth, looks different. It had notably smaller percentages in both the columns. That classifier rather classified a report as good than bad.

It is reasonable to get suspicious when nine values are very similar and one differs notably, but it is also worth noting that the ratio does not vary equally much. The tenth classifier was neither substantially better nor substantially worse than the other nine; it just had another way of assigning the probabilities. It was, however, worth changing the uncertainty setting in order to make sure that there was nothing fundamentally different between the first nine classifiers and the tenth. The exact amounts of bad and good reports removed were decided by the way uncertain classifications were handled. Please recall that 25% were chosen fairly randomly. Changing this setting slightly would make the result look different and perhaps the tenth classifier would not differ as much. A first attempt at showing this was to reduce the level used to 20%. This means that the probability for a report to be good must be less than 20% in order for that report to be flagged

for deletion. Except for this small modification the same settings as above were used and the experiment was rerun. The result can be seen in Table 15.

Table 15: Comparison of classifiers built on different subsets of the data (uncertainty setting 20%).

Subset	Number of leaves	Accuracy	Percentage bad reports removed	Percentage good reports removed	Score
1	236	44%	9%	3%	0.58
2	1535	42%	8%	3%	0.60
3	38	49%	20%	6%	0.56
4	55	43%	7%	2%	0.59
5	845	43%	8%	3%	0.60
6	47	42%	7%	2%	0.59
7	1497	49%	21%	8%	0.57
8	1272	43%	8%	3%	0.60
9	723	42%	7%	3%	0.60
10	45	49%	20%	6%	0.56

The new experiment also showed some difference between different classifiers. The third, seventh and tenth classifiers had comparable results with small internal differences. The same was true for the remaining seven classifiers, but the two different sets differed notably. Given this new information there should not be any reason to suspect that the tenth classifier was substantially different from the rest. In this experiment it produced the same results as the third classifier, even though these two differed when the original uncertainty setting was used.

One of the classifiers seen here had to be selected. There was no need to use the original uncertainty setting so any of the results in these two tables could be selected, which gave a total of twenty possibilities. The first nine classifiers removed a fairly large amount of good reports with the original uncertainty setting. The Customer Service was consulted about this but they were not able to give an exact answer to how big the acceptable loss of good reports is. They could, however, say that the numbers shown here, with numbers almost reaching 20%, is too big. This disqualified the first nine classifiers with the first uncertainty setting. Another large set of classifiers that were less attractive was the ones that removed very few reports with the second uncertainty setting. Putting a system like this in place requires a bit of work and some changes to the existing administrative interface. For this to be worthwhile the potential of work saved has to be more substantial. Only three possibilities remained for the latter uncertainty setting. Two of these, classifier three and ten, were producing the same results, which is better

than the result produced by classifier seven. Classifier seven removed one percentage point more bad reports, but at the same time it removed two percentage points more good reports. The remaining options were now classifier ten with the first uncertainty setting and classifier three and ten with the second uncertainty setting. Out of these the tenth classifier with the first uncertainty setting was chosen. This combination of classifier and uncertainty setting was removing two percentage points more bad reports while only removing one percentage point more good reports. Another appealing characteristic of the tenth classifier is that the produced tree was fairly small. The seventh classifier for example had a substantially more complex tree.

The way classifiers have been compared here is not perfectly fair. It is possible that some other setting, say 22.75%, would make one of the other classifiers to come out as the winner. The fairest method for doing this would be to determine how big the acceptable loss of good reports is. There is no exact number specified for this now, but assume it is 10%. When this level is established each classifier would be allowed to make this many mistakes and the uncertainty setting would be calculated based on that. Ideally this would mean that all classifiers, while having different uncertainty settings, would have the same amount of incorrectly removed good reports. The best classifier would then be easy to spot by simply looking at the one with the highest amount of deletions for bad reports. One drawback with this method is that it is not sure that all classifiers would be able to reach a removal rate of exactly 10% for good reports. In fact, it is not likely that any of them would reach *exactly* 10%. This must be handled in some way and one method is to simply aim for 10% and if that is not possible the closest possible smaller value (e.g., 9.5%) is used. Unfortunately this method of making a more fair comparison requires a software tool in order to be performed efficiently. Writing such tool would require time and it can be questioned if it would be worthwhile. By looking at how the classifiers have behaved, none of them seem to differentiate much. Because of this it was decided that writing the needed tool was not worth the effort and the time could be used in a better way.

One fully functioning classifier had at this point been created. It removed 22% of all bad reports but in the process of doing so 7% of all good reports were removed too. These numbers were far from the wanted numbers, which would be 100% and 0% respectively. Some approaches to improving these numbers were tried and these will be explained in the following sections.

5.6. Favoring Memory Instead of Time

The above classifier was created by using a subset of the available data instead of the entire dataset. The reason for this was that Weka uses a large amount of memory which sets a limit for how much data that can be process. The reason Weka uses so much memory is because it does a fair amount of optimizations

when it comes to time, resulting in short run-times. In some situations the run-time can be important, but this is not really the case here. It would certainly be worth waiting a while longer if it would result in a more accurate classifier. Focus was put on the inner workings of Weka in order to see if the optimization bias could be moved from time to memory. Since Weka is open source, it is possible for everyone interested to modify the program to their needs. After some research this idea was dropped for two reasons. First, the memory requirements would have needed to be changed fairly much. Only 10% of the data could be run with the unmodified version so it was reasonable to expect much work in order for this to work. Second, the most promising lead that came up on how to optimize Weka for memory was to change how database interaction works. Weka traditionally reads the contents of a database into memory at first and then accesses the data from memory each time it needs the data. Having all data cached in memory is not necessary when creating decision trees since some information will never be used more than once, meaning that it is not needed after first use, and so on. Unfortunately it was not considered feasible to modify the memory model. The reason for this was that Weka is built in a way which lets it get input from several sources, including plain text files. The input is then transformed into objects in memory and all processing is done with the help of these objects. Rebuilding this system in order to keep the connection between objects in memory and database tables, in order to use lazy loading for example, was deemed too hard and too risky.

A second approach of solving this problem was to write a new program for creating decision trees. This program would be able to use a completely different memory model than Weka by, instead of loading the entire dataset into memory, utilize the fact that the dataset was already in a database system competent of doing computations. One common operation when constructing decision trees is to calculate the class distribution (for some subset of the data). Weka practically solves this by looping through its in-memory objects and calculating the distribution by looking at the class on these objects.[15] An alternative approach, which could be used by this new program, is to construct the appropriate SQL statements for calculating this and then let the database system, which already manages all data, do the computation itself.[16] Similar solutions can be done to other operations needed when constructing a decision tree.

The new program, called DecisionTreeCreator, uses very little memory. It only handles the actual logic and lets the database system handle the data. It might be worth stressing that the actual data never leaves the database system. The only thing that is transmitted over the network, if DecisionTreeCreator and the database

15 This is actually a simplification. Weka uses all sorts of tricks in order to improve performance and in reality the in-memory objects are not used this naïvely.

16 The SQL statement for doing this could be as simple as

```
SELECT class, COUNT(*) FROM SomeData GROUP BY class;
```

system are running on different hosts, is data in aggregated form. This might reduce network load, but this is not guaranteed. The number of requests to the database system can be large and even if each response is small, the total amount of data can be fairly large. The fact that raw data never leaves the database system might also be worth considering from a security perspective. While the client computer, the one running DecisionTreeCreator, needs less resources, the load on the database system may increase. The database system will have to perform many calculations during an extended period with DecisionTreeCreator. When using Weka the database system will only be affected in the beginning. It is hard to state anything definitive about load on the network and on the database system. What has been mentioned here should hold in the general case, but it is possible to find extreme cases when in does not. If there is a huge amount of data but a very simple pattern among them, transmitting all data would be costly while doing the few needed computations in the database system would be cheap.

DecisionTreeCreator works by building the tree recursively. The process is started by looking at all attributes and trying to find the most suitable attribute to use. This process is orchestrated by the program, but most computations are done in the database system. When the most suitable attribute has been found the process is repeated on each of the subsets created. This is handled by building up database commands and passing them along, making them more and more restrictive at each node. Somewhat simplified this is the same as adding a test to the WHERE clause of the SQL statement at each node in the tree.

The goal of DecisionTreeCreator was to favor memory instead of time which if successful would make it possible to run the entire amount of data available instead of just using a subset. The downside of this was that it would take longer time. Unfortunately it turned out during testing of an early prototype that the time consumption increased more than expected. In a way it can be said that too much weight was put on reducing memory and that this was taken to an extreme. One mitigation to this was to start with this memory efficient program and do time optimizations on that. One natural such optimization was to move some of the calculations back from the database system to the client program. It turned out that the computation for numeric attributes was very time consuming, making it a perfect candidate for optimization. Categorical, or nominal, values are easy to split since there is a natural way of doing it. This is not the case with numeric attributes which can be split in a variety of different ways, making the computations more extensive and costly. These computations can, however, be made fairly efficient if written cleverly. The computations were made even more efficient by reading all the numerical values from the database and saving them in memory. This reduced time consumption drastically for processing numerical attributes and a notable speed-up could be seen in the program as a whole. Worth pointing out here is that this is a step away from the original design decisions. This means that raw data,

not just aggregated data, was transferred from the database system to the client, with all implications that follows from that. One such implication is that the client no longer was able to handle an arbitrary large dataset. In the original version where all computations were made in the database system, the client was practically independent of the size of the dataset. This change modified that statement a bit, but DecisionTreeCreator was still able to handle large amounts of data. The limitation was that all numeric values for one attribute must fit in an array in memory.

Even though the improvement from the first optimization was large the program was still too slow. Before the program was developed any further it was tested by timing how long it took to build a classifier on different subsets of the dataset. The results were then used to estimate how long it would take to run the program on the entire dataset. Table 16 shows the result of running the program three times with different number of examples, but except for that in the same environment with the same parameters. Linear regression was used in order to find an estimate for the time needed to run the program on the entire data set. The least-square criterion [GWF02] was used to fit a line through the available data points and this line was then used in order to make a prediction. The estimate was that the run-time would be little over 15 days. This was a very long time and there was also an associated risk with running a program during such a long period of time. Power failures, network problems and reboots could abort the computations and loosing days or weeks of valuable time. Matters get even worse when considering that the run-time might not be linear as assumed above. The time needed to build a tree depends on how easily the data can be categorized and this might very well increase with an increased number of examples. The most important thing is an approximate value and the one stated above should be accurate enough, but it should be noted as an extra risk that the run-time might not be exact.

Table 16: Run-time measured during initial testing of DecisionTreeCreator.

Number of Examples	Time (minutes)
100	3
500	18
1000	44

The estimated time requirement for running DecisionTreeCreator on the entire dataset was longer than expected. It was unfortunately so long that it was considered unfeasible and also, as mentioned above, very risky. This had the implication that the development of the program was aborted.

5.7. Increasing Memory

The original problem was that Weka had too little memory available in order to process the entire amount of data available. The solutions tried above were first to modify Weka to use less memory and after that to build a new program for building the classifier. There was also another possibility and that was to increase the amount of available memory. In order to get an estimate for the amount of memory needed a similar test as the one above was conducted. The memory requirement was checked for a few runs and the result can be seen in Table 17.

Table 17: Memory usage for Weka.

Number of Examples	Memory (Mb)
1000	20
10000	110
30000	310

These numbers seems to follow a linear pattern very accurately. This is also reasonable when considering how memory is used. The examples are read from the database and saved in memory. This is not affected by other parameters, such as the complexity in patterns, the same way the run-time might be. The estimated amount of memory required to run the entire dataset turns out to be a little less than 5 Gb.

The memory required is large and not something found in the regular workstation today, but it is still not unreasonable. Most servers at Stardoll have 8 Gb memory which means that the program would be able to process all data if it could be run on one of those. Efforts were made in order to make this happen and it looked promising since new servers were delivered at the time. These servers were not yet in use and using one of them for this purpose would not affect or degrade any running services. Unfortunately it turned out that it was not possible to use a server for this work which meant that this path also led to a dead-end.

5.8. Creating a Tree Based on Fewer Attributes

The first method for dealing with the limited amount of memory was to reduce the number of examples. Another possible method mentioned earlier was to reduce the number of attributes. The approach that was used here aimed at a middle way where attributes were removed, but not to the extent where all examples could fit in memory and be processed. Finding a reasonable trade-off was difficult. The approach used was to assume a reasonable value and see the outcome of that attempt. If successful more tweaking can be done.

Reducing the number of attributes was done by iteratively creating decision trees and removing the most important attributes. The randomized 10% of the data,

which was used before to find the most suitable parameters for the decision tree algorithm, was reused for this process. The attribute used in the root node together with the attributes found in the subsequent two levels were extracted. The first iteration extracted seven attributes. The process continued for another six iterations after which a total of 38 attributes had been extracted. This should be an appropriate value to use in the first run.

