In [2]:
```
import pandas as pd
import string
from collections import defaultdict
import gzip
from sklearn import linear_model
import numpy as np
import math
```

In [3]:
```
movies = pd.read_csv(r'C:\Users\Travi\Documents\GitHub\CSE158_HWs\AllMov
```

```
C:\Users\Travi\Anaconda3\lib\site-packages\IPython\core\interactiveshell.p
y:3058: DtypeWarning: Columns (7) have mixed types.Specify dtype option on
import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [4]:
```
len(movies)
```

Out[4]: 329044

In [5]:
```
#dropping any empty overviews or genres
movies = movies.dropna(subset=['overview', 'genres'])
movies = movies[movies['overview'] != 'No overview found']

mult_genres_indices = movies[movies['genres'].str.find('|') != -1].index
movies = movies.drop(mult_genres_indices)

```

In [ ]:
```

```

In [6]:
```
movies['genres'].value_counts().index.values
```

Out[6]:
```
array(['Drama', 'Documentary', 'Comedy', 'Music', 'Animation', 'Horror',
       'Action', 'Western', 'Thriller', 'Family', 'Romance', 'Crime',
       'Science Fiction', 'Adventure', 'Mystery', 'Fantasy', 'War',
       'History', 'TV Movie', 'Foreign'], dtype=object)
```

In [7]:
```
#creating dictionary of the values in the genre column to have a corresp
genreID = dict(zip(list(movies['genres'].value_counts().index.values),ra
```

In [8]:
```
1 genreID
```

Out[8]:
```
{'Drama': 0,
 'Documentary': 1,
 'Comedy': 2,
 'Music': 3,
 'Animation': 4,
 'Horror': 5,
 'Action': 6,
 'Western': 7,
 'Thriller': 8,
 'Family': 9,
 'Romance': 10,
 'Crime': 11,
 'Science Fiction': 12,
 'Adventure': 13,
 'Mystery': 14,
 'Fantasy': 15,
 'War': 16,
 'History': 17,
 'TV Movie': 18,
 'Foreign': 19}
```

In [9]:
```
1 #Replacing the genres so we can
2 movies['genres'] = movies['genres'].replace(genreID)
```

In [10]:
```
1 #Only taking columns we need
2 data = movies[['id','genres','overview']]
```

In [11]:
```
1 data['genres'].value_counts()
```

Out[11]:
```
0     23869
1     21417
2     15734
3      6334
4      5463
5      5329
6      2560
7      2369
8      2174
9      1515
10     1081
11      980
12      923
13      696
14      467
15      418
16      280
17      209
18      138
19      121
Name: genres, dtype: int64
```

In [12]:
```
1  punct = string.punctuation
2  bigram = defaultdict(int)
3  unigram = defaultdict(int)
4  totalbigrams = 0
5  total_unigram = 0
6
7  def counts(d):
8      t = str(d['overview'])
9
10
11     t = t.lower() # lowercase string
12
13     t = [c for c in t if not (c in punct)] # non-punct characters
14     t = ''.join(t) # convert back to string
15     words = t.strip().split() # tokenizes
16
17     for i in range(len(words)):
18         unigram[str(words[i])] += 1 #unigram count
19
20         if (i+1) == len(words):
21             break
22         w1 = words[i]
23         w2 = words[i + 1]
24         bigram[w1 + ', ' + w2] += 1 #bigram count
25
```

In [13]:
```
1 data.apply(counts, axis =1)
```

Out[13]:
```
4         None
9         None
16        None
20        None
26        None
          ...
329033    None
329034    None
329035    None
329037    None
329043    None
Length: 92077, dtype: object
```

In [14]:
```
1 unigram
```

Out[14]:
```
defaultdict(int,
            {'timo': 18,
             'novotny': 7,
             'labels': 31,
             'his': 56209,
             'new': 9549,
             'project': 779,
             'an': 27739,
             'experimental': 422,
             'music': 3481,
             'documentary': 7318,
             'film': 16983,
             'in': 97200,
             'a': 170329,
             'remix': 37,
             'of': 146108,
             'the': 285280,
             'celebrated': 262,
             'megacities': 7,
```

In [15]:
```
1 bigram
```

Out[15]:
```
defaultdict(int,
            {'timo, novotny': 1,
             'novotny, labels': 1,
             'labels, his': 1,
             'his, new': 482,
             'new, project': 13,
             'project, an': 7,
             'an, experimental': 125,
             'experimental, music': 4,
             'music, documentary': 7,
             'documentary, film': 495,
             'film, in': 279,
             'in, a': 10892,
             'a, remix': 7,
             'remix, of': 9,
             'of, the': 33303,
             'the, celebrated': 33,
             'celebrated, film': 2,
             'film, megacities': 1,
```

In [ ]:
```
1
```

In [16]:
```
1 #Taking only the top 1000 unigrams and bigrams
2 bigrams_1000 = sorted(bigram.items(), key=lambda x: x[1], reverse=True)[
3 unigram_1000 = sorted(unigram.items(), key=lambda x: x[1], reverse=True)
```

In [17]:
```
1 bigram_words = []
2 unigram_words = []
3 for i in bigrams_1000:
4     bigram_words.append(i[0])
5 for i in unigram_1000:
6     unigram_words.append(i[0])
```

In [18]:
```
1 len(bigram_words)
```

Out[18]:
```
1000
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

Bigram Count

In [19]:
```
wordId_bigram = dict(zip(bigram_words, range(len(bigrams_1000))))

def feature(datum):
    datum = datum[1]
    feat = [0]*len(bigrams_1000)
    r = ''.join([c for c in str(datum['overview']).lower() if not c in p
    text = r.split()
    for w in range(len(text)):
        if (w+1) == len(text):
            continue
        w1 = text[w]
        w2 = text[w + 1]
        word = str(w1) + ', ' + str(w2)
        if word in bigram_words:
            feat[wordId_bigram[word]] += 1
    feat.append(1) #offset
    return feat

```

In [20]:
```
train_movies = movies.sample(n = 50000)
vaildation_movies =  movies.sample(n = 50000)
test_movies = movies.sample(n = 50000)
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```

In [21]:
```
X = [feature(d) for d in train_movies.iterrows()]
y = [d[1]['genres'] for d in train_movies.iterrows()]
```

In [ ]:
```

```

In [22]:
```
clf = linear_model.LogisticRegression(C=1, class_weight='balanced') # MS
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
```

```
C:\Users\Travi\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Sp
ecify a solver to silence this warning.
  FutureWarning)
C:\Users\Travi\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:469: FutureWarning: Default multi_class will be changed to 'auto' in 0.2
2. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

In [ ]:
```

```

In [23]:
```
predictions
```

Out[23]:
```
array([8, 1, 6, ..., 6, 4, 4])
```

In [24]:
```
correctPrediction = np.array(y) == np.array(predictions)
sum(correctPrediction) / len(correctPrediction)
```

Out[24]:
```
0.38218
```

In [ ]:
```

```

Unigram Count

In [25]:
```
wordId_unigram = dict(zip(unigram_words, range(len(unigram_1000))))

def feature(datum):
    datum = datum[1]
    feat = [0]*len(bigrams_1000)
    r = ''.join([c for c in str(datum['overview']).lower() if not c in p
    text = r.split()
    for w in range(len(text)):
        word = text[w]
        if word in unigram_words:
            feat[wordId_unigram[word]] += 1
    feat.append(1) #offset
    return feat
```

In [26]:
```
X = [feature(d) for d in train_movies.iterrows()]
y = [d[1]['genres'] for d in train_movies.iterrows()]
```

In [27]:
```
clf = linear_model.LogisticRegression(C=1, class_weight='balanced') # MS
clf.fit(X, y)
theta = clf.coef_
predictions = clf.predict(X)
```

```
C:\Users\Travi\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Sp
ecify a solver to silence this warning.
  FutureWarning)
C:\Users\Travi\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.p
y:469: FutureWarning: Default multi_class will be changed to 'auto' in 0.2
2. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

In [28]:
```
correctPrediction = np.array(y) == np.array(predictions)
sum(correctPrediction) / len(correctPrediction)
```

Out[28]: 0.55972

Unigram Count Accuracy = .55

Bigram Count Accuracy = .38

In [ ]:
```

```

In [29]:
```
vaildation = pd.DataFrame(columns = ["TF-IDF", 'Unigram', 'Bigram', 'Acc
```

In [30]:
```
dicta = dict(zip(["TF-IDF", 'Unigram', 'Bigram', 'Accuracy', 'n_estimatc
```

In [31]:
```
dicta
```

Out[31]:
```
{'TF-IDF': [], 'Unigram': [], 'Bigram': [], 'Accuracy': [], 'n_estimators':
[]}
```

Unigram TF-IDF

In [33]:
```
import nltk
from nltk.corpus import stopwords
nltk.download("stopwords")
from nltk.tokenize import word_tokenize
```

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

In [34]:
```
stop_words = set(stopwords.words('english'))
```

In [35]:
```
wordId = dict(zip(unigram_words, range(len(unigram_words))))



