Machine Learning Assignment - Bayesian Spam Filter

Andrei Silviu Tugulan(at14873), Mihail-Calin Ionescu(mi14828)

December 16, 2016

Abstract

Spam content has always represented an issue and has found smart ways to make its way to the larger public. Even though nowadays this problem is mostly encountered in social media and advertisements, the classic example concerning spam detection refers to emails. Many different methods tackling text categorization have been developed. Among these there is a class of early classifiers that use the Bayesian approach. The challenge that comes with this task is extracting the meaningful content from the document, ignoring the irrelevant information, as often documents have a large corpus.

1 Introduction

As any document labeling problem, email classification faces the same challenges and can be broken down in a few steps: decide on the meaningful information, find a way to extract the information from the text, feed the information into the classifier for the training process, and make predictions on future data.

In this project we are taking a closer look at a particular method for text classification that can be also used in the case of emails: the Multinomial Bayes classifier. Moreover, we will look at some more advanced pre-processing techniques that can be applied on the text used to obtain the feature space. At the end, we will use classifier calibration and feature weighting as final tuning methods for our classifier. The solutions in this project have been generated by making heavy use of the sklearn library written in Python.

2 Bayesian methods

2.1 Naive Bayes

Bayes classifiers is a class of linear classifiers that fall into the category of supervised pattern classification. Supervised learning is the task of training a model on labeled data, and use the acquired knowledge to classify new data based on the initial labels.

The word "naive" comes from the fact that the obtained model assumes that the features in the dataset are mutually independent, an assumption that is often broken in real world examples. Even with this unrealistic assumption, naive Bayes classifiers often outperform more complex classification techniques, performing especially well on small data sets. Below we have the formula for P(S|W), which is the probability that an email is spam, given that it contains the word W.

$$P(S \mid W) = \frac{P(W \mid S) \cdot P(S)}{P(W \mid S) \cdot P(S) + P(W \mid H) \cdot P(H)} \quad [1]$$

In the formula, P(W|S) represents the class conditional probability for spam(P(W|H) for ham), P(S) is the probability that any given email is spam(P(H) for ham). P(W|S) can also be read as the proportion of spam emails that also contain the word W.

By combining this probabilities for the set of words in the inputs, we can obtain the combined probabilities.

2.2 Multinomial Bayes

Multinomial Bayes is an optimization of the naive Bayes that can be used to improve the classification results on text documents. While naive Bayes only takes into consideration word presence or absence, Multinomial Bayes keeps track of the word counts. With this additional information, better results are obtained and it is the preferred option for document classification.

3 A Simple Classifier

3.1 Bag Of Words

Before moving to classification, we have to perform some feature extraction on our text data. A good performance of the final classifier is often influenced by how good the feature extraction step was. This problem gains additional complexity in the field of document classification as the feature space is very large. A goal in part 4 will be represented by feature space reduction.

The raw data from the emails cannot be directly fed into the classifier as some numerical feature vectors of fixed size are expected. In order to tackle this, sklearn provides various utilities to extract numerical features from text data: tokenizing (strings are given ids for instance by using whitespace as separation) and counting of each feature. Each individual token frequency will be treated as a feature and the vector of all features will be our sample. Once this format is created, the result can be fed into the Multinomial Bayes classifier.

We do not convert all the words to lowercase, so "Sky" and "sky" will be treated as different words. Also, we only consider the multipart of the email for feature extraction and also the subject of the emails in case the email does not contain any valid words.

3.2 Laplace Smoothing

Since our feature space is constructed of document word frequencies, most of the words will not occur in all of the emails. Additive smoothing permits the assignment of non zero probabilities to words which do not occur in the sample.

3.3 Classification and results evaluation

For the Multinomial Bayes classifier, we used the one provided by the sklearn library. This utility provides parameters for the Laplace smoothing and also for class priors. Thus, the classifier can learn the class' prior probabilities from the training data.

It does not make sense to test our classifier on the training set, a situation that is called overfitting. Instead, we need a new collection of instances that can be used for testing. We used a method called cross-validation, where a test set is held out for final evaluation. The training data is split into k folds(k-1 folds are used as the training data and the result is validated on the remaining fold). Moreover, we decided on using stratified k-fold which keeps the class distribution encountered in the training data.