The attributes extracted, the ones thought to be the most important and influential, were combined to form a new dataset. This dataset was used to build a new decision tree. Since 10% of the original dataset, with approximately 200 attributes, could be processed it is reasonable to assume that somewhere around 50% should be processable in the new dataset, with approximately 40 attributes. Unfortunately testing showed that this was not really the case and the processable amount of data had to be reduced to 40%. The total amount of data was split into three pieces each containing 40% of the data. The first subset contained the first 40%, the second the middle 40% and the third the last 40%. Naturally an overlap was created and the second subset overlapped the first by 10% and the third equally much. When the split was done, a test similar to the one done when creating the first classifier was performed. A decision tree was created with one subset of the data and then that tree was used to classify the other subsets. The parameters that were used when creating the first classifier were used again during the creation of this classifier. Uncertain reports were also handled in the same way, which means that only report with a probability of less than 25% of being good were flagged for removal. The result of these tests can be seen in Table 18.

Table 18: Result of classifiers built on fewer attributes (uncertainty setting 25%).

Subset	Number of leaves	Accuracy	Percentage bad reports removed	Percentage good reports removed	Score
1	672	58%	43%	19%	0.56
2	1256	57%	42%	18%	0.57
3	1276	58%	44%	19%	0.57

A fairly large part of all good reports were removed by all classifiers with the settings above. It is, as mentioned earlier, unacceptable to almost loose one fifth of all good reports written and hence none of the classifiers above showed a promising result. In order to compensate for this the process for handling with uncertain reports could be modified slightly the same way it was modified when the first classifier was produced. The setting used above, 25%, was reduced to 20%. In clear text this means that a report was not removed if the probability that it was good was 20% or more. All other settings were kept as they were above. The result is shown in Table 19.

Table 19: Result of classifiers built on fewer attributes (uncertainty setting 20%).

Subset	Number of leaves	Accuracy	Percentage bad reports removed	Percentage good reports removed	Score
1	672	49%	20%	6%	0.57
2	1256	49%	20%	7%	0.57
3	1276	49%	20%	7%	0.57

The figures presented this time are more similar to the ones seen when training the classifier using all attributes and the losses of good reports are down to more acceptable levels. The best result here, produced by the first classifier, is the same as the already selected classifier produces with the same uncertainty setting. The largest difference between them is that the classifier produced here has many more leaves. A smaller tree is easier to read and understand and it will also be faster to process, hence there is no reason to exchange the already created classifier for this classifier.

5.9. Bagging and Boosting

Bagging and boosting are methods for improving performance by combining several different classifiers. One downside of these ensemble methods is that they require a lot of memory, a resource that was already running low. When all attributes were present these ensemble methods was only able to run on a small amount of the data. One solution to this problem was to use the dataset with the reduced number of attributes. Testing showed that approximately 10% of the data could be processed when the number of attributes had been reduced to 38.

Two different classification algorithms were used in the experiment performed. The first was DecisionStump, which is a simplified decision tree algorithm. It builds trees that are only one level deep and the algorithm is especially constructed for being used with this sort of meta classifier. The second algorithm that was used was the decision tree algorithm used above, J48. This algorithm was used twice with different parameters. First it was used with the standard parameters and second it was used with the parameters that were found to be optimal in Section 5.4 on page 55. These three combinations were used in conjunction with both bagging and boosting and the result is shown in Table 20. The bagging implementation used was Weka's *Bagging*[17] and the boosting implementation was *AdaBoost.M*[18], also available in Weka. The default parameters have been used for both these algorithms, including the number of iterations which defaulted to 10 for both. The result of running the J48 decision tree algorithm without bagging or boosting is also shown in the table for comparison.

17 Default parameters are "-P 100 -S 1 -I 10".
18 Default parameters are "-P 100 -S 1 -I 10".

Table 20: Comparison of ensemble methods.

Meta classifier	Classifier (parameters)	Accuracy	Percentage bad reports removed	Percentage good reports removed	Score
Boosting	DecisionStump	54%	34%	15%	0.58
Boosting	J48 (default)	57%	51%	35%	0.71
Boosting	J48 (optimized	57%	49%	31%	0.67
Bagging	DecisionStump	39%	0%	0%	0.61
Bagging	J48 (default)	58%	47%	24%	0.60
Bagging	J48 (optimized)	57%	42%	19%	0.57
(none)	J48 (default)	57%	42%	18%	0.57
(none)	J48 (optimized)	57%	42%	18%	0.57

One thing that is immediately noticeable is that one classifier is not removing any report, neither bad nor good, at all. This might seem strange, but the reason is simply that no single example was less than 25% likely of being good. A second thing one might notice in the table is that differences between the results are more noticeable than before. Both the removal rate for good report and the removal rate for bad reports vary among the different runs complicating comparison of them. The comparison tool mentioned in Section 5.5 on page 61 would have been to great help. A third thing that might attract attention in this table is that the simple decision tree was performing almost identical no matter if standard parameters or the optimized parameters are used. This can only be seen as a sign that different parameters have different effect in different environments. Recall that the difference between these two sets of parameters was most notably when the data used contained more attributes (see Section 5.4 on page 55).

The two single decision trees can be seen as reference points in this experiment. Their output looks familiar (compare with Table 14 on page 59) and the first classifier was based on a single decision tree. The result of an ensemble classifier must be better than this in order to be considered a possible classifier. It is naturally better to use the single decision tree if it outperforms an ensemble method, but the same is also true if they are performing equally well. The single decision tree is simple to read and understand, which is a benefit in itself. A simple classifier will also, which might be of more importance, require less resources to run. Doing a classification with a single decision tree is faster than doing the same classification using ten or so decision trees.

With the single decision trees as reference points it is now possible to look at the other classifiers one by one, except for bagging with DecisionStump since the result for this classifier is so different from the other results. It might be easiest to

start with the remaining two classifiers utilizing bagging. The one of them that was using the optimized parameters for the J48 decision tree was removing 42% of the bad report and 19% of the good reports. This was not an improvement over the single decision tree. In fact, it was slightly worse in this particular case. The remaining bagging classifier was removing 5 percentage points more bad reports when compared to the single decision trees, which was good, but it was also removing 6 percentage points more good reports, which was very bad. Since the downsides have increased more than the benefits, this classifier was no candidate for the throne either. The same reasoning can be applied to boosting with DecisionStump as classification algorithm. The two remaining classifiers that utilized boosting differ by only 2 percentage points when it comes to removal of bad reports, but with 4 percentage points when it comes to good reports. This makes the parameter optimized version a winner over the version with unoptimized parameters. Unfortunately neither of them was better than the single decision tree. None of the five classifiers tested so far have performed better than the single decision tree. The only remaining hope is bagging combined with DecisionStump. A first attempt at getting a more helpful result then the one presented here was to change the uncertainty settings. If the threshold was changed from 25% to 30%, this particular classifier removed 34% of all bad report and 23% of all good reports. When it comes to the removal rate of bad reports this was similar to how boosting with DecisionStump performed above, but the removal rate for good reports was much worse. In summary it can be said that none of the tested classifiers here performed better than the single decision tree and it was concluded that it was not worthwhile to try to improve the results since initial testing was not promising.

It should be noted that the comparison method used is not guaranteed to find the most optimal classifier. It is not perfectly safe to assume that the classifiers are producing linear results, so a comparison like this might not be perfectly accurate, but it should be accurate enough for a first test like the one above in order to find out if a path is worthwhile following or if it is better to spend time following another. It might also be the case that other uncertainty settings would produce a result that would make one of the ensemble classifiers better than the single decision tree. Bagging with the parameter optimized decision tree has a result quite similar to the single decision tree, and it is possible that it would perform better on another set of data. However, since the increased complexity has an associated cost, if nothing else it requires longer run times as noted above, the advantage should be clear in order to use such a method.

5.10. Building a Custom Ensemble System

A downside with the ensemble methods described in the previous section was that they required a large amount of resources. Even when the attributes were reduced

significantly only 10% of the data could be processed. This is a drawback since the process needs much data in order to recognize all patterns. In some cases, such as for the boosting method, much resources are needed since the process is iterative, but for others there is no obvious reason for this. The ensemble process in itself consists of parts that can be separated from each other. With this in mind it should be possible to construct a tool that requires less memory than Weka does.

An ensemble method is, as previously stated, just several classifiers, in this case decision trees, working together. They are trained independently on different datasets. When they are used to make a classification, the classifiers are each requested to do a classification. The algorithm then collects the results from the classifiers and outputs the most frequent output. From an individual classifier's perspective there is no difference between working alone or as part of an ensemble. This insight can be used to modularize the procedure which in turn saves memory both in the training phase and in the classification phase. Instead of training all classifiers in the same run, which would require the training data for all classifiers to be loaded in memory at the same time, it is possible to only load the data needed for one particular classifier. The data needed to train the first classifier can be loaded into memory before the first classifier is trained, but when that training process has been completed the first set of data can be discarded before the second set is loaded and the second classifier is trained. This process reduces the need for memory during training significantly. This idea can be applied not only to the training phase but also to the classification phase when reports are fed to the algorithm in order to be classified. Weka keeps all trees in memory and runs them all, but this is not needed. It works perfectly fine to run the classifiers one after another instead. The problem that must be taken under consideration is how the voting will be done. If all classifications are done by the same program it is simple to use a variable for this purpose. If the classifiers instead are done separately, perhaps as different programs, some more persistent storage, such as a database, must be used.

Traditionally simple ensemble methods only do a majority vote among the participating classifiers. This is a straight-forward approach but it has its downsides. One such downside is that this scheme does not take in consideration how sure a classifier is in its decision. It is for example possible that four classifiers that are all very certain about one decision should get precedence over six uncertain classifiers that suggest another decision, but the simple majority principle would say otherwise. One possible refinement of the voting system would be to use the certainty levels outputted by the trees. This level originates from a class distribution that the tree will associate with every classification. A typical output from the decision trees above might be “70% GOOD, 30% BAD”. This output can easily be translated to a figure, between 0 (for 0%) and 1 (for 100%), saying how likely this report is to be good according to the decision tree.

In the above example the result would be 0.7. If a set of trees all output numbers like this the overall “goodness” of a report could be found by summing the individual numbers together. Given *n* trees the result would be between 0 and *n*. An alternative to this is to divide the final result with the number of trees. The result in that case would be between 0 and 1.

It was concluded that this setting required a set of individually trained decision trees. Luckily, a set like that already existed. Recall that ten trees were created previously in order to decide which subset the first classifier should be trained on. Those ten trees, which were all trained on different parts of the dataset, could be reused at this point. One issue that required extra attention was the need for validation data. The total amount of data available here, which was all data except for 25% which were kept for testing purposes, must be used both for training the model and validating it. In order to avoid using the same data for both these two purposes, which could lead to overfitting, some subset of the data must not be used to train the (ensemble) classifier. This could not be achieved if all ten decision trees created earlier would be used, since that would mean that all data had been used for training, leaving no data for validation. The solution to this was to only use eight of the ten trees for training. Two different runs were done, one which used the first eight classifiers and one which used the last eight. This did not exhaust all the possible combinations, but since the classifiers' results were similar there was no reason to believe that one set of classifiers would work a lot better together than any other set. However, this could have be revised had the concept proved successful.

It turned out that training the meta classifier could be simplified by training the separate classifiers individually, but using the meta classifier to make classifications was harder. The solution found was to use a tool named StarClassifier that was developed in order to be able to make classification easily on examples stored in a database. This program will be explained in detail in Appendix A were it is used in the way it was designed. For now it is enough to know that this program reads examples from a database, classifies them with the help of a decision tree and writes the probability that the report is good back to the database. This tool could without modification be used for this purpose. The trick used was that the program was run eight times, each time with a different decision tree. Each time it ran it looked at the report, made a classification and wrote that classification back to the database, which had been extended to have eight new attributes, one for each tree to consider. When all the iterations had been made the total result could simply be found by adding the partial results together, preferably by using SUM in a SQL UPDATE statement. This process could be simplified further if the individual results were not considered important. In that case the configuration file to StarClassifier could be updated so it would add the partial results together instead of writing them to different fields in the database.

The result of the two runs can be seen in Table 21. The first subset is when the classifier is trained on the first eight parts of the data and validated on the last two, and the second subset is when the classifier is trained on the last eight parts and validated on the first two. The uncertainty level has been kept the same as in previous runs. This means that only reports that are less than 25% probable of being good have been deleted. Since the result of the eight classifiers have been added together this means that reports must have a value less than 2 in order to be deleted.

Table 21: Result produced by utilizing more data in an ensemble (uncertainty setting 25%).

