UPPSALA
UNIVERSITET

IT 18 027
Examensarbete 15 hp
Juni 2018

Comparing Methods of Text Categorization

Erik Edward

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Comparing Methods of Text Categorization

Erik Edward

The task of automatically categorizing digital documents of text into a set of predefined categories has become essential to organize the massive amount of electronic data handled at companies. Several different machine learning techniques have been proposed to deal with this task, ranging from purely probabilistic methods to deep neural networks.

In this report, four different classification methods; Multinomial Naive Bayes, Random Forest, Multilayer Perceptron (MLP) and Convolutional Neural Networks (CNN) were chosen and compared in terms of accuracy, efficiency and speed on large datasets containing transaction data.

In terms of accuracy all methods performed reasonably well with Random Forest and Multinomial Naive Bayes performing best at 87.48% and 87.47% accuracy respectively. Looking at training speed Multinomial Naive Bayes was the best training in less than a minute, followed by Random Forest and CNN taking roughly two hours, and at last place MLP taking over two days to complete training. Lastly, the time to classify a testing set was compared taking between 10 seconds (MLP) to 70 seconds (Random Forest) to complete the testing.

In conclusion the Multinomial Naive Bayes was determined to be the overall most attractive out of the tested methods, but the MLP and CNN suffered from being unoptimized and the datasets were of questionable quality which may have affected the results.

Handledare: Arne Andersson
Ämnesgranskare: Di Yuan
Examinator: Olle Gällmo
IT 18 027
Tryckt av: Reprocentralen ITC

Contents

1 Introduction

At companies a large amount of transactions are made every day. Supplies have to be restocked, office tools, furniture, hardware, software, employee expenses and many other purchases happen by the minute. To get a good overview of a companies expenses a categorization of all these transactions is essential. The task of categorizing transaction data is not something that easily can be done manually mainly due to the sheer amount of data that has to be handled, but can also be difficult as descriptions of transactions are often very vague. To deal with this an automated method has to be used.

Since transaction data is just text, the task of categorizing transaction data can simply be seen as the problem of categorizing text. Text categorization (or text *classification*) is frequently used in many different areas, a few examples being spam detection, news filtering and sentimental analysis. This task dates back to the '60s, with the most common method before the '90s being *knowledge engineering* which works by manually defining rules to classify a document to a specific category[13]. As hardware got more powerful statistical and machine learning techniques grew in popularity[19].

A problem that occurs when working with machine learning is acquiring a high quality dataset to learn on and representing data in a way the algorithms can do calculations on. Different data will require vastly different pre-processing steps, and the choice may severely affect the performance of the algorithms[19]. Since computers are unable to naturally understand text, a big challenge with text classification is processing text into a numeric representation of the text that can be fed into the algorithms.

In this report the task of categorizing transaction data is tackled. After covering the datasets and the pre-processing of the data, four different classification methods will be explored:

- Multinomial Naive Bayes
- Random Forest
- Multilayer Perceptron (MLP)
- Convolutional Neural Network (CNN)

These methods will then be evaluated and compared in accuracy, efficiency and speed in order to determine the best method for classifying transaction data.

2 Background

Text categorization (or text *classification*) is the task of, given a fixed set of categories C, assigning a *document* to a category $c \in C$. The task can be seen as a *supervised learning problem* where, given a pre-classified dataset, a *classifier* has to be constructed that can be used to classify new incoming documents.

As the need for automatic text classifiers have increased with the steadily increasing amount of electronic data, a multitude of different classification methods and machine learning techniques have been applied to the problem[19]. Bayesian methods, in particular Multinomial Bayes, have despite their simple probabilistic approach proved to perform well on the task[18, 1].

Decision Trees is another method successfully applied to the task with decent results[1, 12]. Random Forest algorithms, which essentially builds a multitude of decision trees and aggregates the results, have also proven to reach respectable results[17].

With neural networks and deep learning becoming well studied in the world of AI, attempts at applying them on the text classification task have also been made[18]. Some attempts at using more conventional neural networks such as Multilayer Perceptrons have been made, but have shown to be poor performing and slow[13, 4]. More promising are Convolutional Neural Networks which with several different models and approaches have shown to reach state-of-the-art performance[9, 8, 22, 21].

Perhaps the most praised method, that unfortunately is not included in this comparison, is Support Vector Machines (SVMs). SVMs have proven to consistently have state-of-the-art performance on many different datasets[18, 1, 7]. Other methods also commonly used include k-Nearest Neighbour[18, 13] and Linear Least-Square Fit[18, 13].

3 Data and Pre-processing

One of the biggest challenges with machine learning has always been acquiring a high quality dataset to train on and representing the data as *features* that can be fed to the algorithms[5]. In the following section the dataset and pre-processing of the data is presented.

3.1 Datasets

The data to be classified is transaction data where a single *record* consists of a supplier name, a short description of the transaction and the category the record belongs to. In order to train the classifiers a *training set* consisting of 372344 records was provided. In addition, a second set consisting of 395019 records was provided as the *testing set* to be classified. The data has 232 different categories and a massive vocabulary consisting of 392244 unique words.

There are however some issues with the datasets provided which may affect the results. The quality of the data was known to be quite poor. As seen in Figures 12, 13 in Appendix, the class frequency is very unbalanced with some classes occurring less than 10 times in total and with other classes occurring over 10000 times. In addition, there are also records in both datasets that are clearly misclassified, descriptions that are very vague and in a few rare cases there are classes in the testing set that never occur in the training set. Despite this, the classification methods were trained and tested under the assumption that all records are correctly classified. This should create an even field to compare the different methods on, but some methods may get punished more by poor quality data than others.

