**Predicting IMDB Movie Rating by Regression & XGB Classifier**

The objective of this project was to learn and implement the Machine learning algorithm using Python and the IMDB Rating dataset. We will be using the basic regression model, XGBoost regression, XGBoost Classification model. XGBoost is a powerful approach for Supervised regression models.

The goal of this project was: “To predict the quality of new contents added to streaming websites based on the movie, genres, cast, votes, and directors”. This kind of approach is necessary to save a huge amount of time and money before promoting and telecasting any content on streaming websites. It is also crucial to know about the audience’s likes and dislikes before its availability for the audience.

Through this project, we are going to predict the success of the movie based on the rating already given to the movie's contents and features which are important to keep in mind before producing and telecasting such content.

The other goal of this project was: “To compare and figure the most applicable algorithm for predicting movie ratings?” The proposed models intend to predict perfect accuracy, but ratings come from the complicated human nature we can see some assumptions here. Predictions that lie within +/-1 rating range can be considered for the linear regression model.

So, later we will check the accuracy of the Supervised Machine Learning technique and how accurately it can predict the ratings of movies.

1. **Data**

Data has been scraped from the publicly available website <https://www.imdb.com>. Kaggle web scraping project and datasets are of sufficient size to develop a good predictive model for movie ratings. To view the original data and related information click below link.

[Kaggle Dataset](https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset)

[https://www.imdb.com](https://www.imdb.com/)

1. **Method**

**Regression model**: Regression Analysis is a type of predictive modeling that is used to find the relationship between dependent and independent variables. Regression is widely used for analyzing data by looking at the fit of a curve/line. The fit of the curve is a line connecting to the data points in such a way that reduces the distances between the data points from the fitting line.

In the regression analysis, we can predict the value of an unknown variable by looking at its relationship with the known variable. In the linear regression method, the dependent variable is continuous, and the independent variable can be continuous or discrete and regression lines come linear.

It is represented by an equation Y=a+b\*X + c,

where a is the intercept,

b is the slope of the line and

c is the error term.

A regression line can be obtained by Least Square Method. Its calculation is based on finding the best-fit line of observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line. between positive and negative values. The regression model can be evaluated by using the metric R-square.

**Lasso method: l1 regularization**

Lasso is standing for Least Absolute Shrinkage and Selection Operator. It can penalize the absolute size of the regression coefficients and reduce the variability which can improve the accuracy of linear regression models. The larger penalty can further shrink the estimates towards zero which is important for the variable selections. If variable groups are highly correlated, lasso shrinks the others to zero and picks only one from them.

**XGBoost Machine Learning**

XG boost (Extreme Gradient Boosting) is widely used for classification and regression problems and gives better performance than other algorithms. It is the execution of gradient boosted decision trees. It is good for the small to medium tabular or structured data. It is used for supervised machine learning algorithms. It handles overfitting by using techniques of regularization. It is enabled with the inbuilt Cross-Validation (CV) function. It can handle the missing values by finding the trends and catching them. It has the power to save the data matrix and reload.

XGboost carries out the gradient boosting decision tree algorithm. Boosting is an ensemble technique that enables the model to make a prediction based on resolving the errors in the new model that has come from the old model. Model performance can be improved by tuning parameters.

These default metrics for the classification type of problem is an error and for regression metric is RMSE.

**Linear Regression Vs XGBoost**

Linear regression is a parametric model: it assumes the target variable can be expressed as a linear combination of the independent variables (plus error). Gradient boosted trees are nonparametric: they will approximate any\* function.

xgboost deprecated the objective Reg; Linear. It has been replaced by reg: squared error, and has always meant minimizing the squared error, just as in linear regression.

So, boost will generally fit training data much better than linear regression, but that also means it is prone to overfitting, and it is less easily interpreted. Either one may end up being better, depending on your data and your needs. (link)

We have started from Regression model and later moved from lasso to XGboost Classifier.

## **Data Preparation and Cleaning**

[Data Cleaning report](https://github.com/ReetuData/Project-1/blob/main/2%20IMDB%20data%20wrangling%20and%20Cleaning%20Part%202.ipynb)

**Step 1: Merging files**

The data set is given in seven small tsv files (all names are listed below). We have loaded and merged them all as a Data Frame named Final\_DF which has 14999145 row and 17 columns. [Link](https://github.com/ReetuData/Project-1/blob/main/1%20Cleaning%20and%20merging%20Data%20part%20-1%20.ipynb)

1. name\_data = name\_data.tsv
2. title\_akas = title\_akas.tsv
3. title\_basics = title\_basics.tsv
4. title\_crew = title\_crew.tsv
5. title\_episode = title\_episode.tsv
6. tilte\_principle = tiltle\_principle.tsv
7. title\_ratings = title\_ratings.tsv

**Step 2: Handling duplicate, unique, and missing values:**

After getting our final Data Frame, we have checked for the Duplicates, Index setting, datatypes, columns names, null/missing, and unique values. To make our data frame tidy, we have removed all duplicate values, renamed columns name as appropriate, checked and filled null values. We have also performed data type conversion as per nature of values. [Link](https://github.com/ReetuData/Project-1/blob/main/2%20IMDB%20data%20wrangling%20and%20Cleaning%20Part%202.ipynb)

**Step 3: Confirming data cleanliness and value types:**

There are a few more things to check column by column. It is to make sure our data is all set for further processing. Like few columns like the year, birthyear\_Director, death year were as object types. We have converted it to numerical as per the nature of the values. We have calculated the age of the director by subtracting the death year from the birth year. We have also derived the age of the movie by subtracting the release year from the current year. Later, We have divided movies into decades based on the movies age.

One final step we have performed before moving further was checking any null/missing values and datatypes. To make sure we have required values as needed. [Link](https://github.com/ReetuData/Project-1/blob/main/3%20data%20handling%20and%20EDA%20.ipynb)

## **EDA**

## [EDA report](https://github.com/ReetuData/Project-1/blob/main/3%20data%20handling%20and%20EDA%20.ipynb)

**Figure 1: Pair plot**

Chart, scatter chart

Description automatically generated

Figure 1 Pair Plot

**Figure 2: Histogram- AveRating**: AveRating for IMDB movies is normally distributed.

Chart, histogram

Description automatically generated

Figure 2. AveRating of IMDB movie

**Figure 3:** **Scatter Plot- AveRating Vs start year:** Movies with the release year 1900 to 1980 have a rating between 3.5 - 8.5. There are few movies with a high rating that were released after 2000. There are few movies with high and low ratings. Most movies have given a rating between 4-8.

Chart, scatter chart

Description automatically generated

Figure 3: Scatter Plot- AveRating Vs start year

**Figfure 4: BoxPlot: AveRating Vs startYear:** Movies that were released from 1900 t0 1980 have a wide range of ratings. It lies from 1.0 to 9.0. There are few movies after 1960 which are showing outliers.

Chart, bar chart

Description automatically generated

Figure 4 BoxPlot: AveRating Vs startYear

**Figure 5: Scatterplot: AveRating Vs Num\_of\_Votes:** Most of the votes have been given a rating from 4-8. There are few movies with fewer ratings and fewer numbers of votes and few movies with a high rating and fewer votes.

# Chart, scatter chart Description automatically generated

Figure 5: Scatterplot: AveRating Vs Num\_of\_Votes

**Figure 6: ScatterPlot: AveRating Vs Dirctor\_age:** Directors age isnt affecting the movie rating. We can see there is no such pattern where age is related to the highly rated movie. Directors from different age groups have created high-rated movies.

# Chart, scatter chart Description automatically generated

Figure 5 ScatterPlot: AveRating Vs Dirctor\_age

**Figure 7:** **ScatterPlot: AveRating Vs Age\_of\_movie:** People are showing a similar tendency over the years for giving movie ratings. They have given a rating based on the story of movies regardless of age.

Chart, scatter chart

Description automatically generated

Figure 6 ScatterPlot: AveRating Vs Age\_of\_movie

**Figure 9: ScatterPlot: AveRating Vs region:** US, GB and PT have wide rating ranges from where s others lie between 4-8.

Chart, scatter chart

Description automatically generated

Figure 9: ScatterPlot: AveRating Vs Region

**Figure 10: ScatterPlot-Age\_of\_movie Vs Region**: GB, US, FR, PT, and ES represent movies old as 120 and new as 10, whereas in the other regions movies age is distributed from 45 to 100.

**Chart, scatter chart

Description automatically generated**

Figure 7 Scatter Plot-Age\_of\_movie Vs Region

**Figure 11: ScatterPlot-Avg\_Rating Vs titleType**: Avg\_Rating for Short movies and movies ranges from 1 to 9 whereas tvepisodes and tvmovies have rated higher than 5.

Chart, scatter chart

Description automatically generated

Figure 11: ScatterPlot-Avg\_Rating Vs titleType

**Figure 12:** **Scatter plot- directors\_age vs AveRating:** The movie rating is not related to the age of the directors. High or low rated movies can be directors of any age group

Chart, scatter chart

Description automatically generated

Figure 12 Scatter plot- directors\_age vs AveRating

**Figure 13****: Scatter Plot: Decade vs AveRating**: Decades 4, 5, and 6 have all ranges of rating as low as 1 to as high as 10. The remaining decades dont have these wide ranges of ratings.

Chart, line chart

Description automatically generated

Figure 13: Scatter Plot: Decade vs AveRating

## **Data preprocessing: Encoding**

[Data preprocessing Report](https://github.com/ReetuData/Project-1/blob/main/5%20Data%20encoding%20and%20Modelling.ipynb)

## Before fitting our data to any model, we must make sure all our categorical features areas are in numerical form. Here we have categorical columns like titleId, title, region, titleType, directors, writers, primaryName\_Director, primaryProfession\_director, Dir\_knownForTitles, Decade. We have used oneHotencoder to convert it into numerical columns.

## Columns like genres, Dir\_knownForTitles contain more than comma-separated values in single columns. For that, we have used the multilabelBinarizer which can easily deal with multi values.

## 

## We have concatenated both encoded data frames and now our data is ready to fit in the model.

## **Modeling**

## We have started with regression model. We have fit the model on to train data set and predict the value of test data set as y\_pred. The model performance was evaluated from the r\_squared values, which was 0.32 in our case.

Comparison between y\_test and y\_predicted values. The red dotted plot shows that y-observed and y-predicted values lies around same range. Both values are concentrated between 4-8 rating range.

Black dotted plot is error plot of y\_observed and y\_predicted values. The error values are mostly crowded near to zero with upper and lower bound 2 and -2.

A picture containing text

Description automatically generated

Figure 14 Comparison between y\_observed and y\_predicted values.

## **Model Validation**

## We have used the Lasso method, XGBoost Regressor, and XGBClassifier for the model validation. We have got 0.12, 0.31, and 0.14 rmse respectively. It seems XGBoost Regressor is giving better performance in all other methods.

1. **Hyperparameter Tunning (tree-based parameters)**

**Hyperparameter Tuning (tree-based parameters)**

**Learning rate (eta values)**: step size shrinkage used in the update to prevent overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative. The default range for eta is 0-1. In our case, we got the best eta 0.1.

**max\_depth:** Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. Default values are 6 and ranges could be 0 to ∞ (0 is only accepted in loss guided growing policy when tree\_method is set as hist or gpu\_hist). In our case, we got 20 as best max\_depth.

**Gamma**: Minimum loss reduction required to make a further partition on a leaf node of the tree... The larger gamma is, the more conservative the algorithm will be. Its default value is 0 and could be 0 to ∞. In our case, we got the best Gamma 0.01

**min\_child\_weight:**Minimum sum of instance weight (hessian) needed in a child. 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. In the linear regression task, this simply corresponds to the minimum number of instances needed to be in each node. The larger min\_child\_weight is, the more conservative the algorithm will be. Its default value is 1 and ranges from 0 to ∞. In our case, we got the best min\_child\_weight6.

**colsample\_bytree:**Subsample ratio of columns when constructing each tree. subsampling of columns with default value 1 and range from 0 to 1. In our case, we got the best colsample\_bytree 0.6.

**Subsample**: Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data before growing trees. and this will prevent overfitting. Subsampling will occur once in every boosting iteration. Its default value is 1 and the range is from 0 to 1, In our case, we got the best subsample 0.6.

**min\_child\_weight:** Minimum sum of instance weight (hessian) needed in a child. 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. In linear regression task, this simply corresponds to minimum number of instances needed to be in each node. The larger min\_child\_weight is, the more conservative the algorithm will be. Its default value is 1 and reange from 0 to ∞. In our case, we got best min\_child\_weight6.

**colsample\_bytree:** Subsample ratio of columns when constructing each tree. subsampling of columns with default value 1 and range from 0 to 1. In our case, we got best colsample\_bytree 0.6.

**Subsample**: Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting. Subsampling will occur once in every boosting iteration. Its default value is 1 and range is from 0 to 1, In our case , we got best subsample 0.6.

# Tunning regularization parameters

**Lambda:**L2 regularization term on weights. By increasing this value will make the model more conservative. The default value is 1. We had the best lambda 1.

**Alpha:** L1 regularization term on weights. Increasing this value will make the model more conservative. Its default value is 0. We got the best alpha is 0.1

**After feeding all the best tree-based and regularization parameters, we have found the five best features which are playing an important role in increasing the movie ratings. These 5 features are f12135 score 97, f12137 score 95, f8741 score 60, f12136 score 60, f7980 score 49**

# Hyper-parameter tuning by grid search

We have also run the gird search to find the best parameter values for the model. After running grid search the best values were colsample\_bytree -0.5, learning\_rate - 0.01, max\_depth - 7, min\_child\_weight - 1, n\_estimators - 200, subsample - 0.6. and the best features we found after feeding all the best parameters values derived from grid search were **f7979 score 150, f12090 score 73,f8697 score 65, f11259 score 63, f12092 score 56.**

# Hyper-parameter tuning by Random Search

# We have also run the Random search to find the best parameter values for the model. After running random search, the best values were subsample - 0.6, n\_estimators - 25, min\_child\_weight - 10, max\_depth - 11, learning\_rate - 1.0, colsample\_bytree - 0.9 and the best features we found after feeding all the best parameters values derived from random search were f7976 score 542, f7979 score 484, f12136 score 477, f12137 score 432, f8741 score 384

## **Credits**

Thanks, Vivek Kumar for being an amazing Springboard mentor throughout my course work.

**11. References**

<https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d>

<https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset>