

The goal of our project is to predict the film box office based on multiple factors like the language, country, IMDB score of the film.

To get the data, we use the open source package OMDb:  
<http://omdbapi.com/>


[OMDb API](#) [Usage](#) [Parameters](#) [Examples](#) [Change Log](#) [API Key](#) [Become a Patron](#) [Contact](#)

# OMDb API

The Open Movie Database

The OMDb API is a RESTful web service to obtain movie information, all content and images on the site are contributed and maintained by our users.

If you find this service useful, please consider making a [one-time donation](#) or [become a patron](#).



Poster API

The Poster API is only available to patrons.

Currently over 280,000 posters, updated daily with resolutions up to 2000x3000.

After getting the key on the website, we can easily extract film data by using function `omdb.get(name/ID)`, either the film name or the film's IMDB ID could both get the information about the film. For example, we give the key word 'Titanic', it will return all the info of this film, including directors, main actors, IMDB rating, etc.

```
In [2]: import omdb
import requests
omdb.set_default('apikey','48cb183')
```

```
In [3]: omdb.get(title='Titanic', fullplot=True, tomatoes=False)
```

```
Out[3]: Item({'actors': 'Leonardo DiCaprio, Kate Winslet, Billy Zane, Kathy Bates', 'awards': 'Won 11 Oscars. Another 110 wins & 75 nominations.', 'box_office': 'N/A', 'country': 'USA', 'dvd': '10 Sep 2012', 'director': 'James Cameron', 'genre': 'Drama, Romance', 'language': 'English, Swedish', 'metascore': '74', 'plot': '84 years later, a 100 year-old woman named Rose DeWitt Bukater tells the story to her granddaughter Lizzy Calvert, Brock Lovett, Lewis Bodine, Bobby Buell and Anatoly Mikailavich on the Keldysh about her life set in April 10th 1912, on a ship called Titanic when young Rose boards the departing ship with the upper-class passengers and her mother, Ruth DeWitt Bukater, and her fiancé, Caledon Hockley. Meanwhile, a drifter and artist named Jack Dawson and his best friend Fabrizio De Rossi win third-class tickets to the ship in a game. And she explains the whole story from departure until the death of Titanic on its first and last voyage April 15th, 1912 at 2:20 in the morning.', 'poster': 'https://images-na.ssl-images-amazon.com/images/M/MV5BMDdmZGU3NDQtY2E5My00ZTliLWlZOTU0MTY4ZGI1YjdiNjk3XkEyXkFqcGdeQXVyNTA4NzY1MzY0._V1_SX300.jpg', 'production': 'Paramount Pictures', 'rated': 'PG-13', 'released': '19 Dec 1997', 'response': 'True', 'runtime': '194 min', 'title': 'Titanic', 'type': 'movie', 'website': 'http://www.titanicmovie.com/', 'writer': 'James Cameron', 'year': '1997', 'imdb_id': 'tt0120338', 'imdb_rating': '7.8', 'imdb_votes': '869,216'})
```

To get all the film data, we use the film title list from the internet and write a loop to save all the film data in a json file.

```
In [6]: import json
file='filmdata.json'
with open(file,'wb') as fc:
    for i in yl:
        info=omdb.get(title=i, fullplot=True, tomatoes=True)
        jsobj=json.dumps(info)

        fc.write(jsobj.encode())

print('finished')
```

finished

Here we met a problem that the request may be terminated by the server of OMDB for unknown reason after a while, so we could only get about 1Mb size data before it was terminated. And each time we saved the existing data and tried again in a new file and get all the film data after 7 times. Then we transform the json file into csv file to make it easier to deal with.

```

import csv
import json
with open('film.json','r') as rd:
    line=json.load(rd)
data=line['film']
with open('movie_metadata.csv','w') as wt:
    csvwriter = csv.writer(wt)
    count = 0
    for emp in data:
        if count == 0:
            header = emp.keys()

            csvwriter.writerow(header)

            count += 1

        csvwriter.writerow(emp.values())

```

We list several factors and some of them need to be vectorized. Some work could be done in the excel like content rating of the film, some of them need to be done in python, like country, language and type of movie.

```

import pandas as pd
import numpy as np
with open('C:/Users/WANGYIFAN/Desktop/gai.csv') as f:
    df= pd.read_csv(f,header=0, delim_whitespace=False,na_values=0,index_col=False)
df=df.fillna(0)
df = df.sample(frac=1.0) # 全部打乱

df.head(6)