Training subsets	Accuracy	Percentage bad reports removed	Percentage good reports removed	Score
1-8	57%	40%	17%	0.56
3-10	57%	41%	17%	0.56

The results shown in Table 21 are fairly similar to the results seen in Table 14 on page 59 from when the decision trees were tested separately. The best result seen here is 40% removal rate for bad reports and 17% removal rate for good reports. In the old experiment the best result, except for the last one which looked fairly different from the others, was 41% removal rate for bad reports and 18% removal rate for good reports, which for example tree three produced. The difference here is fairly small and it is hard to say that one is better than the other just based on the results, but there are two other aspects worth taking into consideration. The first one is about simplicity again. The third decision tree alone has 38 leaves and the eight first classifiers together have 5525 leaves. If these two produce similar result it is probably wiser to go for the first one. It is easier to read and, if necessary, manually fine tune a tree if it is small. It will also take less time to do the actual classifications if the tree is small. Except for the time itself it will be easier to configure the classification system if it only have to be run one time instead of several times after which the outputs have to be further processed. The second benefit of using the single tree is that it produced sensible results when the uncertainty setting was changed. In the original experiment with the single decision trees, as well as above, reports were removed if the probability of them being good was less than 25%. This removed almost 20% of all good reports, which might be too much. One solution to this problem is to change the setting from 25% to, say, 20%. The third single decision tree would under such circumstances remove 20% of the bad reports and 6% of the good ones. The decision tree eventually chosen at this stage, number ten, produced the same results when the setting was changed to this. This does not appear to be as simple with the ensemble method. If the uncertainty setting was changed in the same way

for this classifier almost no reports would be affected at all as seen in Table 22. It is possible that a middle way could be found even for these classifiers by using an intermediate value for the uncertainty setting but it is still a complication and a drawback. This method did not replace the simpler method already used.

Table 22: Result produced by utilizing more data in an ensemble (uncertainty setting 20%).

Training subsets	Accuracy	Percentage bad reports removed	Percentage good reports removed	Score
1-8	44%	9%	2%	0.58
3-10	42%	7%	2%	0.59

5.11. Ensemble with Veto

The ensemble method described above is not the only way multiple classifiers can work together. Another approach is to give all classifiers the right to veto the decision to remove a report. Conceptually this means that all classifiers once again classified a report, but instead of summing the result the maximum value was considered. Recall that the output produced by StarClassifier, which was the tool used, was the probability that the report in question was good. The output was between 0 (i.e., 0%) and 1 (i.e., 100%). If the maximum value was above some threshold, the uncertainty setting, the report was kept. In other words this means that all classifiers had to give the report in question a probability below this threshold in order for the report to get deleted.

In the previously described ensemble method the system worked by letting eight classifiers classify reports and writing the individual results back to the database in eight different rows. Since the same classifications were the foundation in this system too there was no need to redo any of the original work in order to build a system allowing veto rights. The rest of the work was changed slightly and instead of adding a last row holding the sum the maximum value was considered. That value represented the probability that the report was good according to the most optimistic classifier.

The uncertainty setting used was the same as in the previous experiments, meaning that a report was only deleted if the (maximum) probability that the report in question was good was less than 25%. The result of the experiment with this setting is shown in Table 23. Once again the values differ from each other even though the classifiers are expected to behave approximately the same. In fact, three fourths of the participating classifiers are the same in the two runs. The values shown in the table are fairly similar to the values seen in the last two runs when the different subsets were tested when the first classifier was built (see Table 14 on page 59). By comparing these two classifiers in the same way as classifiers have

been compared before it can be seen that this veto system does not provide any improvement over the single decision tree unless it is also followed by an equally sized or bigger increase of mistakes. What has been said about complexity holds in this case too. This veto system was substantially more complex than a single decision tree and using a more complex system should be well justifiable. Simply put, this was not a better system than the one that already existed.

Table 23: Result produced by ensemble with veto.

Training subsets	Accuracy	Percentage bad reports removed	Percentage good reports removed	Score
1-8	57%	40%	16%	0.56
3-10	49%	19%	6%	0.56

5.12. Description of the Classifier

The first classifier created turned out to be the one that finally won. The improvements tested did not outperform this classifier and it was therefore no reason to change. The classifier selected was a decision tree, created by the J48 implementation in Weka. The settings used can be seen in Table 24 (see Table 11 on page 56 for descriptions of the different parameters).

The classifier was trained with 10% of all the training data with all attributes present. This particular subset was the last of the ten tested subsets. Since different error were not equally severe a report had to have a probability of being good of less than 25% in order to be classified as bad. On the remainder of the training data this classifier successfully removed 22% of all bad reports while unfortunately removing 7% of all good reports.

Table 24: Settings used for the classifier.

Parameter	Value used
binarySplits	false
confidenceFactor	0.3
debug	false
minNumObj	15
numFolds	3
reducedErrorPruning	false
saveInstanceData	false
seed	1
subtreeRaising	true
unpruned	false
useLaplace	false

6. Using the Classifier

Training a classifier is not the last step to take in a machine learning process. Some further considerations must be dealt with in order to make the classifier useful in practice. These considerations will be discussed in this section.

6.1. Theoretical Aspects

There are two aspects that must be considered before a filtering system is put in practice: what will happen to the filtered items and which items should be filtered. Both these questions will be discussed below.

Filtering systems aim at separating different classes of entities from a set where they are mixed together. In some cases both classes are wanted but it is desirable to separate them anyway. A large multinational corporation might for example want to separate incoming emails based on which language they are written in. Messages in English should be redirected to an English speaking office, messages in Swedish to a Swedish speaking office, and so on. In some other systems all classes are not wanted. Spam messages are for example not wanted at all. In that case it is desirable to get rid of the unwanted items and keep the others.

Even if one class is not wanted at all there are different approaches that can be employed. The most straight-forward one is to simply delete all unwanted items. It is usually simple and fast to do so and no further processing is needed. The main problem with this approach is that the system can make mistakes. If filtered out items are permanently deleted there is no way of either detecting or undoing those mistakes. In some cases this is serious drawbacks and another solution might be preferable. In a spam filtering system it might be better to keep filtered out messages in a certain folder. The recipient of the messages could then browse through the folder if they wanted. It is also possible to imagine that messages are only kept in this folder for a certain amount of time. Say for example that a spam message is automatically deleted after one week unless someone marks it as non-spam.

In some situations it is for one or another reason a good idea to do initial filtering before the actual filtering is done. More generally the issue is about using several layers of filtering. Possible reasons for using an initial filter include safety and performance. It might be the case that a person receives both notifications

about new software releases and love letters. Both these classes contain spam messages and legitimate messages. However, the love letters are so precious that it is worth letting more spam messages through just to make sure no love letter is ever erased. One solution in this case is to make a rudimentary initial filtering that aims at finding potential love letters and make sure that those bypasses the actual spam filtering. Another reason to use initial filtering is to save resources. It might be the case that the total dataset is very large but some simple constraints can reduce it substantially. A person given the task to find all primes between, say, 100000 and 100050 would probably find the task a bit challenging if no tools were to be used. A first step could be to immediately remove all even numbers, since the only even prime is 2 and that number is not present here. This initial filtering would remove half of all potential numbers almost without any effort.

Initial filtering was used to improve performance in the previously mentioned image analysis system [BAS+98]. The aim of the system was to detect volcanoes on Venus based on a set of images. The approach used was, conceptually, to assume that all pixels in the image could be volcanoes and then a filtering system was used to filter out the actual volcanoes. Letting the filtering system work on all pixels in the image would require too much computational resources. The mitigation used was to employ a Focus of Attention (FOA) system that did a first screening of the image. All pixels that did not show any signs at all of being an actual volcano were filtered away directly.

How an appropriate filter should be designed depends on the domain. In the volcano example it might be appropriate to consider geological features that are easily noticed or known knowledge about images in general. It is not necessary to investigate every pixel in a large area where all pixels look the same since no volcanoes, nor anything else, can be detected there. In the spam filtering example it might be a good idea to consider the email address of the sender or words in the message. Some spam filters let its user create a list of words that acts almost as passwords [Gra02]. All messages that contain any of the words found in the list will bypass the spam filter.

When actually implementing an initial filter it can either be incorporated as part of the primary filter or kept separately. The first method requires changes to the classifier used but not to the surrounding system. If a decision tree is used the first method could simply take the form of adding extra conditions and leaves near the root of the tree. All entities would still go through the same systems but the ones fulfilling the conditions would be handled quickly once they reached the filter. The second method, which does not require any change of the classifier, might be more appropriate if a model which is harder to modify is used. For this to work the surrounding system must be changed so that some entities are not fed into the filter. The change can sometimes be as easy as changing a configuration file.

6.2. StarClassifier

Weka is a good tool for training classifiers and comparing the results of using different parameters. Once a classifier is trained and ready to be used Weka might, however, be a little cumbersome, especially if Weka is not already installed. It is sometimes the case that a classifier is trained and used on different machines. To solve this problem the tool StarClassifier was developed as part of this work. StarClassifier is light-weight, easy-to-use and fairly standalone. The tool is built with the purpose of fitting well in the environment at hand. It should not be hard to setup and configure the tool so that it filters abuse reports according to the model created. StarClassifier makes it possible to make the theoretical work presented here into something practical.

StarClassifier works by first reading examples, in this case abuse reports, directly from a database. The examples are then classified with the help of an already created model that should be made available to the tool. Using Weka is the preferred way of creating the model in question. Once a classification has been made the result is written back to the database. The result outputted is not a class (e.g., BAD) but rather a probability value (e.g., 42%). It is configurable if the value should be the probability that the report is good or if is should be the probability that the report is bad.

An overview of how StarClassifier fits into the overall system can be seen in Figure 7. Weka uses examples from the database to create a model, which was done in previous sections of this report. StarClassifier then uses that model together with input from the database to make classifications. The results of those classifications are written back to the database.

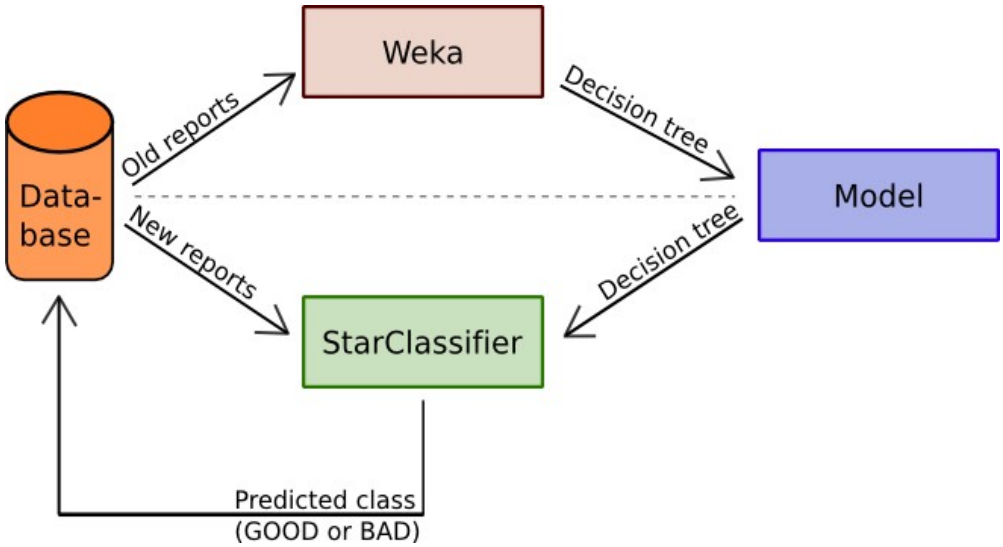

Figure 7: Overview of how StarClassifier fits into the overall system. The overview is simplified slightly by showing the output from StarClassifier as a class. In reality the output is the estimated probability.

StarClassifier is built with Stardoll in mind, but it can be used by anybody who has a similar need. StarClassifier is not tied to this particular domain and can be used whenever a decision tree should be used to make classifications on examples stored in a database. The tool is free software, licensed under the GNU General Public License (GPL) [Fre07]. Instructions about how to obtain and use StarClassifier can be found in Appendix A.

