Text Classifier: Performance Analysis

Philip Leo Pascual, Gustav Santo-Tomas, Benjamin Isip

Abstract

In its essence, this paper covers the process of implementing a text classifier using various statistical learning models, as well as analyzing their performance and their significance. We essentially dove into the realm of text processing to gain an understanding on how text-based data provides insight into the way we could manage large sums of non-numerical information. Our end goal is to seek out how to utilize both natural language processing and statistical learning techniques to aid us in modeling a text classifier. We analyzed statistical learning models such as Multinomial Naive Bayes (NB) and Support Vector Machines (SVM), utilizing natural language processing techniques like the Bag-of-Words and N-gram model as well as the Term Frequency-Inverse Document Frequency approach to train them. As an end result, both models showed to have fairly close accuracy rates, despite numerous approaches applied to the dataset. Within this the paper, we'll cover the nuances of preparing, modeling and analyzing our data, going over the nature of our dataset, the techniques that we used, and the results that we came to acquire by implementing the two stated models. We will then present possible future methods for further exploration.

1 Introduction

The purpose of our project is to perform a comparative analysis of text classification learning models. Classification has been traditionally done manually; for example, books in a library may be manually classified by librarians. However, this process becomes expensive to scale, and there are not many alternatives to automation without making Boolean-type rules that are generally restrictive. Apart from these, machine learning based text classification is a third approach, where the decision criteria ("rules") are learned from the training data itself. Classification has many applications beyond the scope of our project. For example, it could be used for searching, spam detection, and sentiment detection.

Our project seeks to analyze which models work best for text classification. We hypothesize that SVMs will outperform NB while using bigrams to train the models, but that NB will potentially perform better while training the models on unigrams. The reason for this is that NB does not assume any relation between terms and assigns individual probabilities, while SVMs instead find the best hyperplane in vector space and splits the data points accordingly, which inherently assigns the same label to similar data points. Given that bigrams reflect a certain amount of context, we thus hypothesize that SVMs will perform better on bigram vectors.

2 Material

For our analysis, we utilized the 20 Newsgroups dataset, which contains approximately 18,846 news documents divided almost evenly amongst twenty news group labels. While each of the topics is distinct, some labels are more similar than others. For example, the topics comp.sys.ibm.pc.hardware and comp.sys.mac.-hardware are quite closely related compared to others such as misc.forsale and soc.religion.christian. To view all the labels (refer to Table 1). The dataset itself comes pre-split between training and testing data, where it was split between messages posted before and after a specific date. In terms of size, the training set contains approximately 11,314 documents, while the testing set contains approximately 7,532 documents.

Each document in the dataset, in addition to the content, contains the following headers: (1) path, (2) newsgroup, (3) from, (4) subject, (5) Message-ID.

20newsgroup Categories	
alt.atheism	com.graphics
comp.os.ms-windows.misc	comp.windows.x
comp.sys.ibm.pc.hardware	mis.forsale
comp.sys.mac.hardware	rec.autos
sci.electronics	rec.motorcycles
soc.religion.christian	rec.sport.baseball
talk.politics.guns	rec.sport.hockey
talk.politics.mideast	sci.crypt
talk.politics.misc	sci.med
talk.religion.misc	sci.space

Table 1. All 20 newsgroup labels

3 Methods

In implementing a text classifier, we knew we had to use natural language processing techniques (NLP) in order to analyze and train the models that we have chosen. We also came to realize that certain aspects of our data influenced the performance of our models, one being the headers of each document and the other being the number of labels we trained our models on.

Within this section we will be going over the techniques as well as the decisions that we had made in processing and modeling our data.

3.1 Testing Various Versions

The first decision that we made was to test various versions of our dataset based on different applications that we applied to it. For example, we considered a dataset that contained all twenty labels, versus a dataset that only contained four labels. Another was to include headers versus excluding headers, (refer to Table 2). It was in this train of thought that we wanted to explore different methods on our data to see whether any of them made a significant difference. Our result section will go further into why including headers, and decreasing the number of labels considered, affects the overall performance of our models.

Versions
Subset (4) without stop-words
Subset (4) with stop-words
Subset (4) with Headers
Subset (4) with bigram
Subset (4) with trigram
Subset (5)
Subset (10)
Subset (15)
Full without stop-words
Full with stop-words
Full with Headers
Full with bigram
Full with trigram

Table 2. In performing our analysis, we created different versions of our dataset to train our models to see whether certain techniques improved the accuracy of our models.

*Unless otherwise noted, each version excludes stop words and headers, and considers a unigram model.

3.2 Natural Language Processing Techniques

One of the NLP techniques that we used to process our data was the process of standardizing all the text within our dataset by converting all words into lowercase. This in itself allows us to capture every occurrence of a word across a document regardless of its typographic form. If left unstandardized, a lowercase version would be considered as a different word than its uppercase version later down our pipeline.

Another technique that we did was the process of filtering out stop-words. In other words, we removed words that carried little to no semantic weight such as determiners, articles and other function words. By doing this, we reduce the number of words that would otherwise would had contribute little to nothing in training our models.

Lastly, as a way of capturing more information beyond just one word, we came to decide to also model our data based on bigrams as well as trigrams. In doing so, we tried to capture nuances of certain labels based on sequences of words that might be distinct to one label versus another. With this in mind, we moved on to utilizing the Bag-of-Words model as well as the N-gram model to extract features for our models, complementary from Scikit-learn's function CountVectorizer.

3.3 Feature Extraction: Bag-of-Words Model / N-gram Model

In working with text-based data, there comes a sort of nuance that distinguishes it from its numerical counterpart. Since we were working with strings and not numerical values, we had to approach our dataset with a natural language processing practitioner's point of view. By doing so, we came across two models that would enable us to encode our data into numerical representations that we could then use to model both our Naive Bayes and Support Vector Machine classifier.

One of them was the Bag-of-Words model, which could also be considered to be a unigram model. What the Bag-of-Words model does for us is that it converts each document into an array of elements, where the length of the array corresponds to the size of the vocabulary. Each element of the array represents the frequency of a word that appears in the said document. This will give us the term frequency (frequency of each word) of each document, which provides us concrete information about the text, regardless of the syntactic structure that the words appear in. One reason why we chose to do this was to see whether we could model based on the most frequent words that correspond to a particular label. On the other hand, we also wanted to see whether certain sequences of words would also correspond to particular labels, so we also implemented the N-gram model.

In terms of the N-gram model, the only difference between it and the Bag-of-Words model is the number of words that it considers, so instead of a unigram (one word), it counts the frequency of a pair of words that are next to each other for a bigram, or a triplet of words for a trigram. This method captures potential collocations (sequence of words that often occur together) that could contribute to the classification of certain labels versus others. By using both models, we can cover as much features that we could capture from a given text.

However, using just term frequencies would lead us into classifying documents based on the most frequent words, which disregards important words that have more weight in terms of classifying, but are relatively less frequent. This lead us to our feature extraction step, where we use the Term Frequency-Inverse Document Frequency approach to weight certain words over others, fine tuning which words contribute the most to classifying rather than the ones that appear the most.

3.4 Feature Selection: Term Frequency – Inverse Document Frequency

To reduce dimensionality and to separate the features most salient to a given document or topic from features that do not necessarily contribute to the classification, we applied term weighting. The weighting process we selected was term frequency-inverse document frequency (*tf-idf*), where term frequency is calculated as the number of occurrences of any given term, be it a unigram or bigram, in a document (Goldberg, 2017:69). Inverse document frequency, then, can be understood as how common that term is over the entire dataset. This means that for example a term such as "try" may potentially occur in all documents in the dataset, whereas certain topic-specific terms most likely will only occur within their own semantic domain, and are thus the most important features to accurately classify a given document as belonging to a certain label (Manning & Schütze, 1999:543). In many ways, this procedure is similar to the use of stop words (see section Natural Language Processing Techniques), but do also account for words that are specifically used within the dataset, i.e., 20newsgroups.
The tf-idf weights where arrived at using the formula provided by Scikit-learn's function TfIdfTransformer:

$$\boldsymbol{idf(t) = \log\frac{1+n_d}{1+df(d,t)} + 1}$$

where n_d total number of documents, and $df(d,t)$ equals the number of documents that contain term t (Pedregosa et al, 2011). To avoid zero division, +1 smoothing is applied so that each term occurs at least once in each document. The quotient is then log-scaled to reflect that a term occurring e.g. three times in a document is more important for the classification but not three times as important as a word occurring once, as well as to assign weights to terms depending on how domain-specific they are in the dataset (Manning & Schütze, 1999:542-3).

The result of the tf-idf process is a dense matrix containing vectors representing weights for words per document, or label. The weights were then used to train and fit the classification models for label prediction.

3.5 Multinomial Naïve Bayes

The multinomial naïve Bayes classifier was used to classify both training and test set. Naïve Bayes is performed by calculating the prior probability for each label in the dataset set by first calculating each label's frequency in the training set. The likelihood for any given label is then arrived at by multiplying each label's prior probability by the fraction of documents with that same label that contains a given feature. The features, again, are the term vectors. That is, if a dataset contains, e.g., 100 documents labeled as "baseball" and a term vector "ball" occurs in 80% of these, then the likelihood of classifying any document containing the term "ball" as belonging to the label "baseball" is 0.8 (Bird, Klein, & Loper, 2009). However, given that a document may contain any number of term vectors that are not mutually exclusive, the process assigns a likelihood score for each label given the term vectors contained within the document and arrives at the one label with the highest probability.

We used the multinomial naïve Bayes classifier provided by Scikit-learn's function MultinomialNB to train and fit the model according to the vector distribution formula:

$$\boldsymbol{\theta_y = (\theta_{y1}, \ldots, \theta_{yn})}$$

for each label y in the dataset, where n is the size of the vocabulary and θ_{yi} is the probability $P(x_i \mid y)$ of term i occurring in a document with label y. The parameter θ_y is estimated by a Lidstone smoothed version of maximum likelihood, using an α value of 0.01:

$$\boldsymbol{\widehat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}}$$

Where $N_{yi} = \sum_{x \epsilon T} x_i$ represents the number of times term i appears in a document with label y in training set T, and $N_y = \sum_{i=1}^{|T|} N_{yi}$ represents the total count of all terms occurring in document y (Pedregosa et al, 2011).

3.6 Support Vector Machines

A soft-margin support vector classifier was trained and tested on the same datasets as the Naïve Bayes classifier. We implemented a One-vs-All classifying scheme for multiclass labeling, in which a stochastic gradient descent (SGD) classifier was trained with a given value of *K* labels. Each of the *K* SVMs was trained to binarily classify a label *K* against the remaining *K*-1 labels using linear SVMs (Pedregosa, et al., 2011). The linear SVM separates a *p*-dimensional vector space with a *p*-dimensional hyperplane in half and assigns the data points on one side of the hyperplane a label y_i = +1 and the data points on the other side a label y_i = -1, so that the labels resulting from the separating hyperplane can be described thus:

$$\boldsymbol{y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) > 0}$$

for *i* =1,...,*n* (James, et al., 2013:340)

This process is repeated for each label, so that each document is assigned to the label for which $\beta_{0k} + \beta_{1k}x^*_1 + \beta_{2k}x^*_2 + \ldots + \beta_{pk}x^*_p$ is the largest (James, et al., 2013:356). We used stochastic gradient descent (SGD) to minimize a hinge loss function by calculating the coefficients *M* and *C*, where *M* is the margin of a data point's distance to the hyperplane, and *C* is a tuning parameter for (James, et al., 2013:346):

$$\underset{\boldsymbol{\beta_0, \beta_1, \ldots, \beta_p, \epsilon_1, \ldots, \epsilon_n, M}}{\boldsymbol{maximize\ M}}$$
$$\textbf{subject to } \boldsymbol{\sum_{j=1}^{p} \beta_j^2 = 1},$$

$$\boldsymbol{y_i(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_{ip}) \geq M(1 - \epsilon)},$$
$$\boldsymbol{\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C},$$

where $\epsilon_i, \ldots, \epsilon_n$ define the soft margin that allows a certain amount of freedom for data points to appear on the wrong side of the hyperplane. That is, each document is assigned a label y_i = {+1,-1} according to the function $f(x*) = \beta_0 + \beta_1 x^*_1 + \ldots + \beta_p x^*_p$.
Thus, a linear support vector classifier can be formalized as (James, et al., 2013:351):

$$\boldsymbol{f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle}$$

where n is the number of α_i parameters and $\langle x, x_i \rangle$ is the inner product between all pairs of a test observation x and each training observation x_i. For the implementation we used Scikit-learn's function SGDClassifiers for Python 3.6.

3.7 Model Evaluation

The performance of each model for each testing set was determined by calculating the multilabel accuracy according to the formula (Pedregosa et al, 2011):

$$\boldsymbol{accuracy}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{\mathbf{1}}{\boldsymbol{n_{samples}}} \sum_{\boldsymbol{i=0}}^{\boldsymbol{n_{samples}-1}} \mathbf{1}(\hat{\boldsymbol{y}}_{\boldsymbol{i}} = \boldsymbol{y_i})$$

where $\hat{y}_i$ and y_i are the predicted and ground truth labels for the *i*-th observation and where $1(x)$ is an indicator function for each label, such that the function returns 1 if predicted and true labels are equal, and 0 if they are not equal. The resulting accuracy level for the model is the fraction of observations for which the indicator function = 1 over the total number of observations.

