Categorization Algorithms. The good, the bad and the ugly

Group 15: Ann Michelsen, Julia Adamsson, Andreas Eklund

May 14, 2013

Abstract

This paper describes the implementation and testing of different text classification algorithms known as Naive Bayes, Perceptron and Averaged Perceptron. These algorithms are compared to each other to see in what aspects they differ when performing different tasks, if some are better than others and how they compare in run time. Finally the algorithms are also compared to an already implemented K-Nearest Neighbours algorithm. The results show that the simpler classifiers that was implemented in this project performed very well compared to the K-Nearest Neighbours algorithm. Also the performance depended on a greater number of variables; sometimes one classifier was better than the others and sometimes it performed worse.

Contents

1 Introduction and Background

Text recognition is becoming a very important topic within Artificial Intelligence (AI). This because there is a need to categorize and organize documents. It is also becoming more common that documents can be found in digital form, something that is also triggering the need for text recognition[1]. Text recognition can be used for a lot of different tasks such as; classify documents, find valuable information on the Internet for example filtering news after your personal profile and to search through hypertexts [2].

The area of classifying documents can be used to for example look into reviews of different objects to be able to decide what object category a certain review belongs to. There are a number of different algorithms that can be used for text recognition to be able to classify documents; a few of these are Naive Bayes, The Perceptron, Averaged Perceptron and K-nearest-neighbors (KNN) [3]. All these algorithms work in different ways but can be used to achieve the same task on the same basis. This basis is to find attributes or as called in this project; a feature list which is a list with words that are used to do the classification. A feature list can be found in a number of different ways, in this project one certain way is chosen and the same feature list is used for all algorithms. The Naive Bayes and Perceptron algorithms all uses the concept of training on a certain amount of documents and then testing on another part. It is very important that these document sets are distinct since the algorithms must learn to generalize and not memorize.

Naive Bayes is a probabilistic classifying algorithm that is called "naive" because it assumes that the features are conditionally independent of each other in such way that it does not take into account that words that are evaluated together could have another meaning than if they are evaluated separately [3]. For example the review "This book is not good" could have another meaning if the words "not good" are evaluated separately or together. When categorising documents using Naive Bayes first the probability that a specific features will be included in a category have to be found. Then the algorithm can calculate the probability that a specific review belongs to a category from how many features that is included in that review.

The Perceptron is classifying documents by using a weight vector. First the weight vector is trained on a set of documents and moved each time it finds out that the vector was positioned wrong compared to the document. When the weight vector has trained on a certain amount of documents it can then be used to categorize new documents. The Perceptron is easiest to use when there is only two categories involved (true or false) with more categories multiclass perceptron can be used, or a certain number of common perceptrons can be used. A problem that occurs with the Perceptron is that if the set that is trained is sorted, the last elements will influence the vector more than those in the beginning. Therefore a new algorithm called Average Perceptron have been created. In the average perceptron a average weight vector is used that is the average of all the weight vectors that was calculated.

KNN differs from the others in that it does not use training. Instead KNN approximates a classification for a input depending on its k-nearest neighbors. Since there is no training included in KNN it is very significant to get good features otherwise the algorithm will not perform as it would otherwise.

1.1 The problem

The goal in this project was to implement three different categorization algorithms to solve three different text recognition tasks. These algorithms would be able to learn key features of different kinds of texts and thereby be able to classify documents according to the tasks. The algorithms that was implemented in this project were Naive Bayes, Perceptron and Average Perceptron. In addition, a

fourth algorithm that was not implemented by this group was supposed to be utilized, tested and compared to the three algorithms mentioned above.

The first task was to classify documents into different categories (Books, DVDs, Cameras, Music, Health, and Software). The second task was to classify a review in one of the categories as positive or negative. Both these tasks was achieved by implementing the above mentioned algorithms, using these to train on certain documents belonging to known categories. The final task was to train the classifier on one category and then test how it performs on another category.

In addition to the bare implementation of the classifiers, there was also certain subtasks. First, there was a need to make many different kinds of tests to evaluate how well (or bad) the classifiers categorize in comparison to each other. In order to do this a method known as cross-validation, which divides the data into subsets to be able to train on certain sets and validate on other sets, was used to ease up the comparison.

Also, in order to evaluate the classifiers some statistical analysis had to be performed. Confidence intervals had to be found to see how reliable the results was. Also McNemar-tests was carried out to see if the categorizers work well with the same documents, or if that differ much. The McNemar tests will also make it easier to find the documents that the algorithms have trouble categorizing. When those documents had been found, they could be analyzed to see why a certain algorithm had trouble with those documents and for example also look into if the classifiers had troubles with the same documents.

Since there was a need to make many different kind of tests, it was very important to create generic code when implementing the algorithms. This would make it much easier to run all the tests more smoothly, without the need to change the code much between the runs. Also it would be much safer since all the tests could be run at the same time. Otherwise, if a bug was discovered close to the end of the project, certain tests would need to be redone. If the code was ungeneric much work would be needed to be redone. Also it would be simpler to extend the tasks, such as to test with more categories since the categories only are six in this project.

1.2 Results from the literature

Within this project a number of different papers and books was studied to give a better understanding of AI, text recognition, classifiers and how the different algorithms works.

One of these was Thorsten Joachim's paper on: Text Categorization with Support Vector Machines: Learning with Many Relevant Features. The paper discusses different algorithms such as Naive bayes and KNN, which was used to give a better understanding of how the algorithms works and how they could be implemented. Joachim also mentions different techniques that could be used to find the best features, such as using stemming. Stemming means that words like "calculated", "calculates" are mapped to the same word: "calculate". Another concept that was described is to use stoplists to remove features that are considered neutral as for example the words "and" and "or" [2].

Additionally to this scientific paper mentioned, the course book Artificial Intelligence a modern approach by Stuart Russel and Peter Norvig have been used to find information about the algorithms [3]. The book describes Naive Bayes as a classifier that performs well compared to other algorithms and that a boosted version of it that they describe, is one of the best algorithms for general-purpose learning. However, this boosted version is not tested nor evaluated in this project. The book gives an equation for choosing the most likely class (C) (in this project's case a class is a category) given a set of features (x) which is used in this project:

$$P(C|x_1...x_n) = \alpha P(C)\prod_i P(x_i|C)$$

Figure 1: Naive Bayes

Another good thing with Naive Bayes described by the book is that the algorithm has relatively low complexity, meaning it can handle a lot of features and thus scales well.

Russel's and Norvig's book also provides an equation for the weight vector in the Perceptron algorithm, namely the Perceptron learning rule for linear classification. Where w is the weight vector output, h(x) is the hypothesis, y is the true value, alpha is a constant and x is an input vector:

$$w_i \leftarrow w_i + \alpha(y - h_w(x)) \times x_i$$

Figure 2: Perceptron

This equation opens up for three possibilities deciding how the weight vector will change:

1. Output correct$y = h(x)$. Weight vector not changed
2. y is 1 but $h(x) = 0$. w increased if x_i is positive, and decreased if x_i is negative
3. y is 0 but $h(x) = 1$. w decreased whenx_i positive, increased when x_i negative

To be able to evaluate these kinds of algorithms the book describes a method called k-fold cross-validation. This method includes dividing the data in k different subsets and let 1/k of the data serve as test data and the rest as training data. This is done k times, so that all data has served as test data and the number of correctly classified documents is saved each round. An average of the result from cross validation is calculate and used to evaluate the algorithm.

In addition to the course book and scientific papers Wikipedia has been used as a guidance in how to implement the different algorithms. As it cannot serve as a scientific source or reference, it has merely been used as a guidance.

To select good features for the algorithms a number of different methods can be used. One of these is tf-idf. Tf-idf means term frequency–inverse document frequency and is reflecting how good a feature is for a specific document. The more a feature (word) appears in a document the more important that feature becomes for that category. But the more that feature appears in all the documents the harder it is to use it as a good classifier feature [6].

1.3 Already available tools

The programming language used to implement the algorithms was Java together with Eclipse, mostly because all the group members had previous experience with both these tools. As previously mentioned, it was not necessary to implement the last algorithm which motivated to search for an already implemented one. The one used was an already existing library called Machine Learning Library (java-ML) [5].

There exists some tools to use to solve the problem of documents classification. One of these tools is Weka [7] which is a tool that contains a collection of machine learning algorithms such as Naive Bayes. Weka uses a GUI through which the algorithms can be accessed, tested on different datasets and compared. Since Weka is a open source project, all classes and such for the algorithms can be accessed, meaning that in this project we could have chosen to built upon some of Weka's classes. This was however not done, instead all classes were made from scratch. A good use of Weka in this project would be to use its algorithms and results to compare with the results produced in this project.

2 Overview of the architecture

In this project three algorithms was implemented to solve three tasks within text classification. The three algorithms implemented was Naive Bayes, The Perceptron and Averaged Perceptron. This chapter describes how the algorithms was implemented, what was used and future work on the implementation. There is also a brief description on how our implementations was modified to be able to use the already implemented KNN algorithm.

2.1 Selecting Features

All three algorithms uses the same class for selecting their features, whereas the algorithms and their trainers are implemented in three different ways. The features are chosen by iterating through all the words in all documents that the algorithms will train with, and calculating which of these words that are the most frequent. Then a certain amount of these words are chosen to serve as features.

To do this first of all tokenization is done; special characters like slash, dots and commas are removed and after this the words are counted. When the most frequent words was chosen a number of these words are seen as neutral and not giving the texts any special meaning since they occur in all documents. As described in chapter 1.2 a concept called stoplist can be used to get a hold of this problem, and thus a concept that was used and implemented in the project. The stoplist that was used include the most common english neutral words[4]. In the first version that was implemented, words that did not occur more than three times was removed. Another concept that was tried was removing the 50 first words since these often was "neutral" words.

2.2 The algorithms

This chapter will describe how we implemented the different algorithms. The four algorithms that are discussed below are Naive Bayes, Perceptron, Average Perceptron and KNN.

2.2.1 Naive Bayes

As mentioned in the Introduction chapter Naive Bayes is first training the classifier on the probability that a specific feature is included in a category. When training the classifier all the features will first be selected as mentioned in section 2.1. After this the probability that each of these features will exist in the category will be found. When this training is done the classifier will have the probability for each feature for each category stored in a list. Then the task to try to categorize a review starts. This review is first translated to find what features it includes, a vector with zeros and ones will be filled where a one symbolises that the feature was included and a zero that it did not exist. From this the Naive Bayes algorithm can then map each of the ones and zeroes to the probability that that feature

should/should not be included and multiplicate these with each other. The result from this will then be a probability that the review belongs to each of the categories and the one with highest probability is the one that it will be categorized as.

