2306 AML128
BHASKARBABU ALAM

2 Implement bag of words and tf-idf using naive Bayes algorithm check the accuracy

```
import csv
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer
#from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS as
sklearn_stop_words
from sklearn import metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB

[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/sravya/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Reading Sentiment analysis data

```
train = pd.read_csv('train.tsv', sep='\t')
test = pd.read_csv('test.tsv',  sep='\t')
sampleSub = pd.read_csv('sampleSubmission.csv')
```

Analysing data

```
print(train.shape, "\n",
      test.shape
     )

(156060, 4)
 (66292, 3)

print (train.isnull().values.any(), "\n",
       test.isnull().values.any()
       )

False
 False

train.head()

   PhraseId  SentenceId
Phrase  \
0         1           1  A series of escapades demonstrating the adage
...
1         2           1  A series of escapades demonstrating the adage
...
2         3           1                                              A
series
3         4           1
A
4         5           1
series

   Sentiment
0          1
1          2
2          2
3          2
4          2
```

```
test.head()
```

```
   PhraseId  SentenceId
Phrase
0    156061        8545  An intermittently pleasing but mostly routine
...
1    156062        8545  An intermittently pleasing but mostly routine
...
2    156063        8545
An
3    156064        8545  intermittently pleasing but mostly routine
effort
4    156065        8545         intermittently pleasing but mostly
routine
```

```
len(train.groupby('SentenceId').nunique())
```

```
8529
```

```
len(test.groupby('SentenceId').nunique())
```

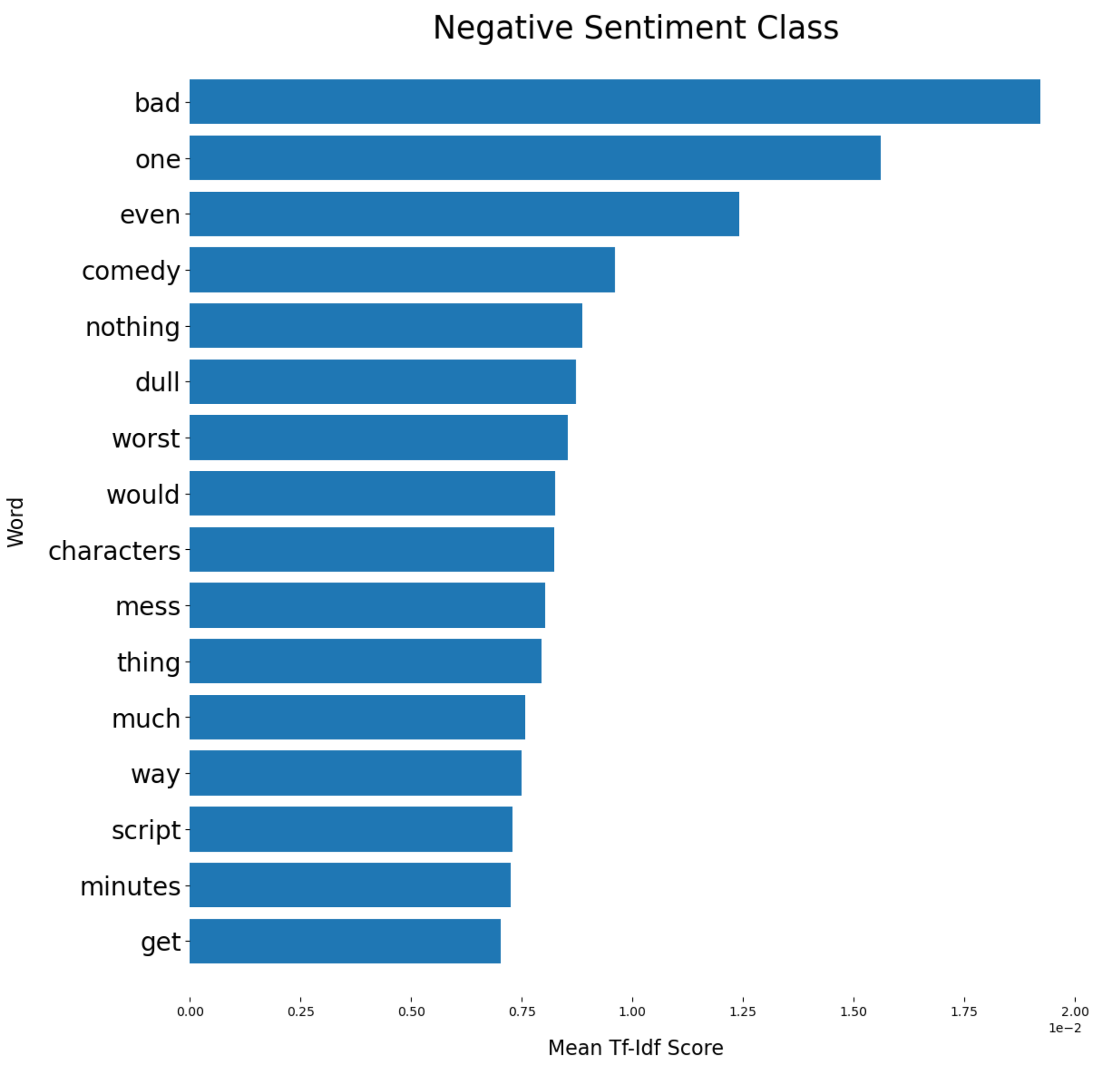
```
3310
```

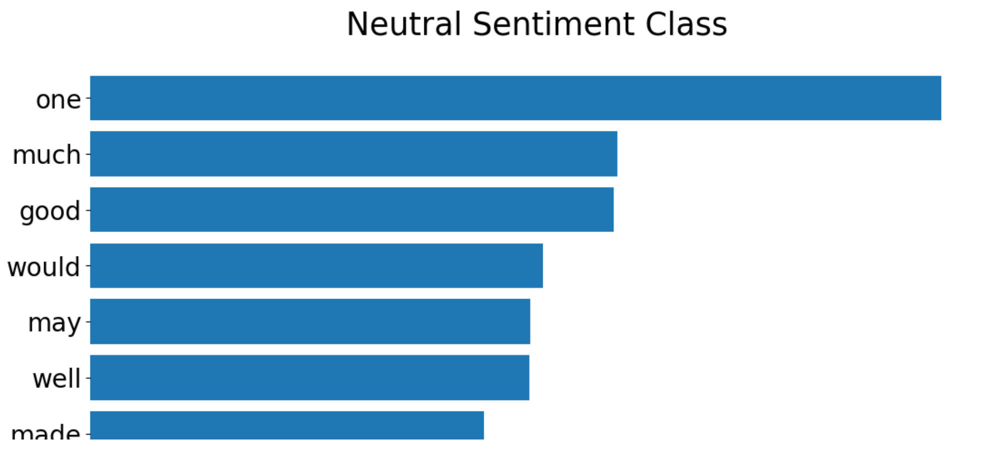
```
#Create df of full sentences
fullSent = train.loc[train.groupby('SentenceId')['PhraseId'].idxmin()]

#Change sentiment to increase readability
fullSent['sentiment_label'] = ''
Sentiment_Label = ['Negative', 'Somewhat Negative',
                  'Neutral', 'Somewhat Positive', 'Positive']
for sent, label in enumerate(Sentiment_Label):
    fullSent.loc[train.Sentiment == sent, 'sentiment_label'] = label
fullSent.head()

     PhraseId  SentenceId                                            
Phrase  \
0           1           1  A series of escapades demonstrating the 
adage ...
63         64           2  This quiet , introspective and entertaining 
in...
81         82           3  Even fans of Ismail Merchant 's work , I 
suspe...
116       117           4  A positively thrilling combination of 
ethnogra...
156       157           5  Aggressive self-glorification and a 
manipulati...