6.3. Handling of Rejected Reports

Up until now reports that have been filtered out have been referred to as deleted or removed, but exactly how reports that are flagged by the system should be handled is not decided. There is no need to decide on this now since it does not change the overall behavior of the system but it is worthwhile to go through the possibilities. The first, and the one used so far, method of dealing with the bad reports is to ignore them completely. This is basically what happens today if a report is classified as bad by a human, but letting a computer do this decision appears to be ethically more complex. Whether or not this is a good idea also depend on the accuracy of the filtering algorithm. The second possible method is to save all the reports that are filtered out for a certain period of time and then send them back to the users who filed the reports. Those users would get a message saying that the report has not been fully understood by the robot system. If a user still wishes to file a report they will have to find the object again and file a new report. At first glance this method might seem inefficient. It would, just to mention one thing, create a huge amount of messages. However, there might be some benefits from a solution like that. Most importantly it might force users to “cool off”. The Customer Service believes that at least some of the bad reports are written by upset users against former friends after a disagreement between the two. It is for example possible for two friends to suddenly fall out in school and then report each other for no reason. If these users are asked a few days later if they really want to go ahead and file the report they will probably change their minds. Another benefit that would be gained is user education. The current system does not give users any feedback about how their reports are handled by the Customer Service. It is, in other words, possible for someone to write bad reports without knowing about it. Perhaps the reports appear very clear and concise to the user writing them. Letting users know when their reports cannot be understood can therefore be a good thing in itself. One last benefit is reduced redundancy. From time to time it happens that a user writes something inappropriate and shortly after, perhaps after being warned, changes it to something appropriate. Reports filed on this content are no longer useful since the problem has already been taken care of. If the users who filed these reports would go back to the content, in order to refile the report, they would notice this.

No matter which of the following methods that will be used the first step will be to compare the classifier's result with the result currently set by the Customer Service. The classifier has been trained and validated using old data. If the preprocessing has been done properly this should match how that particular data would appear during actual classification, but it is possible that a slightly different result would be given. Another reason there might be a difference between old and new data is that the reporting behavior can have changed since the database was copied. A first step could be to simply show in the administrative interface which reports that would have been deleted by an automated system.

6.4. Protected Reports

It has throughout this report been assumed that all filed reports should be filtered through this system. In reality this does not have to be the case since extra conditions can be added in order to protect reports that are more valuable than others. These reports should then bypass the filtering system and automatically get saved. The justification of using these extra conditions is that some reports might be considered so important that all of them should be checked by hand. When a report is filed the report has to be categorized by the reporting user. One of these categories is threats and this might be considered so important that the risk of losing one helpful report in this category is not worth taking. On the other hand one can not let too many reports be caught by this initial filtering since that would defeat the purpose of the system. The category threats is, as mentioned, a category with many bad reports since this is a typical category that is used when reports are filed out of ill will.

The Customer Service was asked about which reports that should get caught by this initial filtering. They came up with two sets of conditions, where one considered the reported user and one considered the reporting user. When it comes to the reported user they wanted all reports concerning previously abusive users and new users to get automatically kept. Reports against these users are likely to be good and they should be checked manually. When it comes to the reporting user the Customer Service wants all reports filed by paying users or power users to bypass the filter. There is no clear definition of what a power user is or what requirements one need to meet in order to be a power user, but these users are the most active users on the site. The idea that paying users should not get filtered by an automated system was also seen during the case studies (see Section 2.3 on page 9) where it was noted that the community Faceparty used a similar policy.

Translating the Customer Service's somewhat vague conditions into exact ones that filtering could be based on is not trivial, nor is it needed right now. These conditions can easily be implemented after the system has been built and they may also be changed afterwards. Something that is good to sort out now is approximately how many reports that will be covered by these rules since this

affect the applicability of the system. In order to investigate this the conditions have to be translated into more strict rules. When it comes to the reported user the original condition was that previously abusive users and new users should pass. A previously abusive user has been defined as a user who has received a warning earlier and a new user has been defied as a user who joined Stardoll the last month. When it comes to the reporting user two other conditions were present. The first stated that paying users should not get filtered and this is easy since no translation or refinement it needed. It is easy and unambiguous to tell if a user is a paying user or not. The second condition is harder since it talks about the undefined term power user. A power user has here been defined to be a user who has more than 200 starpoints. There is no good rationale to do so and this number can not be justified more than any other number in the same order of magnitude. This is the second starpoint level that entitles the user to a reward. The fact that this number is fairly arbitrary should not cause too much concern since the goal here is to find some rudimentary definition for a power user and the starpoint system should be good enough for the purpose. The most important aspect is to capture the most active users and the criteria here select 2.5% of the entire amount of users.

When the four criteria specified above were applied to the dataset it turned out that a large amount of all report were covered by it. More specific, 75% of all reports were covered, meaning that three fourths of all data would never reach the filtering system built. The protection based on who the report was against covered 53% of all reports and the protection based on who filed the report covered 43% of all reports. Since these number together are larger than the number first specified it can be seen that there is an overlap and some reports are protected for multiple reasons. It should be noted that among these protected reports only 41% of all reports are good. This is indeed more than the same figure among all reports, which is 39%, but it could be assumed that the protected reports should have a much higher average quality since they are considered so important that these extra conditions are put in place to protect them.

7. Evaluating the Classifier

Before a classifier is used it is a good idea to estimate its performance. This performance estimate is calculated using the testing data that was put away in the beginning of the work. It is also worthwhile to consider what is a good result and what is not, and doing so require some knowledge about the circumstances and the domain. The domain-knowledge is important because different domains have very different requirements for what is acceptable. Consider for example a classifier that aims at predicting the outcome of roulette. The classifier only has to be slightly better than a random guess in order to make the owner very, very rich. When trying to determining if the next color will be red or black (ignoring green totally) it would be acceptable to have a classifier that is right 51% of the time. If instead a paternity test classifier is considered the result has to be a lot better than just a random guess. In that case it might very well be unacceptable with a classifier that is right 99% of the time. These two cases are extreme and in other situations it is likely that some intermediate performance level is what is needed for the classifier to be acceptable. Four different employees at Stardoll have been interviewed in order to get their view on the classifier's performance and what an acceptable level could be in this case.

7.1. Evaluation on New Data

In the beginning 25% of all valid data was put away for testing purposes, as mentioned in Section 4.2 on page 38. The reason for this was that it is important to test a classifier on data that is completely fresh in order to get an idea on how it would perform on new data. Before this data can be used to test the classifier it has to be preprocessed in a similar way that the training data was. It is vital that the data is presented in the same way during both training and testing. If, for example, missing values were replaced by some dedicated value on the training data, the same replacement must be done on the testing data. The main difference between preprocessing training data and preprocessing testing data is that the latter might be simpler. During training all possible attributes should be calculated because it is not possible to know which will actually be used by the classifier. During testing, on the other hand, this is known since the structure of the tree is known. Only

attributes that are actually considered are necessary to calculate. In this particular classifier only 14, out of the over 200 available at first, attributes were needed.

In Table 25 the result of this test is seen. The result does not surprise much since it is very similar to the one received during training. Between one fifth and one quarter, 22% to be exact, of all bad reports are successfully identified by the system, while a little under one tenth, 8%, of the good reports are incorrectly classified as bad.

Table 25: Result of final test.

Accuracy	Percentage bad reports removed	Percentage good reports removed	Score
49%	22%	8%	0.56

During the final test the uncertainty setting used was the one originally decided upon, in other words reports were removed only if the probability of them being good was less than 25%. This is not necessarily the setting that would be used in a real situation and it is worthwhile looking at how the classifier would perform under different uncertainty settings. Figure 8 shows the percentage of bad reports removed and percentage of good reports removed under uncertainty settings between 0 and 0.5. Using values higher than 0.5 is not relevant since that would put the bias in the wrong direction. It can be seen that the curve is not very smooth. The reason for this is simply that all probability values are not equally common. It is for example the case that no reports are assigned a priority of being good less than 10%.

One might argue that this curve does not provide the necessary information about the classifier and that removing reports randomly would produce a similar curve. Since the number of bad reports already is higher than the number of good reports even a random classifier would remove more bad reports than good reports. This is indeed true but it should be noted that the curve presented shows the percentage of reports removed, not the absolute numbers. If reports were removed randomly the percentage of good reports removed and the percentage of bad reports removed would be the same. The result presented here removes in total 17% of all reports in the system.[19] A random classifier would not make any difference between good and bad reports, hence remove 17% of both categories.

If focus instead is turned to absolute numbers the objection is valid. A curve which shows the number of reports removed will change depending on the distribution between the types of reports. Such a curve can still be interesting and one is shown in Figure 9. The curve is very similar to the one previously presented. The difference is that the area between the two curves is larger.

19 Out of the bad reports, which consist of 61% of all reports, 22% is removed. Out of the good reports, which consist of 39% of all reports, 8% is removed. Put together this means that 61% • 22% + 39% • 8% = 17% of all reports are removed.

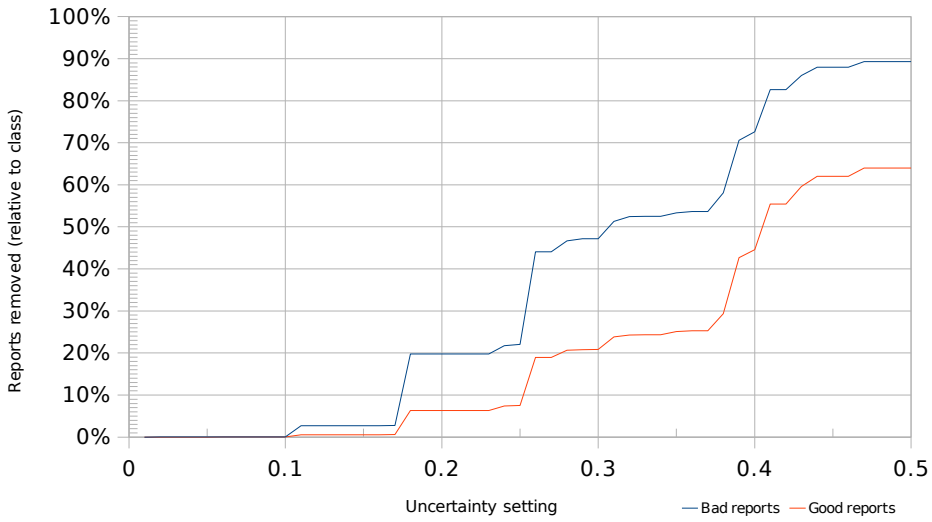

Figure 8: Diagram showing the amount of reports removed relative to the number of reports in each class.

Another way of visualizing the result is to plot a chart showing the correlation between precision and recall. This curve, seen in Figure 10, shows the trade-off between the number of examples matched and the correctness of the matches. The idea is that if only one report is to be removed the classifier can take the one it is most certain about, hence have a high correctness, but it would only do little, hence have a low number of matched examples. Correctness is often referred to as precision and the number of matched examples as recall. Precision is, in this case, the number of removed bad reports divided by the total number of removed

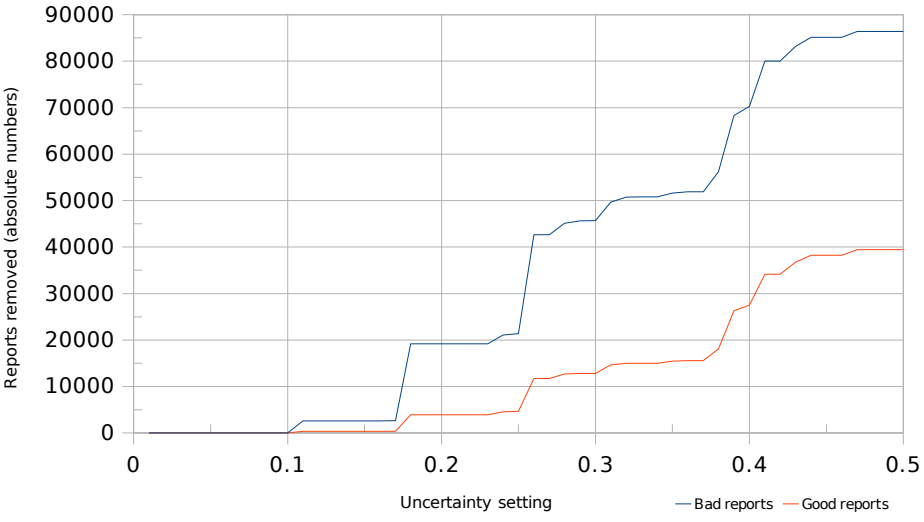

Figure 9: Diagram showing the amount of reports removed.

reports. Recall is the number of removed bad reports divided by the total number of bad reports. By changing the uncertainty settings a curve can be plotted that shows these two values against each other.

