Project Title: Subreddit Prediction from Reddit Comments using Naive Bayes, Linear Regression, and Support Vector Machine
Final Project Report for CS 175, Winter 2016
List of Team Members:

Vigen Amirkhanyan, 36492346, vamirkha@uci.edu
Nicholas Morris, 65163539, npmorris@uci.edu
Munish Juneja, 82377245, mjuneja@uci.edu

1. Introduction and Problem Statement

Our project is intended to classify Reddit post comments into subreddits. Naive Bayes, Logistic Regression and Support Vector Machine are the main algorithms being used to implement the classification. We analyze a huge repository of data collected which contains comments for the entirety of reddit spanning one month's time. Our analysis focuses on extracting a comment meaning and purpose without the use of a human, which would be excessive amounts of time to process manually, which is then classified as belonging to a particular subreddit. Classified comments can help Reddit understand its user base's opinions more precisely. Learning from users' comments, Reddit can use this methodology to recommend related subreddits to its users. A predictive comment category feature could classify a comment into a subreddit as the comment is being typed. This feature can assist the user in posting more relatable comment to that specific post.

2. Related Work

At Cornell University research was done to categorize public comments (Purpura, Cardie & Simons 2008). They used numerous classifiers including Naive Bayes Classifier. The goal of the research was to compare classifier's performance. A conclusion was made that active learner algorithms produce better precision results than non -active learner Algorithms.

Moreover, at Carnegie Mellon University a text categorization experiment was done where Logistic Regression and SVM were used (Zhang, Jin, Yang & Hauptmann 2003). Although, the computer scientists used a modified version of Logistic Regression where Conjugate Gradient was used for optimization. The optimization was done to approximate to SVM results as close as possible. Their hypothesis came true after the experiments were done. The modified LR came very close to SVM in multi class document classification.

3. Data Sets

We located a comment repository of Reddit comments that we used for this project. The dataset was collected over the period of a month by Reddit user u/Stuck_In_the_Matrix and contains roughly 53 million comments in a 32GB file provided in the JSON format. Each object contains useful attributes such as a fulltext copy of each comment, the sum of positive and negative user votes, and the subreddit (label) that the comment was posted in. We will go into more detail on how we will process this data in the Proposed Technical Approach section. All of this data was collected by Reddit user Stuck_In_the_Matrix for a Natural Language Processing exercise. It is available as a .torrent for public use. A preliminary analysis (looking at the first 1,000,000 comments) of the dataset reveals some interesting statistics: [{Comment length (in words): Average=156, Minimum=0, Maximum=10320}, {Comment score (sum of positive and negative votes): Average=5, Minimum=353, Maximum=4716}] Of the entire dataset the following subreddits (labels) were the most common: [AskReddit, nfl, funny, leagueoflegends, pics, worldnews, todayilearned, DestinyTheGame, AdviceAnimals, videos]. In order to reduce the large dataset into a more manageable size we have filtered comments down to comments only existing in these most common subreddits. See Supplemental document Figure 1 for a graphical illustration of subreddit comments distribution.

4. Description of Technical Approach

Our pipeline is composed of the following four sections described below, see Supplemental document Figure 2 for a graphical overview of the pipeline:

Common (data processing) pipeline:

This section of the pipeline precedes each of the other pipeline sections and handles formatting the data into representations that are usable by the classifiers.

1. Load JSON objects from file and initialize Comment class instances with the JSON data.
2. During Comment instantiation, process comment text body by converting all characters to lowercase, removing punctuation, and stripping all leading and trailing whitespace leaving only text that has relevant vocabulary.
3. A DataSet object is initialized with a list of all JSON objects which have been converted to Comment instances. The DataSet object will automatically remove all Comment instances which have been indicated as deleted as there would not be any useful vocabulary remaining as the comment body will have been removed. The given list of Comment instances will then be randomly shuffled to ensure that distribution of Comments is normalized and will not favor any specific subreddit.
4. A TfidfVectorizer is fitted using a list Strings of each Comment instance’s processed text body. The TfidfVectorizer initially creates an internal bag of words representation, this representation is then used to create a term frequency inverse document frequency representation. A vast majority of our features will be generated here.
5. A DictVectorizer is fitted using a list of Dicts of each Comment instance’s hand chosen features. We extracted the top bi-grams for each subreddit. See SUpplemental document Figure 3 for a list of top bi-grams.
6. These fitted vectorizers are then used to transform the training and validation data to create sparse matrices for each vectorizer. These sparse matrices are merged row by row and the features are combined for each row.

