Project : Bag of Words meets Bag of Popcorn

Project Introduction

- **Project** : Analysed NYC-Flight data
- **Technology use** : Python with NumPy and pandas
  : Machine Leaning Algorithm
- **Dataset** : labeledTrainData.tsv
  : testData.tsv
  : unlabeledTrainData.tsv

Importing Packages and Data

Project : Bag of words Meets Bags of Popcorn

Importing Packages and Data

In [27]:
```
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import datetime as dt
import time
import seaborn as sns
import os
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
%matplotlib inline
```

In [28]:
```
your_local_path="Mcintosh HD/Users/rk/Desktop/UPXTECH/PROJECT/NLP/Bag of world"
```

In [29]:
```
cd /Users/rk/Desktop/UPXTECH/PROJECT/NLP/Bag of world
```

```
/Users/rk/Desktop/UPXTECH/PROJECT/NLP/Bag of world
```

Importing packages and Data

Train Data

```
In [30]: train = pd.read_csv("labeledTrainData.tsv", delimiter='\t')
```

```
In [31]: train.head()
```

Out[31]:

	id	sentiment	review
0	5814_8	1	With all this stuff going down at the moment w...
1	2381_9	1	\The Classic War of the Worlds\" by Timothy Hi...
2	7759_3	0	The film starts with a manager (Nicholas Bell)...
3	3630_4	0	It must be assumed that those who praised this...
4	9495_8	1	Superbly trashy and wondrously unpretentious 8...

Importing packages and Data

Test data

In [32]:
```
test = pd.read_csv("testData.tsv", delimiter='\t')
```

In [33]:
```
test.head()
```

Out[33]:

	id	review
0	12311_10	Naturally in a film who's main themes are of m...
1	8348_2	This movie is a disaster within a disaster fil...
2	5828_4	All in all, this is a movie for kids. We saw i...
3	7186_2	Afraid of the Dark left me with the impression...
4	12128_7	A very accurate depiction of small time mob li...

In [34]:
```
train.shape
```

Out[34]:
```
(25000, 3)
```

Data Cleaning and Text Preprocessing

In [35]:
```
from bs4 import BeautifulSoup
import re      # to remove Punctuation and numbers
```

In [36]:
```
from nltk.corpus import stopwords
stopset = set(stopwords.words('english'))
```

Import the stop word list

In [13]:
```
from nltk.corpus import stopwords
print
stopwords.words("english")
```

```
 'ma',
 'mightn',
 "mightn't",
 'mustn',
 "mustn't",
 'needn',
 "needn't",
 'shan',
 "shan't",
 'shouldn',
 "shouldn't",
 'wasn',
 "wasn't",
 'weren',
 "weren't",
 'won',
 "won't",
 'wouldn',
 "wouldn't"]
```

In [37]:
```
def review_to_words( raw_review ):
    # 1. Remove HTML
    review_text = BeautifulSoup(raw_review).get_text()

    # 2. Remove non-letters
    letters_only = re.sub("[^a-zA-Z]", " ", review_text)

    # 3. Convert to lower case, split into individual words
    words = letters_only.lower().split()

    # 4. In Python, searching a set is much faster than searching
    #   a list, so convert the stop words to a set
    stops = set(stopwords.words("english"))

    # 5. Remove stop words
    meaningful_words = [w for w in words if not w in stops]

    # 6. Join the words back into one string separated by space,
    # and return the result.
    return( " ".join( meaningful_words ))
```

In [ ]:
```
num_reviews = train["review"].size
clean_train_reviews = []
for i in range( 0, num_reviews ):

    clean_train_reviews.append( review_to_words( train["review"][i] ) )
```

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

vectorizer = CountVectorizer(analyzer = "word",
                             tokenizer = None,
                             preprocessor = None,
                             stop_words = None,
                             max_features = 5000)
train_data_features = vectorizer.fit_transform(clean_train_reviews)
train_data_features = train_data_features.toarray()
```

In [48]:
```
vocab = vectorizer.get_feature_names()
print(vocab)
```