3.2 Initial Pre-processing

A very common step when working with text classification is to reduce the dimensionality of the data by for example removing words in the vocabulary that rarely occur[1]. However, due to the nature of the data where the descriptions often are very vague, doing a similar reduction on this dataset would result in empty records and thus become impossible to classify. For example a description could consist of nothing but a date or nothing but a unique ID of seemingly random characters. Cleaning up the strings by removing tokens such as parentheses and dashes did not seem to affect performance either, if anything it was observed to worsen the performance. Speculatively, this is most likely due to the fact that vague descriptions are easier to separate with the tokens remaining.

Due to this, the initial pre-processing of the data simply consisted of appending the supplier name and description into a single string. The resulting records had a length between 2 and 20 words.

3.3 Anonymization

The data consists of records where an entities transaction history and/or sensitive personal information may exist. Another classifier that was planned to be included in the comparison was IBM Watson[1] which would require the data to be sent to a third party. To protect the sensitive information an *anonymization* of the data had to made.

There are different ways to anonymize data with a popular model being *k-anonymity*[15], generally accomplished by e.g. *generalization* or *bucketization*. This would however require going through the datasets and manually applying the anonymizations, which was not an option due to the sheer size. It would also result in a severe loss of information needed for classification. The chosen method was instead to create a mapping between words and a unique number. To give an example, the string *"the dog barked at the*

[1]https://www.ibm.com/watson/

neighbours dog" could be mapped into the new string *"1 2 3 4 1 5 2"*.

This does however come with a disadvantage. Many attempts at text classifications use pre-trained word vectors that attempt to model how similar different words are to each other[9, 11]. By using this mapping taking advantage these pre-trained word vectors is not possible and in the case where such a method was used it would have to be generated from scratch.

In order for the comparison between the methods to stay fair, all methods use this anonymized representation of the data.

3.4 The Bag-of-words Model

Machine learning algorithms are unable to understand text, so the words and sentences have to be converted into something they can work with. A simple but effective way of representing text is using *bag of words*. It works by assigning each word in a vocabulary to a unique number. A sentence can then be represented by a vector that is the length of the number of words in the vocabulary. Each index in the array will be equal to the amount of times that word occurs in the sentence.

To give an example, consider the two sentences *I have a dog and a cat* and *I do not have a cat.* Taking all words into account, this could result in a vocabulary [*I, have, a, dog, and, cat, do, not*]. The bag of words representation of the two sentences would end up as: [1,1,2,1,1,1,0,0] and [1,0,1,0,0,1,1,1].

When applying this to the entire dataset there is a problem. The vocabulary has more than 300000 unique words and each record consists of at most 20 words, making each bag of words representation an extremely *sparse array* with over 300000 zeroes and just a few fields of information that is actually relevant. This results in a massive amount of extra unnecessary overhead and required memory. To avoid this overhead, instead of storing the entire vector the data is instead represented by a list of values of the indices which values are not zero.

For the Random Forest and MLP, another pre-processing step is done by calculating the *term frequency–inverse document frequency* (tf-idf) for the input data. Commonly occuring words such as "a", "and", "is" etc often have little to no importance in the classification process of a sentence. Tf-idf is a way of trying to measure the importance of a word during classification by calculating the product between the document frequency of a word, the term-frequency, and the rarity of a word, the inverse document frequency[16].

3.5 n-grams

To properly understand text the context of words in the sentence often play a huge role. When simply counting how often a word occurs in a sentence this contextual information about a word gets lost in translation. A way of maintaining this information is to use *n-grams* where a sequence of n words are taken into account instead of looking at individual words (if $n > 1$). Once again looking at the sentence *I do not have a cat*, different n-gram representations of the sentence can be seen in table 1 below.

$n = 1$	$n = 2$	$n = 3$
I	I do	I do not
do	do not	do not have
not	not have	not have a
have	have a	have a cat
a	a cat	
cat		

Table 1: n-gram word representations of the sentence *I do not have a cat*

N-grams have proven to significantly improve accuracy of certain methods. For a comparison on how different ranges of n-grams affect accuracy for Multinomial Naive Bayes and Random Forest, see Figures 9,10 in Appendix.

4 Methods

In this section the methods to be compared are introduced.

4.1 Multinomial Naive Bayes

Naive Bayes is a purely probabilistic approach to classification, where the idea is to use the probability of how often words occur in classes to determine the probability of a sentence belonging to a certain class. The *naive* part of the method is the assumption that the features (words) are independent of each other.
Given a word vector $X = x_1, x_2, ..., x_n$, classes $C = c_1, c_2, ..., c_m$, the probability of a sentence belonging to a class $c \in C$ is described by: [20]

$$p(c|X) = \frac{p(X|c)p(c)}{p(X)} \tag{1}$$

where

$$p(X|c) = \prod_{i=1}^{n} p(x_i|c) \tag{2}$$

Since $p(X)$ is independent of the classes, it can be neglected during calculations giving the classification rule

$$\hat{c} = \texttt{argmax}\ P(C) \prod_{i=1}^{n} p(x_i|C) \tag{3}$$

Not taken into account with this calculation is the number of occurrences of a word which often play a big role in the classification process. A solution is to use a *multinomial* version of Naive Bayes. Instead, the probability $P(x_i|c)$ of a word x_i belonging to a class c is

$$P(x_i|c) = \frac{N_{ci}}{N_c} \tag{4}$$

where $N_{ci} = \sum_{x \in T} x_i$ is how often the word x_i occurs in the class c in the training set T and $N_c = \sum_{i=1}^{|T|} N_{ci}$ is the total count of words for c.[10]
This approach runs in to the *zero frequency problem* where if a word x has not been encountered before within the class, the product will always result in 0. To avoid this a smoothing is added to the calculations.

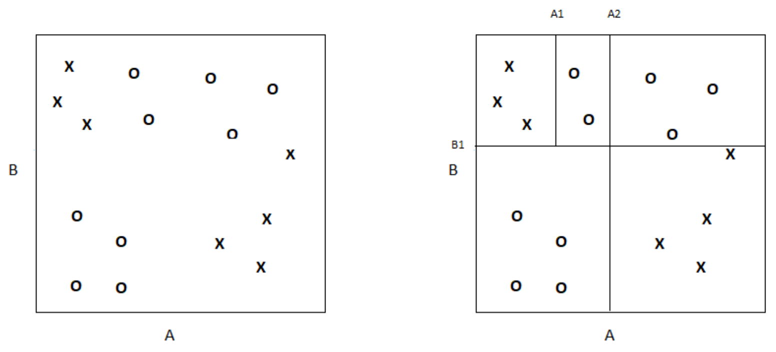
$$P(x_i|c) = \frac{N_{ci} + \alpha}{N_c + \alpha n} \tag{5}$$

where α is the smoothing parameter and n is the number of words in the vocabulary. This smoothing is called *Lidlstone smoothing* or *Laplace smoothing* in the case where $\alpha = 1$. Testing showed that small α values performed best, so the value $\alpha = 10^{-10}$ was used in the experiments.

4.2 Random Forest

Random Forest builds upon constructing a multitude of *decision trees* and aggregating the predictions of the individual trees.

Decision trees attempt to segment a predictor space into regions by taking a set of data and finding a *splitting point* that optimally divides the space in two subspaces. This process is repeated until each subspace does not contain more than one class[6]. Figure 1 below is an example of how a two-dimensional space with two classes, represented by crosses and circles respectively, could be split up into subspaces containing only one class each.



(a) **(b)**

Figure 1: (a) 2-D space with two different classes. (b) Same space with regions separating the two classes.

These regions (or splits) can then be represented as a decision tree by using boolean statements corresponding to the region splits in order to construct a binary tree. Classification of a new point p is done by traversing the tree depending on the values on p until reaching a leaf representing a class. Figure 2 below represents a possible decision tree built from the splits in Figure 1.

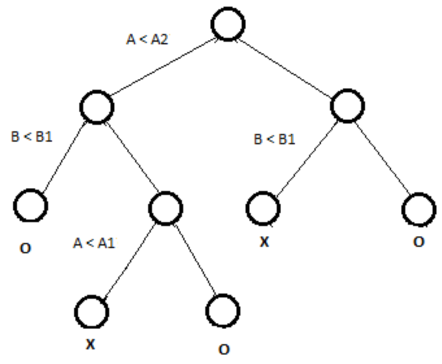


Figure 2: Decision Tree representing the regions in Figure 1.

Text data is quite different from the two-dimensional points in the example, given the massive dimensionality. Still, the way the trees are constructed are conceptually the same where the rules would be in style of "the word *cat* occurs in the sentence". This does however result in very deep trees, some observed to be over a million nodes deep. Another issue with Decision Trees is that they tend to overfit on data they train on. This can be reduced with *pruning* of the tree and attempting to calculate a *purity* of specific splits to decide which splits that are worth keeping, but still tend to be very specialized on the data it trains on. A solution to this is to instead use build a Random Forest[3].

Construction of a Random Forest is done by drawing a sample of the training data and creating a decision tree out of the sample. This process is then repeated until a desired amount of trees are built which together will form a Random Forest[3]. In order to make a classification all individual trees in the forest make a prediction and a winner is determined using a majority vote. For a comparison on how the amount of trees affect performance, see Figure 8 in Appendix. The experiments used a Random Forest consisting of 40 trees.

4.3 Multilayer Perceptron

An *Artificial Neural Network* (ANN) is a mathematical model inspired by the biological neural network structure in the brain[2]. A common class of ANNs is a *Multilayer Percepton* (MLP), a *feed-forward* neural network where data flows through several layers of the network to obtain an output.

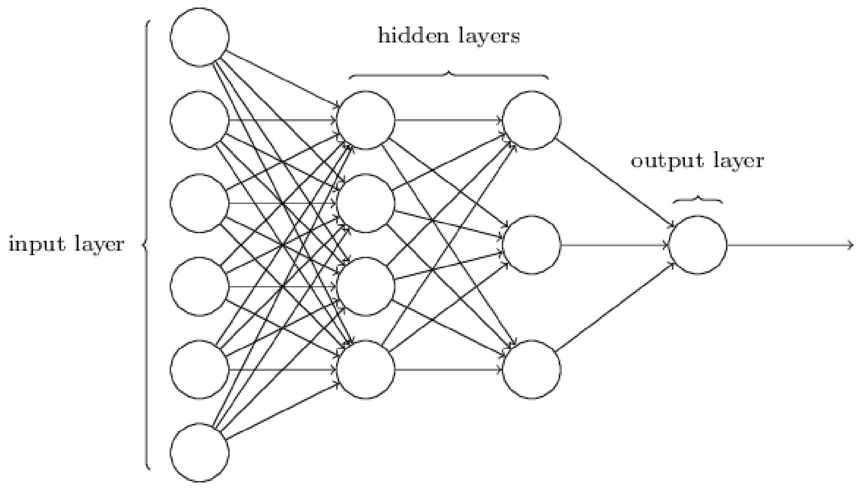


Figure 3: Multilayer Perceptron with two hidden layers. From *https://github.com/ledell/sldm4-h2o/blob/master/sldm4-deeplearning-h2o.Rmd* (3rd June 2018)

The structure of an MLP consists of three parts; the input layer, hidden layers and an output layer, see Figure 3 above. MLPs uses *fully connected* layers, meaning that all neurons in a layer are connected to all neurons in the next layer. The input layer in this case is a representation of the feature vector containing information about the words. Using an *activation function* an output signal is calculated and fed forward towards the next layer of nodes in the network, where the neurons collects and computes the next signal given different weights to the neurons, until it reaches the output layer, which represents the possible classes, where the output is determined.

MLPs are trained through a method called *back propagation.* By using a *gradient descent* optimization method, a single learning iteration (also known as an *epoch*) consists of calculating the output value of the MLP for the training data. The output is compared with the correct result and an error is calculated using a *loss function.* This error is then propagated backwards from the output layer towards the input layer, adjusting the input weights of the neurons[2].

Training neural networks are generally very time consuming[19], which meant no proper experiments to determine the optimal settings and architecture was possible due to lack of time. In addition, due to hardware limitations n-grams of size $n > 1$ could not be used. The experiments used an MLP with two hidden layers with 100 nodes in both layers.

4.4 Convolutional Neural Network

In previous methods n-grams were used in order to extract information about the context of words. Another approach that can be used to extract this information without the use of n-grams is using a type of neural networks called *Convolutional Neural Networks* (CNN)[8]. CNNs uses *convolution layers* and *pooling layers* in order to extract features from the input data before reaching the fully connected output layer.

A convolution layer can be seen as a sliding window that converts a fixed size region into feature vectors. To give an example on a sentence, consider the sentence *I have a black cat* and a convolution *filter* looking at text regions consisting of three words. The filter would look at three different regions: *I have a*, *have a black* and *a black cat* and convert these regions into feature vectors. The pooling layers are usually applied after a convolutional layer and are generally used to reduce dimensionality and providing a fixed sized output by, for example, *max pooling* which takes the maximum value from each convolution in the previous layer[8].

The CNN model used in this comparison is a slightly modified version of an implementation by Denny Britz[2] [3]. Britz implementation is based on a model presented by Kim Yoon[9] which uses a single convolutional and pooling layer, the main difference being that pre-trained vectors are not used in Britz implementation. Instead these word *embeddings* are instead trained from scratch. Given the previous anonymization of the data, pre-trained vectors could not be used regardless which means they would have to be trained from scratch. The architecture of the model is roughly as seen in Figure 4 below.

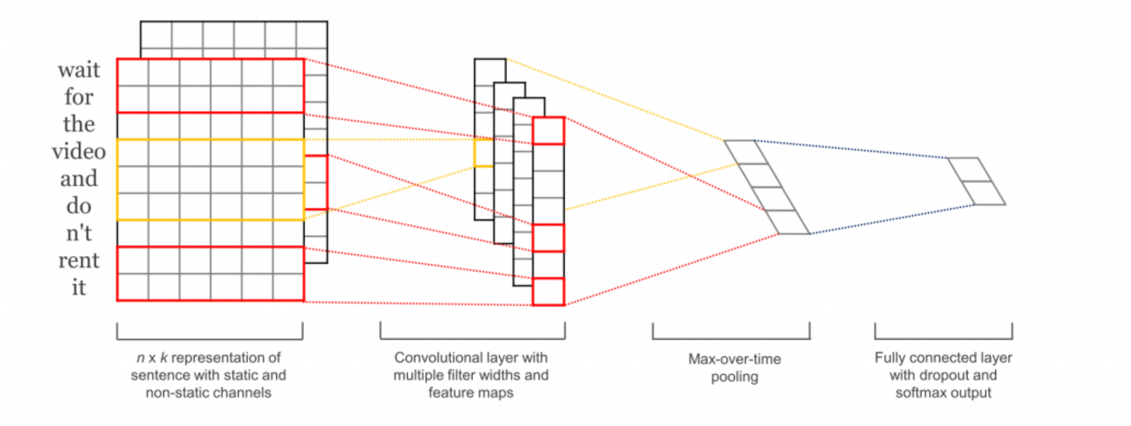

wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 4: CNN architecture. From *Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification*

To begin with, an *embedding layer* is used to convert words into low-dimensional vector representations. After this, the convolutional layer follows which in the implementation uses filters of size 3, 4 and 5. Following this, the features passes through a pooling layer using max pooling that concatenates the convolutions into a single feature vector. In addition, a *dropout layer* is also added which randomly disables half of the neurons, a common way to reduce overfitting in neural networks[14], before finally passing through to the output layer where a prediction is made.

[2]https://github.com/dennybritz/cnn-text-classification-tf (3rd June 2018)
[3]http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/ (3rd June 2018)

5 Performance Comparison

The first metric to compare is how well the different methods perform in pure accuracy. Figure 5 shows the accuracy of predictions on the test set of the different methods.

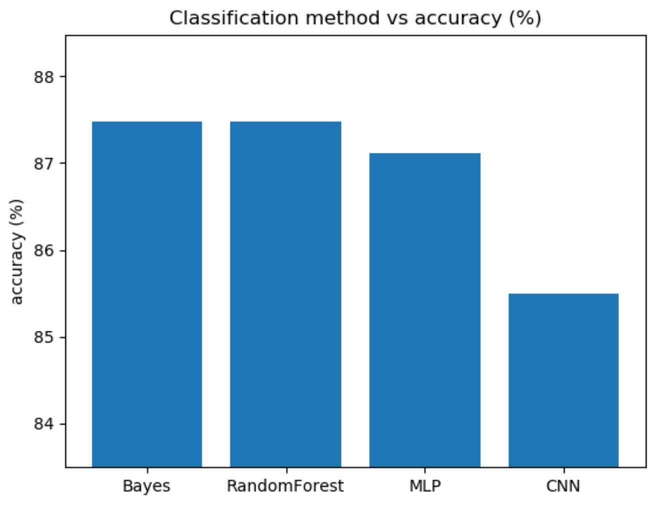


Figure 5: Predictions accuracy for the different methods

Random Forest and Multinomial Bayes performed best with accuracies of 87.48% and 87.47% respectively. MLP came in at third place with an accuracy of 87.11%, and lastly the CNN with an accuracy of 85.50%. The methods were also compared in training time, as seen in Figures 6 and 7 below.