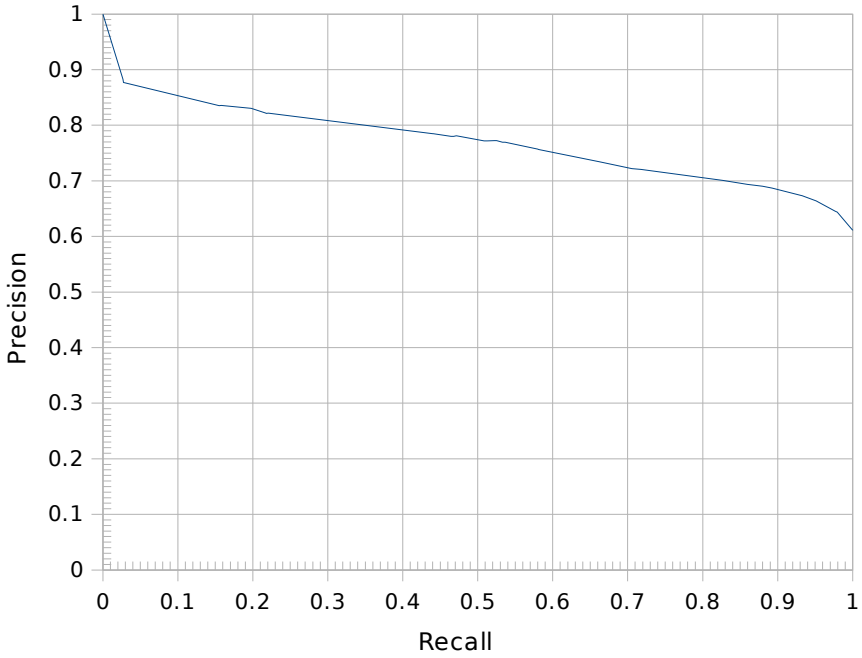

Figure 10: Precision-recall curve.

With the chosen uncertainty setting the precision is 81% and the recall is 22%. The recall is a familiar number and it is simply the percentage bad reports removed which has been listed several times already. The precision is slightly harder to calculate. It is the number of bad reports removed divided by the total number of removed reports. The amount of bad reports removed is 61% • 22% of all reports. The amount of reports removed is 61% • 22% + 39% • 8%. By dividing these numbers the result becomes 81%. Precision can be used to measure how useful this classifier is. If reports would just have been removed randomly 61% of the removed reports would be bad, since that is the class distribution in data. By using this classifier the number is instead 81%, which shows an improvement. It is also possible to look at it from the other way around. When removing reports randomly 39% of the removed reports are good. With this classifier the number changes to 19%.

7.2. External Factors Affecting the Result

The result presented above is not as good as expected. There are two external reasons which might have affected the result. The first of these are lack of data. As written earlier, the data that is distributed in shards have not been available. This means that a lot of metadata, for example the number of friends a user has, how

many times a user has been banned and how many unique guestbooks a user has written in, have been unavailable. It is impossible to know how important this metadata is without looking at it hence it is unknown if the presence of it would change the end result or not. Except for the metadata, the absence of data is worth mentioning. Almost all reportable objects, such as guestbook entries, blog posts, user presentation and so on, are sharded. This is a clear drawback since, strictly speaking, the quality of a report has to do with the possible violations of the rules, and weather a violation has taken place or not can been seen in the reported object. In other words, if a guestbook entry has been reported by a user saying it contains bad language, the guestbook entry itself is the only thing that should be needed to determine if the entry contains bad language or not.

The second external factor that might have affected the result is the noise in the available data. Noise in this sense refers to reports that have been misclassified in the past. Customer Service Representatives handling abuse reports have been setting the report quality since start, but the flag has just been ignored until now and no follow-ups have been done. It is likely that some incorrect classifications have been done, but it has not been feasible to do anything else but blindly trust the quality level that has been set in the database. First of all it is hard to find someone who could reclassify reports and, second, a lot of data was missing. It would also be impossible to manually recheck all but a few reports due to the enormous amount of reports available. However, in some special cases the quality of reports can be known. One example of such a case is when it comes to reports about RealCelebs. It is safe to assume that celebrities specially invited to the site have not committed any violations of the rules; hence 0% of all reports against RealCelebs should be classified as good. In reality this is not the case and 13% of all reports against RealCelebs are classified as good, which shows that quite a few reports have slipped through. One of these reports is stating that Avril Lavigne is selling drugs via Stardoll, which is obviously not true. Another report that is clearly bad is saying that Ashley Olsen is phishing. It is worth noting that the mistakes covered by this simple rule only go one way. Out of a set of reports that are know to be bad, 13% are classified as good. It is also likely that reports that are indeed bad are classified as good.

Another problem is that different Customer Service Representatives might classify uncertain reports in different ways. Since there is no way of telling the difference between the best report ever written and a report that barely passed the test by looking at a reports afterwards, this might be a problem. There is, furthermore, no way of telling which Customer Service Representative that handled a certain report, so it is not possible to filter reports based on that.

The problem of noisy data is not in any way unique to this project. It was for example a problem in the image analysis software [BAS+98] discussed earlier. Different geologists had different options if a certain artifact on the image was in

fact a volcano or not. The solution employed in that study was to let different geologists classify the same image and they also used a notion of certainty. The geologists did not only determine if a certain artifact was a volcano or not but they also assigned the artifact one of four labels depending on how certain it was that the artifact was a volcano. Neither multiple classifications nor certainty labeling could be done in this study. The largest obstacle being that the reported objects were unavailable. In order to get a second opinion from a domain expert all information must be available.

7.3. Interviews

Three interviews were performed with different employees on Stardoll in order to get reactions on this thesis work, their thoughts about an automated system in general and their view on the result presented. Representatives from three different departments were interviewed in order to obtain several perspectives and hence approach the issue from different angles.

7.3.1. Customer Service

Margareta Petersson, Head of Customer Service, and Johannes Schildt, Customer Service Representative, have been present from the beginning of this thesis work and they have been the primary contacts at the Customer Service. Much of their reasoning have already been incorporated in parts of this report, but during this interview they were given the opportunity to express their view on the work as a whole and, especially, the result.

Before this master thesis was initiated the outline of the project had been discussed with Petersson. No details were specifically set in this initial phase but it was given that the work should concern the employment of automated methods in the process of handling abuse reports. Once the work was started a meeting was held with a majority of all Customer Service Representatives, during which a few leads about possible automated methods were discussed. No concrete decisions were made during this meeting and most ideas about how the work could be simplified focused on small changes in the current systems, such as bug fixes. In retrospective it is easy to see that it would have been beneficial to conduct more meetings during the process, especially after the main focus area of the project had been decided upon. This would have made it easier for the representatives to contribute with ideas within the frames at hand.

The Customer Service, and especially Schildt, did at a later point make an attempt to change the focus to prioritize reports. The rationale for this was that a filtering system is only useful if a substantial amount of bad reports are written, while priorities are applicable even in a system with mostly good reports. Schildt believes that future changes in the reporting procedure will make users more prone to write correct reports, which will lower the amount of bad ones. The changes in

question will be discussed more in Section 8.4.3 on page 96. The reason the work was not changed was that much effort had already been put into an automated filtering system and changing the focus would possibly render this work unusable. Additionally a system that assigns priorities comes with other problems, as seen in Section 3.3.1 on page 24.

Petersson and Schildt are not pleased with the result presented since they were expecting both better figures and a more practical result, preferably earlier during the process. They can not give an exact figure for an acceptable trade-off between a removal rate for good reports and a removal rate for bad reports but it is clear that the one provided here is unacceptable. A small loss is acceptable if the win from such a system would be large enough, but in this case the number of bad reports removed is too low to justify the number of good reports removed.

One reason that some reports are classified incorrectly is believed to be because most people working with the reports are extra workers working from home. They are informed and updated about the latest news by their manager but it is hard for them to always be updated on what is happening and how this affects their work since they are not in the office where it all happens. The vast amount of reports can also be difficult to handle since each report may require a fair amount of focus and effort.

7.3.2. Management

The Chief Executive Officer, Mattias Miksche, has been interviewed in order to get reflections from a management perspective. Miksche has been aware of the thesis work during the process and stresses that this is a very important area for Stardoll. A more effective handling of abuse reports has two potential benefits. The first, but not the most important, benefit is the reduced costs. The current manual handling process is costly but, according to Miksche, the cost seen today is acceptable. The second, and far more important factor, is that of responsibility. Over seventeen million young users have signed up for an account on Stardoll and the environment must be safe. It is vital that the necessary actions are taken, and that they are taken in a timely fashion, once a violation has happened. Because of this responsibility Miksche prefers a manual system where all reports written will be reviewed by a Customer Service Representative.

In spite of this, Miksche is very positive to automated methods in general and suggests that such a system should have been put in place earlier. The main focus on these systems should be detection rather than filtering. One example mentioned is an automated system that would detect if a user is trying to make other users give away personal information about themselves, even if the messages are never reported.

The result presented is not good enough to make the system applicable and the method in itself is problematic. The trade-off between removing good reports and

removing bad reports is hard to make since it very much depends on which reports are removed. One single incorrect classified report could potentially, if the report in question was the most important ever filed, have disastrous consequences. Miksche is also concerned about the mistakes made in the current process and says that it would be desirable if this problem could be mitigated.

It is unfortunate that data has been missing during the work. Miksche believes that most patterns are available in the part of the data that has been missing and thinks that the result could have been improved if all data could have been made available for processing.

7.3.3. Development

The person who came up with the idea for this thesis work was Mikael Krantz, System Developer. The reason is that there has always been an assumption that some work could be done in this field. The reason that reports are classified as good or bad when they are handled by the Customer Service is mainly because of the prospective of doing some automated processing based on those flags. The reason to why this work was done in the form of a master thesis is that there is a risk involved since it is hard to know how large the benefits, if any, would turn out to be. This risk together with the extent of the work made it hard to fit into the normal plan. Announcing the work as a master thesis project seemed like a good way to avoid allocating own resources on a potentially unfruitful project but still investigate the matter.

The reason all data could not be used was because it could not be copied from the live database. The reason for that was twofold. First, the live database servers were already dealing with a very heavy load. Copying large amounts of data from them would stress the servers even further and this is unfortunate as it could cause longer response times for the users. Second, the amount of data needed to be copied is very large. The sharding scheme used separates content based on the user id of the user related to the content in question. Different databases are used for different ranges of user id numbers. Content related to the first million users are saved in one database, content related to the second million are saved in another database, and so on. In total there are eighteen shards since the number of users is a bit above seventeen millions now. The average size of one of these shards is approximately 40 Gb, meaning that the total amount of data available is very large, hence difficult to transfer to another environment and hard to handle in general. Krantz thinks that it would be easier to copy just one of the shards since that would reduce the amount and make the data more manageable.

The result presented is reasonable given the circumstances and not surprising. It is unfortunate that the system did not perform better but the risks involved were known beforehand. The system can not be used as it is, but it might be possible to use it in alternative ways. One option is to use the system to flag reports but still

handle all reports manually. It would then be possible to observe which type of reports that are incorrectly classified as bad. Perhaps this would give some insight into how the filtering process could be enhanced. Another alternative is to use the output from the classifier, which is the estimated probability that a report is good, as a priority measurement. It was seen in Section 3.3.1 on page 24 that priority and quality is not necessarily the same, but the correlation might be strong enough to still make this modified system usable. The idea would be to sort the reports in the administrative interface according to priority and show the most prioritized reports first.

Krantz says that one reason for the noise in the data might be because the quality flag has never been used before and no follow-ups have been conducted. It is possible that the flag would be set with more care if the representatives knew that it is actually used and that it is helpful. If this is the case it would be possible to inform about the system now and then retrain the classifier after a while and only base it on new, hopefully less noisy, data.

8. Result and Analysis

This section, which is the last, will present the result from a more general level and analyze it. The previous section evaluated the classifier itself, but the result must also be considered in context. The aim of this thesis was to investigate how an automated method could improve efficiency on a large website. This section also discusses possible starting points for future work in this field.

8.1. Conclusions

The system constructed could with little effort be put in place in order to filter reports. A classifier has been trained and a tool that can make it practical to use it has been developed. The system is in that sense complete and ready to be used.

The available options should be compared when analyzing the system to determine if it should be used or not. The options available are to either put the system in place or to use the manual system. Three different aspects can be considered when doing the comparison.

Accuracy. The classifier's performance is lower than expected, which suggests that the system should not be used. On the other hand it is important to keep in mind that the process used today has drawbacks too. First of all the classifications made are not perfect and mistakes are done. The exact error rate is unfortunately hard to establish but the issue was discussed in Section 7.2 on page 85 where it was seen that the manual process, at least under the circumstances discussed there, had an error rate as high as 13%. Determining the corresponding number for the classifier presented here is easier. Out of the reports removed by the classifier 81% are expected to be bad. This means that 19% of the removed reports are incorrectly removed. The figures are closer than one might think at first.

Economics. Using an automated system could potentially save money. The classifier presented here would remove 17% of all reports. If 20000 reports are filed each week this would equal 3400 less reports to handle each week. Given that the average time needed to handle a report is 40 seconds approximately 38 working hours per week could be saved.

Response time. Response time can be shortened by employing an automated system. Shorter response times can help make Stardoll safer since the appropriate

action can be taken faster when rules are violated. A safer site is desirable from several perspectives.

From a strictly economical perspective there is a trade-off between acting fast and acting correctly. Both approaches risk dissatisfying users which have economical drawbacks if the dissatisfaction means that the users are spending less time on the site or decides to leave it altogether. If speed is shortened at the expense of accuracy, which is the case if the automated method is used, user can get dissatisfied since their reports are ignored. If instead accuracy is prioritized the abuses are left visible on the site for extended periods of time which can cause users to feel that the site is unsafe and unfriendly since inappropriate content is visible. The trade-off between these aspects does not appear to be well studied and it is hard to make any claims about what role it plays in the case at hand.

This thesis work aimed at investigating the employment of an automated method on a large website. The most interesting question to consider is if efficiency can be improved and it appears that improvements indeed can be achieved. The numbers shown above are promising and indicate that time can be saved and resources freed by using automated methods. The problem is that a suitable accuracy has to be found. Even if an automated method increases performance it might not be feasible to put it in place if the accuracy presented by the system is too low. This indicates that systems that complements rather than replaces current processes are extra promising. An automated detection system that would detect abuses before they are filed is an example of such a system. Mistakes by that type of system are not as severe as the ones made by the filtering system and other similar systems.

The classifier presented here is not performing as well as expected. Considering the problems encountered along the way the result is acceptable, but not good enough to make the system usable in practice. Neither the Customer Service nor the Management think that the result showed is acceptable for a usable filtering system. This is unfortunate but the primary aim of this work was to investigate the issue, which has been done.

8.2. Contributions

Several contributions have been made and this sections aims at listing them. The most apparent contribution is that this thesis work has created a system that could be practically used without much work required. Unfortunately the performance might make it inappropriate to use, but parts of the system could be reusable for example if the classifier is retrained at some later point in time. Perhaps it would be possible to obtain less noisy data, for example by letting several Customer Service Representatives collaborate on classifying some of the reports. Using this less noisy data to build a new system would be easier since the work can be based on the work already performed.

Future work in the machine learning field at Stardoll, and possibly elsewhere, can learn from this work. The lessons learned can hopefully make it easier to avoid similar problems in the future. It is also possible that observations made here, for example that the differences between different subsets have been fairly small, can lead to more efficient work.

During this work it has been apparent that automatic filtering, especially in this noisy environment, has serious drawbacks. This work can hopefully make these drawbacks visible for people about to do work in this area. All such people are recommended to read about the interviews performed and also look at the pointers about future research that are available in this section.

StarClassifier has been developed and released under a free software license. The target group for StarClassifier is fairly small and it can only be used for one particular purpose. The tool is, nonetheless, available for anybody who finds the need for it.

8.3. Lessons Learned

Many lessons have been learned during this work. Some of them are extra important and people who later on are working in this field might find them helpful.

Data is difficult to access. There are a lot of practical issues that needs to be solved when working with large amounts of data. Just because data is located in a database on a nearby host it is not necessarily ready to be used. Even if it is not possible to get access to all data it is helpful to know which data that will be available and when.

One lesson learned is that it would have been a good idea to have more meetings with the potential users of the system, in this case primarily the Customer Service. These concepts tend to get fairly abstract and it is easy to misinterpret each other. Having regular status meetings where the progress is discussed is probably a good way to better understand each other. The related lessons that have been learned in software development appear to be applicable here too and getting inspired by agile methods [BA04] might help.

It has been pointed out before that it is often the case that numerous small tools must be developed during a machine learning process [BAS+98]. The reason for building these tools is that there is no easily accessible tool that can do the tasks at hand. If it would be possible to make a platform for doing these operations it would help a lot. At the same time the problem is difficult to solve since each problem domain requires its own set of tools.

8.4. Future Research

The work presented in this report can hopefully function as a basis for future work in this area on Stardoll as well as on other communities. Before doing any future work it is worth considering the perspective from which the issue is approached. The automatic filtering process described here is only one of several ways improvements can be achieved.

The following subsections will discuss possibilities for future work from three different perspectives.

8.4.1. Improved Automatic Filtering

The result presented in this thesis might not look very promising. Several methods have been tested and some enhancement techniques have been tried in order to refined the result, but without any improvements. Based on this it might be easy to argue that automatic filtering looks like a dead-end and that future work is unlikely to result in a usable system. This might be the case but it is also possible that a project with more resources would be able to produce a significantly better result. Some of the steps that were initially planned to be performed had to be skipped due to lack of data or lack of computing resources. The lack of data made it impossible to calculate some of the attributes, and time could have been saved and used for other tasks if more computing resources had been available. The general method used would not need to be changed significantly if all data had been available. The new metadata attributes, such as how many friends a user has, could be treated in the same way metadata has been treated in the currently available dataset. Some challenge is presented considering the actual objects. Attributes concerning those would need to form a new category in the attribute list since the attribute list used in this project only had categories for the reported user, the reporting user and the report. Extra care should be taken when handling attributes related to the actual objects since it is reasonable to believe much information is contained in them and because the current methods would not be applicable. The objects contain text and this text must be handled in some way. One lead is to investigate spam filters, for example, Baysian filtering. It might also be worthwhile investigating if Baysian filtering could be used on a more general level, by for example comparing all content produced by different users and see if a pattern emerges showing abusive users.

In some experiments different subsets of the data has been processed independently but with the same method. During these experiments it was seen that the result is fairly similar in different subsets. This leads to the assumption that it would be beneficial to focus on increasing the number of attributes rather than the number of examples. It has been, see Section 7.3.3 on page 89, said that it might be possible to copy one shard from the live database and use that for future work in this field. This would be welcomed but some issues are worth considering.

If focus is only on the reported object the number of reports that could be used would be approximately 1/18 of all reports, assuming that the reports are distributed equally between the eighteen shards. If, on the other hand, the general behavior of the users are to be to taken into account the data is needed for both the reported user and the reporting user. If reports are filed equally often between all sets of users the amount of reports that could be used would decrease to 1/18 • 1/18 = 1/324 of all reports filed, which is rather small. If historical data would be used this number would be different since users in the first shard have simply had more time to file reports and write objects that could be reported. Except for the problem of selecting a shard with enough data it is important to consider that the data will be biased. In the subsets used in the experiments performed during this project the reports have been randomly selected. In the case one single shard is used only one type of users will be taken into account. If, for example, the first shard is used the reports are either old or written by and against experienced users. Old reports may differ from the average report since, among other things, the interface for filing a report has changed. It appears, for example, that users more often misinterpreted the reporting system for an instant messaging function in the beginning. Report written by or against experienced users may also differ in one or another way. It was seen that experienced users tend to write better report than inexperienced users (see Appendix B).

8.4.2. Other Automated Methods

The idea of shortening the response time by ordering reports according to estimated priority has been present from the beginning. This has both some appealing benefits, such as reduced risks of harming the process, and some complicating characteristics, such as less time saved since all reports still have to be handled. Both these aspects have been discussed in this report and will not be touched more upon in this section. It is probably possible to use a similar method like the one used here. The fact that the flag considered here represented the quality has not affected the choice of method. It would be possible to rerun all steps but instead use a flag representing the priority. The problem with this is that no priority flags are currently available. If it is desirable to follow this road later on it is a good idea to make the necessary modifications to the administrative interface as soon as possible in order to collect enough data. The interface should make it possible to assign, in addition to the report quality, report priority. In order to make it easy for Customer Service Representatives to set this priority the number of options should probably be kept to a minimum. It might even be enough to just have two priority levels: high and low. Given two priority levels and two quality levels the total number of options to select would be four, but the number of options could actually be reduced even more since no bad report could have a high priority. It does not make sense to handle a certain type of bad reports before

any other bad reports. The total number of options would be three (good quality – high priority, good quality – low priority, bad quality).[20] Before the changes are made in the administrative interface it is vital to have clear and precise guidelines for the difference between high and low priority. Failing to set up these guidelines or to implement them properly could severely damage the quality of the priority flag.

One method that has started to look very promising lately is that of automatic detection of specific abusive behavior. An example that has been discussed in this report is to look for chain mails (see Section 3.3.3 on page 26). Other abuses could also be found in the same way and the system could be built in a modular fashion where detection modules for different types of abuses could be added. The result from all these modules could later be shown in a separate view in the administrative interface. The Customer Service Representatives would then have to go through both the reports and the detections made by this system. The possible win in such a system would be that some abuses could be detected even before any report is filed and hopefully abusive users could be noticed and warned before they have the opportunity to commit more abuses. At the same time the potential problems with such system are limited. Falsely detecting an action which is not an abuse would indeed cause some more work for the Customer Service but following up on such a case is not likely to take long. Missing actions that should have been flagged as an abuse is unfortunate but not very severe since there is still a possibility that the abuse will be reported by a user the same way it is today. A further benefit of such a system is that the different modules can work rather independently. It is possibly to fail building a good chain mail detector but still make a usable phishing detector. The largest downside of a system like this is that it is very sensitive to lack of data. Practically none of the work outlined above would have been possible to perform given the data that was given to this project. The system would also need a large amount of data at run-time. In order to make the most out of a system like this it would be necessary to feed it with all text object (e.g., guestbook entries, blog post and private messages) produced by users.

8.4.3. Other Approaches

Automated methods are not the only way response times can be shortened. The Customer Service has for a long time suggested that the interface used when filing a report should be changed. They believe that less bad reports would be filed if it was clearer to the users how a report should be filed and that filing illegitimate reports is a rule violation. Extra steps should be added in the process and the user will get to review their report and verify that they have actually selected the right category and, most importantly, the right object. The Customer Service had a similar request for a change in the process of handling messages about other issues

20 This is in fact the same number of options available today. Recall that there is currently a quality level named “nonsense” which has become interchangeable with “bad”.

than abuses, such as payment problems. This change was made and the number of messages received dropped to almost half. The change in quality is harder to measure since this type of messages are not classified as good or bad, but according to Customer Service Representatives the quality was raised most notably. These two observations suggest that changes in a contact procedure can help minimize the amount of bad content created, which makes this look like a promising solution. One thing to keep in mind is that the number of messages not concerning abuses fluctuates more than the number of reports about abuses. This makes it hard to determine how much of the reduction of messages that was caused by the change in the interface and how much of it was just a natural fluctuation. Nonetheless the method looks promising and a decision has been made that a change should be made to the interface for filing abuse reports, but it is not decided when it should be implemented. It is likely that such a change will affect the users' behavior and it is therefore recommended that any future work regarding automatic processing of the abuse reports are postponed until this change has been made and evaluated. Except for the possible change in behavior this change might also affect how valuable certain automatic methods are. If the number of bad reports is severely reduced, as suggested, an automated filtering system would not be as useful as it appears today. If, on the other hand, the new reporting interface would be seemed as an obstacle by the users causing them not to file reports, an automatic detection system would appear even more appealing.

9. References

[And81] Jonathan Anderson (1981). *Analysing the Readability of English and Non-English Texts in the Classroom with Lix*. In: Paper presented at the Annual Meeting of the Australian Reading Association (Darwin, Australia, August 1981). Available at <http://www.eric.ed.gov/ERICWebPortal/custom/portlets/recordDetails/detailmini.jsp?_nfpb=true&_&ERICExtSearch_SearchValue_0=ED207022&ERICExtSearch_SearchType_0=no&accno=ED207022> (last accessed 2008-02-14).

[BA97] Stefanie Brüninghaus and Kevin D. Ashley (1997). *Using Machine Learning for Assigning Indices to Textual Cases*. Available at <http://citeseer.comp.nus.edu.sg/18045.html> (last accessed 2008-02-22).

[BA04] Kent Beck and Cynthia Andres (2004). *Extreme Programming Explained*. 2 edition. Addison Wesley. ISBN 0321278658.

[BAS+98] Michael C. Burl, Lars Asker, Padhraic Smyth, Usama Fayyad, Pietro Perona, Larry Crumpler and Jayne Aubele (1998). *Leaning to Recognize Volcanoes on Venus*. In: Machine Learning. Vol 30. Available at <http://citeseer.comp.nus.edu. sg/131412.html> (last accessed 2008-02-22).

[Beb08] Bebo (2008). *Contact Us* [www]. Available at <http://www.bebo.com/ContactUs.jsp> (last accessed 2008-02-14).

[Bre94] Leo Breiman (1994). *Bagging Predictors*. In: Machine Learning. Vol 24. Available av <http://citeseer.ist.psu.edu/breiman96bagging.html> (last accessed 2008-05-18).

[DH06] Ilan Dar-Nimrod and Steven J. Heine (2006). *Exposure to Scientific Theories Affects Women's Math Performance*. In: Science. Vol 314. Available at <http://www.ncsu.edu/awf/WomenMathStereotypes.pdf> (last accessed 2008-02-14).

[Fac08a] Facebook (2008*). Statistics* [www]. Available at <http://www.facebook.com/press/ info.php?statistics> (last accessed 2008-03-13).

[Fac08b] Faceparty (2008). *Help + Support* [www]. Available at <http://www.faceparty.com/help/index.aspx> (last accessed 2008-02-14).