Main evaluation pipeline:

This section of the pipeline is used to evaluate the performance of the classifiers we implemented for this project. It depends on the common (data processing) pipeline for generation of n cross validation sets.

1. A function, evaluation, accepts a list of classifiers that should be evaluated on the given dataset. For each classifier a 5 cross validation sets of training and validation data are

generated. Each classifier is fitted on the training sparse matrix and its associated labels. The sparse matrix that is used for fitting is created by the DataSet class fit_transform.

2. The average accuracies for each classifier across the 5 cross validation evaluations is computed using the classifier score method which accepts the testing sparse matrix and its associated labels. Currently we are evaluating MultinomialNB, LogisticRegression, LinearSVC and DummyClassifier (using the “most_frequent” strategy which will always predict the label which is most frequent).

This approach is a refinement of what we had done previously. Rather than repeatedly generate sparse matrices for each classifier we refactored our pipeline to more efficiently generate and merge the sparse matrices for each n cross validation set then reuse the sparse matrices each time we needed to train a classifier. This reduced the amount of time required to evaluate our pipeline while also simplifying our codebase. We were also able to further simplify our codebase by taking advantage of the fit and score functions from the ClassifierMixin base class that the classifiers we employ inherit from. As a result we are able to easily swap classifiers in and out of our testing pipeline.

Human testing pipeline:

This section of the pipeline was used to evaluate the classification performance of the human members of our group. This classification accuracy provides an interesting baseline accuracy against which we compare the machine learning classifiers. It depends on the common (data processing) pipeline for generation of human usable text and subreddit labels.

1. A function, human_test, accepts a list of unprocessed comment bodies with their associated subreddit labels are retrieved from the DataSet. The user is presented with the unprocessed body for each of these comments and is asked to choose which subreddit they think the comment came from. The percent of correct choices is returned.

Hyper-parameter optimization pipeline:

This section of the pipeline was used to reveal the ideal hyper parameters that could be used for the different classifiers we employed in the main evaluation pipeline. It depends on the common (data processing) pipeline for generation of training and testing sets.

1. A function, optimize_params, accepts set of training and testing sparse matrices are generated from the DataSet (75% training data, 25% testing data).
2. For each classifier we are evaluating we build a dictionary that contains the relevant hyper parameters for that classifier. The key being the name of the hyper parameter and the value being the range of values that will be evaluated.
3. An exhaustive GridSearch is performed that will return the combination of hyper parameters that performs best on the given training and testing data sets.
4. These hyper parameters are then used during evaluation of the classifiers in the main evaluation pipeline.

5. Software
The following software was used in the pipeline described above. A majority of the library code used was from sklearn (classifiers and vectorizers). We also utilized text processing libraries from nltk.

Classifiers:
input: training scipy.sparse matrices
output: test set accuracy

- sklearn.naive_bayes.MultinomialNB
- sklearn.linear_model.LogisticRegression
- sklearn.svm.LinearSVC
- sklearn.dummy.DummyClassifier

Vectorizers:
output: test set accuracy

- sklearn.feature_extraction.text.TfidfVectorizer
    - *input*: list of text
- sklearn.feature_extraction.text.DictVectorizer
    - *input*: list of dictionaries

Text processing:
- nltk.tag.PerceptronTagger
    - *input:* word
    - *output:* part of speech
- nltk.corpus.stopwords
    - *input*: requested language
    - *output*: language's stop words

Miscellaneous:
- sklearn.grid_search.GridSearchCV
    - *input*: classifier, parameters
    - *ouput*: best parameters
- scipy.sparse.hstack
    - *input*: n scipy.sparse matrices
    - *output*: combined scipy.sparse matrix
- nltk.collocations.BigramCollocationFinder
    - *input*: iterable list of strings
    - *output*: top n bigrams

6. Experiments and Evaluation

Two baseline evaluations were done to provide an interesting contrast to the results we received from the classifier evaluations. Human classification was done by all three members of our teams, this is an interesting classification as it demonstrates the difficulty of classifying comments to an appropriate subreddit given the limited information per comment instance. A DummyClassifier(strategy=”most_frequent”) is included in sklearn, we used the strategy most_frequent which will always predict the label of the most common label included in the test set, this is an interesting classification as we can see the accuracy of simply predicting the most common subreddit.

