Text Classification

The moviereviews2.tsv dataset contains the text of 6000 movie reviews. The text has been reduced and preprocessed as a tab-delimited file. For more information on this dataset visit http://ai.stanford.edu/~amaas/data/sentiment/

- Perform imports and load the dataset into a pandas DataFrame.
- Data Cleanup: Handle missing values, and NaN
- Split the data into train & test sets. Use test_size=0.33, random_state=42
- Build a pipeline to vectorize the data, then train and fit a model. You may use whatever model you like and LinearSVC.
- Run predictions and analyze the results. Report the confusion matrix and classification report.

In [52]:
```
import numpy as np
import pandas as pd

df = pd.read_csv('../TextFiles/moviereviews2.tsv', sep='\t')
df.head()
```

Out[52]:

	label	review
0	pos	I loved this movie and will watch it again. Or...
1	pos	A warm, touching movie that has a fantasy-like...
2	pos	I was not expecting the powerful filmmaking ex...
3	neg	This so-called "documentary" tries to tell tha...
4	pos	This show has been my escape from reality for ...

In [53]:
```
#Check for NaN values
df.isnull().sum()
```

Out[53]:
```
label      0
review    20
dtype: int64
```

There are 20 reviews that is null.

In [54]:
```
#Check for empty strings
blanks = []

for i,lb,rv in df.itertuples():
    if type(rv)==str:
        if rv.isspace():
            blanks.append(i)

len(blanks)
```

Out[54]:
```
0
```

There aren't any empty strings.

In [55]:
```
#Data Cleanup
df.dropna(inplace=True)
```

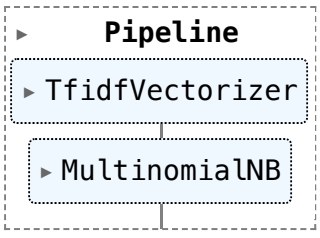
In [56]:
```
#Check for NaN values after cleanup
df.isnull().sum()
```

Out[56]:
```
label     0
review    0
dtype: int64
```

In [57]:
```
#Split train and test sets
from sklearn.model_selection import train_test_split

y = df['label']
X = df['review']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [58]:
```
#Build a pipeline
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB

text_clf = Pipeline([('tfidf', TfidfVectorizer()),
                     ('clf', MultinomialNB()),
])

text_clf.fit(X_train, y_train)
```

Out[58]:



In [59]:
```
#Run predictions and analyze the results
from sklearn import metrics

predictions = text_clf.predict(X_test)
print(metrics.confusion_matrix(y_test,predictions))
```

```
[[940  51]
 [136 847]]
```

In [60]:
```
#Print a classification report
print(metrics.classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

         neg       0.87      0.95      0.91       991
         pos       0.94      0.86      0.90       983

    accuracy                           0.91      1974
   macro avg       0.91      0.91      0.91      1974
weighted avg       0.91      0.91      0.91      1974
```

Sentiment Analysis

Task #1 : Write a function (word_vector) that takes in 3 strings(words), performs a -b + c arithmetic, and returns a top-ten closest results (cosine similarity) after performing vector arithmetic on your own words. The goal is to come as close to an expected word as possible.

In [98]:
```
import spacy
nlp = spacy.load('en_core_web_md')
```

In [99]:
```
from scipy import spatial
cosine_similarity = lambda x, y: 1 - spatial.distance.cosine(x, y)
```

In [100…
```
def word_vector(a,b,c):
    a_vector = nlp.vocab[a].vector
    b_vector = nlp.vocab[b].vector
    c_vector = nlp.vocab[c].vector

    new_vector = a_vector - b_vector + c_vector
    computed_similarities = []

    for word in nlp.vocab:
        if word.has_vector:
            if word.is_lower:
                if word.is_alpha:
                    similarity = cosine_similarity(new_vector, word.vector)
                    computed_similarities.append((word, similarity))

    computed_similarities = sorted(computed_similarities, key=lambda item: -item[1])

    return [w[0].text for w in computed_similarities[:10]]
```

In [101…
```
word_vector('forest','tree','flower')
```

Out[101]:
```
['forest',
 'flower',
 'and',
 'where',
 'these',
 'space',
 'those',
 'that',
 'there',
 'havin']
```

Task #2 : Write a function to perform VADER Sentiment Analysis on your own review. The function returns a set of "SentimentIntensityAnalyzer" polarity scores based on written review. Consider returning a score of "Positive" , "Negative" or "Neutral"

In [68]:
```
import nltk
nltk.download('vader_lexicon')
```

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

Out[68]:
```
True
```

In [69]:
```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
```

In [70]:
```
def rating_score(string):
    scores = sid.polarity_scores(string)
    if scores['compound'] == 0:
        return 'Neutral'
    elif scores['compound'] > 0:
        return 'Positive'
    else:
        return 'Negative'
```

In [71]:
```
good_review = 'The season is of to a high standard with hysterical one liners, terrific pranks from Jim, Michael's hi
```

In [72]:
```
sid.polarity_scores(good_review)
```

Out[72]:
```
{'neg': 0.028, 'neu': 0.724, 'pos': 0.248, 'compound': 0.8591}
```

In [73]:
```
rating_score(good_review)
```

Out[73]:
```
'Positive'
```

In [74]:
```
bad_review = 'I find season 8 bearable and there are some standouts but this episode is truly the worst episode of the
```

In [75]:
```
sid.polarity_scores(bad_review)
```

Out[75]:
```
{'neg': 0.205, 'neu': 0.655, 'pos': 0.14, 'compound': -0.4215}
```

In [76]:
```
rating_score(bad_review)
```

Out[76]:
```
'Negative'
```

In [ ]:
```

```