Movie Revenue: Predicting How Much a Movie Will Make!

This dataset was found on kaggle competition to see if it is possible to predict the revenue of the movie, based a feature vector contain, movie id, title, budget, genres, crew, cast etc. However, for the sake of this project, the I will be using a smaller feature vector, to test which algorithm or tricks learned in class will work the best for this data.

First we will load up the data and start with some visualiztions to get a better idea of what our data looks like.

```
import pandas as pd
import os
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from sklearn import preprocessing
import tensorflow
from tensorflow.keras import models
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import io
from sklearn.model_selection import train_test_split
from google.colab import files
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn import tree, metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
from sklearn.preprocessing import StandardScaler
```

```
#Loading Data
def load_data():
    uploaded = files.upload()
    data = pd.read_csv(io.StringIO(uploaded["train_2.csv"].decode("ISO-8859-1")))

    y = data['revenue']

    data = data.drop('revenue',axis = 1)
    x = data.values

    numExamples = x.shape[0]
    allOnes = np.ones((numExamples,1))
    x = np.concatenate((x,allOnes),axis = 1)



    return(x,y)
```

```
x_train, y_train = load_data()
```

```
Choose Files  train_2.csv
```

- **train_2.csv**(application/vnd.ms-excel) - 142149 bytes, last modified: 5/14/2019 - 100% done

```
Saving train_2.csv to train_2 (5).csv
```

So it looks like we have 3000 training examples for seven different columns of data. The training and test was split by the person who provided the original data. Here I took 6 simple columns to try to predict the outcome, the title, the budget, the runtype, movie genre, numerical reprentation of the genres with the added regularization term of all ones.

```
x_train.shape
```

```
(3000, 6)
```

Checking the first row of the data to see what it looks like.

```
print(x_train[0])
```

```
['2:22' 22531334 99.0 'Drama' 1 1.0]
```

Within the data, there was a lot of zeros in the budget data, (812 entries were zeros). So instead of leaving them zeros, I replaced them with the average budget.

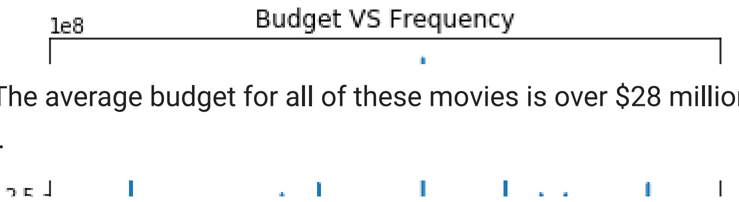
```
title = x_train[:,0]
budget = x_train[:,1]
runtime = x_train[:,2]
genres = x_train[:,3]
#New Genres is a converted array with numbers corresponding to the most popular genres for e
new_genres = x_train[:,4]
```

```
np.mean(budget)
```

```
28629815.179333333
```

```
plt.plot(budget)
plt.title("Budget VS Frequency")
```

```
Text(0.5, 1.0, 'Budget VS Frequency')
```



Wow! The average budget for all of these movies is over $28 million dollars! I wish I had that much money.

```
#Most popular genres
l = []
n = len(genres)
for i in range(n):
    if genres[i] not in l:
        l.append(genres[i])


from collections import Counter
most_genres = Counter(genres.flat).most_common(len(l))
most_genres
```

```
[('Drama', 787),
 ('Comedy', 606),
 ('Action', 521),
 ('Adventure', 188),
 ('Horror', 170),
 ('Crime', 148),
 ('Thriller', 116),
 ('Animation', 76),
 ('Documentary', 71),
 ('Fantasy', 68),
 ('Romance', 67),
 ('Science Fiction', 41),
 ('Family', 36),
 ('Mystery', 33),
 ('Music', 20),
 ('War', 20),
 ('History', 16),
 ('Western', 13),
 ('Foreign', 2),
 ('TV Movie', 1)]
```

