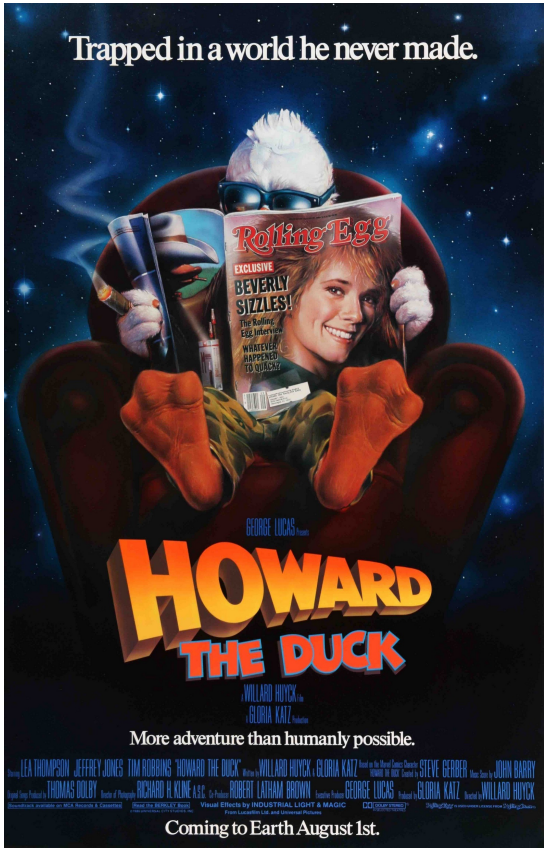
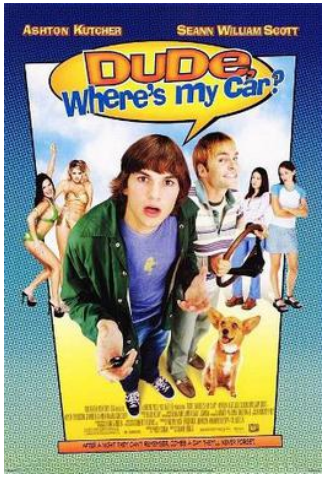
Predicting Movie Success

Mark Hubbard
Itay Segal
Nisha Choondassery
Craig Barbisan

fppt.com





"People never forget two things, their first love and the money they wasted watching a bad movie."
- Amit Kalantri

Frame the Problem

- Classification
  - Will the movie be successful?
- Regression
  - How much revenue will the movie generate?

Defining "Success"

- Number of awards received?
- Positive reviews?
- Tickets sold?
- Break-even?
- Profitable?

- revenue = 2.5 x budget

Feature selction and reduction - continued

```
[207] movies_model = movies_full_filtered.copy().drop('id',axis=1)
```

```
[208] # define target value and features
      target = 'revenue'
      features = list(movies_model.columns)
      features = [f for f in features if f!=target]

      feature_set = movies_model[features]
      target_col = movies_model[target]
```

prepare the train and test data sets

```
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(movies_model, test_size=0.3)

X_tr = train_set[features]
y_tr = train_set[[target]]

X_te = test_set[features]
y_te = test_set[[target]]

#add a new target variable for classification

y_tr_cl = np.ravel(y_tr)/np.ravel(train_set.budget)>2.50
y_te_cl = np.ravel(y_te)/np.ravel(test_set.budget)>2.50
```


JURASSIC WORLD
FALLEN KINGDOM

Data Categories

Movie metadata

Credits (cast and crew)

MovieLens User Ratings

MovieLens Keywords

Movie Identifiers (IMDB, TMDB)

JSON Document - credits

```
print(credits.crew)
```

```
'department': 'Directing', 'gender': 2, 'id': 7879, 'job': 'Director', 'name': 'John Lasseter', 'p
'department': 'Production', 'gender': 2, 'id': 511, 'job': 'Executive Producer', 'name': 'Larry J.
'department': 'Directing', 'gender': 2, 'id': 26502, 'job': 'Director', 'name': 'Howard Deutch', '
'department': 'Directing', 'gender': 2, 'id': 2178, 'job': 'Director', 'name': 'Forest Whitaker',
'department': 'Sound', 'gender': 2, 'id': 37, 'job': 'Original Music Composer', 'name': 'Alan Silv
'department': 'Directing', 'gender': 2, 'id': 638, 'job': 'Director', 'name': 'Michael Mann', 'pro
'department': 'Directing', 'gender': 2, 'id': 2226, 'job': 'Director', 'name': 'Sydney Pollack', '
'department': 'Writing', 'gender': 2, 'id': 2075, 'job': 'Screenplay', 'name': 'David Loughery', '
'department': 'Directing', 'gender': 2, 'id': 37710, 'job': 'Director', 'name': 'Peter Hyams', 'pr
'department': 'Directing', 'gender': 2, 'id': 10702, 'job': 'Director', 'name': 'Martin Campbell',
```