```
'model', 'models', 'modern', 'modesty', 'molly', 'mom', 'moment', 'moments', 'mon', 'money', 'monk', 'monkey', 'monke
ys', 'monster', 'monsters', 'montage', 'montana', 'month', 'months', 'mood', 'moody', 'moon', 'moore', 'moral', 'mora
lity', 'morgan', 'morning', 'moronic', 'morris', 'mostly', 'mother', 'motion', 'motivation', 'motivations', 'motive
s', 'mountain', 'mountains', 'mouse', 'mouth', 'move', 'moved', 'movement', 'movements', 'moves', 'movie', 'movies',
'moving', 'mr', 'mrs', 'ms', 'mst', 'mtv', 'much', 'multi', 'multiple', 'mummy', 'mundane', 'murder', 'murdered', 'mu
rderer', 'murderous', 'murders', 'murphy', 'murray', 'museum', 'music', 'musical', 'musicals', 'muslim', 'must', 'mye
rs', 'mysteries', 'mysterious', 'mystery', 'nail', 'naive', 'naked', 'name', 'named', 'namely', 'names', 'nancy', 'na
rration', 'narrative', 'narrator', 'nasty', 'nathan', 'nation', 'national', 'native', 'natural', 'naturally', 'natur
e', 'navy', 'nazi', 'nazis', 'nd', 'near', 'nearby', 'nearly', 'neat', 'necessarily', 'necessary', 'neck', 'ned', 'ne
ed', 'needed', 'needless', 'needs', 'negative', 'neighbor', 'neighborhood', 'neighbors', 'neil', 'neither', 'nelson',
'neo', 'nephew', 'nerd', 'nervous', 'network', 'never', 'nevertheless', 'new', 'newly', 'newman', 'news', 'newspape
r', 'next', 'nice', 'nicely', 'nicholas', 'nicholson', 'nick', 'nicole', 'night', 'nightmare', 'nightmares', 'night
s', 'nine', 'ninja', 'niro', 'noble', 'nobody', 'noir', 'noise', 'nominated', 'nomination', 'non', 'none', 'nonethele
ss', 'nonsense', 'nonsensical', 'normal', 'normally', 'norman', 'north', 'nose', 'nostalgia', 'nostalgic', 'notable',
'notably', 'notch', 'note', 'noted', 'notes', 'nothing', 'notice', 'noticed', 'notion', 'notorious', 'novak', 'nove
l', 'novels', 'nowadays', 'nowhere', 'nuclear', 'nude', 'nudity', 'number', 'numbers', 'numerous', 'nurse', 'nuts',
'nyc', 'object', 'objective', 'obnoxious', 'obscure', 'obsessed', 'obsession', 'obvious', 'obviously', 'occasion', 'o
ccasional', 'occasionally', 'occur', 'occurred', 'occurs', 'ocean', 'odd', 'oddly', 'odds', 'offended', 'offensive',
'offer', 'offered', 'offering', 'offers', 'office', 'officer', 'officers', 'official', 'often', 'oh', 'oil', 'ok', 'o
kay', 'old', 'older', 'oliver', 'olivier', 'ollie', 'omen', 'one', 'ones', 'online', 'onto', 'open', 'opened', 'openi
```

Training the Random forest

In [83]:
```
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(n_estimators = 500)
forest = forest.fit( train_data_features, train["sentiment"] )
```

In [51]:
```
# Create an empty list and append the clean reviews one by one
num_reviews = len(test["review"])
clean_test_reviews = []
```

In [53]:
```
print ("Cleaning and parsing the test set movie reviews...\n")
for i in range(0,num_reviews):
    if( (i+1) % 1000 == 0 ):
        print ("Review %d of %d\n" % (i+1, num_reviews))
    clean_review = review_to_words( test["review"][i] )
    clean_test_reviews.append( clean_review )
```

```
Review 1000 of 25000

Review 2000 of 25000

Review 3000 of 25000

Review 4000 of 25000

Review 5000 of 25000

Review 6000 of 25000

Review 7000 of 25000

Review 8000 of 25000
```

Get a bag of words for the test set, and convert to a numpy array

In [69]:
```
test_data_features = vectorizer.transform(clean_test_reviews)
test_data_features = test_data_features.toarray()
```

In [84]:
```
result = forest.predict(test_data_features)
print (result)
```

```
[1 0 1 ..., 0 1 1]
```

In [67]:
```
test.shape
```

Out[67]:
```
(25000, 2)
```

In [85]:
```
output = pd.DataFrame( data={"id":test["id"], "sentiment":result} )
print (output)
```

```
             id  sentiment
0      12311_10          1
1       8348_2           0
2       5828_4           1
3       7186_2           1
4      12128_7           1
5       2913_8           0
6       4396_1           0
7        395_2           1
8      10616_1           0
9       9074_9           1
10      9252_3           0
11      9896_9           1
12       574_4           1
13     11182_8           1
14     11656_4           0
15      2322_4           1
16      8703_1           0
17      7483_1           0
18     6007_10           1
19     12424_4           0
20      4672_1           0
21     10841_3           0
22      8954_7           1
23      7392_1           0
24     10288_8           1
25      5343_4           0
26      4950_1           0
27      9257_4           0
28      8689_3           0
29      4480_2           0
...        ...         ...
```

```
...          ...          ...
24970    6857_10            1
24971   11091_8             1
24972    4167_2             1
24973     679_4             0
24974   10147_1             0
24975    6875_1             0
24976    923_10             1
24977    6200_8             0
24978    7208_8             1
24979    5363_8             1
24980    4067_8             0
24981    1773_7             1
24982   1498_10             1
24983  10497_10             1
24984   3444_10             1
24985     588_2             0
24986    9678_9             1
24987    1983_9             0
24988    5012_3             0
24989   12240_2             1
24990    5071_2             0
24991    5078_2             0
24992   10069_3             1
24993    7407_8             1
24994    7207_1             0
24995   2155_10             1
24996     59_10             1
24997    2531_1             0
24998    7772_8             1
24999  11465_10             1

[25000 rows x 2 columns]
```

In [ ]:

Thank you