Xiaoyu Lu
xlu14@vols.utk.edu

Assignment 1: Naive Bayes classification of news articles

1 Introduction

The objective of this assignment is to use the Naive Bayes algorithm to build a classifier to automatically categorize news articles into different topics. I used the Reuter's dataset (`Reuters-21578`), which include thousands of news article items, each with its own topic label. The data are saved in 22 separate `*.sgm` files. For this assignment, I am only focusing on the below topics:

> money, fx, crude, grain, trade, interest, wheat, ship, corn, oil, dlr, gas, oilseed, supply, sugar, gnp, coffee, veg, gold, soybean, bop, livestock, cpi.

2 Data pre-processing

The following technical steps were taken to pre-process the dataset:

- Parsing the `*.sgm` files to create topic/body pairs. I used the `BeautifulSoup` library in `Python` to parse the raw data, and a list was created to store the articles of interest. Each news article was saved as a `(topic, body)` tuple.
- Handle the multi-labeled articles. Some of the articles have multiple topics that belong to the topic list, and these topics are separated with a hyphen. A very common example is `fx-money`, which means that the article is relevant to both topics fx and money. In this assignment, I created duplicates of articles of multiple topics of interest, and stored each `(topic, body)` as a separate entry. This would compromise the model performance, since only one type of prediction of the identical articles would be considered correct.
- Removing modifiers, numbers, and stop words to reduce the noise. The `NLTK` and `re` libraries provides strong tools for such purposes. The `re` was used to remove all punctuation and numbers from the articles, and the `NLTK` library provides a collection of stop words based on which the articles are abridged. To remove the modifiers (mainly tense-related), the `PorterStemmer2` (also known as the `SnowballStemmer`) was used.

The final output of the processing is saved as `train_test_data.txt`. A total number of 3625 articles were retrieved. The top 10 articles are saved as `top10.txt`.

3 Building a representation that transforms text for learning

The articles were transformed using the TF-IDF model within `sklearn` and `pyspark`, separately. The two libraries provides the necessary tools to perform vectorization and transformation of the data.

It took `sklearn` 0.370s to process the entire TF-IDF transformation; where as `pyspark` only used 0.023s to finish the transformation. The transformed articles formed a sparse matrix which will be the features in our classifier.

4 Training a Naive Bayes Classifier

We randomly split the data three ways into the training/testing/validation sets. Three splits were used for this purpose:(1) 50/40/10; (2) 60/30/10; (3)70/20/10. For each split condition, we trained the model for 10 times, and the parameters and accuracy of each model are stored as key-value pairs in a dictionary. The performance of each split condition is shown in Table 1

Table 1: Model accuracy

Split	**Accuracy**
50/40/10	0.491
60/30/10	0.500
70/20/10	0.509

The test set was used to obtain the accuracy of each split. The mean accuracy of the 30 models was 0.500. Generally, the accuracy of the model increases as we have a higher proportion of training data. In this case the highest performing model was produced with a split of 70/20/10 with a mean accuracy of 0.509.

5 Testing the Naive Bayes Classifier

The validation set was used to obtain the final accuracy of the classifier. The best model performance on the validation set was an accuracy of 0.52, achieved with the train/test/cv split of 70/20/10.

6 Summary

For this assignment, I built a simple Naive Bayes classifier to classify articles based on the topics. As the result suggests, when using the highest percent of data for training, the model produced the highest accuracy. A possible future development might be to gather more data to see if the accuracy improves.

Only a limited number of topics were used for this assignment. It would be interesting to see what the model performance will be when more topics are used. Also, we only used single topic for each article entry, without considering multi-labeled articles.

Last, I trained the model based on the TFIDF result created using the cleaned text. I did not consider the combination of multiple words and their influence to the model. Ngram provides such capability, and it would be interesting to see if the model improves.