Project 2: Text Classification of Reddit Conversations

Team Name on Kaggle: EquiTeam

Rohan Pradhan *(260613559),* Ariane Schang *(260553025),* and Zachary Warnes *(260581958)*

I. Introduction

Given a dataset of categorized conversations on Reddit, an online public forum, we were tasked with building a classifier that could decide the topic of the given conversation. In order to solve this topic modeling problem, we implemented a k-Nearest Neighbor algorithm, a Naive Bayes model, and a Support Vector Machine. We also used the sci-kit learn library to implement a variety of other models. Of those models, the Multinomial Naive Bayes implementation resulted in the highest accuracy we saw, at 95.4 percent correctly predicted topics.

II. Related Work

Text classification has a wide range of applications from an early start in classification in email spam detection to one of its current uses in the live analysis of social media platforms to calculate public perception. Through natural language processing, text classification has played a vital role in the field of artificial intelligence as exhibited recently by Microsofts TayTweets [1]. There are many statistical procedures that can be used to classify text including K-Nearest neighbors Support vector machines TF-IDF/ Rocchio, Converging algorithms and the exponential gradient technique. In general, research has proven that using the nave Bayes classifier is the most efficient on large data since it is based on the probabilistic principals of expectation maximization [3]. There are many techniques to extract information from text data such as n-grams, tokenizing, and bag of words [2]. Through this paper we describe various techniques in text classification and their implementations in order to categorize text into several different categories.

III. Problem Representation

We preprocessed the information in multiple ways. First, we removed all the "XML-esque" tags from the conversations. This was done using the cleanHTML package for Python. Once the tags were removed, we used the PorterStemmer to stem on all words. Originally, we had tried using WordNetLemmatizer to lemmatize the words based on their word and part of speech. However, using the WordNetLemmatizer only resulted in a very modest improvement in accuracy, while taking significantly longer to process. Thus we decided to only use the PorterStemmer.

For the K-Nearest Neighbor and Support Vector Machine classifiers feature design and selection, we used the CountVectorizer from SciKit learn to create a vectorized representation of the words. We then fed this into to the TfIdfTransformer from SciKit learn to fit the data a TfIdf representation to improve accuracy and remove noise. It is important to note, that in the actual implementation, we simply used the TfIdfVectorizer which combines the above two steps into one step. Additionally, inside the TfIdfVectorizer, we specified to remove stop words from the English corpus and put all characters in the input to a lowercase format.

We decided which features to use by ranking by TfIdf values of all the various features. We selected the top 550 features which had the highest cumulative TfIdf weight values.

For the Naive Bayes implementation, we initially used the whole vocabulary and represented the features with a frequency table for each conversation. As will be explained later, any words that occurred less than 20 times were removed from the vocabulary as well as words that appeared in more than 1/3 of the articles.

IV. Algorithm Selection and Implementation

A. Naive Bayes

Due to the large size of the data set, cross-validation was deemed unnecessary. The data was therefore split into a train and validation set of size ratio 80:20. This left 132,000 samples in the training set, and 33,000 samples in the validation set. The testing set had already been separated from the data we were given. In the interest of time and for the classifier to finish training in a reasonable time frame, we ultimately had to use a subsample of the training set. Prediction time for the Naive Bayes classifier is very rapid, so I kept a large validation set of 33,000 samples. However, I sub-sampled the training set to 30,000 training samples.

In order to create a Naive Bayes implementation, the data was vectorized using each word as a feature. The words were stemmed so as to decrease the total number of words in the vocabulary, and to associate words whose meanings are the same. We used the PorterStemmer from the NLTK library. A frequency table was created which used each word in the whole vocabulary and counted the number of times it occurred in each instance of the training set. From this frequency table, the log-probabilities of belonging to each class were calculated for every word. When predicting new samples, these probabilities were summed over each word in the conversation. The maximum probability was then used as the correct answer.