A problem that will occur with Naive Bayes is that if the vector for the review includes a zero the whole calculation and thus the probability will be zero. To compensate for this something that is called smoothing is used (Laplace smoothing) [3]. This means that in the trainer a +1 will be added so that the feature vector will not include any zeroes. Thus, the equation will not be zero and therefore a more correct answer can be given.

When Naive Bayes was implemented, it was based on the equation and implementation described earlier in chapter 1.2. However the alpha value was not used:

$$P(C|x_1...x_n) = \alpha P(C) \prod_i P(x_i|C)$$

Figure 3: Naive Bayes

Algorithm 1 Pseudocode Naive Bayes

```
int naiveBayesAlgorithm(int[] review, NaiveBayesTrainer trainer){
  double product = 1;
  double highestProbability = 0;
  int trueCategory = -1;
  double[][] categoryFeatures = trainer.getCategoryFeatures();
    for j = 0 to categoryFeatures.length
    product = (1 / number of categories) so that each
    category from the beginning has equal probability.
    Later this can be changed if there is a bigger
    chance that some categories are more common than others.
      for i = 0 to reveiw.length
         if (review[i] > 0)
            The feature was in the documents,
            add probability by multiplying it with
            the rest of the product
            product *= categoryFeatures[j][i];

         else
            The feature was not in the document,
            the probability that the feature would
            not be included is multiplied with the product.
            product *= (1-categoryFeatures[j][i]);
      if (product > highestProbability)
         highestProbability = product;
         trueCategory = j;
        return trueCategory;
end naiveBayesAlgorithm
```

2.2.2 Perceptron

In the Perceptron algorithm a PerceptronEntry object is used to store a file's feature list together with a int variable depicting if the true value of the review's category is true or false. These perceptron Entries (also called weights) are then used to train a weight vector. This weight vector is trained by iterating through the list of weights and if a weight is found to be on the wrong side of the vector the vector is moved depending on the three alternatives described in chapter 1.2. Figure 5 shows the weight vector (red line), if the test subject would be above the line it will be classified as a circle and below as a triangle. If there are more than two categories a multiclass perceptron can be used. Another alternative is to use the perceptron that works with only two categories x number of times, but then the weight vectors have to be normalized to each other.

The equation for modifying the weight vector is based on the equation from the same section:

$$w_i \leftarrow w_i + \alpha(y - h_w(x)) \times x_i$$

Figure 4: Perceptron

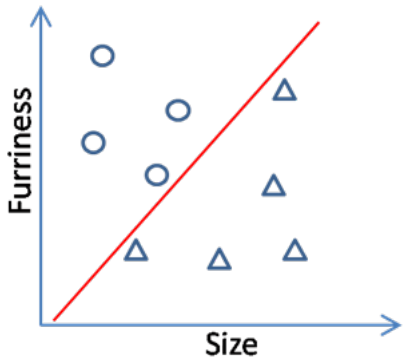

Figure 5: Picture explaining the Perceptron [8]

Algorithm 2 Pseudocode Perceptron algorithm

```
int perceptronAlgorithm(int[] review, double[][] weights
        double highestValue = very small number;
        int trueCategory = -1;
        for k = 0 to length of weights reviewValue = 0;
                for i =0 to length of review
                        reviewValue += weights[k][i] * review[i];
                        if (reviewValue > highestValue)
                            highestValue = reviewValue ;
                                trueCategory = k;
        return trueCategory
end perceptronAlgortihm
```

2.2.3 Average Perceptron

As mentioned earlier the Perceptron algorithm have problems when the set is sorted. Since it tend to make the weight vector be more influenced by the last features. Instead Average perceptron can be used. It works in the same way as the normal Perceptron but instead it finds a average weight vector.

2.2.4 K-nearest-neighbors

Compared to the other algorithms the KNN classifier is never trained. Instead the KNN is first translating all the documents into feature lists but then it "plots" these points. The review is then decided by finding how many of one type of dot that are within the k neighbors. For example in Figure 6 if the green dot is the review that should be categorized and k is three neighbours it will be classified as a red triangle but if k instead is five it will be categorized as a blue square. To avoid ties, k is chosen to be an odd number.

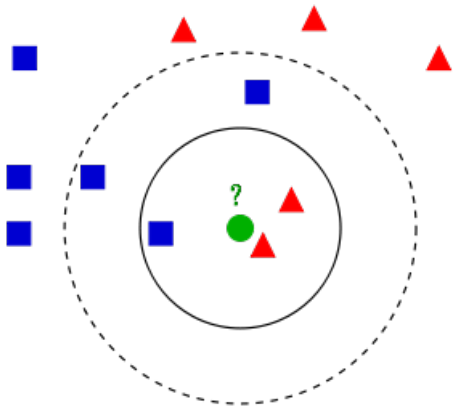

Figure 6: Explaining KNN [9]

2.3 Cross validation

A k-Fold interface with a kFoldClassifier method is implemented in each algorithm to make sure they all use k-fold cross-validation, described in section 1.2. The cross validation is generally implemented the same for each one. It handles the data in different segments depending on the value of the k-fold, to be able to train on a certain amount of data and test on the remaining data. It iterates over all of the data in each category and lets one segment of files at a time be used as test data and the rest as training data, until all files have been tested.

