**Predicting IMDB Score**

**Pranav Srivastava**

**Abstract** The main objective of this project is to follow steps in a machine learning lifecycle to build a model, based on the IMDB dataset, that predicts IMDB score of a movie.

**Summary of Results** : Multi-class classification model is created using RandomForestClassifier and it has an accuracy of 76.09

**1** **Introduction**

I have chosen IMDB dataset from Kaggle for this study. It contains 28 features and 5043 instances. In the table below you can see the list of features. “imdb\_score” at index 26 is our target class. Rest of the features are the predictors.

|  |  |
| --- | --- |
| index | Features |
| 1 | color |
| 2 | director\_name |
| 3 | num\_critic\_for\_reviews |
| 4 | duration |
| 5 | director\_facebook\_likes |
| 6 | actor\_3\_facebook\_likes |
| 7 | actor\_2\_name |
| 8 | actor\_1\_facebook\_likes |
| 9 | gross |
| 10 | genres |
| 11 | actor\_1\_name |
| 12 | movie\_title |
| 13 | num\_voted\_users |
| 14 | cast\_total\_facebook\_likes |
| 15 | actor\_3\_name |
| 16 | facenumber\_in\_poster |
| 17 | plot\_keywords |
| 18 | movie\_imdb\_link |
| 19 | num\_user\_for\_reviews |
| 20 | language |
| 21 | country |
| 22 | content\_rating |
| 23 | budget |
| 24 | title\_year |
| 25 | actor\_2\_facebook\_likes |
| 26 | imdb\_score |
| 27 | aspect\_ratio |
| 28 | movie\_facebook\_likes |

I have focused on following predictions:

**multi-class classification** – predict if the movie is

bad (0-4),

average (4-6),

good (6-8),

excellent (8-10)

Many people believe that movies are reflection of the society. The motivation to build models using this dataset is to predict class of the movie using classification or exact rating of the movie using regression. Knowing the score helps a user to have an idea about the movie before they start watching it or buying the tickets. Preprocessing of data, feature selection, model selection, model evaluation, hyperparameter optimization for tuning best models are the important phases in machine learning lifecycle and are implemented using scikit-learn in this project.

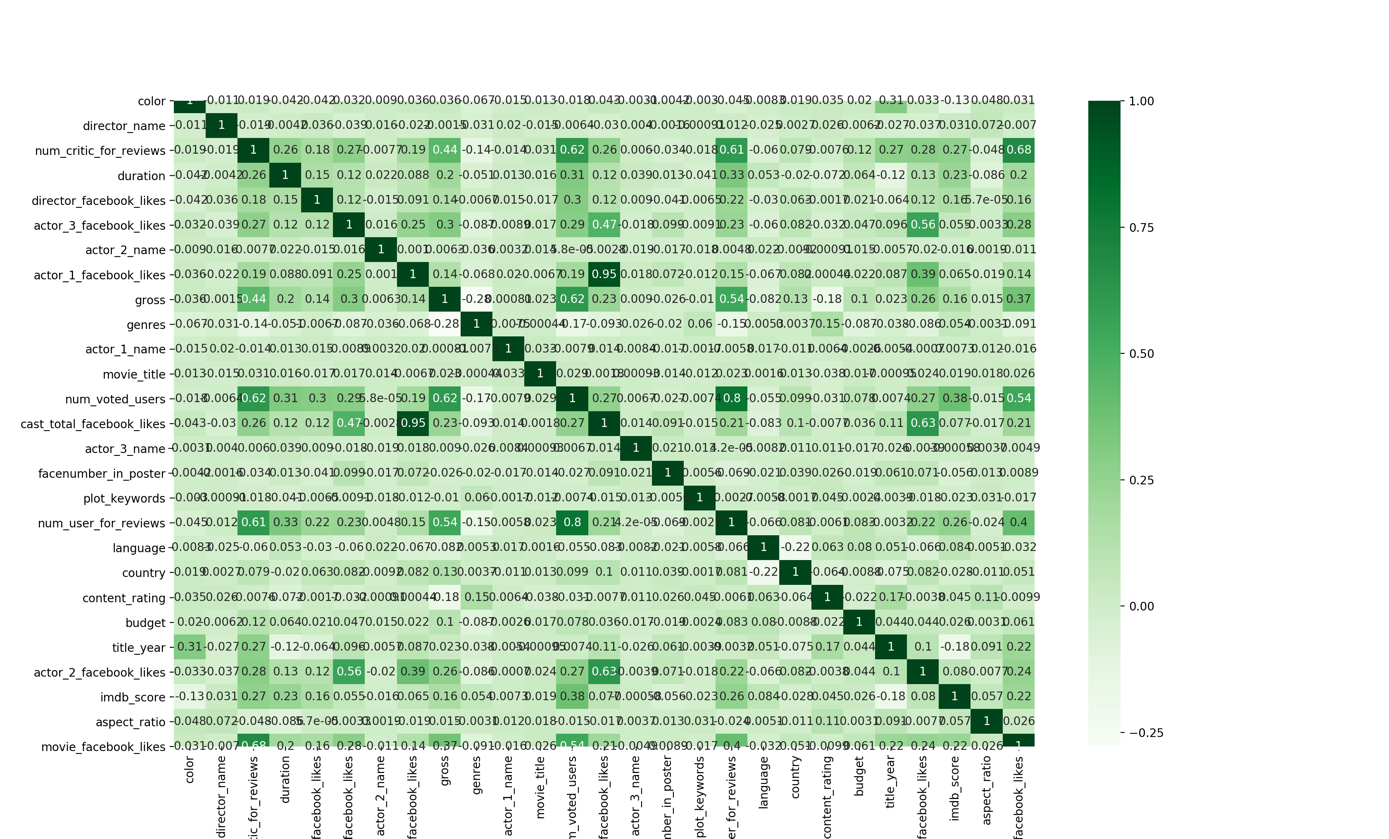
There are many academic papers available on internet that used this dataset. I found an interesting paper from Jackson State University with name “Predicting Movie Success Using Machine Learning Algorithms”. Paper is available here/reference: <http://www.laccei.org/LACCEI2017-BocaRaton/student_Papers/SP499.pdf>

This paper proposes a way to predict how successful a movie will be prior to its arrival at the box office. It is a multi-class classification problem with five categories for the “success rate” of movie. The interesting part is that they used two datasets (IMDB and YouTube) and merged it into one, in order to build a model that can achieve a better prediction.

**2 Research**

Feature Selection is one of the most important steps in a machine learning lifecycle. In a dataset every column is a feature. Few features are correlated with the target class and many irrelevant features are not correlated with the target class. In other words, only few features will have an impact on the output variable. If the irrelevant features are not removed, then it is highly probable that they will make the model perform worst. Hence, there comes a need to carry out feature selection.

Below is the heatmap before the feature selection is done. We can see that there is a high correlation between many feature columns.

