Text Classification of Conversations from National Public Radio Excerpts

Daniel Galeano Timardeep Kaur Arneja Sandra Maria Nawar

260623579 260617404 266065071

McGill University McGill University Concordia University

Montreal, Quebec, Canada Montreal, Quebec, Canada Montreal, Quebec, Canada

daniel.galeanogranados@mail.mcgill.ca [timardeep.arneja@mail.mcgill.ca](mailto:timardeep.arneja@mail.mcgill.ca) [sandra.nawar@mail.mcgill.ca](mailto:sandra.nawar@mail.mcgill.ca)

***Abstract-*-**We are living in the world where we are generating lots of volumes of data on daily basis. Automatic Text Classification comes provides an efficient way of classifying and managing the documents that are scattered and whose numbers is continuously increasing. Text Categorization, from decades, has drawn the attention of researchers and is one of the well-studied problems in the field of machine learning. In this work we aim to classify the excerpts of conversations transcribed from interviews on National Public Radio into one of the categories ‘author’, ’music’, ’movies’ and ‘interviews’ .We will compare the performance of Simple Naïve Bayes, k-nearest neighbors algorithm, and other advanced algorithms such as Support Vector Machines using various kernels on the test set provided.

1. **INTRODUCTION**

Text Classification is a task of assigning categories to different texts and this classification can provide conceptual view of document collection and has important applications in real world [3]. With the advent of Big Data and volumes of data that is generated these days, it has become practically impossible to go over all the data and then provide labels for them. So, instead of classifying labels for all the texts manually, Statistical Text categorization uses machine learning methods to learn the classification rules based on human labelling of Training Dataset. In this study, we aim to classify the excerpts of conversations from National Public Radio into one of the categories ‘author’, ‘music’, ’movies’ and ‘interviews’. Our Training Dataset contains a lot of text, but all the text is not useful and does not convey to us any meaningful or intuitive information as to which category particular text from test dataset belongs to. So, we are implementing bag-of-words method to extract useful words from our training set. The success of any Text Classification Algorithm to a large extent depends on the feature selection. In this work, we are using Mutual Information (MI selector). After successfully extracting the features from training set, we are using the training data to train the Classifiers such as Naïve Bayes, Support Vector Machines and k-nearest neighbors. We are also considering certain heuristics which are specific to the dataset while selecting the features with the sole aim to make classifier perform better. Finally, we will present our results and discussion on each of the `classifier implemented in this paper.

1. **DATA PRE-PROCESSING**

Before we can start with writing an algorithm to perform the task for Text Classification, it was utmost important to familiarize ourselves with the data that is provided to us. We notice that in the dataset provided, we have the first column as ‘Id’ which we will not use for the task of Classification, so we get rid of that column. We also removed stop words and punctuation characters by using the regular expressions Python module *re* and the natural language toolkit *nltk*. Manual manipulation was also implemented to avoid dedicated stop words such as *\_\_eos\_\_*.

*Data Encoding*

The abstraction of natural language texts is essential to optimize and facilitate the implementation of classifiers. The conversation strings were encoded into a vector of token counts, representing the number of iterations of each word in a conversation. Feature selection also was implemented in order to reduce the number of elements in this vector representation. For example for a features space of 300 words, each conversation would be encoded into a vector of 300 counters. We used the text count vectorizer from the *sklearn* Python library to implement this encoding.

1. **FEATURE SELECTION**

­­­­A major challenge in text classification consists on the selection of features due to the rich nature of natural languages. The English language for example contains more than a million words; which represents a very large feature space, and which results computationally infeasible.

*Mutual Information*

In this project we selected a feature selection method based on our need to reduce computational cost, and the specific intention to classify text. We implemented the Mutual Information (MI) feature selector, which measures how much information the presence/absence of a word contributes to making the correct classification decision of a conversation.

The concept of MI is defined in information theory for two discrete random variables as follows:

In the case of text classification, one of the random variables indicates if a document contains a given word, while the other variable indicates if the conversation belongs to a given topic. So for example the random variable takes the value of one if the conversation contains the word, and zero it doesn’t. Similarly the random variable takes the value of one if the conversation belongs to the topic , or zero otherwise. The mutual information is then calculated for each word in the training, for each class. For a total of unique words, we computed mutual information values. Even though the computational cost is high, this feature selection approach allowed us to prioritize the features available.

Chapter 13 in [4] does a good job at describing the algorithm to implement mutual information in order to extract features in a text classification problem. The given equation to calculate MI is represented in terms of counters for implementation purposes.

For example represents the number of conversations where the word is absent, but classified with topic. Similarly represents the number of conversations that were not classified with topic and where the word was absent. Please refer to [4] for a more detailed description of this equation.

Since MI indicates how much information a word contains about a given class, we selected the words with the highest MI values for each class to build the features space of our classifier. Different sizes of the features space were tested.

The following table shows the top 5 words, with descending priority, from each of the classifiers when using the MI algorithm to prioritize features, which are actually making sense for each of these classes.

|  |  |
| --- | --- |
| **Topic** | **Top Five Features** |
| *Author* | book, write, read, author, story |
| *Movies* | film, movie, scene, actor, director |
| *Music* | song, album, band, record, play |
| *Interview* | president, time, say, government, look |

1. **CLASSIFIERS**
2. NAÏVE BAYES CLASSIFIER

We first implemented the multinomial Naïve Bayes classifier using a probabilistic model, which is a supervised learning method. Using our training set we train a multinomial Naïve Bayes to predict which documents falls in which class.

METHODOLOGY

The algorithm works by calculating probabilities. So first we compute the conditional probabilities of a document being in class c, i.e. the probability of being in class c given the following document. To do so we calculate the product of the priors and the product of the conditional probabilities, i.e. the probability of each token of a document given in class c.

The priors are found by computing the sum of samples in class c divided by the sum of samples. Then the conditional probabilities are computed by counting the frequency of each token t in each class and divide by the total number of tokens in each class in the training set. To eliminate having probabilities of zero, we use add-one or Laplace smoothing, which simply adds one to each token count. Using these priors and conditional probabilities we train the Naive Bayes classifier and use it to make predictions.

To be able to predict from the Naïve Bayes classifier we find the best class for each document by maximizing the probability of being in class c given a document d.

Multiplying these conditional probabilities may results in unstable results. Therefore it is better to perform the computation by adding logarithms of probabilities instead of multiplying probabilities. So we sum the log of the class prior and sum of log of conditional probabilities of this class for each word occurring in the sample. We repeat for each of the 4 classes.

So the algorithm works by assigning a 4 class score for each sample and then we assign the class with the highest score to the underlying sample.

RESULTS AND DISCUSSIONS

We use the multinomial Naïve Bayes algorithm to compare accuracy with the number of features. We setup two experiments to explore this relation using different features selection methods. In the first experiment we selected features randomly and compared the accuracy with the number of features up to 13000. In the second experiment we used mutual information methods to select features up to 380. The following plot compares the two methods.

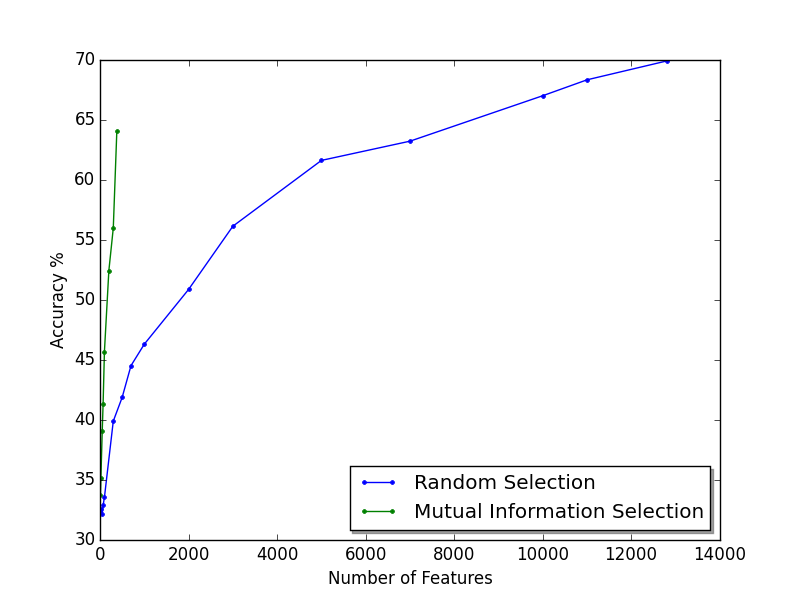


Figure 1 Number of Features vs Accuracy

As expected, the accuracy obtained with mutual information increases faster than the one with random selection. With 300 features for example, mutual information results in 64%, while random selection delivers less than 40%, and only reaches 60% with more than 4000 features.

1. K-NEAREST NEIGHBORS:

METHODOLOGY

K-Nearest Neighbors is a non-parametric method used in machine learning for the task of Classification or Regression. To classify the random example coming out of the Test Set, we find the K examples that are closest to the query point, and we use the voting process to determine the class to which the given example is likely to belong. The choice of K plays the most important role in determining the performance and quality of prediction for K-Nearest Neighbors Algorithm. While the smaller value of K can lead to large variance on Test Data, large value of K may lead to bias on Test Data Set.

In our implementation of K-NN algorithm, we are taking each example from test set, and comparing it with all examples present in training set in order to find common words between test and train sets. For each common word, we are scoring each word for each classifier as follows:

Where refers to total number of examples in the training set, and refers to number of times the word has appeared in the training data. It is quite intuitive to score the classifier like this because it is making sure that the word which appears more frequently in the text gets less importance than the word that appears less frequently in Training Data Set.

We setup an experiment where 90% of data was used to train the Classifier and 10% of Data was used to test the performance of Classifier. In this experiment we ran our classifier on the Training Dataset over various number of neighbors K; and found that the optimal value of K to be 10. With K = 10, we achieved an accuracy of 62.3%. Fig1 shows the relation between number of neighbors K and accuracy.

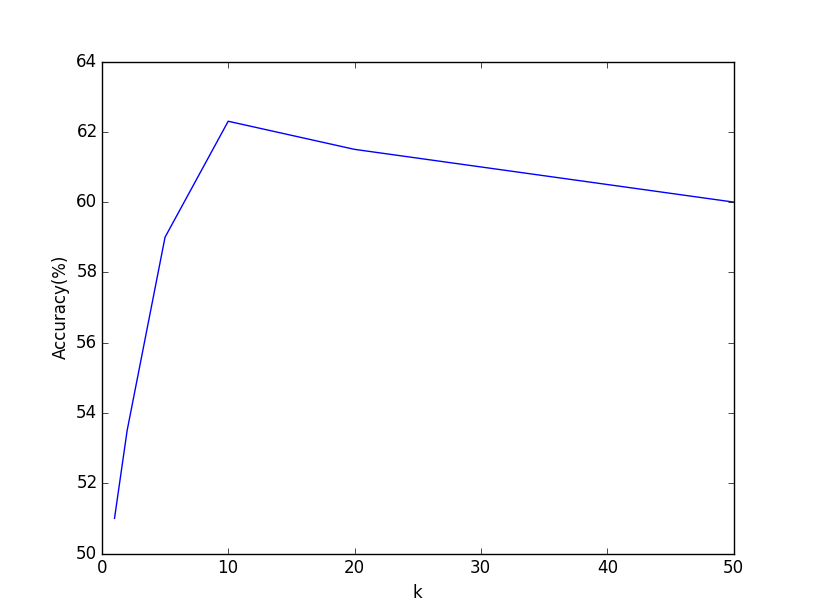


Figure 2 k (number of nearest neighbors) vs Accuracy

*Cosine distance metric Implementation*

While we were not satisfied with our results using K-Nearest Neighbors, We attempted to perform an another implementation of K-nearest neighbors using graphlab module available in Python. This module contains an inbuilt functions for calculating word counts and TF-IDF(Term Frequency- Inverse Document Frequency) and cosine distances for each examples and it also provides a convenient way to handle and visualize the data. We did not perform any pre processing on data in this implementation and fed the raw data to classifier. This classifier gave us accuracy of 74% on Training Dataset (90% of Data was used to train the classifier and 10% of Data was used to test the Classifier) with K =10.

RESULTS AND DISCUSSIONS

While we can increase the accuracy of Classification of text by using 2nd implementation of K-NN using graphlab, but we have to pay a cost for it in terms of computations and time it takes to run the entire Dataset. We also observed that K-NN is slowest among the other Classification Algorithms implemented by us because for each example it calculates the distance from all the examples in Training set and it becomes more cumbersome in the case of Text Classification where we have lots of features and lots of examples in our Dataset. But however, the performance of K-NN is impressive taking into account its simplicity.

1. SUPPORT VECTOR MACHINES (SVM)

A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification. Separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class, the margin, since in general the larger the margin the lower the generalization error of the classifier. This indicates that classes have been well separated. When fitting vectors for multiple classes the training vectors are implicitly mapped into a higher dimensional space by the function.

METHODOLOGY

Support Vector Machines has been implemented with different kernels (Linear, Radial basis (Gaussian), 3rd degree Polynomial). It is recommended to use linear kernels for text categorization, as most of text classification problems are linearly separable. It has the ability to meet high generalization performance without requiring any prior knowledge even with a very high dimensional feature set. The core idea of linear SVM is to find a hyper plane that keeps the maximum points of a class on the same side while maximizing the distance of hyper plane from the classes. This hyper plane thus minimizes misclassifying error in the test set. We will show the results of different kernels and compare it to answer the question if it is worth it to fit more complex kernels. Problem with SVM in python [using scikit-learn runs endlessly and never completes execution](http://datascience.stackexchange.com/questions/989/svm-using-scikit-learn-runs-endlessly-and-never-completes-execution). So we only run it for a small number of features and training set.

