COMPSCI 273A IMDB Analysis

December 13, 2019

0.1 Introduction

[1]:
```
import os
import json
import string
import nltk
import sklearn.linear_model
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier as SGD
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.naive_bayes import MultinomialNB as MNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
import numpy as np
import pandas as pd
from nltk.corpus import stopwords
```

[39]:
```
# Remove punctuations and stopwords in documents
def ReadOneFile(fileName):
    contents = []
    with open(fileName, 'r', encoding='UTF-8') as file:
        for line in file:
            contents.append(line.rstrip('\n').lower())
    result = ''.join(contents)

    #remove punctuations
    special_char = ["‘", "’", ".", "-", "“", "”"]
    result = result.translate(str.maketrans('', '', string.punctuation)).
 ↪translate({ord(c): 'special char' for c in special_char})
    result = result.split()

    #remove stopwords
    stop_words = stopwords.words('english')
    result = [w for w in result if w not in stop_words]
```

```
    return result

# Processing documents to save in format as [{word_1: count, word_2: count, ...
 ↪_FileID_: DocID, _CLASS_: 0 or 1},...]
def ReadFiles(fileName):
    data = []
    directory_top = "C:/Users/nicho/Desktop/IMDB_Dataset/" + fileName + "/"
    for data_class in os.listdir(directory_top):
        directory_class = directory_top + data_class + "/"
        for file in os.listdir(directory_class):
            words = ReadOneFile(directory_class + file)
            example = {x:words.count(x) for x in words}
            example['__FileID__'] = file
            example['__CLASS__'] = 1 if data_class[:3] == 'pos' else 0
            data.append(example)
    return data
```

[ ]:
```
data_train = ReadFiles("train")
# data_test = ReadFiles("test")
df = pd.DataFrame(data_train).fillna(0)
```

Because the processing ran pretty slow inside Jupyter, we did the processing in Python IDE and saved the results to local.

[23]:
```
# load results
with open(r'C:\Users\nicho\Desktop\data_train_wostopNpunc.txt', 'r',
 ↪encoding='UTF-8') as f:
    data_train = json.load(f)
```

Processing all the documents in train folders will result in an 25000 x 121224 array. However, this size exceeds the maximum size that can be allocated. Therefore, we will choose only 2000 documents here instead of all doucments. 1000 from neg and 1000 from pos

[72]:
```
first_1000 = data_train[:1000]
last_1000 = data_train[-1000:]
```

[73]:
```
df = pd.DataFrame(first_1000 + last_1000).fillna(0)
print(df.shape)
```

```
(2000, 28391)
```

[74]:
```
df.head()
```

[74]:

	story	man	unnatural	feelings	pig	starts	opening	scene	terrific	\
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	
1	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

```
3    1.0  0.0        0.0       0.0  0.0     0.0      0.0    2.0       0.0
4    1.0  0.0        0.0       0.0  0.0     0.0      0.0    0.0       0.0

   example  ...  barkers  verges  rosyhued  bluecollar  overtures  illiteracy  \
0      1.0  ...      0.0     0.0       0.0         0.0        0.0         0.0
1      0.0  ...      0.0     0.0       0.0         0.0        0.0         0.0
2      0.0  ...      0.0     0.0       0.0         0.0        0.0         0.0
3      0.0  ...      0.0     0.0       0.0         0.0        0.0         0.0
4      0.0  ...      0.0     0.0       0.0         0.0        0.0         0.0

   plottool  colorless  fluffand  swallowthough
0       0.0        0.0       0.0            0.0
1       0.0        0.0       0.0            0.0
2       0.0        0.0       0.0            0.0
3       0.0        0.0       0.0            0.0
4       0.0        0.0       0.0            0.0

[5 rows x 28391 columns]
```

0.2 Training/Validation Split

For now, we will do the test only on these 2000 docuement to see the performance. Therefore, we will split the data into training set and validation set. Later on, we will do the testing with test sets.

[77]:
```
features = df.drop(['__FileID__', '__CLASS__'], axis=1)
labels = df.__CLASS__
X_train, X_val, Y_train, Y_val = sklearn.model_selection.
 ↪train_test_split(features, labels, test_size=0.2, random_state=42)
```

[79]:
```
print(X_train.shape, X_val.shape, Y_train.shape, Y_val.shape)
```

```
(1600, 28389) (400, 28389) (1600,) (400,)
```

0.3 Multiple Model Implementations

***Note: Because the models are running pretty slow in Juptyer, we ran all the models locally and saved the result.**

0.3.1 Logistic Regression

[81]:
```
LogisticReg = sklearn.linear_model.LogisticRegression(solver='lbfgs')
LogisticReg.fit(X_train, Y_train)
print("Training accuracy:", LogisticReg.score(X_train, Y_train), "\nValidation␣
 ↪accuracy:",
      LogisticReg.score(X_val, Y_val))
```

```
Training accuracy: 1.0
Validation accuracy: 0.89
```

0.3.2 Signle Decision Tree

[ ]:
```
DecisionTree = tree.DecisionTreeClassifier(criterion='entropy')
DecisionTree.fit(X_train, Y_train)
print("Training acc:", DecisionTree.score(X_train, Y_train), "\nValidation acc:",
      DecisionTree.score(X_val, Y_val))
```

```
Training acc: 1.0
Validation acc: 0.685
```

[ ]:
```
parameters = {"max_depth": [None, 10, 100, 1000],
              "min_samples_split": [5, 10, 50, 100, 500, 1000],
              "min_samples_leaf": [10, 100, 1000],
              "max_leaf_nodes": [None, 10, 100, 1000],
              }
dt_search = GridSearchCV(DecisionTree, parameters)
dt_search.fit(X_train, Y_train)
print("The best parameters: " + str(dt_search.best_params_))
```