Overwhelmingly the popular genre within the data is drama, followed behind comedy and action.

```
drama = np.where(x_train[:,3] == 'Drama')
comedy = np.where(x_train[:,3] == 'Comedy')
action = np.where(x_train[:,3] == 'Action')

drama = np.array(drama)
d = drama.shape[1]
dBudget = [d]
dBudget = x_train[drama,1]

comedy = np.array(comedy)
c = comedy.shape[1]
cBudget = [c]
cBudget = x_train[comedy,1]


action = np.array(action)
a = action.shape[1]
```

```
aBudget = [a]
aBudget = x_train[action,1]



print("Drama:: $" + str(np.mean(dBudget)))
print("Comedy:: $" + str(np.mean(cBudget)))
print("Action:: $" + str(np.mean(aBudget)))
```

```
Drama:: $21159820.776365947
Comedy:: $21515632.58910891
Action:: $39847202.690978885
```

Though Drama may be the leading genre, Action has a much higher budget for movies leading with over $39 miilion dollars! Will this have an effect on revenue?

Next, I will be turning the titles into ids to make it accessible and doing a one hot encoding of the genres to make easier to fit different algorithms and models to.

```
#Replacing the titles with id numbers.
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

ids = []
w, h = 2,x_train.shape[0];
#Creating a dictionary for each of the titles
ids_dic = [[0 for x in range(w)] for y in range(h)]

for i in range(n):
  ids.append(i)
  ids_dic[i][0] = i
  ids_dic[i][1] = title[i]


x_train[:,0] = ids
x_train[0:5]
```

```
array([[0, 22531334, 99.0, 'Drama', 1, 1.0],
       [1, 30000000, 79.0, 'Action', 3, 1.0],
       [2, 22531334, 97.0, 'Drama', 1, 1.0],
       [3, 35000000, 123.0, 'Drama', 1, 1.0],
       [4, 1500000, 102.0, 'Horror', 5, 1.0]], dtype=object)
```

```
#One Hot encoding genres
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder

S = new_genres
le = LabelEncoder()
S = le.fit_transform(S)
#print(S)
ohe = OneHotEncoder(categories='auto')
one_hot = ohe.fit_transform(S.reshape(-1,1)).toarray()
#print(one_hot[900:950])

#print(one_hot[0])
```

```
x = x_train
x = np.concatenate((x,one_hot), axis = 1)
x[0]
```

```
array([0, 22531334, 99.0, 'Drama', 1, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
       0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
       0.0], dtype=object)
```

```
#Creating training set containing only budget,all ones and categorical genre
x = np.delete(x,0,1)
x = np.delete(x,1,1)
x = np.delete(x,1,1)
x= np.delete(x,1,1)
x[0]
```

```
array([22531334, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
       0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], dtype=object)
```

To correctly perform Logistic Regression to estimate revenue, I will be modifying our target to 0 or 1s. 0 will indicate if they will not make a profit, and 1 if they do make a profit

```
n = x.shape[0]
logTarget = np.zeros((n,))
for i in range(n):
  if x[i,0] >= y_train[i]:
    logTarget[i] = 0
  else:
    logTarget[i] = 1
logTarget[0:20]
```

```
array([0., 1., 0., 1., 0., 1., 0., 1., 1., 0., 0., 1., 0., 1., 0., 1., 0.,
       1., 1., 1.])
```

COOL!

The data set given did not have actual data results for the revenue under the test data. To make sure of accuracy for test data, I split the given training data into train, and test with a 70/30 split.

```
.7 * x.shape[0]
```

```
2100.0
```

```
train = x[0:2100]
test = x[2100:]
targetTrain = y_train[0:2100]
targetTest = y_train[2100:]
logTargetTrain = logTarget[0:2100]
logTargetTest = logTarget[2100:]
print("Train shape: " + str(train.shape))
print("Test shape: " + str(test.shape))
print("Target Train shape: " + str(targetTrain.shape))
print("Target Test shape: " + str(targetTest.shape))
print("Log. Target Train shape: " + str(logTargetTrain.shape))
print("Log. Target Test shape: " + str(logTargetTest.shape))
```

```
Train shape: (2100, 22)
Test shape: (900, 22)
Target Train shape: (2100,)
Target Test shape: (900,)
Log. Target Train shape: (2100,)
Log. Target Test shape: (900,)
```