The cross validation gives a quota on the number of correctly classified documents in all categories, which can be used to compare the algorithms with each other. Also the quotas can be used for the statistical analysis, to evaluate the result from the classifiers.

2.4 Future work

Further extension of this project might include to use methods such as stemming and tf-idf which was described in chapter 1.2. This could be used to improve the feature selector and thus provide the algorithms with better features. The tokenization method that is used right now is also something that can be explored and done in a different way. Additionally, it could be changed so that documents that have a very low probability to belong to any of the categories or probabilities in a very similar interval, are pointed out instead of being categorized. For example if one document has a 70.1 % chance to be in the book category and a 70.2 % chance to be in the DVD category, it will be classified

Algorithm 3 Pseudocode for kFold

```
kFoldClassifier(int kFold, String[] folders, String[] features)
   int[number of categories][number of files][feature list for each file]
   allData, trainingData, validationData
   for segment = 0 to segment < kFold
       int[] nrOfTrainingFiles, nrOfValidationFiles, nrOfRest
       fillTrainingAndValidationData();
       PerceptronEntry[][] trainingSet;
       PerceptronTrainer pt;
       for each category
          for each review
             int result = multiPerceptronAlgorithm(review,
             pt.WeightVector);
             if (correctly classified)
                correct++;
        Quota calculated depending on the number of correctly
        classified documents
end kFoldClassifier end kFoldClassifier
```

as a DVD even though it is a very small difference in the probability. Future work possibilities are further discussed in chapter 4.

3 Results and Evaluation

As described earlier we had two different types of implementations for the Perceptron and The Average Perceptron. In the first type the training set was looped through from start to end when training, and in the second type the elements was selected randomly from the training set. It became clear when testing that it was better to select the elements randomly, however then there was no guarantee that all elements was trained on. For the Averaged Perceptron it did however not matter much whether or not the elements were selected randomly, at least not when the feature list was bigger (i.e. more than 100 features).

Another interesting note about the Averaged Perceptron is that when taking elements randomly and looping more times through the trainer (500 times), it actually performed worse as compared to when it looped through the trainer fewer times (100 times). This result is surprising since we expected that it would not reduce the performance, only increase it or at worst be about the same. For the Perceptron however this difference did not affect the result notably (See Appendix B table 2), but when we removed the randomness in selecting elements from the training set it's performance was much increased by looping more.

Normally, when selecting the features for the feature list, a stop word list was used to remove words that probably did not help specifying the category. When we removed that stop word list, the Perceptron and Averaged Perceptron started to perform much better than Naive Bayes (about 7-9 percent points), for as long as the feature list was more than 100 elements long.

When comparing using the stop word list with no stop word list, we can see a remarkable increase

in performance on all algorithms when using fewer than 500 feature elements. However when using more elements than that, the performance difference was reduced but it was still clearly better.

The KNN algorithm performed worse than the other algorithms in all comparisons and it was also the algorithm that took the longest time to run. From the tests, a conclusions can be drawn that the KNN algorithm performs better if there are fewer feature elements, unlike for the other algorithms.

3.1 Text Categorization

Naive Bayes mostly performs equally good or better than the other algorithms when it comes to how big proportion of the testing documents that was categorized correctly. The tests show that when there are fewer features in the feature list, the Naive Bayes clearly outperform the other algorithms. However, when the feature list are bigger, the Perceptron and Average Perceptron starts to perform better compared to the Naive Bayes. When it was tested with a feature list size of 1000 both the Perceptron and the Average Perceptron performed slightly better (86%) than Naive Bayes (85%). The KNN only had about 50% correct guesses and actually performed worse when the amount of features was increased.

For Naive Bayes and the Perceptron it was also tested how they categorized with only two categories (Book and DVD) with a feature list of 300 and 600. Both of the categorizers had 92-93% correct with both feature list sizes (See appendix B table 4).

3.2 In-domain sentiment analysis

For the in-domain sentiment analysis, all of the algorithms that was implemented in this project performed about equally good with slightly above 70% correct guesses, whilst the KNN only had 56-59% correct. All implemented algorithms did best (77-78% correct) when it was camera reviews. A conclusion can therefore be drawn that it is easier to see whether a review is positive or negative when it is trained and tested on camera reviews compared to for other reviews, for some reason.

3.3 Out of domain sentiment analysis

Our results show that generally all the algorithms performed worse when doing out of domain sentiment analysis, which is perhaps not that surprising. This means that it probably exists different jargons when reviewing different categories, for instance a bad music review is having different words when compared to a bad book review.

Some combinations between training set and validation set are giving better results than others. For instance when training on health and validating on camera the correctness is 66% for Naive Bayes, whilst training on software and validating on book the correctness is only 54%. This means that it is probably a more similar jargon on reviews between health and camera reviews as compared to the similarity of the jargon between software and book reviews.

3.4 Statistical analysis

For the calculation of the confidence intervals, we chose the data for 250 features as can be seen in table 1, appendix B. Naive bayes had a confidence interval between 79.65% to 81.08%, whereas the Perceptron had a confidence interval between 78.27% to 79.73%, the Average Perceptron between 78.53% to 79.99% and the KNN had it's confidence interval between 52.27% to 54.06%.