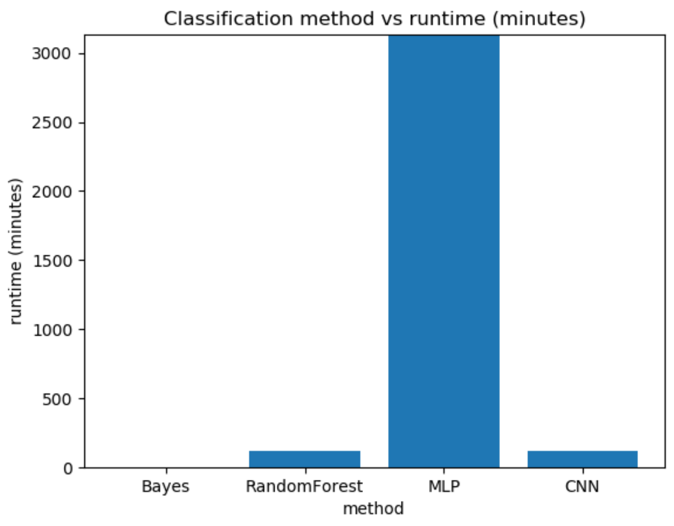


Figure 6: Total runtime (in minutes) for the different classification methods on a computer using Windows 10 on a 2.50 GHz Intel Core i5, with 8 GB RAM. The CNN was run on a computer using Windows 7 on a NVIDIA GTX 970 GPU using CUDA 2.7 parallellisation, with 16 GB RAM and a 3.50 GHz Intel Core i7

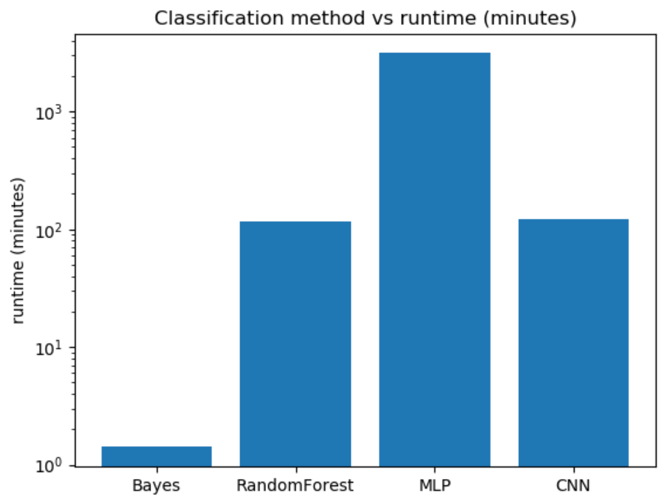


Figure 7: Total runtime (in minutes) for the different classification methods on a computer using Windows 10 on a 2.50 GHz Intel Core i5, with 8 GB RAM. The CNN was run on a computer using Windows 7 on a NVIDIA GTX 970 GPU using CUDA 2.7 parallellisation, with 16 GB RAM and a 3.50 GHz Intel Core i7

The Bayes was by far the fastest, training in seconds. Random Forest and CNN both trained in about two hours, and in last place is the MLP taking over two days to train.
Lastly, the time it took each method to classify the test data was also compared, as seen in Figure 8.

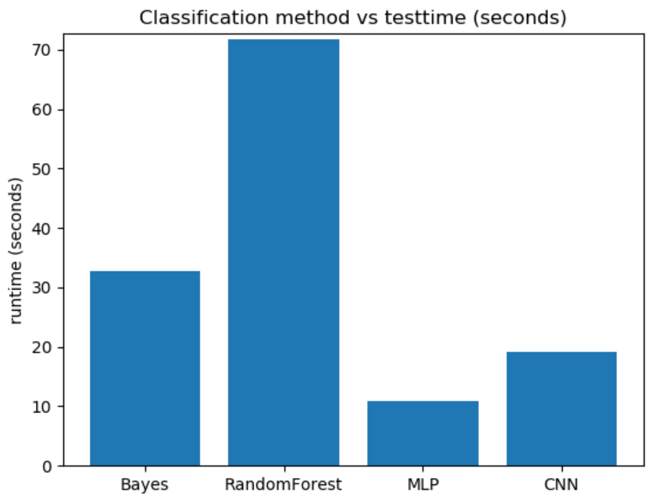


Figure 8: Total testtime (in seconds) for the different classification methods on a computer using Windows 10 on a 2.50 GHz Intel Core i5, with 8 GB RAM. The CNN was run on a computer using Windows 7 on a NVIDIA GTX 970 GPU using CUDA 2.7 parallellisation, with 16 GB RAM and a 3.50 GHz Intel Core i7

In this metric the MLP came out ahead taking 10 seconds to complete the tests, followed by the CNN at 20 seconds. Bayes comes in at third with a testing time of 32 seconds and in last place the Random Forest taking 70 seconds to complete the tests.

6 Discussion

In terms of pure accuracy we can see that Random Forest and Bayes ended up with the best accuracy, with Random Forest pulling very slightly ahead. Not far behind is MLP, and last but not least the CNN implementation. Most surprising to me is the performance of the MLP. The choice of using an MLP was out of pure personal interest as I had previously used it to classify simple drawings and wanted to see how well it would work on a problem like this. It was expected to fail completely, but proved to both outperform the CNN and almost be on par on the other two methods. Further, due to the large training time no proper tuning of layer sizes and other parameters was possible, instead a (not so) qualified guess was made. Even further, due to memory issues it did also not use any n-grams which proved to vastly improve performance in the Bayes and Random Forest methods. Given more time and more powerful hardware it is very likely that the MLP could outperform the other methods in pure accuracy.

The CNN also suffered from the issue of being poorly optimized. Given more time experiments on using more adding convolutional and pooling layers and different convolutional layer sizes could be made. The results from experimenting with n-grams showed that sequences of 4 words and higher made the performance worse. Since the convolutional layers looked at sequences of 3, 4 and 5 words either having a second convolutional layer that looks on smaller sequences of word or just simply reducing the size of the sequences could very likely improve the performance.

Two factors that surely affects the results are that the quality of the training- and test-data was quite poor and the massive dimensionality of the input data resulting from the large amount of words in the vocabulary. Neural Networks generally have difficulties handling such large amount of input nodes.[19] Some methods may also get punished harder by mislabeled data than others, for example neural networks which train by minimizing a loss function could very well have more trouble when mislabeled data is presented than for example the Bayesian approach that is purely statistical.

Looking at the total runtime, which includes pre-processing, training the classifier and running tests, it is clear that the Bayes approach comes ahead by far. Being purely statistical the "training" of the classifier is not much more computationally challenging than parsing the data itself. Building the Random Forest took roughly 10-15 minutes per tree per core. With 40 trees in the forest and 4 cores the forest took roughly 2 hours to build. Training the CNN had a training time similar to the Random Forest, but was trained on a GPU. Training the CNN on a CPU was observed to be roughly 50 times slower, which would have resulted in a training time close to that of the MLP. The MLP had by far the slowest training time taking days, mainly due to the method being slow but also had the misfortune of only being able to train on CPU using a single core at the same time.

The Bayesian classifier comes with another few desirable advantages. If a user wants to know exactly why a sentence was classified to a specific class it is very simple to present the weight different words in the sentence had towards a class. This makes it possible to find flaws in the classification process and fine-tune the algorithm if desired. The other methods are very "black box" as in it takes input and presents a result, with little way of knowing *why* it got that result. Random Forest can technically present exactly why the path it took to got the result it got, as well as visualize the decision trees that are built. Due to the massive vocabulary and dimensionality the decision trees have been observed to be over a million nodes deep, making such a visualization practically worthless.

Another advantage for Bayes is what happens if new training data is presented, or perhaps more relevant if mislabeled entries in the old training data are corrected. For a Bayesian classifier it is simple to recalculate only the affected classes by correcting how often words occur within a class. The other methods can not deal with these tasks without retraining completely wasting a massive amount of time and computational power.

7 Conclusion

In conclusion, all four methods explored have proven to be viable methods for categorizing text. Despite the poor quality of the data and the massive dimensionality of the input, all four classifiers accomplished an accuracy of over 85%. Random Forest and Multinomial Bayes performed best in pure accuracy at 87.48%,and 87.47% accuracy respectively. The MLP came in at third with an accuracy of 87.11%, lastly followed by the CNN with an accuracy of 85.47%. The MLP and CNN does however suffer from being unoptimized and could very likely pull ahead in performance with proper tuning.

Looking at training time the Multinomial Bayes classifier, being purely probabilistic, wins by a landslide taking less than 30 seconds to train. In second and third place are Random Forest and the CNN, taking about 2 hours to train. The CNN does however take advantage of GPU-parallelization which was observed to speed up training by a factor of 50. The MLP was by far the slowest, taking over two days to properly train.

In terms of testing time the MLP pulled ahead, classifying the test-data in 10 seconds. The second fastest classifier was the CNN taking just below 20 seconds, followed by the Multinomial Bayes classifier taking slightly more than 30 seconds. The slowest in terms of testing time was the Random Forest, taking over a minute to complete the classifications.

Other factors that have to be taken into account is that the Random Forest, MLP and CNN are very *black box*, in that it is difficult to know *why* the classifiers predict a certain class. With the Bayesian classifier it is simple to present exactly why a class was chosen by looking at the weights of individual words in a sentence. This, combined with being by far the fastest to train, makes the Multinomial Bayes classifier arguably the most attractive of the four.

8 Future Work

Despite getting acceptable results, there is still plenty of work to be done. As mentioned earlier, the neural networks are very unoptimized and can definitely be improved to get on par, if not become more accurate than Random Forest and Bayes. Given how well smaller n-gram sizes performed it makes sense that using smaller convolution layer sizes could improve performance. The CNN used only had a single convolution and pooling layer. Using larger and deeper CNNs have proven to perform better than smaller in some cases[21], so adding additional layers could also improve performance. Due to lack of memory, n-grams was not used in the MLP which proved to vastly improve the performance of Random Forest and Bayes. With better hardware it is not unlikely that simply using n-grams with the same architecture could make the MLP outperform the other methods.

Arguably the biggest improvement available, which would improve the performance of all methods, would be to clean up and improve the quality of in particular the training set but also the testing set. The most obvious improvement is correcting mislabeled records that does not only negatively affect the training process of the classifiers, but also may reward incorrect guesses and punish correct guesses on the testing set. Another improvement would be to have a more balanced training set. As seen in appendix Figure 12 the training set is heavily skewed towards a few classes, with a few classes never occurring at all. This may lead to the classifiers over-training on those particular classes, but as seen in Figure 14 there does not seem to be any connection between class frequency in train data and accuracy per class (for the Bayes classifier), with the exception being the classes that never occur at all in the data.

Lastly, it would be interesting to see how other methods compare to the ones explored in this report. In particular Support Vector Machines would be interesting, given their reputation of state-of-the-art performance on text classification[7].

References

[1] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management*, pages 148–155. ACM, 1998.

[2] A. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley, 2007.

[3] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[4] X. Glorot, A. Bordes, and Y. Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[5] M. A. Hall. Correlation-based feature selection for machine learning. 1999.

[6] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[7] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[8] R. Johnson and T. Zhang. Effective use of word order for text categorization with convolutional neural networks. *arXiv preprint arXiv:1412.1058*, 2014.

[9] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[10] V. Metsis and et al. Spam filtering with naive bayes – which naive bayes? In *THIRD CONFERENCE ON EMAIL AND ANTI-SPAM (CEAS*, 2006.

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[12] M. K. Saad and W. Ashour. Arabic text classification using decision trees. In *Proceedings of the 12th international workshop on computer science and information technologies CSIT*, volume 2, pages 75–79, 2010.

[13] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[15] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[16] J. D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[17] Q. Wu, Y. Ye, H. Zhang, M. K. Ng, and S.-S. Ho. Forestexter: an efficient random forest algorithm for imbalanced text categorization. *Knowledge-Based Systems*, 67:105–116, 2014.

[18] Y. Yang and X. Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 42–49. ACM, 1999.

[19] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Icml*, volume 97, pages 412–420, 1997.

[20] H. Zhang. The optimality of naive bayes. *AA*, 1(2):3, 2004.

[21] X. Zhang and Y. LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

[22] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

A Appendix

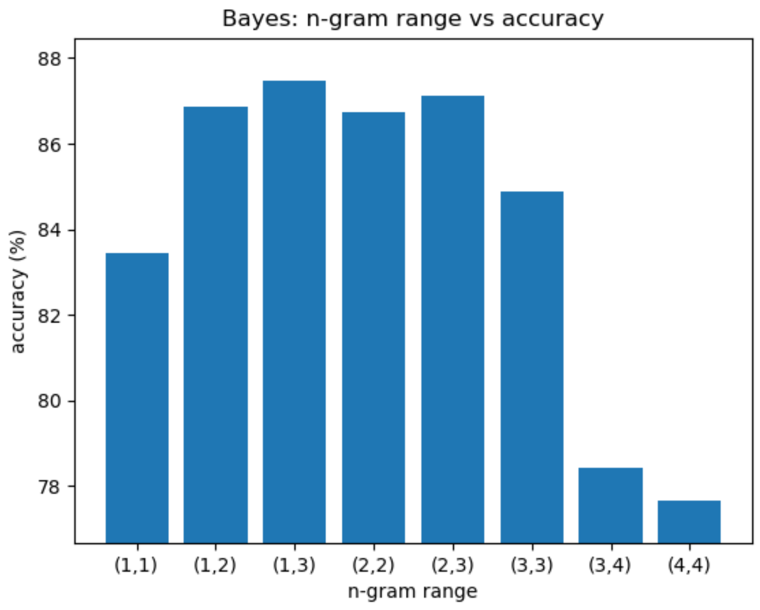


Figure 9: Bayes: ngram vs accuracy

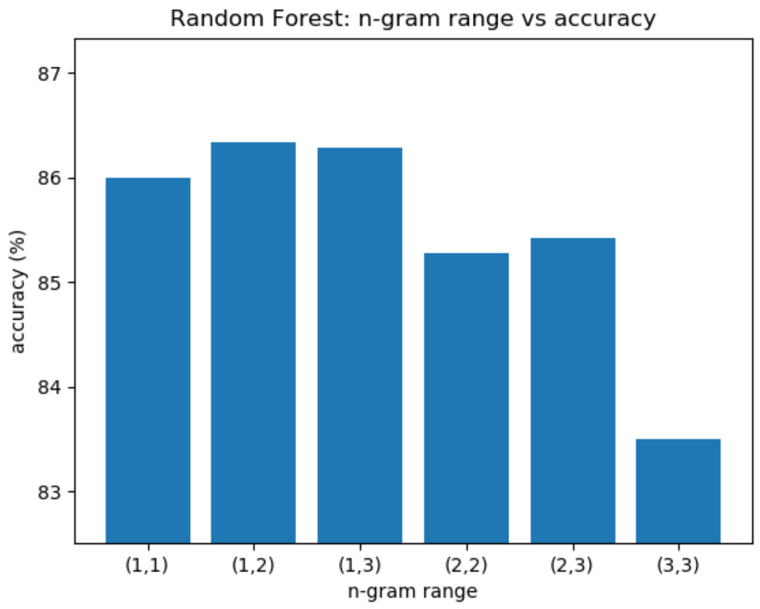


Figure 10: Random Forest: ngram vs accuracy

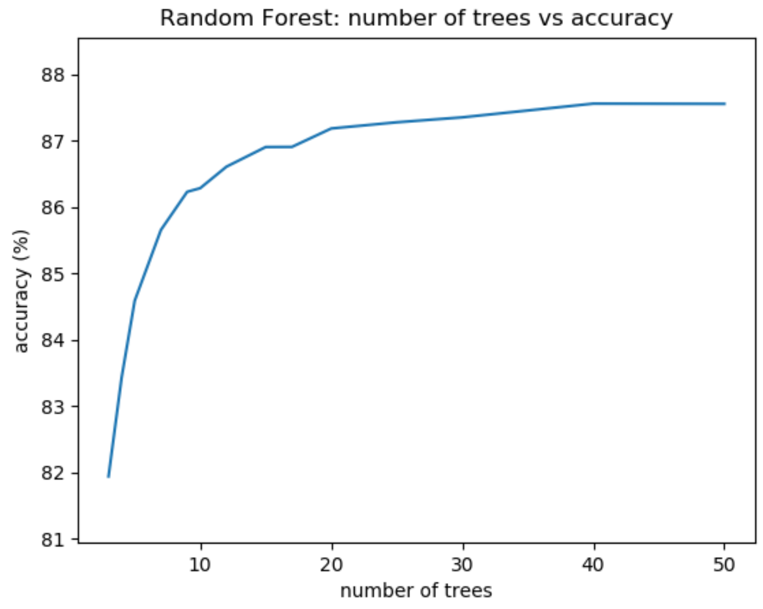


Figure 11: Random Forest: trees vs accuracy

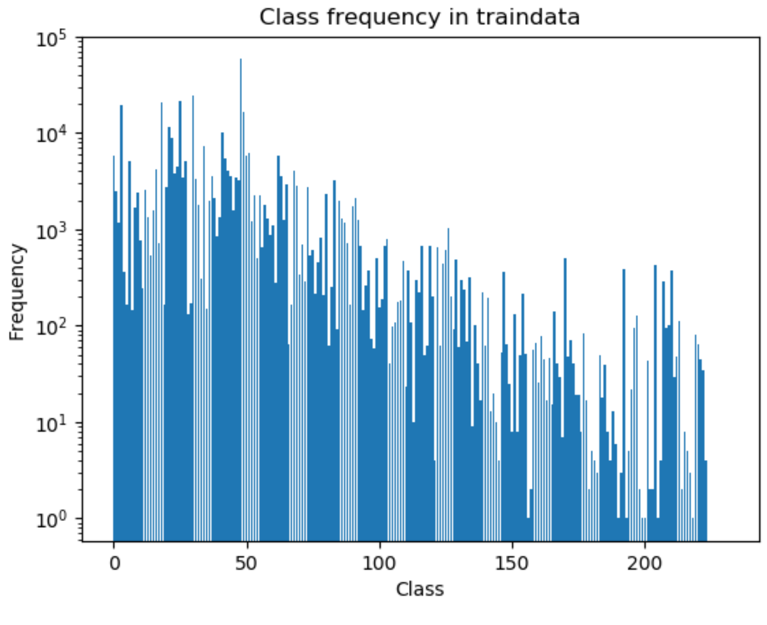


Figure 12: Class frequency in train data

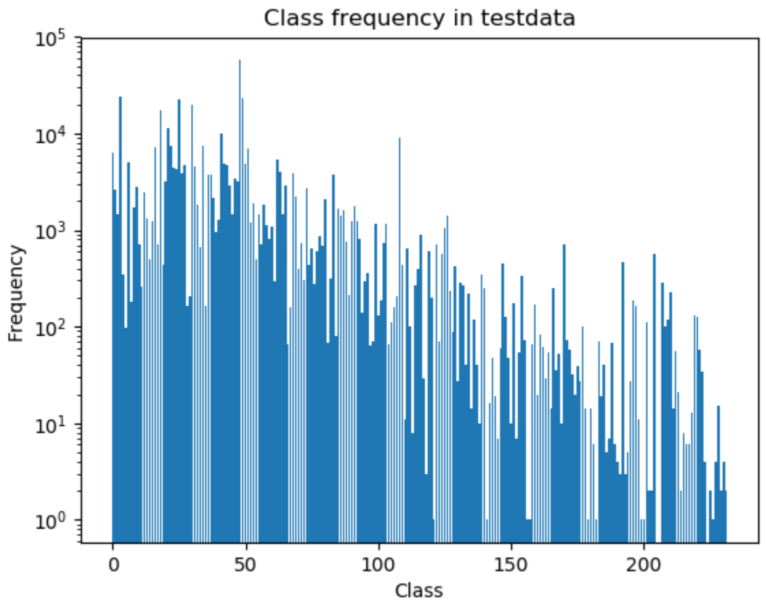


Figure 13: Class frequency in test data

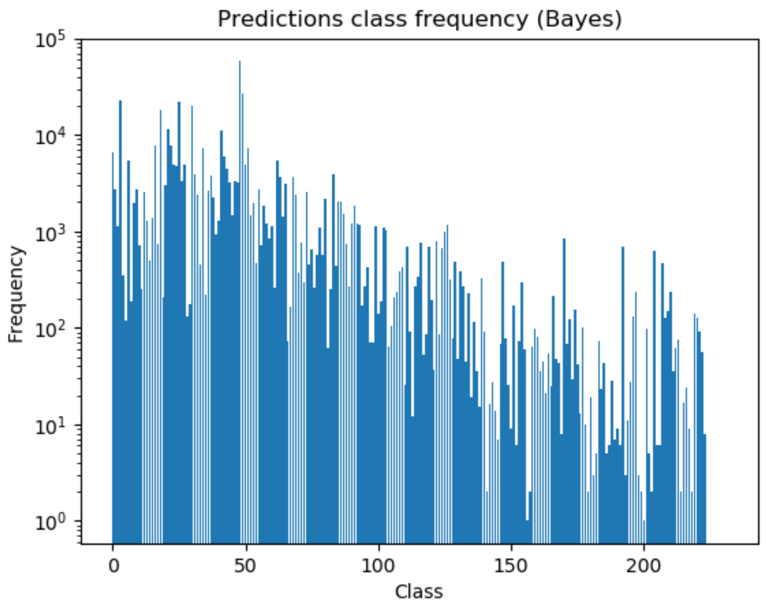


Figure 14: Class frequency in Bayes predictions

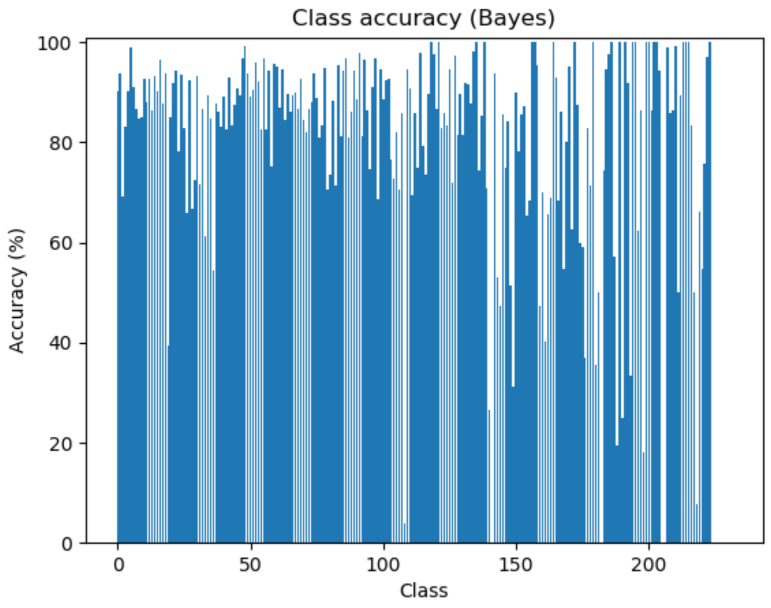


Figure 15: Accuracy per class Bayes