Linear Regression with Ridge and Lasso

```
#Linear Regression
regression = LinearRegression()
regression.fit(train, targetTrain)
train_score=regression.score(train, targetTrain)
test_score=regression.score(test, targetTest)

#Ridge Regression
trainR = Ridge(alpha=0.01) # higher the alpha value, more restriction on the coefficients; l
trainR.fit(train, targetTrain)
Ridge_train_score = trainR.score(train,targetTrain)
Ridge_test_score = trainR.score(test, targetTest)


#Lasso Regression
lasso = Lasso()
lasso.fit(train,targetTrain)
trainL_score =lasso.score(train, targetTrain)
testL_score =lasso.score(test,targetTest)


print ("linear regression train score:"+ str(train_score))
print ("linear regression test score:" + str(test_score))

print ("ridge regression train score:" + str(Ridge_train_score))
print ("ridge regression test score:"+ str(Ridge_test_score))

print ("lasso regression train score:" + str(trainL_score))
print ("lasso regression test score:"+ str(testL_score))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/ridge.py:125: LinAlgWarni
  overwrite_a=True).T
linear regression train score:0.5453805855206585
linear regression test score:0.5582337591912087
ridge regression train score:0.545380578901538
ridge regression test score:0.5582358047828411
lasso regression train score:0.5453805855203838
lasso regression test score:0.5582337718476873
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/coordinate_descent.py:492
  ConvergenceWarning)
```

Logistic Regression Model

To correctly perform Logistic Regression to estimate revenue, I will be modifying our target to 0 or 1s. 0 will indicate if they will not make a profit, and 1 if they do make a profit

```
from sklearn.linear_model import LogisticRegression
model1 = LogisticRegression()
model1.fit(train,logTargetTrain)
predictions = model1.predict(test)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWa
  FutureWarning)
```

```
print ("Logistic regression Train Accuracy :: "+ str(metrics.accuracy_score(logTargetTrain,
print ("Logistic regression Test Accuracy :: " + str(metrics.accuracy_score(logTargetTest, m
```

```
Logistic regression Train Accuracy :: 0.549047619047619
Logistic regression Test Accuracy :: 0.5555555555555556
```

Logistic Regression gives us about a .55 percent accuracy! While not a great percentage, considering we have a limited feature vector and not doing too much test regarding the features themselves, that is not too bad.

Multi-Class Logistic Regression

Because we were only predicting revenue, there is no multi-class logisitic regression. For a future project, I thought it would be fun to see if it is possible to predict the genres of a movie given budget and revenue. Since this doesn't fit in consistancy with the rest of the project, this will have to be a future project!

Convolutional Neural Network

```
#CNN shapes were off, needs more work
# network = models.Sequential()
# network.add(layers.Conv2D(32,(3,3), activation = 'relu', input_shape = (2100,22,1)))
# network.add(layers.MaxPooling2D((3,3)))
# network.add(layers.MaxPooling2D((2,2)))
# network.add(layers.Conv2D(64,(3,3), activation = 'relu'))

# network.add(layers.Flatten())
# network.add(layers.Dense(64, activation = 'relu'))
# network.add(layers.Dense(10, activation = 'softmax'))

# network.compile(optimizer = 'rmsprop', loss = 'categorical_crossentropy',\
#                  metrics = ['accuracy'])

# network.fit(train,logTargetTrain, epochs = 5, batch_size = 64)
```

Decision Trees

```python
dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
dtree.fit(train, logTargetTrain)
pred_train = dtree.predict(train)
# how did our model perform?
count_misclassified = (logTargetTrain != pred_train).sum()
print('Misclassified samples:: {}'.format(count_misclassified))
accuracy_train = metrics.accuracy_score(logTargetTrain, pred_train)
print('Accuracy:: {:.2f}'.format(accuracy_train))
```

```
Misclassified samples:: 550
Accuracy:: 0.74
```

```python
# use the model to make predictions with the test data
y_pred = dtree.predict(test)
# how did our model perform?
count_misclassified = (logTargetTest != y_pred).sum()
print('Misclassified samples:: {}'.format(count_misclassified))
accuracy_test = metrics.accuracy_score(logTargetTest, y_pred)
print('Accuracy:: {:.2f}'.format(accuracy_test))
```

```
Misclassified samples:: 241
Accuracy:: 0.73
```

