COMP 551: Applied Machine Learning
Mini-Project 2

Patrick Brebner 260889870, Mohamed Maoui, Dannie Fu 260874695

March 12, 2020

Abstract: In this project, we investigated the text classification performance of five models on two textual datasets. The five models that were explored were Logistic Regression, Decision Trees, Support Vector Machines (SVM), Adaboost, and Random Forest. The textual datasets were the 20 news group dataset and the IMDB large movie reviews dataset. Grid search was used to determine the best hyper-parameters for each model, which were then used to compute and compare the final performance of the models. We found that SVM and Logistic Regression performed the best, with scores of 69.95% and 68.88% for the 20 news group dataset, and 87.76% and 88.36% for the IMDB reviews dataset.

1. Introduction

Text classification is a common machine learning task that involves categorizing textual data or documents based on its contents. The purpose of this project is to investigate the text classification performance of five different models on two textual datasets. The datasets that we explored were the 20 news group dataset, and the IMDB large movie reviews dataset; the five models that were used were Logistic Regression, Decision Trees, Support Vector Machines, Adaboost, and Random Forest. Grid search cross validation was used to determine the best parameters on the training set; these parameters were then used on the testing set to compute the final performance metric.

For the 20 newsgroup dataset, the task was to classify the newsgroup documents into the various different news group categories. The task for the IMDB reviews dataset was to classify a review as positive or negative, based on the contents. We found that overall, the binary classification task had better accuracies for all the models compared to the multiclass classification task. Furthermore, the SVM and logistic regression models had the best performance and the decision tree had the worst. The random forest classifier had decent performance however the run time far exceeded the other models.

2. Related Work

Previous text classification analysis has been performed on these two popular datasets using a variety of models. Adi et al. evaluated the performance of a logarithmic based Naive Bayes classifier on the 20 newsgroup dataset [1]. In their analysis, they did stemming, tokenization, as well as removal of stop words to preprocess the textual data. They found that their Naive Bayes classifier had an accuracy of 86%, which performed well in comparison to the other two models, Icsiboost-bigram and Expected Maximum algorithm (EM), they used, which received accuracies of 71.6% and 79%, respectively. In another paper, Dadgar et al. use a model based on Term Frequency-Inverse Document Frequency (TF-IDF) and Support Vector Machines (SVM) [2]. Their process was composed of three steps: 1) text preprocessing, 2) feature extraction based on TF-IDF, and 3) Classification based on SVM. Their preprocessing pipeline consisted of cleaning the text of punctutations, such as exclamation marks, quotations, and dates. Upper cases were transformed into lower cases prior to tokenization, and stop words were filtered. Using their model, they found a classification performance of 94.93%.

For the IMDB dataset, Yasen et al. performed a sentiment analysis on the reviews for eight popular classifiers: Naive Bayes, Decision Trees, SVM, Bayes Network, K Nearest Neighbour, Ripeer Rule Learning, Random Forest, and Stochastic Gradient Descent. They performed word filtering, stemming, and tokenization, then feature extraction prior to classification. They found that Random Forests produced the best result, with a classification accuracy of 96.01%. [3]

3. Datasets and Setup

Two datasets were analyzed in this project: the 20 newsgroup dataset and the IMDB large movie reviews dataset. With most machine learning text classification projects, data preprocessing is the first step. This can

include, but is not limited to, cleaning the text data, normalizing the text data, and finally converting the text to feature vectors that the classification models will be able to understand.

3.1. 20 Newsgroup Dataset

The 20 newsgroup dataset is a popular dataset for text analysis and classification. The dataset contains around 18,846 newsgroup documents which can be categorized almost evenly into 20 topics. These topics are: comp.graphics, comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, comp.windows.x, rec.autos, rec.motorcycles, rec.sport.baseball, rec.sport.hockey, sci.crypt, sci.electronics, sci.med, sci.space, misc.forsale, talk.politics.misc, talk.politics.guns, talk.politics.mideast, talk.religion.misc, alt.atheism, soc.religion.christian.

The sklearn library provides a built-in dataset loader for the 20 newsgroup dataset. Using this function, we were able to load a shuffled subset of the data for the training and testing sets. This gave us 11,314 documents for the training set and 7,532 documents for the testing set. Headers, footers, and quotes were also removed from each document, leaving only the body of text.

3.2. IMDB Large Movie Reviews Dataset

The IMDB Large Movie Review Dataset contains 50,000 movie reviews along with the associated binary sentiment polarity labels. The dataset is split equally between training and testing sets (25,000 each) with positive and negative reviews evenly distributed. In order to reduce correlation between reviews, only 30 reviews from one movie were allowed and training and test sets contain a variety of different movies so no significant performance could be obtained by memorizing specific movie related terms. Finally, the movie reviews only include polarizing reviews with all reviews that have neutral ratings being removed. The dataset can be retrieved from https://ai.stanford.edu/ amaas/data/sentiment/.

3.3. Data Preprocessing

Real world data can be messy with many characters in the text not conveying any useful information or adding unnecessary features making the classification more difficult. A common first step in preprocessing the data is to clean the data. This might mean removing punctuation, white spaces, numbers, and converting all letters to lower case. This removes many unnecessary characters and simplifies the data for easier classification. There are a few different ways to implement these changes, some of which can be done before text feature vectorization or during the feature vectorization process as will be discussed below.

A possible next step in text preprocessing is text normalization. Stemming and Lemmatization are text normalization techniques which reduce a word down to its base or root form. The idea is that there are usually many words that are derived from the same word such as Playing, Plays, and Played all coming from the root of Play. This is why these techniques are commonly found in web search results and information retrieval where you want to include all variations of the searched text. Stemming and Lemmatization both perform text normalization but have different approaches that come up with distinct results. Stemming is generally more aggressive and works by indiscriminately cutting off the ending or beginning of words by taking a list of common prefixes and suffixes into account. This means that the results of stemming are sometimes not actual words. Lemmatization, on the other hand, focuses more on the base of the word by using a large dictionary as reference to determine the correct root word to use. Both of these can be implemented using the Python "nltk" package quite easily however they might not necessarily have a positive effect on the model accuracy and therefore should be used cautiously. In this project we will experiment with these text normalization techniques to determine if they are effective for our datasets.

Finally, in order to do classification on these textual datasets, numerical features must be extracted from the text files. A method of doing so is called the "Bag of Words" technique, which assigns an integer id to every word in all the documents in the data set. Then, for each document, the occurrence of each word that appears is counted and stored. It's at this step that you are able to choose whether the "Bag of Words" contains only individual words, called unigram, or to include combinations of words, called bigram for when groups of two words are also included. Using bigram vectorization has the potential to improve the model accuracy however the number of features of the dataset drastically increases. In this project we will experiment with the use of bigram vectorization. The "Bag of Words" vectorization technique can be implemented in Python using the sklearn library's "CountVectorizor" function. The CountVectorizor function has several parameters that perform other common text data preprocessing steps discussed above. For example, by default it converts all characters to lowercase prior to tokenizing. As well, there is an option to remove stop words and strip accents from the characters. Stop words can be set to whichever words you want to remove from the text however they are usually words that appear frequently and convey very little information to help with classification.

There are many preset word lists that can be used in python such as the 'english' word list we used in this project however it can sometimes be more beneficial to create your own word list that is more specific to the dataset. Following the Count Vectorizer, the counts of each word for each document were normalized to a term frequency representation using sklearn's "TfidfTransformer" function. This step was to ensure that any remaining commonly occurring words, such as "a, the, is, etc.", which do not contribute valuable information for the classifier, do not skew the classifier. These methods were implemented using the sklearn functions [4].

4. Proposed Approach

In this report, we explore five popular classification models: Logistic Regression, Decision Trees, Support Vector Machines, Adaboost, and Random Forest. Each model has strengths and weaknesses and the goal of this project is to determine which model performs the best on two text classification datasets. The models were implemented using the corresponding sklearn function [4]. Each model has a set of adjustable hyper-parameters that can affect the classification accuracy. To determine the best performing hyper-parameter we used a sklearn Grid Search function. Grid Search takes a dictionary of hyper-parameter values and determines the optimized parameters by using a cross validated grid search over a parameter grid. This exhaustive search can be computationally expensive so only the parameters seen as most important were applied and only a small set of values for each parameter were used. Essentially, the parameter choices were based on what we learned in class along with whether or not their impact on accuracy was worth the computational expense. This means that the grid search, although very useful in optimizing many parameters, is still only searching a small selection of the possible parameter values. In some cases the grid search results were used as a reference and the parameters were slightly adjusted when running the models if it improved accuracy. In the following subsections, each classification model will be further discussed along with the hyper-parameters chosen for the respective grid search.