```

	Color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facet
936	1.0	Brad Bird	318.0	111.0	663.0	
54	1.0	Kevin Allen	55.0	100.0	8.0	
1447	1.0	Bobby Farrelly	109.0	114.0	101.0	
1707	1.0	Robert Iscove	60.0	90.0	7.0	
699	1.0	Kirk De Micco	257.0	98.0	16.0	
344	1.0	Ron Shelton	132.0	116.0	41.0	

6 rows × 28 columns

```
2]: #country
xcnew=[]
for item in xc:
    if item=='USA':
        item=0

    else:
        item=1
    xcnew.append(item)
xcnew=np.array(xcnew)
print(xcnew)

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

```
0]: #language
xl=np.array(df['language'])
xlnew=[]
for item in xl:
    if item=='English':
        item=1

    else:
        item=0
    xlnew.append(item)
xlnew=np.array(xlnew)
print(xlnew)

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

```
In [4]: #type of the movie
xg = np.array(df['genres'])
xaction = []
for item in xg:
    if 'Action' not in item:
        item = 0
    else:
        item = 1
    xaction.append(item)
xaction = np.array(xaction)
print(xaction)

xadventure = []
for item in xg:
    if 'Adventure' not in item:
        item = 0
    else:
        item = 1
    xadventure.append(item)
xadventure = np.array(xadventure)
print(xadventure)

xcomedy = []
for item in xg:
    if 'Comedy' not in item:
        item = 0
    else:
        item = 1
    xcomedy.append(item)
xcomedy = np.array(xcomedy)
print(xcomedy)
```

All the factors and how they are vectorized are listed below:

1. the color of the film: Black and White(0), Color(1)
2. type of the movies (We set 8 categories, Action, Adventure, Comedy, Family, Crime, Thriller(Horror, Mystery), Fantasy(Sci-Fi), Biography(History, Documentary), and use one-hot coding to solve this )
3. content rating of the film: Unrated(0), G(1), PG(2), PG-13/NC-17(3), R(4)
4. IMDB score(from 0 to 10)
5. year of releasing(taking the price of different years into consideration, we filtered and removed all the films before year 2000)
6. budget of the film
7. country: USA(0), other countries(1)
8. language: English(0), other languages(1)
9. facebook likes of the film
- 10.facebook likes of directors and main actors and the cast
- 11.aspect ratio
- 12.number of critics in reviews for the film
- 13.number of people who voted for the film

In total, we get 2719 valid film data and we randomly select 2000 of them to be training data and the rest of them be the test data. Because of the magnitude difference of different films, the accuracy of the

prediction is based on the allowance of 100% error, trying to make prediction on the magnitude of the box office.

After considering all the factors, we get 22 features of training and test data.

```
: #Xtrain=df[['Color', 'content_rating', 'budget', 'title_year', 'imdb_score']]
from sklearn.preprocessing import scale
X1=np.array(df[['Color', 'content_rating', 'title_year', 'imdb_score', 'cast_total_facebook_li
               'actor_1_facebook_likes', 'director_facebook_likes', 'num_critic_for_reviews

X=np.c_[X1, xcnew, xlnew, xaction, xadventure, xcomedy, xfamily, xcrime, xthriller, xfantasy, xbiogr
X = scale(X)

y=np.array(df['gross'])
#y=np.log10(y)

nt=2000

Xtr=X[:nt, :]
ytr=y[:nt]
Xts=X[nt:, :]
yts=y[nt:]
X.shape[1]
```

: 22

First, we use linear regression model to make prediction.

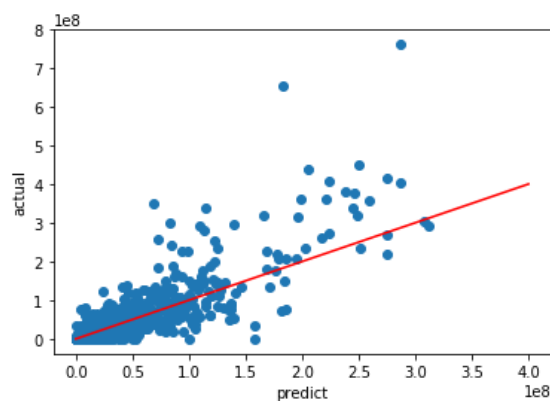
```
In [7]: from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(Xtr,ytr)
y_pred = regr.predict(Xts)

RSS_tr = np.mean((y_pred-yts)**2)/(np.std(yts)**2)
print("Normalized RSS = {0:f}".format(RSS_tr))
acc=np.mean(np.fabs((y_pred-yts)/yts<1))
print("Accuracy on test data = %f" % acc)
```

```
Normalized RSS = 0.395879
Accuracy on test data = 0.710938
```