In order to evaluate the classifier, we need to decide on a metric that is representative depending on what we want to achieve. We chose to use the precision which intuitively estimates the ability of the classifier not to label as positive a negative sample (ham as spam). This aspect is particularly useful as it is more important to have a low number of false positives than to have a low number of false negatives. It is easier for a human to filter a spam email in their inbox rather than always checking their spam box to see if there are any good emails. However, recall is also an essential metric to evaluate the overall performance of the classifier. To include it in our assessment, we also compute the f1-score which is the harmonic mean of the precision and recall.

A random classifier was chosen as a ground truth and we compare its results against the Multinomial Bayes classifier(Table1). It also helps to learn the class prior before the classification step as we get a precision of 0.94 with the prior and a precision of 0.93 without.

Clf/Measure	Precision	F1-score
Multinomial Bayes(prior)	0.93	0.97
Multinomial Bayes(non-prior)	0.92	0.97
Random Clf	0.49	0.58

Table 1: Comparison between Multinomial Bayes and rnd classifier.

Some experiments have been carried out with different values of the Laplace smoothing parameter that goes into the classifier. With a value of 1, precision was 0.95 and with a value of 0.5 the precision was 0.94. Some words will have a count of 1, and by giving 1 to the ones that have a count of 0 can achieve unrealistic results, this being our motivation for these experiments.

4 Pre-processing Methods

Often, the most significant part in using a Machine Learning algorithm for object detection is choosing the most relevant features for the problem at hand. Spam detection is no different from this rule, in fact, the large number of features resulted from text documents adds another level of complexity to the task. To deal with this, some pre-processing techniques can be used before feeding the training data into the classifier.

4.1 Stop Words

Stop Words are usually filtered out in the process of classifying spam emails. They refer to the most common words in the language(e.g. the, is, at, which, on etc). Due to their high frequency in any document, these words carry little information about the type of the email, thus can be ignored. We implement our stop words filtering before the classification process. Removing stop words helps reducing feature space dimensionality, only keeping features with more relevancy.

4.2 Stemming and Lemmatisation

In information retrieval, stemming is the process of reducing the derived words to their root form. For example, a stemmer should be able to identify the strings "mapping", "maps" as based on root "map". Thus, instead of creating a count frequency for each word, there will be just a root where the counts of its derivative words will accumulate. This should reduce the feature space and lead to more accurate predictions.

Lemmatisation is a more complex process than stemming since it intends to find word meaning and is focused on identifying the correct part of speech. It looks for the meaning of a word in a sentence as well as in a larger corpus. The difference between lemmatisation and stemming is that stemming operates on a single word and it is not concerned with the context.

4.3 N-Grams

N-Grams are used in text mining, natural language processing, probability, computational biology and many more. In this model, a token is not just a word, but a sequence of n words. For example, if we have the document "I will eat", a bigram(n=2) will consider as features: "I", "will", "eat", "I will", "will eat". An unigram(n=1) is just the initial set of tokens.

This helps to include some context in our feature space. For example, we may have two words that have a low spammicity overall, but they are always encountered in the same order in spam emails. By including the bigrams in our feature space, we will now be able to get better results in this scenario. N-grams come with a trade off as they take time to compute and there is more space needed to hold the training set which now contains the n-grams.

4.4 Term Frequency – Inverse Document Frequency(tf-idf)

Tf-idf aims to depict the importance of a token in the training set. This calculation is commonly used as a weight factor in text mining to increase the applicability of the information extracted from the document. Tf-idf considers the number of times a token occurs, but also takes into consideration word frequency to encode that some tokens have a bigger count in the corpus. The motivation behind using this method was that it leads to a more accurate representation of the corpus than the word counts.

The value of tf-idf is calculated by the formula: tfidf(t,d,D) = tf(t,d) * idf(t,D) [2], where t is the current term, and d is a document in the set of documents D. The tf is simply the number of times the word appears in the document. The idf measures if the term is encountered often or rarely in the corpus, achieved by taking the logarithm of the total number of documents divided by those containing the word. The formula used by our implementation is $idf(t) = log\frac{1+N}{1+df(t,d)} + 1$[3]. We apply it to every single feature, such that the values change from token counts to tf-idf weights. Thus, the tf-idf value is going to be high for tokens with a high term frequency in the given document and a low frequency of the token in the whole collection of documents.

Before feeding the new feature values into the classifier, we also normalize all the frequency vectors using the Euclidian norm. The tf-idf values are quite sensible to a massive feature space and we will look at this in the experiments.

4.5 Further Additions

Additional to the methods described above, we also tried to remove the words with a high frequency within the given documents(corpus specific stop words). This would be useful if we had, for example, emails from a particular domain that may use specific vocabulary but will not be removed by the stop words stage.

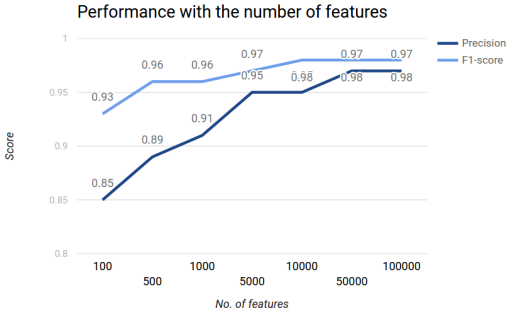

Figure 1: Performance on no. of features

4.6 Experiments and Results

For this stage, we implemented all of the pre-processing techniques described above, and experimented with different combinations of parameters. The results can be different if two pre-processing steps are tried together(Table2). All the experiments depicted in Table2 have been carried out in that order. At each step, we kept the best version we obtained so far.

We first applied stop words, then experimented with both lemmatisation and stemming. The best performance was achieved by using lemmatisation. These steps help to improve the overall results mainly due to reducing the feature space. We then experimented with n-grams for different values of n.

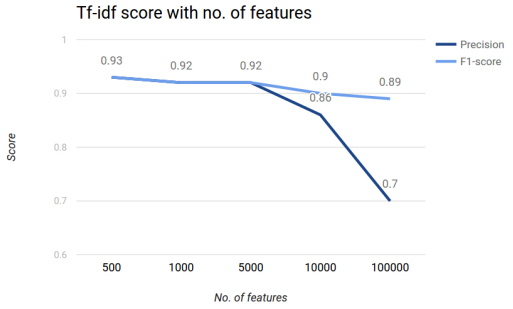

Figure 2: Tf-idf decreasing with the increase of feature no.

Classification results are quite dependent on the parameters. This can be observed clearly when experimenting with tf-idf. The performance for tf-idf decreases drastically when increasing the number of features(Figure2). This is mostly because of reasons explained in subsection 4.4.

However, without tf-idf, the performance increases with the size of the training set(Figure1). This was expected as we are overfitting on the training set.

The last modification was removing corpus specific stop words. The best performance is obtained after this step with 0.97 precision and 0.98 f1-score. This is due to concentrating on the words that carry more meaningful information. We also experimented with different token frequencies(0.5, 0.7, 0.8)-Table2. This means that tokens with a frequency higher than that value will be removed.

As we compare the results with and without these techniques, we can clearly see an improvement. From a precision score of 0.93 to 0.97(highlighted in Table2). in the training set, we reduced the number of false positives from 9 to 2(out of 500 emails) which is what we set to do at the beginning.

We chose to compare our classifier against Decision Trees, as these are easy to understand and implement. To ensure fair comparison, we evaluated the classifiers before and after applying any pre-processing. Also, to determine the best performance for Decision Trees, we used a pipeline where we fed all of our parameters to find the optimal ones. Stratified kfold(k=10) validation was used in both cases. In Figure3 we see the results before and after applying the parameters. Multinomial Bayes outperforms Decision Trees in both cases. This is most probably because Decision Trees often tend to overfit the data, performing bad on new items.