4.1. Logistic Regression

Logistic regression is a discriminative linear classifier that learns the posterior probability and tries to find a decision boundary that best separates each class. The hyper parameters that were chosen for the grid search were "penalty", which specifies the norm used, "max_iter", which is the maximum number of iterations, and "tol", which is the tolerance for stopping criteria.

4.2. Decision Trees

Decision trees is a type of model that splits and classifies the data according to various "tests", creating a flowchart-like structure with nodes and branches. The hyper parameters that were chosen for the grid search were "criterion", which is the the function to measure the quality of a split, "max_depth", the maximum depth of the tree, and "max_features", the number of features to look at when searching for the best split.

4.3. Support Vector Machines

Support Vector Machines is a discriminative classifier that generates a hyperplane that optimally separates the data. Support vector machines are a common choice for text classification models. The hyper parameters that were chosen for the grid search were "C", which is the regularization parameter, and "max_iter","penalty", and "tol".

4.4. AdaBoost

AdaBoost, or Adaptive boosting, is a model that combines many "weak classifiers", such as decision tree stumps, into a single "strong classifier". At every iteration, each sample is given a weight which is adjusted depending on if it is misclassified or not. The hyper parameters that were chosen for the grid search were "n_estimators", the maximum number of estimators, and "learning_rate", which is the amount the contribution from a classifier shrinks. The base estimator was left as the default setting which is a decision tree with a depth of one.

4.5. Random Forest

Random Forest is a model similar to decision trees, however every tree in the "forest" only has access to a random subset of features and training samples. For regression tasks, the random forest will take the average of all the individual tree estimates; for classification tasks, the random forest will take the majority vote. The hyper parameters chosen for the grid search were "n_estimators", "criterion", "max_depth", "max_features", and "bootstrap", which determines if bootstrap samples are used to build each tree.

5. Results

The classification accuracies of the five models using the best hyper parameters on each dataset will be discussed in the following section. The best hyper-parameters were found by doing a grid search 5-fold cross validation and are shown in the appendix in Table 3. The accuracy results for unigram vectorization with TF-IDF and removal of 'english' stop words are shown in Table 1. In general, we found that SVM and logistic regression performed the best, and decision trees was one of the worst performers. Overall, we can see that the binary classification task on the IMDB dataset achieved higher accuracies compared to the multiclass classification task on the 20 newsgroup dataset.

This corresponds to previous literature done on text classification using these models. In 1997, Joachim reported results for a binary text classification task using SVM. He showed that SVM received a lower error compared to other classifiers, stating that SVM is well suited for text classification as it handles high dimensional space well and can linearly separate the data [5]. On the other hand, decision trees may not perform as well because of the high dimensionality of textual data. Furthermore, due to its hierarchical structure, features are forced to be checked in a specific order, reducing its ability to learn more complex rules or biasing the results if one class is more dominant.

Results				
	20 News Group Dataset		IMDB Movie Reviews Dataset	
Model	Accuracy (%)	Recall (%)	Accuracy (%)	Recall (%)
Logistic Regression	68.88	67.47	88.36	88.32
Decision Trees	43.50	42.59	73.42	83.74
Support Vector Machines	69.95	68.73	87.76	87.06
AdaBoost	45.65	44.85	86.16	88
Random Forest	61.56	60.07	85.66	85.14

Table 1: Final Model Results

5.1. 20 News Group Dataset Results

As shown in Table 1, the models with the best performance were Logistic Regression, SVM, and Random Forest, with SVM having the best score of 69.95%. Decision Trees and Adaboost performed the worst, with scores below 50%.

Figure 1 in the Appendix shows the confusion matrix for the Decision Trees classifier. We can see that a heavy portion of misclassified samples are between classes that are very similar, such as comp.graphics, comp.os.ms-windows.misc, comps.sys.ibm.pc.hardware, talk.politics.misc and talk.politics.mideast, and rec.sport.hockey and rec.sport.baseball. For example, there are 65 instances that were classified as rec.sport.hockey when the true class was rec.sport.baseball. We can also see that many classes were wrongly classified as the rec.auto class.