Feature Engineering

```
[87]  # extract the director from the crew field

      def get_director(x):
          for i in x:
              if i['job'] == 'Director':
                  return i['name']
          return 'Unknown'

      movies['director'] = movies['crew'].apply(get_director)
```

```
[105] ## load stop words file

      def get_stop_words(stop_file_path):
          """load stop words """

          with open(stop_file_path, 'r', encoding="utf-8") as f:
              stopwords = f.readlines()
              stop_set = set(m.strip() for m in stopwords)
              return frozenset(stop_set)

      #load a set of stop words
      stopwords=get_stop_words("movies/resources/stopwords.txt")
```

```
[106] # title
      # Vectorizing the titles - using word counts
      count_vectorizer = CountVectorizer(max_features=1000, binary=True, max_df=0.8,stop_words=stopwords)
      titles = movies["title"].replace(np.nan,'')
      titles_transformed = count_vectorizer.fit_transform(titles)
```

New Feature: season

```
[93]  # calculate the season of the release (spring, summer, fall, winter)
      def season_of_date(date):

          if pd.isnull(date):
            return 'unknown'

          year = str(date.year)
          seasons = {'spring': pd.date_range(start='21/03/'+year, end='20/06/'+year),
                     'summer': pd.date_range(start='21/06/'+year, end='22/09/'+year),
                     'autumn': pd.date_range(start='23/09/'+year, end='20/12/'+year)}
          if date in seasons['spring']:
              return 'spring'
          if date in seasons['summer']:
              return 'summer'
          if date in seasons['autumn']:
              return 'autumn'
          else:
              return 'winter'

      # create a new column
      movies['season'] = (movies['release_date']
                              .fillna(pd.NaT)
                              .apply(lambda x: season_of_date(x))
                          )
```