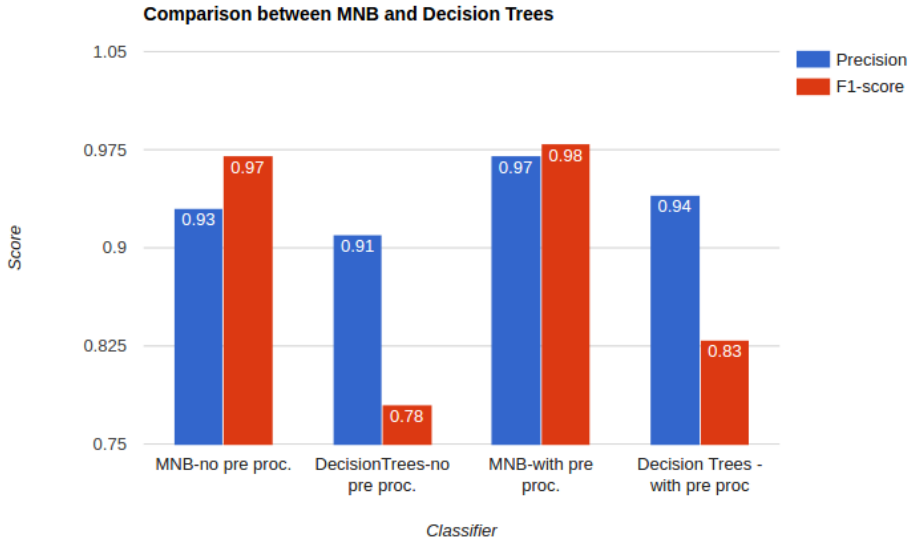

Figure 3: Multinomial Bayes compared with Decision Trees

Method/Measure	Precision	F1-score
Stop words	0.94	0.97
Stemming	0.94	0.97
Lemmatisation	0.95	0.98
Stemming+Lemma	0.94	0.97
n-grams(n=2)	0.97	0.98
n-grams(n=3)	0.96	0.98
n-grams(n=4)	0.89	0.96
tfidf(without max. features)	0.61	0.85
tfidf(max. features=500 with bigrams)	0.91	0.95
tfidf(max. features=500 without bigrams)	0.89	0.95
corpus stop words(0.5)	0.97	0.98
corpus stop words(0.7)	0.94	0.97
corpus stop words(0.8)	0.96	0.98

Table 2: Results after each stage of pre-processing

4.7 Calibration

For a spam filter, we are mostly interested in finding out if an email is either spam or ham, but the probability estimate can give you some confidence about the prediction. It also gives you better flexibility as you can use it as an indicator of where to select a new threshold to distinguish between the two classes, instead of the default (0.5) one. Thus, it is worth taking a look at calibration, which is the process of taking a classifier and creating a function that maps its scores into probability estimate. It should not change the ranking performance of the classifier.

The model we have developed is sensitive to overtraining, which means that the probability estimates tend to be extremely close to either 0 or 1, this is also because Naïve Bayes assumes feature independence. Even so, the classifier is able to make good classification decisions, but the probability estimates are quite bad and the mean squared error can be high in the case that it gets the answer wrong. This is generally the case for Naive Bayes classifiers, which tend to give bad estimations, but are good at ranking.

Initially we have plotted the reliability curve of the classifiers to check how well calibrated they are. We have used a similar method to k folds cross validation when doing this. The classifiers were trained using a portion of the data (80%) and the probability estimates were tested using the rest of the data. To do this, we used the built in scikit calibration curve method[5] which sorts the training data according to their scores and then split it into equally sized subsets called bins. After

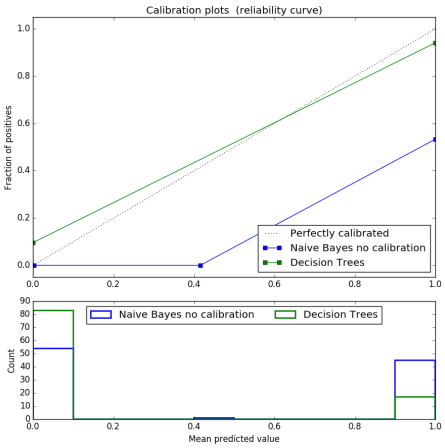

Figure 4: Naive Bayes and Decision Trees reliability curves

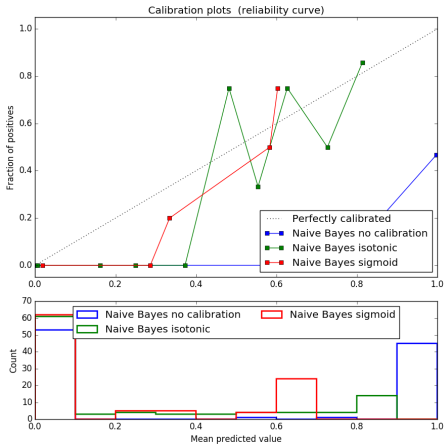

Figure 5: Calibrated Naive Bayes reliability curves

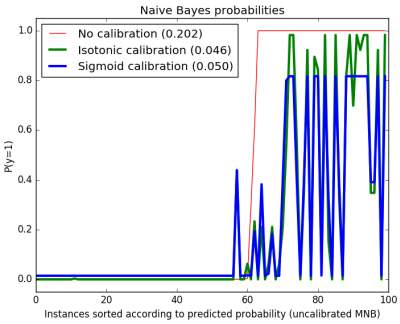

Figure 6: Naive Bayes calibration

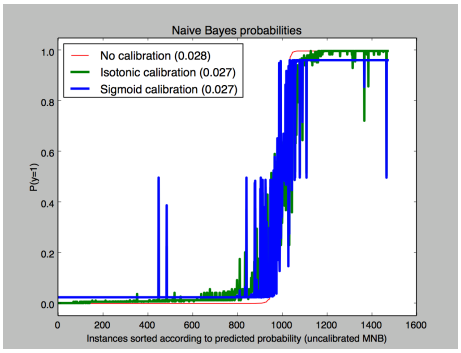

Figure 7: Naive Bayes calibration on the large train set

this, all of the training data instances are plotted according to their estimated probability and the actual probability and are also placed within one of the bins base on their score. The original Naive Bayes classifier is clearly quite far from being an ideally calibrated classifier, while the decision tree classifier already produces good probability estimates does not need to be calibrated (Figure4).

To improve the Naïve Bayes classifier so we can obtain scores that more realistically depict the chances of an email of being either ham or spam, we experimented with two calibration methods: isotonic regression and Platt calibration. To assess the new score, we have used the Brier score which measures the mean squared difference between the predicted probability assigned to the outcome of an email and the actual outcome, which means that the lower the score, the better calibrated the predictions of a classifier are.

The first method we used to try to calibrate our spam filter was the parametric approach based on Platt calibration (that works for binary classification), which passes the output through a sigmoid function. Again, the data was split into two sets, one used for training the classifier and one for testing in order to avoid introducing unwanted bias.

The second tested method was using isotonic regression that finds an arbitrary increasing function that is then used to fit the classifier. It usually requires more data to be able to yield meaningful results. We believe that is why on the initial data set there was quite a bit of variance for the isotonic calibration. When using a larger data set (5000 emails), it seems to outperform the sigmoid calibration (Figure8).

Both methods reduced the Brier score, especially on the intial dataset, from 0.20 to 0.04 when using isotonic calibration (Figure6). Thus, both methods yield a better calibrated classifier, but the isotonic regression seems to work slightly better with the Naïve Bayes classifier when the data set is larger (Figure8). Also, the initial training dataset for is relatively small and we found that the results are sometimes inconsistent.

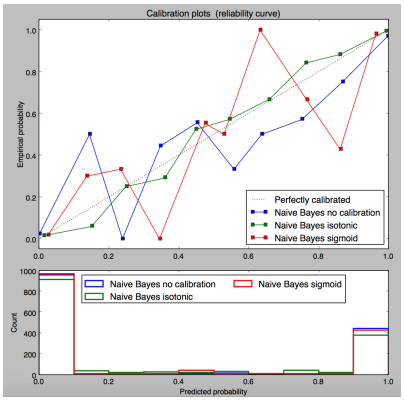

Figure 8: Calibrated Naive Bayes reliability curves on the lrge training set

References

[1] Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal, Byesian Spam Detection, Jeremy J.Eberhardt, University of Minnesota, Morris, http://digitalcommons.morris.umn.edu/cgi/viewcontent.cgi?article=1024&context=horizons

[2] Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[3] Scikit-learn documentation: Feature Extraction http://scikit-learn.org/stable/modules/feature_extraction.html

[4] Alexandru Niculescu-Mizil. Rich Caruana *Predicting Good Probabilities With Supervised Learning*. In Proceedings of the 22nd international conference on Machine learning (ICML '05). ACM, New York, NY, USA, 625-632. DOI=http://dx.doi.org/10.1145/1102351.1102430

[5] Scikit-learn documentation: Probability calibration, `http://scikit-learn.org/stable/modules/calibration.html`