Assignment 3: Text Classification

Name: Swaminathan Sivaraman
SBU ID: 110951180
CSE 537 Artificial Intelligence
Stony Brook University

Files and Scripts included:

1) **This report**

2) **[Part a] text_analyzer.py:** This is the script used to analyze all algorithms in part (a). It accepts as input the location of the training and test data and an output png filename to draw the plots. This script will also print results in a tabular format
   Example run: text_analyzer.py /user/Selected_20NewsGroup /user/output_plots.png

3) **[Part b] get_best_naive_bayes.py:** This is the script used to check all 8 or more configurations for the Naïve Bayes algorithm, which was best-performing in part (a). It accepts as input the location of the training and test data and outputs results for all configurations in a tabular format.
   Example run: get_best_naive_bayes.py /user/Selected_20NewsGroup

4) **[Part b - IMPORTANT] generate_model_with_best_config.py:** This script is used to generate a new model based on the best config. It accepts as input the location of the training data alone and generates a model and dumps it to a .pkl file in the current working directory.
   Example run: generate_model_with_best_config.py /user/Selected_20News/Training
   Pickle file generated:
   full_model.pkl

5) **[Part b] full_model.pkl:** A full_model.pkl file generated using the above script with the given training and test data

6) **[Part b - IMPORTANT] run_model_with_best_config.py:** This script is used to run the model generated by generate_model_with_best_config.py. It accepts as input the location of the test data alone. It expects the above full_model.pkl file to be in the current working directory.
   Example run: run_model_with_best_config.py /user/Selected_20NewsGroup/Test

1. Basic Comparison with Baselines:

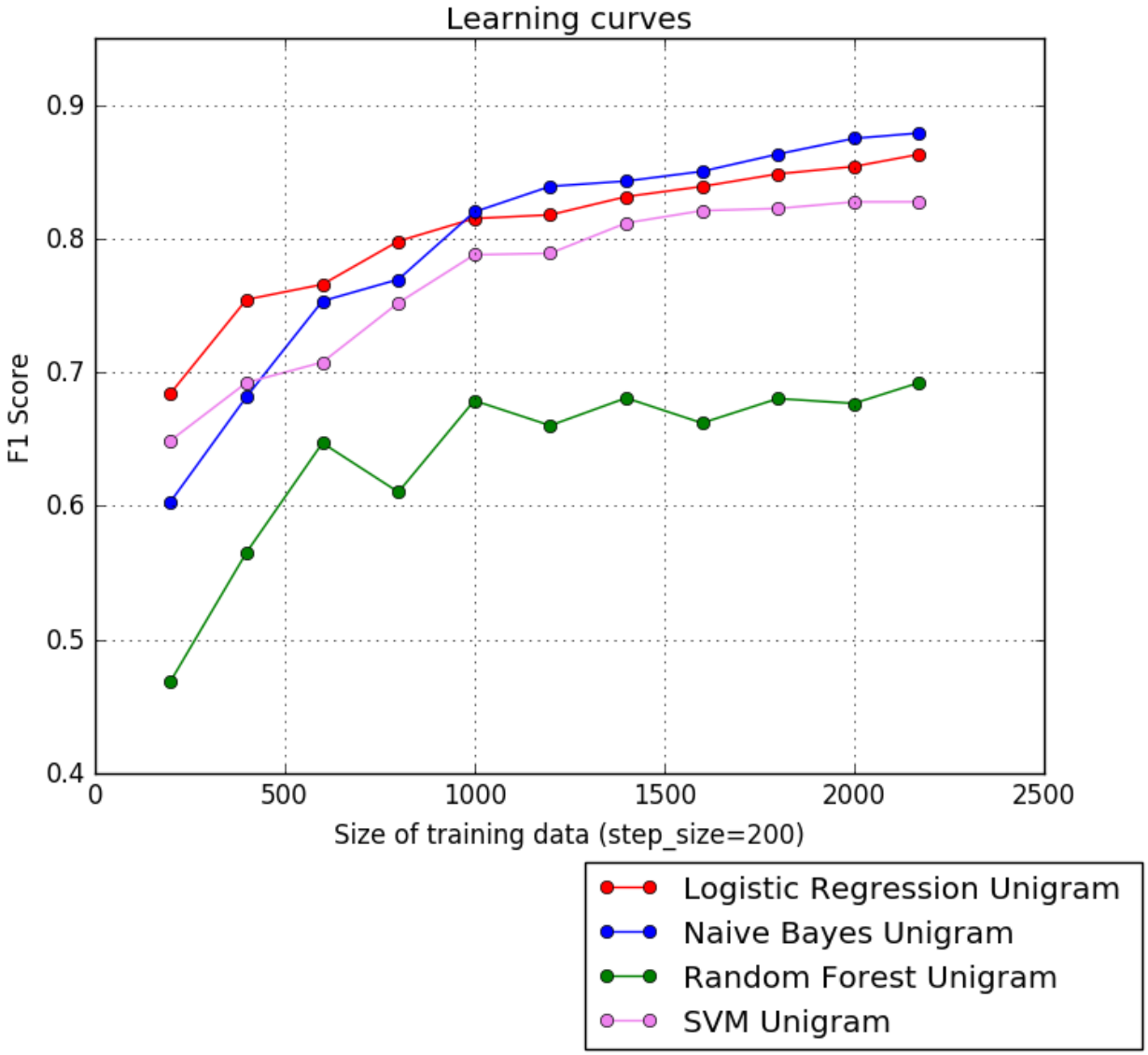
a. A table of macro-average of precision/recall and F1 values for all 8 runs (4 classifiers x two representations)