Early on, our team made an assumption that we could filter comments less than a certain threshold as we intuitively believed that comments which are longer would contain vocabulary that is more representative of the subreddit it belongs to. Our initial results justified this assumption as it improved performance of MultinomialNB and LogisticRegression. Later we added LinearSVC and discovered that although it generally outperformed the other classifiers it performed more poorly on the filtered dataset. As it was our most performant classifier we decided to abstain from filter the data so that our performance on our most accurate classifier would improve.

The following tables and charts were generated from experiments that were conducted over the course of the quarter. They reflect the improvements made as our pipeline implementation improved.

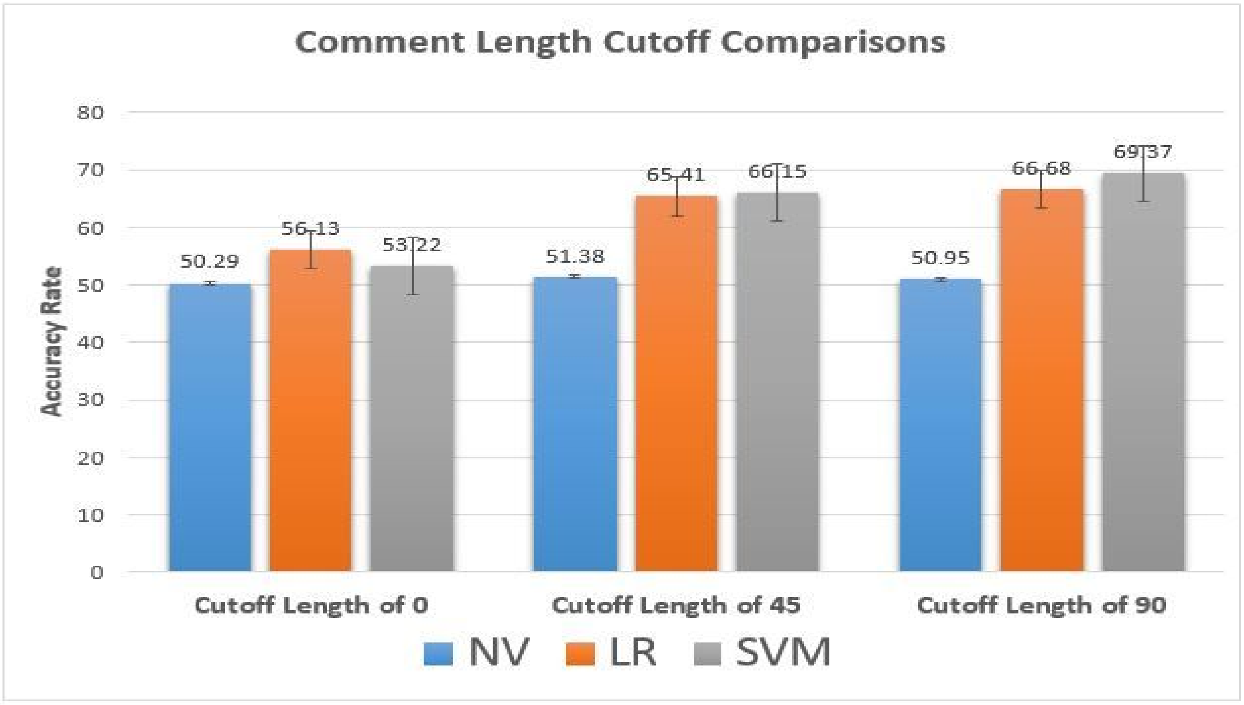
Plaintext			
Sample size	Length cutoff	Filtered size	Human accuracy
20	0	20	22.00%
100	0	100	16.00%

Human classification of the given input data was more difficult than we had anticipated. Vocabulary trends proved to be more difficult to recognize than we had at first imagined. This made us feel better about the accuracy that our classifiers were able to achieve.