The McNemar tests (See Appendix B table 9-12) shows that usually the Perceptron and the Naive Bayes are capable of categorize the same documents right. But there are some categories where one of them performs worse than the other. One of these is the Health category where the Perceptron classified 420 documents wrong where Naive Bayes classified them right and there is only 230 documents that the perceptron could classify right but not the Naive Bayes (See Appendix B table 11). The Health category and the Music category are the only two categories where the perceptron classified more wrong compared to Naive Bayes. The Music category seems to be the one that both algorithms classifies the most right and same documents right. Except from that there is not much data that stands out, they perform almost as good at the same documents.

The documents that was misclassified by both Naive Bayes and the Perceptron that we inspected had a few common traits. First and foremost, we only inspected misclassified book reviews because of time issues. The misclassified reviews seemed to talk a lot about religion, which perhaps is not so often related to books (See Appendix C). They also had some special assembled words like "must-read", that is translated to "mustread", and similar, which would explain why they did not influence the classifications, since these words are so rare that they probably did not make it to be part of the feature list. Finally one book review seemed to talk a lot about game consoles, which would suggest that it is about software. The correctly classified book reviews, on the other hand, seemed to more often than not contain both the words "book" and "read".

It was also tested if there was a significant difference when modifying the size of the training set. Theses tests were done by modifying the amount of loops in the N-Fold, but as can be seen in table 15 appendix B no significant difference was measured. It could also have been tested to reduce the amount of elements in the training set even more, but it was not done in this project.

4 Discussion and Conclusions

While implementing the algorithms we started out simple to get something working at first and after that we improved the algorithms as we progressed. This was a very good way to proceed and we learned a lot by not diving into the most advanced things at first, and risk getting stuck. One example of this is our feature selector which we first just had returning a previously selected set of words that we chose ourselves, then upgraded it to use the most frequent words in the training sets if the words occurred more than three times. This was an easy way to start and to build on upon; then we added that the first 50 words should be removed since many of the most frequent words are words that appear in every document and thereby does not say anything special about a certain category. In our last version we used a stoplist instead of removing just the 50 most frequent ones.

During the project we have also learned the general mindset about machine learning, how to think when implementing such algorithms. We have also learned about how and when to use these algorithms, for instance that Naive Bayes is good for tasks like spam filtering and that the algorithms over all are good at different tasks. What was the most interesting was to also see this when testing our algorithms to see how they performed within the different tasks. What is also interesting with this project is that this area within AI is not the area one at first think of as AI, at least we in this project group did not do that. This made the implementation very interesting and fun to do, and we could all feel that we learned something new. It is always fun to do something that you have seen working before but did not really know how it was made, for example when a website can learn what you like and based on that give a suggestion on something new you might like, such as Filmtipset [11].

Overall the project went well, with about as good results from our classifiers as was expected. But because of the limited time scope in this project there was no time to explore every aspect that could

be tested and there are a lot of improvements that could be done, mainly in the area of where the features are selected. Stemming, which is explained in chapter 1.2, was never implemented but this could probably improve the probability of categorize the reviews more correct. As for example both the feature "book" and "books" are selected as two different features, and there would probably be a point in keeping those as the same word and same feature.

Another improvement could be to instead of using a list of stopwords as we are right now, use tf-idf is explained in chapter 1.2. This could possibly improve our text categorization a lot compared to how it is done in the current version. The stoplist is working much better than the simple methods we used at first, but tf-idf might improve this further since it also reflects how good a certain feature is for a specific category.

It is also important to consider how many features one should use for the different classifiers. We tested with 50 to 1000 features and used almost 12000 documents within six categories. Our tests showed that 50 features always was a too small amount and that the more features that was used within the interval, the better result. What would be interesting to test but which we did not do is to test with more features than the number of documents. This would probably show that single words would be too strong classifiers.

Another area that we did not have time to explore very much and therefore chose to keep simple is the tokenization of words. There are many different ways how one can do the tokenization, we chose to remove almost all special characters even though for example an exclamation mark or a question mark after a sentence gives it a total different meaning. However, they were removed anyway to keep the feature selector quite simple because of the limited time scope. Though, one special character that consciously was kept is the dollar sign, since we suspected it could be important. For example if people tended to write more about prices in the book category they might use the dollar sign and then this might be a good feature. To improve the tokenization one could test to include and exclude different kinds of special characters and one could also try to use a toolkit such as OpenNLP [10] which can do the tokenization for you.

A further continuation of this project could also include to improve the Perceptron trainer by having a non-constant alpha value that decreases over time (so that it stabilizes itself). Right now we are using a fixed number deciding how many iterations that are allowed in the trainer and we have tested with some different values. But it would also be interesting to let it end the loop when the total error in the weight vector does not change or let the alpha value decrease between iterations, to see how this could improve the result.

For the Perceptron and Averaged Perceptron algorithm we are right now using the files they train on in a random order, this is because of the Perceptron which might get biased based on the last documents it trained on since that is the last document that influenced the weight vector. To be able to classify documents from more than two categories there are two different approaches one can chose when using the Perceptron. Either, a multiclass Perceptron can be made, or six different Perceptrons can be made. We chose to do six different Perceptrons which meant we also had to normalize the different weight vectors, because it seemed like the easiest thing to do. If we had more time, we could try with implementing a real multiclass Perceptron.

As was mentioned in the future work section an improvement that can be made is to give more information about the "chance" that a certain document belongs to a certain category. For example if the value to belong to any category is very low our classifiers still says it belongs to the category with the highest probability. Also, if the value to belong to several categories is very similar this could also be conveyed. An extension could be that the classifiers instead point these documents out instead of classifying them. This would be interesting because then we could analyze those documents and try

to see what made them so hard to classify.

Some of the tests we made included to identify documents that the classifiers had problems classify and whether or not the classifiers had problems classifying the same documents. This was made with McNemar tests and we found that the health category was having the most significant difference between the Naive Bayes and the Perceptron. The Perceptron was better at categorizing health reviews though, and we found that the Perceptron was better at sorting most categories. However with the health and music categories the Naive Bayes was better, and it was doing so much better at classifying those categories that it had a better overall result (See table 11 in appendix B).

Most of our results showed that the difference between our different classifiers was so small that the differences were barely statistically significant, with the exception of the KNN which was worse than the others at most classification tasks.

As was described in chapter 1.2 there are tools available to do classification for example Weka. We chose to do all our code from scratch, but it would have been possible to use something already available and extend it. A good use of for example Weka in this project would be to use its algorithms and results to compare with the results produced in this project. Unfortunately this could not be prioritized due to the limited time scope.

In this project we have also learned a bit about the complexity of the different algorithms and the different parts. One can divide the complexity into three parts; finding the features, train the algorithms on their training sets and then to test and find out what category a certain review belongs to. We did not have the time to optimize this, but it is still interesting to give it some thought since it can be a crucial aspect when choosing among the algorithms. For example if one of them performs very well in predicting the right category but is very slow, then one might chose an algorithm that performs less good but is fast. The Naive Bayes has a complexity depending on the number of training examples (n) and the number of features (m), giving O(n*m) [12]. KNN:s complexity of making a prediction is very slow since it requires O(p*q) for each test example, where p is the number of training examples in space R^q [13]. The Perceptron, which is a linear classifier takes only linear time. However, this is difficult to talk about in exact terms since it depends alot on the implementation.

In conclusion, the results we got from our tests show that the Naive Bayes algorithm was best at most tasks, while the KNN was worst. The Naive Bayes code was also comparably easy to implement. If we are ever gonna implement a classifier for some project in the future, we would probably chose Naive Bayes rather than the other algorithms we tested in this project. Not only because it performed best, but also since it have relatively low complexity. In that case, it would have been nice to put more effort into the tokenization of words as well as the selection of features, by for example using stemming and tf-idf. We believe it could be greatly improved upon as compared to this project's version. All in all we learned a lot by doing this project and we feel like the knowledge we acquired can be used in future utilizations.

References

[1] Sebastiani, F. (1999). "Machine Learning in Automated Text Categorization", ACM Computing Surveys, Vol. 34, No. 1. Available from: http://nmis.isti.cnr.it/sebastiani/Publications/ACMCS02.pdf This paper is covering text categorization that falls within the machine learning paradigm.

[2] Joachims, T. (1997). "Text Categorization with Support Vector Machines", Learning with Many Relevant Features. Technical report from the University of Dortmund, 1997. Available from: http://www.cs.cornell.edu/People/tj/publications/joachims_97b.pdf This paper is mainly about why SVM's is good to use in text recognition. But the paper also mentions many other good stuff as well such as stemming, stoplist, number of features and such things. The paper is also describing kNN, naive bayes and other algorithms shortly.

[3] Russel. S, Norvig. P (2010), Artificial Intelligence - a modern approach. Covering Naive Bayes, Perceoton and k-fold cross-validation: 13.5.2, 20.1, 20.2.2, 22.1.2, 18.4, 18.6.3, 18.7. This book is covering the AI subject very well. There is a sections that is taking up the subject smoothing both Laplace smoothing with bad performance and backoff model which is a better approach. The book is also covering Naive Bayes, the Perceptron and KNN well and a lot more.

[4] List of English Stop Words. Available from: http://norm.al/2009/04/14/list-of-english-stop-words/. This link gave us the words to our stoplist that was used in this project.

[5] Abeel.T (2012) "Java machine Learning Library (Java-ML)". Available from: http://java-ml.sourceforge.net/ This is not a paper but a reference to the library that was used for the KNN.

[6] Lang. K (1995) NewsWeeder Learning to Filter Netnews. Available from: http://www.researchgate.net/publication/2388465_NewsWeeder_Learning_to_Filter_Netnews. This focus of this report is how to give a personal news to a user from his/her profile. The paper is also describing Tf-idf.

[7] Weka 3: Data Mining Software in Java. Available from: http://www.cs.waikato.ac.nz/ml/weka/

[8] Machine Learning – Artificial Neurons / Perceptrons. Available from: http://computersciencesource.wordpress.com/2010/01/06/year-2-machine-learning-artificial-neurons-perceptrons-part-1/ A figure that was used to easier explain the Perceptron algorithm.

[9] K-nearest neighbors algorithm Available from: http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm A figure that was used to easier explain the KNN algorithm.

[10] Welcome to Apache OpenNLP. Available from: http://opennlp.apache.org/

[11] Filmtipsen Available from: http://nyheter24.se/filmtipset/

[12] Fleizach.C, S.Fukushima (1998) A naive Bayes classifier on 1998 KDD Cup Available from: http://sysnet.ucsd.edu/~cfleizac/cse250b/project1.pdf

[13] Elkan.C (2011) Nearest Neighbor Classification Available from: http://cseweb.ucsd.edu/~elkan/250B/nearestn.pdf

Appendix

A Individual stories

A.1 Andreas

In this project we mostly worked together in a sort of pair programming, so everybody was more often than not involved in everything. However, at times we split the work. That meant that me and Julia worked more with implementing the N-fold cross-validation and other code related to the evaluation and fixing bugs in our classifiers, while Ann focused more on finding and implementing the KNN algorithm, adapting our feature data so that the KNN could use them. I also helped with adapting the data for the KNN.

Since we wanted everyone to be involved with everything it is hard to find much to write about the individual contributions. However, the first thing we did was try to implement a simple variant of the Naive Bayes algorithm with a fixed feature list. To use the feature list we needed a feature translator which we implemented, working together.

The feature translator was implemented together with a feature selector. At first the feature selector was a simple one, returning an array of words that we selected by hand. When we had that done to a certain degree, I started with trying to create a good feature selector. To do that I also made a file finder that looped through all documents, counting words and selecting the most frequently occurring words in all documents.

The Naive Bayes algorithm was quite easy to implement. However when we started with doing N-Fold cross-validation, many bugs started to appear because we sent in wrong data and similar bugs. We did not notice this until late, because it still gave somewhat reasonable results. However when we compared to other groups it became clear that either we had bugs in the code, or that our feature selector was crap. The feature selector was fine, however, so I spent much time in the end of the coding phase trying to figure out where the bugs where.

When we had a functional version of the naive bayes done, we started to implement the perceptron. To do that, I had to read a lot about perceptrons. After that it was quite easy to implement, since the algorithm is rather straightforward. However, it was harder when we started to implement the multiclass perceptron. We went with creating many Perceptrons whose results were compared with each other, after normalizing the results. I spent much time implementing that, including fixing bugs that became apparent.

Finally, we spent some time implementing the Averaged Perceptron. It was very easy, since it is very similar to the Perceptron. After that, I reran most of the tests to fill our tables with data since our first tests was performed with buggy versions of the classifiers. Finally we all wrote about similar amounts in the report. I think that all in all everybody in the group spent about the same amount of time and effort working with this project.

A.2 Ann

During this project there have been very much pair programming which means that all three of us have been coding on the same computer to make sure that everything works as it should and also so make sure that everyone understands all the different parts. Because of this it is very hard to answer what each individual member contributed with.

I would say that everyone have in some sense been involved in all the different parts. Things that

might differ is that I don't know that much about the implementation of kFold validation because when the other two group members implemented this I instead implemented so that the KNN worked. The KNN is a java library but I had to change the way we handled our data so that it would fit this library. The KNN implementation took longer time than I thought it would, this mostly because before I first used another implementation of KNN that I found on the web. But this implementation was not a good one, it got stuck in its own loops and crashed. But the Java-ML library worked fine. I first used it with their data so to be sure that it actually worked as it should.

We have all been writing the report together and we have almost all the time been sitting together in school. Both because it is easier to discuss things with each other and because it feels like more is being done in school compared to home. I think the choice to sit much in school was a very good, and something that I hope we will continue with during the next project.

A.3 Julia

While working in this project of three people we have been applying a lot of pair programming as well as "tripple" programming where everyone has been the one in front of the keyboard. This means that it is hard to describe exactly who did what. We started out working with the Naive Bayes all three of us which was good to get everyone started and make sure that we all had the same idea of what we should do. Before finishing all tasks on the same algorithm we started out doing the Perceptron and then the averaged Perceptron. Most of this has been made in pairs or all three of us; I have probably been involved in doing everything except the KNN "implementation" because that is a part where we split up. Otherwise I think everyone has been involved in coding in almost every class.

The section I have been certainly involved in is among other things the k-fold together with Andreas. We made two version of it, because the first one was coded very sloppy and was not very understandable. Together we structured it up and made it both more organized and overall better. This made us understand the k-fold very well and made it easy to apply to every algorithm.

In the end we wanted to do the mcNemar tests so I added some code parts where we could handle this data. Then I was also involved in coding the test class and run test as well filling in the data from the tests in out tables. The last thing we did (but also parallel with coding) was writing the report where we all have written about the same amount.

The project was a lot of hard work but it went well in the end and we got along good and worked well in the group!

B Data tables

Text Categorization	Naive Bayes	Perceptron	Average Perceptron	KNN
50 features	0.719	0.641	0.662	0.515
100 features	0.772	0.723	0.723	0.548
250 features	0.804	0.799	0.792	0.532
500 features	0.833	0.834	0.840	0.480
1000 features	0.849	0.861	0.859	0.489
250 featurers*	-	0.794	0.449	-

Table 1: Text Categorization with six categories, stoplist, 50 rounds in the Perceptron trainer and random order for the Perceptron. *500 rounds in the Perceptron trainer

Text Categorization	Naive Bayes	Perceptron	Average Perceptron	KNN
50 features	-	0.170	0.364	-
100 features	-	0.187	0.577	-
250 features	-	0.257	0.765	-
500 features	-	0.348	0.823	-
1000 features	-	0.501	0.852	-
250 features*	-	0.490	0.445	-

Table 2: Text Categorization with six categories, without random order for the Perceptron. *500 rounds in the perceptron trainer

Text Categorization	Naive Bayes	Perceptron	Average Perceptron	KNN
50 features	0.393	0.330	0.339	0.312
100 features	0.575	0.570	0.583	0.406
250 features	0.679	0.753	0.768	-
500 features	0.741	0.832	0.824	-
1000 features	0.788	0.859	0.859	-

Table 3: Text Categorization with six categories and without stoplist, with random order for the Perceptron.