def feature(datum):
    datum = datum[1]
    feat = [0]*len(unigram_words)
    r = ''.join([c for c in datum['overview'].lower() if not c in punct]
    text = r.split()
    unigram_wordId = dict(zip(unigram_words, [0]*1000))
    text = [w for w in text if not (w in stop_words)]
    for w in text:
        if w in unigram_words:
            unigram_wordId[w] += 1

    for i in unigram_wordId:
        if len(text) == 0 or unigram[i] == 0:
            feat[wordId[i]] = 0
            continue

        tf = unigram_wordId[i]
        idf = math.log(len(train_movies)/unigram[i], 10)
        feat[wordId[i]] = tf/idf
    feat.append(1) #offset
    return feat



```

In [36]:
```
X = [feature(d) for d in train_movies.iterrows()]
y = [d[1]['genres'] for d in train_movies.iterrows()]
```

In [ ]:
```

```

```
Testing new classifiers
```

In [37]:
```
# clf = linear_model.LogisticRegression(C=1, class_weight='balanced') #
# clf.fit(X, y)
# theta = clf.coef_
# predictions = clf.predict(X)
```

In [38]:
```
# correctPrediction = np.array(y) == np.array(predictions)
# sum(correctPrediction) / len(correctPrediction)
```

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [40]:
```
# clf = RandomForestClassifier()
# clf.fit(X, y)
# predictions = clf.predict(X)
```

In [41]:
```
# correctPrediction = np.array(y) == np.array(predictions)
# sum(correctPrediction) / len(correctPrediction)
```

In [42]:
```
X_vaild= [feature(d) for d in vaildation_movies.iterrows()]
y_vaild = [d[1]['genres'] for d in vaildation_movies.iterrows()]
```

```
RandomForest best perfoming classifier
```

In [43]:
```
for i in range(10, 100, 20):
    clf = RandomForestClassifier(n_estimators = i)
    clf.fit(X, y)
    predictions = clf.predict(X_vaild)
    correctPrediction = np.array(y_vaild) == np.array(predictions)
    dicta['TF-IDF'].append('Yes')
    dicta['Unigram'].append('Yes')
    dicta['Bigram'].append('No')
    dicta['Accuracy'].append(sum(correctPrediction) / len(correctPredict
    dicta['n_estimators'].append(i)
```

In [44]:
```
dicta
```

Out[44]:
```
{'TF-IDF': ['Yes', 'Yes', 'Yes', 'Yes', 'Yes'],
 'Unigram': ['Yes', 'Yes', 'Yes', 'Yes', 'Yes'],
 'Bigram': ['No', 'No', 'No', 'No', 'No'],
 'Accuracy': [0.75206, 0.77082, 0.77172, 0.77382, 0.77462],
 'n_estimators': [10, 30, 50, 70, 90]}
```

In [ ]:
```

```

In [ ]:
```

```

test set for bigram tf-idf

In [45]:
```
X = [feature(d) for d in test_movies.iterrows()]
y = [d[1]['genres'] for d in test_movies.iterrows()]
```

In [46]:
```
predictions = clf.predict(X)
```

In [47]:
```
correctPrediction = np.array(y) == np.array(predictions)
sum(correctPrediction) / len(correctPrediction)
```

Out[47]:
```
0.77366
```

In [ ]:
```

```

In [ ]:
```

```

bigram tf-idf

In [48]:
```
wordId = dict(zip(bigram_words, range(len(bigram_words))))



def feature(datum):
    datum = datum[1]
    feat = [0]*len(bigram_words)
    r = ''.join([c for c in datum['overview'].lower() if not c in punct]
    text = r.split()
    bigram_wordId = dict(zip(bigram_words, [0]*1000))
    text = [w for w in text if not (w in stop_words)]