Random Forests

```python
clf = RandomForestClassifier()
model = clf.fit(train, logTargetTrain)
predictions = model.predict(test)
# Train and Test Accuracy
print("Train Accuracy :: " + str(accuracy_score(logTargetTrain, model.predict(train))))
print("Test Accuracy  :: " + str(accuracy_score(logTargetTest, predictions)))
```

```
Train Accuracy :: 0.8252380952380952
Test Accuracy  :: 0.6644444444444444
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:246: FutureWarning:
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

K Means Clustering

```python
kmeans = KMeans(n_clusters=2, max_iter=600, algorithm = 'auto')
kmeans = kmeans.fit(train)
```

```python
correct = 0
for i in range(len(train)):
    predict_me = np.array(train[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == logTargetTrain[i]:
        correct += 1

print("Train Accuracy:: " + str(correct/len(train)))
```

```
correct_test = 0
for i in range(len(test)):
    predict_me = np.array(test[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == logTargetTest[i]:
        correct_test += 1

print("Test Accuracy:: " + str(correct_test/len(test)))
```

```
Train Accuracy:: 0.49952380952380954
Test Accuracy:: 0.53
```

Manipulating the different clusters and max_ters wihtin a kmeans cluster

```
kmeans1 = KMeans(n_clusters=10, max_iter=1000, algorithm = 'auto')
kmeans1 = kmeans.fit(train)


correct1 = 0
for i in range(len(train)):
    predict_me1 = np.array(train[i].astype(float))
    predict_me1 = predict_me1.reshape(-1, len(predict_me1))
    prediction1 = kmeans1.predict(predict_me1)
    if prediction1[0] == logTargetTrain[i]:
        correct1 += 1

print("Train Accuracy:: " + str(correct/len(train)))

correct1_test = 0
for i in range(len(test)):
    predict_me2 = np.array(test[i].astype(float))
    predict_me2 = predict_me2.reshape(-1, len(predict_me2))
    prediction2 = kmeans1.predict(predict_me2)
    if prediction2[0] == logTargetTest[i]:
        correct1_test += 1

print("Test Accuracy:: " + str(correct1_test/len(test)))
```

```
Train Accuracy:: 0.49952380952380954
Test Accuracy:: 0.53
```

Doesn't make an improvement.

```
#Would like to try scaling the data, but need more understanding of the StandardScaler befor

# scaler = StandardScaler()
# scaled_train = train
# scaled_test = test
# scaled_train = scaler.fit(scaled_train)
# scaled_test = scaler.fit(scaled_test)

# sc_kmeans = KMeans(n_clusters=2, max_iter=600, algorithm = 'auto')
# sc_kmeans = sc_kmeans.fit(scaled_train)

# sc_correct = 0
# for i in range(len(scaled_train)):
#     predict_me3 = np.array(scaled_train[i].astype(float))
#     predict_me3 = predict_me3.reshape(-1, len(predict_me3))
#     sc_prediction = sc_kmeans.predict(predict_me3)
#     if sc_prediction[0] == logTargetTrain[i]:
```

```
#         sc_correct += 1

# print("Scaled Train Accuracy:: " + str(sc_correct/len(scaled_train)))

# sc_correct_test = 0
# for i in range(len(test)):
#     predict_me4 = np.array(scaled_test[i].astype(float))
#     predict_me4 = predict_me4.reshape(-1, len(predict_me4))
#     sc_prediction_test = sc_kmeans.predict(predict_me4)
#     if sc_prediction_test[0] == logTargetTest[i]:
#         sc_correct_test += 1