[FP97] Tom Fawcett and Foster J. Provost (1997). *Adaptive Fraud Detection*. In: Data Mining and Knowledge Discovery. Vol 1 (3). Available at <http://citeseer.ist.psu.edu/fawcett97adaptive.html> (last accessed 2008-02-25).

[Fre07] Free Software Foundation (2007). *GNU General Public License*. Available at <http://www.gnu.org/licenses/gpl.html> (last accessed 2008-06-04).

[Gra02] Paul Graham (2002). *A Plan for Spam*. In: Graham, Paul (2004). Hackers & Painters: Big Ideas From The Computer Age. O'Reilly. ISBN 0596006624. Available at <http://paulgraham.com/spam.html> (last accessed 2008-02-14).

[GWF02] Frank R. Giordano, Maurice D. Weir, and William P. Fox (2002). *A First Course In Mathematical Modeling*. 3 edition. Brooks Cole. ISBN 0534384285.

[Kel07] Packy Kelley (2007). *The 2007 AO 100 Top Companies* [www]. Available at <http://alwayson.goingon.com/permalink/post/15899> (last accessed 2008-02-14).

[LHK98] Mario Lenz, André Hübner and Mirjam Kunze (1998). *Question Answering with Textual CBR*. In: Lecture Notes in Computer Science. Vol 1495. Available at <http://citeseer.comp.nus.edu.sg/147319.html> (last accessed 2008-02-22).

[LS95] Pat Langley and Herbert A. Simon (1995). *Applications of Machine Learning and Rule Induction*. In: Communications of the ACM. Vol 38 (11). Available at <http://citeseer.ist.psu.edu/langley95applications.html> (last accessed 2008-02-23).

[Oza06] Nikunj C. Oza (2006). *Ensemble Data Mining Methods*. In: Encyclopedia of Data Warehousing and Mining. Available at <http://ti.arc.nasa.gov/people/oza/publications/files/oza06.pdf> (last accessed 2008-05-18).

[PNM+98] Lawrence Page, Sergey Brin, Rajeev Motwani and Terry Winograd (1998). *The PageRank Citation Ranking: Bringing Order to the Web*. Available at <http://citeseer.ist.psu.edu/page98pagerank.html> (last accessed 2008-02-25).

[SKS05] Abraham Silberschatz, Henry F. Korth and S. Sudarshan (2005). *Database System Concepts*. 5 edition. McGraw-Hill Higher Education. ISBN 007124476X.

[Tan01] Andrew S. Tanenbaum (2001). *Modern Operating Systems*. 2 edition. Pearson Education. ISBN 0130926418.

[TKS06] Domonkos Tikk, Zsolt T. Kardkovács and Ferenc P. Szidarovszky (2006). *Voting with a Parameterized Veto Strategy: Solving the KDD Cup 2006 Problem by means of a Classifier Committee*. Available at <http://www.sigkdd.org/explorations/issues/8-2-2006-12/7-tikk-Winner-task2.pdf> (last accessed 2008-05-18).

[Web07] Webware staff (2007). *Webware 100 Award Winner* [www]. Available at <http://www.webware.com/8301-13546_109-9729504-29.html> (last accessed 2008-02-14).

[WF05] Ian H. Witten and Eibe Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. 2 edition. Morgan Kaufmann. ISBN 0120884070.

[Wik07] Wikipedia contributors (2007). *List of social networking websites* [www]. Wikipedia, The Free Encyclopedia. Version 155012280. Available at <http://en.wikipedia.org/w/index.php?title=List_of_social_networking_websites&oldid=155012280> (last accessed 2008-02-14).

[Uni98] United States Code (1998). *Children's Online Privacy Protection Act of 1998*, 15 U.S.C §6501, et seq. Available at <http://www.ftc.gov/ogc/coppa1.htm> (last accessed 2008-02-14).

[Xan08] Xanga (2008). *Xanga Help* [www]. Available at <http://help.xanga.com/policeofficer.htm> (last accessed 2008-02-14).

Appendix A: StarClassifier

StarClassifier is a light-weight software tool that can perform classifications on a set of examples in a database and write the result back to the database in question. The classifications made are based on a decision tree that needs to be presented in textual form to the tool. StarClassifier is built in Java and has no dependencies in itself, but in order for it to function properly the appropriate database driver must be available. The tool is not specific to Stardoll and can be used for all tasks where examples need to be classified according to a decision tree.

The result that StarClassifier writes back to the database is not the estimated class, but rather the probability of the example belonging to a pre-specified class. In the case of determining if an abuse report is good or bad it is for example possible to configure the system to output the probability of the report being good. The results, which will range from 0 to 1, can then be used in order to take appropriate actions.

Downloading and Compiling

StarClassifier is released under the GNU General Public License (GPL) and can be downloaded without charge.[21] The tool can either be downloaded in the form of an already compiled JAR file or as a ZIP file containing all the source files. In the first case the tool is ready to be used directly, but in the latter case the user has to first compile the files and create a JAR file. One simplest method for doing so is to utilize the build file distributed together with the source. This option requires that the build tool Apache Ant[22] is installed on the system. If it is, StarClassifier can be built and bundled together as a JAR file by using the target "jar" as seen in the following example. The JAR file, named `StarClassifier.jar`, will be available in the folder "jar" after the execution has finished.

```
unzip StarClassifier-src.zip
ant jar
```

21 The needed files are available at <http://starclassifier.udd.be/>.
22 Found at <http://ant.apache.org/>.

Decision Tree File

One of the inputs needed by StarClassifier is a decision tree in Weka's plain text format. One easy way to produce such tree is to run Weka Explorer and copy the tree in question from the result buffer. Please note that the format used here is not the same as Weka's binary format for saving models. The reason that the plain text format is used is that it is easier for humans to read and, if necessary, manually tweak.

In order to see how the tree file can look like and how to produce it the decision tree classifying pizzas presented in Section 2.4.1 on page 13 will be revisited. What was not discussed in that section was how input data used to create the tree could look like, but for completeness of this guide this will be touched upon now. Let us assume that the input data looks like the one in Table 26. One might notice that this data do not completely follow the rules specified in the tree previously. It can for example be seen that one pizza is tasty even though it is not warm. The reasons for these deviations are twofold. First, real data are seldom perfect and noise is likely to occur. Second, StarClassifier's ability to output a probability instead of a class is best seen if the data can not be perfectly classified. If the only outputs would be 0 (i.e., 0%) and 1 (i.e., 100%) the program would not be too different from one that just outputted the class, which in this case would be "TASTY" or "NOT TASTY".

The data presented in the table should be opened in Weka Explorer. Once it has been loaded the id attribute should be removed in order to prevent Weka from finding coincidental patterns. It is for example possible to split the data on this attribute and say that pizzas with an id value larger than 10 is tasty and all other are not tasty. This would in fact only classify one instance incorrectly and therefore look promising to an algorithm, but a human would quickly see that this pattern is of no value since the id attribute is only available as a help to keep track of the example, they do not tell us anything about the examples in themselves. Once the id attribute has been removed the J48 tree can be created. The result buffer will contain much information, including the actual tree. This tree is what StarClassifier needs so it has to be saved in a separate file, which in the following will be assumed to be named `pizza.tree`. The content of this file should be like the following.

```
curry = YES: NOT TASTY (3.0)
curry = NO
|   warm = NO: NOT TASTY (4.0/1.0)
|   warm = YES: TASTY (13.0/3.0)
```

The plain text format should be fairly straight forward to read and understand. It can be compared to the tree in Figure 2 on page 14. The major difference

between this format and the graphical format used earlier is that the text format contains some information about how good the rules classified the training data. The first number within the parentheses, or the only one if there is only one number present between them, tells how many examples that applies to that rule. It is for example four pizzas that neither have curry nor are warm. If a second number is present within the parentheses it shows how many examples that are incorrectly classified. One pizza in the training data, the one with id number 7, was cold but still tasty. This caused one incorrectly classified example; hence the second argument is 1 for the leaf in question.

Table 26: Example data for the pizza example.

id	curry	warm	class
1	YES	NO	NOT TASTY
2	YES	YES	NOT TASTY
3	YES	YES	NOT TASTY
4	NO	NO	NOT TASTY
5	NO	NO	NOT TASTY
6	NO	NO	NOT TASTY
7	NO	NO	TASTY
8	NO	YES	NOT TASTY
9	NO	YES	NOT TASTY
10	NO	YES	NOT TASTY
11	NO	YES	TASTY
12	NO	YES	TASTY
13	NO	YES	TASTY
14	NO	YES	TASTY
15	NO	YES	TASTY
16	NO	YES	TASTY
17	NO	YES	TASTY
18	NO	YES	TASTY
19	NO	YES	TASTY
20	NO	YES	TASTY

Configuration File

Before StarClassifier can be run it has to have, in addition to a decision tree as described in the previous section, a configuration file describing how database interaction will work. This file mainly specifies how examples are fetched from

the database and how the result is written back, but since some other information is needed to facilitate the database interaction the file has in the end five attributes. These attributes all start with an at sign (@) and all lines that do not start with such a sign, including empty lines, are ignored. They can for example be used as comments. The five attributes will be described in turn below.

@driver. The database driver that should be loaded. For MySQL this would normally be `com.mysql.jdbc.Driver`. Also note that this driver must be made available to StarClassifier by including the appropriate entry in the classpath variable.

@url. The JDBC URL that is used to connect to the database. This would normally include the host on which the database server is running, the database name, the user name and finally the password for that user.

@id. Examples need to be identified in some way and StarClassifier assumes this is done by using a unique id attribute. This is the attribute used to link together example loaded with attributes saved. The name of this attribute, which does not have to be id, is configured with this option.

@load_sql. This option specifies the SQL statement that should be used when loading examples. The attributes that should be fetched should be whatever arguments that are needed in the tree and the id attribute specified above. It is valid to select all attributes (i.e., `SELECT * FROM SomeData`) since unused arguments do not interfere with the process in any way, but since all loaded data is read into memory this can take up unnecessary amounts of space. It is possible to include a WHERE clause and use that to limit the number of examples that are read.

@save_sql. Once the result has been calculated it has to be saved in the database and this attribute specifies the SQL statement to use for that. This statement should contain two question marks (?) which will be replaced with data before the statement is run. The first question mark will be replaced by the probability and the second with the id argument.

Exactly how the configuration file will look depends on the specific needs, but one example is listed below. This file will in the following be referred to as `pizza.conf`. It is assumed that the database is named `CLASSIFICATIONS`, available on the machine `sam` and that it can be accessed with user name `spade` and password `trace`. The database system has been assumed to be MySQL and the standard JDBC Driver[23] is used. The id attribute's name is “id” and the other attributes are named as they have been named previously (i.e., “curry” and “warm”). The same table, `UnclassifiedPizzas`, is used to both load data from and save data to. The new attribute “probability” has been added to the table and this is where the result is to be written.

23 Found at <http://dev.mysql.com/downloads/connector/j/>.

```
@driver com.mysql.jdbc.Driver
@url jdbc:mysql://sam/CLASSIFICATIONS?
  user=spade&password=trace
@id id
@load_sql SELECT id, curry, warm FROM UnclassifiedPizzas;
@save_sql UPDATE UnclassifiedPizzas SET probability = ? WHERE
  id = ?;
```

In some cases it might not be appropriate to use the same table for both these purposes and then a new table can be created and INSERT statements can be used to save the result. Assume for example that a new table named `PizzaClassifications` should be used. This table should have one id attribute linking pizzas in one table together with pizzas in the other table and one probability attribute that will hold the actual result. In this scenario the @save_sql option from above should be changed to the following.

```
@save_sql INSERT INTO PizzaClassifications(probability, id)
  VALUES(?,?);
```

The probability attribute will be updated with a value between 0 and 1 which means that this attribute should be able to handle floating point numbers. If this is inappropriate for one or another reason an IF statement can be used to assigned a class at this point. This means that the uncertainty setting has to be decided on before classification can be made and the result has to be incorporated in the configuration file. For simplicity it has been assumed that classification errors are equally severe both ways meaning that the split value used is 0.5. The following option assign to the attribute class either the string URK or the string YUMMY.

```
@save_sql INSERT INTO PizzaClassifications2(class, id)
  VALUES(IF(? < 0.5, "URK", "YUMMY"), ?);
```

Running

StarClassifier needs three arguments to run. The decision tree and the configuration file, which are the major components, have already been discussed. The remaining argument is a specification of the class the probability should be calculated from. In the case of Stardoll's abuse reports the focus has been on how probable it is that a report is good, but there is nothing saying that it would be incorrect to instead calculate the probability that a report is bad. These two values always sum to one so given one of them it is easy to find the other, but it is important to know which has been given.