Experiments were run to decide which minimum occurrence value was optimal. That is, if a word only shows up in less than a given number of articles, it is removed from the vocabulary. This is done to decrease the size of the feature space by removing words that don't often occur. This testing was done over a training set of 20,000 samples and used over the whole validation set (Fig. 1). The optimal value was found to be removing words that occurred less than 20 times, so I used this with all further testing. I also removed the words that occurred

TABLE I. MOST DESCRIPTIVE WORDS FOR NAIVE BAYES

Category	Word	Log Probability
Hockey	*game*	-4.31
Movies	*movi*	-3.5932
NBA	*game*	-4.5365
News	*peopl*	-4.75313
NFL	*team*	-4.42262
Politics	*peopl*	-4.63123
Soccer	*player*	-4.6631
World News	*peopl*	-4.8668

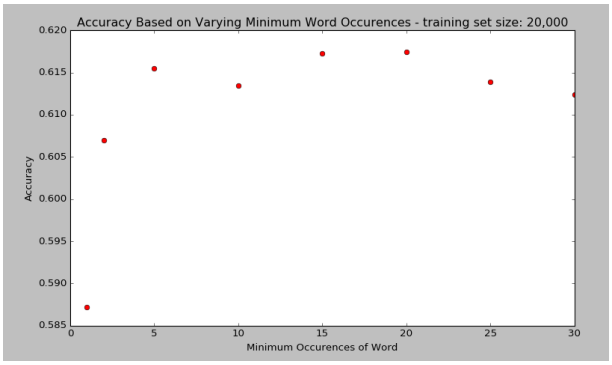

Fig. 1. Accuracy based on varying minimum word occurence.

in more than 1/3 of all articles, as these were deemed to be non-descriptive. Stop words were also removed.

The highest probabilities for each word category, or the most descriptive word feature is listed in Table 1, with it's log probability. All these words are, intuitively, strongly correlated to their categories. However, we can already see that the classifier may have a hard time differentiating between World News, Politics and News as it has the same most important word in each category.

The final feature space had 5,734 features over 30,000 training samples.

B. K-Nearest Neighbors

In our implementation of the k-Nearest Neighbors classifier, we were required to make many design decisions to allow for proper scalability, within reason, of our classifier. The main issue with the k-Nearest Neighbors classifier is that it is very computationally expensive, and therefore is very difficult to scale properly. In a simplified representation, our classifier's run time efficiency was dependent on the number of training examples, the number of validation/testing points, and the number of features we were using. Therefore, our k-NN classifier had a run-time efficiency of $O(n * m * d)$, where n represents the number of testing/validation points, m represents the number of training examples, and d represents the dimensionality of the feature space. Because of this, we had to be very careful in our implementation of the classifier, and the amount of data we passed into the classifier.

As discussed in our problem representation, for K-NN, we preprocessed the data using a TF-IDF Vectorizer. This vectorizer creates a sparse matrix of each relevant word and associates it with a respective weight which is dependent on the inverse document frequency of that word. As we ran our classifier with larger and larger training sets, the size of the matrix grew very quickly. This caused two problems. First, as the size of the matrix grew, this meant that our feature space of words also grew. The problem with this was, as the feature space became very large, a tremendous amount of noise was introduced, causing our results to become more inaccurate. Secondly, a very large matrix also slowed down our algorithm to an unacceptable point. The solution to this was to limit our feature space. We created a function to take in the sparse matrix from the TF-IDF Vectorizer, and only use the top X features based on the inverse document frequency. This meant we were only using features that held tremendous weight, thereby reducing the noise of irrelevant data. After some trial and error, we decided that limiting our feature space to the best 550 features yielded the best results.