4 Results

Within this section, we will be going over the results of our models and the different applications that we applied to our dataset. Our first analysis will be on the differences between the full test set containing 20 labels with stop-words, versus the full set without stop-words; like-wise for the sub test sets containing only 4 labels. Then we will compare the full test set versus the sub test set without stop-words. Our second analysis will on the differences between including headers versus without headers for both the full test set and the sub test set. Third, we will analyze the differences between the number of labels considered. Lastly, we will go over the differences among the different n-grams that we used.

In terms of including and excluding stop-words within the full test set, there appears to be no real difference in terms of accuracy for both models. The NB classifier had an accuracy of 70.1% without stop-words and 70% with stop-words. Whereas the SVM classifier had an accuracy of 69.5% without stop-words and 69.7% with stop-words. This can be seen within (Figure 1) where the accuracies show to have no differences if stop-words were included or excluded. The same can be said in regards to the sub test set, where both the NB and SVM classifier showed fairly close accuracies, 83.7% without stop-words and 84.1% with stop-words for NB, 82.3% without stop-words and 81.2% with stop-words for SVM (refer to Table 7).

As one can see in (Figure 1), there is a substantial gap between the accuracies of the full test sets in comparison to those of the sub test sets. Four trials were run on the test set comparing the performance of the models on sets with and without stop-words. After visual inspection of (Figure 1), Welch's t-test was performed on the two sets without stop words, comparing a mean accuracy level of *both* classifiers for the full test set containing 20 labels on the one hand, and the subset with 4 labels on the other. The results showed a statistically significant p-value of 0.0148, which is p<0.05.

Another Welch's t-test was performed on the two sets with stop words comparing a mean accuracy level of *both* classifiers for the full test set containing 20 labels on the one hand, and the subset with 4 labels on the other. The results showed a p-value of 0.069, which is p>0.05 and not statistically significant.

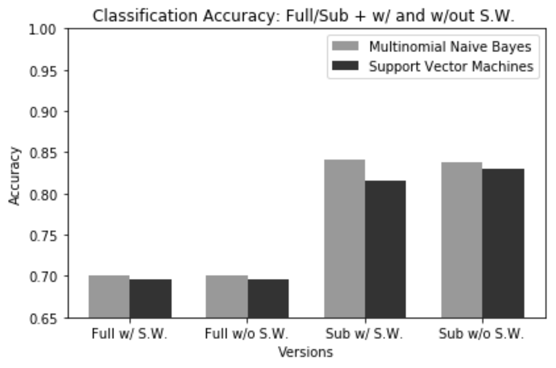


Figure 1. Significant of excluding stop-words

Four trials were run on the test set comparing the performance of the models on sets with and without headers included in the sets. After visual inspection of (Figure 2), Welch's t-test was performed on the two sets without headers, comparing a mean accuracy level of *both* classifiers for the full test set containing 20 labels on the one hand, and the subset with 4 labels on the other. The results showed a statistically significant p-value of 0.0148, which is p<0.05 (see Figure 2).

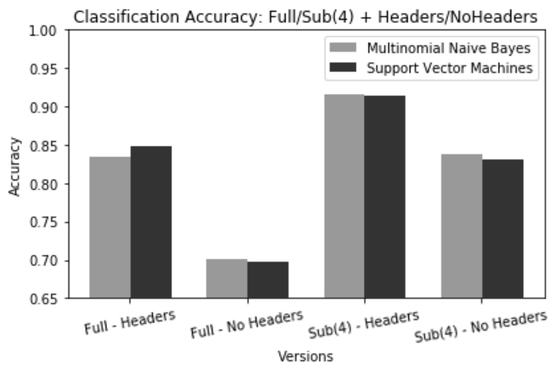


Figure 2. Significance of excluding headers

To test classification accuracy by number of labels, we ran four trials in which we changed the number of labels we ran through our model. The results, as seen below in (Figure 3), show that as we increased the number of labels, the relative accuracy consistently decreased on both multinomial naive Bayes as well as Support Vector Machines. However, the relative accuracy of multinomial naive Bayes was always slightly higher than Support Vector Machines for all cases. (Refer to Table 7) for the accuracy values of each trial.

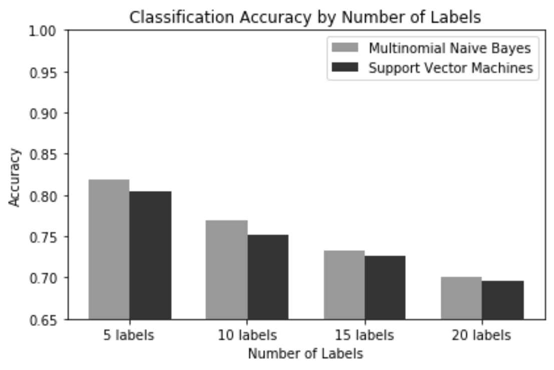


Figure 3. Significance of the number of labels

Six trials were run on full and subset data sets, testing for differences in model accuracy in using various n-grams as input for our model. Originally, we hypothesized that SVMs would outperform NB while using bigrams to train the models, but that NB will potentially perform better while training the models on unigrams. However, as seen above in (Figure 4), grouping n-grams together did not seem to make a significant difference in accuracy for either model. The more substantial increase in accuracy came from using the subset versus original set rather than n-grams. In particular, when using SVMs on bi-grams on the full set of labels, the model performed with 69.6% accuracy, and using NB on unigrams resulted in 70.1% accuracy.

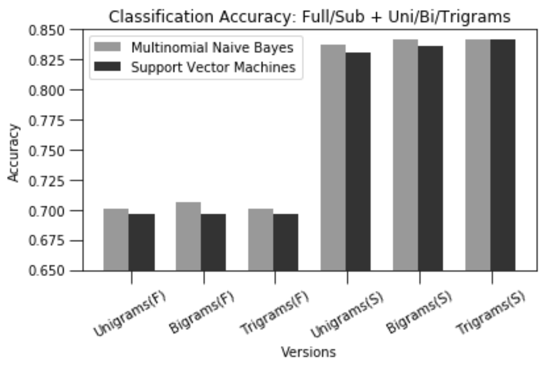


Figure 4. Significance in terms of N-grams
*This graph has an upper limit of .85

After testing both the NB and the SVM classifiers on 13 different versions of the test set, we calculated the mean and standard deviation of each classifiers' accuracy across all test sets (see Table 7).

The NB classifier reached a mean accuracy of 78.7% with a standard deviation of 0.069, whereas the SVM classifier reached a mean accuracy of 77.9% with a standard deviation of 0.070. We calculated both Student's and Welch's t-tests for independent samples and arrived at t statistics of 0.268, where Student's t-test was accompanied by a p-value of 0.790, and Welch's t-test was accompanied by a p-value of 0.790 (identical to the third decimal value). None of the tests displayed a statistically significant difference (p0.05 in both tests) between the classifier accuracy rates.

The accuracy for each label was calculated for both the NB and the SVM classifiers on two different test sets, one in which all term vectors corresponded to unigrams, and one in which term vectors corresponded to bigrams (refer to Table 3, 4, 5, 6). The metrics in the reported tables are calculated as: Precision is the ratio of true positives over true positives and false positives added together.

Recall is the ratio of true positives over true positives and false negatives added together. This is also known as sensitivity or sometimes true positive rate. F1-score is as a weighted mean of precision and recall, ranging from 0 to 1. Support is the number of occurrences of each label in the target dataset.

Naive Bayes performed most accurately for the label ‘rec.sport.baseball’ (f1: 0.87) and the least accurately for ‘talk.religion.misc’ (f1: 0.30) while using unigrams (see Table 3). The mean f1-score for NB on unigrams was 0.70. While using bigrams (see Table 4) Naive Bayes performed most accurately for ‘rec.sport.baseball’ (f1: 0.85) and the least accurately for ‘talk.religion.misc’ (f1: 0.26). The mean f1-score for NB for bigrams was 0.70.

Support Vector Machines performed most accurately for the label ‘rec.sport.hockey’ (f1: .087) and the least accurately for the label ‘talk.religion.misc’ (f1: 0.33) while using unigrams (see Table 5). The mean f1-score for SVMs on unigrams was 0.69. While using bigrams (see Table 6), Support Vector Machines performed most accurately for ‘rec.sport.baseball’ (f1: 0.82) and the least accurately for ‘talk.religion.misc’ (f1: 0.31). The mean f1-score for SVMs on bigrams was 0.70. No statistically significant difference was discernible for the comparison of classification on unigrams or bigrams for either model.

5 Conclusion

No statistically significant conclusion could be drawn as to one classifier performing more accurately across all test trials than the other. Instead, across all 13 tests the classifiers performed almost identical in terms of classification accuracy. Worth noting was that contrary to our original hypothesis, the classifiers performed nearly identical in terms of mean f1-scores on both unigrams and bigrams. Instead, certain labels, e.g., ‘rec.sport.hockey’ varied in terms of precision from the SVM unigram model against all other models.

6 Future Work

In regards to future work, the tf-idf paradigm of reducing dimensions could possibly be exchanged for, or combined with, another estimator, e.g., principal component analysis (PCA). Another possible step would be to select the *k* best tf-idf vectors by applying e.g. a chi-squared test for comparison, and then use only these as predictors. Beyond this, more pre-processing of the text could be useful, such as lemmatization and stemming, in order to further ensure that the best possible term vectors are used.

Furthermore, it would be valuable to test the models on the dataset using cross-validation rather than hold-out validation.

7 References

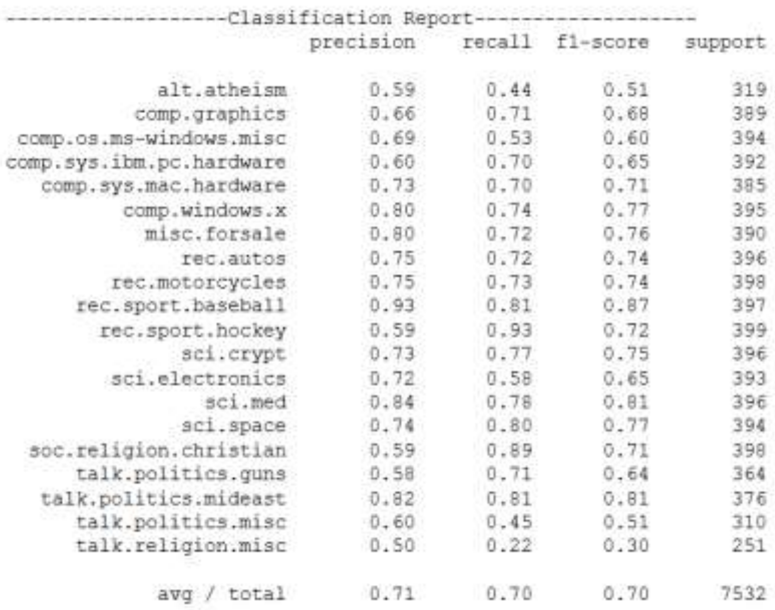
Bird, S., Klein, E, & Loper, E. 2009. Natural Language Processing with Python. O’Reilly Media: Sebastopol, California.

Goldberg, Y. 2017. Neural Network Methods for Natural Language Processing. Ed.: Graeme, H. Morgan & Claypool Publishers.

Manning, C. D. & Schütze, H. 1999. Foundations of Statistical Natural Language Processing. MIT Press: Cambridge, Massachusetts.

Pedregosa *et al.* 2011. Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825

Naïve Bayes: Unigram



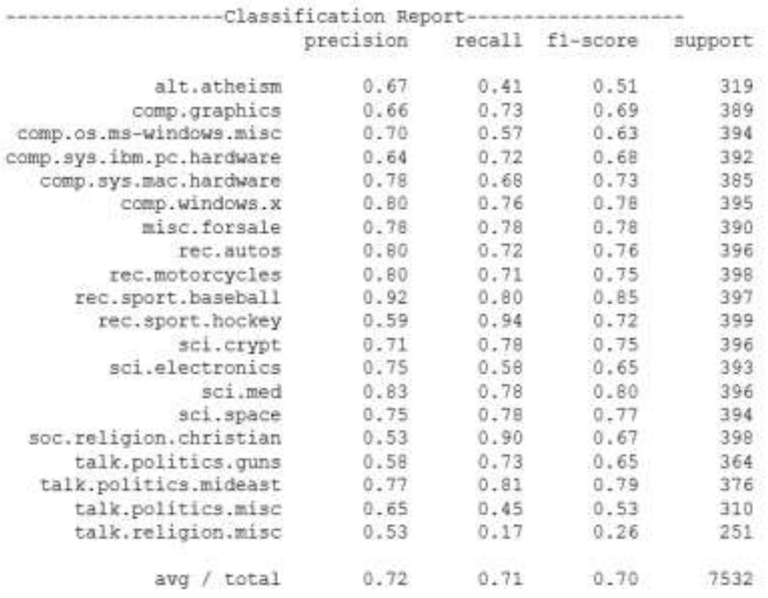
```
--------------------Classification Report--------------------
                          precision    recall  f1-score   support

             alt.atheism       0.59      0.44      0.51       319
           comp.graphics       0.66      0.71      0.68       389
 comp.os.ms-windows.misc       0.69      0.53      0.60       394
comp.sys.ibm.pc.hardware       0.60      0.70      0.65       392
   comp.sys.mac.hardware       0.73      0.70      0.71       385
          comp.windows.x       0.80      0.74      0.77       395
            misc.forsale       0.80      0.72      0.76       390
               rec.autos       0.75      0.72      0.74       396
         rec.motorcycles       0.75      0.73      0.74       398
      rec.sport.baseball       0.93      0.81      0.87       397
        rec.sport.hockey       0.59      0.93      0.72       399
               sci.crypt       0.73      0.77      0.75       396
         sci.electronics       0.72      0.58      0.65       393
                 sci.med       0.84      0.78      0.81       396
               sci.space       0.74      0.80      0.77       394
  soc.religion.christian       0.59      0.89      0.71       398
      talk.politics.guns       0.58      0.71      0.64       364
   talk.politics.mideast       0.82      0.81      0.81       376
      talk.politics.misc       0.60      0.45      0.51       310
      talk.religion.misc       0.50      0.22      0.30       251

             avg / total       0.71      0.70      0.70      7532
```

Table 3: Naive Bayes on unigrams.

Naïve Bayes: Bigram



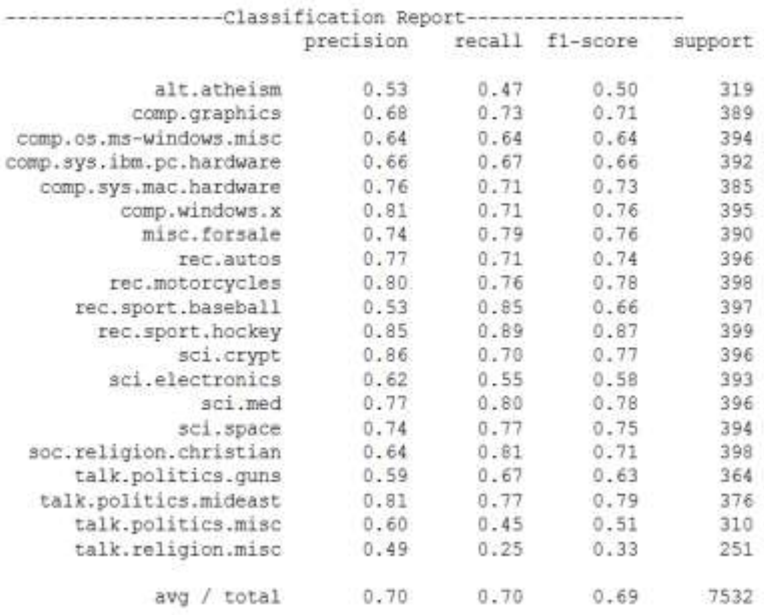
```
--------------------Classification Report--------------------
                          precision    recall  f1-score   support

             alt.atheism       0.67      0.41      0.51       319
           comp.graphics       0.66      0.73      0.69       389
 comp.os.ms-windows.misc       0.70      0.57      0.63       394
comp.sys.ibm.pc.hardware       0.64      0.72      0.68       392
   comp.sys.mac.hardware       0.78      0.68      0.73       385
          comp.windows.x       0.80      0.76      0.78       395
            misc.forsale       0.78      0.78      0.78       390
               rec.autos       0.80      0.72      0.76       396
         rec.motorcycles       0.80      0.71      0.75       398
      rec.sport.baseball       0.92      0.80      0.85       397
        rec.sport.hockey       0.59      0.94      0.72       399
               sci.crypt       0.71      0.78      0.75       396
         sci.electronics       0.75      0.58      0.65       393
                 sci.med       0.83      0.78      0.80       396
               sci.space       0.75      0.78      0.77       394
  soc.religion.christian       0.53      0.90      0.67       398
      talk.politics.guns       0.58      0.73      0.65       364
   talk.politics.mideast       0.77      0.81      0.79       376
      talk.politics.misc       0.65      0.45      0.53       310
      talk.religion.misc       0.53      0.17      0.26       251

             avg / total       0.72      0.71      0.70      7532
```

Table 4: Naive Bayes on bigrams.

Support Vector Machine: Unigram



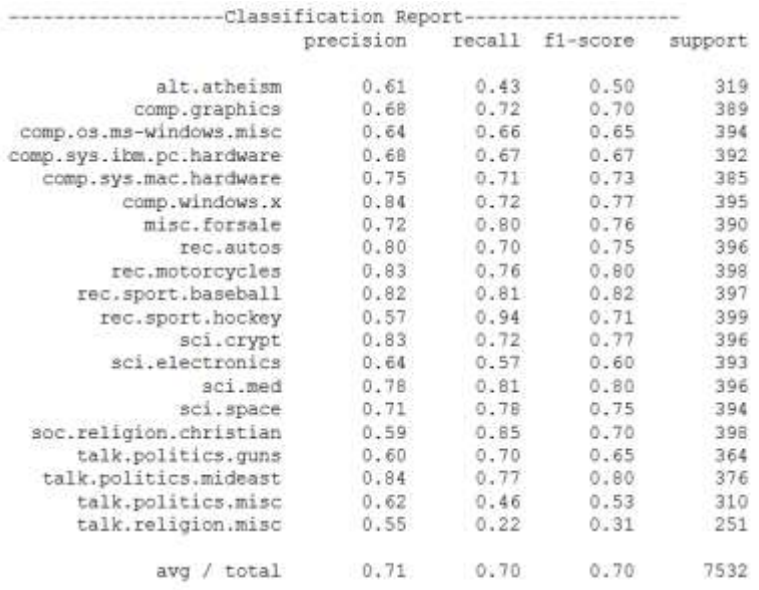
```
--------------------Classification Report-------------------
                          precision    recall  f1-score   support

             alt.atheism       0.53      0.47      0.50       319
           comp.graphics       0.68      0.73      0.71       389
 comp.os.ms-windows.misc       0.64      0.64      0.64       394
comp.sys.ibm.pc.hardware       0.66      0.67      0.66       392
   comp.sys.mac.hardware       0.76      0.71      0.73       385
          comp.windows.x       0.81      0.71      0.76       395
            misc.forsale       0.74      0.79      0.76       390
               rec.autos       0.77      0.71      0.74       396
         rec.motorcycles       0.80      0.76      0.78       398
      rec.sport.baseball       0.53      0.85      0.66       397
        rec.sport.hockey       0.85      0.89      0.87       399
               sci.crypt       0.86      0.70      0.77       396
         sci.electronics       0.62      0.55      0.58       393
                 sci.med       0.77      0.80      0.78       396
               sci.space       0.74      0.77      0.75       394
  soc.religion.christian       0.64      0.81      0.71       398
      talk.politics.guns       0.59      0.67      0.63       364
   talk.politics.mideast       0.81      0.77      0.79       376
      talk.politics.misc       0.60      0.45      0.51       310
      talk.religion.misc       0.49      0.25      0.33       251

             avg / total       0.70      0.70      0.69      7532
```

Table 5: Support Vector Machine on unigrams.

Support Vector Machine: Bigram



```
--------------------Classification Report-------------------
                          precision    recall  f1-score   support

             alt.atheism       0.61      0.43      0.50       319
           comp.graphics       0.68      0.72      0.70       389
 comp.os.ms-windows.misc       0.64      0.66      0.65       394
comp.sys.ibm.pc.hardware       0.68      0.67      0.67       392
   comp.sys.mac.hardware       0.75      0.71      0.73       385
          comp.windows.x       0.84      0.72      0.77       395
            misc.forsale       0.72      0.80      0.76       390
               rec.autos       0.80      0.70      0.75       396
         rec.motorcycles       0.83      0.76      0.80       398
      rec.sport.baseball       0.82      0.81      0.82       397
        rec.sport.hockey       0.57      0.94      0.71       399
               sci.crypt       0.83      0.72      0.77       396
         sci.electronics       0.64      0.57      0.60       393
                 sci.med       0.78      0.81      0.80       396
               sci.space       0.71      0.78      0.75       394
  soc.religion.christian       0.59      0.85      0.70       398
      talk.politics.guns       0.60      0.70      0.65       364
   talk.politics.mideast       0.84      0.77      0.80       376
      talk.politics.misc       0.62      0.46      0.53       310
      talk.religion.misc       0.55      0.22      0.31       251

             avg / total       0.71      0.70      0.70      7532
```

Table 6: Support Vector Machine on bigrams.

Test Accuracies Across Versions		
Versions	**Naive Bayes**	**SVM**
Sub (4) w/o stop-words	.837	.823
Sub (4) w/ stop-words	.841	.812
Sub (4) w/ Headers	.915	.911
Sub (4) w/ bigram	.841	.840
Sub (4) w/ trigram	.841	.836
Sub (5)	.819	.801
Sub (10)	.769	.754
Sub (15)	.733	.724
Full w/o stop-words	.701	.695
Full w/ stop-words	.700	.697
Full w/ Headers	.833	.851
Full w/ bigram	.705	.696
Full w/ trigram	.700	.696

Table 7: Naïve Bayes and Support Vector Machines accuracy rates across all versions