(All scores are macro-averaged scores)

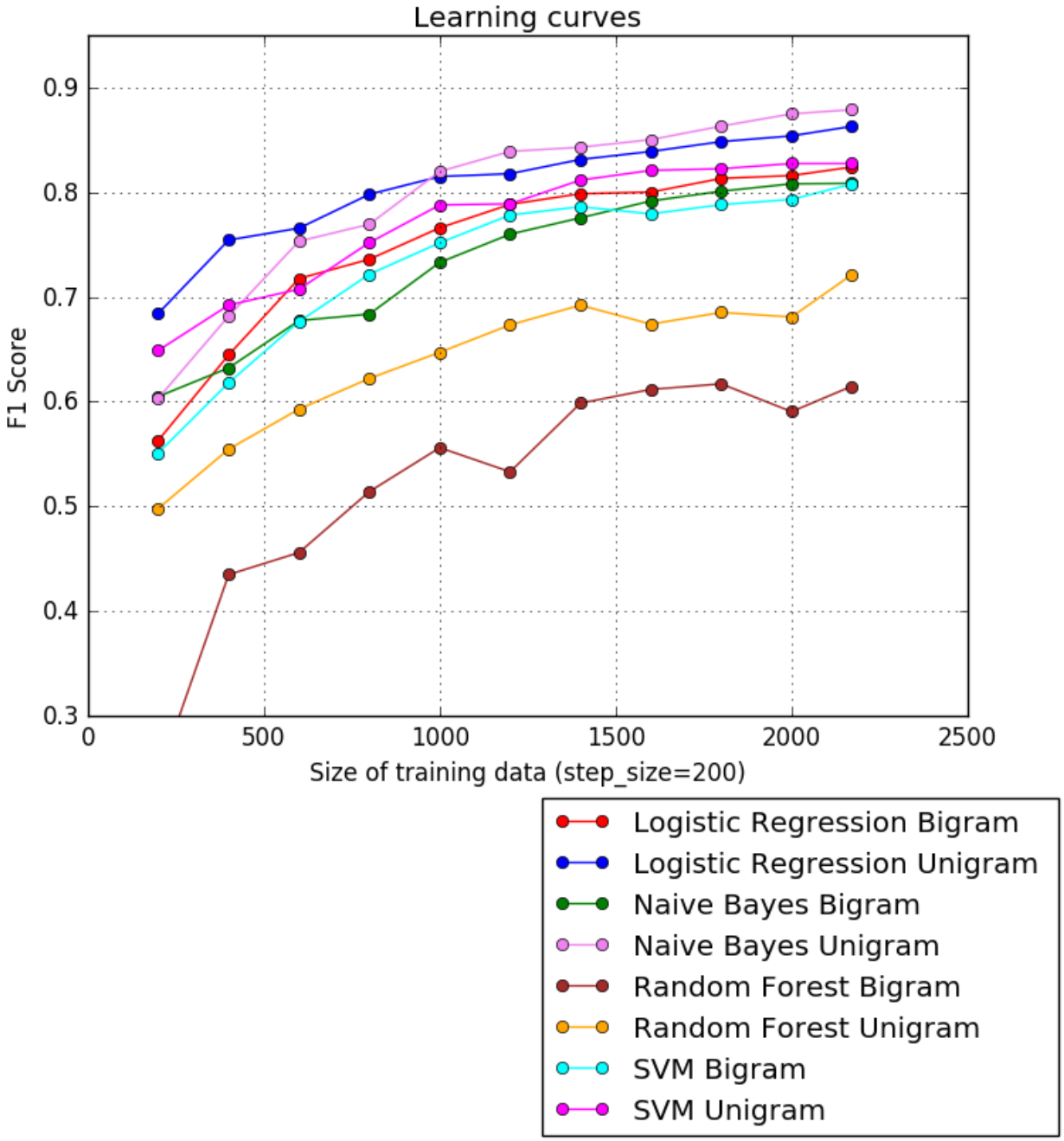
Classifier	Precision	Recall	Precision/Recall	F1 Score
Logistic Regression Bigram	0.85361	0.81754	1.04413	0.82396
Logistic Regression Unigram	0.86987	0.85982	1.01168	0.86292
Naïve Bayes Bigram	0.87666	0.7977	1.09899	0.80854
Naïve Bayes Unigram	0.91566	0.86888	1.05384	0.87891
Random Forest Bigram	0.72734	0.6311	1.15251	0.61978
Random Forest Unigram	0.73657	0.70657	1.04245	0.69498
SVM (LinearSVC) Bigram	0.83459	0.8014	1.04141	0.80766
SVM (LinearSVC) Unigram	0.83213	0.82574	1.00775	0.82744

b. A learning curve result, with the performance of each classifier with just the unigram representation. The learning curve is a plot of the performance of the classifier (F1 on the y-axis) on the dev fold, when trained on different amounts of training data (size of training data on the x-axis).

Only unigram results:



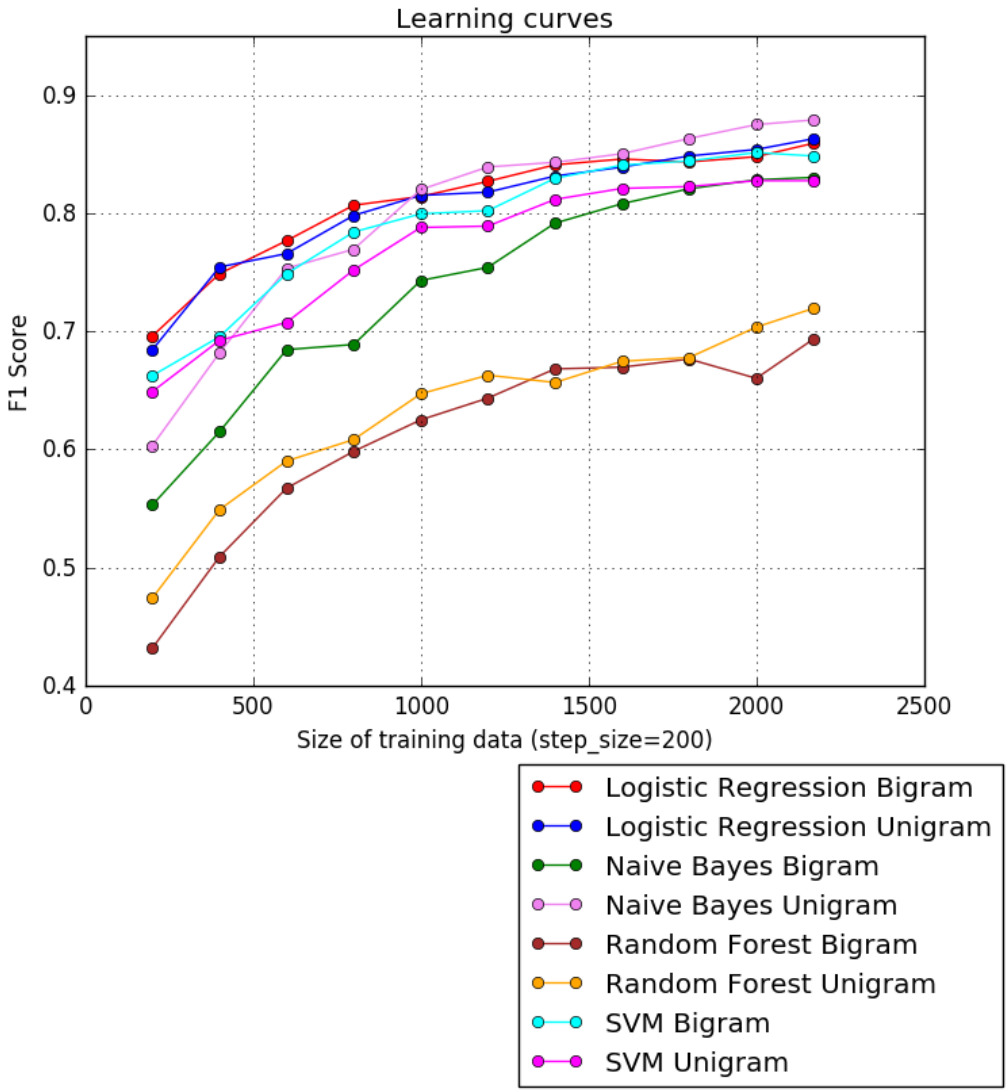
Both unigram and bigram results:



c. Describe your findings and make arguments to explain why this is the case.