Text Categorization	Naive Bayes	Perceptron
300 features	0.925	0.926
600 features	0.930	0.926

Table 4: Text Categorization with only two categories, Book and DVD.

In-domain sentiment analysis	Book	DVD	Software	Music	Health	Camera
Naive Bayes	0.706	0.723	0.740	0.721	0.749	0.783
Perceptron	0.712	0.700	0.734	0.699	0.714	0.777
Avg. Perceptron	0.703	0.709	0.744	0.687	0.714	0.770
KNN	0.582	0.590	0.560	0.573	0.570	0.572

Table 5: In-domain sentiment analysis with six categories and without stoplist.

Out-of-domain sentiment analysis Naive bayes	Book	DVD	Software	Music	Health	Camera
Book	-	0.613	0.605	0.573	0.572	0.557
DVD	0.605	-	0.602	0.561	0.593	0.601
Software	0.546	0.546	-	0.585	0.624	0.627
Music	0.584	0.621	0.565	-	0.596	0.576
Health	0.565	0.605	0.646	0.599	-	0.657
Camera	0.559	0.566	0.643	0.559	0.665	-

Table 6: Out-of-domain sentiment analysis - Naive Bayes. Training in y-led and testing in x-led.

Out-of-domain sentiment analysis Perceptron	Book	DVD	Software	Music	Health	Camera
Book	-	0.579	0.568	0.573	0.572	0.556
DVD	0.515	-	0.531	0.561	0.593	0.602
Software	0.528	0.539	-	0.585	0.624	0.627
Music	0.537	0.586	0.555	-	0.596	0.576
Health	0.548	0.546	0.586	0.599	-	0.656
Camera	0.525	0.587	0.622	0.559	0.665	-

Table 7: Out-of-domain sentiment analysis - Perceptron. Training in y-led and testing in x-led.

Out-of-domain sentiment analysis Average Perceptron	Book	DVD	Software	Music	Health	Camera
Book	-	0.598	0.549	0.525	0.593	0.537
DVD	0.533	-	0.581	0.582	0.561	0.564
Software	0.519	0.544	-	0.571	0.621	0.609
Music	0.533	0.576	0.579	-	0.576	0.587
Health	0.520	0.573	0.579	0.546	-	0.593
Camera	0.541	0.526	0.644	0.534	0.638	-

Table 8: Out-of-domain sentiment analysis - Average Perceptron. Training in y-led and testing in x-led.

Book	Perceptron	Correct classified	Not Correct classified
Naive Bayes			
Correct classified		1662	72
Not Correct classified		106	160

Table 9: McNemar with 250 features - Books

DVD	Perceptron	Correct classified	Not Correct classified
Naive Bayes			
Correct classified		1514	95
Not Correct classified		161	230

Table 10: McNemar with 250 features - DVD

Health	Perceptron	Correct classified	Not Correct classified
Naive Bayes			
Correct classified		1216	420
Not Correct classified		230	134

Table 11: McNemar with 250 features - Health

Software	Perceptron	Correct classified	Not Correct classified
Naive Bayes			
Correct classified		1393	97
Not Correct classified		183	242

Table 12: McNemar with 250 features - Software

Camera	Perceptron	Correct classified	Not Correct classified
Naive Bayes			
Correct classified		1537	67
Not Correct classified		148	247

Table 13: McNemar with 250 features Camera

Music	Perceptron	Correct classified	Not Correct classified
Naive Bayes			
Correct classified		1721	87
Not Correct classified		48	144

Table 14: McNemar with 250 features Music

	Naive Bayes	Perceptron
kFold = 2	0.798	0.783
kFold = 6	0.803	0.790
kFold = 20	0.802	0.792
kFold = 50	0.803	0.789

Table 15: kFold with 250 features, with random order and with stoplist

C Example of Misclassified Documents

- For anybody, who wish to learn about the Orthodox Church, and Christianity in general for that matter, this book is "must-read".
- This breviary is disappointing to Traditional Catholics! Novus Ordo Catholics will be right at home with this but if you are a Traditional Orthodox Catholic who favors the Roman Rite and who has been traumatized by the profanations inflicted by iconaclastic innovations, this is the same ole same ole Vatican II "stuff."

- Should Martha Rules have included any information regarding her going to prison as a "business decision"? I would have been interested in reading about the pr strategy used in this "criminal spectacle" designed to make the question of Martha/Mdiddy's innocence a moot point. Are honest business practices being pushed to the side? By normalizing, generalizing, and minimalizing Martha/Mdiddy's crimes, are we loosing our morality? How important is integrity to a leader...former CEO of a company? Be "thoughtful

- Playstation and X-box have nothing over the toy of the future: a time-travel device which allows for visits to the past. There's only one problem with Theo's enjoyment of the latest model: it appears to be defective, and his interactions with peoples of the past increasingly holds dangers of changing the future. The debut titles in the newe "Jump-Man Rule" series by James Valentine, Don't Touch Anything provides quick action and unpredictable twists of plot, including romance, making it a most unusual, satisfying adventure story recommended for ages 8-12.