McGill University

Assignment 2

Nabil Chowdhury
ID: 260622155
COMP 551

February 26, 2018

1 Converting Yelp and IMDB Datasets into Bag of Words (BoW)

Since the algorithms used in this assignment expect vector inputs, the reviews in both datasets must first be converted into a fixed length vector representation. Each of the training, testing, and validation splits of both datasets are converted into Binary BoW and Frequency BoW representations (as explained by the assignment). Scikit-learn's CountVectorizer was used to generate both BoW representations. The output files associated with this step can be found in the code submission on MyCourses, inside the "260622155/Submission/" directory. The files should be as follows:

1. **Yelp**
   - yelp-vocab.txt - The generated vocabulary (10000 words) from Yelp's training set

     The following files mirror the provided files, but replace words with the vocabulary IDs (words not in the vocabulary are ignored):
   - yelp-train.txt
   - yelp-valid.txt
   - yelp-text.txt
2. **IMDB**
   - imdb-vocab.txt - The generated vocabulary (10000 words) from IMDB's training set

     The following files mirror the provided files, but replace words with the vocabulary IDs (words not in the vocabulary are ignored):
   - imdb-train.txt
   - imdb-valid.txt
   - imdb-text.txt

2 Yelp Binary BoW Classifications and F1 Scores

2.1 F1 Score Average

Scikit-learn's f1_score method was used to calculate the F1 scores for all classifications in this assignment. Micro averaging was used (average='micro'). This is preferred in multi-class classifications tasks such as for Yelp, where there is a noticeable class imbalance. Micro averaging aggregates the total true positives, false positives and false negatives and then calculates the F1 score. By contrast, macro averaging calculates the F1 score for each label and then takes the unweighted mean. For example, we have the following label counts in Yelp's training set (quite unbalanced):

Label	**No. of examples in yelp_train**
1	522
2	641
3	997
4	2468
5	2372

2.2 Baseline scores using Random Uniform and Majority-class Classifiers

The table below shows the F1 scores for the Random Uniform classifier on each of the three splits of the Yelp dataset. These scores make sense since there are fives labels, and so the likeliness of matching the label is 1/5.

Split	**F1 Score**
Train	0.20385
Valid	0.21
Test	**0.197**

F1 Scores for the Majority-class Classifier are shown below. These scores also make sense as the majority class here is 5, which makes up roughly 33.9% of the training set, 30% of the validation set, and 33.2% of the test set.

Split	**F1 Score**
Train	0.35257
Valid	0.356
Test	**0.351**

2.3 Naive Bayes Classifier

Scikit-learn's BernoulliNB classifier was used to train this dataset, since the inputs are binary vectors. The α parameter, which controls Lidstone/Laplace smoothing was tuned using Scikit's GridSearchCV. An α value of 0 indicates no smoothing, which could be problematic in the case where a feature that is not in the training set appears in the test set, leading to a posterior probability of 0. Smoothing mitigates this by always providing some non-zero probability in such cases. However, since we build a vocabulary from the training set, and then vectorize the validation and test sets based on this vocabulary, we will never encounter this issue as only words that appear in the vocabulary are considered. The following range of α values were considered and the best was found:

Parameter	**Range**	**Increment**	**Best Value**
α	[0.01, 1.00]	0.01	0.02

Here are the F1 Scores using BernoulliNB with tuning:

Split	**F1 Score**
Train	0.72814
Valid	0.672
Test	**0.4355**

2.4 Decision Tree Classifier

Scikit-learn's DecisionTreeClassifier was used. Decision Trees tend to overfit, and so must be tuned carefully. There are many hyperparameters that Scikit provides, but only three were chosen for tuning in this assignment as they were most relevant to overfitting. These are:

- max_depth: The depth of the tree (longest root to leaf path). Large depth may cause overfitting while a small depth may hinder its ability to learn.
- max_features: The maximum number of features (as a percentage of total features) to be considered in building the tree. Decision trees tend to overfit with a large number of features, so this parameter is a good candidate for tuning.
- min_samples_leaf: Minimum number of samples required to be a leaf node. A small number may lead to overfitting while a large number causes over-generalization.

Initially, large ranges were used for tuning the model. The ranges were narrowed iteratively until good ranges were found for each of the parameters (to save computation time). Scikit's documentation was used to come up with the initial ranges. Below are the final ranges used for tuning with GridSearchCV:

Parameter	**Range**	**Increment**	**Best Value**
max_depth	[13, 16]	1	16
max_features	[0.1, 0.4]	0.1	0.3
min_samples_leaf	[3, 5]	1	5

The F1 scores for each split:

Split	**F1 Score**
Train	0.57128
Valid	0.594
Test	**0.376**

The scores, even with tuning are rather low across all sets. One cause of this could be due to the unbalanced nature of the Yelp dataset. Classes 4 and 5 dominate throughout all three splits, and this could have led to a biased decision tree.