Plaintext features (Tfidf) with default hyper parameters						
Sample size	Length cutoff	Filtered size	MultinomialNB accuracy	LogisticRegression accuracy	LinearSVC accuracy	DummyClassifier(strategy=”most

						_frequent") accuracy
100000	0	92244	43.38%	54.77%	51.30%	43.02%
200000	0	-	44.67%	56.12%	55.12%	-
200000	45	-	44.18%	50.11%	51.71%	-
200000	90	-	42.76%	46.37%	49.11%	-
500000	0	499240	50.29%	56.13%	53.22%	-
500000	45	63262	51.38%	65.41%	66.15%	-
500000	90	22367	50.95%	66.68%	69.37%	-

Maximum accuracy of 69.37% occurred on a sample size of 500,000 using a length cutoff of 90 words using the LinearSVC classifier.



Plaintext (Tfidf) + hand selected features with default hyper parameters						
Sample	Length	Filtered	Multinomia	LogisticRe	LinearSVC	DummyCla

size	cutoff	size	INB accuracy	gression accuracy	accuracy	ssifier(strategy=”most_frequent”) accuracy
200000	0	184709	45.38%	56.39%	54.21%	42.01%
200000	0	184715	45.33%	56.48%	52.21%	42.01%
500000	0	462376	47.83%	59.65%	56.97%	23.55%

Maximum accuracy of 59.65% occurred on a sample size of 500,000 using a length cutoff of 0 words using the LogisticRegression classifier.

Plaintext (Tfidf) + hand selected features with tuned hyper parameters						
Hyper parameters tuned using a GridSearch with sample size of 1000:						
LogisticRegression	{'penalty': 'l2', 'C': 1e-10, 'class_weight': 'balanced'}					
MultinomialNB	{'alpha': 0.5181818181818181819, 'fit_prior': True}					
LinearSVC	{'penalty': 'l2', 'loss': 'hinge', 'C': 1e-10}					
Sample size	Length cutoff	Filtered size	Multinomial NB accuracy	LogisticRegression accuracy	LinearSVC accuracy	DummyClassifier(strategy=”most_frequent”) accuracy
100000	0	92256	44.41%	43.02%	43.02%	43.02%
200000	0	184719	44.74%	42.01%	42.01%	42.01%
500000	0	462379	44.79%	44.64%	44.64%	44.64%

Maximum accuracy of 44.79% occurred on a sample size of 500,000 using a length cutoff of 0 words using the MultinomialNB classifier.

We had expected that tuning hyperparameters would have improved our classification accuracy but in our experiments we were not able to realize any benefit. Instead, our accuracy decreased to the of the DummyClassifier(strategy=”most_frequent”)

6. Discussion and Conclusion

This project led to great many unexpected results and difficulties which we had not anticipated. One of the first things we encountered was that while we were able to complete a pipeline for feeding data to the classifiers in a previous assignment, its scope was too simple and more importantly, we found that we did not realize the mechanics behind the pipeline well enough to implement it in a larger scope. Extensive research, office hours, and a hefty explanation of sparse matrices and how it formatted data allowed us to meticulously split up the pipeline object into many different objects which better suited our needs.

Our understanding was challenged at a deep level with this hurdle, and was further challenged when we tried to implement new features using a dictvectorizer. We realised that a large part of our accuracy was going to be based on the bag of words created on the data, but when we implemented new features we had almost no gain due to the bag of words powering most of the features. We expected notable accuracy improvements which were not matched at all. We now know that the features generated from this bag of words largely surpass what one can add on manually. The manually added features mostly consisted of the top 2 b-grams from each subreddit. When the bag of words is converted into features, the number of features was much higher than we anticipated - somewhere in the hundreds of thousands. With the miniscule amount of features we can contribute manually, the bag of words features understandably dominated immensely leading to very little improvement in accuracy when adding our own features. Additionally, we expected sizeable gains with our hyperparameter tuning as we thought we were manipulating the classifiers at a deep level, but we weren't able to realize an accuracy improvement. We attempted to tune the following parameters for our classifiers. MultinomialNB, alpha, this parameter affects additive smoothing. LogisticRegression, penalty {l1, l2} and the C coefficient. LinearSVC, penalty {l1, l2}, C coefficient. We used a GridSearchCV to locate optimal parameters. We think that the failure to improve accuracy may be due to a the nature of the vocabulary of the sample data used in the GridSearchCV. If we had more time we would attempt GridSearchCV on a sample set that we could ensure is representative of the rest of our data set.