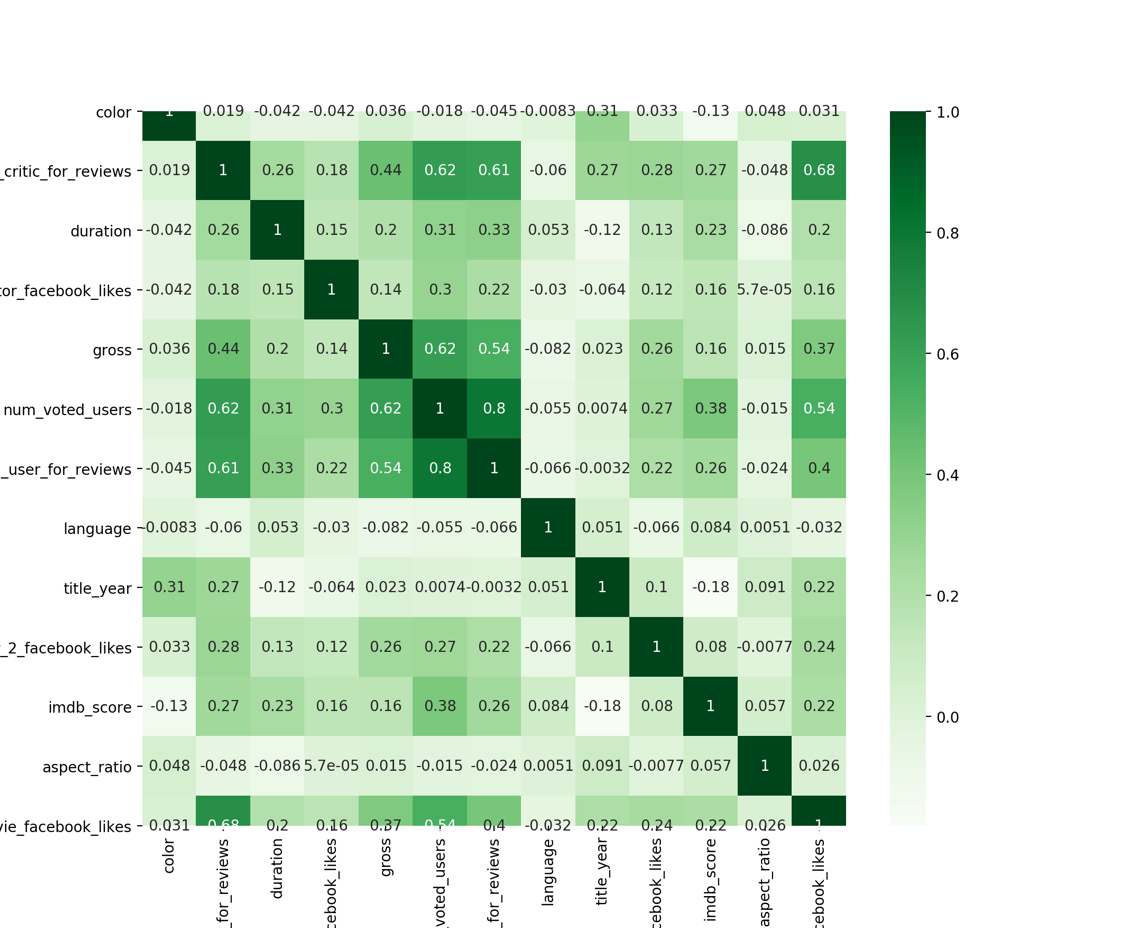


I have implemented below three techniques for the feature selection.

1. **Univariate feature selection:** This is the list of 12 most relevant selected feature column.

Selected features=['num\_voted\_users', 'num\_user\_for\_reviews', 'movie\_facebook\_likes', 'num\_critic\_for\_reviews', 'duration', 'title\_year', 'director\_facebook\_likes', 'gross', 'color', 'language', 'aspect\_ratio', 'actor\_2\_facebook\_likes'

