News Classification Data Mining Project

Junghwan Yim
Jun-Won Sung

University at Buffalo
School of Engineering and Applied Sciences

Introduction

- **Lots of news sources**
  - News is produced by a company or broadcasting station, an uncountable amount of news per day.
- **Lost of News Keywords**
  - News is written on a variety of topics, including entertainment, economics, politics, and international affairs.

DataSet

- **Get News Dataset**
  - News datasets can be collected using crawlers or newspapers such as The New York Times, Washington Post, and Bloomberg.
- **Remove Stopwords**
  - Designate words or expressions that can come from all kinds of categories as stopwords.

newsgroups_train
alt.atheism
comp.graphics
comp.os.ms-windows.misc
rec.sport.baseball
soc.religion.christian
…

Stopwords
"for", "com", 'as', "have", "this", "be", "are", "not", "edu", "it", "and", "in", "you", "that", "is", "of", "to", "the", 'just', 'line'

Remove Stopwords

```
[ ]  from sklearn.feature_extraction import text
     cats = list(newsgroups_train.target_names)
     newsgroups_train = fetch_20newsgroups(subset='train', categories=cats)
     my_stop_words =  text.ENGLISH_STOP_WORDS.union(
         ["for", "com", 'as', "have", "this", "be", "are", "not", "edu", "it", "and", "in", "you", "that", "is", "of", "to", "the", 'just', 'line']
     )
```

TF-IDE

- **Term Frequency - Inverse Document Frequency**
  - TF (term frequency) is a value indicating how often a particular word appears in a document. The higher this value, the more important it can be considered in the document.
  - However, the importance of a word decreases if it does not appear much in one document and appears frequently in other documents.
  - It is called DF (document frequency), and the inverse of this value is called IDF (inverse document frequency).
  - TF-IDF is a value obtained by multiplying TF and IDF, and the higher the score, the less frequently it appears in other documents.

TF-IDE

```
[ ]  from sklearn.feature_extraction.text import TfidfVectorizer
     vectorizer = TfidfVectorizer(stop_words=my_stop_words)
     vectors = vectorizer.fit_transform(newsgroups_train.data)
     vectors_test = vectorizer.transform(newsgroups_test.data)

[ ]  from sklearn.model_selection import cross_val_score
     from sklearn.model_selection import train_test_split
     cv_train_features = vectors
     train_label_names = newsgroups_train.target
     cv_test_features = vectors_test
     test_label_names = newsgroups_test.target
```

Models

- **Naive Bayes Model**
  - Naive Bayes is a classification technique that is widely used in spam filters, text classification, sentiment analysis, and recommendation systems.

- **Support Vector Machine**
  - SVM are a set of supervised learning methods used for classification, regression and outlier detection.

- **Random Forest**
  - A random forest is an ensemble machine learning model. It makes a classification by aggregating the classifications of many decision trees.

Naive Bayes Model

```
[ ]  %%time
     from sklearn.naive_bayes import MultinomialNB
     mnb = MultinomialNB(alpha=1)
     mnb.fit(cv_train_features, train_label_names)
     mnb_bow_cv_scores = cross_val_score(mnb, cv_train_features, train_label_names, cv=5)
     mnb_bow_cv_mean_score = np.mean(mnb_bow_cv_scores)
     print('CV Accuracy (5-fold):', mnb_bow_cv_scores)
     print('Mean CV Accuracy:', mnb_bow_cv_mean_score)
     mnb_bow_test_score = mnb.score(cv_test_features, test_label_names)
     print('Test Accuracy:', mnb_bow_test_score)
```

```
CV Accuracy (5-fold): [0.8833407  0.88245692 0.88024746 0.87671233 0.88152078]
Mean CV Accuracy: 0.8808556359503378
Test Accuracy: 0.817578332448221
CPU times: user 1.25 s, sys: 0 ns, total: 1.25 s
Wall time: 1.26 s
```

SVM

```
[ ]  %%time
     from sklearn.svm import LinearSVC
     svm = LinearSVC(penalty='l2', C=1, random_state=42)
     svm.fit(cv_train_features, train_label_names)
     svm_bow_cv_scores = cross_val_score(svm, cv_train_features, train_label_names, cv=5)
     svm_bow_cv_mean_score = np.mean(svm_bow_cv_scores)
     print('CV Accuracy (5-fold):', svm_bow_cv_scores)
     print('Mean CV Accuracy:', svm_bow_cv_mean_score)
     svm_bow_test_score = svm.score(cv_test_features, test_label_names)
     print('Test Accuracy:', svm_bow_test_score)
```

```
CV Accuracy (5-fold): [0.91736633 0.92797172 0.92355281 0.92355281 0.92882405]
Mean CV Accuracy: 0.9242535416747251
Test Accuracy: 0.8509028146574615
CPU times: user 16.2 s, sys: 0 ns, total: 16.2 s
Wall time: 16.2 s
```

❖ Random Forest

```
[ ]  %%time
     from sklearn.ensemble import RandomForestClassifier
     rfc = RandomForestClassifier(n_estimators=10, random_state=42)
     rfc.fit(cv_train_features, train_label_names)
     rfc_bow_cv_scores = cross_val_score(rfc, cv_train_features, train_label_names, cv=5)
     rfc_bow_cv_mean_score = np.mean(rfc_bow_cv_scores)
     print('CV Accuracy (5-fold):', rfc_bow_cv_scores)
     print('Mean CV Accuracy:', rfc_bow_cv_mean_score)
     rfc_bow_test_score = rfc.score(cv_test_features, test_label_names)
     print('Test Accuracy:', rfc_bow_test_score)

     CV Accuracy (5-fold): [0.72249227 0.73486522 0.73840035 0.74326116 0.73916888]
     Mean CV Accuracy: 0.7356375756851171
     Test Accuracy: 0.6577270313329793
     CPU times: user 40 s, sys: 25.5 ms, total: 40 s
     Wall time: 40.2 s
```

Conclusion

- As a result, the accuracy was highest in the order of SVM, Naive Bayes, and Random Forest. However, in terms of runtime, Naive Bayes was faster than SVM. Random Forest has the least performance and time-efficiency, which can be analyzed as a result of this data being too sparse.

Thank you