We knew that we could achieve an accuracy a sizeable amount better than dummy classifier, but our classifiers at the beginning were heavily skewed by askreddit and difficult to work with. We are glad to see that after much tweaking and many additions, we've gotten it to a maximal 69.37% which exceeded our expectations after a rough beginning.

The project's current approach to how we get and process the data would be difficult to scale to larger scopes. Currently we only receive one month's worth of reddit comment data which already takes 32GB uncompressed. We simply filter out the list into the labels we're interested in (top 10) but if we wanted to work across a larger label field with an appreciable amount of data, we would run into significant hurdles. Most importantly, Reddit has a massive userbase with a staggering 638,959 subreddits. We simply cannot process or gain enough information on these subreddits to make it viable that we figure out how we should label a comment. It would be more helpful if most of these were not active, but many tens of thousands of them are frequently visited and receive posts. At this scale, we are simply forced to compromise and disregard several important labels which would be of use. Additionally, as we take in more and more data to feed these subreddits our data usage would skyrocket.

Our classifiers tools could be improved to get better accuracy from our data. Machine learning is an active field and Multinomial Bayesian Network, Linear Regression, and Linear SVC are still in active research. What’s interesting is the deep neural networks which have just recently popped up in the last few years as a strong contender in the artificial intelligence field. Stanford has been researching using a deep learning approach to natural language processing which forgoes a direct bag of words representation in favor of different representations such as grammar and structure (Deep Learning in Natural Language Processing). Their project looks promising and we hope other institutions explore deep learning in the context of natural language processing. More research and application needs to be performed with deep neural networks because, as of now, there simply hasn’t been a lot of time for it to mature in its utilization. We predict that in a few years, there will be quantifiable strides forward in classification from these tools and they will become more accessible to the public.

Many natural language tools are available currently and it would be interesting if these were better fleshed out and developed, especially if it is usable for a more naive userbase. There are a great many things that can be harvested and interpreted on the internet and tools are available but difficult to work with and require extensive knowledge. With more tools that are easier to use, we could see a huge boost as new people which aren’t as familiar with coding are able to identify and process some subsection of the near-infinite amount of data on the internet to generate enlightening results. After all, with abstractions of powerful concepts such as these, it becomes more important what concepts the abstractions are applied to for rather than the individual details of their coding. The presentations in class have showed this - many projects I would have never thought of and I’m sure many are ideas that are being implemented for the first time.

Another direction that this idea could take for the future is the development of peripherals which harvest data from your body including emotions and place this into context of what the user is doing. This could be used to an extremely hefty degree when classifying data and would make processing data which doesn’t contain much information leaps and bounds better. Context such as this would propel artificial intelligence which aggregates data into an amazingly accurate tool with more flexibility.

References:

S. P., C. C., & J. S. (2008). Active Learning for e-Rulemaking: Public Comment Categorization. Retrieved February 19, 2016, from https://www.cs.cornell.edu/home/cardie/papers/dgo-active-learning-2008.pdf

Zhang, J., Jin, R., Yang, Y., & Hauptmann, A. (2003). Modified Logistic Regression: An Approximation to SVM and Its Applications in Large-Scale Text Categorization. Retrieved March 11, 2016, from http://www.aaai.org/Papers/ICML/2003/ICML03-115.pdf

Deep Learning in Natural Language Processing. (n.d.). Retrieved March 11, 2016, from http://nlp.stanford.edu/projects/DeepLearningInNaturalLanguageProcessing.shtml