Machine Learning Engineer Nanodegree

Capstone Project

Jude Chen

February 22, 2018

I. Definition

Project Overview

Movie database such as **IMDb** has been well-known of collecting information from films including cast, storyline, production company, budget and technical specs. The audiences can share their reviews online and rate the film from 1 to 10. We are curious about the reasons why people like these great movies and this is the question getting more important to the production industry. We use the datasets from Kaggle[1], collected 5000 movies from **TMDb**. Due to the copyright concern, Kaggle replaced their movie datasets from IMDb to The Movie Database (TMDb). The project is going to train several models to learn from the movie data, to dig out the features of the movie basic information which lead to a successful movie and the result will be represented as the movie rating.

Problem Statement

Our goal is to find out the characteristics which make a good movie and further predict that movie is going to be successful or not. We use the 'vote average' data to be our target value and draw a line by 7.0 to judge if it is a good movie, it may get a voting score of 7.0 or higher. The other movie data like 'budget', 'popularity', 'revenue', ... will be preprocessed and trained to generate a best model. This model is going to predict the movie rating whenever a new series of movie data comes in.

By going through these tasks, we should get closer to our goal:

- **Data Preprocessing:** The data abnormalities and characteristics such as missing value, json features and skewed data will be preprocessed for the training.
- **Implementation:** Utilize 'Supervised Learning' to train a model that predicting the target voting. Use different metrics for evaluating the model performance.
    - **Metrics:** accuracy score, f_beta score, cv score, running time of training and testing sets

[1] https://www.kaggle.com/tmdb/tmdb-movie-metadata

    - **Models:** Ada Boost, Random Forest, Gradient Boosting, XGBoost
- **Refinement:** By tuning parameters and feature selection to improve the model predicting performance.

Metrics

We use these metrics to evaluate the model performance in both training and testing data sets:

- **Accuracy score:** The accuracy classification score can help us to justify whether each model is predicting well against the actual voting average. Compared to the actual voting average, the correctly predicted classification score shows how the movies are correctly predicted as good movies(voting > 7.0) or general movies(voting < 7.0) by each model.
- **F_beta score:** We choose the beta = 0.5 which means we put more weights to precision. Because we care more about the correctly predicted successful movie which is rating over 7.0, we put more emphasis on the precision criteria, that is what we predicted as successful movie were actually rating > 7.0.
- **CV score:** Because our data is about only 5000 movies, we want to add variety to the data input and to avoid overfitting on existing data, k-fold cross validation is used to split data into k sets and the model is trained by different testing sets each time to get the average score. This is help our model to predict well on unseen data.
- **Running time:** The running time spent is considered to leverage between accuracy and efficiency. Between 2 models both predicting a good accuracy score, a model spending a faster running time may be our better choice.

II. Analysis

Data Exploration

The data is collected from Kaggle, TMDB 5000 Movie Dataset[2] including 2 data files - credits data and movies data. Credits data consists of a movie's casts and crews, movies data consists of the information like budget or revenue of each movie. The more detailed discussion will be made within the data visualization.

The data characteristics or abnormalities will be addressed in the sections:

- **Feature Distribution:** The skewed feature data will be displayed and should apply data transformation during data preprocessing.
- **Categorical values:** The features with categorical values or json values will be introduced and should apply One-hot encoding[3] during data preprocessing.
- **Missing values:** The feature data row with missing values will be indicated and excluded.
- **Outliers:** We focus on the outliers which data value equals to 0. It may because the data is not well collected and we don't want those incomplete data to impact the training program.

---

[2] https://www.kaggle.com/tmdb/tmdb-movie-metadata
[3] https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

Movies Data

budget	genres	homepage	id
237000000	[{"id": 28, "name": "Action"},	http://www.avatarmovie.com/	19995

keywords	original_language	original_title	overview	popularity
[{"id": 1463, "name": "culture clash"}, {"id": 2964, "name": "future"},	en	Avatar	In the 22nd century, a paraplegic	150.437577

production_companies	production_countries	release_date	revenue	runtime	spoken_languages
[{"name": "Ingenious Film Partners", "id": 289}, {"name":	[{"iso_3166_1": "US", "name": "United States of America"},	2009-12-10	2787965087	162	[{"iso_639_1": "en", "name": "Espa\u00f1ol"}]

status	tagline	title	vote_average	vote_count
Released	Enter the World of Pandora.	Avatar	7.2	11800

Fig.1 Data input - Movies

Movies Feature Exploration:

Numerical values

- **budget:** continuous
- **id:** movie id
- **popularity:** continuous
- **revenue:** continuous
- **runtime:** continuous
- **vote_count:** continuous
- **vote_average:** *(Target value)* continuous. The audiences can share their reviews online and rate the film from 1 to 10. It will be treated as our target value and be turned into a classification value 1 or 0 representing whether the value is >=7.0 or <7.0.

Categorical values

- **status:** Post Production, Released, Rumored

Json values

- **genres:** Drama, Comedy, Thriller, Action, Romance, ...
- **keywords:** woman director, independent film, duringcreditsstinger, based on novel, murder, ...
- **production_companies:** Warner Bros., Universal Pictures, Paramount Pictures, Twentieth Century Fox Film Corporation, Columbia Pictures, ...
- **production_countries:** United States of America, United Kingdom, Germany, France, Canada, ...
- **spoken_languages:** English, Fran\xe7ais, Espa\xf1ol, Deutsch, Italiano, ...

String values

- **homepage:** movie web page
- **overview:** movie overview
- **tagline:** movie tagline
- **original_language:** movie original language
- **original_title:** movie original title
- **title:** movie title

DateTime values

- **release_date:** DateTime

Credits Data

movie_id	title	cast	crew
19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "credit_id": "5602a8a7c3a3685532001c9a",	[{"credit_id": "52fe48009251416c750aca23", "department": "Editing", "gender": 0, "id": 1721, "job": "Editor", "name": "Stephen E. Rivkin"}, {"credit_id":

Fig.2 Data input - Credits

Credits Feature Exploration:

Numerical values

- **movie_id:** movie id

Json values

- **cast:** movie actors - Robert De Niro, Matt Damon, Samuel L. Jackson, ...
- **crew:** film crew - Steven Spielberg, Woody Allen, Martin Scorsese, ...

String values

- **title:** movie title

Feature Distribution

- According to the scatter matrix plot, the numerical feature 'budget', 'id', 'popularity', 'revenue', 'vote_count' is in **right-skewed** distribution.
- The feature 'runtime' is slightly **right-skewed**.
- The target value 'vote_average' is slightly **left-skewed**, most of the value falls into 7~8.
- We will apply log transformation to highly **right-skewed** features - 'budget', 'popularity', 'revenue', 'vote_count' to reduce the range of values caused by outliers which may negatively affect the learning model performance. The feature 'id' will be dropped because it does not help.

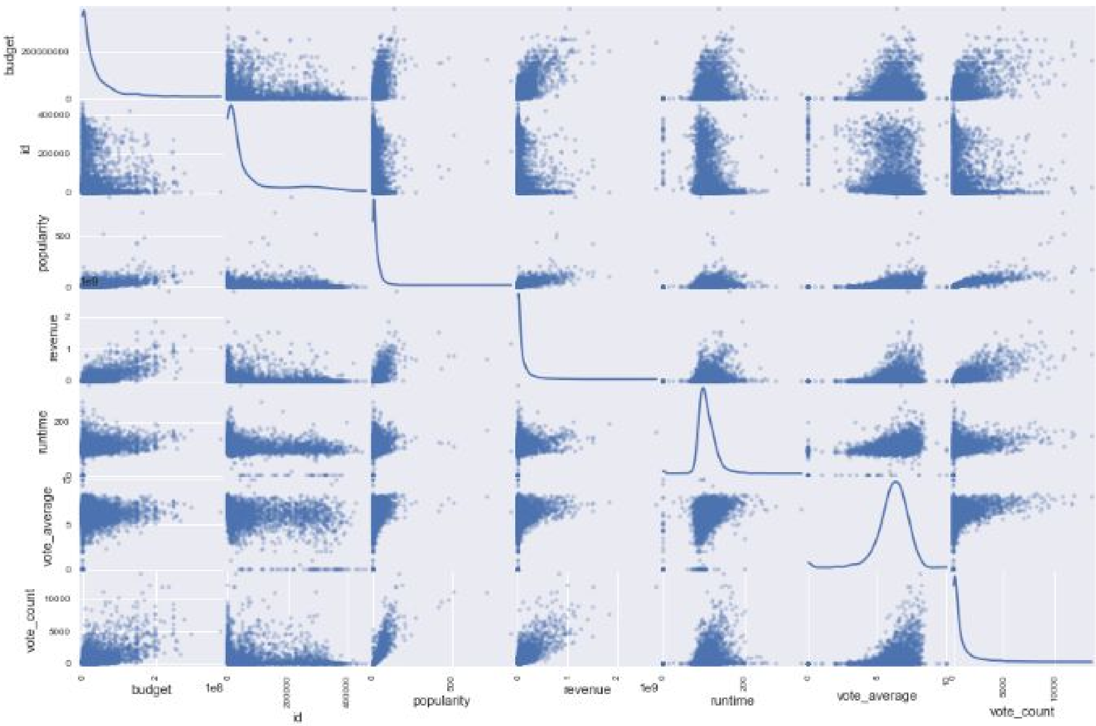


Fig.3 Feature distribution of movies data

Categorical values

Categorical values will be transformed by **one-hot encoding**.

- **Status:** Post Production, Released, Rumored

Json values will be preprocessed and filtered because they contains so many items.

- **cast:** movie actors - Robert De Niro, Matt Damon, Samuel L. Jackson, ...
- **crew:** film crew - Steven Spielberg, Woody Allen, Martin Scorsese, ...
- **genres:** Drama, Comedy, Thriller, Action, Romance, ...
- **keywords:** woman director, independent film, duringcreditsstinger, based on novel, ...
- **production_companies:** Warner Bros., Universal Pictures, Paramount Pictures, ...
- **production_countries:** United States of America, United Kingdom, Germany, France, ...
- **spoken_languages:** English, Fran\xe7ais, Espa\xf1ol, Deutsch, Italiano, ...

Missing values

- 'release_date', runtime' are features we care, the missing values in these features will be dropped.
- 'homepage', 'overview', 'tagline' are data that will not be used because they are not measurable.

Outliers

- There are many data value are 0s. It is weird to have 0 budget or 0 runtime in a movie.

- Applying to the numerical features: 'budget', 'popularity', 'revenue', 'runtime', 'vote_count', 1576 outliers are indicated as 0 value data, we will drop it in the data preprocessing.

```
Outliers for feature 'budget': 1037
```

	budget	genres	homepage	id	keywords	original_language	original_title
265	0	[{u'id': 35, u'name': u'Comedy'}, {u'id': 14, ...	NaN	10588	[{u'id': 977, u'name': u'cat'}, {u'id': 1155, ...	en	The Cat in the Hat
321	0	[{u'id': 35, u'name': u'Comedy'}]	NaN	77953	[{u'id': 6078, u'name': u'politics'}, {u'id': ...	en	The Campaign
359	0	[{u'id': 12, u'name': u'Adventure'}, {u'id': 1...	http://www.foxmovies.com/movies/alvin-and-the-...	258509	[{u'id': 10986, u'name': u'chipmunk'}, {u'id':...	en	Alvin and the Chipmunks: The Road Chip

Fig.4 Zero-Outliers of data - budget

Exploratory Visualization

We want to figure out the relationship among features, the heatmap will show how close they correlates to each other. A series of scatter and regression plots will be discussed in more detail about how each feature related to the target value.

- **Feature Relevance:** A heatmap can show the relevance among the numerical features: 'budget', 'popularity', 'revenue', 'runtime', 'vote_count', 'id', 'vote_average'. The most correlated features will be picked up and discussed in more details.
- **Feature Exploration:** The features have higher relevance with the target value 'vote_average' will be chosen. The scatter plot of distribution of each chosen feature with target value will be plotted and a regression line will be indicated for the relationship. The feature 'release_date' will also be discussed in the relevance with 'vote_average'.

Feature Relevance

- According to the heatmap, the numerical feature 'runtime', 'vote_count', 'popularity', 'revenue', 'budget' has more relevance with the target 'vote_average'.

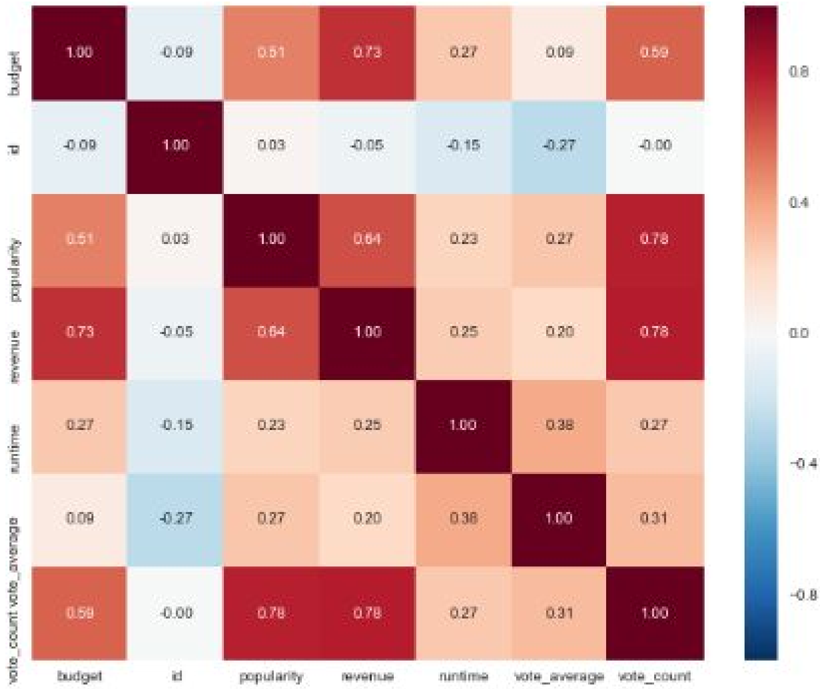
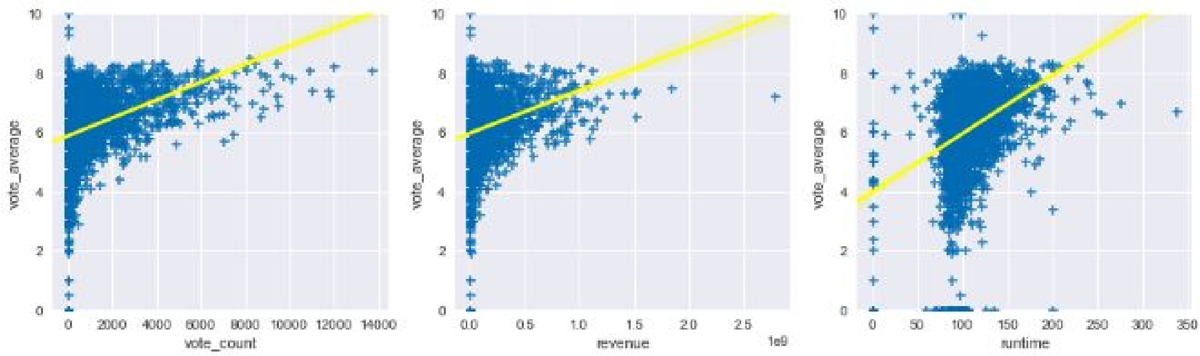


Fig.5 Feature relevance - Heatmap

Feature Exploration

- **vote_count** gets higher, the vote_average are usually better. If this is a good movie, more people may have watched it and they are willing to vote for a suggestion for whom haven't seen it.
- **revenue** gets higher, the vote_average are usually better. A good movie will attract people to watch it and therefore gets a good revenue.
- **runtime** usually falls in 80~120 minutes. It generally shows when the runtime is higher, the vote_average will be better. The movie has more time to tell the story to the audience is better than nothing to tell.
- **popularity** gets higher, the vote_average grows definitely higher. The popularity can represent how the people love to watch this movie.
- **budget** does not really impact on the vote_average. The regression line is almost flat. Good movies can have a low budget or a high budget spent on them.
- **release_date** largely falls in 2001~2017. It seems that the old movies can get a better vote_average but I think it's because the old data was not well collected.



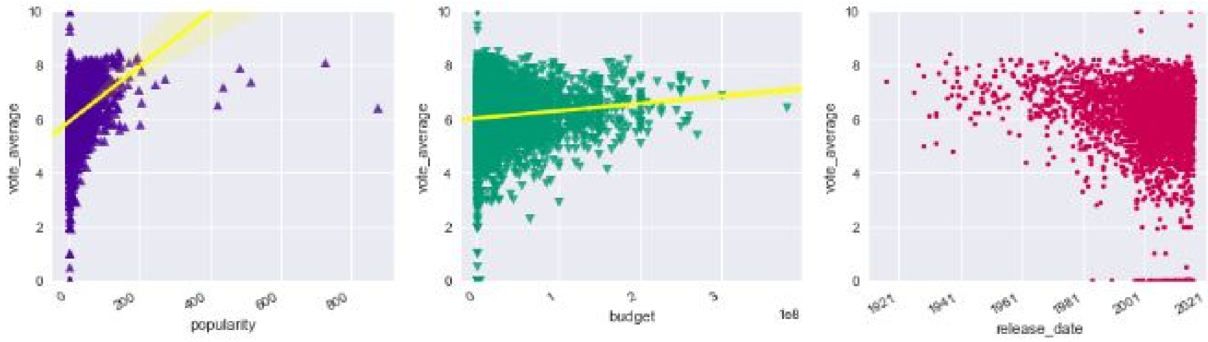


Fig.6 Feature exploration of movies data

The 'vote_average' shows a slightly **left-skewed** distribution, and the highest distribution is around vote_average 6 to 8. This will be our target value, the classification will be set at 7.0, the good movies vote_average > 7.0 are labeled as 1, the other general movies with vote_average < 7.0 are labeled as 0.

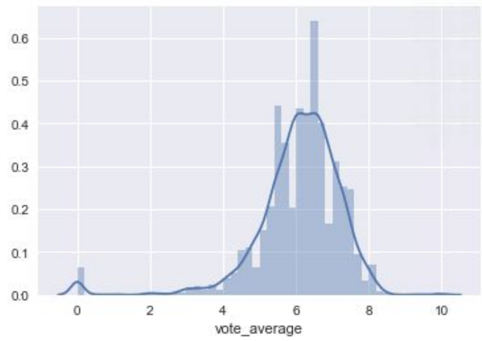


Fig. 6-1 Feature exploration - vote_average

Algorithms and Techniques

The problem to predict a good movie is going to be a classification problem. The target value is separated by 7.0 and the good movie we want to find is those of voting 7.0 or higher. We use the Ensemble Methods of Supervised Learning to solve this problem.

In this project, we want to figure out the feature importance and relevance to our target. We choose Ensemble Methods such as 'Ada Boost', 'Random Forest' and 'Gradient Boosting' which can automatically learn feature interactions well. They can still handle lots of irrelevant features or outliers well, this characteristic can deal with some irrelevant features in json input data like movie keywords or genres. Besides, we also implement 'XGBoost', the outperforming method in Kaggle competition which is an advanced version of 'Gradient Boosting'.

We use the default parameters implemented by learning models, for example in XGBoost, default min_child_weight=1, subsample=1. Furthermore, we will tune parameters to improve model performance in the Refinement section.

- **Ada Boost:** The boosting learning is trying to combine several weak learners to form a strong rules. It starts by a first learner predicting data set and gives equal weight to them. If certain data prediction is incorrect, it put higher weight on them. Iteratively adding learners and doing the previous process, it will stop until it reach the maximum number of models or accuracy.

- **Random Forest:** It uses the bagging learning to build many independent models and combine them using averaging methods like weighted average or majority voting. It builds each model by a decision tree which use some randomness in selecting the attribute to split at each node. The training data set is being sampling with replacement, the decision trees are fitted using the bootstrap samples and combined by voting the output.
- **Gradient Boosting:** It trains a sequence of model that gradually minimizes the loss function (the inaccuracy of the predictions) by using Gradient Descent (to take steps proportional to the negative of the gradient). The learning procedure consecutively fit new models for predicting a more accurate target value.
- **XGBoost:** It is called e**X**treme **G**radient **Boost**ing, an advanced version based on Gradient Boosting. XGBoost uses a more regularized model formalization to control the complexity of the model which can help to avoid over-fitting. The general principle of regularization is that it want both a simple and predictive model.

Benchmark

We introduce 'Decision Tree' to compare with our training models, according to the research [A movie recommender system based on inductive learning][4] and [Automatic movie ratings prediction using machine learning][5], the decision tree is often chosen to be the training model for movie data predicting voting values.

- **Decision Tree:** It trains a flowchart-like structure as a tree. First, it picks a best question (attribute) as a root node that split the data into subsets following the answer path (branch) and iteratively pick another question. Repeat it until all the questions are answered, the result (leaf node) will be a class label. The best attribute to choose is which we can get the most uncertainty decreased (maximum Information Gain).

III. Methodology

Data Preprocessing

- **Input data handling**: Includes data separation into features and target. Target data **vote_average** will be encoded into 0: vote_avergae < 7.0 and 1: vote_avergae >= 7.0 to represent a good movie. vote_average is spilt from other data and copied as target data.Some features are irrelevant or unmeasurable to our training results will be dropped.
- **Drop missing data**: Outliers with value 0 in any column that found in previous section will be dropped. Some missing data with value **NaN** will also be dropped.
- **Feature Scaling and Normalizing**: In order to solve data skewing problem, log transformation and MinMaxScaler will be applied to form a better feature distribution.

[4] http://ieeexplore.ieee.org/document/1460433/

[5] http://ieeexplore.ieee.org/document/5967324/

- **Json input handling**: Some data columns given as json format will be flattening and using Word Cloud to pick the keywords as features. These features will apply one-hot encoding as new data input.
- **Training data separation**: Since we have 5000 movie datasets, we can split the data into Training sets(70%) and Testing sets(30%).

Feature and Target

- **vote_average** is split from other data and copied as target data.
- The rest of columns(budget, popularity, release_date, revenue, status, vote_count) in movies data or credits data will be the feature data.
- Some movies data (genres, keywords, production_companies, production_countries, spoken_languages) or credits data(cast, crew) in json format will be excluded for preprocessing.
- Some features (id, spoken_languages, original_language, original_title, overview, tagline, title) are irrelevant or unmeasurable to our training results will be dropped.

```
# Set target = vote_average
target_raw = movies.copy()
target_raw = movies['vote_average']

# The rest of data will be features
features_raw = movies.copy()
features_raw2 = credits.copy()
features_raw.drop('vote_average', axis = 1, inplace=True)
```

```
features_drop_na = features_raw.copy()

# Drop irrelevant and unmeasurable features
features_drop_na.drop('id', axis = 1, inplace=True)
features_drop_na.drop('homepage', axis = 1, inplace=True) # web url is irrelevant
features_drop_na.drop('original_language', axis = 1, inplace=True) # focus on spoken_languages
features_drop_na.drop('original_title', axis = 1, inplace=True) # original_title is not measurable
features_drop_na.drop('overview', axis = 1, inplace=True) # overview is not measurable
features_drop_na.drop('tagline', axis = 1, inplace=True) # tagline is not measurable
features_drop_na.drop('title', axis = 1, inplace=True) # title is not measurable

# Drop json features
features_drop_na.drop('genres', axis = 1, inplace=True)
features_drop_na.drop('keywords', axis = 1, inplace=True)
features_drop_na.drop('production_companies', axis = 1, inplace=True)
features_drop_na.drop('production_countries', axis = 1, inplace=True)
features_drop_na.drop('spoken_languages', axis = 1, inplace=True)

features_drop_na.head(5)
```

	budget	popularity	release_date	revenue	runtime	status	vote_count
0	237000000	150.437577	2009-12-10	2787965087	162.0	Released	11800
1	300000000	139.082615	2007-05-19	961000000	169.0	Released	4500
2	245000000	107.376788	2015-10-26	880674609	148.0	Released	4466
3	250000000	112.312950	2012-07-16	1084939099	165.0	Released	9106
4	260000000	43.926995	2012-03-07	284139100	132.0	Released	2124

Fig. Data - feature and target preprocessing

Drop Missing data

In case the missing data will impact on the training result as a bad feature, we will drop them and reset the data row index.

- 1576 outliers with value 0 was indicated from the Data Exploration, they will be dropped.
- Missing data as below will also be dropped:

Feature	Value	Count
release_date	NaT	1
runtime	NaN	2

```
# Collect missing data row index
nulls = pd.isnull(features_drop_na).any(1).nonzero()[0]

# Append missing data into 0 value outliers list
nas = np.asarray(missing_outliers)
for i in nulls:
    if i not in nas:
        print(i)
        nas = np.append(i, nas)

print('Missing data', len(nas))
```

```
('Missing data', 1576)
```

```
# Drop missing data for movies data
features_drop_na.drop(nas, inplace=True)
features_drop_na.reset_index(drop = True, inplace=True)
display(features_drop_na.describe())
```

	budget	popularity	revenue	runtime	vote_count
count	3.227000e+03	3227.000000	3.227000e+03	3227.000000	3227.000000
mean	4.067877e+07	29.051491	1.213181e+08	110.720793	977.893090
std	4.439974e+07	36.169863	1.863361e+08	20.970364	1414.538507
min	1.000000e+00	0.019984	5.000000e+00	41.000000	1.000000
25%	1.050000e+07	10.475904	1.704008e+07	96.000000	178.000000
50%	2.500000e+07	20.415572	5.519828e+07	107.000000	471.000000
75%	5.500000e+07	37.345773	1.463949e+08	121.000000	1148.000000
max	3.800000e+08	875.581305	2.787965e+09	338.000000	13752.000000

```
# Drop missing data for credits data
features_raw2.drop(nas, inplace=True)
features_raw2.reset_index(drop = True, inplace=True)
display(features_raw2.describe())
```

	movie_id
count	3227.000000
mean	44601.870778
std	74281.771931
min	5.000000
25%	4954.500000
50%	11442.000000
75%	45256.000000
max	417859.000000

```
# Drop missing data for target data
target_raw.drop(nas, inplace=True)
target_raw.reset_index(drop = True, inplace=True)
display(target_raw.describe())
```

```
count    3227.000000
mean        6.313263
std         0.859921
min         2.300000
25%         5.800000
50%         6.300000
75%         6.900000
max         8.500000
Name: vote_average, dtype: float64
```

Fig. Drop missing data

Feature Scaling

The numerical data is not normally distributed, according to the scatter_matrix plot, the data is highly right-skewed in some features. We apply **logarithmic transformation** on these features 'budget', 'popularity', 'revenue', 'vote_count' to reduce the range of very large and very small values that may negatively affect the learning model to perform a bad result.

```
# Log-transform to skewed features
skewed = ['budget', 'popularity', 'revenue', 'vote_count']

features_log_transformed = features_drop_na.copy()

features_log_transformed[skewed] = features_drop_na[skewed].apply(lambda x: np.log(x + 1))

features_log_transformed.describe()
```

	budget	popularity	revenue	runtime	vote_count
count	3227.000000	3227.000000	3227.000000	3227.000000	3227.000000
mean	16.801634	2.998848	17.496617	110.720793	6.038886
std	1.660813	0.935795	2.067760	20.970364	1.446256
min	0.693147	0.019787	1.791759	41.000000	0.693147
25%	16.166886	2.440250	16.651076	96.000000	5.187386
50%	17.034386	3.064118	17.826442	107.000000	6.156979
75%	17.822844	3.646644	18.801818	121.000000	7.046647
max	19.755682	6.776029	21.748578	338.000000	9.529012

Fig. Feature Scaling

Feature Normalizing

After applying feature scaling to data, we apply **normalization** to ensure that each feature is treated equally when applying supervised learners. We use the MinMaxScaler [6] to scale and translate each feature into the range of 0 to 1 on the data set.

```
# Features shrinked range into 0 to 1 (default)
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
numerical = ['budget', 'popularity', 'revenue', 'runtime', 'vote_count', 'release_date']

features_log_minmax_transform = pd.DataFrame(data = features_log_transformed[numerical])
features_log_minmax_transform[numerical] = scaler.fit_transform(features_log_transformed[numerical])
features_log_minmax_transform.describe()
```

	budget	popularity	revenue	runtime	vote_count	release_date
count	3227.000000	3227.000000	3227.000000	3227.000000	3227.000000	3227.000000
mean	0.845034	0.440935	0.786942	0.234750	0.605005	0.144733
std	0.087124	0.138508	0.103612	0.070607	0.163680	0.132628
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.811736	0.358256	0.744573	0.185185	0.508636	0.058267
50%	0.857244	0.450595	0.803469	0.222222	0.618370	0.111388
75%	0.898605	0.536816	0.852343	0.269360	0.719058	0.179948
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Fig. Feature Normalization

We can see that after feature scaling and normalization, the feature 'popularity' and 'vote_count' are turned into normal distribution. The highly right-skewed characteristic of the other features is decreased.

[6] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

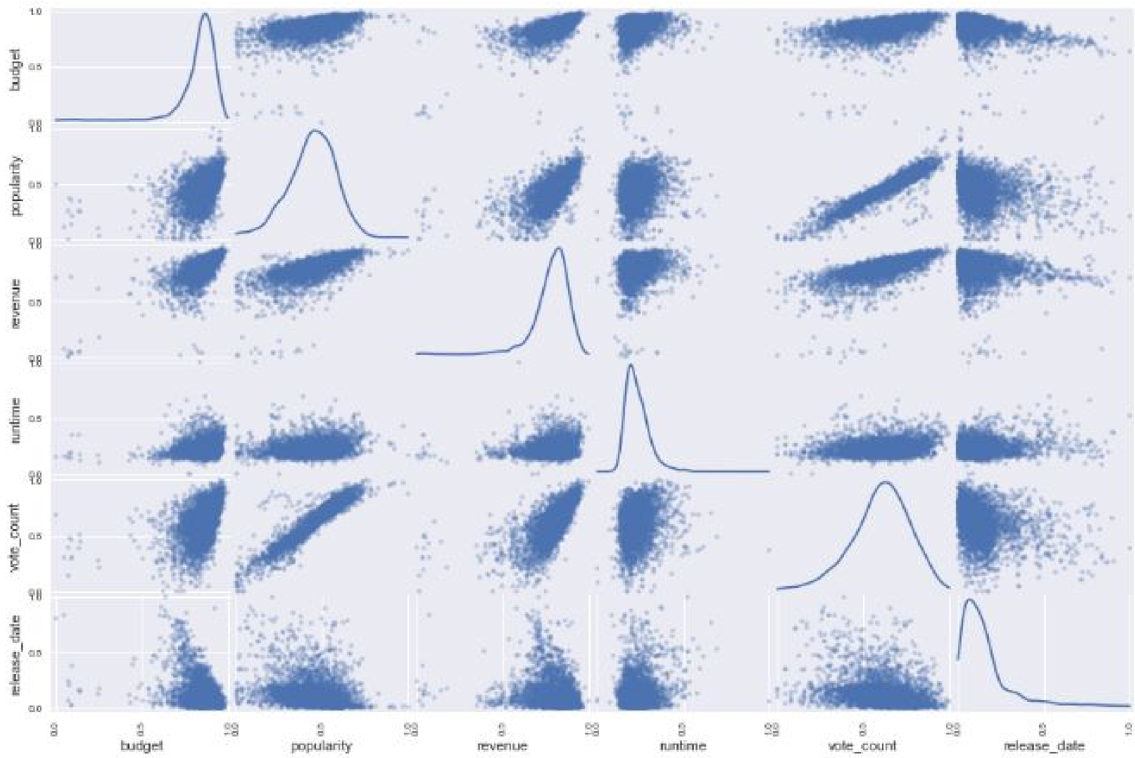


Fig.7 Feature scaling & normalizing on movies data

Json Input Handling

We access the json data with some important keywords in each movie. The frequency of each keyword is collected from the **Word Cloud**, we will pick the best of those as features because we want the data as much as possible to train. We will then apply **One-hot encoding** to those data as new feature data.

- **Credits**: Directors, Actors
- **Movies**: genres, keywords, production_companies, production_countries, spoken_languages

The data type transformed among dataframe, list, series is kind of complication parts to deal with forthe word cloud generating and feature preprocessing.

```
# Find keywords in each movie; Return "NaN" when not finding values
# Return string instead of np.nan for later preprocessing
def safe_access(container, index_values):
    result = container
    try:
        for idx in index_values:
            result = result[idx]
        return result
    except IndexError or KeyError:
        return "NaN"
```

```
# Get actors and directors from the credits data
data = pd.DataFrame()
data['actor1'] = features_raw2.cast.apply(lambda x: safe_access(x, [0, 'name']))
data['actor2'] = features_raw2.cast.apply(lambda x: safe_access(x, [1, 'name']))
data['actor3'] = features_raw2.cast.apply(lambda x: safe_access(x, [2, 'name']))
data['director'] = features_raw2.crew.apply(lambda x: safe_access_with_logic(x, 'job', 'Director', 'name'))

result = []
for i in range(len(data['actor1'])):
    tmp = []
    tmp.append(data['actor1'][i])
    tmp.append(data['actor2'][i])
    tmp.append(data['actor3'][i])
    result.append(tmp)
data['actors'] = pd.Series(data=result)

data.iloc[0:5, 3:5]
```

	director	actors
0	James Cameron	[Sam Worthington, Zoe Saldana, Sigourney Weaver]
1	Gore Verbinski	[Johnny Depp, Orlando Bloom, Keira Knightley]
2	Sam Mendes	[Daniel Craig, Christoph Waltz, Léa Seydoux]
3	Christopher Nolan	[Christian Bale, Michael Caine, Gary Oldman]
4	Andrew Stanton	[Taylor Kitsch, Lynn Collins, Samantha Morton]

Fig. Json data preprocessing

Word Cloud

We use the WordCloud[7] to generate the frequency in each json data keywords.

```
from wordcloud import WordCloud
# Generate a word cloud image
def createWordCloud(cnt, max_num, max_size):
    wordcloud = WordCloud(background_color="black",
                          max_words=max_num,
                          max_font_size=max_size);
    wordcloud.generate_from_frequencies(cnt)
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.show()
```

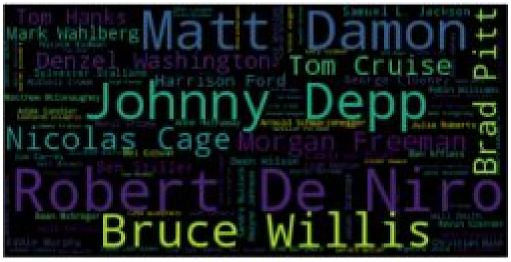
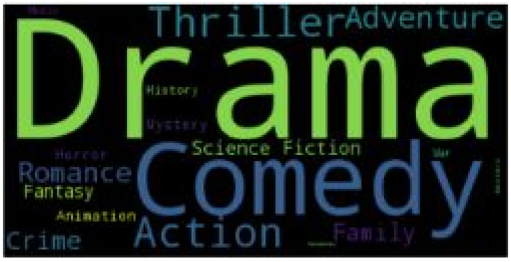
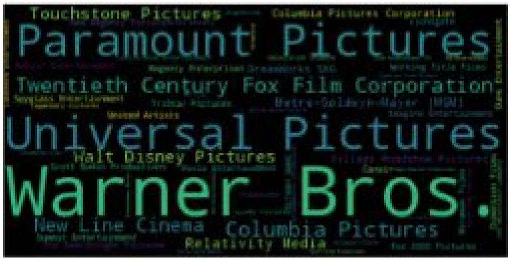
```
# Create word count
# Skip NaN and empty string
def wordCount(text):
    wordCnt = {}
    text = text.replace('', np.nan)
    text = text.replace('NaN', np.nan)
    text.dropna(axis=0)

    wordCnt = text.value_counts().to_dict()
    return wordCnt
```

```
# Create word count
cnt = wordCount(data['director'])

# Create word cloud of top 200
createWordCloud(cnt,200,200)

# Get top 50 word count
director = sorted(cnt.items(), key=itemgetter(1), reverse=True)[:50]

print("director", len(cnt))
print(director)
```

Table 1 Word Cloud of movies data

Directors	Actors

[7] https://github.com/amueller/word_cloud

	
Genres	**Keywords**
	
Production Companies	**Production Countries**
	
Spoken languages	
	

One-hot Encoding

Because there are so many features ('actors' has 3647 different values, 'keywords' has 8131 values), we do one-hot encoding on the features of higher frequency generated from WordCloud. The new feature column is named as 'feature_keyword'. If the keyword value is appeared in this movie, then we apply 1 to the data, else apply 0.

	director_Steven Spielberg	director_Clint Eastwood	director_Ridley Scott	director_Martin Scorsese	director_Renny Harlin	director_Steven Soderbergh	director_Robert Zemeckis
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0

Fig.8 One-hot encoding on movies data

The final data is 3227 movies and 307 features, sample:

	budget	popularity	revenue	runtime	vote_count	release_date	status_Post Production	status_Released	director_Steven Spielberg
0	0.975234	0.740114	1.000000	0.407407	0.982676	0.067479	0	1	0
1	0.987599	0.728577	0.946630	0.430976	0.873588	0.093102	0	1	0
2	0.976975	0.690595	0.942256	0.360269	0.872730	0.008733	0	1	0
3	0.978035	0.697187	0.952708	0.417508	0.953348	0.041500	0	1	0
4	0.980092	0.560260	0.885573	0.306397	0.788647	0.045086	0	1	0

5 rows × 307 columns

Fig.9 Preprocessed feature set of movies data

Result Encoding

- The vote_average data is encoded into 0: value < 7.0 and 1: value >= 7.0 as the target value.
- Target data with voting >= 7.0 is **24%**(768) in total data, while voting < 7.0 is **76%**(2459)
- The target value shows **right-skewed** in the distribution because the skewness value is > 0. (skewness value > 0 means that there is more weight in the right tail of the distribution)

```
# 1: Good movie, 0: General movie, separated by vote=7.0
target = target_raw.apply(lambda x:1 if x >= 7.0 else 0)
num_general, num_good = (target.value_counts())
num_total = len(target)

print('vote >= 7.0', num_good, "{:.0f}%".format(num_good/num_total * 100))
print('vote <  7.0', num_general, "{:.0f}%".format(num_general/num_total * 100))
```

```
('vote >= 7.0', 988, '21%')
('vote <  7.0', 3815, '79%')
```

```
stats.skew(target)
```

```
1.4561297167206089
```

Shuffle and Split Data

The 3227 data after preprocessing is split into training sets(2258) and testing sets(969) for model training.

```python
from sklearn.model_selection import train_test_split
# training data:70%, testing data:30%
features_final = features_preprocessed.copy()
X_train, X_test, y_train, y_test = train_test_split(features_final,
                                                    target,
                                                    test_size = 0.30,
                                                    random_state = 0)

print "Training set has {} samples.".format(X_train.shape[0])
print "Testing set has {} samples.".format(X_test.shape[0])
```

```
Training set has 2258 samples.
Testing set has 969 samples.
```

Implementation

We will introduce several benchmark methods and learning models to predict the movie data.

- **Benchmark:** Decision Tree.
- **Models**: Ada Boost, Random Forest, Gradient Boosting, XGBoost.
- **Metrics**: accuracy score, f_beta score, cv score, running time.

Decision Tree

The Decision Tree first picks a question as a root node (ex:vote count > 4000?), that split the data into subsets following the answer path (yes:count>4000; no:count<4000). Pick the other question (ex:movie genre is drama?) and iteratively repeat the previous steps until no more questions to answer and the result is reached (ex:vote average = 1 or 0?).

```python
# Training and Evaluating results of 'Decision Tree'
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()

results['Decision Tree'] = train_predict(dt, X_train, y_train, X_test, y_test)

pp.pprint(results['Decision Tree'])
```

```
{'acc_test': 0.80392156862745101,
 'acc_train': 1.0,
 'cv_score': 0.69319327973257 25,
 'f_test': 0.55188679245283023,
 'f_train': 1.0,
 'pred_time': 0.011000156402 58789,
 'train_time': 0.089999914169 31152}
```

Fig.11 Decision Tree

Ada Boost

It create a weak learner (ex:vote count > 4000?) to train the data set and put more weights on the incorrectly predicted data, ex: (voting < 7.0) was incorrectly predicted to be (vote average = 1). After iterative adding new learners (ex:popularity > 100?)and repeat the processes, it will combine learners to form a strong rule to predict a good movie.

```python
# Training and Evaluating results of 'Ada Boost'
from sklearn.ensemble import AdaBoostClassifier

ab = AdaBoostClassifier()

results['Ada Boost'] = train_predict(ab, X_train, y_train, X_test, y_test)

pp.pprint(results['Ada Boost'])
```

```
{'acc_test': 0.82662538699690402,
 'acc_train': 0.84853852967227639,
 'cv_score': 0.73967609002622403,
 'f_test': 0.59883720930232565,
 'f_train': 0.71229050279329609,
 'pred_time': 0.06300020217895508,
 'train_time': 0.7129998207092285}
```

Fig.12 Ada Boost

- **Testing Accuracy Score** = *0.83* which is better than decision tree.
- **Testing F score** = *0.60* which is better than decision tree.
- **Training scores** are lower than decision tree, *no over-fitting*.
- **CV Score** = *0.74* which is better than decision tree.
- **Running time** < *0.8* secs, predicting time is faster than training time.

```python
# Create plot feature importnace of 'Ada Boost'
ab_importances = ab.feature_importances_

feature_plot(ab_importances, X_train, 30)
```

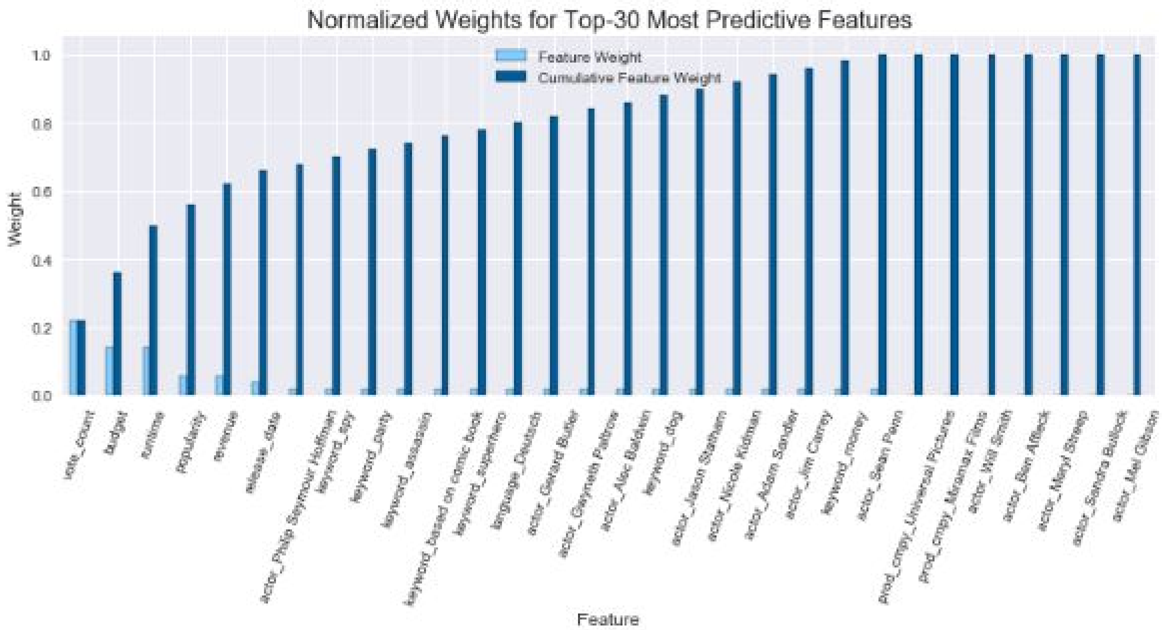


Fig.13 Feature importance - Ada Boost

- **Feature Importance** top ratings are numerical features add up to > 60% weights: 'vote_count', 'budget', 'runtime', 'popularity', 'revenue' and 'released_date'.
- I think it is because all the data have values in these numerical features unlike the json features after one-hot encoding which have little data marked as valid 1s. It is hard for the model to learn json features' characteristic.

Random Forest

It builds many decision trees that use some randomness in selecting the attribute at each node. For example, the models are not allowed to ask the same question (ex:movie genre is drama?) because of randomness.

The bootstrapped training data is also used, for example, Model_A is given the information that (movie genre is 'drama' has higher voting average) and (movie genre is 'action' also has higher voting average) while Model_B is given that (movie genre is drama has 'much' higher voting average) and (movie genre is 'documentary' has 'lower' voting average).

The result will combine the decision trees (random forests) by voting the output.

```
# Training and Evaluating results of 'Random Forest'
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

results['Random Forest'] = train_predict(rf, X_train, y_train, X_test, y_test)

pp.pprint(results['Random Forest'])
```

```
{'acc_test': 0.8286893704850361,
 'acc_train': 0.98184233835252432,
 'cv_score': 0.73967925996368977,
 'f_test': 0.59677419354838712,
 'f_train': 0.98106060606060608,
 'pred_time': 0.06000018119812012,
 'train_time': 0.1399998664855957}
```

Fig.14 Random Forest

- **Testing Accuracy Score** = *0.83* which is better than decision tree.
- **Testing F score** = *0.60* which need to be improved.
- **Training scores** = *0.98* which shows kind of *over-fitting*.
- **CV Score** = *0.74* which is better than decision tree.
- **Running time** < *0.2* secs which is faster than Ada Boost.

- **Feature Importance** top ratings add up to > 50% weights: 'vote_count', 'runtime', 'released_date', 'popularity', 'budget' and 'revenue'.
- **'vote_count'** is still the top-1 relevant feature.
- Another features are 'genre_Comedy', 'genre_Drame' and 'genre_Thriller' those add up to 60% movie genres in our data.

```python
# Create plot feature importnace of 'Random Forest'

rf_importances = rf.feature_importances_

feature_plot(rf_importances, X_train, 30)
```

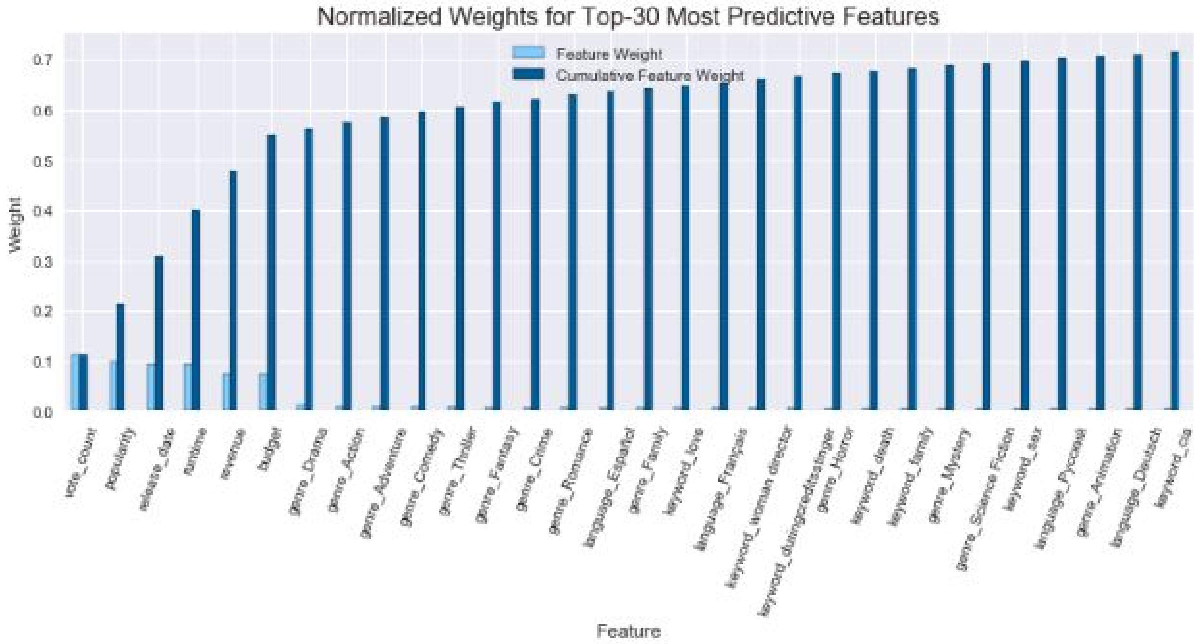


Fig.15 Feature importance - Random Forest

Gradient Boosting

It trains a sequence of model that gradually minimizes the loss function (inaccuracy of the predictions).

In detail steps: Y is target, M(x) is learning model, error_1 is the predicting inaccuracy.

1. Y = M(x) + error_1: Guess error_1 have same correlation with target value (Y). It adds a model to predict error_1
2. error_1 = G(x) + error_2: Another model G(x) is fitted to predict the target value (error_1)
3. Y = M(x) + G(x) + error_2: Combine these process, it can get a better accuracy by minimizing the loss function

```python
# Training and Evaluating results of 'Gradient Boosting'
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier()

results['Gradient Boosting'] = train_predict(gb, X_train, y_train, X_test, y_test)

pp.pprint(results['Gradient Boosting'])
```

```
{'acc_test': 0.8410732714138287,
 'acc_train': 0.89371124889282549,
 'cv_score': 0.74586351978329157,
 'f_test': 0.64086294416243639,
 'f_train': 0.83255159474671669,
 'pred_time': 0.021999835968017578,
 'train_time': 1.9300000667572021}
```

Fig.16 Gradient Boosting

- **Testing Accuracy Score** = *0.84* which is better than decision tree.
- **Testing F score** = *0.64* which is slightly improved.
- **Training scores** = *0.89* which is slightly *over-fitting*.
- **CV Score** = *0.75* which is better than Ada Boost.
- **Running time** < *2* secs, predicting time is much faster than the training time.

```
# Create plot feature importnace of 'Gradient Boosting'
gb_importances = gb.feature_importances_

feature_plot(gb_importances, X_train, 30)
```

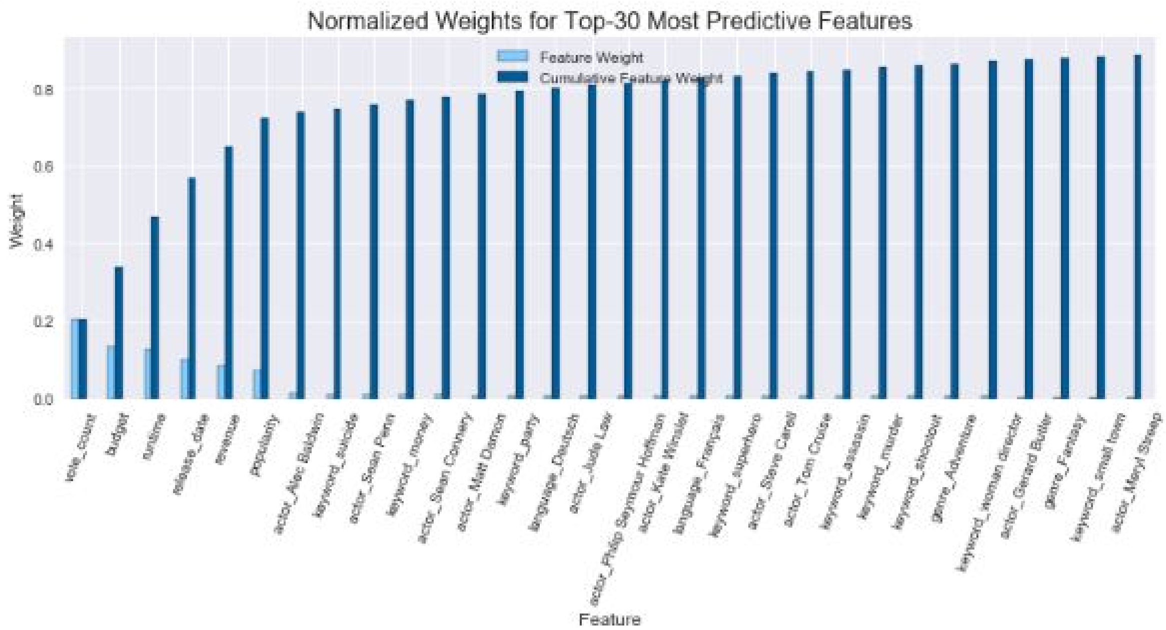


Fig.17 Feature importance - Gradient Boosting

- **Feature Importance** top ratings add up to > 70% weights: 'vote_count', 'budget', 'runtime', 'released_date', 'revenue' and 'popularity'.
- **'vote_count'** is still the top-1 relevant feature.
- Another features are 'actor_Alec Baldwin', 'actor_Sean Penn'. The actors have acting in movies with voting average in both > 7.0 and < 7.0.

XGBoost

The data handling resembles to Gradient Boosting. Additionally, the Regularization formalization is used to control the complexity and avoid over-fitting.

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Fig.18 XGBoost complexity[8]

[8] http://xgboost.readthedocs.io/en/latest/model.html

XGBoost Regularization

- T is the number of leaves in a boosting tree
- w is the score for each leaves
- The more leaves we have, the more free parameters we have. Thus the large the weights are, the more complex the model is.
- It penalizes the more number and the weights of leaves since the model is too complex.

```
# Refer to the code path
import os
mingw_path = 'C:\\Program Files\\mingw-w64\\x86_64-7.2.0-posix-seh-rt_v5-rev1\\mingw64\\bin'
os.environ['PATH'] = mingw_path + ';' + os.environ['PATH']

# Training and Evaluating results of 'XGBoost'
import xgboost as xgb
xgbm = xgb.XGBClassifier()

results['XGBoost'] = train_predict(xgbm, X_train, y_train, X_test, y_test)

pp.pprint(results['XGBoost'])
```

```
{'acc_test': 0.85036119711042313,
 'acc_train': 0.88263950398582813,
 'cv_score': 0.76167142157286527,
 'f_test': 0.6696428571428571,
 'f_train': 0.80910852713178294,
 'pred_time': 0.078000068664550078,
 'train_time': 1.5649998188018799}
```

Fig.19 XGBoost

- **Testing Accuracy Score** = *0.85* which is the best.
- **Testing F score** = *0.67* which is the best.
- **CV Score** = *0.76* which is the best.
- **Training scores** = *0.88* which is *no over-fitting*.
- **Running time** < *2.5* secs, predicting time is much faster than the training time.

- **Feature Importance** top ratings add up to >80% weights: 'vote_count', 'budget', 'runtime', 'released_date', 'revenue' and 'popularity'.
- **'vote_count'** is still the top-1 relevant feature.
- Another features are `'actor_Alec Baldwin', 'language_Español', 'language_Français', 'language_Deutsch'.

```python
# Create plot feature importnace of 'XGBoost'
xgbm_importances = xgbm.feature_importances_

feature_plot(xgbm_importances, X_train, 30)
```

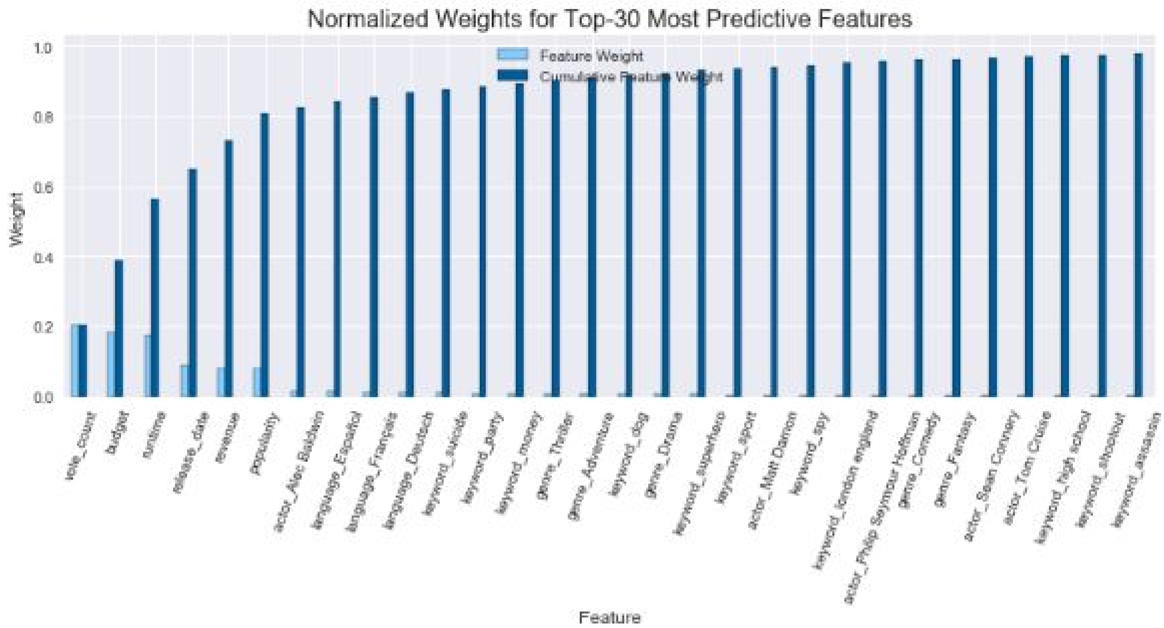


Fig.20 Feature importance - XGBoost

Refinement

We choose the 2 best models 'XGBoost' and 'Gradient Boosting' to apply the refinement methods. The 'feature selection' is used for dimensionality reduction by training the most relevant features. The model parameters fine tuning are then applied to search for the best combination for the movie data sets.

Feature Selection

Since we have 307 transferred features, we want to reduce the dimensionality for runtime saving. We apply feature selection to the movie data sets and choosing the most relevant features from 1 to 30 to see how many features can result to a best testing score.

XGBoost best features selected top 8: 'vote_count', 'budget', 'runtime', 'released_date', 'revenue', 'popularity, actor_Alec Baldwin' and 'language_Español'

```python
# feature selection for XGBoost
select_X_train, select_X_test = feature_select(xgbm, X_train, y_train, X_test, y_test, 'xgb')
```

```
(0.014729951, 8, 85.758513931888544)
```

Fig.21 Feature selection - XGBoost

Gradient Boosting best features selected top 6: 'vote_count', 'budget', 'runtime', 'released_date', 'revenue' and 'popularity'.

```
# feature selection for Gradient Boosting
select_X_train, select_X_test = feature_select(gb, X_train, y_train, X_test, y_test, 'gb')
```

```
(0.071319539796969575, 6, 85.242518059855527)
```

Fig.21 Feature selection - Gradient Boosting

Parameters Fine Tuning

We use GridSerachCV to evaluate the best parameters combination.

According to [xgboost](http://xgboost.readthedocs.io/en/latest/parameter.html)[9] and [gradient boosting](https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/)[10] parameters setting, we choose some items for fine tuning to avoid overfitting:

- **min_child_weight** [default=1] If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. The larger min_child_weight will prevent the training from **over-fitting** while too high values can lead to under-fitting.
- **max_depth** [default=6] Maximum depth of a tree, increase this value will make the model more complex and likely to be **over-fitting**.
- **subsample** [default=1] Subsample ratio of the training instance. It randomly collected a ratio of the data instances to grow trees and this will prevent **over-fitting**.
- **min_samples_leaf** [Gradient Boosting] Defines the minimum samples required in a terminal node or leaf. Used to control **over-fitting**.

Initial solution - XGBoost

```
Initial solution:
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1)
```

Final solution - XGBoost

```
parameters = {'min_child_weight':[2,3,4,5], 'max_depth':[1,2,3,4], 'subsample':[0.5,0.6,0.7,0.8]}

scorer = make_scorer(fbeta_score, beta=0.5)

# parameters tuning for XGBoost
best_xgbm = paramFineTune(select_X_train, y_train, select_xgbm, parameters, scorer)
```

```
Final solution:
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=3, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=0.8)
time: 19.0889999866
```

- **max_depth: 3 => 3** Too high value will result to over-fitting.

---

[9] http://xgboost.readthedocs.io/en/latest/parameter.html

[10] https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/

- **min_child_weight: 1 => 3** Larger value will prevent over-fitting.
- **subsample: 1 => 0.8** It randomly collected ratio of data instances to grow trees to prevent overfitting.

```
# evaluating refinement for XGBoost
grid_results['XGB_Best'] = train_predict(best_xgbm, select_X_train, y_train, select_X_test, y_test)

pp.pprint(grid_results['XGB_Best'])
```

```
{'acc_test': 0.85758513931888547,
 'acc_train': 0.88131089459698853,
 'cv_score': 0.76446211924728391,
 'f_test': 0.68840579710144933,
 'f_train': 0.7990654205607477,
 'pred_time': 0.019999980926513672,
 'train_time': 0.4010000228881836}
```

- **Testing Accuracy Score** = *0.86* which is better than original *0.85*.
- **Testing F score** = *0.69* which is better than original *0.67*.
- **CV Score** = *0.76* which is similar to original.
- **Training scores** = *0.88* which is similar to original.
- **Running time** < *0.4* secs which is much faster than the original *2 secs* because of **feature selection**.

Initial solution - Gradient Boosting

```
Initial solution:
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_split=1e-07, min_samples_leaf=1,
              min_samples_split=2, min_weight_fraction_leaf=0.0,
              n_estimators=100, presort='auto', random_state=None,
              subsample=1.0, verbose=0, warm_start=False)
```

Final solution - Gradient Boosting

```
parameters = {'min_samples_leaf':[2,3,4,5], 'max_depth':[1,2,3,4], 'subsample':[0.5,0.6,0.7,0.8]}

scorer = make_scorer(fbeta_score, beta=0.5)

# parameters tuning for Gradient Boosting
best_gb = paramFineTune(select_X_train, y_train, select_gb, parameters, scorer)
```

```
Final solution:
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_split=1e-07, min_samples_leaf=3,
              min_samples_split=2, min_weight_fraction_leaf=0.0,
              n_estimators=100, presort='auto', random_state=None,
              subsample=0.7, verbose=0, warm_start=False)
time: 35.486000061
```

- **max_depth: 3 => 3** Too high value will result to over-fitting.
- **min_samples_leaf: 1 => 2** Larger value will prevent over-fitting.
- **subsample: 1 => 0.6** It randomly collected ratio of data instances to grow trees to prevent over-fitting.

```python
# evaluating refinement for Gradient Boosting
grid_results['GB_Best'] = train_predict(best_gb, select_X_train, y_train, select_X_test, y_test)
pp.pprint(grid_results['GB_Best'])
```

```
{'acc_test': 0.84623323013415896,
 'acc_train': 0.88928255093002662,
 'cv_score': 0.76291173165038462,
 'f_test': 0.65563725490196079,
 'f_train': 0.8179297597042513,
 'pred_time': 0.0,
 'train_time': 0.33100008964538574}
```

- **Testing Accuracy Score** = *0.85* which is better than original *0.84*.
- **Testing F score** = *0.66* which is better than original *0.64*.
- **CV Score** = *0.76* which is better than original *0.75*.
- **Training scores** = *0.89* which is similar to the origin.
- **Running time** < *0.4* secs which is much faster than the original *2 secs* because of **feature selection**.

IV. Results

Model Evaluation and Validation

'XGBoost' and 'Gradient Boosting' both perform well on the movie data. After refinement, the feature selection can effectively reduce the training time without losing the testing accuracy and the parameter chosen can help the original model to prevent overfitting and slightly improve the testing accuracy score.

Feature Selection makes the original 307 features to shrink to 6~8 features in the models. We use the accuracy score to evaluate the best feature combination. Also, we can see from the feature importance that the top 6 features has composed of > 70% total weight. Re-running the model after refinement, we can see the training time and the prediction time both improved largely because we choose only the best important features to train.

Parameters Fine Tuning focus on not to be over-fitting to the data. By using 'GrisSearchCV', we gave a series of candidates to train and the result shows that compared to the initial solution, the parameters increased or decreased to avoid over-fitting and slightly improve the final result.

Cross-Validation is utilized by 3-fold validation (cv-score) to improve the data robustness by each time selecting different testing set while the rest are the training sets from the input data.

Justification

- **XGBoost** has the best testing accuracy score and fbeta score in the first run learning models. Comparing to the benchmark 'Decision Tree' that has the testing accuracy of 0.80, 'XGBoost' is much better of 0.85. According to the f-beta score, we have the 'XGBoost' of 0.67 and 'Decision Tree' of 0.55. However, the training of 'XGBoost' is 2.1 sec which is not good comparing to the 'Decision Tree' is 0.08 sec.
- **XGBoost after refinement** has the best result in testing accuracy(0.857) which is better than the 'Gradient Boosting'(0.846) after refinement and the training time of 'XGBoost

Best'(0.40s) is much better than 'XGBoost'(2.14s). The great improvement of running time in 'XGBoost' is because of the feature selection without losing the testing accuracy.

- **XGBoost** has the best result in cv-score, it shows the model robustness to the change of input data.
- According to the accuracy score, running time performance and the robustness to data, I would choose **XGBoost** to be the best model for the movie data.

