PROJECT REPORT

CHESS (KING-ROOK V/S KING)

OR-568

Applied Predictive Analytics

Instructor: DR. VADIM SOKOLOV

Team:

HARITHA TUMMANAPALLY (G01197018)

MAYANK DUBEY (G01220775)

VENKATA VAMSI RAM PATIBANDLA (G01209767)

SAISUDHA SURYANARAYANAN (G01228903)

1. INTRODUCTION

Chess is a two-player strategy board game played on an 8 × 8 checkered board with 64 squares. The history of chess can be drawn back to around 1500 years, it was started in the North of India and spread throughout the world. Each player begins with 16 pieces one king, one queen, two rooks, two knights, two bishops and eight pawns. Each type of piece moves uniquely of which the most powerful is the queen and the least powerful is the pawn. Although the rules of the game have been amended multiple times the main objective is to checkmate the opponent's king by placing it under an inescapable threat of capture. The invention of databases and chess engines in the 20th century revolutionized chess.

The goal of the project is to predict the number of moves (from 0-16) required to win for white with a king and a rook, given the position of white king, white rook and black king. It is a multiclass classification problem. The dataset has 28,056 rows and has the following attributes

1. White King file (column) - X1
2. White King rank (row) - X2
3. White Rook file -X3
4. White Rook rank -X4
5. Black King file - X5
6. Black King rank -X6
7. optimal depth-of-win for White in 0 to 16 moves, otherwise drawn {draw, zero, one, two, ..., sixteen}. -Y

As a team we are fascinated with the game. The rich history and the complexity (10 to the power 120 games possible) of the game dragged us to this project. The chess Engine came to limelight when deep blue, IBM Engine beat Garry Kasparov (in year 1997) the Russian Grand Master and world champion at that time. After that chess Engines took a huge turn and engines like stock fish were developed which is based on brute force algorithms. In late 2017 Google developed an AI based chess Engine which learned the game, in just 4 hours playing with itself and beat stockfish8. We got inspired by the engines and how they evolve that is why we want to explore this dataset.

We are using the following methods on the dataset:

- Multinomial Logistic Regression
- Decision Tree
- Random Forest
- Boosting algorithm's

In a chessboard a, b, c, d, e, f, g, h be the columns and 1, 2, 3, 4, 5, 6, 7, 8 be the rows where we place the pieces.

If we take an example from the dataset, the white king is placed in column a and row 1, white rook is placed in column c and row 2, black king is placed in column c and row 1 and if it is black to move then the result is a draw as shown in Figure 1.

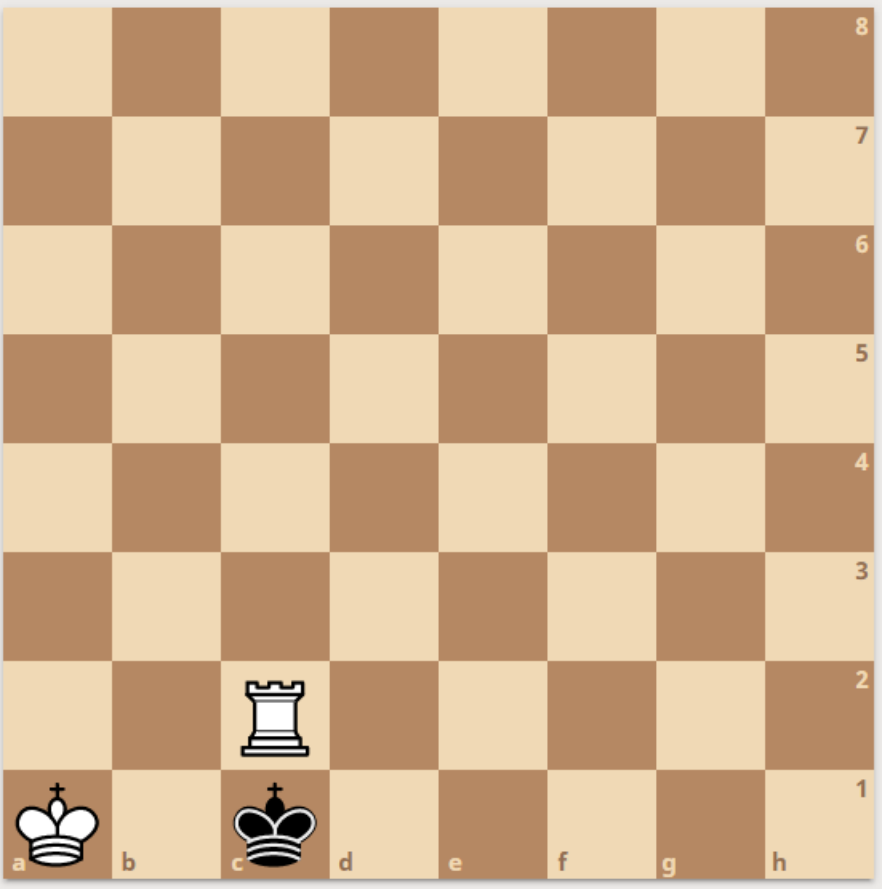

Figure 1

2. EDA

```
In [62]: df = df.rename(columns={"a": "White King file (column)", "1": "White King rank (row)",
                "b": "White Rook file" ,"3": "White Rook rank" ,
                 "c": "Black King file", "2" : "Black King rank",
                 "draw": "moves to win"})
```

```
In [63]: df.head()
```

Out[63]:

	White King file (column)	White King rank (row)	White Rook file	White Rook rank	Black King file	Black King rank	moves to win
0	a	1	c	1	c	2	draw
1	a	1	c	1	d	1	draw
2	a	1	c	1	d	2	draw
3	a	1	c	2	c	1	draw
4	a	1	c	2	c	3	draw

Firstly, renamed the columns present in the data frame with more relevant and simpler words like white king file, black king rank, moves to win etc.

moves_count = df['moves to win'].value_counts()

In the above statement ‘moves_count’ is the predicting variable to plot the number of moves. A bar plot is outputted as a result of checking the occurrence of a draw in the game and frequency of each move is plotted and we can say that in 14 moves we have the highest frequency to win the game.

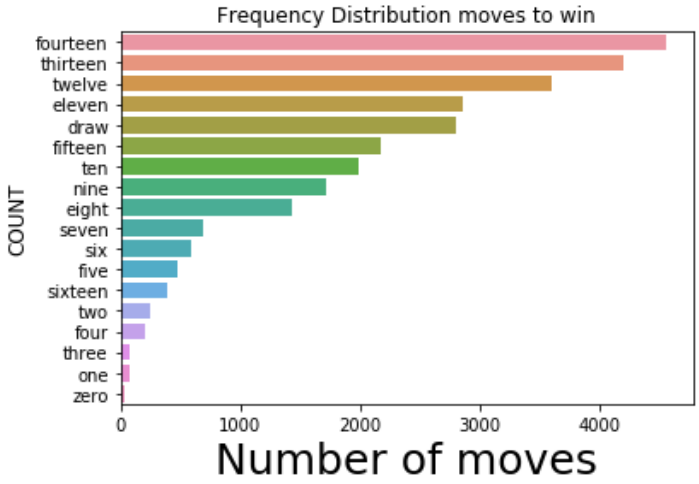

Figure - 2

From moves to win column ‘draw’ rows are removed.

```
In [69]: df2.head()
```

Out[69]:

	White King file (column)	White King rank (row)	White Rook file	White Rook rank	Black King file	Black King rank	moves to win
2795	c	1	a	3	a	1	0
2796	c	1	a	4	a	1	0
2797	c	1	a	5	a	1	0
2798	c	1	a	6	a	1	0
2799	c	1	a	7	a	1	0

Boxplot is probably one of the most common type of graphic. It gives a nice **summary** of one or several **numeric variables**. The line that divides the box into 2 parts represents the **median** of the data. The end of the box shows the upper and lower **quartiles**. The extreme lines show the highest and lowest value excluding **outliers**.

