Benchmark Various Models for Sentiment Analysis

This notebook benchmarks Logistic regression, Random Forest and XGBoost models for sentiment analysis

Define the imports

In [1]:
```
import pandas as pd
import numpy as np
import bz2
import os
import matplotlib.pyplot as plt
import re
import nltk
```

Load database required for removing stopword and lemmatization

In [2]:
```
nltk.download('stopwords')
stop_words = set(nltk.corpus.stopwords.words('english'))

nltk.download('wordnet')
lemmatizer = nltk.stem.WordNetLemmatizer()

# Downloads all english dictionary words
nltk.download('words')
english_words = set(nltk.corpus.words.words())
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Siva\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Siva\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\Siva\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
```

Define a function to normalize words in a sentence

We do the following

- Convert all words to lower case, so we are doing not analyzing words with different case as different words
- Drop any stop words like I, me, this, is ...
- Remove words that are not in english dictionary.

- Remove punctuations
- Lemmatize words. This is converting different forms of a word to a base form. E.g convert word like caring to care, bats to bat

In [3]:
```
punctuations = "!@#$%^&*()_-+={[}]|\:;'<,>.?/~`"

def to_words(text):
    words = []
    tokens = re.findall('\w+', text)
    for w in tokens:
        # Convert to lower
        w = w.lower()

        # Remove punctuations
        w = "".join([char for char in w if char not in punctuations])

        # Don't add word if it is a stopword
        if w not in stop_words:

            # Make sure it is valid english word
            if w in english_words:
                # Lemmatize word
                w = lemmatizer.lemmatize(w, 'v')  #Assume most of the review is v
                words.append(w)

    return words
```

Define a function that will load the reviews file and convert it to normalized words and return the sentiment labels and words as array

In [4]:
```
def load_data(txt_bz_file):
    sentiments = []
    reviews = []

    with bz2.open(txt_bz_file, "rt", encoding='utf-8') as bz_file:
        for line in bz_file:
            # Label and review are separated by space
            label, review = line.split(' ', maxsplit=1)

            # label has a format __label__2  we just need the last number
            sentiments.append(int(label[9:]))

            # The title and the body are separated by :, so we split them
            title, body = review.split(':', maxsplit=1)

            title_part = " ".join(to_words(title))
            body_part = " ".join(to_words(body))

            sentence = " ".join([title_part, body_part])
            reviews.append(sentence)
    return sentiments, reviews
```

Load the training set

In [5]:
```
train_sentiments, train_reviews = load_data('../data/sample_train.ft.txt.bz2')
```

Load the test set

In [6]:
```
test_sentiments, test_reviews = load_data('../data/sample_test.ft.txt.bz2')
```

Do count vectorization and create a dataframe for train and test data

In [7]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

# max_df=0.85 - Ignore words that occur in 85% of the reviews. They are not going
# min_df=5 - Ignore words that happen less than 5 times in the entire dataset. Si
count_vect = CountVectorizer(max_df=0.85, min_df=5)

# We need the vectorizer to account for all words that may only exists in test da
count_vect.fit(train_reviews + test_reviews)
tfidf_transformer = TfidfTransformer(smooth_idf=True, use_idf=True)

train_counts = count_vect.transform(train_reviews)
train_tfidf = tfidf_transformer.fit_transform(train_counts)
train_df = pd.DataFrame(train_tfidf.toarray(),
             columns=count_vect.get_feature_names())

test_counts = count_vect.transform(test_reviews)
test_tfidf = tfidf_transformer.fit_transform(test_counts)
test_df = pd.DataFrame(test_tfidf.toarray(),
             columns=count_vect.get_feature_names())
```

LogisticRegression

Build a LogisticRegression model and see how well it performs

In [8]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

clf = LogisticRegression()
# Fit the model on the trainng data.
clf.fit(train_df, train_sentiments)

# Predict
test_sentiments_predicted = clf.predict(test_df)

# Print accuracy score and confusion matrix

print('Accuracy score of LogisticRegression = ', accuracy_score(test_sentiments,

print('Confusion Matrix for LogisticRegression')
print(confusion_matrix(test_sentiments, test_sentiments_predicted))
print('F1 Score = ', f1_score(test_sentiments, test_sentiments_predicted))
```

```
Accuracy score of LogisticRegression =  0.846
Confusion Matrix for LogisticRegression
[[420  78]
 [ 76 426]]
F1 Score =  0.8450704225352113
```

RandomForest

Build a RandomForest model and see how well it performs

In [9]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score
from sklearn.model_selection import GridSearchCV

#hyper_params = {
#    'n_estimators': [16, 32, 64, 128, 256, 500],
#    'max_features': ['auto', 'sqrt', 'log2'],
#    'max_depth' : [10, 15, 20, 30, 40],
#    'criterion' :['gini', 'entropy']
#}

#rfc = GridSearchCV(RandomForestClassifier(), hyper_params, cv= 5)
#print("Best Parameters: {}".format(rfc.best_params_))

rfc = RandomForestClassifier(n_estimators=128, max_features='sqrt', max_depth=30,
rfc.fit(train_df, train_sentiments)
test_sentiments_predicted = rfc.predict(test_df)

print("Acuracy Score for RandomForest = ", accuracy_score(test_sentiments, test_s
print('Confusion Matrix for RandomForest')
print(confusion_matrix(test_sentiments, test_sentiments_predicted))
print('F1 Score = ', f1_score(test_sentiments, test_sentiments_predicted))
```

```
Acuracy Score for RandomForest =  0.838
Confusion Matrix for RandomForest
[[409  89]
 [ 73 429]]
F1 Score =  0.8346938775510203
```

XGBoost

In [10]:
```
from xgboost import XGBClassifier

xgb = XGBClassifier()
xgb.fit(train_df, train_sentiments)
test_sentiments_predicted = rfc.predict(test_df)

print("Acuracy Score for XGBoost = ", accuracy_score(test_sentiments, test_sentim
print('Confusion Matrix for XGBoost')
print(confusion_matrix(test_sentiments, test_sentiments_predicted))
print('F1 Score = ', f1_score(test_sentiments, test_sentiments_predicted))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\xgboost\sklearn.py:1146: UserWarnin
g: The use of label encoder in XGBClassifier is deprecated and will be removed
in a future release. To remove this warning, do the following: 1) Pass option u
se_label_encoder=False when constructing XGBClassifier object; and 2) Encode yo
ur labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[19:27:23] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.
0/src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric
used with the objective 'binary:logistic' was changed from 'error' to 'loglos
s'. Explicitly set eval_metric if you'd like to restore the old behavior.
Acuracy Score for XGBoost =  0.838
Confusion Matrix for XGBoost
[[409  89]
 [ 73 429]]
F1 Score =  0.8346938775510203
```

In [ ]: