Text Classification of Conversations from National Public Radio Excerpts

Daniel Galeano
260623579
McGill University
Montreal, Quebec, Canada
daniel.galeanogranados@mail.mcgill.ca

Timardeep Kaur Arneja
260617404
McGill University
Montreal, Quebec, Canada
timardeep.arneja@mail.mcgill.ca

Sandra Maria Nawar
260665071
Concordia University
Montreal, Quebec, Canada
sandra.nawar@mail.mcgill.ca

***Abstract*--**We are living in a world where we are constantly generating a great amount of data. Automatic Text Classification provides an efficient way of classifying and managing documents whose numbers is continuously increasing. Text classification 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 while comparing the performance of three different classifiers: Naïve Bayes, k-nearest neighbors, Support Vector Machines. **Our team name in Kaggle is timardeep.**

I. INTRODUCTION

Text Classification is a task of assigning categories to different texts, this classification can provide conceptual view of document collection and has important applications in the real world [3]. With the advent of Big Data it has become impractical classify documents 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 the National Public Radio into one of the categories: *author, movies, music and interview*. We implement a bag-of-words method to facilitate the representation of conversations and to perform feature selection. 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.

II. DATA PRE-PROCESSING

As part of the pre-processing we removed stop words and punctuation characters by using Python libraries such as the regular expressions module *re* and the natural language toolkit *nltk*. Manual inspection and manipulation had to be implemented to discard specific stop words such as *__eos__*, which were not discarded by the available Python tools. The conversations were also tokenized based on words, and the characters were set to lower case only.

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.

III. FEATURE SELECTION

One of the major challenges 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 to represent.

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 X, Y as follows:

$$MI(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log(\frac{p(x,y)}{p(x)p(y)})$$

In the text classification use case, 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 X_w takes the value of one if the conversation contains the word w, and zero it doesn't. Similarly the random variable Y_t takes the value of one if the conversation belongs to the topic t , or zero otherwise. The mutual information is then calculated for each word in the training set, for each class. Considering that the training set contained approximately 9.0×10^4 unique words, we had to compute 3.6×10^5 mutual information values. Even though the computational cost is high, this feature selection approach allowed us to prioritize the features available to be able to reduce the feature space.

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.

$$MI(X;Y) = \frac{N_{11}}{N} log_2\left(\frac{NN_{11}}{N_{1.}N_{.1}}\right) + \frac{N_{01}}{N} log_2\left(\frac{NN_{01}}{N_{0.}N_{.1}}\right) + \frac{N_{10}}{N} log_2\left(\frac{NN_{10}}{N_{1.}N_{.0}}\right) + \frac{N_{00}}{N} log_2\left(\frac{NN_{00}}{N_{0.}N_{.0}}\right)$$

For example N_{01} represents the number of conversations where the word X is absent, but classified with topic Y. Similarly N_{00} represents the number of conversations that were not classified with topic Y and where the word X 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 5 words with the highest mutual information values for each class.

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

IV. CLASSIFIERS

A. NAÏVE BAYES CLASSIFIER

We implemented the multinomial Naïve Bayes classifier using a probabilistic model, which is a supervised learning method.

METHODOLOGY

The algorithm works by calculating probabilities. So first we compute the conditional probabilities of a document being in a class *c*. 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*.

$$P(c|d) \infty P(c) \prod_{1 \le k \le n} P(t|c)$$

The priors are found by computing the sum of samples in class c divided by the total sum of samples P(c) = $\frac{N_c}{N}$. 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 as follows P (t / c) = $\frac{T_{ct}+1}{\sum_t T_{tc}+1}$

Using these priors and conditional probabilities we train he Naive Bayes classifier and use it to make predictions. To be able to predict from the Naïve Bayes classifier we find the class for each document by maximizing the probability of being in class c given a document d.

$$c = argmax_c P(c|d)$$

Multiplying these conditional probabilities may results in unstable results and numerical problems. 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.

$$c = argmax_c \left(logP(c) + \sum_{1 \le k \le n} logP(t|c) \right)$$

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.

1. RESULTS AND DISCUSSIONS

We use the multinomial Naïve Bayes algorithm to compare accuracy with the number of features. The train data set was split into train and validation sets with a 9:1 ratio in order to measure the accuracy of our Naïve Bayes classifier. We setup two experiments to explore this relation using different features selection methods. In the first experiment we selected features randomly collected data for different number of features up to 13000. In the second experiment we used the mutual information method to select features and computed accuracy for different number of features up to 380. The following plot compares the two methods.

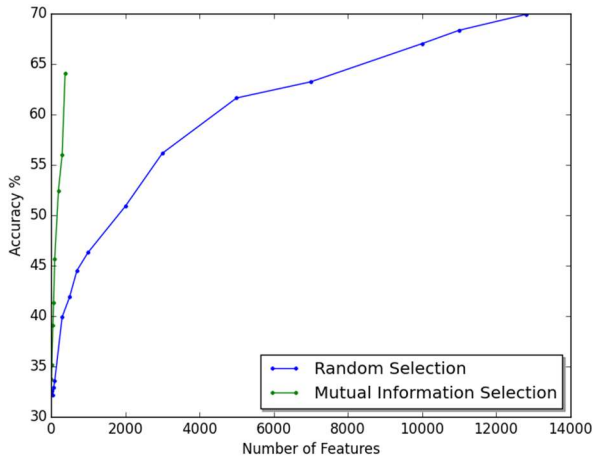

Figure 1 Number of Features vs Accuracy

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% accuray, while random selection delivers less than 40%; and only reaches 60% accuracy with more than 4000 features.

