CSE 537 Assignment 3: Text Classification

Submitted by
Bharathkrishna Guruvayoor Murali
SBU ID : 110945903

1)

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

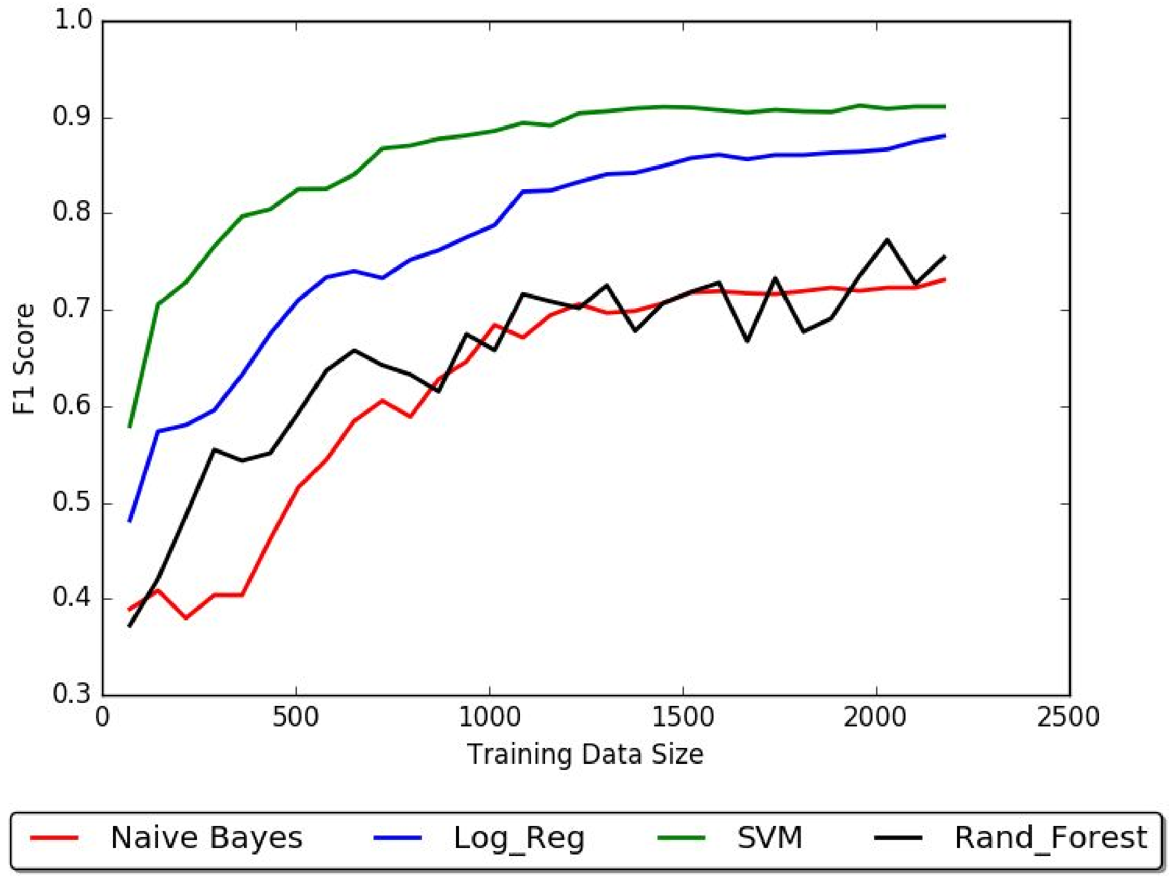
Below are the results of all the algorithms used with their baseline / default configurations.
All the results are populated using the tfidf-vectorizer, using metrics.precision_recall_fscore_support from Scikit.

The values are in the order of precision, recall and fscore.

	Naive Bayes	Logistic Regression	SVM	Random Forest
Unigram representation	0.89077354426619126 0.75235122220046846 0.7309441625440396	0.92321077375445115 0.8735950676905450 0.88020311549999608	0.927708924936139 0.903910787704757 0.910880632740603	0.8035462406677196 0.74900963149078720 .7500380684462078
Bigram representation	0.88991941879317638 0.75920547905472535 0.74551386773200501	0.9155030987969377 0.8580891470338706 0.8644717459678213	0.935387500511505 0.915094048888018 0.921661360969703	0.79179918088384720 .7151271509060452 0.7091764999639617

b) A learning curve result, where you show the performance of each classifier with just the unigram representation.

The performance analysis is shown in the graph below. X axis represents the training data, and Y axis shows the F1 score. The training data was split into 20 sizes, and F1 scores were calculated for these portions to plot the graph between training data and F1 scores.



c) Describe your findings and make arguments to explain why this is the case.
The learning curve allows us to measure the rate at which the algorithm is able to learn. The maximum point is usually when the slope starts to recede. So from the above example, we can see that the curve is gradually tending towards a constant value (ie. F1 score of 1 in this case). It initially starts learning through the training data and the slope gets wide at maximum point where it tends to approach the constant state. At this point it is able to pick up new examples from test data and find new results from the data. Hence, we can see the size of the training data needed to train the model properly. We can observe that SVM reaches the constant level faster than Naive Bayes or Logistic Regression. Hence, it would need lesser data for training. Also, random forest increases constantly, but has repeated ups and downs due to the selection of random value subsets of training data.

The learning curve in all the cases can be seen to improve with the number of training examples. This makes it evident that with more data, the learning becomes better. If the training set is small, high bias/low variance classifiers (e.g., Naive Bayes) have an advantage over low bias/high variance classifiers (e.g., SVM), since the latter will overfit. But low bias/high variance classifiers start to become better as the training set grows (they have lower asymptotic error), since high bias classifiers aren't powerful enough to provide accurate models[1]. Using this concept, even though I could see during trial and error that Naive Bayes obtained an f1 score of 0.94 to 0.95 with C=0.01, I am taking SVM with tfidf+bigram as the best classifier option, because it looks more reliable and the training using various configurations gives expected results, meaning that it is more realistic than Naive Bayes or other classifiers.

Bigram groups two words at a time, instead of one in unigram model. Hence, it is expected that bigram range will give a more realistic output. This is also a motivation to choose SVM because it acts in accordance with the expectations . Bigrams here give a better score than unigram representations for SVM.

<u>2)My best configuration</u>

<u>a)</u>

The baseline results in the best score of 0.921 for SVM using tfidf vectorizer with bigram. Hence, choosing SVM as the prefered classifier for the best configuration. All the below results are done on bigram model as it is observed to have better scores in 1.a.

<u>8 configurations</u>

1. **Using count vectorizer + stop_words filter**

Precision	:	0.91710000494639898
Recall	:	0.89673365927762916
F1 Score	:	0.90315515077271091

2. **Using count vectorizer + stop_words filter + stemming (porter stemmer)**

Precision	:	0.91321511334182659
Recall	:	0.89746784971031202,
F1 Score	:	0.90272789429461098

3. **Using tfidf vectorizer + stop_words filter + stemming (porter stemmer)**

Precision	:	0.9359149458815601
Recall	:	0.92051793056190045
F1 Score	:	0.92597736176150958

4. **Using tfidf vectorizer + stop_words filter + stemming + penalty L1**

Precision	:	0.91392812594759654
Recall	:	0.89986957696128556
F1 Score	:	0.90463978661365529

5. **Using tfidf vectorizer + stop_words filter + stemming + penalty L2 + C=1 + select Percentile (30)**

Precision	:	0.94203095150463567

Recall : 0.92627724662272404
F1 Score : 0.93187218114597647

6. **Using tdidf vectorizer + stop_words filter + stemming + penalty L2 + stripping headers**
   Precision : 0.89185027552529006
   Recall : 0.8753197082091555
   F1 Score : 0.88041532975745374
7. **Using tfidf vectorizer + stop_words filter + stemming + penalty L2 + remove Punctuation**
   Precision : 0.928923517017189
   Recall : 0.91872680502077486
   F1 Score : 0.92259840296064399

8. **Using tfidf vectorizer + stop_words filter + stemming + penalty L2 + C=6.0 + select Percentile (30)**
   Precision : 0.94546658109398374
   Recall : 0.93619788154335892
   F1 Score : 0.93990253001344992

Fine tuning the above configuration, I have selected the below as the best configuration :

9. Using tfidf vectorizer + stop_words filter + stemming + penalty L2 + C=5.0 + select Percentile (25)
Precision : 0.94585932751967894
Recall : 0.93746050780598522
F1 Score : 0.94085930031766229

Final best configuration details :

- SVM with linear kernel
- Feature representation : TFIDF Vectorizer (bigram)
- n-Gram model : Bigram
- With filter stop-words and porter stemmer
- Feature selection : SelectPercentile (25)
- Regularizer : L2
- HyperParameter: Penalty parameter C = 5.0

2.b)

<u>HOW TO RUN</u>

Python version used: 3.5.x
Below is description about each file, and syntax to run.

=> default_all.py

This file contains default (baseline) configuration, for question 1, has code to plot graph, find f1 score for all 4 classifiers with unigram and bigram configuration.
I have used tfidf vector for analysis, but anyways the file has code for both tfidf and count vectorizers.
Outputs Precision, recall and F1 score for each configuration and displays the graph for all unigram classifications.

```
Syntax   :    python default_all.py <Training-folder> <Test-folder>
Example : python default_all.py Training Test
```

=> model_new.py

This file contains the code for the 8 configurations tried out for question 2.
Outputs the precision, Recalls and F1 score for each configuration.

```
Syntax : python model_new.py <Training-folder> <Test-folder>
Example : python model_new.py Training Test
```

=> best.py

This file has the best configuration classifier, mentioned in 2.a. It creates a model and it is stored in a pickle file. It accepts command line arguments Training folder and pickle file name to use for storing the model. This model can be used in test_file.py for preicting on the test data.

```
Syntax : python best.py <Training-folder> <pickle-filename>
Example: python best.py Training model.pkl
```

=> test_file.py

This file imports a trained model, stored as a pickle file. Then, it uses the model to make predictions on the test data. Note that the tokenizer function needs to be defined in this file, as the pickle needs to invoke code to tokenize the test data.
Outputs the precision, recall and F1 score.

```
Syntax    :   python test_file.py <model-pkl> <Test-folder>
Example :   python test_file.py model.pkl  Test
```

2.C) Explain your result based on your best understanding of the configuration options.

SVM has theoretical guarantees regarding overfitting, and with an appropriate kernel it can work well even if the data isn't linearly separable in the base feature space.[1]

TFIDFVectorizer provides a better model as it uses the relative frequencies normalized by the inverse of the number of newsgroups in which the word was seen, as opposed to the count vectorizer which just takes the number of times each word was observed. Stop words are the high frequency words that do not provide much additional information to the predictions. Hence, these can be filtered out before training the classifier to get better realistic model. Stemming is the process for reducing words to their base form. This will give a better idea on frequency of the words, as it minimizes the different representations of same word. We can see that stemming improves the performance of the classifier.[4]

The feature selection used in the best configuration is select percentile. This selects features according to a percentile of the highest scores. It has helped in preventing non-trivial words from affecting the accuracy of predictions. Regularization is used to avoid overfitting. The L1 and L2 values are 'lamda', the regularization constant. If this is very large, models with high complexity are ruled out. If lambda is small, models with high training errors are ruled out.[2]. Stripping off the headers lowers the F-score because it is more realistic, as it removes the news related metadata. The penalty parameter C indicates to the SVM how much wrong classification should be avoided. For large values of C, a smaller-margin hyperplane will be chosen, if that hyperplane is found to perform better in getting all points classified correctly. A very small value of C will make the optimizer to look for larger-margin separating hyperplane. So for very small values of C, we will get misclassified examples even if data is linearly separable.[7]

REFERENCES

1. http://blog.echen.me/2011/04/27/choosing-a-machine-learning-classifier/
2. https://datanice.github.io/machine-learning-101-what-is-regularization-interactive.html
3. http://stackoverflow.com/questions/4617365/what-is-a-learning-curve-in-machine-learning
4. https://www.codeschool.com/blog/2016/03/25/machine-learning-working-with-stop-words-stemming-and-spam/
5. http://scikit-learn.org/stable/documentation.html
6. NLTK documentation
7. http://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel
8. The links provided in HW3 question