2.5 Linear SVM Classifier

Scikit-learn's LinearSVC was used. SVMs work well in high dimensions such as in our case, even when the number of features exceeds the number of examples. We have 7000 training examples in the Yelp dataset, which is 3000 less than the number of features. The following hyperparameters were tuned:

- C: This is the penalty parameter. A large value of C indicates a greater penalty term, meaning that the classifier will try to correctly classify all of the training points by choosing a smaller margin hyperplane. This may lead to overfitting. Conversely, a smaller C chooses a larger margin hyperplane at the cost of misclassifying some points.
- max_iter: The maximum number of iterations to run for training. It was noticed that the default value of 1000 performs worse than smaller values when used in combination with the penalty parameter C.

Similar to the approach used for finding the ranges for the Decision Tree Classifier, a large range was first chosen and then narrowed down iteratively. The final ranges used for GridSearchCV are as follows:

Parameter	**Range**	**Increment**	**Best Value**
C	$\{0.01, .0373, 0.1389, 0.5179, 1.9307, 7.1969, 26.827, 10\}$	Logbase 10	0.01
max_iter	[10, 90]	10	10

The F1 scores for each split:

Split	F1 Score
Train	0.83271
Valid	0.827
Test	**0.5085**

LinearSVC performs much better than the other classifiers. However, the score is still relatively low. This is likely because of the number of dimensions exceeding the number of examples. Even though SVMs are effective in such cases (it outperforms the rest), it is still affected by the lack of examples/too many features. Dimensionality reduction would likely increase the F1 score.

3 Yelp Frequency BoW Classifications and F1 Scores

The same hyperparameter ranges were used here for consistency.

3.1 Baseline scores using Random Uniform and Majority-class Classifiers

Note that frequency has no impact whatsoever on the performance of these baseline classifiers, as the random and majority classifiers do not use this additional information. The results for the random classifier are similar, and the the results for the majority classifier is identical to that of binary BoW.

F1 Scores for Random Uniform Classifier:

Split	F1 Score
Train	0.20257
Valid	0.193
Test	**0.191**

F1 Scores for Majority-class Classifier:

Split	F1 Score
Train	0.35257
Valid	0.356
Test	**0.351**

3.2 Naive Bayes Classifier

Scikit-learns's GaussianNB was used, as specified by the assignment (since the feature vectors are floats due to normalization). Scikit provides no hyperparameters to tune for this classifier. Below are the F1 scores: F1 Scores for Majority-class Classifier:

Split	F1 Score
Train	0.747
Valid	0.278
Test	**0.284**

3.3 Decision Tree Classifier

Using the same ranges as for Question 2, the following best values for each of the three hyperparameters were found:

Parameter	Best Value
max_depth	13
max_features	0.1
min_samples_leaf	4

F1 Scores:

Split	F1 Score
Train	0.549
Valid	0.519
Test	**0.3865**

We can see that the performance is similar when compared to binary BoW. However, we notice that although frequency BoW has lower scores for training and validation sets, it has a slightly higher score for the test set.

3.4 Linear SVM Classifier

Using the same rangers as for Question 2, the following best values for each of the two hyperparameters were found:

Parameter	Best Value
C	0.51795
max_iter	20

F1 Scores:

Split	F1 Score
Train	0.74771
Valid	0.752
Test	**0.5345**

Similar to what was observed for the Decision Tree, Linear SVM for frequency BoW performs worse on training and validation, but better on test.

3.5 Comments

One key factor that is likely causing poor performance across all the classifiers is the quality of the features. Many of the most frequent words in the vocabulary are stop words such as 'the' and 'and', that add no value. Either removing the stop words from the vocabulary, or using The TfidfVectorizer instead of CountVectorizer would likely provide better results. TfIdf stands for Term Frequency Inverse Document Frequency. Term Frequency measures the frequency of a word, whereas Inverse Document Frequency measures the importance of a word. This is useful for weighing down frequent stop words like 'the', and thus giving it less importance.

Another factor previously mentioned was that the number of features exceeds the number of examples in the training set (by 3000). This affects both the Linear SVM and Decision Tree (although decision tree does not use all the features during tuning).

We notice that GaussianNB performs terribly (worse than the majority class classifier) for both the validation and test sets. This is likely due to the high frequencies of stop words that skew the probability away from its true label. This is less pronounced in the binary BoW case as all words are given equal weight.

Finally, we notice that LinearSVM beat all other classifiers for both binary and frequency BoW. Its advantage in high-dimensional/fewer examples scenarios made it outperform Decision Trees, which was likely still too complex after tuning.

Overall, the scores of frequency BoW are still similar to those of binary BoW for Yelp. It seems that this additional info would have been useful had we reduced the number of features, removed stop words from the vocabulary or had more training examples.

4 Repeat 2 and 3 with IMDB Dataset

This is binary classification task with a balanced dataset (equal number of positive and negative labels for train, valid and test). Better results are expected here.

The same hyperparameter ranges from Yelp were used here.

4.1 IMDB Binary BoW

4.1.1 Random Uniform Classifier

Since this is a binary classification task, the chances of getting a label right is 0.5. Therefore these results make sense.

Split	F1 Score
Train	0.497
Valid	0.4997
Test	**0.49744**

4.1.2 Bernoulli Naive Bayes Classifier

The Naive Bayes Classifier performs very well on the IMDB dataset with the following tuned parameters. This can be attributed to an easier classification task (binary vs 5-class)

Tuned Parameters:

Parameter	Best Value
α	0.11

F1 Score:

Split	F1 Score
Train	0.8692
Valid	0.8668
Test	**0.84036**

4.1.3 Decision Tree Classifier

Decision Tree also outperformed Yelp for Binary BoW. We have more training data to work with (15000 examples), and so the tree is less likely to overfit. However, it still falls short to naive bayes.

Tuned Parameters:

Parameter	Best Value
max_depth	16
max_features	0.4
min_samples_leaf	3

F1 Score:

Split	F1 Score
Train	0.8032
Valid	0.8105
Test	**0.7314**

4.1.4 Linear SVM Classifier

Linear SVM also outperformed Yelp for Binary Bow. As we have seen with yelp, Linear SVM has outperformed all other classifiers.

Tuned Parameters:

Parameter	Best Value
C	0.01
max_iter	10

F1 Score:

Split	F1 Score
Train	0.95387
Valid	0.9523
Test	**0.87952**

4.2 IMDB Frequency BoW

4.2.1 Random Uniform Classifier

Again, frequency makes no difference to the random classifier. We have expected results close to 0.5, which is the probability of a random guess.
F1 Score:

Split	F1 Score
Train	0.50293
Valid	0.4964
Test	**0.50316**

4.2.2 Gaussian Naive Bayes

GaussianNB greatly outperforms Yelp frequency BoW, but falls short compared to Imdb BoW. This is expected since the high frequencies of stop words affect naive bayes the most.
F1 Score:

Split	F1 Score
Train	0.86933
Valid	0.7673
Test	**0.70544**

4.2.3 Decision Tree Classifier

Here, we notice that Decision Tree actually performs worse for frequency BoW as compared to binary BoW for IMDB. However, the results are quite close for all datasets.

Tuned Parameters:

Parameter	Best Value
max_depth	14
max_features	0.4
min_samples_leaf	5

F1 Score:

Split	F1 Score
Train	0.78246
Valid	0.7844
Test	**0.71776**

4.2.4 Linear SVM Classifier

Tuned Parameters:

Parameter	Best Value
C	1.9307
max_iter	40

F1 Score:

Split	F1 Score
Train	0.94687
Valid	0.9421
Test	**0.88276**

4.3 Comments

The much better performance of IMDB over Yelp can be attributed to a few key differences in their datasets:

1. IMDB had more than double the training examples (15000) than Yelp (7000)
2. IMDB was a binary classification task whereas Yelp was a 5-class classification task
3. IMDB had enough training data to exceed the number of features

The performance of binary BoW was similar to that of frequency BoW (except for Naive Bayes, as explained). Frequency BoW should have performed better than binary BoW. However, the presence of high-frequency stop words negated the advantage gained by having the frequencies of all the features.

References

[1] All code and writing was done solely by me, Nabil Chowdhury. Questions were discussed at a high level with Tiffany Wang.

[2] http://scikit-learn.org/stable/modules/tree.html

[3] http://scikit-learn.org/stable/modules/svm.html#svm-classification

[4] https://stats.stackexchange.com/questions/31066/what-is-the-influence-of-c-in-svms-with-linear-kernel

[5] http://www.tfidf.com/

[6] General scikit-learn documentation for each of the classes used.