B. 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:

$$w_{score} = \frac{log(N_t)}{1 + f_w}$$

Where N_t refers to total number of examples in the training set, and f_w 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.

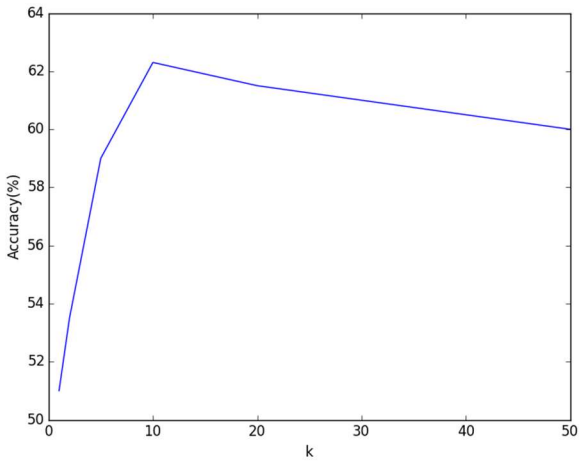

Figure 2 k (number of nearest neighbors) vs Accuracy

Cosine distance metric Implementation

While we were not satisfied with our results using K-Nearest Neighbors, we attempted to perform another implementation of K-nearest neighbors using the graphlab module available in Python. This module contains an inbuilt functions for calculating word counts; TF-IDF (Term Frequency-Inverse Document Frequency) and cosine distances for each examples. We did not perform

any pre-processing on data in this implementation and fed the raw data to classifier.

RESULTS AND DISCUSSIONS

This K-NN classifier using cosine distance metrics gave us an accuracy of 74%. While we can increase the accuracy of Classification of text by using 2nd implementation of K-NN using GraphLab, we have to pay a high cost in terms of computation. 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. The performance of K-NN is still impressive taking into account its simplicity.

C. SUPPORT VECTOR MACHINES (SVM)

A support vector machine constructs a hyper-plane or set of hyper-planes in a high 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. 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 stochastic gradient descent using default loss here 'hinge' loss, which gives a linear SVM with different epochs, obtaining the highest accuracy with 100 epochs as shown in the following diagram. Which makes perfect sense because as we increase the number of iterations the parameters of our support vector gets better optimized reaching the global minimum.

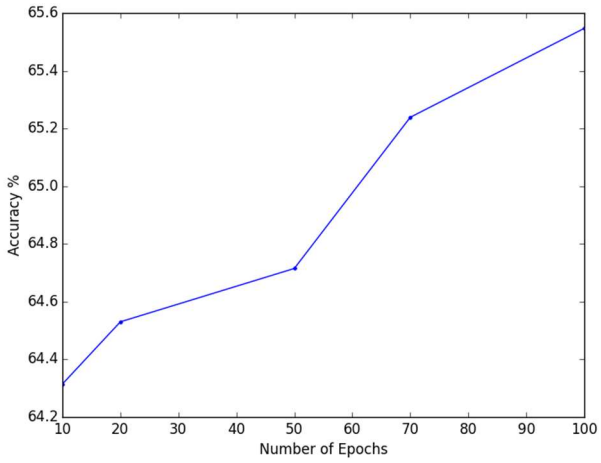

Figure 3 Number of epochs vs Accuracy

Since 100 epochs returns the best accuracy we continue using 100 epochs to try to explore other loss functions as shown in the following table:

Loss functions	Accuracy
Log	64.314330
Modified Huber	62.157165
Squared_Hinge	51.741140

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.

V. CONCLUSION

In this project we explored different text classification algorithms for a given data set. We also implemented feature selection methods to improve both the accuracy and computational feasibility of our solutions.

The following table shows the best accuracies for each of the algorithm attempted in Kaggle.

Kaggle Results	
Classifier	*Accuracy*
K-NN	0.48884
Naïve Bayes	0.59398
SVM	0.59060

Our results show that our Naïve Bayes classifier generated the highest accuracy among the algorithms that were tested. This results does not correlate with other studies like the one obtained by Joachim [2], where SVM has the best accuracy. These results are also unexpected and disappointing because the Naïve Bayes classifier makes the assumption of independence within the features, which is not applicable in a natural language context. In the case of K-NN we had many computational challenges that did not allow us to attempt higher number of neighbors or more complex ways to measure distance.

In terms of feature selection we tested our mutual information algorithm and successfully showed that it can deliver higher accuracies with less number of features compared to random selection.

VI. 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

Other methods that can be explored for feature selection, which is the case of principal component analysis (PCA) for example, which would consider all the features together, and which could better represent the complex natural language connections between words.

I. 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, Prabhakar Raghavan and Hinrich Schütze, 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