COMP 551 - Assignment 3

Tiffany Wang - 260684152

February 26, 2018

Acknowledgement

This assignment was fully completely by myself, Tiffany Wang. However, I discussed answers and methodologies with Daniel Lim, John Wu, Frank Ye and Nabil Chowdhury.

Question 1

The datasets were preprocessed by removing punctuation marks and extra spacing. The Counter method extracts the top 10,000 words (further used as model features) from the datasets. The output files of the vocabulary set are *IMDB-vocab.txt* and *yelp-vocab.txt*The output files of the vectorized samples can be found as *IMDB-<set>.txt* and *yelp-<set>.txt*.

The vectorized bag of words are generated using CounterVectorizer in the scikit-learn feature_extraction package. The frequency bag of words are normalized vectors obtained from the CounterVectorizer.

Question 2 - Yelp Dataset Binary Bag of Word

The frequencies of each class in the dataset are the following:

Dataset	Class	Freq	Dataset	Class	Freq	Dataset	Class	Freq
Training	1	522	Validation	1	84	Testing	1	143
	2	641		2	96		2	190
	3	997		3	164		3	300
	4	2468		4	356		4	702
	5	2372		5	300		5	665

1. Random Classifier

The prediction vector is randomly generated with integer entries ranging from 0 to 5, since Yelp is a five class dataset.
F1-score: train = 0.112 valid = 0.124 test = 0.109

The probability of random guesses is 20% for five classes. Since the prediction is randomly generated and the class samples counts are not balanced, it is expected that the f1_score is lower than that of random guess.

2. Majority Class Classifier

The prediction output of every sample is set to the majority class in the training set.

F1-score: train = 0.353   valid = 0.356   test = 0.351

From the frequency table, we notice that class 4 is more frequent, and the dataset is unbalanced. Therefore, the result is expected to be higher than the that of the random guess (20%). However, as the prediction is not based on any training model, the f1-score is still relatively low.

3. Naive Bayes Classifier

I used Bernoulli Naive Bayes Classifier from scikit-learn to train my model in this section. I tuned the hyper-parameter α to smooth the maximum likelihood estimator. The parameter is inversely proportional to the weight applied to the prior knowledge. I varied the hyper-parameter from 0.5 to 0.7.

F1-score without smoothing: train = 0.50   valid = 0.385   test = 0.417
F1-score without smoothing: train = 0.62   valid = 0.569   test = 0.412

Best parameter: $\alpha = 0.595$
The smoothing prevents zero probability on the features that are not present in the training set. This helps improve the prediction when the features have yet been fed into the model. Although the improvement in performance of the test set is slim. The training and validation sets' f1-score improved by 8% and 18% respectively.

4. Decision Tree Classifier

I used the Decision Tree Classifier from scikit-learn to train my model. One of the disadvantages of a decision tree may be its over complexity and tendency to overfit. Therefore, I tuned the max_depth and max_leaf_node, max_features and min_samples_leaf parameters in order to construct a decision tree with the best complexity for the model.

1. max_depth

I first varied the parameter from 100 to 500 due to the large quantity of samples in the training set. I thought it should be around log(nb_features). However, the results were not optimal, so I lowered to 10 to 20, where there was an improvement of 5% to 6% better.

2. max_leaf_node

The varied the max_leaf_nodes parameter from 2000 to 4000. It is not optimal to create too many nodes with single samples in the tree. Therefore, setting a maximum number of nodes helps reduce the complexity in this term. The number of leaf nodes should not be too small compared to the number of samples in order to avoid generalization

3. max_features

I varied the parameter from 1000 to 3000. Since considering all the 10,000 features would lead to a overly complex tree. The hyper range was chosen with the reasoning as for max_leaf_node.

4. min_samples_leaf

This correlates inverse proportionally with the max_depth. I varied this parameter from 3 to 8, as the recommended value in the scikit-learn documentation was 5 [1].

Tree Classifier Results

F1-score: train = 0.489 valid = 0.482 test = 0.412

Best parameters:	max_depth = 10	max_leaf_node = 3250
	max_features = 2100	min_samples_leaf = 7

Although the Decision Tree Classifier deals with noisy data, its performance is not idea because the number of samples is a lot less (30%) than the number of features. The hyper-parameters helped reduce the complexity of the tree, yet the poor performance can be explained by the possibility of overfitting caused by its overly complex structure.

5. Linear SVM Classifier

I used the LinearSVC model from scikit-learn for this section. I chose the "one-vs-rest" multi-class model because this would allow the SVM to train each class as its own model, essentially binarizing the dataset for each class. In terms of parameter tuning, I varied max_iter from 500 to 1000, as the default is set to 1000, and lower iterations would restrict the model from over complexing the decision boundary and render better results.

F1-scores: train = 0.996 valid = 0.995 test = 0.444
Best parameters: max_iter = 550

Linear SVM is effective in high dim space, including when the quantity of samples is less than the dimension. It also trains the model as five different binary models. The performances on the training and validation sets are precise, as expected since the parameters were trained with these sets, yet they are still strong compared to the other classification models. This means that linear decision boundary is a good representation of the data. The prediction on the test set is however still lower than 50%, which could be explained with the sparsity and the high dimensionality of the dataset.

Relative Performance
This is a multi-class problem, therefore Random and Majority Class classifiers do not fit. The high performance scores of Naive Bayes and Linear SVM show that a linear decision boundary is a good representation of for the dataset. However, SVM beats Naive Bayes due to its high performance in high dimensional spaces. Although Decision Tree are powerful to predict non-linear complex data, it has a greater over-fitting disadvantage on linear systems, which can be observe by its less precise performance.

Question 3 - Yelp Dataset Frequency Bag of Word

Using the same hyper-parameter tuning ranges as the in Question 2, I obtained the following prediction results using the frequency bag of words.

- Random Classifier
  - f1_score train: 0.122
  - f1_score valid: 0.123
  - f1_score test: 0.127
- Majority Class Classifier
  - f1_score train: 0.3525
  - f1_score valid: 0.356
  - f1_score test: 0.351
- Naive Bayes Classifier
  - f1_score train: 0.738
  - f1_score valid: 0.273
  - f1_score test: 0.285
- Decision Tree Classifier
  - f1_score train: 0.562
  - f1_score valid: 0.531
  - f1_score test: 0.388

Best Parameters:	max_depth = 12	max_leaf_node = 2000
	max_features = 1900	- min_samples_leaf = 5

- Linear SVM Classifier
  - f1_score train: 0.806
  - f1_score valid: 0.807
  - f1_score test: 0.525

  Best Parameters: max_iter = 20

The prediction scores of both Random and Majority Class Classifiers are similar because they did not depend on the input samples. The three training models show a worse performance with this dataset. In fact, stop words, such as ”the” and ”and”, hold heavier weights in the model and overtake the importance of the keywords, for instance ”best” and ”bad”. This problem affects Naive Bayes the most, as it ruins the probabilities of each feature, and renders Naive Bayes to have an even worse performance than Majority Class. Otherwise, the relative performance between Linear SVM and Decision Tree remains the same.

A way to avoid this problem would be to remove the stop words, or to construct the frequency bag of words with inverse-document-frequency reweighting. This would be done with scikit-learn TfidfVectorizer, with use_idf = True.

Question 4 - IMDB Dataset

IMDB Binary Bag of Word

- Random Classifier
  - f1_score train: 0.497
  - f1_score valid: 0.493
  - f1_score test: 0.507
- Naive Bayes Classifier
  - f1_score train: 0.814
  - f1_score valid: 0.821
  - f1_score test: 0.734
- Decision Tree Classifier
  - f1_score train: 0.793
  - f1_score valid: 0.798
  - f1_score test: 0.734

  Best Parameters:
  max_depth = 17
  max_leaf_node = 4000
  max_features = 3000
  min_samples_leaf = 7
- Linear SVM Classifier
  - f1_score train: 0.999
  - f1_score valid: 1.0
  - f1_score test: 0.834

  Best Parameters: max_iter = 1000

IMDB Frequency Bag of Word

- Random Classifier
  - f1_score train: 0.499
  - f1_score valid: 0.494
  - f1_score test: 0.500
- Naive Bayes Classifier
  - f1_score train: 0.869
  - f1_score valid: 0.767
  - f1_score test: 0.707
- Decision Tree Classifier
  - f1_score train: 0.797
  - f1_score valid: 0.795
  - f1_score test: 0.726

  Best Parameters:
  max_depth = 14
  max_leaf_node = 4000
  max_features = 4000
  min_samples_leaf = 7
- Linear SVM Classifier
  - f1_score train: 0.931
  - f1_score valid: 0.928
  - f1_score test: 0.885

  Best Parameters: max_iter = 20

IMDB is a balanced two-class dataset, which help improve the prediction performance for all models. Especially, the Random Classifier has a close to theoretical performance of 50%. However, we can observe that Naive Bayes' performance is comparable against Decision Tree. This is due to the simplicity and the two-class property of the datasets which allow both models to perform well.

Reference

1. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

   – Decision Tree: http://scikit-learn.org/stable/modules/tree.html

   – Linear SVM: http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

   – Naive Bayes: http://scikit-learn.org/stable/modules/naive_bayes.html