Weighted Rating

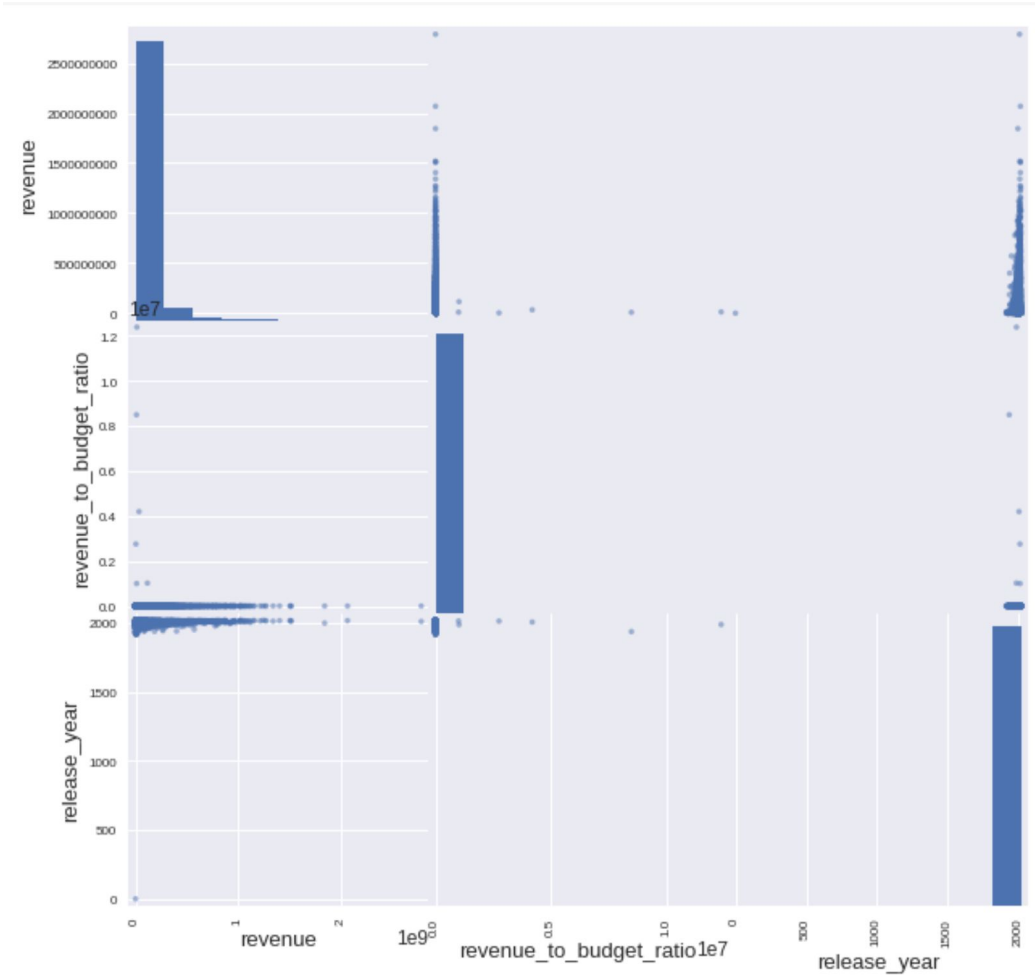
New Feature: weighted_rating

Weighted Rating (WR) = $\left(\frac{v}{v+m} \cdot R\right) + \left(\frac{m}{v+m} \cdot C\right)$ where,

- v is the number of votes for the movie
- m is the minimum votes required to be listed in the chart
- R is the average rating of the movie
- C is the mean vote across the whole report.

Source: https://www.kaggle.com/rounakbanik/movie-recommender-systems

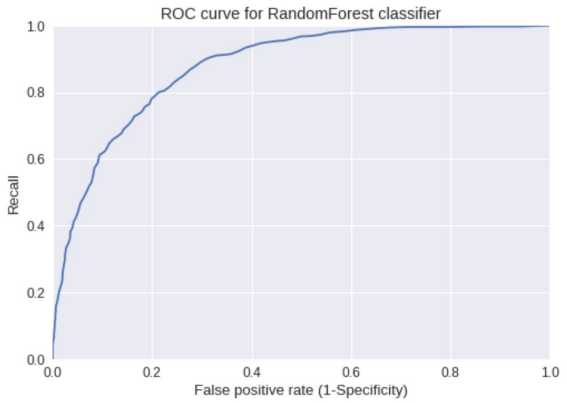
Feature Selection



Classification Algorithms

Algorithm	Accuracy
Logistic Regression	0.7837
K-Nearest Neighbour	0.7619
Decision Tree	0.7983
Random Forest	0.8080
AdaBoost	0.7623
Voting Classifier	0.7890

Random Forest Classification




```
[212] print ("Area under the curve is ", roc_auc_score(y_te_cl,final_cl_model.predict_proba(X_te)[:,1]))
```

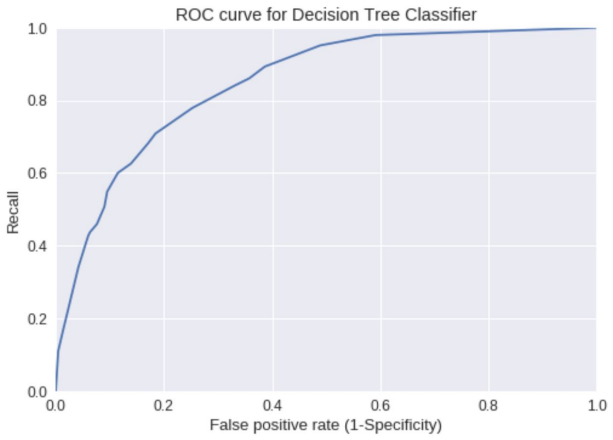
```
Area under the curve is  0.8771569966355831
```

Classification Report

```
[213] print(classification_report(y_te_cl, y_pred_cl))
```

	precision	recall	f1-score	support
False	0.83	0.89	0.86	1524
True	0.74	0.63	0.68	738
micro avg	0.81	0.81	0.81	2262
macro avg	0.79	0.76	0.77	2262
weighted avg	0.80	0.81	0.80	2262

Decision Tree Classifier




```
[218] print ("Area under the curve is ", roc_auc_score(y_te_cl,tree_cv.predict_proba(X_te)[:,1]))
```

```
Area under the curve is  0.8486425858353072
```

```
[216] print(classification_report(y_te_cl, y_pred_tree))
```

	precision	recall	f1-score	support
False	0.82	0.89	0.85	1524
True	0.72	0.60	0.65	738
micro avg	0.79	0.79	0.79	2262
macro avg	0.77	0.74	0.75	2262
weighted avg	0.79	0.79	0.79	2262

Regression Algorithms

Algorithm	Error
Linear Regression	75100665
Ridge Regression	75375487
Lasso Regression	75003445
ElasticNet	75000055

Conclusions

- Release year doesn’t correlate with revenue
- ElasticNet provided best regression results but error was too high to be useful
- Random Forest produced the best classification results
- Better at predicting failure than success, which is also a useful prediction