- **sns.boxplot(x="White King rank (row)", y="moves to win", data=df2)**

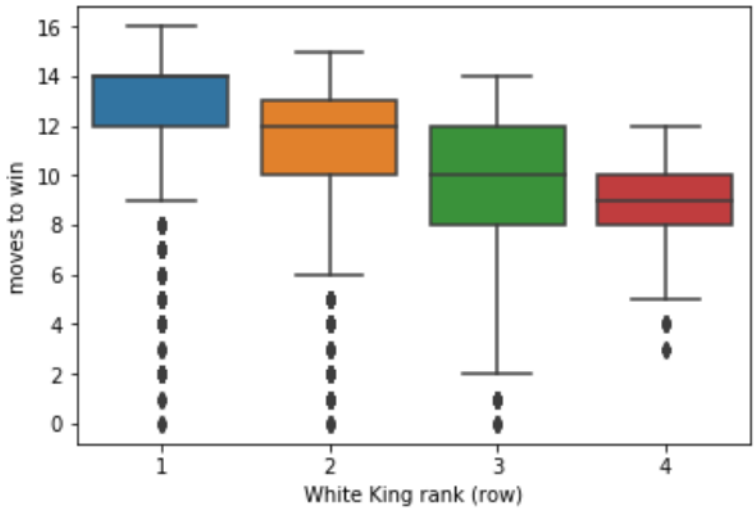

Figure – 3

In the above figure white king's moves are calculated in order to win, the maximum moves for him to win are 16. We can say that the boxplot is biased as it has only 1, 2, 3, 4 rank (row) which are only one fourth of the chess board.

- **sns.boxplot(x="White King file (column)", y="moves to win", data=df2)**

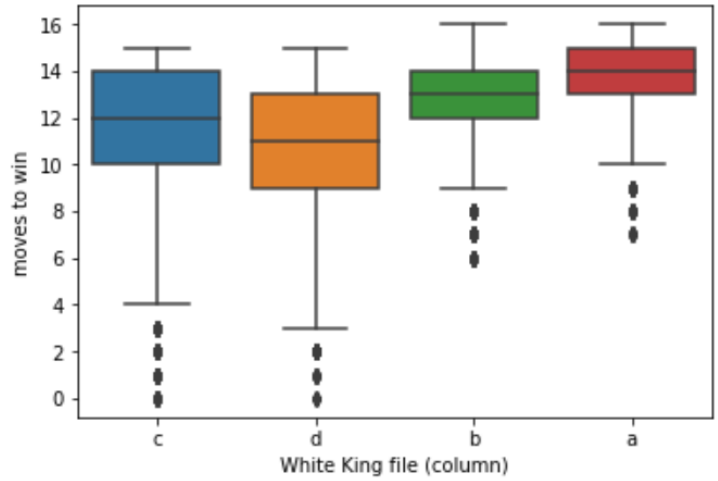

Figure – 4

The boxplot above consider only a, b, c, d files (column) to predict the moves to win and is incomplete as it has only a part of the chess board.

- **sns.boxplot(x="Black King file", y="moves to win", data=df2)**

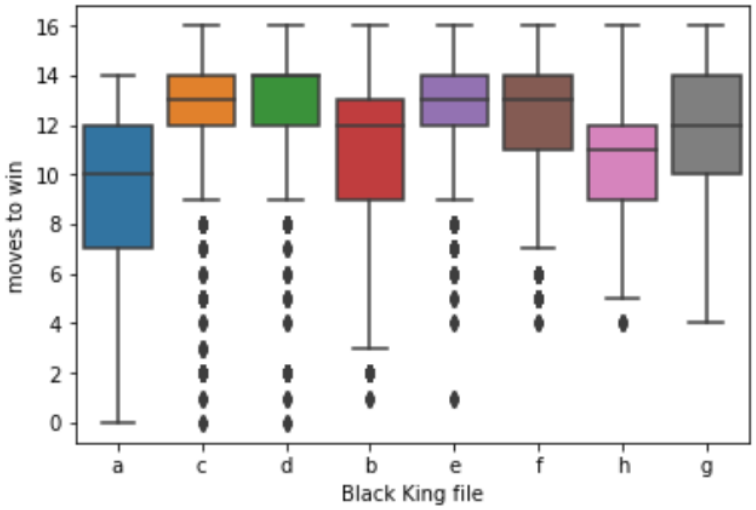

Figure – 5

The above boxplot represents the black king's number of moves for it to win the game in 0-16 moves in the columns (a, b, c…, g) of the board. It has more frequency to win in between 7 and 12 moves. We can say this is completer and more accurate when compared to the white king's plots. X-axis is black king file and y-axis is moves to win.

- **sns.boxplot(x="Black King rank", y="moves to win", data=df2)**

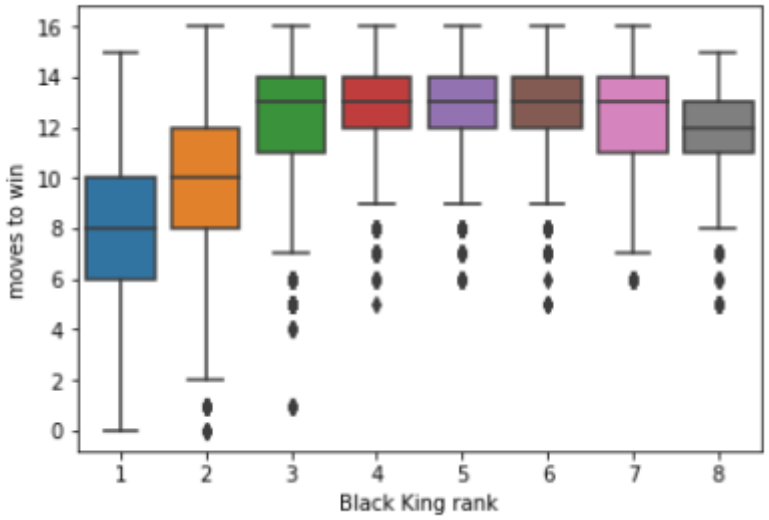

Figure – 6

The x-axis represents black king's rank and y-axis moves to win. In the above boxplot black king's moves are predicted in the following rows 1, 2, 3…, 8 and represent the number of

moves to win from one to sixteen and has the highest probability to win in 4, 5 and 6 row and in between 12 to 14 moves.

- **sns.boxplot(x="Black King rank", y="moves to win", hue ="White King file (column)", data=df2)**

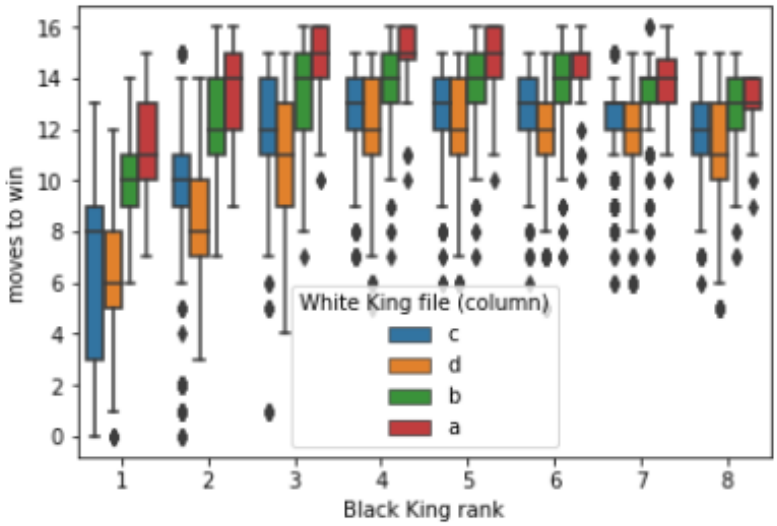

Figure – 7

This box plot is the comparison of white king's and black king's number of moves to win in the game in x-axis we represent the row in which black king moves and y-axis is the number of moves to win. As we can see the blue color represents 'c' column and has highest probability.

Relationship between position and chances of winning?

The objective is to checkmate black king with white king and rook. We require both white king and rook to checkmate black king. The checkmate happens at the back rank and more often on corner squares which is done with the black king and rook. King moves one step in any direction and Rook moves in horizontal and vertical direction (straight line), So the relationship which we observe is,
The minimum chebyshev distance from black king to corner squares is directly proportional to number of moves for checkmate.

The minimum chebyshev distance from white king to white rook corner squares is directly proportional to number of moves for checkmate.

The minimum chebyshev distance from black king to white rook, is inversely proportional to number of moves for checkmate as black king can attack the rook.

Rook is required for checkmate task as well, so we take the Manhattan distance between rook, white king and rook, black king.

3. MODELLING (BINARY CLASSIFICATION):-

Binary classification is the simplest machine learning technique which categories data into two points eg: 0 or 1, T or F, Black or White etc.
Here we have classified draw as 0 and remaining 15 moves as 1. When found the number of moves to win.

```
In [10]: mapping_y = mapping = {'zero':1,'one':1,'two':1,'three':1,'four':1,'five':1,'six':1,'seven':1,
                                'eight':1,'nine':1,'ten':1,'eleven':1,'twelve':1,'thirteen':1,'fourteen':1,
                                'fifteen':1,'sixteen':1, 'draw':0}

In [13]: model1_df['White King rank (row)'] = model1_df['White King rank (row)'].map(mapping_x)
         model1_df['White Rook rank'] = model1_df['White Rook rank'].map(mapping_x)
         model1_df['Black King rank'] = model1_df['Black King rank'].map(mapping_x)

In [14]: moves_count = model1_df['moves to win'].value_counts()
         moves_count

Out[14]: 1    25260
         0     2795
         Name: moves to win, dtype: int64
```

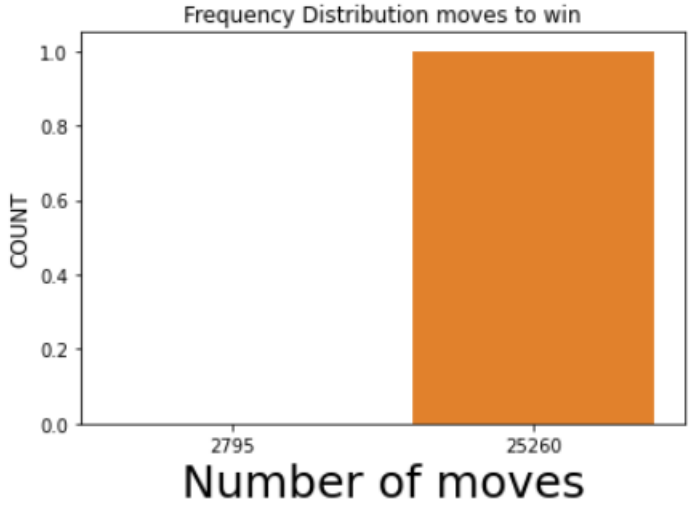

Figure – 8

We have found that 1 has more count when compared to 0 (draw). It is very obvious that the data is unbalanced, an uneven dataset is one in which the target variable has more observations in one specific class than the other class.

PYCARET:

For this dataset we have used an open source library called PyCaret which is used to train and deploy machine learning models. This library helps replace hundreds of lines of codes in a few words. This is very simple and easy to use. PyCaret provides all the models with Accuracy, AUC, Recall, Precision, F1 and Kappa.

```
In [30]: compare_models(fold = 2)
```

Out[30]:

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	Light Gradient Boosting Machine	0.994700	0.999200	0.997100	0.997000	0.997100	0.970500
1	CatBoost Classifier	0.994500	0.999700	0.996300	0.997500	0.996900	0.969300
2	Extra Trees Classifier	0.938200	0.965200	0.998800	0.936700	0.966800	0.530400
3	Decision Tree Classifier	0.927000	0.792100	0.960600	0.958500	0.959500	0.589400
4	Random Forest Classifier	0.922300	0.909600	0.985200	0.932400	0.958100	0.439400
5	Gradient Boosting Classifier	0.913700	0.974500	0.999600	0.912800	0.954300	0.221500
6	Extreme Gradient Boosting	0.908800	0.965700	0.999500	0.908400	0.951800	0.148800
7	K Neighbors Classifier	0.908400	0.825600	0.989300	0.915700	0.951100	0.245900
8	Logistic Regression	0.900400	0.629600	1.000000	0.900400	0.947600	0.000000
9	SVM - Linear Kernel	0.900400	0.000000	1.000000	0.900400	0.947600	0.000000
10	Ridge Classifier	0.900400	0.000000	1.000000	0.900400	0.947600	0.000000
11	Ada Boost Classifier	0.900400	0.629100	1.000000	0.900400	0.947600	0.000000
12	Linear Discriminant Analysis	0.900400	0.625400	1.000000	0.900400	0.947600	0.000000
13	Quadratic Discriminant Analysis	0.898800	0.921900	0.917700	0.968700	0.942000	0.535500
14	Naive Bayes	0.813400	0.634100	0.872600	0.916100	0.893800	0.126400

From the above information, accuracy is very intuitive, yet it may be a very poor measure for imbalanced data.

Why Accuracy fails for imbalanced classification?

Classification accuracy is the most-used metric for evaluating classification models (Rocca, 2019). The reason for its wide use is because it is easy to calculate, easy to interpret, and is a single number to summarize the model's capability (Rocca, 2019). When the class distribution is slightly skewed, accuracy can still be a useful metric (Rocca, 2019). When the skew in the class distributions are severe, accuracy can become an unreliable measure of model performance (Rocca, 2019). Thus, accuracy fails for imbalanced classification (Rocca, 2019).

