Predictions for Movie Success

Brooke Mosby and Michael Hansen

March 31, 2018

Abstract

Cinema has become one of the highest profiting industries over the past century. The total box office revenue in North America alone amounted to $11.38 billion in 2016. With the possibility of great success, there is also a large risk of financial failure. This data exploration is motivated by answering the question of what makes a movie successful. There is plenty of quantative data available for movies, such as the movies' budget, the release date, ratings etc., but in this analysis an attempt will be made to quantify movie information that is less measurable and then predict movie success.

```
import pandas as pd
import numpy as np
import networkx as nx
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.model_selection import train_test_split as tts
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn import linear_model
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from catboost import Pool, CatBoostRegressor
import re, json, requests, seaborn, warnings
warnings.filterwarnings( 'ignore' )
from bs4 import BeautifulSoup
from matplotlib import pyplot as plt, rcParams
%matplotlib inline
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import NoSuchElementException
from ipyparallel import Client
import time
```

1 Introduction

Research has been done to determine what aspects of a movie make it more successful; however, much of this research is contradictory. The research paper "Early Predictions of Movie Success:

the Who, What, and When of Profitability" states movies with a motion picture content rating 'R' will likely have lower profits, whereas the research paper "What Makes A Great Movie?" states a motion picture content rating 'R' will have higher a box-office. Both papers analyzed thousands of movies, but came to opposite conclusions. Some variables used to predict movie success in these studies, included budget, motion picture content rating, and actor popularity.

Based on these previous models, the dataset used will include movie title length, run-time, motion picture content rating, director, genre, release date, actors, an actor popularity score, average salary of the actors in the movie, budget, and opening weekend box-office revenue for predictor variables. The actor popularity score will be calculated from a network of actors connected through the movies they appeared in together and from the average actor income. Movie success will be determined by the awards it is nominated for, the awards it won, the MetaCritic score, the IMDb rating, and the profit of the movie.

2 Data Scraped, Downloaded, Cleaned & Engineered

2.1 Beginning Dataset

A beginning dataset is downloaded from IMDb with 10,000 movies, each entry containing the movie title, URL on IMDb rating, run-time, Year, Genres, Num Votes, Release Date, Directors. From this dataset, additional information on the movie budget, gross income, opening weekend box revenue, actors, Oscar nominations, Oscars won, other award nominations, other awards won, MetaCritic score, and content rating is scraped and cleaned. The data points will be collected from IMDb, which is a reputable source for information, according to their website,

> "we [IMDb] actively gather information from and verify items with studios and filmmakers".

2.2 Cleaning Data

After gathering each data point, the data set is complete, although the information is not clean or uniform. The first step to clean the data will be to remove all commas across each column in the DataFrame. Removing commas will make it easier to convert monetary amounts to ints. Next each date in the Release column will be changed to a pandas date object, which will simplify any calculations that rely on the release date of the movie. Each monetary amount will be converted into an int and converted into USD. Each unique genre will be made into a column, with a true or false boolean for each movie entry.

2.3 Feature Engineering

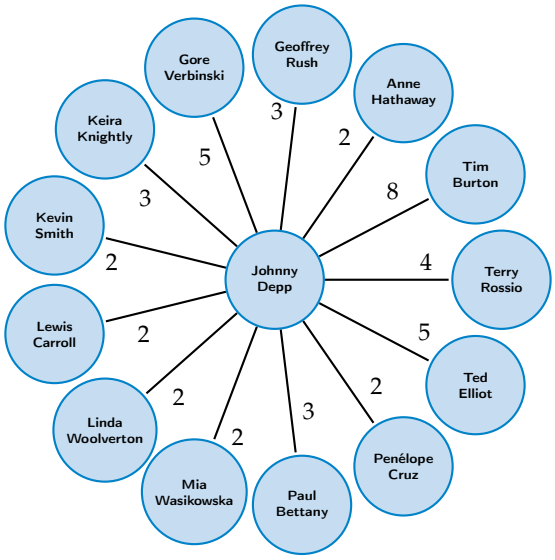
To resolve the disagreement in monetary amounts, due to inflation, a dataset containing the CPI for each year from 1914 will used to adjust the monetary amounts. The CPI, Consumer Price Index, describes the amount of purchasing power the average consumer has. The length of the movie title will be added, and a NetworkX graph of all actors will be made. This network will connect nodes of actors to each other, if they appear in a movie together. The edges of the network will be weighted by the amount of movies the actors appear in together. An actor popularity score

will be calculated from the actors appearing in the movie, based on how many other movies they appear in with other actors and the actor's income.

The total variables in the new dataset are movie title, title length, motion picture content rating, run-time, IMDb rating, genres, MetaCritic score, Oscar nominations, Oscar wins, other award nominations, other award wins, director, release date, budget, opening weekend, gross, profit, budget adjusted for inflation, opening weekend adjusted for inflation, gross adjusted for inflation, profit adjusted for inflation, the top ten actors in the movie, and actor popularity score. A separate network will hold the actor nodes and their connections.

2.4 Actor Network

A network of actors is made to help determine the success of a movie. Each node in the network is an actor that has appeared in a movie, and that actor node will have an edge to another actor node if they appeared in a movie together. The weights on each edge will correspond to the amount of movies the actors, that the edge connects, have appeared in together. In the example below, edges with Johnny Depp having weight 1 were ignored. The other nodes and edges demonstrate how the graph was assembled, actor to actor with the edge weight being the number of movies they appear in together according to the dataset used.




```
df = pd.read_csv("Result_Data/total_engineered.csv",encoding = "ISO-8859-1")
del df["Unnamed: 0"]
df = df.fillna(df.mean())
print(", ".join(list(df.columns)))
```

```
Actor_0, Actor_1, Actor_2, Actor_3, Actor_4, Actor_5, Actor_6, Actor_7, Actor_8,
Actor_9, Budget, Directors, Gross, IMDb Rating, Meta Score, Num Votes, Opening Weekend,
```

Oscar Nominations, Oscar Wins, Other Nominations, Other Wins, Release Date, Runtime (mins), Title, Year, Genre: Short, Genre: Comedy, Genre: Fantasy, Genre: Film-Noir, Genre: War, Genre: Musical, Genre: Sport, Genre: Biography, Genre: Action, Genre: Fantasy, Genre: Animation, Genre: Biography, Genre: Mystery, Genre: Musical, Genre: Romance, Genre: Thriller, Genre: Film-Noir, Genre: History, Genre: Western, Genre: Drama, Genre: Sci-Fi, Genre: Horror, Genre: Romance, Genre: Adventure, Genre: Family, Genre: Sci-Fi, Genre: Animation, Genre: Music, Genre: Music, Genre: History, Genre: Mystery, Genre: Thriller, Genre: Comedy, Genre: Crime, Genre: Horror, Genre: Drama, Genre: War, Genre: Western, Genre: Adventure, Genre: Family, Genre: Action, Genre: Crime, Content Rating: PASSED, Content Rating: TV-MA, Content Rating: X, Content Rating: NC-17, Content Rating: TV-14, Content Rating: M, Content Rating: GP, Content Rating: TV-PG, Content Rating: PG, Content Rating: PG-13, Content Rating: G, Content Rating: NR, Content Rating: APPROVED, Content Rating: UNRATED, Content Rating: M/PG, Content Rating: TV-13, Content Rating: NOT RATED, Content Rating: TV-G, Content Rating: R, Decade, Profit, Budget_Adjusted, Gross_Adjusted, Open_Adjusted, Profit_Adjusted, Profit_Bool, Total Nominations, Total Wins, Length of Title, Directors Prev Number Movies, Directors Prev Total Profit, Directors Prev Mean Profit, Directors Prev Mean IMDb, Directors Prev Mean Meta, Directors Prev Mean Num Votes, Directors Prev Mean Nominations, Directors Prev Mean Wins, Actor Weights

```
df_x = df[['Actor_0', 'Actor_1', 'Actor_2', 'Actor_3', 'Actor_4','Actor_5',
           'Actor_6', 'Actor_7', 'Actor_8', 'Actor_9', 'Budget',
           'Directors', 'Release Date', 'Runtime (mins)', 'Title', 'Year',
           'Genre: Short', 'Genre:  Comedy', 'Genre: Fantasy',
           'Genre: Film-Noir', 'Genre: War', 'Genre: Musical',
           'Genre:  Sport', 'Genre: Biography', 'Genre: Action',
           'Genre:  Fantasy', 'Genre:  Animation', 'Genre:  Biography',
           'Genre: Mystery', 'Genre:  Musical', 'Genre:  Romance',
           'Genre: Thriller', 'Genre:  Film-Noir', 'Genre:  History',
           'Genre: Western', 'Genre: Drama', 'Genre: Sci-Fi',
           'Genre:  Horror', 'Genre: Romance', 'Genre: Adventure',
           'Genre:  Family', 'Genre:  Sci-Fi', 'Genre: Animation',
           'Genre:  Music', 'Genre: Music', 'Genre: History',
           'Genre:  Mystery', 'Genre:  Thriller', 'Genre: Comedy',
           'Genre:  Crime', 'Genre: Horror', 'Genre:  Drama',
           'Genre:  War', 'Genre:  Western', 'Genre:  Adventure',
           'Genre: Family', 'Genre:  Action', 'Genre: Crime',
           'Content Rating: PASSED', 'Content Rating: TV-MA',
           'Content Rating: X', 'Content Rating: NC-17',
           'Content Rating: TV-14', 'Content Rating: M',
           'Content Rating: GP', 'Content Rating: TV-PG',
           'Content Rating: PG', 'Content Rating: PG-13',
           'Content Rating: G', 'Content Rating: NR',
           'Content Rating: APPROVED', 'Content Rating: UNRATED',
           'Content Rating: M/PG', 'Content Rating: TV-13',
           'Content Rating: NOT RATED', 'Content Rating: TV-G',
           'Content Rating: R','Decade', 'Budget_Adjusted',
```

```
            'Length of Title', 'Directors Prev Number Movies',
            'Directors Prev Mean Profit', 'Directors Prev Mean IMDb',
            'Directors Prev Mean Meta', 'Directors Prev Mean Num Votes',
            'Directors Prev Mean Nominations', 'Directors Prev Mean Wins',
            'Actor Weights']]
df_y = df[['Gross', 'IMDb Rating', 'Meta Score', 'Num Votes',
            'Oscar Nominations', 'Oscar Wins', 'Other Nominations',
            'Other Wins','Profit','Gross_Adjusted',  'Profit_Adjusted',
            'Profit_Bool', 'Total Nominations', 'Total Wins']]
```

3 Methods

Various supervised machine learning models are used to predict certain characteristics of the movie that determine a movie's success. The variables that our models attempt to predict are the movie's IMDb rating, Metacritic Score, the number of votes it recieves on IMDb, the number of Oscar nominations, the number of Oscar wins, the number of other award nominations, the number of other award wins, the number of total award nominations, the number of total award wins, gross, the profit,the gross adjusted for inflation, the profit adjusted for inflation, and finally whether or not a movie turned a profit. These varaibles will hereon be referred to as the movies' independent variables. For each model the data is split into a training and testing set to determine how accurate each method proves.

3.1 PCA

Principal Component Analysis (PCA) is used to reduce the dimensionality of the large dataset while retaining important information. For each Machine Learning model, PCA is used to reduce dimensionality to 10, 20, and 50 components, and then compared to find the best dimensionality. Because our dataset contains over 100 columns, PCA is essential for our models to avoid extremely costly computations and produce more accurate results.

```
pca = PCA()
df_x_temp = df_x.select_dtypes(include=['float64','int',
            'bool']).astype('float')
df_y_temp = df_y.select_dtypes(include=['float64','int',
            'bool']).astype('float')
tr_x, tt_x, tr_y, tt_y = tts(df_x_temp, df_y_temp, test_size = .2)

def results(es_y1):
    """
    Accepts a regression prediction for variables and prints
    accuracy values of prediction, so it is easier to digest.
    Returns Nothing
    """
    print("\nMovie Gross Average Percent Error:")
```

```
print((abs(es_y1[0:,0]-np.array(tt_y.astype(float))[0:,0])
      /abs(np.array(tt_y.astype(float))[0:,0])).mean()*100,"%")
print("\nIMDb Rating Average Percent Error:")
print((abs(es_y1[0:,1]-np.array(tt_y.astype(float))[0:,1])
     /abs(np.array(tt_y.astype(float))[0:,1])).mean()*100,"%")
print("\nMeta Score Average Percent Error:")
print((abs(es_y1[0:,2]-np.array(tt_y.astype(float))[0:,2])
     /abs(np.array(tt_y.astype(float))[0:,2])).mean()*100,"%")
print("\nNumber of Votes Average Percent Error:")
print((abs(es_y1[0:,3]-np.array(tt_y.astype(float))[0:,3])
     /abs(np.array(tt_y.astype(float))[0:,3])).mean()*100,"%")
print("\nOscar Nominations Average Error:")
print(abs(es_y1[0:,4]-np.array(tt_y.astype(float))[0:,4]).mean())
print("\nOscar Wins Average Error:")
print(abs(es_y1[0:,5]-np.array(tt_y.astype(float))[0:,5]).mean())
print("\nOther Nominations Average Error:")
print(abs(es_y1[0:,6]-np.array(tt_y.astype(float))[0:,6]).mean())
print("\nOther Wins Average Error:")
print(abs(es_y1[0:,7]-np.array(tt_y.astype(float))[0:,7]).mean())
print("\nProfit Average Percent Error:")
print((abs(es_y1[0:,8]-np.array(tt_y.astype(float))[0:,8])
     /abs(np.array(tt_y.astype(float))[0:,8])).mean()*100,"%")
print("\nGross Adjusted Average Percent Error:")
print((abs(es_y1[0:,9]-np.array(tt_y.astype(float))[0:,9])
     /abs(np.array(tt_y.astype(float))[0:,9])).mean()*100,"%")
print("\nProfit Adjusted Average Percent Error:")
print((abs(es_y1[0:,10]-np.array(tt_y.astype(float))[0:,10])
     /abs(np.array(tt_y.astype(float))[0:,10])).mean()*100,"%")
print("\nProfit Bool Average Error:")
print(abs(es_y1[0:,11]-np.array(tt_y.astype(float))[0:,11]).mean())
print("\nProfit Bool Accuracy:")
es_y1_profit = (es_y1[0:,11]+.5).astype(int)
tt_y1_profit = np.array(tt_y.astype(float))[0:,11]
print(sum((es_y1_profit==tt_y1_profit).astype(int))/len(tt_y1_profit))
print("\nTotal Nominations Average Error:")
print(abs(es_y1[0:,12]-np.array(tt_y.astype(float))[0:,12]).mean())
print("\nTotal Wins Average Error:")
print(abs(es_y1[0:,13]-np.array(tt_y.astype(float))[0:,13]).mean())
```

3.2 Linear Regression Model

The simple supervised machine learning model, linear regression, attempts to predict the movies' independent variables. The model attempts to predict these varaibles with a linear approach for modelling the relationship between the scalar and categorical dependent variables and the explanatory variables, that we have previously determined. Because this is simple to implement

and does not contain costly computations, it is a great starting model for our data.

```
# Implementing linear regression
parameters = {'pca__n_components':[10,20,50]}
regr = linear_model.LinearRegression()
pipe = Pipeline(steps=[('pca', pca), ('regr', regr)])
estimator = GridSearchCV(pipe, parameters, n_jobs = -1, verbose = 1)
estimator.fit(tr_x,tr_y)
es_y1 = estimator.best_estimator_.predict(tt_x)
results(es_y1)
```

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits


[Parallel(n_jobs=-1)]: Done   4 out of   9 | elapsed:    0.7s remaining:    0.9s
[Parallel(n_jobs=-1)]: Done   9 out of   9 | elapsed:    1.8s finished
```

```
print(estimator.best_params_)
```

```
{'pca__n_components': 50}
```

3.2.1 Linear Regression Model Analysis

From the results of this model we can see there are very few things that linear regression can predict accurately. It is also important to note that the best estimator was PCA with 50 components, also the largest number of components used in the predictions. There are some variables that seem to have potential for a large increase in accuracy The first notable varaible is the IMDb rating, with has an error percentage of about 10%, or about a letter grade off on average. This has significant potential to be increased. The Oscar nominations and wins are predicted extremely accurately from linear regression alone, and the boolean for whether or not a movie returns a profit is correct more than 70% of the time, which is perhaps one of the most important variables movie producers will look for, and one that will hopefully have better accuracy with different models. Most other variables do extremely poorly when predicted with linear regression.

3.3 Ridge Regression Model

```
# Implementing Ridge regression
regr = linear_model.Ridge()
parameters = {'pca__n_components':[10,20,50]}
pca = PCA()
pipe = Pipeline(steps=[('pca', pca), ('regr', regr)])
```

```
estimator = GridSearchCV(pipe, parameters, n_jobs = -1, verbose = 1)
estimator.fit(tr_x,tr_y)
es_y2 = estimator.best_estimator_.predict(tt_x)
results(es_y2)
```

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
```

```
[Parallel(n_jobs=-1)]: Done   4 out of   9 | elapsed:    0.5s remaining:    0.6s
[Parallel(n_jobs=-1)]: Done   9 out of   9 | elapsed:    1.7s finished
```

```
print(estimator.best_params_)
```

```
{'pca__n_components': 50}
```

3.3.1 Ridge Regression Model Analysis

Ridge Regression results perfom as poorly as the linear regression model, with very few notable differences. The best estimator was still 50 PCA components.

3.4 Random Forest Regression Model

The random forest model is a type of additive model that makes predictions by combining decisions from a sequence of base models. A random forest is made by growing a binary tree, and then at each node splitting the data into two children nodes based on residual sum of squares, or RSS. For random forests using regression, the predicted value at a node is the average response variable for all observations in the node.

```
# Implementing random forest
parameters = {'pca__n_components':[10,20,50],'rfr__n_estimators':[10,100],
              'rfr__criterion':['mae', 'mse'], 'rfr__max_features':['auto',
              'sqrt', 'log2']}
pca = PCA()
rfr = RFR()
pipe = Pipeline(steps=[('pca', pca), ('rfr', rfr)])
estimator = GridSearchCV(pipe, parameters, n_jobs = -1, verbose = 1)
estimator.fit(tr_x,tr_y)
es_y3 = estimator.best_estimator_.predict(tt_x)
results(es_y3)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
```

```
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 31.6min
```

```
print(estimator.best_params_)
```

3.5 Catboost Regression Model

There exist several implementations of random forest boosters, among these are GBM, XGBoost, LightGBM, and Catboost. Catboost seems to outperform the other implementations, even by using only its default parameters. Not only is it more accurate, but also faster than the other methods, making it an ideal machine learning model to implement on our dataset. In Boosting each tree is grown using information from previously grown trees, each tree is fit on a modified version of the original data set. Because the random forest regression gave the best results, the next natural step was to examine how Catboost performed on our dataset.

```
# Implementing Catboost
df_y_temp.iloc[0]  = (df_y_temp.iloc[0]  - df_y_temp.iloc[0].mean())
                     / np.std(df_y_temp.iloc[0])
df_y_temp.iloc[3]  = (df_y_temp.iloc[3]  - df_y_temp.iloc[3].mean())
                     / np.std(df_y_temp.iloc[3])
df_y_temp.iloc[8]  = (df_y_temp.iloc[8]  - df_y_temp.iloc[8].mean())
                     / np.std(df_y_temp.iloc[8])
df_y_temp.iloc[9]  = (df_y_temp.iloc[9]  - df_y_temp.iloc[9].mean()
                     / np.std(df_y_temp.iloc[9])
df_y_temp.iloc[10] = (df_y_temp.iloc[10] - df_y_temp.iloc[10].mean())
                     / np.std(df_y_temp.iloc[10])

tr_x, tt_x, tr_y, tt_y = tts(df_x_temp, df_y_temp, test_size = .2)
es_y4 = np.zeros_like(tt_y)

#we use a for loop, because Catboost does not play nicely with GridSearch

for iterations in [50,100]:
    for learning_rate in [.25,.05]:
        for depth in [6,2]:
            estimator = CatBoostRegressor(iterations=iterations,
                                          learning_rate=learning_rate,
                                          depth=depth,l2_leaf_reg=64)
            for i in range(14):
                estimator.fit(tr_x,tr_y.iloc[0:,i],verbose=False)
                es_y4[0:,i] = estimator.predict(tt_x)

            results(es_y4)
```

4 Conclusion

Results	Linear	Ridge	Random Forest	Catboost
Movie Gross Average Percent Error:	21174.044777587114	21163.699509043057	28360.003945479428	13358471.898599533
IMDb Rating Average Percent Error:	10.943972540307133	10.9444872552927	11.57944501421292	11.92775514288592
Meta Score Average Percent Error:	22.233386159641668	22.232002873434183	21.272463125108494	24.406280317642874
Number of Votes Average Percent Error:	227.25520075838648	227.13037457013425	217.09410466870564547 9428	35898.3470609781 5
Oscar Nominations Average Error:	0.36882244830576283	0.3687895465107543	0.34224349349349 35	0.3601782427881311
Oscar Wins Average Error:	0.27553806746243514	0.2754840332277056	0.23656406406406405	0.22244438571207187
Other Nominations Average Error:	8.999050858301405	8.99762356785237	8.54305055055055	8.17355760204589
Other Wins Average Error:	5.735743540563 35	5.735122254309773	5.568672988464655	5.355991969204856
Profit Average Percent Error:	654.222134871736	653.8567426068672	895.3850933188331	10991214.869575225
Gross Adjusted Average Percent Error:	27478.05735135932	27462.440433014228	33359.201330645235	12931284.582092846
Profit Adjusted Average Percent Error:	708.89155030277 22	708.3469770591662	822.3867180805835	10188342.603258893
Profit Bool Average Error:	0.37615950104784 81	0.3761846360719753	0.34198198198 1982	0.31605895288023267
Profit Bool Accuracy:	0.7212212212212212	0.7207207207207207	0.7657657657657657	0.7492492492492493
Total Nominations Average Error:	9.127342662202654	9.126066676232615	8.6553007173840 5	8.286937975403768
Total Wins Average Error:	5.874335561778571 5	5.873626874233154	5.734206021497688	5.49345921232842

After analyzing the results of our current machine learning algorithm implementations, we have seen that profit is a hard thing to predict. Due to the wide range of movies that we train on and the substantial difference in profit they have, we have decided to focus less on the amount of money and whether or not the movie turns a profit. We can predict this with fairly good accuracy and with more fine tuning these methods could become even more accurate. Catboost had a higher error than the rest of the methods in predictions that tend to have a huge inaccuracy. However, in the things we can predict, Catboost was just as good and a great deal faster than the rest. If more fine tuning were required, it would be with Catboost. This would be quite simple and very quick. Overall Catboost was the most effective method.