However, even by limiting our feature space to 550 features, our algorithm still performed very slowly and could not handle any data set with a relevant size. This made sense since, the more features we used, the higher the dimension of matrix and test vector. This was a significant bottleneck while calculating the distances to determine the best k-nearest neighbors. Therefore, to make our k-NN classifier tractable, we had to significantly reduce the dimensionality of our feature space. Simply reducing the number of features we used from 550 did not work, since the accuracy severely suffered. Thus we decided to use some sort of decomposition technique to map the higher dimensional feature space onto a lower dimensional feature space. We tried two different decomposition techniques, Principal Component Analysis (PCA) and Truncated Singular Value Decomposition (TSVD). Originally, we tried the PCA technique to reduce the dimensionality of the feature space from 550 dimensions to 25 dimensions. PCA works by finding the "principal components" (important data points) in the data and then projecting this important data into a lower dimension. It does this through the use eigenvectors and eigenvalues. This however gave us problems since the matrix we were passing in for PCA was extremely sparse. This was problematic since PCA tries to center the data using the covariance matrix before computing the decomposition. However, with large sparse data sets, the amount of memory needed to do this accurately increases very quickly. Due to this, the initial results we got from using PCA to reduce feature space dimensionality were very poor. To solve this, we decided to use TSVD instead. TSVD is very similar to PCA, however, it does not do it's computations on the covariance matrix, rather on the input matrix directly. Because of this, TSVD allows for sparser, larger matrices. However, when we just used TSVD, our results were still quite inaccurate. Therefore, we decided to normalize the results of the TSVD, which drastically improved our accuracy and performance.

$$X \approx X_k U_k \sum_k V_k^T$$

SVD being applied onto input feature set to create lower dimension approximation of X

Once we had reduced the dimensionality of our feature

set, we had to calculate the distances between the vectors to determine the best k-nearest neighbors. We tried 4 different distance formulas including: Euclidean Distance, Manhattan (CityBlock) Distance, Minkowski (3rd degree) Distance, and the Cosine Distance. Ultimately, the cosine distance proved to provide the most accurate results.

$$cos(x, y) = 1 - \frac{x \cdot y}{||x||_2 \cdot ||y||_2}$$

After calculating the distances, we had to decide the number of neighbors, k, to set in our classifier. We began trying out with k=5, however, our results were not generalized enough. Therefore, we ultimately decided to set k=7. By setting k=7, we generalized our results well enough to avoid over-fitting, but also prevented overgeneralizing.

As for our train/validation split, we ultimately had to limit the amount of data we used to allow for the classifier to complete in a realistic amount of time. Even with the dimension optimizations, running our k-NN classifier with sets larger than 50,000 entries resulted in running times that were too long in our situation. We therefore decided to use a data set of 50,000 entries in our classifier. After analysis of the data, we saw that the data was very well distributed. Therefore, we used a 90/10 train/validation split on our classifier. This meant we "trained" on 45,000 entries, and validated on 5,000 entries.

C. Support Vector Machines

IMPLEMENTATION

The focus of our original SVM implementation was on the optimization of the primal form.

$$f(w, \beta) = (1/m) \sum_i (1 - y_i(w^T x_i - \beta)_+ + \lambda ||w||_2 \quad (1)$$

where λ is a scalar and represents the regularization parameter with $\lambda \geq 0$. In this instance, L2 regularization was used to help with over-fitting the data. The representation above also makes use of an average of the hinge loss. Unfortunately, our implementation provided sub-optimal results and a sklearn implementation were used to achieve the results of the table to follow. This version was tested using an incredibly high cost to simulate infinity and an optimized gamma to find the ideal margins. Regarding prediction, one vs. all prediction was used which isolates each of the classes against all others.

FEATURES

A vectorized approach was used for creating input data for the SVM similar to that of the kNN method of feature selection described. With such a large feature space the weights returned from the primal method in our initial representation are small in magnitude suggesting that any one word from the feature space plays a minute role in the classification of the test data. One should also consider the number of classes that we are trying to predict since if the data is assumed to be i.i.d. then for each 1 vs. all classification, the problem becomes binary and potentially skewed.

TABLE II. ACCURACY OF NAIVE BAYES IMPLEMENTATION ON VALIDATION SET

Category	Precision	Recall	f1-score
Hockey	0.93	0.93	0.93
Movies	0.96	0.94	0.95
NBA	0.94	0.95	0.94
News	0.71	0.70	0.70
NFL	0.93	0.94	0.94
Politics	0.79	0.79	0.79
Soccer	0.94	0.96	0.95
World News	0.81	0.82	0.82
Average	0.81	0.82	0.88