So, which metrics can we use when our data is unbalanced?

1) <u>AUC CURVE:</u>

AUC stands for "Area under the ROC Curve." The range of AUC is from 0 to 1 (Rocca, 2019). When a model is 100% wrong then it has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.AUC provides a comprehensive measure of performance across all possible classification thresholds (Rocca, 2019). One way of interpreting AUC is by finding the probability of the model that ranks a random positive more highly than a random negative (Rocca, 2019).

AUC is suitable for the following two reasons:

- AUC is **scale invariant**. It measures how well predictions are ranked, rather than their absolute values (Rocca, 2019).

- AUC is **classification-threshold-invariant**. It helps in measuring the quality of the model's predictions irrespective of what classification threshold is chosen (Rocca, 2019).

2) Confusion matrix:

Confusion matrix is a summary of the predictions made by a classification model organized into a table by class (Rocca, 2019). Each row of the table indicates the actual class and each column represents the predicted class (Rocca, 2019). A value in the cell is a count of the number of predictions made for a class that are actually for a given class (Rocca, 2019). The cells on the diagonal represent correct predictions, where a predicted and expected class align (Rocca, 2019).

3) Dividing Data into test train

To model data we have to divide the entire dataset into Training and testing sets. The model is usually fit into a training set which contains an output. A test model is subset of the trained model which is used to assess the performance of the model. 80% of the data usually comes under training data and the remaining 20% is for testing data set.

4) ALGORITHMS:

For the King-Rook vs. King dataset we apply three supervised classification algorithms Decision Tree, Logistic Regression (Benchmark model), Random Forest.\

1. Decision Tree:

It is one the most popular machine learning algorithm used for both classification and regression tasks. The main goal of a Decision Tree is to construct a training model which can help to predict the class or value of the target variable from prior data (training data).

```
In [34]: dt = tree.DecisionTreeClassifier()

In [35]: dt.fit(X_train,y_train)

Out[35]: DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                                max_depth=None, max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort='deprecated',
                                random_state=None, splitter='best')
```

AUC:

```
In [36]: probs = dt.predict_proba(X_test)[:,1]
         fpr, tpr, thresholds = roc_curve(y_test, probs)

In [37]: auc(fpr,tpr)

Out[37]: 0.9672690344203411
```

AUC CURVE :

```
In [38]: plt.plot(fpr,tpr)
Out[38]: [<matplotlib.lines.Line2D at 0x1313ac9d2b0>]
```

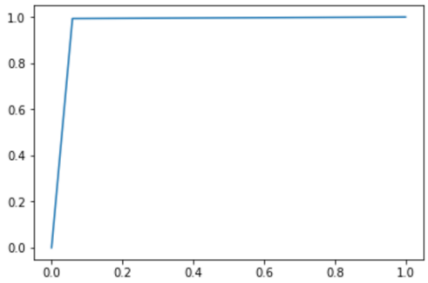

Figure – 9

This plot provides 90% of the AUC probability distinguishing the FPR and TPR.

ACCURACY:

```
In [39]: clas = dt.predict(X_test)

In [40]: acc = accuracy_score(y_test, clas)

In [41]: acc
Out[41]: 0.9882373908394225
```

CONFUSION MATRIX:

```
In [42]: confusion_matrix(y_test, clas)
Out[42]: array([[ 527,   33],
                [  33, 5018]], dtype=int64)
```

2. Logistic Regression:

Logistic regression is one of the supervised classification algorithms used for binary classification. It gives a discrete binary outcome between 0 and 1. This algorithm works by measuring the relationship between the dependent variable and one or more independent variables.

```
In [43]: lr = LogisticRegression()

In [44]: lr.fit(X_train,y_train)
Out[44]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=1, l1_ratio=None, max_iter=100,
                            multi_class='auto', n_jobs=None, penalty='l2',
                            random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                            warm_start=False)
```

AUC:

```
In [45]: probs = lr.predict_proba(X_test)[:,1]
         fpr, tpr, thresholds = roc_curve(y_test, probs)

In [46]: auc(fpr,tpr)
Out[46]: 0.6361399439997737
```

AUC CURVE :

```
In [47]: plt.plot(fpr,tpr)
Out[47]: [<matplotlib.lines.Line2D at 0x1313a2e5320>]
```

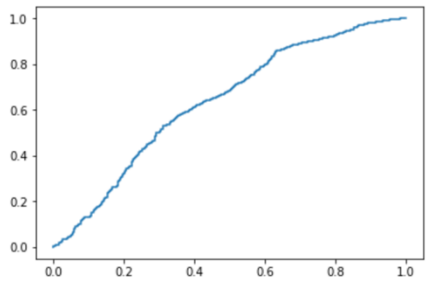

Figure – 10

From the above plot, it is difficult to find as both negatives and positives are mixed up in the graph.

ACCURACY:

```
In [48]: clas = lr.predict(X_test)

In [49]: acc = accuracy_score(y_test, clas)

In [50]: acc
Out[50]: 0.9001960434860097
```

CONFUSION MATRIX:

```
In [51]: confusion_matrix(y_test, clas)
Out[51]: array([[   0,  560],
                [   0, 5051]], dtype=int64)
```

3. Random Forest:

Random Forest is a predictive modeling algorithm which is used for both classification and regression tasks. It works well with default hyper parameter. It can be used to rank the importance of variables in a regression or classification problem.

```
In [52]: rf = RandomForestClassifier(n_estimators = 100, n_jobs= -1)

In [53]: rf.fit(X_train,y_train)

Out[53]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                                criterion='gini', max_depth=None, max_features='auto',
                                max_leaf_nodes=None, max_samples=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=100,
                                n_jobs=-1, oob_score=False, random_state=None, verbose=0,
                                warm_start=False)
```

AUC:

```
In [54]: probs = rf.predict_proba(X_test)[:,1]
         fpr, tpr, thresholds = roc_curve(y_test, probs)

In [55]: auc(fpr,tpr)

Out[55]: 0.9986645148061204
```

AUC CURVE:

```
In [56]: plt.plot(fpr,tpr)

Out[56]: [<matplotlib.lines.Line2D at 0x131338c02b0>]
```

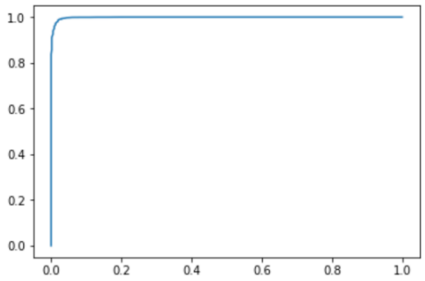

Figure – 11

From the above plot, we can distinguish between False Position rate and true positive rate clearly.

ACCURACY:

```
In [57]: clas = rf.predict(X_test)

In [58]: acc = accuracy_score(y_test, clas)

         acc

Out[58]: 0.9625735163072536
```

CONFUSION MATRIX:

```
In [59]: confusion_matrix(y_test, clas)
Out[59]: array([[ 350,  210],
                [   0, 5051]], dtype=int64)
```

CONCLUSION:

By using machine learning algorithms we can easily predict the outcomes of chess endgames. These models are trained with the dataset and gives the accuracy score as results. From all the algorithms Random Forest algorithm gives the best accuracy score in King-Rook vs. King with an AUC of 99.8% when compared to others.