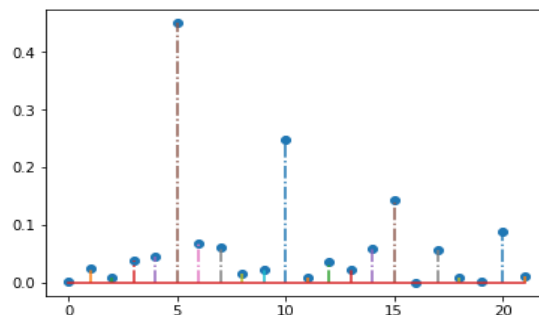
```
In [52]: #plt.plot(yts,y_pred,'o')
plt.scatter(y_pred,yts)
plt.xlabel('predict')
plt.ylabel('actual')
plt.plot([0,4e8],[0,4e8], 'r')
```

Out[52]: [<matplotlib.lines.Line2D at 0x2109bc6fac8>]



From the plot and  $RSS=0.39$  we can see that most data fit well, but when the box office get larger, the data become sparse.

```
In [9]: natts=X.shape[1]
ym = np.mean(ytr)
syy = np.mean((ytr-ym)**2)
Rsqr = np.zeros(natts)
beta0 = np.zeros(natts)
beta1 = np.zeros(natts)
for k in range(natts):
    xm = np.mean(Xtr[:,k])
    sxy = np.mean((Xtr[:,k]-xm)*(ytr-ym))
    sxx = np.mean((Xtr[:,k]-xm)**2)
    beta1[k] = sxy/sxx
    beta0[k] = ym - beta1[k]*xm
    Rsqr[k] = (sxy)**2/sxx/syy
plt.stem(Rsqr, '-.')
plt.show()
#print('0:2d' Rsqr=[1:f'%.format(k, Rsqr[k])])
```



```
In [44]: d = 16 # Number of neurons to use
r=np.argsort(-Rsqr)
Isel=r[:d]
print('The neurons with the ten highest R^2 values = ...',Isel[:10])
```

The neurons with the ten highest  $R^2$  values = ... [ 5 10 15 20 6 7 14 17 4 3]

```

In [45]: Xt=np.zeros((nt,d))
Xtss=np.zeros((719,d))
Xts.shape
for i in range(nt):
    for j in range(d):
        Xt[i][j]=Xtr[i][Isel[j]]
for i in range(719):
    for j in range(d):
        Xtss[i][j]=Xts[i][Isel[j]]
regr.fit(Xt,ytr)
y_pred = regr.predict(Xtss)
RSS_ts = np.mean((y_pred-yts)**2)/(np.std(yts)**2)
print("Normalized RSS = {0:f}".format(RSS_ts))
acc=np.mean(np.fabs((y_pred-yts)/yts<1))
print("Accuracy on test data = %f" % acc)

Normalized RSS = 0.398204
Accuracy on test data = 0.709473

```

We tried to rule out some irrelevant factors and pick the 16 most relevant factors out of the total 22 factors, but the RSS and accuracy did not change a lot. On the contrary, if we try to reduce the factors to even less, the accuracy would fall a lot.

Then, we used KNN to make the prediction based on the average of the values of its  $k$  nearest neighbors. Here, we found that set  $k=3$  could reach the best result. We got the  $RSS=0.38$  and  $0.698$  accuracy with KNN method, which is basically the same with linear regression. And after ruling out 6 irrelevant factors, we got the lowest RSS, which was  $0.36$ , and accuracy was  $0.70$ . Relatively speaking, KNN gave us a better result than linear regression.



```

: from sklearn.neighbors import KNeighborsRegressor
knn = KNeighborsRegressor(3)
knn.fit(Xtr, ytr)
y_pred = knn.predict(Xts)
RSS_ts = np.mean((y_pred-yts)**2)/(np.std(yts)**2)
print("Normalized RSS = {0:f}".format(RSS_ts))
acc=np.mean(np.fabs((y_pred-yts)/yts<1))
print("Accuracy on test data = %f" % acc)
plt.scatter(y_pred,yts)
plt.xlabel('predict')
plt.ylabel('actual')
plt.plot([0, 4e8], [0, 4e8], 'r')

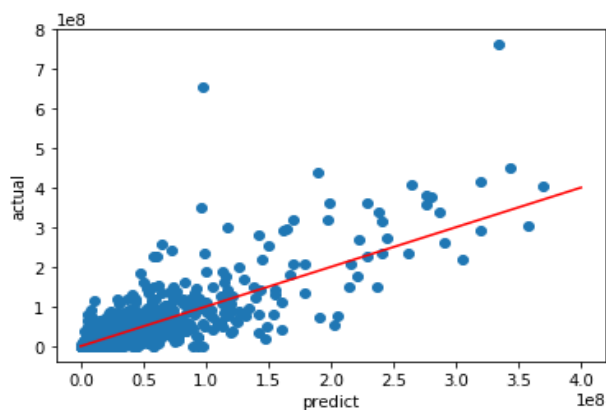
```

Normalized RSS = 0.382022  
Accuracy on test data = 0.698242

```

: [ <matplotlib.lines.Line2D at 0x2109c07b828>]

```



```

]: knn.fit(Xt,ytr)
y_pred =knn.predict(Xtss)
RSS_ts = np.mean((y_pred-yts)**2)/(np.std(yts)**2)
print("Normalized RSS = {0:f}".format(RSS_ts))
acc=np.mean(np.fabs((y_pred-yts)/yts<1))
print("Accuracy on test data = %f" % acc)

```

Normalized RSS = 0.360300  
Accuracy on test data = 0.701172

Finally, we tried to use neural network for the prediction. The result is not very good because of the great difference in value, and it seems the prediction and the test data vary in magnitude.

Accuracy on test data = 0.701174

```
In [72]: Xmean=np.mean(Xtr,axis=0)
Xstd=np.std(Xtr,axis=0)
Xtr_scale=(Xtr-Xmean[None,:])/Xstd[None,:]
Xts_scale=(Xts-Xmean[None,:])/Xstd[None,:]
```

```
In [73]: from keras.models import Model, Sequential
from keras.layers import Dense, Activation
import keras.backend as K
from keras import optimizers
K.clear_session()
nin = X.shape[1] # dimension of input data
nh = 80 # number of hidden units
nout = 1 # number of outputs = 10 since there are 10 classes
model = Sequential()
model.add(Dense(nh, input_shape=(nin,), activation='relu', name='hidden'))
model.add(Dense(nout, activation=None, name='output'))
opt = optimizers.Adam(lr=0.001) # beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0
model.compile(optimizer=opt,
              loss='mean_squared_error',
              metrics=['mean_squared_error'])
```

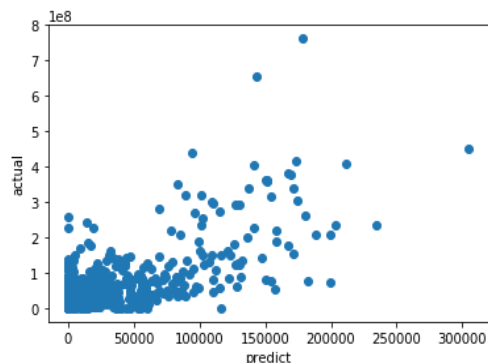
```
In [74]: model.fit(Xtr_scale, ytr, epochs=500, batch_size=100, validation_data=(Xts_scale,yts),verbose=False)
```

```
Out[74]: <keras.callbacks.History at 0x2109dcd1da0>
```

```
In [75]: y_pred=model.predict(Xts_scale)
plt.plot(y_pred,yts,'o')

plt.xlabel('predict')
plt.ylabel('actual')
```

```
Out[75]: Text(0,0.5,'actual')
```



To sum up, the KNN model could get the best prediction result, with  $RSS=0.36$ , which seems to be good. To be fair, the accuracy is not that good since we only get 70% accuracy with that wide range. The neural network method should be working well with proper setting, but due to the time limit we only managed to reach this level, and we will try to modify this method after the final.

There are several reasons for the low accuracy.

1. Limited film data. At the beginning, we got about 5500 pieces of film data, but the range of the releasing year could be about 100 years from 1910-2016, so considering the ticket price and purchasing power of people, we only used the films from 2000 to 2016 for training and testing, which only had about 2700 in total. Actually, we believe the real valid data should be after year 2010. However, due to the lack of data and many films in other countries with high productions like India and China could not be counted. So we had to expand our range to promise the number of data. That's to some extent lower the accuracy of predicting since the ticket price has doubled comparing to 2000.
2. The box office of different films vary greatly in magnitude. Some famous films, like Avatar, earned 700 million dollars in the USA, while an ordinary drama only had 10 thousand. In this situation, a 10000 dollars difference only makes 0.01% error for Avatar, but the error for the drama will be 100%. This often makes huge error during prediction. So, while our RSS is only 0.36, we achieve the accuracy above 70% when we allow 100% prediction error.
3. It is hard to find a factor that really matters. Based on our plot, we can see that the factor that with highest  $R^2$  is the total facebook likes of the cast. Which is surprising because some casts in China

may not even have a facebook. In fact, the factor we value most like IMDB score has little effect on the result. So it is difficult to find the most important factor and improve the accuracy.