D. Other Methods

Using the Sci-Kit Learn library, many other classifiers were applied over the data. Because these classifiers were much quicker than the ones we implemented, we also included varying ranges of n-grams of the vocabulary. To do this, the CountVectorizer tool was used to transform the data into a sparse matrix, and the TfidfTransformer tool was used to create a probability matrix for each word. This data was then used with many different algorithms, all implemented in the SKLearn Library. Varying n-gram ranges were used, from only single tokens, to including all n's smaller than 6. The optimal value for n was 4, so this value was used in all the classifiers. Various values were used for minimum and maximum occurrences to be included in the classifier. The optimal values were found to be a minimum of 2 and a maximum of one half of the documents. The algorithms used were: Multinomial Naive Bayes, Linear SVM , Logistic Regression, and a Random Forest Classifier. The Linear SVM uses the Stochastic Gradient Descent with a 'hinge' loss function, l2 regularization, and an alpha of 10xe-3. The Logistical Regression uses the SAG solver. SAG was used for the logistical regression as it is a multi-class problem.

V. TESTING AND VALIDATION

A. Naive Bayes

The Naive Bayes results are listed in Table 2. The total, final accuracy is 87.6 percent which represents a relatively accurate prediction. We can see the final score heavily brought down by the news, world news and politics predictions. This was expected considering the most discriminative word for these three categories were the same. With an f1-score of 70 percent, 82 percent, and 79 percent (respectively), the classifier is still managing to differentiate between the various types of news. Movies and Soccer both had very high results with 95 percent f1-scores. This is not unexpected as movies has a very unique vocabulary compared to the rest of the categories.

Using a smaller training set may have compromised the accuracies, but enough time was saved that it seemed to be a reasonable compromise given the accuracies.

B. K-Nearest Neighbors

The overall accuracy of our K-NN was approximately 52%. The category that had the highest accuracy was the MOVIES category, that had an approximate accuracy of 81%. The category that had the lowest accuracy was the NBA category,

TABLE III. ACCURACY OF K-NN IMPLEMENTATION ON VALIDATION SET

Category	Accuracy
Overall	0.5166
NBA	0.286231884058
NFL	0.444089456869
Hockey	0.396496815287
Movies	0.811178247734
News	0.466772151899
Worldnews	0.517087667162
Politics	0.599326599327
Soccer	0.571879936809

that had an approximate accuracy of 29%. In general, we believe that the accuracy was quite low because of the limited data set that we could use (due to time constraints).

As discussed before, the K-NN algorithm is extremely computationally inefficient. Because of this, we were forced to create many optimizations/workarounds to allow for a reasonable computation time. Even with this, we were forced to severely limit the data-set we trained and validated with due to our time constraints. We presume that one of the main reasons for our relatively low accuracy rate was the use of decomposition techniques to reduce the dimensionality of our feature set. By reducing our feature set all the way from 550 dimensions down to a reasonable 25 dimensions, we believe that some information may have been lost or skewed.

Additionally, we were limited in the amount of pre-processing we could do in a reasonable amount of time. Originally, we were planning on using a variety of techniques to harvest deep relationships in the text. These techniques included using a WordNetLemmatizer to lemmatize words based on context instead of individual word stems or using N-grams to see the relationships of contiguous sequences of words. However, both these techniques significantly slowed down the pre-processing phase. Additionally, when we tested these techniques on smaller data sets, we did not see a drastic improvement in accuracy.

In terms of our actual results, our results make sense. This can be seen upon further examination of our confusion matrix for the k-NN implementation. For example, in the NBA category, the classifier predicted the the NBA category the most out of all other categories. This is expected. Interestingly, if you look at the misclassifications, the majority of misclassifications occur within other categories pertaining to sports (NFL, hockey, and soccer). This again makes sense, since when the k-NN algorithm works it finds the closest distances to to other similar points. This means that all the categories related to sports probably have similar common words, which is to be expected. This phenomena occurs also within the categories of ”Worldnews”, ”News”, and ”Politics”. Again, this is to be expected since all three of these categories are quite similar, and could have very similar common words between all of them. This also explains why the ”Movies” category has such a high accuracy relative to the other categories. No other category is very similar to the ”Movies” category, hence there is a low misclassification rate.