# print("Scaled Test Accuracy:: " + str(sc_correct_test/len(scaled_test)))
```

Conclusion

```
print ("linear regression train score :: "+ str(train_score))
print ("linear regression test score :: " + str(test_score))

print ("ridge regression train score :: " + str(Ridge_train_score))
print ("ridge regression test score :: "+ str(Ridge_test_score))

print ("lasso regression train score :: " + str(trainL_score))
print ("lasso regression test score :: "+ str(testL_score))

print ("Logistic regression Train Accuracy :: "+ str(metrics.accuracy_score(logTargetTrain,
print ("Logistic regression Test Accuracy :: " + str(metrics.accuracy_score(logTargetTest, m

print('Decision Tree Train Accuracy :: {:.2f}'.format(accuracy_train))
print('Decision Tree Test Accuracy :: {:.2f}'.format(accuracy_test))

print("Random Forest Train Accuracy :: " + str(accuracy_score(logTargetTrain, model.predict(
print("Random Forest Test Accuracy  :: " + str(accuracy_score(logTargetTest, predictions)))

print("K Means Train Accuracy :: " + str(correct/len(train)))
print("K Means Test Accuracy :: " + str(correct_test/len(test)))
```

```
linear regression train score :: 0.5453805855206585
linear regression test score :: 0.5582337591912087
ridge regression train score :: 0.545380578901538
ridge regression test score :: 0.5582358047828411
lasso regression train score :: 0.5453805855203838
lasso regression test score :: 0.5582337718476873
Logistic regression Train Accuracy :: 0.549047619047619
Logistic regression Test Accuracy :: 0.5555555555555556
Decision Tree Train Accuracy :: 0.74
Decision Tree Test Accuracy :: 0.73
Random Forest Train Accuracy :: 0.8252380952380952
Random Forest Test Accuracy  :: 0.6644444444444444
K Means Train Accuracy :: 0.49952380952380954
K Means Test Accuracy :: 0.53
```

Overall, the clear winner in terms of Accuracy is the Decision Tree! With very little information got a 74% accuracy for the training data and 73% accuracy for the test data. The next would have to be the Random Forest, but it is concerning that the accuracy difference between training and test has a larger variance than the others, so it is not super consistant. Everything else ranged in the 50-55% accuracy range which again, with little data we had, I would say that these algorithms do a pretty good job of predicting if the movie will at least make the budget in revenue based on the budget and genre. There is always room for improvement however.

Future Considerations

There is a lot more I'd love to do with this data. I had to condense much of the data for sake of time, like excluding tag lines, cast, crew which would have required a lot more manipulations to make the data readable to these algorithms. With a lot more to the feature vecture, it would be interesting to do PCA to see how it would effect the outcome. I'd love to answer other questions like if it is possible to predict genres(not that that would be particularly useful but you never know) or if popularity or certain actors influences the overall revenue. This would be useful for producers to know if increasing the budget would do anything or if a cheaper cost in hiring actors is ideal. I tried it a little bit(unfortunately it didn't work) but doing more work with feature scaling to see how this would effect the data. Trying more types of algorthms and manipulating the input like number of clusters or criteron really make a difference to optimize it. 50% is pretty good but I'd love to do better! Another thing I would love to do is spend more time on visualizations. Those are my favorite part! Again, my timing got in the way so I was unable to do much of what I wanted, but is definitely something I want to improve on in the future.

My Thoughts

Being a Data Scientist is fun but a lot of work! I really appreciated this course and everything learned. If there was one thing I really learned about this project in particular is that what they say about cleaning the data takes about 90% of the time and the rest is being a data scientist. Knowing your data really well will go a long way. Using the shape function on an array is a life saver! By doing this project I also learned what I really need to work on in regards to my understanding, like Neural Networks. I understand the basic concepts but knowing the shapes and the best flow for a neural network was more than what I had time for within this project. However, I feel like I proved a lot to myself in my understanding of everything as well. I got nervous for this project because I wasn't sure if I would be able to accomplish everything and get the algorithms to work, yet I did! I am sure I made a few mistakes, could have cleaned up my code a lot more, and overall made it more appealing, but I am pretty proud of what I have done so far even though there is still a long way to go!