Machine Learning Engineer Nanodegree

Capstone Project

Bjarki Mar Stefansson

August 17th, 2018

I. Definition

Project Overview

A model to predict movie ratings on movies made in USA and are in English as well as in Color, these data are from a large data set that spans many years of data and was obtained from kaggle. This kind of model could be used by movie companies to predict the movie rating if they have a certain amount of data before hand and want to produce a movie. A similar approach has been done before for example in this article, a gentleman called Vladimir uses simple Linear Regression to predict a movie rating from a large dataset but only a few features. As well as in this article printed in *International Conference on Rough Sets and Knowledge Technology,* where the authors developed user rating prediction models which used classification technique (linear combination, multiple linear regression, neural networks).

Problem Statement

The problem to be solved is to see if a movie score can be predicted. When movie production companies are developing a movie they highly depend on good reaction from audience, however, movie score is therefore not the one concrete value to be successful, it can also be popular even though the score is low such as Transformers: revenge of the fallen that has only around 6.0 in score but had a revenue of roughly 830m USD with budget of 150m USD. If a model is created to compute these scores than movie companies can potentially predict their score with a set of features (director, actors and so on) that they are choosing from to maximise their score and their revenue.

First of some cleaning and preprocessing is needed on the data, potential missing values in the data, so that needs to be checked. The strategy is to implement several models so that we can compare between models which will compute the best score. The techniques that we will be implementing will be Linear SVC, SVC, Logistic regression and then tree techniques, first with a decision tree and then with random forest. From these classifiers, we will compute a set of score that will distinguish the performance of the model, by having five different techniques we can see what model gives the best performance and then fine-tune the best model to get event better performance.

Metrics

Each model will compute a F1 score also known as balanced F-score (harmonic mean of precision and recall) as a metric.. A naive approach will be used using linear model to define the benchmark of the initial F1 score so that can be used for rest of the models to get the f1 score higher. The F1 score is calculated as follows:

```
F1 = 2 * (precision * recall) / (precision + recall)
```

F1 is chosen as the dataset are imbalanced and F1 score is one of the metrics that are recommended to use on imbalanced datasets such as mentioned in this forum.

II. Analysis

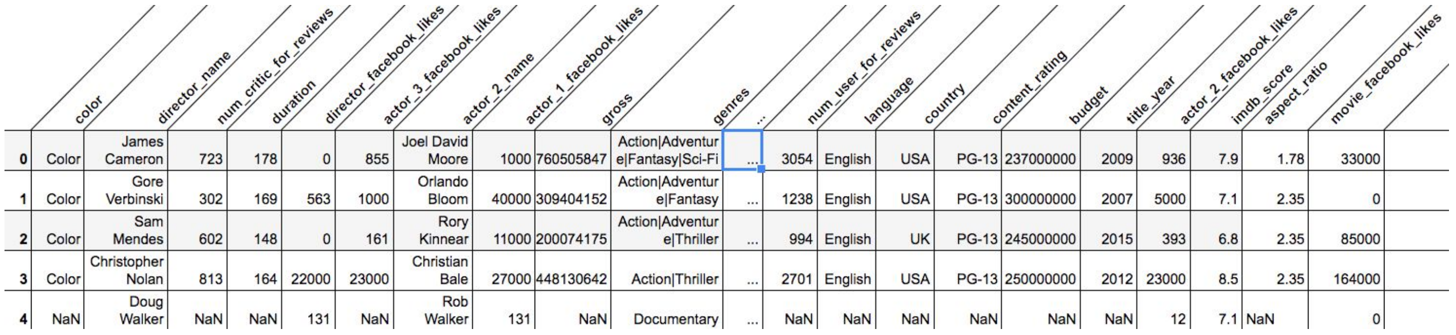
Data Exploration

The data was obtained from kaggle which is about 5000 movies. The data consists of 28 columns:

Color	**String**: If the movie is in color or not
Director_name	**String**: Name of the director of the movie
Num_critic_for_reviews	**Integer**: Number of critics for the movie
Duration	**Integer** : Duration of the movie in minutes
Director_facebook_likes	**Integer**: # of Facebook likes that the director has
Actor_3_facebook_likes	**Integer:** # of Facebook likes that the movie actor nr 3 has
Actor_2_name	**String**: Name of movie actor nr 2
Actor_1_facebook_likes	**Integer**: # Facebook likes that the movie actor nr 1 has
Gross	**Integer**: Gross revenue of the movie in USD
Genres	**String**: Types of genres that the movie has
Actor_1_name	**String**: Name of movie actor nr 1
Movie_title	**String**: Title of the movie
Num_voted_users	**Integer**: Number of votes that the movie has
Cast_total_facebook_likes	**Integer**: Total number of Facebook likes that the cast has
Actor_3_name	**String**: Name of movie actor nr 3

- **Facenumber_in_poster** **Integer**: What actors are on the poster (displayed as the number of the actor 1, 2 or 3)
- **Plot_keywords** **String**: Plot keywords, several words describing the plot i.e. avatar|future|marine|native|paraplegic
- **Movie_imdb_link** **String**: IMDb link to the movie
- **Num_user_for_reviews** **Integer**: # of users that wrote a review about the movie
- **Language** **String**: Language of the movie
- **Country** **String**: Country of the movie
- **Content_rating** **String**: What the ratings is, i.e PG-13
- **Budget** **Integer**: Budget that the movie had for production
- **Title_year** **Integer**: Year that the movie was released
- **Actor_2_facebook_likes** **Integer**: # of Facebook likes that the movie actor nr 2 has
- **Imdb_score** **Float**: Movie score (0.0-10.0
- **Aspect_ratio** **Float**: The aspect ratio of an image describes the proportional relationship between its width and its height
- **Movie_facebook_likes** **Integer**: Number of Facebook likes the movie has

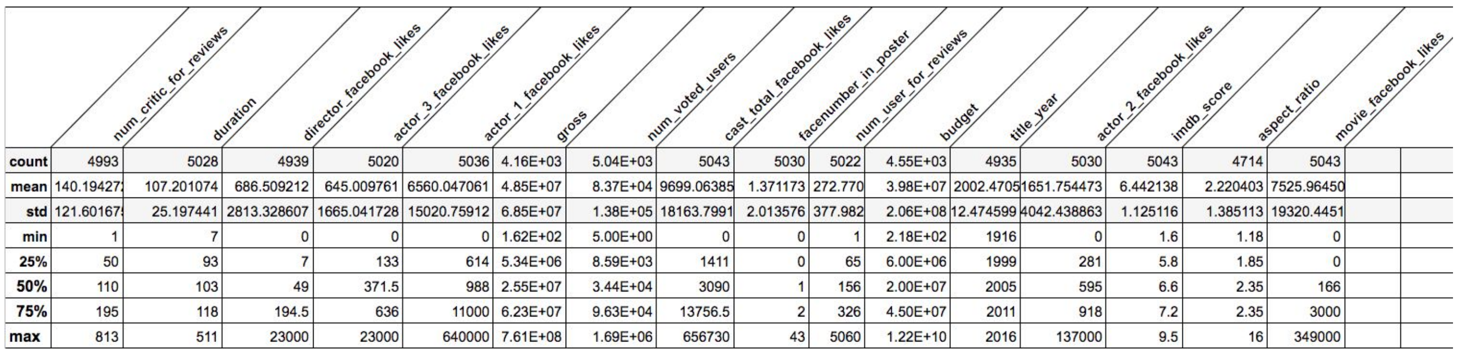
Here below is a data sample from the data as well as statistics of the data, there are missing columns in both of these data samples as it is 28 columns, this is the output that is generated using 'display' function, however, this does describe the data well enough to see in that there is a need for analyzing the data further. Nearly all numerical columns are integers except



	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_facebook_likes	gross	genres	...	num_user_for_reviews	language	country	content_rating	budget	title_year	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
0	Color	James Cameron	723	178	0	855	Joel David Moore	1000	760505847	Action|Adventure|Fantasy|Sci-Fi	...	3054	English	USA	PG-13	237000000	2009	936	7.9	1.78	33000
1	Color	Gore Verbinski	302	169	563	1000	Orlando Bloom	40000	309404152	Action|Adventure|Fantasy	...	1238	English	USA	PG-13	300000000	2007	5000	7.1	2.35	0
2	Color	Sam Mendes	602	148	0	161	Rory Kinnear	11000	200074175	Action|Adventure|Thriller	...	994	English	UK	PG-13	245000000	2015	393	6.8	2.35	85000
3	Color	Christopher Nolan	813	164	22000	23000	Christian Bale	27000	448130642	Action|Thriller	...	2701	English	USA	PG-13	250000000	2012	23000	8.5	2.35	164000
4	NaN	Doug Walker	NaN	NaN	131	NaN	Rob Walker	131	NaN	Documentary	...	NaN	NaN	NaN	NaN	NaN	NaN	12	7.1	NaN	0

Image 1: data statistics

As you can see in the Image 1, with movie id 4, there are many missing values in that row, these kinds of rows can potentially be many, so there is a need for analyzing missing values further. Genres column have many genres in the same column, it is stored as a string with '|' as a separator, to be able to utilize the genres type, a preprocessing is needed to be able to use that for the models.



	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_1_facebook_likes	gross	num_voted_users	cast_total_facebook_likes	facenumber_in_poster	num_user_for_reviews	budget	title_year	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes		
count	4993	5028	4939	5020	5036	4.16E+03	5.04E+03	5043	5030	5022	4.55E+03	4935	5030	5043	4714	5043		
mean	140.19427	107.201074	686.509212	645.009761	6560.047061	4.85E+07	8.37E+04	9699.06385	1.371173	272.770	3.98E+07	2002.4705	1651.754473	6.442138	2.220403	7525.96450		
std	121.60167	25.197441	2813.328607	1665.041728	15020.75912	6.85E+07	1.38E+05	18163.7991	2.013576	377.982	2.06E+08	12.474599	4042.438863	1.125116	1.385113	19320.4451		
min	1	7	0	0	0	1.62E+02	5.00E+00	0	0	1	2.18E+02	1916	0	1.6	1.18	0		
25%	50	93	7	133	614	5.34E+06	8.59E+03	1411	0	65	6.00E+06	1999	281	5.8	1.85	0		
50%	110	103	49	371.5	988	2.55E+07	3.44E+04	3090	1	156	2.00E+07	2005	595	6.6	2.35	166		
75%	195	118	194.5	636	11000	6.23E+07	9.63E+04	13756.5	2	326	4.50E+07	2011	918	7.2	2.35	3000		
max	813	511	23000	23000	640000	7.61E+08	1.69E+06	656730	43	5060	1.22E+10	2016	137000	9.5	16	349000		

Image 2: data statistics

There are 5043 movies in the dataset, only 3 columns contain the correct amount of values, that are cast_total_facebook_likes, movie_facebook_likes, imdb_scores rest of them have missing values. If we take a brief look at the statistics, we have an average imdb_score of 6.44 and average duration of 107.2 minutes, according to Screen Actor Guilds, they consider movies to be 80 minutes or longer, and from those statistics our average data is well above that, however, minimum duration in the data is 7 minutes, this needs to be analyzed further more.

Exploratory Visualization

The following two images consists of the major features, as it was that many features, the scatter plot was divided into two images, one that contained non-social media features while the other scatter plot is strictly social media features along with imdb score, there are many features and many data points, but just by looking at these there is some correlation.

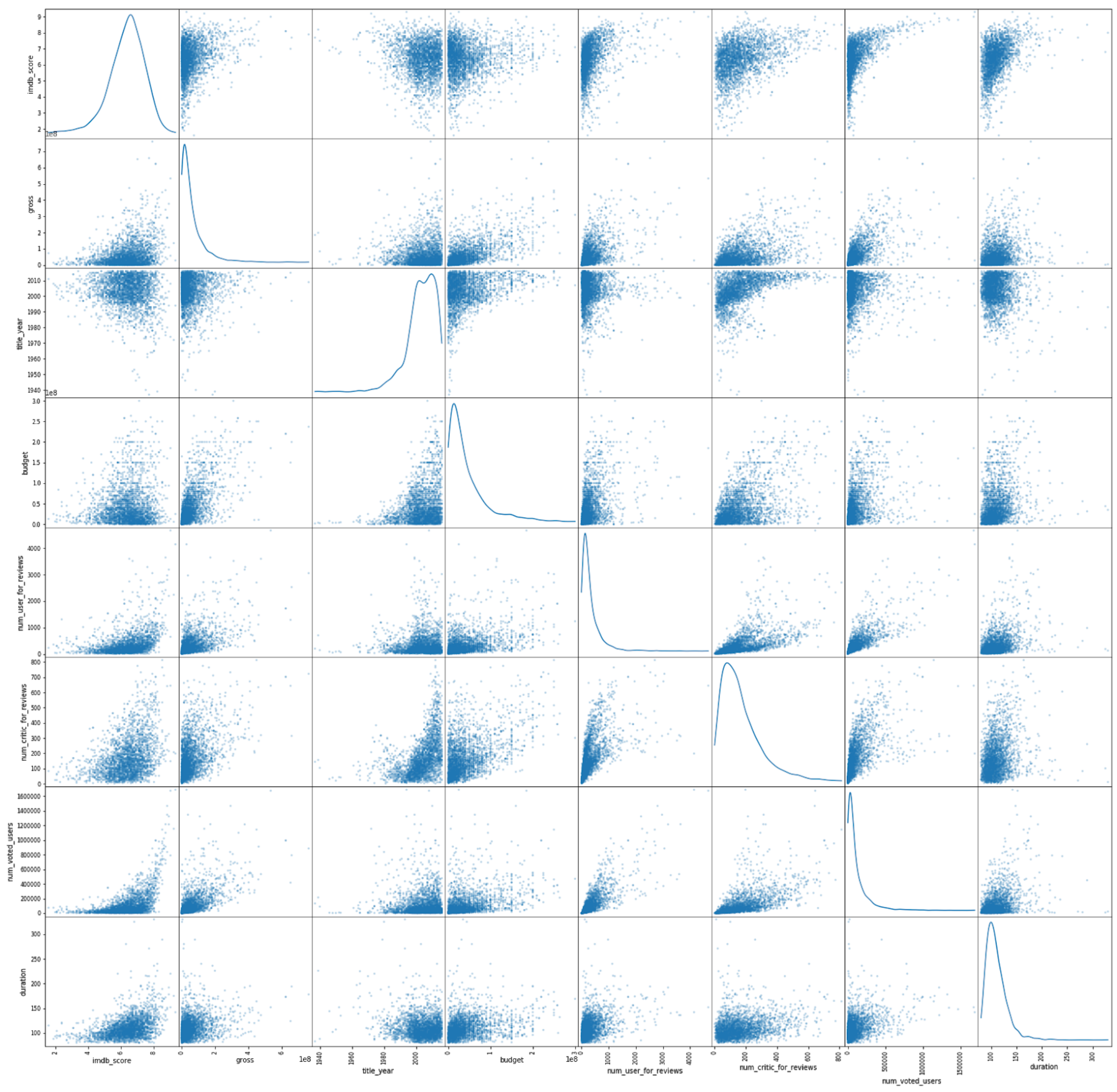


Image 3: Scatter plot

When looking at the scatter plot in image 3, and going through imdb_score column it is noticeable that there is some correlation between IMDB score and a number of critics, votes, user reviews, gross and budget, as we are trying to predict the IMDB score, then this would be the starting point implementing a prediction model. There are of course outliers here and there but there seems to be no single one, there is often few of them together, e.g. column gross and row num_voted_users there are outliers few outliers but they are close together. We imbalance classes such as title year, as we have many movies released after 2000 than before 2000 this does also happen in many of the other classes but it is most noticeable when going through the title year column, possible it is due to the fact that internet got more popular each year and therefore e.g. number of votes increases by year.

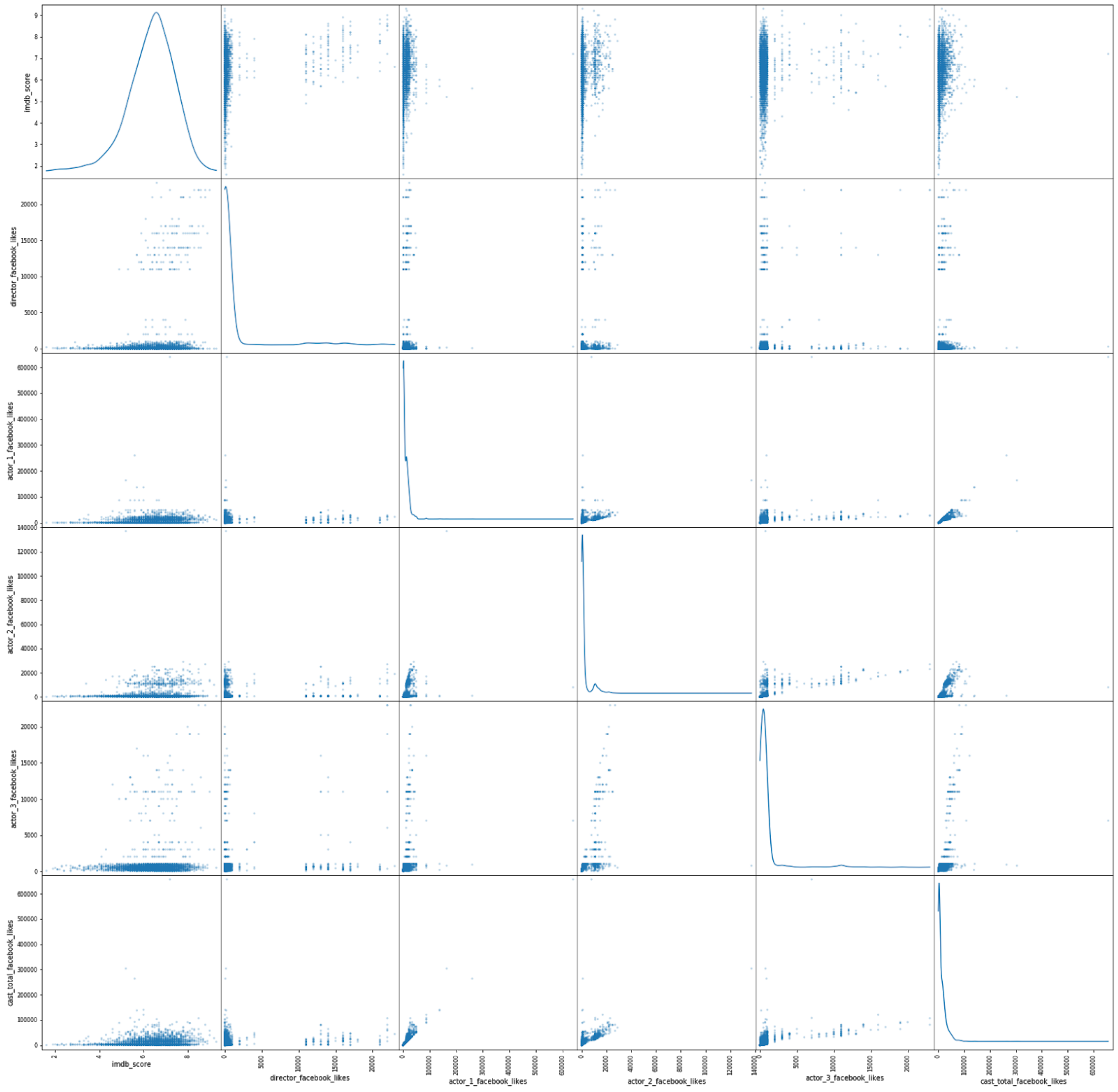


Image 4: Social media scatter plot

As you can see in image is also some correlation between imdb score and all the social media features in the first column. If we take a look at if there are any outliers here, then the most noticeable one is in column actor_3_facebook_likes, row cast_total_facebook_likes as the outlier as way above and long way from most of the other data points.

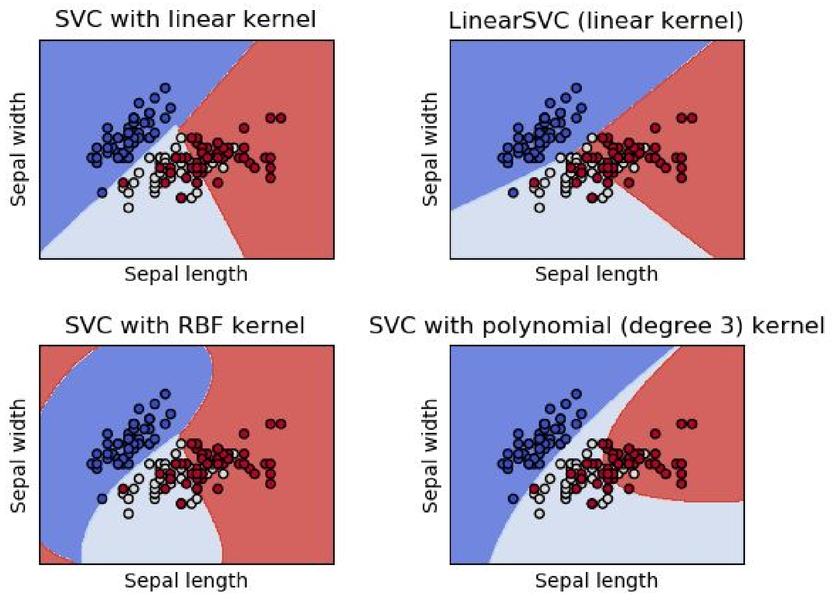
Algorithms and Techniques

There will be few learners used, these are supervised learners such as:

- **Linear SVM** and **SVM**
    - *Strengths:* it is useful for both hard margin as well as soft margins data, It is easy to visualise the decision boundary. works well on a smaller dataset and is also quite accurate. It uses a subset of training points and therefore efficient
    - *Weaknesses:* Isn't suited for larger datasets as the training time can be quite high. It is less effective on datasets which are more noisier and are with overlapping classes
    - *What makes this model a good candidate:* It has a decent amout of high accuracy and works well on smaller datasets and therefore this good be a great model for the problem
- **Logistic Regression**
    - *Strengths: the algorithm can be regularized to avoid overfitting. Can be updated easily with new data using stochastic gradient descent.*
    - *Weaknesses: Favours to underperform when there are many non-linear decision boundaries. Not flexible enough to easily capture more complex relationships.*
    - *What makes this model a good candidate:* The predictions are mapped between 0 and 1 and in that way it can be interpreted as class probabilities
- **Decision Tree** and **Random Forest**
    - *Strengths:* Data classification without much calculations, it produces a highly accurate classifier. Easy to use and easy to implement
    - *Weaknesses:* While the problem grows, the calculation grows exponential. It is prone to overfitting for some noisy datasets.
    - *What makes this model a good candidate:* As the problem does not have that many features or too much data, this model can be a great use on the problem as Decision Tree or Random Forest does not require that much domain knowledge and performs rather quickly

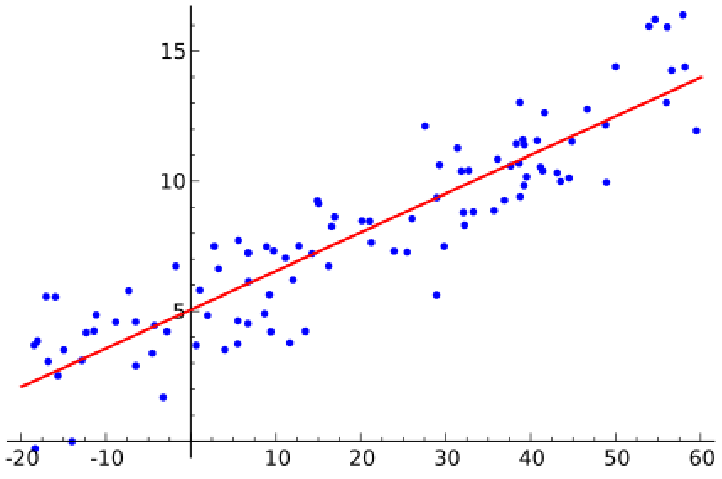
These 5 models are robust and are fairly straightforward in implementing and therefore a great choice for this project, they are chosen to be able compare the score between them.

If we go into more details as how these algoritms are trained, then in SVM the data his marked in one of the categories, an SVM training algorithm builds a model and categorises the sample into one category or the other one, for the SVM that we are using in this project are Linear SVM and then when we use with RBF kernel which can be visualized in image 5.



SVM visulation, image from sklearn

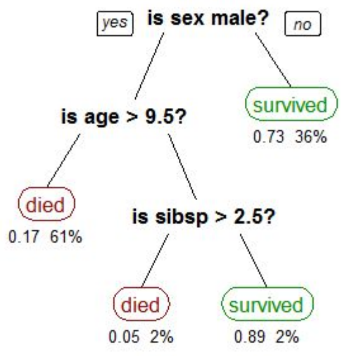
An SVM model is a description of the samples as data points in space, mapped so that the samples of the separate categories are separated by a gap that is as wide as possible. New samples are then mapped into that space and then predicted to a category based on which side of the gap they fall.



Logistic regression, image from wikipedia

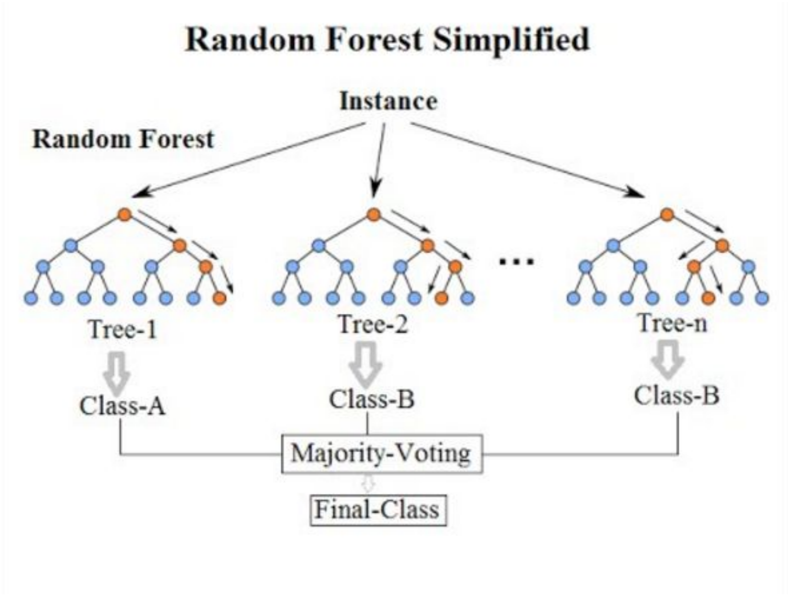
In logistic regression a probability output is computed for the given input point belongs to a certain class, if we think of a model with only two classes a probability is generated for that entry and categorized that data into set of category based on that probability, the premise of logistic regression is that an assumption is that the input space could be seperated into good regions for each class divided by a boundary between them similar as seen in image 6, the two catagories are divided by the red line.

For the decision tree it can be visualised as a flowchart-like structure in which each internal node represents a "test" on an attribute , each branch represents the outcome of the test, and each leaf node represents a class label. The paths from root to leaf represent classification rules.



Decision tree

The algorithm uses a decision tree similar to the image above as a predictive model to go from observations about an item that are represented in branches to the conclusion about the target value represented in the leaves.



Random forest, image gotten from this [article]()

The Random forest is similar to decision tree but is different in the way that they operate by constructing a number of decision trees at training and outputting the class that is the mode of the classification Random forest have a way of averaging the multiple generated deep decision trees that are trained on different parts of the same training set, that can be seen in the image above.

Linear SVM is chosen to act as an benchmark for the rest of models and will be using default values. SVM will also be using its default values.

In logistic regression the parameter range C is set from 0.001, 0.01, 0.1, 1 and 10, as we allow some regularization and give it a little bit of freedom otherwise we are using the default values.

In Decision Tree we use few *min_sample_leaf* options and we iterate through them, otherwise we use the default values.

In random forest we also have few *min_sample_leaf* options to iterate and find the best leaf option, *n_estimators* is set to 300, it was chosen to be high to ensure to get the best score. *Max_features* is set to 0.3, that was chosen after few iterations and fine tuning and considered great choice for this project, rest of the parameters are default values.

The data will be split into random train and test subsets using the *train_test_split* from there, the subsets will be passed on to the five models for predicting the score.

Benchmark

A naive approach will be used using Linear SVM as a benchmark F1 score, this F1 score will then be used to replicate or get a better F1 score using the other 4 models. The Linear SVM F1 score will be computed along with the other models. The benchmark value gotten is by using the Linear SVM:

```
Predicting movie score with all movies:
Training/Testing set has 2276/570 samples.
------
Linear SVM has the F1 score of: 0.3404
```

And the benchmark score will be 0.**3404**

III. Methodology

Data Preprocessing

From the beginning, the data contains roughly 5000 movies. However, this project is focusing on only movies from USA, in English and color and are 80 minutes or longer (according to Screen Actor Guilds). The data is therefore shrunk down to these movies and we end up with 3477 movies. Some of these movies do however have missing data, a function 'dropna' is used to remove all rows that contain any missing data, and the final count of movies is then 2846 movies. As mentioned before, the genres are one long string with several genres in one column, therefore a implementation of extracting all these genres and put into a separate column, created a column for each and one genre and adding a boolean value for those genres for each movie. Each movie has from one to five genres, so these values are different from movies. Similar is done with the content rating of the movies.

Before the models can use the data, the data needs to be encoded, and LabelEncoder is therefore chosen for the job. LabelEncoder is a utility class to help normalize labels such that they contain only values between 0 and n_classes-1.

MinMax scaling is also used on all the numerical values:

```
['num_critic_for_reviews',
'director_facebook_likes', 'actor_3_facebook_likes',
 'actor_1_facebook_likes', 'gross',
'num_voted_users', 'cast_total_facebook_likes',
'num_user_for_reviews', 'budget',
'actor_2_facebook_likes',
'Movie_facebook_likes']
```

The MinMax values is set to use the default values (0 and 1)

Now the data is ready for use in the models.

Implementation

All the implementation was done in the same file: 'predicting_movie_score.ipynb'. Where all the implementation is done: reading in the data, analysing, preprocessing and then implement algorithms models.

The implementation are in different functions, Encode data function, scale function, split function and then each and one supervised learner model is in a function:

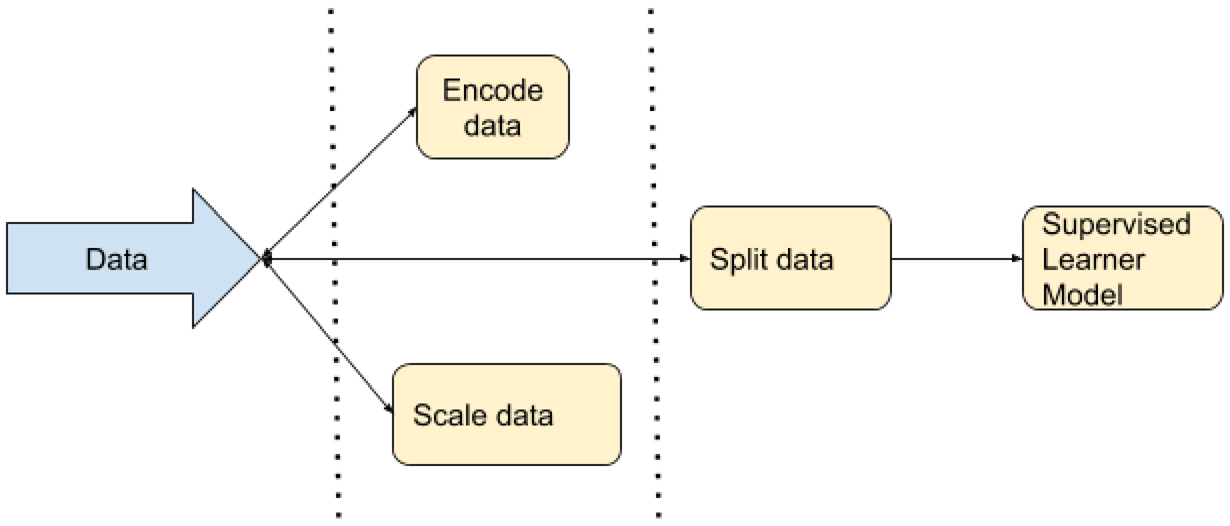


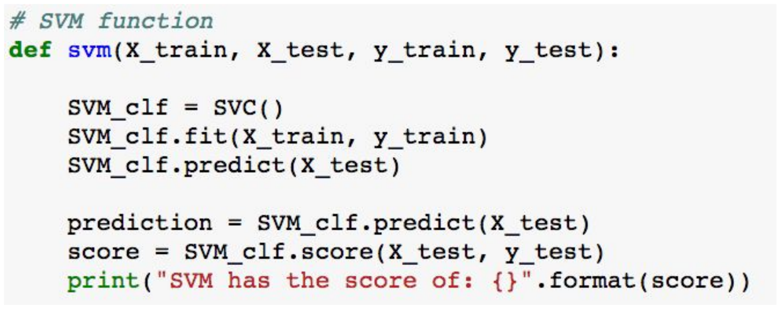
Image 5: flowchart of the data

First the data is encoded in function *dfToEncode(df_enc, y)* and than scaled in the function *scaling(dataX),* from there, the data is sent to split data function called *train_test(X,y),* where it is split into training and testing sets, in the same function, the data is forwarded to all the supervised learner models function:

- *linearSVM(X_train, X_test, y_train, y_test)*
- *svm(X_train, X_test, y_train, y_test)*
- *logsticRegression(X_train, X_test, y_train, y_test)*
- *decisionTree(X_train, X_test, y_train, y_test)*
- *randomForrest(X_train, X_test, y_train, y_test)*

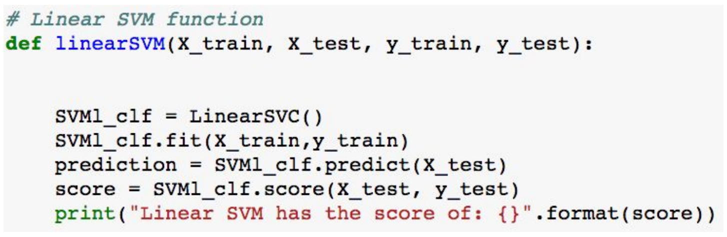
From the model functions, a F1 score is computed for that particular learner model, the F1 score function is used is provided from sklearn.metric module. All models were tested with default values and F1 score calculated and documented before any fine-tuning the parameters takes place, all the results between default values and parameters tuned models can be seen in the result section.

Following images are images of the learner models that were used, as mentioned before, each learner as it own function, when called it will fit the data and then compute the prediction score.



```python
# SVM function
def svm(X_train, X_test, y_train, y_test):

    SVM_clf = SVC()
    SVM_clf.fit(X_train, y_train)
    SVM_clf.predict(X_test)

    prediction = SVM_clf.predict(X_test)
    score = SVM_clf.score(X_test, y_test)
    print("SVM has the score of: {}".format(score))
```

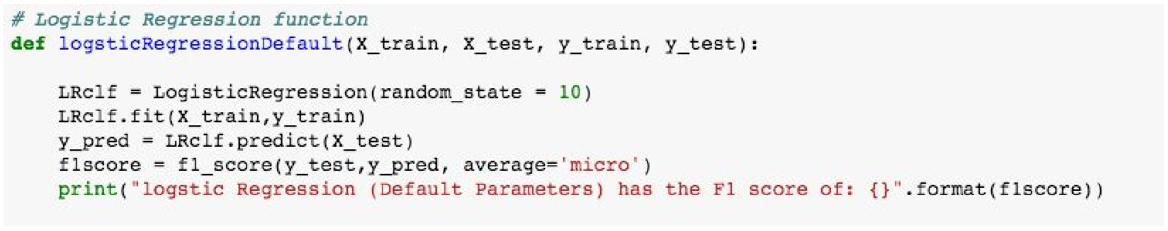
Image 6: SVM



```python
# Linear SVM function
def linearSVM(X_train, X_test, y_train, y_test):


    SVM1_clf = LinearSVC()
    SVM1_clf.fit(X_train,y_train)
    prediction = SVM1_clf.predict(X_test)
    score = SVM1_clf.score(X_test, y_test)
    print("Linear SVM has the score of: {}".format(score))
```

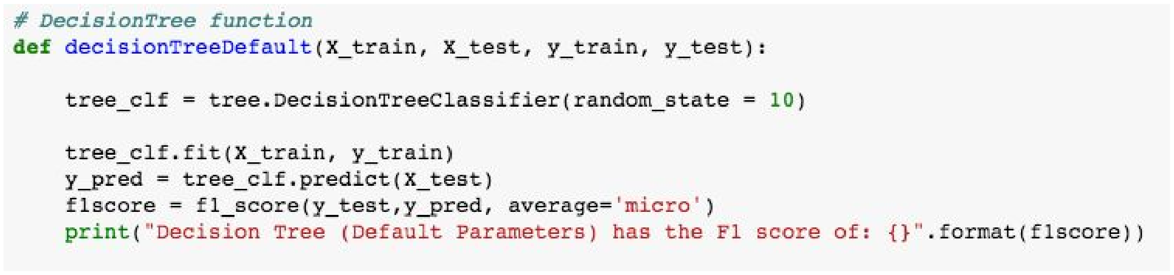
Image 7: Linear SVM

Here we have the SVM at image 6 and LinearSVM in image 7 with the default values in the parameters



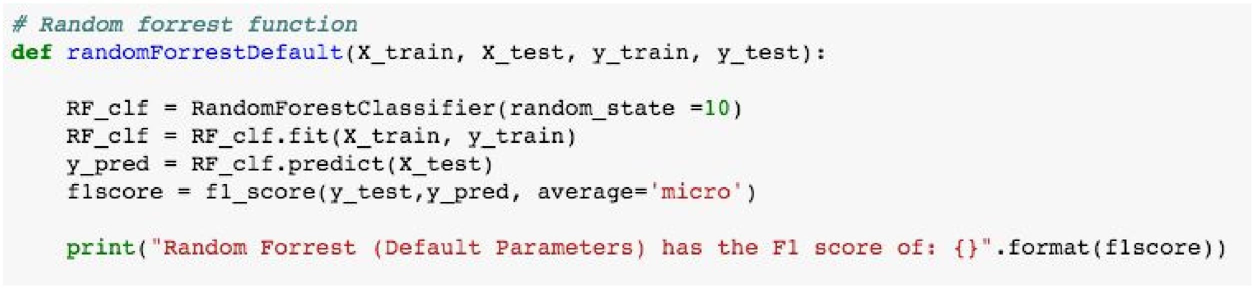
```python
# Logistic Regression function
def logsticRegressionDefault(X_train, X_test, y_train, y_test):

    LRclf = LogisticRegression(random_state = 10)
    LRclf.fit(X_train,y_train)
    y_pred = LRclf.predict(X_test)
    f1score = f1_score(y_test,y_pred, average='micro')
    print("logstic Regression (Default Parameters) has the F1 score of: {}".format(f1score))
```

Image 8: Logistic regression



```python
# DecisionTree function
def decisionTreeDefault(X_train, X_test, y_train, y_test):

    tree_clf = tree.DecisionTreeClassifier(random_state = 10)

    tree_clf.fit(X_train, y_train)
    y_pred = tree_clf.predict(X_test)
    f1score = f1_score(y_test,y_pred, average='micro')
    print("Decision Tree (Default Parameters) has the F1 score of: {}".format(f1score))
```

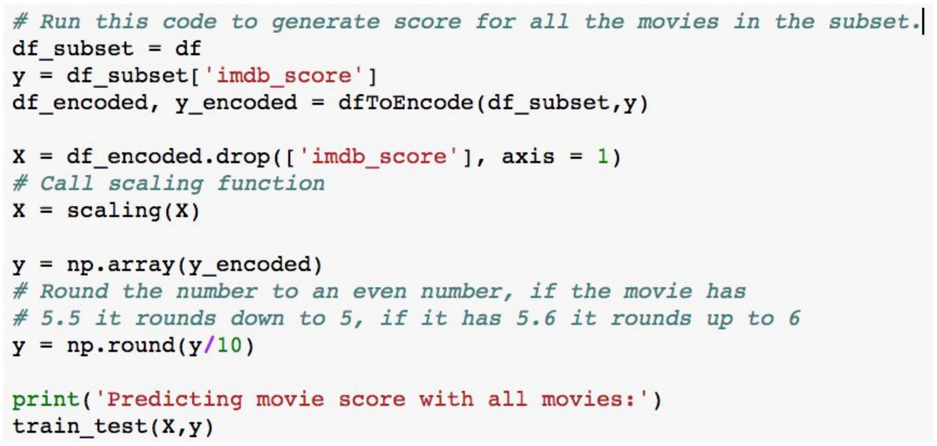
Image 9: Decision Tree



```python
# Random forrest function
def randomForrestDefault(X_train, X_test, y_train, y_test):

    RF_clf = RandomForestClassifier(random_state =10)
    RF_clf = RF_clf.fit(X_train, y_train)
    y_pred = RF_clf.predict(X_test)
    f1score = f1_score(y_test,y_pred, average='micro')

    print("Random Forrest (Default Parameters) has the F1 score of: {}".format(f1score))
```

Image 10: Random forest

As you can see from image 8, 9 and 10, these are all learners with their default values in parameters. This was the initial setup and therefore the benchmark of upcoming tuning for better results. Performance of these default learners can be seen in section IV Results.



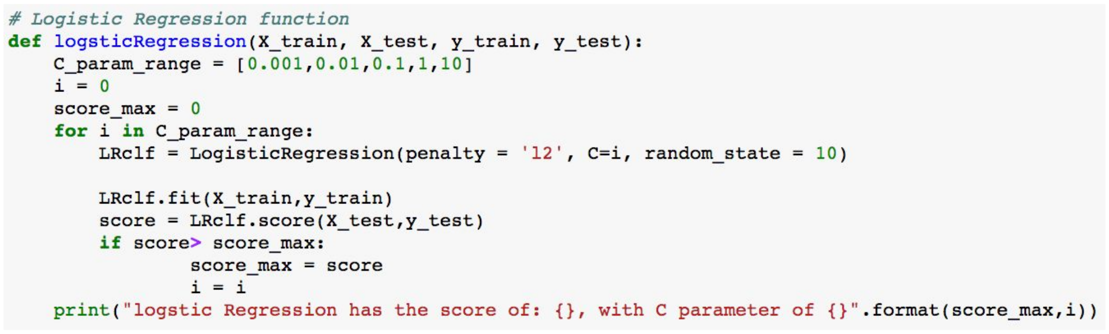
```
# Run this code to generate score for all the movies in the subset.
df_subset = df
y = df_subset['imdb_score']
df_encoded, y_encoded = dfToEncode(df_subset,y)

X = df_encoded.drop(['imdb_score'], axis = 1)
# Call scaling function
X = scaling(X)

y = np.array(y_encoded)
# Round the number to an even number, if the movie has
# 5.5 it rounds down to 5, if it has 5.6 it rounds up to 6
y = np.round(y/10)

print('Predicting movie score with all movies:')
train_test(X,y)
```

Image 11: Code to run initate the model.

This code in image 11 is the code that calls all the functions, here it can also create a different subset of the 'df' data, such as only choosing certain types of movies etc. This code is the initial solution of this project, the data is all movies made in the USA, in English, 80 minutes or longer and in Color, the amount of those movies are 2846.

When implementing this, the number one problem was encoding the data, at first, one-hot encoding was used but some problems came up that could be solved with that encoding, therefore LabelEncoder was used and finally, the models worked. MinMax was chosen even though there are several others scaling function to choose from as it is just pretty much straightforward, scaling to values between 0 and 1.
For the algorithms, the

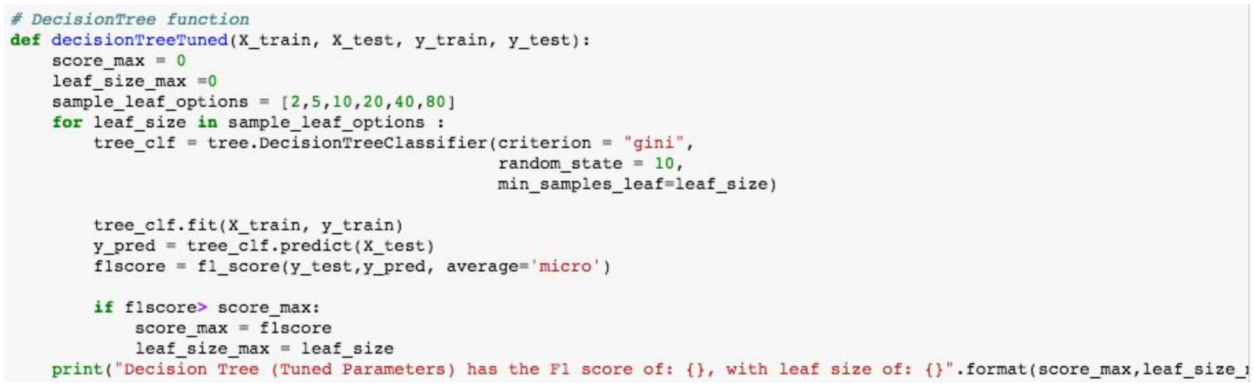
Refinement



```
# Logistic Regression function
def logsticRegression(X_train, X_test, y_train, y_test):
    C_param_range = [0.001,0.01,0.1,1,10]
    i = 0
    score_max = 0
    for i in C_param_range:
        LRclf = LogisticRegression(penalty = 'l2', C=i, random_state = 10)

        LRclf.fit(X_train,y_train)
        score = LRclf.score(X_test,y_test)
        if score> score_max:
                score_max = score
                i = i
    print("logstic Regression has the score of: {}, with C parameter of {}".format(score_max,i))
```

Image 12: Tuned Logistic regression

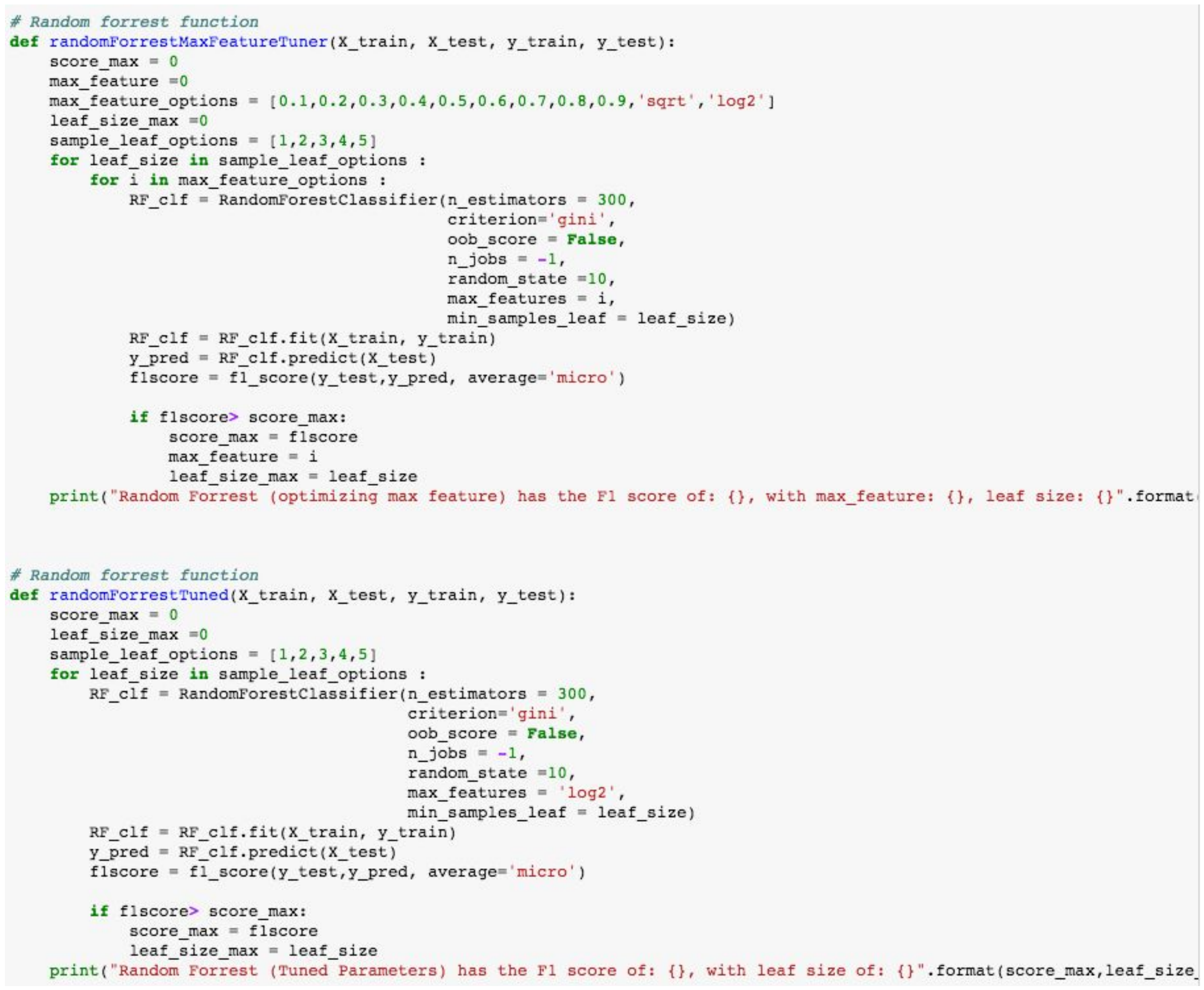
Here we have an image of the Logistic regression, it iterates through 5 different C values and then outputs the best score.



```
# DecisionTree function
def decisionTreeTuned(X_train, X_test, y_train, y_test):
    score_max = 0
    leaf_size_max =0
    sample_leaf_options = [2,5,10,20,40,80]
    for leaf_size in sample_leaf_options :
        tree_clf = tree.DecisionTreeClassifier(criterion = "gini",
                                               random_state = 10,
                                               min_samples_leaf=leaf_size)

        tree_clf.fit(X_train, y_train)
        y_pred = tree_clf.predict(X_test)
        f1score = f1_score(y_test,y_pred, average='micro')

        if f1score> score_max:
            score_max = f1score
            leaf_size_max = leaf_size
    print("Decision Tree (Tuned Parameters) has the F1 score of: {}, with leaf size of: {}".format(score_max,leaf_size_
```

Image 13: Tuned decision Tree

Here we have DecisionTree learner that iterates through 6 different min_sample_leaf options, the optimal min leaf size of those values will then output the best score, other parameters are set to default.



```
# Random forrest function
def randomForrestMaxFeatureTuner(X_train, X_test, y_train, y_test):
    score_max = 0
    max_feature =0
    max_feature_options = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,'sqrt','log2']
    leaf_size_max =0
    sample_leaf_options = [1,2,3,4,5]
    for leaf_size in sample_leaf_options :
        for i in max_feature_options :
            RF_clf = RandomForestClassifier(n_estimators = 300,
                                            criterion='gini',
                                            oob_score = False,
                                            n_jobs = -1,
                                            random_state =10,
                                            max_features = i,
                                            min_samples_leaf = leaf_size)
            RF_clf = RF_clf.fit(X_train, y_train)
            y_pred = RF_clf.predict(X_test)
            f1score = f1_score(y_test,y_pred, average='micro')

            if f1score> score_max:
                score_max = f1score
                max_feature = i
                leaf_size_max = leaf_size
    print("Random Forrest (optimizing max feature) has the F1 score of: {}, with max_feature: {}, leaf size: {}".format


# Random forrest function
def randomForrestTuned(X_train, X_test, y_train, y_test):
    score_max = 0
    leaf_size_max =0
    sample_leaf_options = [1,2,3,4,5]
    for leaf_size in sample_leaf_options :
        RF_clf = RandomForestClassifier(n_estimators = 300,
                                        criterion='gini',
                                        oob_score = False,
                                        n_jobs = -1,
                                        random_state =10,
                                        max_features = 'log2',
                                        min_samples_leaf = leaf_size)
        RF_clf = RF_clf.fit(X_train, y_train)
        y_pred = RF_clf.predict(X_test)
        f1score = f1_score(y_test,y_pred, average='micro')

        if f1score> score_max:
            score_max = f1score
            leaf_size_max = leaf_size
    print("Random Forrest (Tuned Parameters) has the F1 score of: {}, with leaf size of: {}".format(score_max,leaf_size_
```

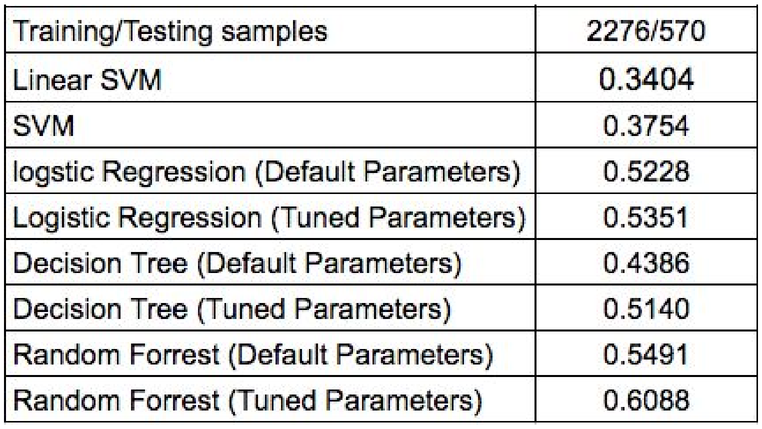
Image 14: Tuned Random forest

Similar with Decision tree, here the sample leaf options are 5 different leaf size, other parameters are default, except for max_feature, log2 in this case, after several iterations and fine tuning the parameters with different float value, a log2 was chosen as it gave promising score value.

IV. Results

Model Evaluation and Validation

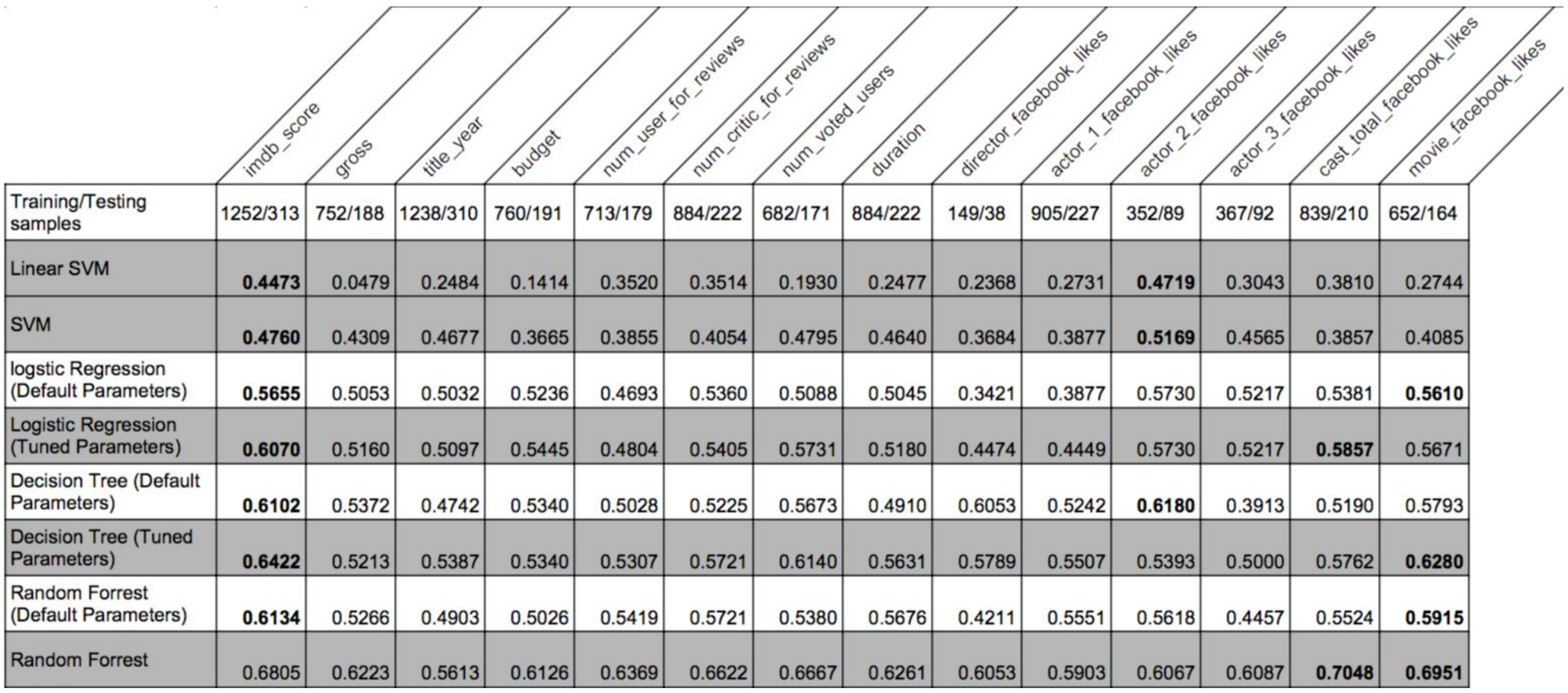
When the model is run with all the movies (English, USA, >79 minutes, in color) A set of scores are predicted, in the below table, these scores can be seen. The training samples are 80% of the data and testing samples are therefore 20% of the data. The Linear SVM gave **0.3404** in F1 score, which acts as a benchmark for the following learners. It can be clearly seen that Random Forest has the highest score, nearly two times higher then the benchmark score. Other noticeable scores are Logistic Regression and Decision Tree, they have similar scores both of them have all the default parameters except in decision Tree we iterate through several *min_sample_leaf* options.



Training/Testing samples	2276/570
Linear SVM	0.3404
SVM	0.3754
logstic Regression (Default Parameters)	0.5228
Logistic Regression (Tuned Parameters)	0.5351
Decision Tree (Default Parameters)	0.4386
Decision Tree (Tuned Parameters)	0.5140
Random Forrest (Default Parameters)	0.5491
Random Forrest (Tuned Parameters)	0.6088

Image 15: Model run with all movies

Now what happens if we change the data little bit, manipulate it so that only movies above the mean of the feature is chosen, i.e. imdb has a mean of 6.44, so we only take a look at movies above that mean, same is done with the other features and documented in the table below.



	imdb_score	gross	title_year	budget	num_user_for_reviews	num_critic_for_reviews	num_voted_users	duration	director_facebook_likes	actor_1_facebook_likes	actor_2_facebook_likes	actor_3_facebook_likes	cast_total_facebook_likes	movie_facebook_likes
Training/Testing samples	1252/313	752/188	1238/310	760/191	713/179	884/222	682/171	884/222	149/38	905/227	352/89	367/92	839/210	652/164
Linear SVM	**0.4473**	0.0479	0.2484	0.1414	0.3520	0.3514	0.1930	0.2477	0.2368	0.2731	**0.4719**	0.3043	0.3810	0.2744
SVM	**0.4760**	0.4309	0.4677	0.3665	0.3855	0.4054	0.4795	0.4640	0.3684	0.3877	**0.5169**	0.4565	0.3857	0.4085
logstic Regression (Default Parameters)	**0.5655**	0.5053	0.5032	0.5236	0.4693	0.5360	0.5088	0.5045	0.3421	0.3877	0.5730	0.5217	0.5381	**0.5610**
Logistic Regression (Tuned Parameters)	**0.6070**	0.5160	0.5097	0.5445	0.4804	0.5405	0.5731	0.5180	0.4474	0.4449	0.5730	0.5217	**0.5857**	0.5671
Decision Tree (Default Parameters)	**0.6102**	0.5372	0.4742	0.5340	0.5028	0.5225	0.5673	0.4910	0.6053	0.5242	**0.6180**	0.3913	0.5190	0.5793
Decision Tree (Tuned Parameters)	**0.6422**	0.5213	0.5387	0.5340	0.5307	0.5721	0.6140	0.5631	0.5789	0.5507	0.5393	0.5000	0.5762	**0.6280**
Random Forrest (Default Parameters)	**0.6134**	0.5266	0.4903	0.5026	0.5419	0.5721	0.5380	0.5676	0.4211	0.5551	0.5618	0.4457	0.5524	**0.5915**
Random Forrest	0.6805	0.6223	0.5613	0.6126	0.6369	0.6622	0.6667	0.6261	0.6053	0.5903	0.6067	0.6087	**0.7048**	**0.6951**

Image 16: Model Run with and without default parameters

It is interesting to see that the Linear SVM score is often worse than the benchmark F1 score (0.3404) set earlier, 8 out of 14 times, however, Random forest score is 9 out of 14 times higher than *all movies* F1 score in previous table and worth noticing that Random forest has always the highest score of these learners. The highest score from this observation is 0.7048 which was gotten from *cast_total_facebook_likes*, and then next highest is 0.695 from *movie_facebook_likes*.

Justification

As mentioned in the model evaluation, the final results are higher than the benchmark score set earlier. By iterating through many of the features and see what above mean feature gives us and often get similar or even better than other scores often leads to the conclusion that the model is significant enough to solve the problem of predicting the score.

V. Conclusion

Free-Form Visualization

Now lets take a look at a feature importance from the Random forest model that has been tuned, as you can see on image 17, there are 47 features but after about feature nr. 6, it decreases rapidly, which means that there are alot of features that do not contribute much for the learners.

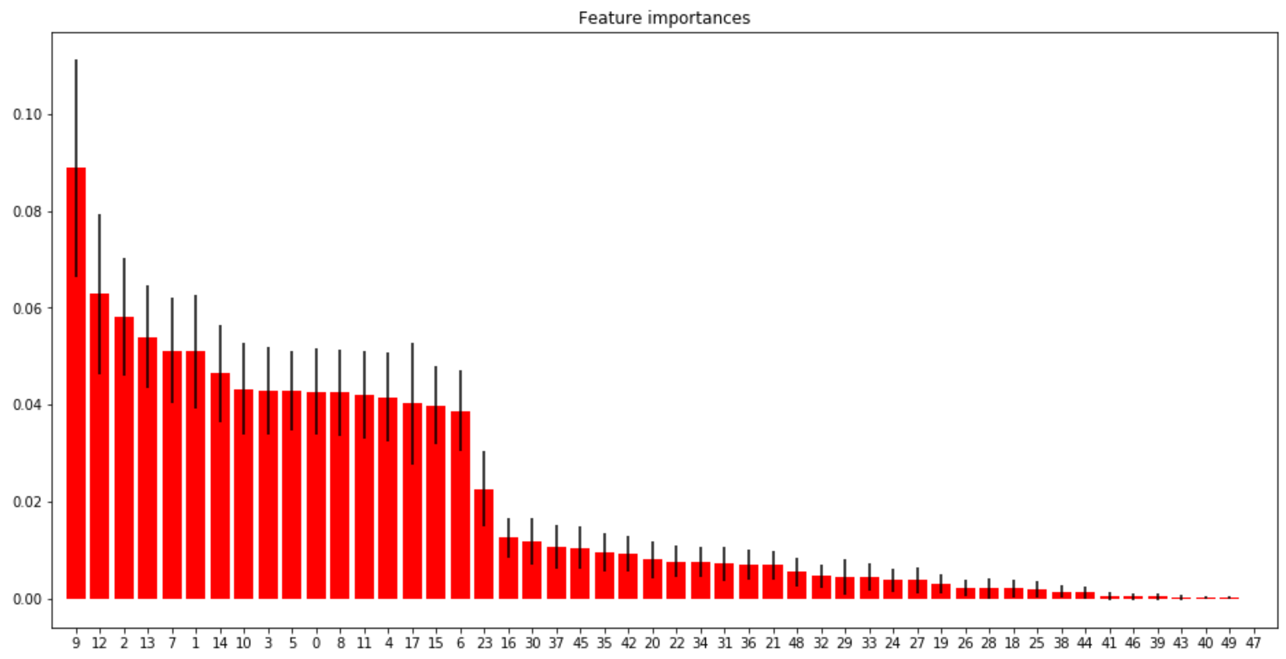
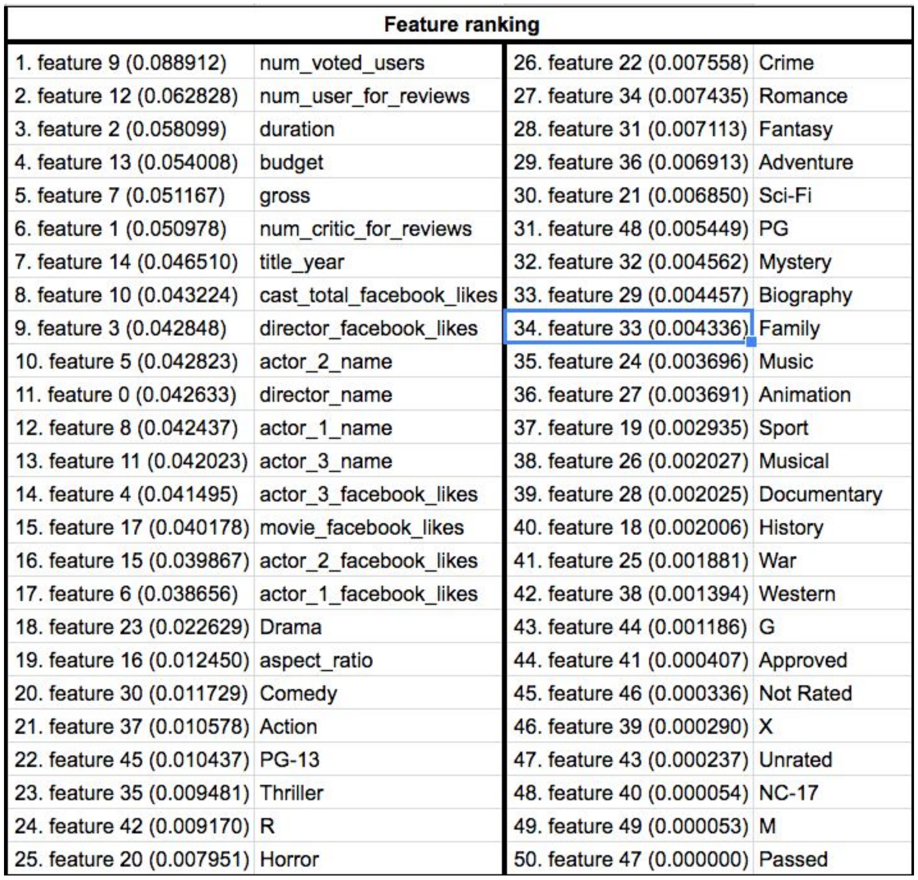


Image 17: feature importance plot

In image 18, the total list and the name of the features that represents the numbers on the plot on image 17 can be seen that the top features are mainly, votes, critics, reviews, budget, gross and social media which we are not surprised about. After *aspect ratio* feature it decreases rapidly, which makes you think that potentially only around top 20 features would benefit the model in predicting the F1 score well enough or maybe better.



Feature ranking			
1. feature 9 (0.088912)	num_voted_users	26. feature 22 (0.007558)	Crime
2. feature 12 (0.062828)	num_user_for_reviews	27. feature 34 (0.007435)	Romance
3. feature 2 (0.058099)	duration	28. feature 31 (0.007113)	Fantasy
4. feature 13 (0.054008)	budget	29. feature 36 (0.006913)	Adventure
5. feature 7 (0.051167)	gross	30. feature 21 (0.006850)	Sci-Fi
6. feature 1 (0.050978)	num_critic_for_reviews	31. feature 48 (0.005449)	PG
7. feature 14 (0.046510)	title_year	32. feature 32 (0.004562)	Mystery
8. feature 10 (0.043224)	cast_total_facebook_likes	33. feature 29 (0.004457)	Biography
9. feature 3 (0.042848)	director_facebook_likes	34. feature 33 (0.004336)	Family
10. feature 5 (0.042823)	actor_2_name	35. feature 24 (0.003696)	Music
11. feature 0 (0.042633)	director_name	36. feature 27 (0.003691)	Animation
12. feature 8 (0.042437)	actor_1_name	37. feature 19 (0.002935)	Sport
13. feature 11 (0.042023)	actor_3_name	38. feature 26 (0.002027)	Musical
14. feature 4 (0.041495)	actor_3_facebook_likes	39. feature 28 (0.002025)	Documentary
15. feature 17 (0.040178)	movie_facebook_likes	40. feature 18 (0.002006)	History
16. feature 15 (0.039867)	actor_2_facebook_likes	41. feature 25 (0.001881)	War
17. feature 6 (0.038656)	actor_1_facebook_likes	42. feature 38 (0.001394)	Western
18. feature 23 (0.022629)	Drama	43. feature 44 (0.001186)	G
19. feature 16 (0.012450)	aspect_ratio	44. feature 41 (0.000407)	Approved
20. feature 30 (0.011729)	Comedy	45. feature 46 (0.000336)	Not Rated
21. feature 37 (0.010578)	Action	46. feature 39 (0.000290)	X
22. feature 45 (0.010437)	PG-13	47. feature 43 (0.000237)	Unrated
23. feature 35 (0.009481)	Thriller	48. feature 40 (0.000054)	NC-17
24. feature 42 (0.009170)	R	49. feature 49 (0.000053)	M
25. feature 20 (0.007951)	Horror	50. feature 47 (0.000000)	Passed

Image 18: list of feature importance

Reflection

From the beginning, we were interested to see if a movie score can be predicted. Gathering the data, the correct data took some time but luckily a good data was found on kaggle. Exploring the data took some time, figure out what was a great idea to do with it and what data was irrelevant. The most difficult thing was what visualizing to use that gives the project a great picture of what to solve. After figuring out the technique then implementing the algorithms weren't that much of a problem as we were familiarized with the algorithms used in this project. Fine tuning the parameters and figure out which encoder to use took quite some time, at one point frustrating to do, but in the end, great results came through.

Improvement

The models could be potentially fine-tuned even more, possible using grid-search to iterate over many parameters to maximize the prediction score.

Another improvement could be using the plot_keywords column, as it was too complicated to use the plot_keywords as it was too many unique plot keyword, an improvement could using the top e.g. 100 or 1000 frequent plot keyword and add that as features.

The algorithms that were used in this project were algorithms that we were already familiarized with. Potentially trying out new and possible better-suited algorithms for this project could lead to a better score.

I wouldn't be surprised that there exists some sort of prediction model that the movie companies use that are much better than this model. This is, of course, multi-million/billion industry with a lot of money at stake.