Fig.22 Model Evaluation

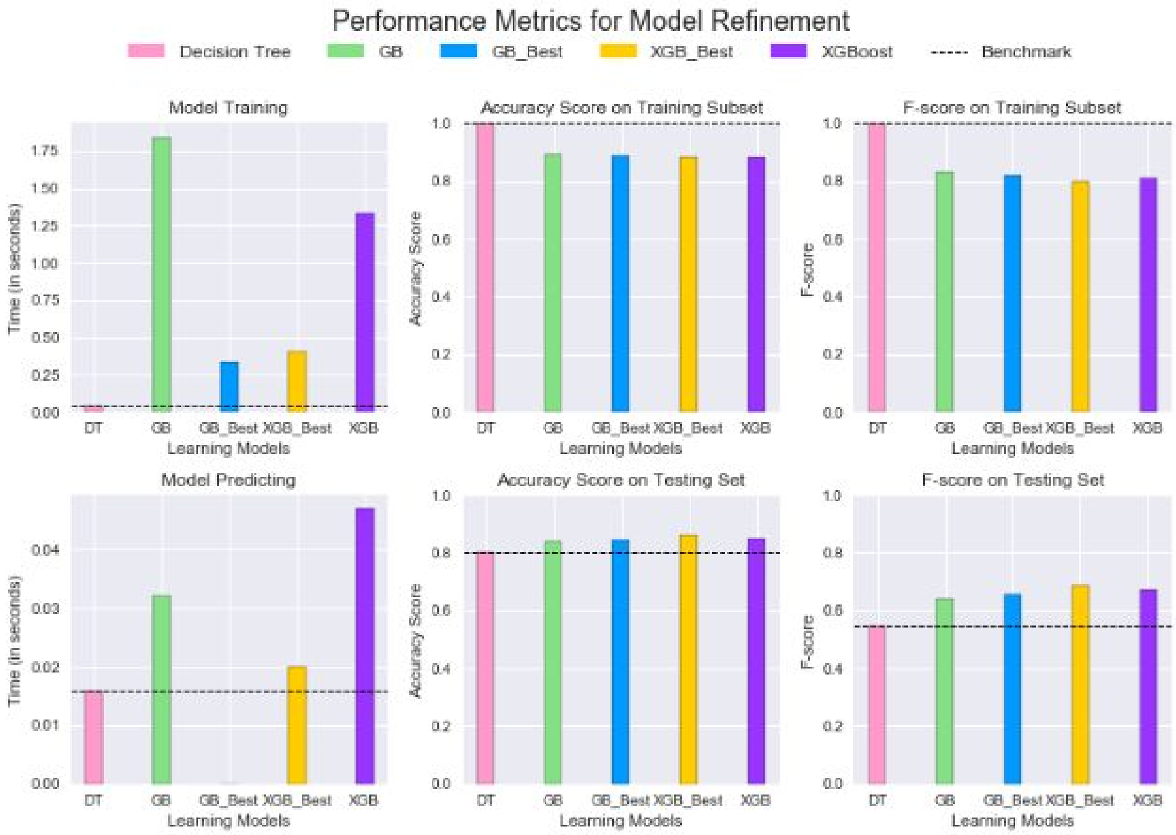


Fig.23 Refinement Model Evaluation

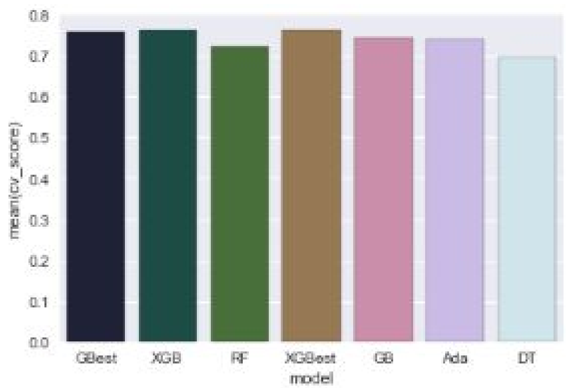


Fig.24 Model Evaluation - Cross Validation score

V. Conclusion

Free-Form Visualization

- According to **feature importance** in the previous ensemble methods, they all show that the numerical features - 'vote_count', 'budget', 'runtime', 'released_date', 'revenue' and 'popularity' are the best relevant features. The top 1 related feature is always be 'vote_count', that is, if there are more voting for the movie, the rating is more likely to be higher.

- We choose the best model 'XGBoost' to show the feature importance. Combined with the **feature selection**, we indicate the top 8 relevant features which get the best score - 'vote_count', 'budget', 'runtime', 'released_date', 'revenue', 'popularity', 'actor_Alec Baldwin' and 'language_Español'. The top 6 features are numerical features that each movie data should have a reasonable value in it after we drop the missing value data. I think the other json format features are not as important as those numericals because they are not valid in each movie. The feature actor and language are valid in movie data with voting average in both > 7.0 and < 7.0.
- The cumulative feature weights is added up for each feature weight rating before it in model 'XGBoost'.(ex. Top-2 cumulative feature weights are Top-1 weights + Top-2 weights) It shows that the top-8 features add up to > 80% feature importance in total 307 features. It is quite enough to select only these 8 features rather than more to be the training sets.

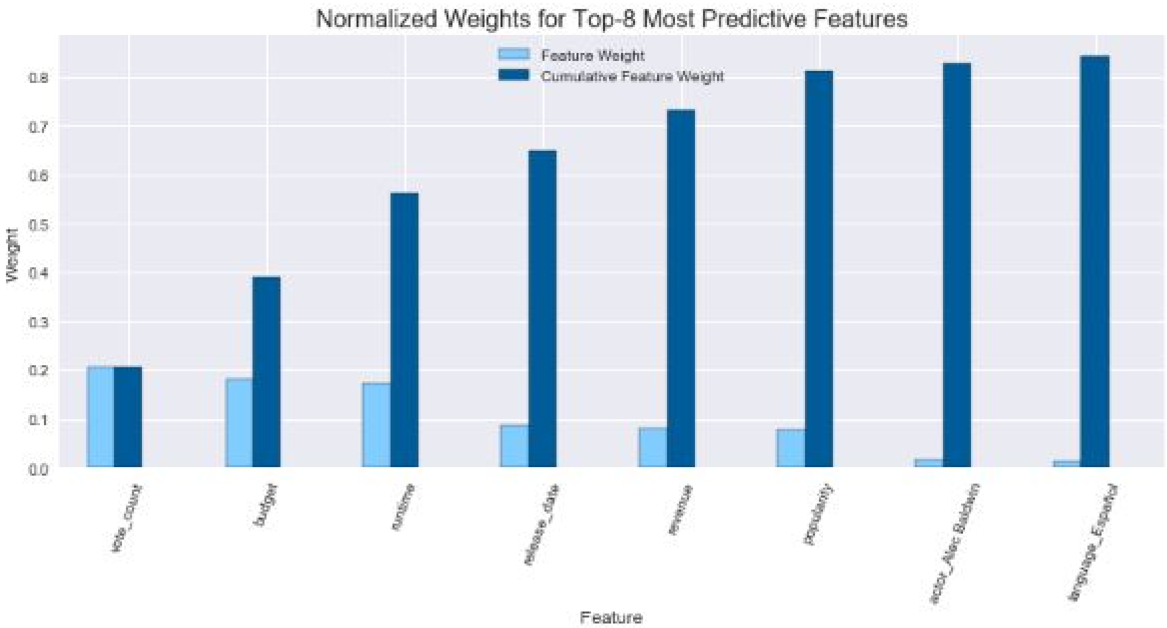


Fig.25 Feature Importance - XGBoost

Reflection

In this project, we use the **Supervised Learning** to predict the movie data voting average and find which features result into a higher rating. We have do some data preprocessing such as 'drop outliers' and 'feature scaling' before the model training to keep the useful information and normalizing the data to be easy for training. After training, we also adopt some methods such as 'feature selection' and 'GridSearchCV' to do model refinement. The target of training generated a best model - 'XGBoost' to predict the movie data in high accuracy.

- It seems that the movie cast like actors or directors do not show big influence on movie voting. Though we sometimes think that certain actors or directors may create good movies. I think that maybe the feature sample is not as much as enough to learn.

- Some relevant features like revenue or vote_count are information we only collect after the movie was made while that may be the reference to make the sequels.
- This movie data includes lots of json features to deal with, and it will make a large number of features if we flatten it. Some feature selection methods are needed to apply on it.
- The result accuracy which I expected to be better, but it's ok. The famous movie database online like IMDB can also used this model to predict while the input data need to be well prepared.
- About the data collected problem is that the old movie data is much less than the movie made in recent years. Other data features may be considered like awards, metascore, writers, certification ...

Improvement

We want to introduce another method: **Unsupervised Learning** to look into the data information. We can use **PCA** to apply feature transformation on data. The data will be transformed into different clusters that best describe the movie data and we can use the unsupervised learning such as 'KMeans' or 'GaussianMixture' to classify them into the clusters representing the good movie group or general movie group.

Feature Transformation - PCA

- **PCA with 307 features**: The full features sets did not classify the clusters well. The best PC only got 0.082 scores.
- **PCA with 6 features**: Instead, we choose the outstanding features: 'budget', 'popularity', 'revenue', 'runtime', 'vote_count', 'release_date' to apply PCA. The results show 2 PCs add to about 0.81 that covers a great portion of feature distribution.
- **1st PC = 0.62** with all the 6 features are negative-weighted. It may indicate a cluster with bad popularity, revenue, ... Compared to the data, we guess that it represents the vote < 7.0.
- **2nd PC = 0.19** with 5 features appears in positive-weighted. We guess the clusters represents the vote >= 7.0. The key why they are voting high has many reasons like high revenue, high popularity or high vote count.

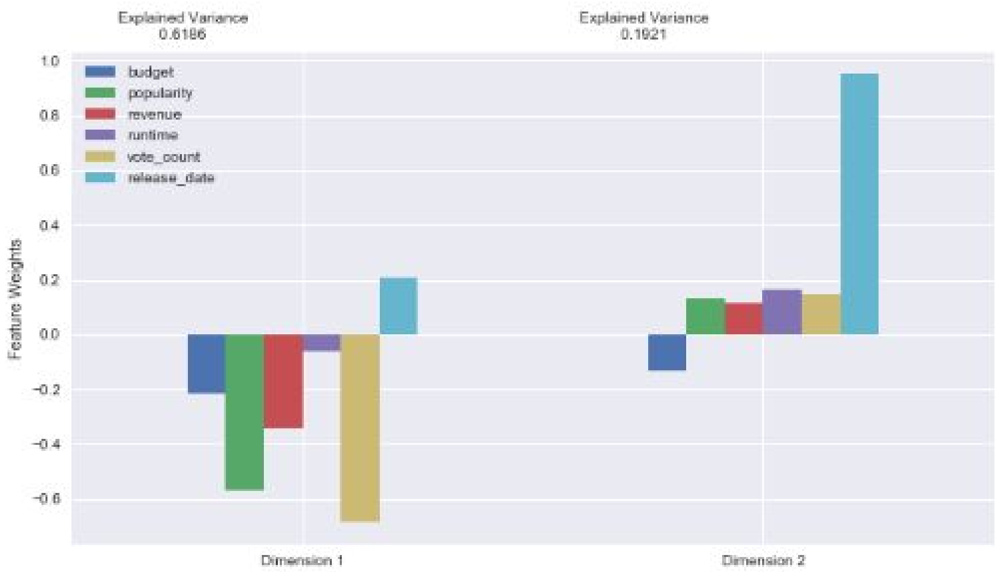


Fig.26 Feature Transformation - PCA