4. MODELLING

MULTICLASS CLASSIFICATION WITHOUT FEATURE ENGINEERING:

In machine learning, multiclass classification is the practice of classifying into one or more classes. It is also a problem in the ground of supervised learning. The main task is the prediction of one class out of a finite set of three or more possible classes for each instance.

From the figure, we can say supervised learning is distinguished into classification and regression. First, it is classification because the values are finite but in regression it is continuous. Furthermore, the classification is divided into multi-label and single-label since each instance can be associated with any number of classes, also known as labels. Finally, the model is learned to be from a set of k classes with $k > 2$, i.e. a set $\{1,2,...,k\}$. This makes the problem slightly different and difficult than binary classification with $k = 2$. Ordinal classification is a specialization of multiclass classification.

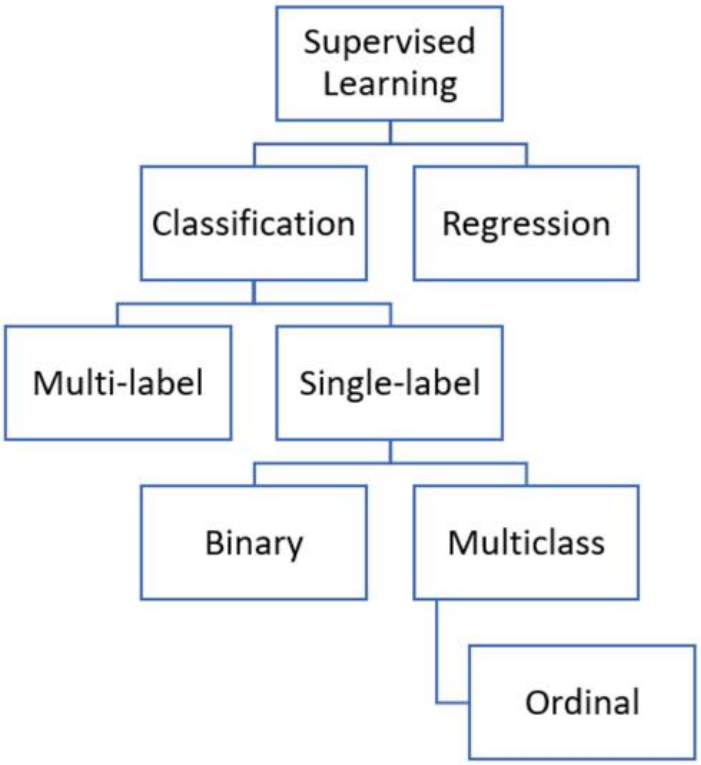

Figure – 12 (Furnkranz , Zopf, & Einreichung, 1970)

Here we have classified each move from 0 to 16 which helped us find the number of counts for each move to win.

```
In [12]: modelm_df= modelm_df[modelm_df['moves to win'] != 'draw']

In [13]: mapping_y = mapping = {'zero':0,'one':1,'two':2,'three':3,'four':4,'five':5,'six':6,'seven':7,
                                'eight':8,'nine':9,'ten':10,'eleven':11,'twelve':12,'thirteen':13,'fourteen':14,
                                'fifteen':15,'sixteen':14}

In [14]: modelm_df['moves to win'] = modelm_df['moves to win'].map(mapping_y)

In [16]: mapping_x = {1:'1',2:'2',3:'3',4:'4',5:'5',6:'6',7:'7',8:'8'}

In [17]: modelm_df['White King rank (row)'] = modelm_df['White King rank (row)'].map(mapping_x)
         modelm_df['White Rook rank'] = modelm_df['White Rook rank'].map(mapping_x)
         modelm_df['Black King rank'] = modelm_df['Black King rank'].map(mapping_x)

In [18]: moves_count = modelm_df['moves to win'].value_counts()
         moves_count

Out[18]: 14    4943
         13    4194
         12    3597
         11    2854
         15    2166
         10    1985
         9     1712
         8     1433
         7      683
         6      592
         5      471
         2      246
         4      198
         3       81
         1       78
         0       27
         Name: moves to win, dtype: int64
```

PYCARET:

```
In [39]: compare_models(fold = 2)
```

Out[39]:

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	CatBoost Classifier	0.684400	0.000000	0.679100	0.684100	0.683100	0.639400
1	Extra Trees Classifier	0.682800	0.000000	0.651700	0.681900	0.680900	0.637200
2	Light Gradient Boosting Machine	0.658800	0.000000	0.609900	0.659300	0.657300	0.609900
3	Random Forest Classifier	0.597000	0.000000	0.525000	0.594300	0.592200	0.537600
4	Decision Tree Classifier	0.568500	0.000000	0.506400	0.634900	0.580700	0.509600
5	K Neighbors Classifier	0.539600	0.000000	0.440200	0.539900	0.531700	0.469500
6	Gradient Boosting Classifier	0.497500	0.000000	0.463200	0.499400	0.486900	0.420000
7	Extreme Gradient Boosting	0.473200	0.000000	0.447900	0.472100	0.456500	0.389500
8	Logistic Regression	0.388100	0.000000	0.286900	0.368200	0.364100	0.288900
9	Ada Boost Classifier	0.381400	0.000000	0.295400	0.367400	0.361900	0.281600
10	Linear Discriminant Analysis	0.363000	0.000000	0.305200	0.338300	0.335300	0.258700
11	Ridge Classifier	0.329200	0.000000	0.147900	0.263800	0.265500	0.205600
12	SVM - Linear Kernel	0.327100	0.000000	0.322700	0.318900	0.295900	0.223800
13	Quadratic Discriminant Analysis	0.198600	0.000000	0.199800	0.289900	0.183900	0.115400
14	Naive Bayes	0.153300	0.000000	0.153600	0.213600	0.100600	0.088100

From the above information for modelling 2.2 we have considered only the accuracy and multilabel confusion matrix. of a model rather than AUC because the value seems to be zero for all the models.

MULTI-LABEL CONFUSION MATRIX :

The multi-label confusion matrix helps in calculating class-wise or sample-wise multilabel confusion matrices, and in multiclass tasks, labels are binarized while confusion matrix calculates matrix between every two classes in one confusion matrix.

ALGORITHMS:

In this modelling we apply three multiclass supervised classification algorithms with no feature engineering. The algorithms are Decision Tree, Logistic Regression (Benchmark model), Random Forest.

1.Decision Tree:

```
In [42]: dt =  OneVsRestClassifier(tree.DecisionTreeClassifier())

In [43]: dt.fit(X_train,y_train)

Out[43]: OneVsRestClassifier(estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                               class_weight=None,
                                                               criterion='gini',
                                                               max_depth=None,
                                                               max_features=None,
                                                               max_leaf_nodes=None,
                                                               min_impurity_decrease=0.0,
                                                               min_impurity_split=None,
                                                               min_samples_leaf=1,
                                                               min_samples_split=2,
                                                               min_weight_fraction_leaf=0.0,
                                                               presort='deprecated',
                                                               random_state=None,
                                                               splitter='best'),
                             n_jobs=None)
```

ACCURACY:

```
In [44]: probs = dt.predict_proba(X_test)

In [45]: clas = dt.predict(X_test)

In [46]: acc = accuracy_score(y_test, clas)
         acc

Out[46]: 0.7697941409342834
```

MULTI-LABEL CONFUSION MATRIX:

```
In [47]: multilabel_confusion_matrix(y_test, clas)

Out[47]: array([[[5048,    0],
                 [   1,    3]],

                [[5031,    0],
                 [   3,   18]],

                [[4998,    1],
                 [   6,   47]],

                [[5028,    2],
                 [   5,   17]],

                [[5014,    0],
                 [   8,   30]],

                [[4957,    2],
                 [  32,   61]],

                [[4922,   20],
                 [  18,   92]],

                [[4872,   23],
                 [  55,  102]],

                [[4727,   26],
                 [  77,  222]],

                [[4627,   54],
                 [ 104,  267]],

                [[4562,   82],
                 [ 128,  280]],

                [[4428,   74],
                 [ 136,  414]],

                [[4228,  127],
                 [ 169,  528]],

                [[4056,  138],
                 [ 235,  623]],

                [[3928,  133],
                 [ 170,  821]],

                [[4191,  481],
                 [  16,  364]]], dtype=int64)
```

2. Logistic Regression:

```
In [48]: lr =  OneVsRestClassifier(LogisticRegression(multi_class='ovr'))

In [49]: lr.fit(X_train,y_train)

Out[49]: OneVsRestClassifier(estimator=LogisticRegression(C=1.0, class_weight=None,
                                                          dual=False, fit_intercept=True,
                                                          intercept_scaling=1,
                                                          l1_ratio=None, max_iter=100,
                                                          multi_class='ovr', n_jobs=None,
                                                          penalty='l2',
                                                          random_state=None,
                                                          solver='lbfgs', tol=0.0001,
                                                          verbose=0, warm_start=False),
                             n_jobs=None)
```

ACCURACY:

```
In [50]: probs = lr.predict_proba(X_test)

In [51]: clas = lr.predict(X_test)

In [52]: acc = accuracy_score(y_test, clas)
         acc

Out[52]: 0.4075613618368963
```

MULTI-LABEL CONFUSION MATRIX:

```
In [53]: multilabel_confusion_matrix(y_test, clas)

Out[53]: array([[[5048,    0],
                 [   4,    0]],

                [[5029,    2],
                 [  20,    1]],

                [[4962,   37],
                 [  19,   34]],

                [[5030,    0],
                 [  19,    3]],

                [[5005,    9],
                 [  25,   13]],

                [[4915,   44],
                 [  55,   38]],

                [[4880,   62],
                 [  68,   42]],

                [[4865,   30],
                 [ 154,    3]],

                [[4499,  254],
                 [ 152,  147]],

                [[4515,  166],
                 [ 293,   78]],

                [[4544,  100],
                 [ 390,   18]],

                [[4157,  345],
                 [ 433,  117]],

                [[3923,  432],
                 [ 420,  277]],

                [[3595,  599],
                 [ 445,  413]],

                [[3280,  781],
                 [ 273,  718]],

                [[4540,  132],
                 [ 223,  157]]], dtype=int64)
```

---

3. Random Forest:

```
In [54]: rf = OneVsRestClassifier(RandomForestClassifier(n_estimators=500, n_jobs=-1))

In [55]: rf.fit(X_train,y_train)

Out[55]: OneVsRestClassifier(estimator=RandomForestClassifier(bootstrap=True,
                                                              ccp_alpha=0.0,
                                                              class_weight=None,
                                                              criterion='gini',
                                                              max_depth=None,
                                                              max_features='auto',
                                                              max_leaf_nodes=None,
                                                              max_samples=None,
                                                              min_impurity_decrease=0.0,
                                                              min_impurity_split=None,
                                                              min_samples_leaf=1,
                                                              min_samples_split=2,
                                                              min_weight_fraction_leaf=0.0,
                                                              n_estimators=500,
                                                              n_jobs=-1, oob_score=False,
                                                              random_state=None,
                                                              verbose=0,
                                                              warm_start=False),
                             n_jobs=None)
```

ACCURACY:

```
In [56]: probs = rf.predict_proba(X_test)

In [57]: clas = rf.predict(X_test)

In [58]: acc = accuracy_score(y_test, clas)
         acc

Out[58]: 0.8046318289786223
```

MULTI-LABEL CONFUSION MATRIX:

```
In [59]: multilabel_confusion_matrix(y_test, clas)

Out[59]: array([[[5048,    0],
                 [   0,    4]],

                [[5031,    0],
                 [   1,   20]],

                [[4997,    2],
                 [   1,   52]],

                [[5028,    2],
                 [   4,   18]],

                [[5013,    1],
                 [   7,   31]],

                [[4947,   12],
                 [  14,   79]],

                [[4915,   27],
                 [  15,   95]],

                [[4869,   26],
                 [  48,  109]],

                [[4693,   60],
                 [  53,  246]],

                [[4592,   89],
                 [  97,  274]],

                [[4556,   88],
                 [ 117,  291]],

                [[4391,  111],
                 [ 117,  433]],

                [[4213,  142],
                 [ 119,  578]],

                [[4040,  154],
                 [ 186,  672]],

                [[3873,  188],
                 [ 144,  847]],

                [[4587,   85],
                 [  64,  316]]], dtype=int64)
```

CONCLUSION FOR MODELLING 3:

From the algorithms, the accuracy results obtained are

Accuracy using decision tree – 76.9%

Accuracy using Logistic Regression – 40.7%

Accuracy using Random Forest – 80.4%

From all the algorithms found, it is proved that Random Forest gives us best accuracy score in King-Rook vs. King.

5. MODELLING

MULTICLASS CLASSIFICATION WITH FEATURE ENGINEERING:

What and Why is Feature Engineering Important?

The process of using domain knowledge to extract raw data via data mining techniques is known as feature engineering (Rathna, Ken, & Ambi, 2012). Engineering good features allows us to represent the structure of the data accurately thereby creating a better model (Rathna, Ken, & Ambi, 2012). Features can be engineered by decomposing or splitting features from external sources or by combining them into new features (Rathna, Ken, & Ambi, 2012). Feature engineering can be considered as applied machine learning itself (Rathna, Ken, & Ambi, 2012). Feature engineering also helps us solve issues like,

- Missing values can be imputed in data cleaning (Rathna, Ken, & Ambi, 2012).
- Normalize or standardize the data if they do not belong to the same dimension (Rathna, Ken, & Ambi, 2012).
- Filtering out redundant information in feature selection (Rathna, Ken, & Ambi, 2012).

Since we are carrying out our modelling on chess dataset, we are using Manhattan distance and Chebyshev distance.

MANHATTAN DISTANCE:

Manhattan distance is defined as the distance between two points measured along axes at right angles (IEEE, n.d.). In a plane with p1 at (x1, y1) and p2 at (x2, y2), it is |x1 - x2| + |y1 - y2| (IEEE, n.d.).