    for w in range(len(text)):
        if (w+1) == len(text):
            continue
        w1 = text[w]
        w2 = text[w + 1]
        word = str(w1) + ', ' + str(w2)
        if word in bigram_words:
            bigram_wordId[word] += 1

    for i in bigram_words:
        if len(text) == 0 or bigram[i] == 0:
            feat[wordId[i]] = 0
            continue

        tf = bigram_wordId[i]
        idf = math.log(len(train_movies)/bigram[i], 10)
        feat[wordId[i]] = tf/idf
    feat.append(1) #offset
    return feat

```

In [ ]:
```

```

In [49]:
```
X = [feature(d) for d in train_movies.iterrows()]
y = [d[1]['genres'] for d in train_movies.iterrows()]
```

```
Testing new classifiers
```

In [50]:
```
# clf = linear_model.LogisticRegression(C=1, class_weight='balanced') #
# clf.fit(X, y)
# theta = clf.coef_
# predictions = clf.predict(X)
```

In [51]:
```
# correctPrediction = np.array(y) == np.array(predictions)
# sum(correctPrediction) / len(correctPrediction)
```

In [52]:
```
X_vaild= [feature(d) for d in vaildation_movies.iterrows()]
y_vaild = [d[1]['genres'] for d in vaildation_movies.iterrows()]
```

In [53]:
```
# from sklearn.ensemble import RandomForestClassifier
```

In [54]:
```
# clf = RandomForestClassifier()
# clf.fit(X, y)
# predictions = clf.predict(X)
```

In [55]:
```
# correctPrediction = np.array(y) == np.array(predictions)
# sum(correctPrediction) / len(correctPrediction)
```

In [ ]:
```

```

```
Again RandomFroest best perfoming classifer
```

In [56]:
```
for i in range(10, 100, 20):
    clf = RandomForestClassifier(n_estimators = i)
    clf.fit(X, y)
    predictions = clf.predict(X_vaild)
    correctPrediction = np.array(y_vaild) == np.array(predictions)
    dicta['TF-IDF'].append('Yes')
    dicta['Unigram'].append('No')
    dicta['Bigram'].append('Yes')
    dicta['Accuracy'].append(sum(correctPrediction) / len(correctPredict
    dicta['n_estimators'].append(i)
```

In [57]:
```
1 dicta
```

Out[57]:
```
{'TF-IDF': ['Yes',
  'Yes',
  'Yes',
  'Yes',
  'Yes',
  'Yes',
  'Yes',
  'Yes',
  'Yes',
  'Yes'],
 'Unigram': ['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'No', 'N
o'],
 'Bigram': ['No', 'No', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Ye
s'],
 'Accuracy': [0.75206,
  0.77082,
  0.77172,
  0.77382,
  0.77462,
  0.28576,
  0.28622,
  0.28622,
  0.28614,
  0.28626],
 'n_estimators': [10, 30, 50, 70, 90, 10, 30, 50, 70, 90]}
```

In [58]:
```
1 pd.DataFrame(dicta)
```

Out[58]:

	TF-IDF	Unigram	Bigram	Accuracy	n_estimators
0	Yes	Yes	No	0.75206	10
1	Yes	Yes	No	0.77082	30
2	Yes	Yes	No	0.77172	50
3	Yes	Yes	No	0.77382	70
4	Yes	Yes	No	0.77462	90
5	Yes	No	Yes	0.28576	10
6	Yes	No	Yes	0.28622	30
7	Yes	No	Yes	0.28622	50
8	Yes	No	Yes	0.28614	70
9	Yes	No	Yes	0.28626	90

test set for bigram tf-idf

In [ ]:
```
1
```

In [61]:
```
1 X = [feature(d) for d in test_movies.iterrows()]
2 y = [d[1]['genres'] for d in test_movies.iterrows()]
```

In [62]:
```
1 predictions = clf.predict(X)
```

In [63]:
```
1 correctPrediction = np.array(y) == np.array(predictions)
2 sum(correctPrediction) / len(correctPrediction)
```

Out[63]: 0.28752

In [ ]:
```
1
```

Unigram TF-IDF is by far the better predictor for this dataset

In [ ]:
```
1
```