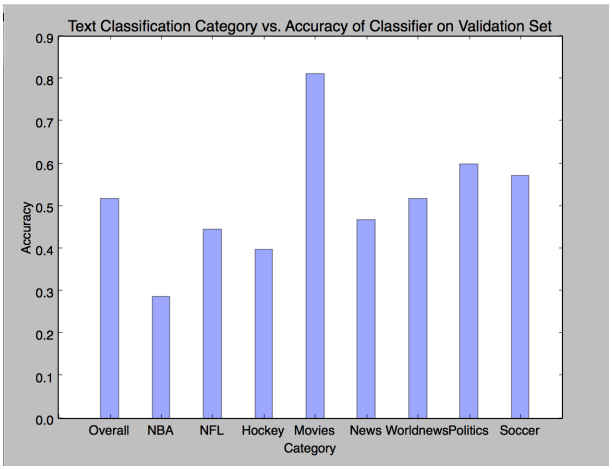

Fig. 2. Accuracy of k-NN classifier for each category.

C. Support Vector Machines

Due to large nature of the data provided only around 6000 examples were used for training. The following table represents the effectiveness of three different kernels on the training set.

Averages	Linear	Gaussian
Precision	0.80	0.81
Recall	0.73	0.36
f1-score	0.74	0.36

Each kernel performed relatively well and results improved with larger training sets. The Gaussian kernel did have a lower recall is most test suggesting a tendency to predict negative outcomes for each test. The lower F1-score is potentially a direct result of the difference in recall.

OPTIMIZATION

Optimization for the original SVM implementation was done using the cvxpy library which offers modules to maximize or minimize specifically convex functions. This restricts the type of quadratic programming problems present when solving for parameters in SVMs. This basic package requires programming problems to be in Disciplined Convex Programming (DCP) form, this form forces functions to be optimized to have known curvature and from a set of base functions.

In particular, versions of the dual form of this problem are not convex and require additional packages. cvxpy offers a package extension which can be used to calculate an approximation to the optimization of the dual by adding the constraint

$$\sum_i \alpha_i <= C$$

where C is some to be selected. Without this additional constraint, this form remains an unbounded concave function. For the original implementation, the focus was directed to the primal form of the SVM which is a convex function and can be implemented with relative ease using the basic cvxpy modules. However, there is an immediate benefit of using the dual since one can make use of the Kernel trick. This allows

TABLE IV. CONFUSION MATRIX FOR K-NN IMPLEMENTATION

	NBA	NFL	HOCKEY	MOVIES	NEWS	WORLDNEWS	POLITICS	SOCCER
NBA	158	102	121	32	33	13	10	83
NFL	87	278	111	39	19	14	8	70
HOCKEY	77	86	249	50	22	13	13	118
MOVIES	10	8	16	537	26	32	9	24
NEWS	4	8	8	36	295	128	136	17
WORLDNEWS	1	6	11	38	135	348	105	29
POLITICS	4	1	9	13	131	74	356	6
SOCCER	39	68	66	30	29	32	7	362

mapping of the input data to higher dimensional feature space and may allow for more detailed separation across some higher dimension. Due to the complex nature of the input data, neither linear nor linear kernels were particularly more effective as shown in the table. This technique also helps to decrease the time used when compared to complex feature space mappings. The choice was made to focus on a Gaussian kernel, or RBF kernel, which leads to an infinite-dimensional representation of the input data.

D. Other Methods

All the accuracies reported in Table 3 were found using half of the training set to train the data and the whole validation set to find the accuracies. For the submission on Kaggle, we ran the two top ranking classifiers over the entire training set. These were the Multinomial Naive Bayes and the Logistical Regression. The final accuracies for these two over the entire training set were: 0.95671 for the Naive Bayes, and 0.94851 for the Logistical Regression. As expected, the SKLearn implementations were far more accurate and time efficient than our own implementations.

TABLE V. ACCURACY OF EACH SKLEARN ALGORITHM ON VALIDATION SET

Algorithm	**Accuracy**
Multinomial NB	0.935546
Linear SVM	0.9141
Random Forest	0.842879
Logistical Regression	0.9206

VI. DISCUSSION

A. Naive Bayes

The Naive Bayes classifier was in both our own implementations and in the SKLearn implementation, the highest-performing classifier. The simplicity of its implementation makes it a very attractive classifier. Given more time and if our implementation was trained using the entire data-set, it's likely the accuracy could have increased. One of the disadvantages of this classifier is that it can make absolutely no prediction for words that do not appear in the training set.