Another method that executes faster is SVM with Stochastic gradient descent optimization. It is sometimes favored because of its efficiency and ease of implementation. However it has some disadvantages such that it requires a number of hyper parameters including regularization parameter and the number of iterations. In addition is it sensitive to feature scaling. We have also implemented this method and we will compare it to the previous SVM results.

So we run the SVM with Stochastic gradient descent optimization using sklearn-linear model - SGDClassifier - library with different number of epochs and different loss functions. The epochs are the number of iterations used to find the kernels in the training set and loss functions are the support vector machine kernels. These return different results. We try the following kernels:

The default is hinge, which returns a linear support vector machine. The *log* loss gives logistic regression, a probabilistic classifier. For our data it’s the multinomial logistic regression. We run that for 10, 20, 50 and 100 epochs. Another soft method support vector machine method is *modified huber* brings tolerance to outliers as well as probability estimates. *squared\_hinge* is like hinge but is quadratically penalized.

RESULTS AND DISCUSSION

We run SVM using log loss with different epochs, obtaining the highest accuracy with 50 epochs as shown in the following diagram.

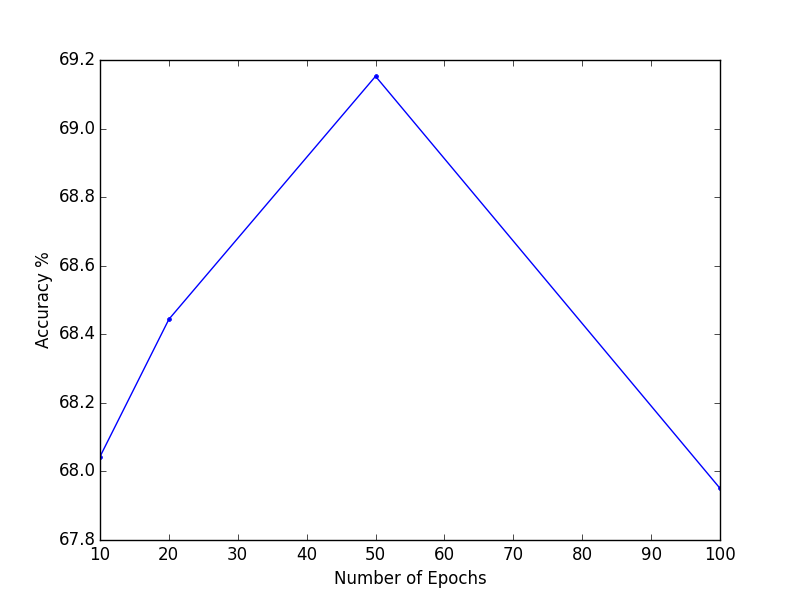


Figure 3 Number of epochs vs Accuracy

As we increase the number of epochs for the default loss (hinge loss) the accuracy increases up to 50 epochs, but it then decreases as we increase the epochs further. We continue using 50 epochs to try to explore other loss functions as shown in the following table:

|  |  |
| --- | --- |
| Epochs | Accuracy |
| 10 | 64.314330 |
| 50 | 64.714946 |
| 100 | 65.546995 |

Comparing the three other loss functions it turns out that modified Huber returns the best accuracy among the three since it’s a soft method but it does not beat the log loss.

|  |  |
| --- | --- |
| Loss functions | Accuracy |
| *Log* | 64.314330 |
| *Modified Huber* | 62.157165 |
| *squared\_hinge* | 51.741140 |

1. **CONCLUSION**
2. **FUTURE WORK**

*Deep Learning*

An alternative to improve the results found in this project could be to consider classification methods that explore the semantics and the temporal aspects of the data provided. The recurrent neural networks for example [5] provide tools to consider the sequential ordering in natural language, as well as the temporal representation through the use of neural networks with memory.

*Enhance Feature Selection*

ACKNOWLEDGMENT

REFERENCES

[1] Vapnik, Vladimir. "The Support Vector Method of Function Estimation." Nonlinear Modeling. N.p.: Springer US, 1998. 55-85.

[2] Joachims, Thorsten. "Text Categorization with Support Vector Machines: Learning with Many Relevant Features." (n.d.): n. pag. Web.

[3] <http://www.scholarpedia.org/article/Text_categorization>

[4] [Christopher D. Manning](http://nlp.stanford.edu/~manning/), [Prabhakar Raghavan](http://theory.stanford.edu/~pragh/) and [Hinrich Schütze](http://www.cis.uni-muenchen.de/personen/professoren/schuetze/), Introduction to Information Retrieval, Cambridge University Press. 2008.

[5] Garen Arevian, ‘‘Recurrent Neural Networks for Robust Real World Text Classification’’, IEEE/WIC/ACM International Conference on Web Intelligence, 2007