The first argument that should be given to StarClassifier is the path to the decision tree, the second is the name of the class to base the calculations on and the third is the configuration file. There is one additional thing that needs to be

kept in mind and that is to make the needed database driver, as specified in the configuration file, accessible. One of several ways of doing so is to include the driver and StarClassifier in the classpath and to call StarClassifier with its fully qualified name as seen in the example below.[24] The driver has in the example been assumed to be named `mysql-connector-java-5.1.5-bin.jar` and located in the same folder as `StarClassifier.jar`. The default class has been assumed to be "TASTY" meaning that the higher the output is the more likely the pizza is to be tasty.[25]

```
java -cp StarClassifier.jar:mysql-connector-java-5.1.5-
  bin.jar be.udd.starclassifier.StarClassifier pizza.tree
  TASTY pizza.conf
```

StarClassifier will read all the examples described by the configuration file into memory at the beginning of execution. Depending on the size of the dataset this might make the default amount of memory available in the Java Virtual Machine (JVM) too little. If this is the case the maximum amount of memory can be changed with the Xmx option. The following example shows how to allow StarClassifier to allocate up to 256 Mb of memory.

```
java -Xmx256m -cp StarClassifier.jar:mysql-connector-
  java-5.1.5-bin.jar be.udd.starclassifier.StarClassifier
  pizza.tree TASTY pizza.conf
```

If the memory can not be increased enough it is possible to split up the computations in different runs. Multiple configuration files can be listed after each other and they will then be processed one after another. In order for this to work it is necessary that the two configuration files describe different sets of data in the @load_sql option. It is for example possible to use the id attribute for this and let all examples with an id less than some value be processed in one configuration file and let all other examples be processed in another configuration file.

One possibility to avoid looking at the id attribute to find a suitable split point is to utilize the ISNULL condition. Assume that the attribute that should be updated, "probability", is NULL for all examples in the beginning and that this attribute is updated with the probability once the example in question has been processed. This means that all unprocessed examples have the probability attribute set to NULL. The following two lines from a configuration file will select (up to) 10000 unclassified examples and classify them. The number of examples to select should match how many examples that can be processed given the memory limitations in question. In this case the same configuration file can be listed on the

24 Note that the -jar java option means that other classpath options are ignored.
25 If it would have been chosen to be "NOT TASTY" this argument would have been needed to be quoted to avoid it from being treated as two separate arguments.

command line multiple times to cover all instances. If the database contains 35000 unclassified example the configuration file should be listed four times.

```
@load_sql SELECT id, curry, warm FROM Pizzas WHERE
  ISNULL(probability) LIMIT 10000;
@save_sql UPDATE Pizzas SET probability = ? WHERE id = ?;
```

The above example has solved some of the problems but it is still cumbersome since it requires knowledge about how many unclassified examples are available. To solve even this problem and make it more practical for batch runs the exit code of the program can be investigated. If the program has been executed without any errors occurring but without processing any examples, meaning that the load SQL statement did not select any rows, the exit code will be 100. During normal execution it will be 0. These pieces of information can be put together and a simple script can be written to make the program loop as long as the exit code is 0.[26] If such a looping script will be used it is vital that the configuration file is written so that the SQL statement for loading the examples at some point will return an empty result, otherwise execution will continue indefinitely.

Pre- and Post-processing

It should be noted that StarClassifier only does the actual classification and that additional processing is likely to be needed both before it can be run and after it has finished. The processing done before should calculate all needed attributes and make them readily available. In the pizza example it is possible that the original database does not have an attribute “curry”, but instead have a list of ingredients. In such case the preprocessing phase should include going through all ingredients looking for curry and write the result to this new attribute. Preprocessing can be very tedious and time-consuming, but it is likely the tasks needed can be written together in the form of a script. If that is the case the preprocessing script can simply be run before StarClassifier. The output might perhaps not need much more processing but it has to be interpreted in some way. In most cases the result should not just be stored in a database table but rather used as a base for some decision. In Stardoll's case the outputted value should changed whether an abuse report should be kept or if it should be removed. There are different ways to handle this. One possibility is to use different tables for reports that are estimated to be good and

26 In the shell Bash the exit code of the previously executed program is $?. An outline to a looping script written for such environment could look like the following.

```
exit_code=0
while [ $exit_code -eq 0 ]
do
   [code to execute]
   exit_code=$?
done
```

for reports that are estimated to be bad, and then only handle the reports in the first table. In this case some additional processing is needed after StarClassifier has finished running. Another possibility is to just assign the values to the report and change the other systems to take this into account. The administrative interface showing reports could be changed to only show reports which have a higher value than some threshold. In this case no additional post-processing script needs to be run.

In the general case there are three phases needed. First, the preprocessing phase where the attributes are collected. Second, the actual classification phase during which StarClassifier produces the output. Third, the post-processing phase where the result is handled in some way.

Limitations

StarClassifier is built with just one task in mind and that is to use a J48 decision tree to classify examples in a database. When it comes to this particular task the tool should be fairly comprehensive. StarClassifier can handle numeric as well as categorical attribute values, and also NULL values which are interpreted as missing values. Large amounts of data, different database systems and table layout should not present any problem to the tool either. When it on the other hand comes to tasks outside the original scope the tool has some limitations. It is for example not possible to use other classifications algorithms than J48.

One limitation that should be noted is that StarClassifier only can handle two classes and it will output incorrect results if the decision tree has more than two classes. The reason for this limitation is that Weka's plain text format for decision trees does not include enough information to do a correct classification when more than two classes are present.

It would be possible to build a tool that would be more widely applicable by incorporating classes from Weka. By using Weka's binary format, instead of the plain text format, it would be possible to handle more than two classes. By taking this even further it would be possible to borrow the entire load module for classifiers from Weka which would make it possible to use this tool even for other types of classifiers, such as rule based classifiers. The downside of all this is that increased flexibility and scope comes with a cost. Using a binary decision tree format makes the trees less readable and harder to modify, incorporating classes from Weka increases dependency and so on. It has been a goal during development to make a tool that solves the problem while being simple and standalone.

Appendix B: Effects of Individual Attributes

It is interesting to look at attributes in isolation to see what effect one attribute has. One of the attributes that the Customer Service pointed out as being important was if a user has invested money into their account. It turned out that reports against paying users are good in 38% of all cases, which indicates that this might not be as important as believed. Recall that the average among all reports was 39%. If only the most spending half of the paying users are considered this number is down at 37%, which indicates a trend but perhaps not as useful as many might have believed. It is worth noting that this correlation is also seen when turning the focus to the person filing the report. Reports filed by paying users are good in 41% of the cases. A third attribute that was brought up by the Customer Service was if the user who was reported had the kidlock activate. This did indeed turn out to be relevant since only 21% of reports filed against kidlocked users are good. It is, however, necessary to also say that reports like this are quite rare since only about 1% of all reports are filed against users with this lock in place. The last attribute mentioned by the Customer Service as one that could possibly be of extra importance is if the reported person has recently been cover girl. The assumption is that other users, who think that they are more worthy the title, is jealous and report the actual cover girl on false ground. When looking into the matter it turns out that this assumption does not hold. Among the reports written against users who have been cover girl during the last week, 42% are good. This indicates that a larger, rather than smaller, portion of reports filed against recent cover girls are good, but this conclusion should be taken with a grain of salt since the number of reports filed against this group of users is very small (only 220 in the entire training data).

Analyzing the description text written in the report was believed to be risky since the field was used in different ways by different users, but some correlation between how this field was filled in and the report quality can be seen. Reports which have a description that contains both uppercase and lowercase letter, which can be seen as a rudimentary quality measure of a text, are good in 41% of the cases, while reports that do not meet this criterion are good in 38% of the cases. The average Lix value, considering only reports that have a description present, is 21. Reports that have a Lix value higher than average are good in 41% of the

cases, while the same number is 37% for Lix values below average. Reports that contains both uppercase and lowercase letter and have a Lix value above average are god in 42% of all cases. This indicates that well-written descriptions are an indicator for a good report. The length on the other hand does not appear to have any effect. The average length of a comment is 54 characters, but the proportion of good reports is the same for reports with descriptions both shorter and longer than this.

Reports can easily be grouped by either the type of the reported object, e.g. blog post or guestbook entry, or the category, e.g. threats or asks for password. It might be interesting to look at the statistics within these two groups. The type of the reported object appears to have significant effect on the result. Reports filed concerning blog posts are good in 54% of all cases, while reports concerning friend lists[27] are only good in 15% of all cases. It is worth noting that these two object types are both quite rare and only present in about 2% of the reports each. It should also be noted that these differences are present, albeit not as clear, even if focus is turned towards the more common types of objects. Reports concerning profile page presentations, the most common object among reports, are good in 33% of all cases, while reports concerning guestbook entries, the second most common object, are good in 47% of all cases. The type of object that has been reported appears to have significance, and so does the category the report was filed in. Reports concerning bad language are good in 41% of all cases, while reports concerning threats only are good in 32% of all cases.

It has been shown above that quality differences can be found by focusing at the attributes the Customer Service though were important and also by considering the natural grouping of reports. These quality differences can also be seen by looking at the other attributes, but these are too many to go through one by one so focus will be on the most important ones. In general a few strong, and perhaps anticipated, connections can be seen.

The reporting user's activity. A report's quality tends to improve if the reporting user is an active user. One possible reason for this is that active users have experience of what the site should look like and have learned to know what sort of behavior is accepted. How long the user has been a member, the number of logins and the number of products a user has bought all seem to have importance. Reports written by users who have sent at least one broadcast message[28] are good in 42% of all cases, while the same number is 37% for users who have not engaged in this activity.

27 It might appear strange that friend lists can even be reported, but there is a reason. A user can promote some of its friends as best friends, which means that these friends will be listed on the user's profile page. Together with the name of the friend and a picture of their doll, a short comment can be written. This comment might be offensive and is therefore reportable.

28 User have the possibility to write messages, called broadcast messages, that are shown to all logged in users for a few seconds.

The reported user's history. A user who has violated the rules earlier appears to be likely to do so again. Reports written against users who have gotten a warning in the past are good in 48% of all cases. If final warnings are considered instead this number is up at 53%.

The general reporting history. The amount of previously filed reports against the reported user and the amount of previously filed reports by the reporting user seems to affect. The quality of a report against a reported user seems to lie in line with the quality of previously reports against that user. Reports filed against users who have been correctly reported before are good in 52% of all cases. This number is up at 57% if the other report was filed within the last month. This pattern also holds when looking at it from the other side, i.e. the quality of reports filed against users which have been incorrectly reported earlier are lower than the average. It is possible to ignore the quality of the previously filed reports and only look at the number of them. The more reports that are filed against a user and the more recent they are, the better are the reports that are filed against that user. When it comes to the reporting user, the more reports that have been filed, the better they were and the more recent they were filed, the more likely the user is to write a good report. User writing their first report writes good reports in 35% of the cases, while users who have written more than 12 reports write good reports in 44% of all cases. Users who have some time in the past written a good report write good reports in 45% of all cases.

It is worth pointing out that the reporting user's activity is listed as an influential factor above while the reported user's activity is not mentioned. The case is that both seem to be of importance but the former seem to matter much more. It is for example not possible to notice any mentionable difference between reports filed against users who have sent a broadcast message and users who have not, while a fairly large difference could be seen on this point when focusing on the user writing the report. On some other aspects, there is a difference in both cases, but not as large. It is for example possible to look at the number of products bought by the users. The median value for the number of products a user has when they file a report or when a report is filed against them is 20, so the split point between few products and many products will be there. When considering a user as a reporter the difference between few and many products is fairly large. A user who has few products will write good reports in 36% of all cases while a user with many products will write good reports in 41% of all cases. If focus instead is turned against the reported user this difference will be much smaller. Reports filed against users with few products are good in 39% of all cases while reports filed against users with many products are good in 37% of all cases.

It might also be worth pointing out the trends when it comes to user activity. When looking at a report the reported user's activity has a negative influence on the quality of a report while the reporting user's activity has a positive influence.

One interpretation of this is that active users have been around long enough for learning how the site works and how one should behave to comply with the rules. They are also, thanks to their experience, good at spotting unaccepted behavior and they write good reports about it. Another reason why experienced users appears to follow rules better than less experienced users is that users who do not intend to follow the rules are deleted from the site as soon as this is noticed. In the same way users who were behaving in an unacceptable manner have been contacted by the Customer Service and asked to improve their behavior.

Please note that these numbers might be misleading. For this work the interesting aspect is the proportion between good and bad reports, not so much the number of reports in itself. In most other situations the latter is of great importance. In the text above it has been stated that the proportion between good and bad reports written against paying and non-paying users is almost the same. This is true, but it is not true to based on this state that paying users and non-paying users are sharing a similar behavior. It might be the case that very few reports are filed against either category of users in the first place. It is possible that a group of users are behaving in such a way that no other users file reports against them, but once they do the reports are having the same quality as the average report.