Unigrams perform better than bigrams:

Contrary to expectations, the unigram results are better than the bigram results for all cases. (Note: In the experiments, the unigram version used only unigrams and the bigram version used only bigrams, that is, no version used both unigrams and bigrams). I assume this is because, when we start using bigrams, though we get more contextual information, the frequency of each bigram goes down. And so, we need more training data to get our model to perform better than the unigram. Also, it has been mentioned in this paper that it is better to use a combination of unigrams and bigrams rather than using bigrams alone, since some unigrams can also sometimes have a large bearing on the results. In fact, re-running the above tests with a unigram+bigram produced the below results where most of the "bigram" (unigram+bigram now) models performed a lot better with the SVM bigram performing better than the unigram version.



Performance of the 4 different algorithms:

The algorithms performed best to worst in this order - Naïve Bayes, Logistic Regression, SVM and Random Forest. Naïve Bayes, Logistic Regression and SVM are almost similar in performance. It is quite unexpected that Naïve Bayes is working so well, for being such a simple algorithm. But it is mentioned here (paper) that Naïve Bayes performs really well if there's a uniform distribution of words across all samples. The SVM model is also model and the performance goes above 90 % when we use a TifdfVectorizer instead of a CountVectorizer. The reason SVM performs so well here is due to these reasons - Fewer irrelevant features, Document vectors are sparse, most text categorization problems are linearly separable (taken from link). Logistic Regression also performs almost as well as the SVM, but is slightly less performant. This link could be a reason. We are facing a classification problem here and SVMs purely solve a classification problem by trying to figure out the dividing plane. But logistic regression draws a curve to fit the data and then converts that into a classification problem depending on which side of the curve the test data falls on. So, it is doing possibly unnecessary work, where it could be trying to optimize a portion of the curve which is not near the middle. Random Forest comes lost since it generally needs a larger amount of training samples than SVM to work well (source).

References:

1) http://stats.stackexchange.com/questions/23490/why-do-naive-bayesian-classifiers-perform-so-well/23491#23491
2) https://www.researchgate.net/publication/221439320_The_Optimality_of_Naive_Bayes
3) https://www.quora.com/Why-does-linear-kernelized-SVM-perform-much-better-on-text-data-than-on-image-data/answer/Charles-H-Martin
4) http://www.cs.cornell.edu/people/tj/publications/joachims_98a.pdf
5) https://www.researchgate.net/post/What_is_the_best_algorithm_for_supervised_text_categorization_of_papers
6) http://stats.stackexchange.com/questions/58684/regularized-logistic-regression-and-support-vector-machine
7) https://www.quora.com/Why-does-linear-kernelized-SVM-perform-much-better-on-text-data-than-on-image-data
8) https://www.researchgate.net/post/Is_random_forest_better_than_support_vector_machines

2. My Best Configuration:

a. Exploration results (20 points) -- You should provide results from 8 configurations picking two choices from each of the options below.

The Naïve Bayes unigram algorithm performs best out of the lot in the above tests. I then poked around with Naïve Bayes with the different configurations suggested. Following are the results (all scores are macro-averaged scores),

Model no.	Model name	Precision	Recall	Precision/Recall	F1 Score
1	Naive Bayes alpha=0.01; unigram; countvectorizer; no stemmer; no stopper; no feature selection	0.93089	0.91702	1.01513	0.92202
2	Naive Bayes alpha=0.01; unigram; tfidfvectorizer; no stemmer; no stopper; no feature selection	0.92564	0.89916	1.02944	0.90683
3	Naive Bayes alpha=0.01 fit_prior=False; unigram; tfidfvectorizer; no stemmer; no stopper; no feature selection	0.93343	0.91232	1.02313	0.91907
4	Naive Bayes alpha=0.01; unigram; countvectorizer; stemmer; no stopper; no feature selection	0.92759	0.91252	1.01651	0.91785
5	Naive Bayes alpha=0.01; unigram; countvectorizer; no stemmer; stopper; no feature selection	0.92966	0.91503	1.01599	0.92028
6	Naive Bayes alpha=0.01; unigram; countvectorizer; stemmer; stopper; no feature selection	0.92626	0.91055	1.01726	0.91607
7	Naive Bayes alpha=0.01 fit_prior=False; unigram; tfidfvectorizer; no stemmer; stopper; no feature selection	0.93277	0.91629	1.01798	0.92188

8	Naive Bayes alpha=0.01 fit_prior=False; unigram; tfidfvectorizer; stemmer; stopper; no feature selection	0.92959	0.91522	1.0157	0.92013
9	Naive Bayes alpha=0.01; unigram; countvectorizer; no stemmer; stopper; feature selection - SelectPercentile=80	0.92914	0.91702	1.01322	0.9214
10	Naive Bayes alpha=0.01 fit_prior=False; unigram; tfifdfvectorizer; stemmer; stopper; feature selection - SelectPercentile=80	0.93481	0.92045	1.0156	0.92532
11	Naive Bayes alpha=0.01; unigram; countvectorizer; no stemmer; stopper; feature selection - SelectFromModel threshold=39	0.932	0.91702	1.01633	0.9224
12	Naive Bayes alpha=0.01 fit_prior=False; unigram; tfidfvectorizer; stemmer; stopper; feature selection - SelectFromModel threshold=30	0.93463	0.9217	1.01403	0.92616

As seen, the model "Naive Bayes alpha=0.01 fit_prior=False; unigram; tfidfvectorizer; stemmer; stopper; feature selection - SelectFromModel threshold=30" has the best F1 score of 0.92616.

The above table can be generated by running the get_best_naive_bayes.py script as follows,

```
>> python get_best_naive_bayes.py /user/Selected_20NewsGroup
```

b. Code (20 points) -- Export the best configuration training model

There are two scripts - one for generation of the model with training data, and one for testing the generated model with test data.

1. generate_model_with_best_config.py
   This script accepts as input the location of the training data and generates a model. The model is saved to a 'full_model.pkl' file in the current working directory.

   Example: python generate_model_with_best_config.py /user/Selected_20NewsGroup/Training

2. run_model_with_best_config.py
   This script accepts as input the location of the test data. It looks for a 'full_model.pkl' file in the current working directory, converts it into relevant Python scikit-learn objects and tries to predict the results in the test data. It reports macro-averaged precision, recall, precision/recall and F1 scores.

   Example: python run_model_with_best_config.py /user/Selected_20NewsGroup/Test

c. Explanation (10 points) -- Explain your result based on your best understanding of the configuration options.

In part (a), Naïve Bayes gave a score of only 0.87. A noticeable increase was achieved on setting hyperparameter 'alpha' to 0.01. 'alpha' is the smoothing factor, and will make sure rare occurrences of words don't affect the overall result. But it seems that in our training data, rarely occurring words also do have an influence on the results. So, reducing the smoothing factor made the score go up to 0.92. Using TfidfVectorizer initially brought down the score but as it tries to reduce the weight of very commonly-occurring words like "a" and "the", setting the hyperparameter 'fit_prior' to False for this case helped, as this does not assign prior class weights. The CountVectorizer with fit_prior=True and the TfidfVectorizer with fit_prior=False gave almost similar results. Interestingly, both stemming and stopping brought down the score for the CountVectorizer case, with the stemmer heavily bringing down the score. Based on my tests with the SVM separately, stemming is always bringing down the score. On experimenting with the PorterStemmer, it is returning the wrong words for some cases, which may defeat the purpose of stemming itself. We would need more data for it to be useful. Stopping increased the score in the SVM experiments leading me to believe that it is a useful action here but not with the Naïve Bayes+CountVectorizer case. For the TfidfVectorizer case with Naïve Bayes, stemming and stopping do help, as TfidfVectorizer will try to reduce the frequency of very-high-frequency words and if stop words are removed, the vectorizer's actions will actually be useful.

The feature selectors produce a marginal increase in the scores. The SelectPercentile selector with a percentage threshold of 80% performs best. This is probably expected, since words in text would generally be uniformly distributed with equal importance but there would be some

amount of unnecessary information or repetition. The SelectFromModel selector also performs about the same but slightly better. This link mentions that the SelectFromModel feature selector is generally good for sparse data, something which would be the case for a bag-of-words model.

References:

1) https://en.wikipedia.org/wiki/Additive_smoothing
2) http://stackoverflow.com/questions/30498021/scikit-how-to-choose-alpha-of-multinomailnb
3) http://stackoverflow.com/questions/10407266/scikits-learn-and-nltk-naive-bayes-classifier-performance-highly-different
4) http://scikit-learn.org/stable/modules/feature_selection.html