The best parameters: {'max_depth': None, 'max_leaf_nodes': 1000, 'min_samples_leaf': 10, 'min_samples_split': 50}

[ ]:
```
DecisionTree2 = tree.DecisionTreeClassifier(criterion = "entropy", max_depth =␣
↪None, max_leaf_nodes = 1000,
                                            min_samples_leaf = 10,␣
↪min_samples_split = 50)
DecisionTree2.fit(X_train, Y_train)
print("Training acc:", DecisionTree2.score(X_train, Y_train), "\nValidation acc:
↪",
      DecisionTree2.score(X_val, Y_val))
```

```
Training acc: 0.82
Validation acc: 0.6625
```

0.3.3 Adaboost

[ ]:
```
Boost = AdaBoostClassifier(base_estimator=DecisionTree2, n_estimators=100)
Boost.fit(X_train, Y_train)
print("Training acc:", Boost.score(X_train, Y_train), "\nValidation acc:",
      Boost.score(X_val, Y_val))
```

```
Training acc: 1.0
Validation acc: 0.845
```

0.3.4 Random Forests

[ ]:
```
RandomForest = RandomForestClassifier(criterion = 'entropy', n_estimators=100)
RandomForest.fit(X_train, Y_train)
print("Training acc:", RandomForest.score(X_train, Y_train), "\nValidation acc:",
      RandomForest.score(X_val, Y_val))
```

```
Training acc: 1.0
Validation acc: 0.855
```

[ ]:
```
parameters = {"min_samples_split": [2, 5, 10, 20],
              "max_depth": [None, 2, 5, 10, 20],
              "min_samples_leaf": [1, 5, 10, 20],
              "max_leaf_nodes": [None, 5, 10, 20, 50],
              }
rfc_search = GridSearchCV(RandomForest, parameters)
rfc_search.fit(X_train, Y_train)
print("The best parameters: " + str(rfc_search.best_params_))
```

```
The best parameters: {'max_depth': None, 'max_leaf_nodes': None, 'min_samples_leaf': 1, 'min_sam
```

[ ]:
```
RandomForest2 = RandomForestClassifier(criterion = "entropy", max_depth = None,␣
 ↪max_leaf_nodes = None,
                                        min_samples_leaf = 1,␣
 ↪min_samples_split = 5)
RandomForest2.fit(X_train, Y_train)
print("Training acc:", RandomForest2.score(X_train, Y_train), "\nValidation acc:
 ↪",
      RandomForest2.score(X_val, Y_val))
```

```
Training acc: 1.0
Validation acc: 0.88
```

0.3.5 AdaBoost

[ ]:
```
Boost = AdaBoostClassifier(base_estimator=RandomForest2, n_estimators=100)
Boost.fit(X_train, Y_train)
print("Training acc:", Boost.score(X_train, Y_train), "\nValidation acc:",
      Boost.score(X_val, Y_val))
```

```
Training acc: 1.0
Validation acc: 0.8725
```

0.3.6 SVM

[ ]:
```
SVM = SVC(probability=True)
SVM.fit(X_train, Y_train)
print("Training acc:", SVM.score(X_train, Y_train), "\nValidation acc:",
```

```
      SVM.score(X_val, Y_val))
```

```
Training acc: 0.988125
Validation acc: 0.8775
```

[ ]:
```
parameters = [{'kernel': ['rbf'], 'gamma': [0.01, 0.005, 0.001], 'C': [0.5, 1, 1.
↪5, 2, 4]},
              {'kernel': ['linear'], 'C': [0.001, 0.01, 0.1, 1]}]
svm_search = GridSearchCV(SVM, parameters, cv=5, scoring="roc_auc", n_jobs=4)
svm_search.fit(X_train, Y_train)
print("The best parameters: " + str(svm_search.best_params_))
```

```
The best parameters: {'C': 4, 'gamma': 0.001, 'kernel': 'rbf'}
```

[ ]:
```
SVM2 = SVC(probability=True, kernel='rbf', C=4 ,gamma=0.001)
SVM2.fit(X_train, Y_train)
print("Training acc:", SVM2.score(X_train, Y_train), "\nValidation acc:",
      SVM2.score(X_val, Y_val))
```

```
Training acc: 0.985625
Validation acc: 0.8975
```

0.3.7 Multiple Naive Bayes

[ ]:
```
NaiveBayes = MNB()
NaiveBayes.fit(X_train, Y_train)
print("Training acc:", NaiveBayes.score(X_train, Y_train), "\nValidation acc:",
      NaiveBayes.score(X_val, Y_val))
```

```
Training acc: 0.99
Validation acc: 0.9325
```

0.3.8 SGD

[ ]:
```
sgd = SGD(max_iter=5, random_state=0,loss='modified_huber',n_jobs=4)
sgd.fit(X_train, Y_train)
print("Training acc:", sgd.score(X_train, Y_train), "\nValidation acc:",
      sgd.score(X_val, Y_val))
```

```
Training acc: 0.995
Validation acc: 0.88
```

[ ]:
```
parameters = {'alpha': [0.1, 0.5, 1, 1.5]}
sgd_search = GridSearchCV(sgd,parameters , scoring='roc_auc', cv=20)
sgd_search.fit(X_train, Y_train)
print("The best parameters: " + str(sgd_search.best_params_))
```

```
The best parameters: {'alpha': 0.1}
```

[ ]:
```
sgd2 = SGD(max_iter=5, random_state=0,loss='modified_huber',n_jobs=4,alpha=0.1)
sgd2.fit(X_train, Y_train)
print("Training acc:", sgd2.score(X_train, Y_train), "\nValidation acc:",
      sgd2.score(X_val, Y_val))
```

```
Training acc: 0.99875
Validation acc: 0.9025
```

[ ]:
```

```