Manhattan distance is the rectilinear distance between two points (IEEE, n.d.). For example, if we look at the figure below the purple line depicts the Manhattan distance between the two points A and B (IEEE, n.d.).

Figure -13 (IEEE, n.d.)

In chess, the distance between squares on the chessboard for rooks is measured in Manhattan distance [4].

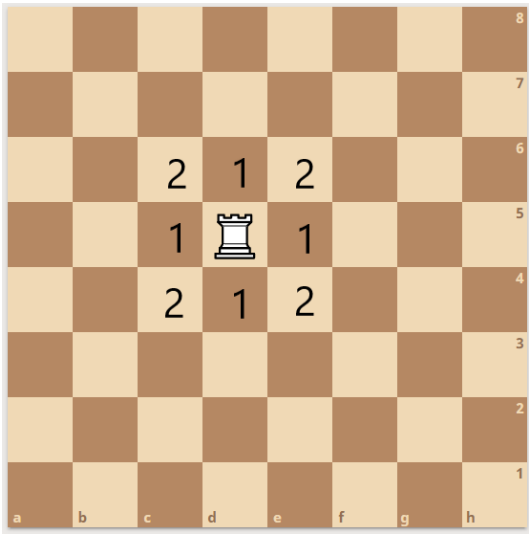

Figure – 14

For example, if we place the rook in column d and row 5 the Manhattan distance to column e and row 6 is 2 (as shown in the above figure).

Chebyshev Distance:

The Chebyshev distance is also known as chessboard distance, as in the chess game it is the minimum number of moves needed by a king to go from one square on the chess board to another (IEEE, n.d.).

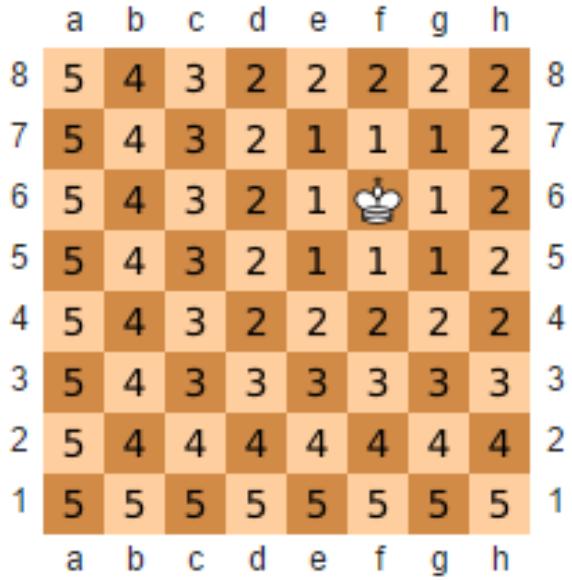

Figure – 15

For example, if we place the king in column g and row 6 the Chebyshev distance to the square at column b and row 5 is 4 (as shown in the above figure).

Implementing Manhattan and Chebyshev distance.

```
In [18]: def manhattan(x1,x2,y1,y2):
             n = len(x1)
             distance = []
             for i in range(0,n):
                 dis = abs(x2[i]-x1[i]) + abs(y2[i]-y1[i])
                 distance.append(dis)
             return distance
```

```
In [19]: def chebyshev(x1,x2,y1,y2):
             n = len(x1)
             distance = []
             for i in range(0,n):
                 dis = max(abs(x2[i]-x1[i]) , abs(y2[i]-y1[i]))
                 distance.append(dis)
             return distance
```

After using Manhattan and Chebyshev we can find the distances between white and black Kings with their ranks.

In [25]:
```
modelm_df.head()
```

Out[25]:

	White King file (column)	White King rank (row)	White Rook file	White Rook rank	Black King file	Black King rank	white King file num	White Rook file num	Black King file num	che dis between kings	che dis between WK R	che dis between BK R
0	c	1	a	3	a	1	3	1	1	2	2	2
1	c	1	a	4	a	1	3	1	1	2	3	3
2	c	1	a	5	a	1	3	1	1	2	4	4
3	c	1	a	6	a	1	3	1	1	2	5	5
4	c	1	a	7	a	1	3	1	1	2	6	6

PYCARET :

In [46]:
```
compare_models(fold = 2)
```

Out[46]:

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa
0	CatBoost Classifier	0.673500	0.000000	0.666900	0.673400	0.672200	0.626800
1	Light Gradient Boosting Machine	0.642300	0.000000	0.624900	0.642200	0.640500	0.590800
2	Extra Trees Classifier	0.613600	0.000000	0.551600	0.611700	0.610600	0.557700
3	Random Forest Classifier	0.547100	0.000000	0.476300	0.543300	0.542000	0.480400
4	Decision Tree Classifier	0.501600	0.000000	0.466000	0.576500	0.515300	0.434100
5	Gradient Boosting Classifier	0.499100	0.000000	0.498800	0.500900	0.490000	0.422100
6	Extreme Gradient Boosting	0.476900	0.000000	0.473200	0.479200	0.464500	0.394800
7	K Neighbors Classifier	0.429200	0.000000	0.316900	0.430800	0.417900	0.340000
8	Logistic Regression	0.408600	0.000000	0.331500	0.395100	0.392300	0.316100
9	Ada Boost Classifier	0.392300	0.000000	0.319800	0.379700	0.377000	0.296600
10	Linear Discriminant Analysis	0.383400	0.000000	0.348100	0.366900	0.364700	0.288100
11	SVM - Linear Kernel	0.353600	0.000000	0.342100	0.349800	0.342300	0.261200
12	Ridge Classifier	0.353600	0.000000	0.192500	0.338400	0.304400	0.238800
13	Naive Bayes	0.161200	0.000000	0.174800	0.253300	0.113400	0.090100
14	Quadratic Discriminant Analysis	0.151200	0.000000	0.183200	0.223400	0.135100	0.081600

1. Decision Tree:

```
In [49]: dt =  OneVsRestClassifier(tree.DecisionTreeClassifier())

In [50]: dt.fit(X_train,y_train)

Out[50]: OneVsRestClassifier(estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                               class_weight=None,
                                                               criterion='gini',
                                                               max_depth=None,
                                                               max_features=None,
                                                               max_leaf_nodes=None,
                                                               min_impurity_decrease=0.0,
                                                               min_impurity_split=None,
                                                               min_samples_leaf=1,
                                                               min_samples_split=2,
                                                               min_weight_fraction_leaf=0.0,
                                                               presort='deprecated',
                                                               random_state=None,
                                                               splitter='best'),
                                 n_jobs=None)
```

ACCURACY:

```
In [51]: probs = dt.predict_proba(X_test)

In [52]: clas = dt.predict(X_test)

In [53]: acc = accuracy_score(y_test, clas)
         acc

Out[53]: 0.7448535233570863
```

MULTI-LABEL CONFUSION MATRIX:

```
In [54]: multilabel_confusion_matrix(y_test, clas)
Out[54]: array([[[5045,    0],
                 [   2,    5]],

                [[5038,    0],
                 [   2,   12]],

                [[5009,    0],
                 [   5,   38]],

                [[5034,    3],
                 [   2,   13]],

                [[5009,    0],
                 [  11,   32]],

                [[4951,    7],
                 [  22,   72]],

                [[4934,    6],
                 [  34,   78]],

                [[4896,   18],
                 [  63,   75]],

                [[4736,   29],
                 [  89,  198]],

                [[4651,   63],
                 [ 124,  214]],

                [[4599,   79],
                 [ 152,  222]],

                [[4380,  102],
                 [ 146,  424]],

                [[4192,  119],
                 [ 203,  538]],

                [[4084,  137],
                 [ 217,  614]],

                [[3917,  125],
                 [ 190,  820]],

                [[4016,  601],
                 [  27,  408]]], dtype=int64)
```

2. Logistic Regression:

```
In [55]: lr =  OneVsRestClassifier(LogisticRegression(multi_class='ovr'))

In [56]: lr.fit(X_train,y_train)
Out[56]: OneVsRestClassifier(estimator=LogisticRegression(C=1.0, class_weight=None,
                                                          dual=False, fit_intercept=True,
                                                          intercept_scaling=1,
                                                          l1_ratio=None, max_iter=100,
                                                          multi_class='ovr', n_jobs=None,
                                                          penalty='l2',
                                                          random_state=None,
                                                          solver='lbfgs', tol=0.0001,
                                                          verbose=0, warm_start=False),
                             n_jobs=None)
```

ACCURACY:

```
In [57]: probs = lr.predict_proba(X_test)

In [58]: clas = lr.predict(X_test)

In [59]: acc = accuracy_score(y_test, clas)
         acc

Out[59]: 0.3988519398258116
```

MULTI-LABEL CONFUSION MATRIX:

```
In [60]: multilabel_confusion_matrix(y_test, clas)

Out[60]: array([[[5045,    0],
                 [   4,    3]],

                [[5038,    0],
                 [  14,    0]],

                [[4976,   33],
                 [   9,   34]],

                [[5032,    5],
                 [  10,    5]],

                [[4996,   13],
                 [  26,   17]],

                [[4915,   43],
                 [  46,   48]],

                [[4893,   47],
                 [  74,   38]],

                [[4891,   23],
                 [ 132,    6]],

                [[4559,  206],
                 [ 150,  137]],

                [[4549,  165],
                 [ 249,   89]],

                [[4532,  146],
                 [ 345,   29]],

                [[4201,  281],
                 [ 456,  114]],

                [[3880,  431],
                 [ 501,  240]],

                [[3560,  661],
                 [ 440,  391]],

                [[3205,  837],
                 [ 291,  719]],

                [[4471,  146],
                 [ 290,  145]]], dtype=int64)
```

3. Random Forest:

```
In [61]: rf = OneVsRestClassifier(RandomForestClassifier(n_estimators=1000, n_jobs=-1))

In [62]: rf.fit(X_train,y_train)

Out[62]: OneVsRestClassifier(estimator=RandomForestClassifier(bootstrap=True,
                                                              ccp_alpha=0.0,
                                                              class_weight=None,
                                                              criterion='gini',
                                                              max_depth=None,
                                                              max_features='auto',
                                                              max_leaf_nodes=None,
                                                              max_samples=None,
                                                              min_impurity_decrease=0.0,
                                                              min_impurity_split=None,
                                                              min_samples_leaf=1,
                                                              min_samples_split=2,
                                                              min_weight_fraction_leaf=0.0,
                                                              n_estimators=1000,
                                                              n_jobs=-1, oob_score=False,
                                                              random_state=None,
                                                              verbose=0,
                                                              warm_start=False),
                             n_jobs=None)
```

ACCURACY :

```
In [63]: probs = rf.predict_proba(X_test)

In [64]: clas = rf.predict(X_test)

In [65]: acc = accuracy_score(y_test, clas)
         acc

Out[65]: 0.7893903404592241
```

MULTI-LABEL CONFUSION MATRIX:

```
In [66]: multilabel_confusion_matrix(y_test, clas)
Out[66]: array([[[5045,    0],
                 [   4,    3]],

                [[5038,    0],
                 [   1,   13]],

                [[5004,    5],
                 [   1,   42]],

                [[5033,    4],
                 [   6,    9]],

                [[5007,    2],
                 [   6,   37]],

                [[4942,   16],
                 [  14,   80]],

                [[4914,   26],
                 [  26,   86]],

                [[4895,   19],
                 [  38,  100]],

                [[4711,   54],
                 [  67,  220]],

                [[4630,   84],
                 [  81,  257]],

                [[4604,   74],
                 [ 127,  247]],

                [[4319,  163],
                 [ 117,  453]],

                [[4151,  160],
                 [ 162,  579]],

                [[4053,  168],
                 [ 179,  652]],

                [[3840,  202],
                 [ 140,  870]],

                [[4530,   87],
                 [  95,  340]]], dtype=int64)
```

CONCLUSION FOR MODELLING 5:

From the algorithms, the accuracy results obtained are

Accuracy using decision tree – 74.4%

Accuracy using Logistic Regression – 39.8%

Accuracy using Random Forest – 78.9%

By using featuring engineering, it is proved that Random Forest gives us best accuracy score in King-Rook vs. King.

6. Conclusion:

Who will benefitted from the project?

Surely this is not the project to help the Grandmaster and International Master. This is helpful for the people who is new in the chess and wants to learn how to checkmate with the king and rook, although project doesn't give moves but gives the optimal moves which is required to checkmate the king with king and rook.

7. Libraries Used:

1. **Numpy**
   NumPy¶. (n.d.). Retrieved from https://numpy.org/
2. **Pandas**
   pandas. (n.d.). Retrieved from https://pandas.pydata.org/
3. **Sklearn**
   learn. (n.d.). Retrieved from https://scikit-learn.org/stable/
4. **Pycaret**
   Home. (2020, May 8). Retrieved from https://pycaret.org/
5. **Seaborn**
   statistical data visualization¶. (n.d.). Retrieved from https://seaborn.pydata.org/
6. **Matplotlib**
   Visualization with Python¶. (n.d.). Retrieved from https://matplotlib.org/

8. Data set:

1) (n.d.). Retrieved from https://archive.ics.uci.edu/ml/datasets/Chess%20(King-Rook%20vs.%20King)

9. References

Furnkranz , J., Zopf, M., & Einreichung, T. d. (1970, 1 1). *Semantic Scholar*. Retrieved 4 25, 2020 from https://www.semanticscholar.org/paper/Sentiment-Classification-of-Chess-Annotations-Fürnkranz-Zopf/ba6e901d378f282efa8ccc4eaaef521d0406135d

IEEE. (n.d.). *IEEE*. Retrieved 04 25, 2020 from http://www.ieee.ma/uaesb/pdf/distances-in-classification.pdf

Rathna, Ken, & Ambi. (2012, 22 May). *Wordpress*. Retrieved 04 25, 2020 from https://lyfat.wordpress.com/2012/05/22/euclidean-vs-chebyshev-vs-manhattan-distance/

Rocca, B. (2019, March 30). *Toward data science*. Retrieved April 25, 2020 from https://towardsdatascience.com/handling-imbalanced-datasets-in-machine-learning-7a0e84220f28