     Sentiment    sentiment_label
0            1  Somewhat Negative
63           4           Positive
81           1  Somewhat Negative
116          3  Somewhat Positive
156          1  Somewhat Negative

#Add non-helpful stopwords to stopword list
Stopwords = list(set(stopwords.words('english')))
Stopwords.extend(['movie','movies','film','nt','rrb','lrb',
                      'make','work','like','story','time','little'])

#Create tfidf vectorizer object & fit to full sentence training data
tfidf_vectorizor = TfidfVectorizer(min_df=5,
                             max_df=0.5,
                             analyzer='word',
                             strip_accents='unicode',
                             ngram_range=(1, 3),
                             sublinear_tf=True,
                             smooth_idf=True,
                             use_idf=True,
                             stop_words=Stopwords)

tfidf_vectorizor.fit(list(fullSent['Phrase']))


#Create bag of word vectorizer for comparison in evaluation section
BoW_vectorizer = CountVectorizer(strip_accents='unicode',
                                 stop_words=Stopwords,
                                 ngram_range=(1,3),
                                 analyzer='word',
                                 min_df=5,
                                 max_df=0.5)
```

```
BoW_vectorizer.fit(list(fullSent['Phrase']))

CountVectorizer(max_df=0.5, min_df=5, ngram_range=(1, 3),
                stop_words=['again', 'until', 'up', 'very', 'while',
's', 'don',
                            'weren', 'having', 'she', 'why', 'he',
'me', 'such',
                            'other', 'can', 'doesn', 'wasn', 'and',
'at',
                            'more', 'ain', 'both', 'being', 'your',
'any',
                            'himself', 'them', 'over', 'all', ...],
                strip_accents='unicode')

#functions to create graphics below from tf-idf matrices
#adapted from : https://buhrmann.github.io/tfidf-analysis.html
def top_tfidf_feats(row, features, top_n=20):
    topn_ids = np.argsort(row)[::-1][:top_n]
    top_feats = [(features[i], row[i]) for i in topn_ids]
    df = pd.DataFrame(top_feats)
    df.columns = ['feature', 'tfidf']
    return df

def top_feats_in_doc(Xtr, features, row_id, top_n=20):
    row = np.squeeze(Xtr[row_id].toarray())
    return top_tfidf_feats(row, features, top_n)

def top_mean_feats(Xtr, features, grp_ids=None, min_tfidf=0.1,
top_n=10):
    if grp_ids:
        D = Xtr[grp_ids].toarray()
    else:
        D = Xtr.toarray()

    D[D < min_tfidf] = 0
    tfidf_means = np.mean(D, axis=0)
    return top_tfidf_feats(tfidf_means, features, top_n)

def top_feats_by_class(Xtr, y, features, min_tfidf=0.1, top_n=16):
    dfs = []
    labels = np.unique(y)
    for label in labels:
        ids = np.where(y==label)
        feats_df = top_mean_feats(Xtr, features, ids,
min_tfidf=min_tfidf, top_n=top_n)
        feats_df.label = label
        dfs.append(feats_df)
    return dfs
```

```
def plot_tfidf_classfeats_h(dfs, num_class=9):
    fig = plt.figure(figsize=(12, 100), facecolor="w")
    x = np.arange(len(dfs[0]))
    for i, df in enumerate(dfs):
        ax = fig.add_subplot(num_class, 1, i+1)
        ax.spines["top"].set_visible(False)
        ax.spines["right"].set_visible(False)
        ax.set_frame_on(False)
        ax.get_xaxis().tick_bottom()
        ax.get_yaxis().tick_left()
        ax.set_xlabel("Mean Tf-Idf Score", labelpad=16, fontsize=16)
        ax.set_ylabel("Word", labelpad=16, fontsize=16)
        ax.set_title(str(df.label) + ' Sentiment Class', fontsize=25)
        ax.ticklabel_format(axis='x', style='sci', scilimits=(-2,2))
        ax.barh(x, df.tfidf, align='center')
        ax.set_yticks(x)
        ax.set_ylim([-1, x[-1]+1])
        ax.invert_yaxis()
        yticks = ax.set_yticklabels(df.feature)

        for tick in ax.yaxis.get_major_ticks():
                tick.label.set_fontsize(20)
        plt.subplots_adjust(bottom=0.09, right=0.97, left=0.15,
top=0.95, wspace=0.52)
    plt.show()

class_Xtr = tfidf_vectorizor.transform(fullSent['Phrase'])
class_y = fullSent['sentiment_label']
class_features = tfidf_vectorizor.get_feature_names_out()
class_top_dfs = top_feats_by_class(class_Xtr, class_y, class_features)
plot_tfidf_classfeats_h(class_top_dfs, 7)
```

```
/var/folders/nz/v4c3dsmn1293s89vq32wmzq00000gn/T/
ipykernel_43187/1374921368.py:55: MatplotlibDeprecationWarning: The
label function was deprecated in Matplotlib 3.1 and will be removed in
3.8. Use Tick.label1 instead.
  tick.label.set_fontsize(20)
```


Negative Sentiment Class
bad
one
even
comedy
nothing
dull
worst
would
characters
mess
thing
much
way
script
minutes
get
Word
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e−2
Mean Tf-Idf Score


Neutral Sentiment Class
one
much
good
would
may
well
made

Sentiment Analysis using Machine Learning Models

```
phrase = np.array(train['Phrase'])
sentiment = np.array(train['Sentiment'])
# build train and test datasets
phrase_train, phrase_test, sentiment_train, sentiment_test =
train_test_split(phrase,

sentiment,

test_size=0.2,

random_state=4)

#TF-IDF
train_tfidfmatrix = tfidf_vectorizor.fit_transform(phrase_train)
test_tfidfmatrix = tfidf_vectorizor.transform(phrase_test)

#Vectorizer (Bag of Words Model)
train_simplevector = BoW_vectorizer.transform(phrase_train)
test_simplevector = BoW_vectorizer.transform(phrase_test)

def train_model_predict (classifier, train_features, train_labels,
                      test_features):
    classifier.fit(train_features, train_labels)
    predictions = classifier.predict(test_features)
    return predictions
```

Naive Bayes Model

TF-IDF vectorization

```
model1 = MultinomialNB()
NBPredictions = train_model_predict(model1, train_tfidfmatrix,
sentiment_train,
                             test_tfidfmatrix)
```

'Bag of Words' vectorization

```
NBPredictions2 = train_model_predict(model1, train_simplevector,
sentiment_train,
                             test_simplevector)
```

Logistic Regression Model

TF-IDF vectorization

```
model2 = LogisticRegression(solver = 'liblinear', multi_class = 'ovr')
LogisticRegressionPredictions = train_model_predict(model2,
train_tfidfmatrix, sentiment_train,
                             test_tfidfmatrix)
```

'Bag of Words' vectorization

```
LogisticRegressionPredictions2 = train_model_predict(model2,
train_simplevector, sentiment_train,
                             test_simplevector)
```

Evaluation

```
def get_metrics(true_labels, predicted_labels, feature):
    print(feature)
    print('Accuracy:', np.round(metrics.accuracy_score(true_labels,
                                                  predicted_labels), 4))
    print('Precision:', np.round(metrics.precision_score(true_labels,
                                                  predicted_labels,
                                                  average='weighted'),
4))
    print('Recall:', np.round(metrics.recall_score(true_labels,
                                                  predicted_labels,
                                                  average='weighted'),
4))
    print('F1 Score:', np.round(metrics.f1_score(true_labels,
                                                  predicted_labels,
                                                  average='weighted'),
4))
    print('\n')

get_metrics(NBPredictions, sentiment_test, 'Naive Bayes & TF-IDF
Scores: ')
get_metrics(NBPredictions2, sentiment_test, 'Naive Bayes & Bag of
Words Scores: ')
```

```
Naive Bayes & TF-IDF Scores:
Accuracy: 0.6129
Precision: 0.7561
Recall: 0.6129
F1 Score: 0.6586


Naive Bayes & Bag of Words Scores:
Accuracy: 0.6021
Precision: 0.7198
Recall: 0.6021
F1 Score: 0.639
```

```
get_metrics(LogisticRegressionPredictions, sentiment_test, 'Logistic
Regression & TF-IDF Scores: ')
get_metrics(LogisticRegressionPredictions2, sentiment_test, 'Logistic
Regression & Bag of Words Scores: ')
```

```
Logistic Regression & TF-IDF Scores:
Accuracy: 0.6284
Precision: 0.7464
Recall: 0.6284
F1 Score: 0.6646


Logistic Regression & Bag of Words Scores:
Accuracy: 0.612
Precision: 0.739
Recall: 0.612
F1 Score: 0.6509
```

As observed above, accuracy is slightly higher for TF-IDF for both Naive Bayes and Bag of Words