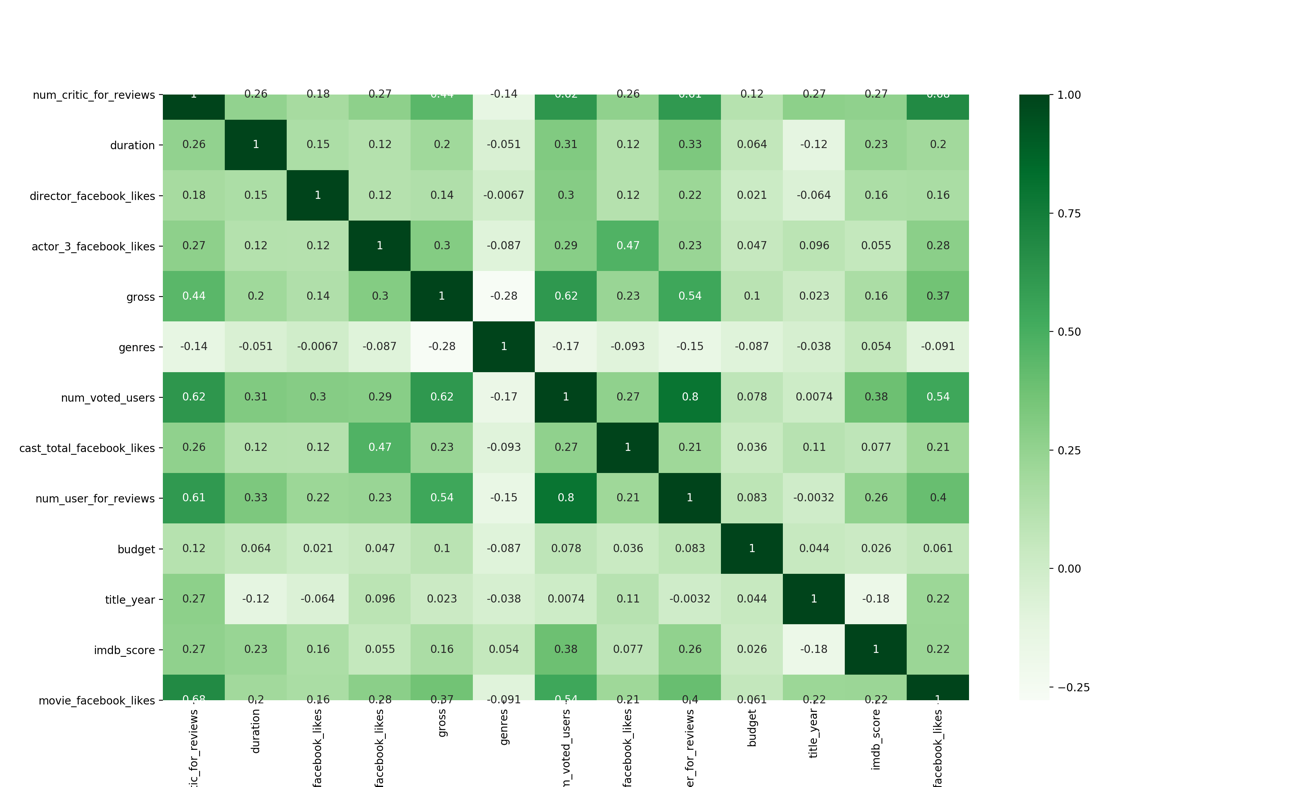
This heatmap is generated after dropping the feature columns that were not in the top 12 relevant features. We can notice that there are less features which are highly correlated with other features.



1. **Tree based feature selection**: This is the list of top 12 feature columns selected by this technique.

Selected features = ['num\_voted\_users', 'duration', 'genres', 'num\_critic\_for\_reviews', 'title\_year', 'num\_user\_for\_reviews', 'gross', 'budget', 'director\_facebook\_likes', 'movie\_facebook\_likes', 'cast\_total\_facebook\_likes', 'actor\_3\_facebook\_likes']

This heatmap is generated after dropping the feature columns that were not in the top 12 relevant features. This picture looks better than univariate feature selection.



1. **Greedy feature selection:** This is the list of top 12 feature columns selected by this technique.

Selected features = ['director\_name', 'num\_critic\_for\_reviews', 'duration', 'director\_facebook\_likes', 'actor\_3\_facebook\_likes', 'actor\_2\_name', 'actor\_1\_facebook\_likes', 'genres', 'num\_user\_for\_reviews', 'title\_year', 'actor\_2\_facebook\_likes']

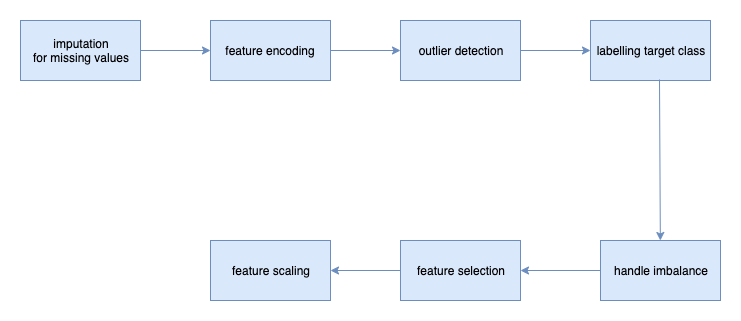
This heatmap is generated after dropping the feature columns that were not in the top 12 relevant features. This picture looks even better than tree based feature selection.

****

**Result of the research : ‘**Tree Based Feature Selection’ stands out to be the best and will have used it in the further steps.

**3 Methodology**

**3.1 Sequence of preprocessing steps**



I undertook the **preprocessing steps** mentioned in the figure above. Brief explanation for choosing them -

1. Imputation – as there were a lot of missing values but were less than 30% in each row or column I chose not to drop them but impute them. I used SimpleImputer class in combination with ‘most\_frequent’ and ‘mean’ strategy parameter for string and integer feature columns respectively.
2. Feature encoding – it was required because few feature columns had string values
3. Outlier detection – I used it to check if there were major outlier in the dataset. However, I did not implement the handling techniques because I did not see major outliers in the dataset.
4. Labeling target class – this was necessary as I was aiming for multi-class classification problem. I binned the ‘imdb\_score’ column into 4 quartiles/labels (bad, average, good, excellent).
5. handle imbalance – after labeling the target class I detected imbalance in the target class and handled it using SMOTE
6. Feature selection – this is also my research topic in section 2. In order to reduce the complexity of model, reduce overfitting I applied this step.
7. Feature scaling – as my dataset was having huge variation in the feature column so I used this step to normalize the feature columns. This helped in having correct evaluation of models which are heavily dependent on distance metric like kNN.

**3.2 Range of models and hyper-parameter optimization techniques**

I used below models for initial model building phase.

I have used **GridSearchCV** to evaluate the range of parameters for the best performing models. GridSearchCV is basically based on **brute force search**.

Further, I have used cross fold validation to validate the results.

|  |  |
| --- | --- |
| Model | Range of Parameters examined |
| RandomForestClassifier : A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default). | n\_estimators : The number of trees in the forest.  max\_features : The number of features to consider when looking for the best split.  n\_jobs : The number of jobs to run in parallel. |
| DecisionTreeClassifier: A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. | criterion: The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain.  max\_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min\_samples\_split samples.  min\_samples\_leaf: The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min\_samples\_leaf training samples in each of the left and right branches.  Splitter: The strategy used to choose the split at each node. Supported strategies are “best” to choose the best split and “random” to choose the best random split.  max\_features: The number of features to consider when looking for the best split |
| KNeighborsClassifier: Classifier implementing the k-nearest neighbors vote. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors | n\_neighbors: Number of neighbors to use  p : Power parameter for the Minkowski metric.  algorithm: Algorithm used to compute the nearest neighbors. ball\_tree, kd\_tree, brute, auto are options.  leaf\_size: Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.  weights: weight function used in prediction. Possible values are ‘uniform’, ‘distance’ etc |
| SVC: C-Support Vector Classification.  The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. | c: Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.  gamma: Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’ |

**4 Evaluation**

**4.1 Hyperparameters optimization** As I briefly discussed about the hyperparameters optimization in the last section, here are the results to present for the models.

Below functions are used to generate results for hyper parameters optimization

sklearnMetricSVM(self)

sklearnMetricsRandomForest(self)

sklearnMetricsDecisionTree(self)

sklearnMetricsKNN(self)

1. **RandomForestClassifier:**

param\_grid = {

'n\_estimators': [100, 150, 200, 250, 300],

'max\_features': ['auto', 'sqrt', 'log2'],

'n\_jobs' : [-1,1,2,3]

}

**Random Forest Classifier:** {'max\_features': 'log2', 'n\_estimators': 100, 'n\_jobs': -1}

**best score** : 0.7223769801941258

**(ii) DecisionTreeClassifier:**

parameters={ 'criterion':['gini','entropy'],

'max\_depth': np.arange(3, 15),

'min\_samples\_leaf':np.arange(1,10),

'splitter' : ['best', 'random'],

'max\_features': ['auto', 'sqrt', 'log2']}

**Best parameters for Decision Tree Classifier:** {'criterion': 'gini', 'max\_depth': 9, 'max\_features': 'auto', 'min\_samples\_leaf': 2, 'splitter': 'best'}

criterion, max\_features, splitter already maches the default values

**(iii) kNearestNeighbour:**

param\_grid= [ {'clf\_\_n\_neighbors': list(range(1, 5)),

'clf\_\_p':[1, 2, 3, 4, 5],

'clf\_\_algorithm':['auto','kd\_tree'],

'clf\_\_leaf\_size': list(range(40, 50))}]

**Best params:** {'clf\_\_algorithm': 'auto', 'clf\_\_n\_neighbors': 3, 'clf\_\_p': 1} with a score of 0.6006200151021756

**(iv) SVC:**

parameters = {'C': [1, 10],'gamma': [0.001, 0.01, 1]}

**best score:** 0.6690381801878333

**best params:** SVC(C=10, break\_ties=False, cache\_size=200, class\_weight=None, coef0=0.0,

decision\_function\_shape='ovr', degree=3, gamma=1, kernel='rbf', max\_iter=-1,

probability=False, random\_state=None, shrinking=True, tol=0.001,

verbose=False)

**4.2 Evaluation Metrics** I have used two evaluation metrics to show the results. Below is the code snippets that produces the evaluation metrics results.

def evaluationMetrics(self, model):

#This function creates a confusion matrix, produces F1 score

#Also, this function prints the accuracy and plots confusion matrix

if model == "RandomForestClassifier":

#model = RandomForestClassifier()

model = RandomForestClassifier(max\_features = 'log2', n\_estimators = 100, n\_jobs = -1)

elif model == "DecisionTreeClassifier":

#model = DecisionTreeClassifier()

model = DecisionTreeClassifier(max\_depth = 9, min\_samples\_leaf = 2)

elif model =="kNN":

#model = KNeighborsClassifier()

model = KNeighborsClassifier(algorithm = 'auto', n\_neighbors = 3, p =1)

#model = RandomForestClassifier()

model.fit(self.X\_train, self.y\_train)

y\_pred= model.predict(self.X\_test)

accuracy = accuracy\_score(self.y\_test, y\_pred)

print("Accuracy of model: ", accuracy )

cf\_mat= confusion\_matrix(y\_true=self.y\_test, y\_pred=y\_pred)

print('Confusion matrix of model:\n', cf\_mat)

#generates the F1 score

print(classification\_report(self.y\_test,y\_pred))

sns.heatmap(confusion\_matrix(self.y\_test,y\_pred),annot=True,lw =2,cbar=False, linewidths=.5)

plt.ylabel("True Labels")

plt.xlabel("Predicted Labels")

plt.title("CONFUSION MATRIX")

#plt.tight\_layout()

plt.show()

1. **Confusion Matrix:** The confusion matrix is used to describe the performance of a classification model on a set of test data for which true values are known. We can see correct prediction along the diagonal.



**b) F1 Score:** This comes from the confusion matrix. Based on the confusion matrix, we can calculate the precision and the recall scores. More details [here](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html)

**Models and evaluation metrics:**

Note-unfortunately I haven’t added confusion matrix heatmap in this report as it was already exceeding the desired size. Plus there is a bug with sns and my matrix are a bit broke. However, it was in the code.

**RandomForestClassifier**

**Confusion matrix and accuracy F1 Score**

precision recall f1-score support

1 1.00 0.08 0.15 36

2 0.69 0.53 0.60 299

3 0.77 0.92 0.84 623

4 0.90 0.56 0.69 50

accuracy 0.75 1008

macro avg 0.84 0.52 0.57 1008

weighted avg 0.76 0.75 0.73 1008

before the optimal parameters are applied:

Before the optimal parameters are applied

Accuracy of model: 0.7589285714285714

Confusion matrix of model:

[[ 2 25 9 0]

[ 0 164 135 0]

[ 0 48 572 3]

[ 0 0 23 27]]

After the optimal parameters are applied

Accuracy of model: 0.7609126984126984

Confusion matrix of model:

[[ 2 23 11 0]

[ 0 167 132 0]

[ 0 51 570 2]

[ 0 0 22 28]]

**DecisionTreeClassifier:**

**Confusion matrix and accuracy F1 score**

precision recall f1-score support

1 0.00 0.00 0.00 36

2 0.55 0.59 0.57 299

3 0.77 0.79 0.78 623

4 0.67 0.58 0.62 50

accuracy 0.69 1008

macro avg 0.50 0.49 0.49 1008

weighted avg 0.67 0.69 0.68 1008

Before

Accuracy of model: 0.6458333333333334

Confusion matrix of model:

[[ 5 17 14 0]

[ 21 143 131 4]

[ 17 112 470 24]

[ 0 1 16 33]]

After

Accuracy of model: 0.6944444444444444

Confusion matrix of model:

[[ 0 26 10 0]

[ 4 177 117 1]

[ 2 116 493 12]

[ 0 2 18 30]]

**kNearestNeighbour:**

**Confusion matrix and accuracy F1 Score**

Before the optimal parameters are applied

Accuracy of model: 0.6884920634920635

Confusion matrix of model:

[[ 7 20 9 0]

[ 9 147 143 0]

[ 4 100 518 1]

[ 0 4 24 22]]

After the optimal parameters are applied

Accuracy of model: 0.6805555555555556

Confusion matrix of model:

[[ 9 18 9 0]

[ 18 144 137 0]

[ 13 98 509 3]

[ 0 2 24 24]]

precision recall f1-score support

1 0.23 0.25 0.24 36

2 0.55 0.48 0.51 299

3 0.75 0.82 0.78 623

4 0.89 0.48 0.62 50

accuracy 0.68 1008

macro avg 0.60 0.51 0.54 1008

weighted avg 0.68 0.68 0.67 1008

**SVC:**

**Confusion matrix and accuracy F1 Score**

precision recall f1-score support

1 0.00 0.00 0.00 36

2 0.52 0.18 0.26 299

3 0.67 0.94 0.78 623

4 0.93 0.54 0.68 50

accuracy 0.66 1008

macro avg 0.53 0.41 0.43 1008

weighted avg 0.61 0.66 0.60 1008

Before the optimal parameters are applied

Accuracy of model: 0.6418650793650794

Confusion matrix of model:

[[ 0 1 35 0]

[ 0 21 278 0]

[ 0 16 607 0]

[ 0 1 30 19]]

After the optimal parameters are applied

Accuracy of model: 0.6607142857142857

Confusion matrix of model:

[[ 0 13 23 0]

[ 0 53 246 0]

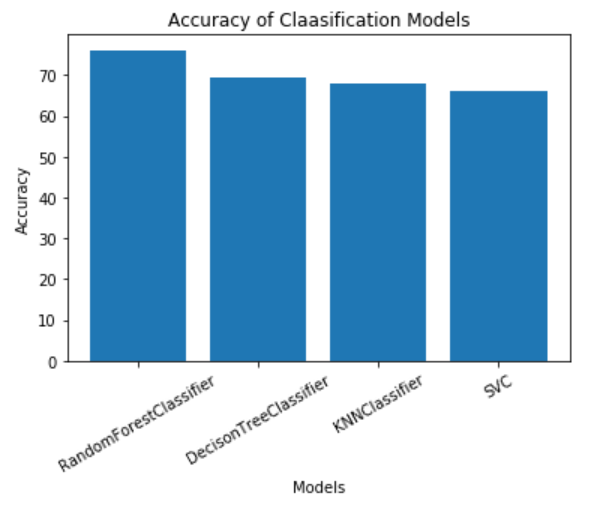
[ 0 35 586 2]

[ 0 1 22 27]]

**5 Conclusions and Future Work**

It can be concluded from this study that RandomForestClassifier model has the highest accuracy (76.09) among all the other models used in this study.

In the future I would like to extend this model to a higher accuracy. I am very inspired by various techniques that were used in the “Predicting Movie Success Using Machine Learning Algorithms” paper that I mentioned in first section. Also, I would like continue the research on hyperparameter optimization which I believe will be very useful while learning ‘Deep Learning’.



Used generategraph(self) in my code to generate below result

References –

https://scikit-learn.org/

https://en.wikipedia.org/wiki/K-nearest\_neighbors\_algorithm

https://towardsdatascience.com/common-classification-model-evaluation-metrics-2ba0a7a7436e

https://www.kaggle.com/carolzhangdc/imdb-5000-movie-dataset