Figure 2 in the Appendix shows the confusion matrix for the SVM classifier. We can see that, as with Decision Trees, the majority of misclassified instances are between two similar classes. For example, the class with the highest number of misclassified instances was the talk.politics.misc class, which had 84 instances that were classified as talk.politics.guns. In general, however, we see that SVM does a better job of separating classes than Decision Trees.

5.2. IMDB Dataset Results

As seen in Table 1, the accuracies for the binary classifcation IMDB dataset were much higher than the multiclass 20 News group dataset with Logistic Regression and Support Vector Machines performing the best at around 88% accuracy. Decision tree was the worst performer, as expected, and Adaboost and Random Forest performed decently with around 86% accuracy. Although these are relatively decent accuracy scores, if the speed of the models is taken into account then Adaboost and Random Forest are not very desirable, taking much longer than Logistic Regression and Support Vector Machines.

5.3. The Effect of Preprocessing Techniques

To test the effects of preprocessing on the accuracy of classification, the IMDB dataset was prepared with four commonly used but distinct methods. The four methods were unigram vectorization with no text normalization, adding stemming before unigram vectorization, adding lemmatization before unigram vectorization, and bigram vectorization with no text normalization. All methods included the TF-IDF step and removal of the 'english' stop words. The accuracy of the models for each preprocessing method can be seen in Table 2. The results show

that the additional text preprocessing of stemming or lemmatization does not necessarily improve performance. In fact, the accuracy decreased for Logistic Regression, Support Vector Machines, AdaBoost and Random Forest with the Decision Tree model improving slightly. Finally, the bigram vectorization improved performance in all models except Decision Trees. The accuracy of Support Vector Machines was the highest at 89.40 %. This increase in accuracy does come at a cost with the number of features going from 92,446 for unigram vectorization to 1,913,493 for bigram vectorization. This sharp increase in features makes the classification and grid search much more computationally expensive. Grid search for models that are already very slow, such as Random Forest, start to become impractical. The decision whether or not the small increase in accuracy is worth the time and effort would need to be made based on the project requirements.

Results				
	Unigram	Stemming	Lemmatization	Bigram
Model	Accuracy (%)	Accuracy (%)	Accuracy (%)	Accuracy (%)
Logistic Regression	88.36	87.94	88.10	89.11
Decision Trees	73.42	73.58	73.78	73.07
Support Vector Machines	87.76	87.35	87.53	89.40
AdaBoost	86.16	86.12	86.06	86.62
Random Forest	85.66	85.12	85.32	86.37

Table 2: Results of Varying the IMDB Dataset Preprocessing Techniques

6. Discussion and Conclusions

Text classification is a common machine learning task which has applications such as spam detection, tagging content, or search engine optimization. As such, it's important to use the most accurate classification model for your application. The models analyzed in this project each have their strength and weaknesses that become apparent when applied to text classification.

We found that of the five models, SVM and Logistic Regression had the highest classification accuracy, whereas decision trees was one of the worst performers. Furthermore, the binary classification task achieved higher overall scores as compared to the multiclass classification task. We observed that, although the Random Forest classifier performed well, the computation time far exceeded those of Logistic Regression and SVM, both of which had better results, and therefore it would not be the optimal classifier to use.

An important aspect of this project's tasks was the text preprocessing, which may vary based on the source of your data but usually involves both cleaning and feature vectorization. However, it was found that more preprocessing does not necessarily give better results as was seen with text normalization and the IMDB Dataset. Other methods such as bigram feature vectorization improved accuracy but made the model much more computationally expensive. Whether or not this improvement in accuracy is worth the extra computational work would need to be decided based on the project requirements. Overall, this experiment illustrated the need to methodically plan out your text preprocessing rather than just apply a general technique to all datasets.

Moving forward, future efforts should be concentrated on improving the preprocessing pipeline, as well as to explore the other hyper-parameters for each model. For example, we could refine the list of stop words and include more options when testing the hyper parameters in the grid search. Particularly for the 20 newsgroup dataset, we can explore other approaches to improve the classification results. For example, stemming and lemmatization were not implemented in the preprocessing of the 20 newsgroup dataset for this report, so, as with the IMDB dataset, this technique is something to explore. Furthermore, to try and improve the results for the Adaboost classifer, we could also try to transform the multiclass problem into multiple two class problems as is commonly done with multiclass tasks. In general, other models to explore would be Naive Bayes, which, like SVM, is also considered to be very effective for text classification, or the Pegasos model, which is an adaptation of SVM.

In conclusion,we explored text classification using five different models. We observed that the preprocessing was important, however some techniques did not improve the classification performance. We observed that some types of models performed better at text classification due to the nature of their algorithm, while others were more limited in their capability to handle textual data, particularly for multiclass data. Finally, we explored how other factors, such as run time, impacted the overall benefit of using a model.

7. Statement of Contributions

Patrick analysed the IMDB dataset. Dannie analysed the 20 news group dataset.

References

[1] A. O. Adi, E. Çelebi, Classification of 20 news group with naïve bayes classifier, in: 2014 22nd Signal Processing and Communications Applications Conference (SIU), IEEE, 2014, pp. 2150–2153.

[2] S. M. H. Dadgar, M. S. Araghi, M. M. Farahani, A novel text mining approach based on tf-idf and support vector machine for news classification, in: 2016 IEEE International Conference on Engineering and Technology (ICETECH), IEEE, 2016, pp. 112–116.

[3] M. Yasen, S. Tedmori, Movies reviews sentiment analysis and classification, in: 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT), IEEE, 2019, pp. 860–865.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[5] T. Joachims, Text categorization with support vector machines: Learning with many relevant features, in: European conference on machine learning, Springer, 1998, pp. 137–142.

Appendix

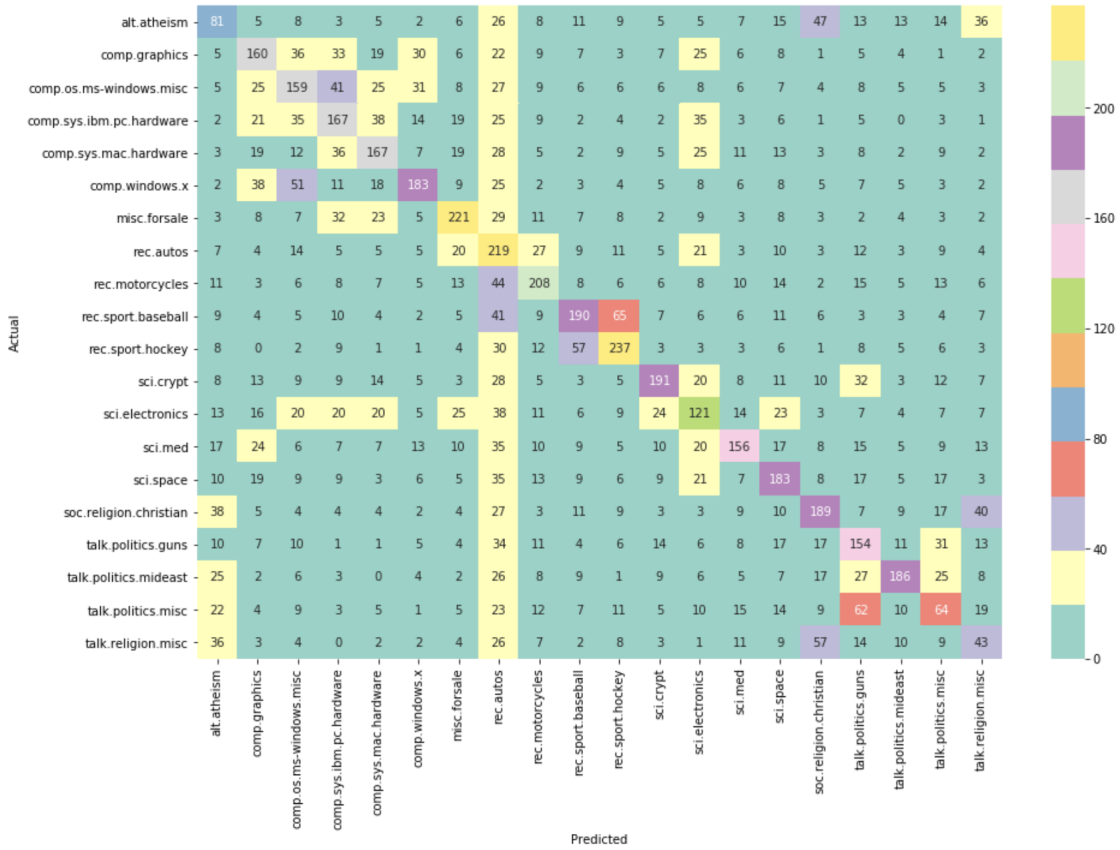

Figure 1: Decision tree confusion matrix for all 20 Labels

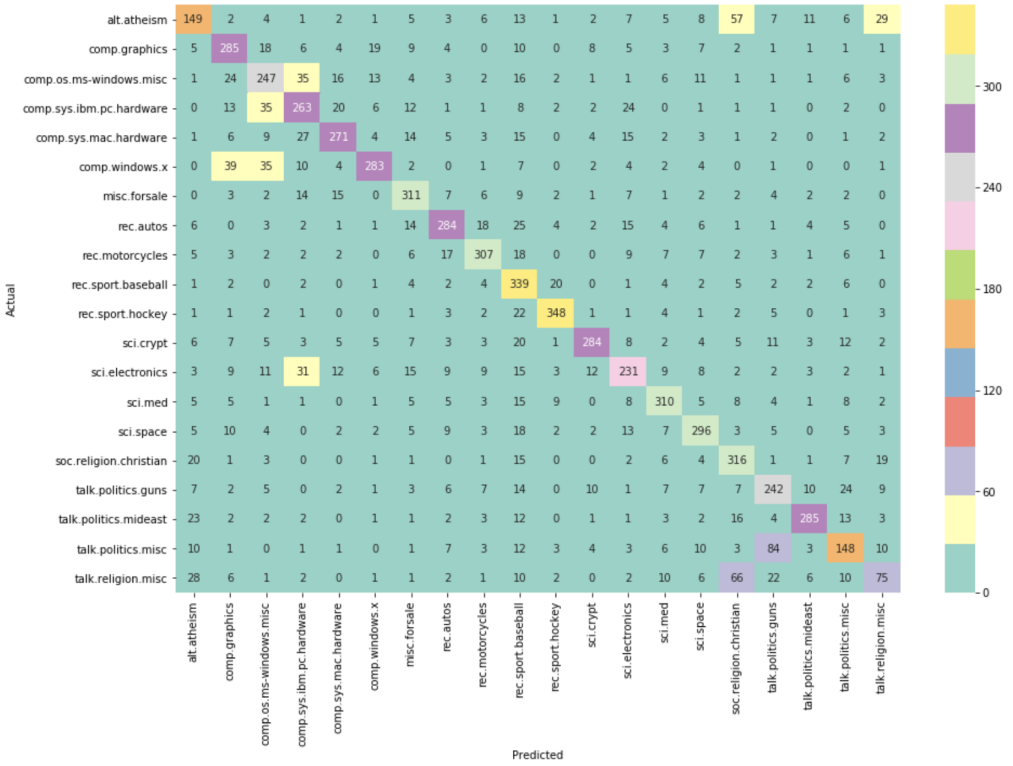

Figure 2: SVM Confusion Matrix for all 20 Labels

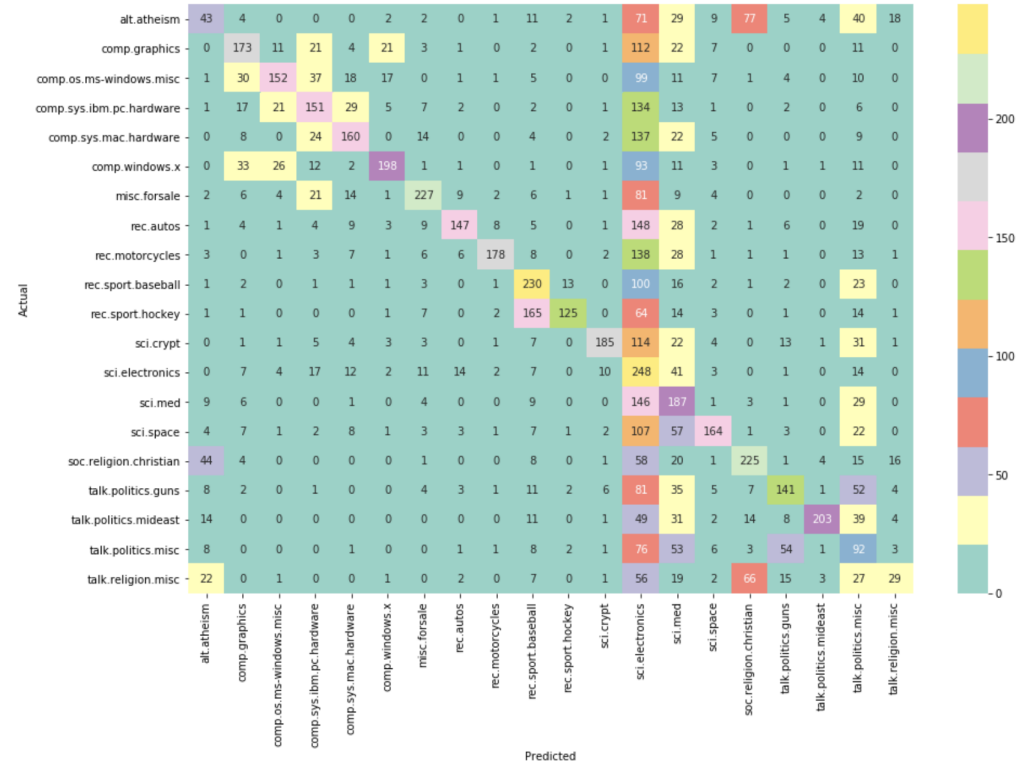

Figure 3: Adaboost Confusion Matrix for all 20 Labels

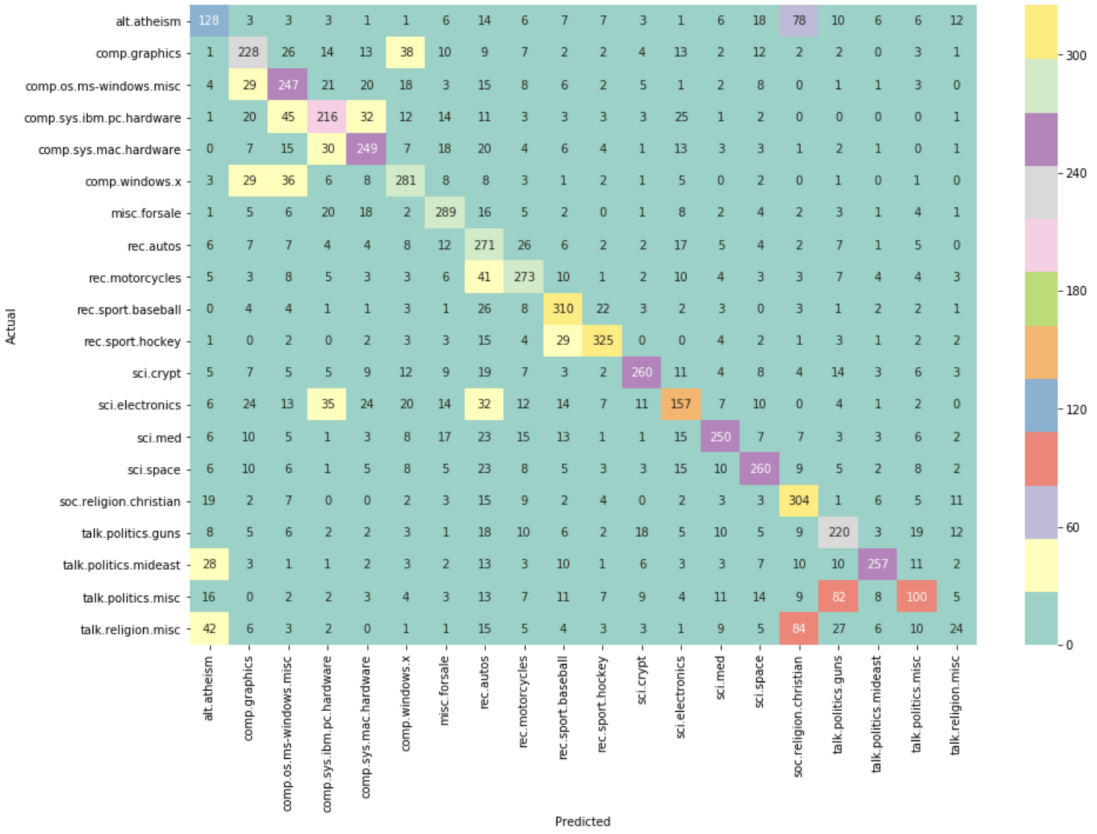

Figure 4: Random Forest Confusion Matrix for all 20 Labels

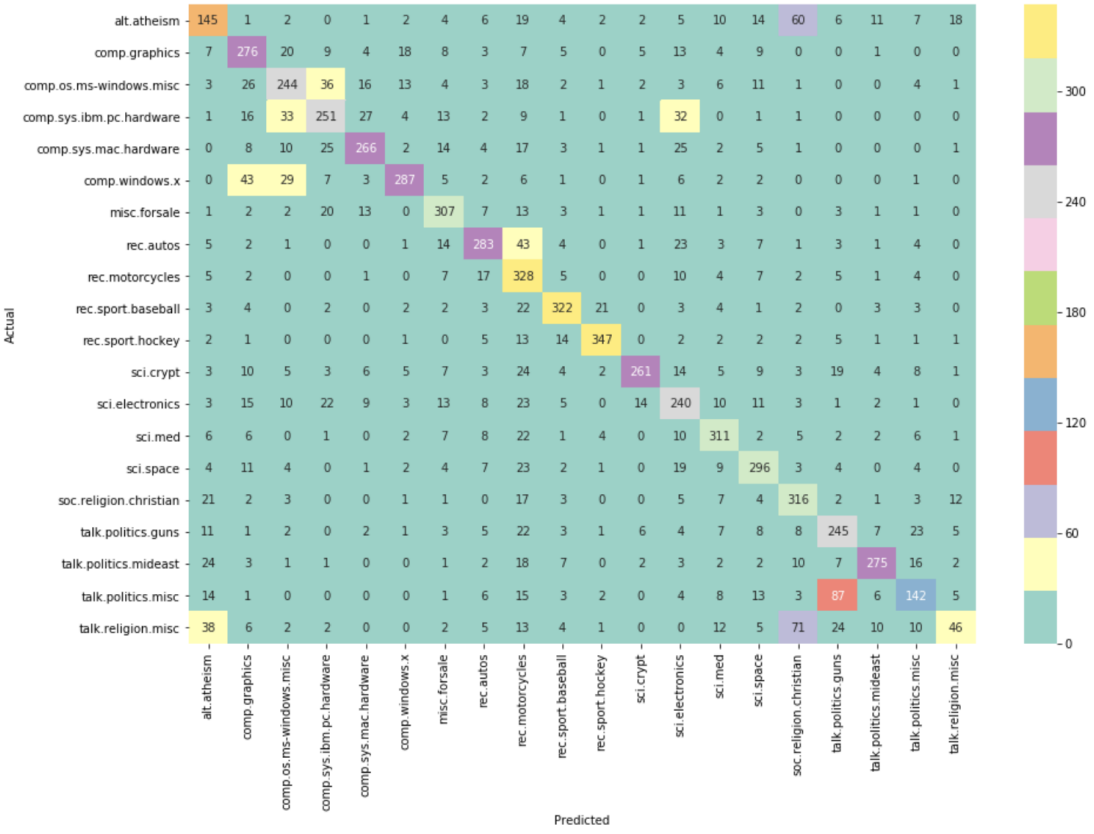

Figure 5: Logistic Regression Confusion Matrix for all 20 Labels

Best Hyper Parameters Found Using GridSearchCV		
	20 News Group Dataset	IMDB Movie Reviews Dataset
Logistic Regression	penalty = 'l2' max_iter = 5000 tol = 0.01	penalty = 'l2' max_iter = 5000 tol = 0.1
Decision Trees	criterion = 'gini' max_depth = None max_features = None	criterion = 'gini' max_depth = 20 max_features = None
Support Vector Machines	C = 0.5 maxiter = 1000 penalty = 'l2' tol = 0.1	C = 0.5 maxiter = 1000 penalty = 'l2' tol = 0.01
AdaBoost	base_estimator = None n_estimators = 200 learning_rate = 0.5	base_estimator = None n_estimators = 500 learning_rate = 0.5
Random Forest	n_estimators = 60 criterion = 'gini' max_depth = None max_features = 'sqrt' bootstrap = False	n_estimators = 120 criterion = 'gini' max_depth = None max_features = 'sqrt' bootstrap = False

Table 3: Best Hyper-Parameter Values