B. K-Nearest Neighbors

As discussed extensively in this report, the k-NN classifier is a computational expensive, lazy learner. The benefit of this is that it is simple to implement the raw implementation and easy to follow. However, because of this, it becomes very inefficient, and it is tough to make it a tractable classifier for larger data-sets. This is where using k-NN can become complicated. To use k-NN for a realistic application with a large data set, either one has to have a very powerful machine or significantly optimize the solution. We had to elect for the later.

Because we did not have access to extremely powerful machines, we were forced to extensively optimize our implementation of the k-NN classifier. Therefore, a pro of our methodology and approach was that we created a fairly optimized version of the k-NN algorithm. Even though we still could not use the entire dataset, the optimizations allowed for a significantly higher amount of data to be used given our time constraints.

While our approach was optimized, specifically through the use of decomposition techniques to reduce the high dimensionality of the feature set, our approach limited accuracy. As discussed in earlier sections of the report, originally, we had plans to do tremendous pre-processing including using N-grams and Lemmatizers. This was not possible, since our approach elected for a more optimized approach, thereby increasing speed but also removing the option to use complex pre-processing techniques. Additionally, to reduce the number of dimensions in the feature set, we used Truncated Singular Value Decomposition. While this significantly speeded up the classifier, some information may have been lost or skewed. Therefore, our accuracy may have been affected to due to this optimization. In whole, while our approach for the K-NN did include many optimizations, in the future, we believe that we should of struck a better balance between the amount of optimizations and the raw accuracy of the classifier.

C. Support Vector Machines

Support Vector Machines offer an intuitive method for separating data for classification; they offer detailed information regarding the structure of each class. Problems only arise with the fine tuning of many different parameters which highly depend on the construction of the particular SVM. In our problem, the emphasis was placed of the high dimensionality of the data set and as a result, the focus was on combinations including kernels and although analysis showed no specific outlier the Gaussian kernel was favored for reasons explained above. Further investigation into different kernels may be an area for improvement, specifically; a string kernel may have proven extremely useful for the problem described. Furthermore, a larger test set could have been attempted during training to improve the accuracy of the classifier. Where SVMs start to waver in their efficacy is when the data set is highly

skewed, margins may become too large and have a detrimental effect on the results. Also, problems arise during multiple class classification, essentially this comes back to the problem with skewed data where the potential to classify data incorrectly increases. Ultimately, support vector machines offer a diverse set of representations can be used in vastly different implementations and are quite robust as a classification method.

VII. Statement of Contributions

A. Rohan Pradhan

Rohan implemented, validated, and tested the K-Nearest Neighbor Classifier. He also implemented the pre-processor and feature selector using a normalized TfIdf Vectorizer used in the K-Nearest Neighbor and SVM Classifiers.

B. Ariane Schang

Ariane implemented, validated, and tested the Naive Bayes Classifier. She also implemented, validated and tested the SKLearn implementations.

C. Zachary Warnes

Zachary implemented, validated, and tested the SVM Classifier.

We hereby state that all the work presented in this report is that of the authors.

References

[1] Neff, G., Nagy, P. (2016). Automation, Algorithms, and Politics— Talking to Bots: Symbiotic Agency and the Case of Tay. International Journal Of Communication, 10, 17. Retrieved from http://ijoc.org/index.php/ijoc/article/view/6277

[2] Ostendorf, M. (2005). Text Classification by Augmenting the Bag-of-words Representation with Redundancy-compensated Bigrams.

[3] Nigam, K., Mccallum, A.K., Thrun, S. et al. Machine Learning (2000) 39: 103. doi:10.1023/A:1007692713085

[4] https://docs.scipy.org/doc/scipy-0.18.1/reference/tutorial/optimize.html

[5] https://www.cs.cmu.edu/ ggordon/SVMs/new-svms-and-kernels.pdf

[6] http://stackoverflow.com/questions/33728518/cvxpy-error-cannot-multiply-two-non-constants-while-solving-svm-dual