Capstone Project

Tic-Tac-Toe Endgame Data Set

Machine Learning Engineer Nanodegree

Allugundu Mahammad Suhel

February 7th, 2019

1.Definition

Project overview:

Tic-tac-toe (American English), **noughts and crosses** (British English) or **Xs and Os**, is a paper-and-pencil game for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

This project is based on game give the outcome of the results configurations at the end of tic-tac-toe games. This gives accurate results regarding the classification of possible configurations of tic-tac-toe game. It takes the data from the UCI machine learning (https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame.)

The aim of this project is Binary classification task on possible configurations of tic-tac-toe game.

Problem Statement

The main aim of my project is to predict the Binary classification task on possible configurations of tic-tac-toe game. For doing this I selected the dataset from UCI

(https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame)

So my goal is to predict the outcome of the game by using the given attributes and the class labels.

Here class labels are positive and negative . Here I am using classification models to find the accuracy of each model and select the model which will have high accuracy. Here the input parameter is given in the form of training data.

Metrics

I want to use accuracy score as an evaluation metric for the prediction of credit approval. In the data set the class labels (+,-) are very closely balanced so we can use accuracy score as the evaluation metric. Here I am predicting the accuracy score of the selected models. We will select a model whose accuracy score is greater than all the other models and we treat it as the best.

For finding out the accuracy we have the following formula:

Accuracy Score= TP+TN/TP+FP+FN+TN

True Positives: are the values which are correctly predicted as positives

True Negatives: are the values which are correctly classified as negatives.

False Positives: are the values which are wrongly classified as positives. These are also type-1 errors.

False Negatives: are the values which are wrongly classified as negatives. These are also called as type-2 errors.

II. Analysis

Data Exploration

	TL	TM	TR	ML	MM	MR	BL	BM	BR	class
count	958	958	958	958	958	958	958	958	958	**958**
unique	3	3	3	3	3	3	3	3	3	**2**
top	x	x	x	x	x	x	x	x	x	**positive**
freq	**418**	**378**	**418**	**378**	**458**	**378**	**418**	**378**	**418**	**626**

Attributes

The data set containing following attributes:

- TL : top left square {x, o, b}
- TM : top middle square {x, o, b}
- TR : top right square {x, o, b}
- ML : middle left square {x, o, b}
- MM : middle middle square {x, o, b}
- MR : middle right square {x, o, b}
- BL : bottom left square {x, o, b}
- BM : bottom middle square {x, o, b}
- BR : bottom right square {x, o, b}
- class :{positive, negative}

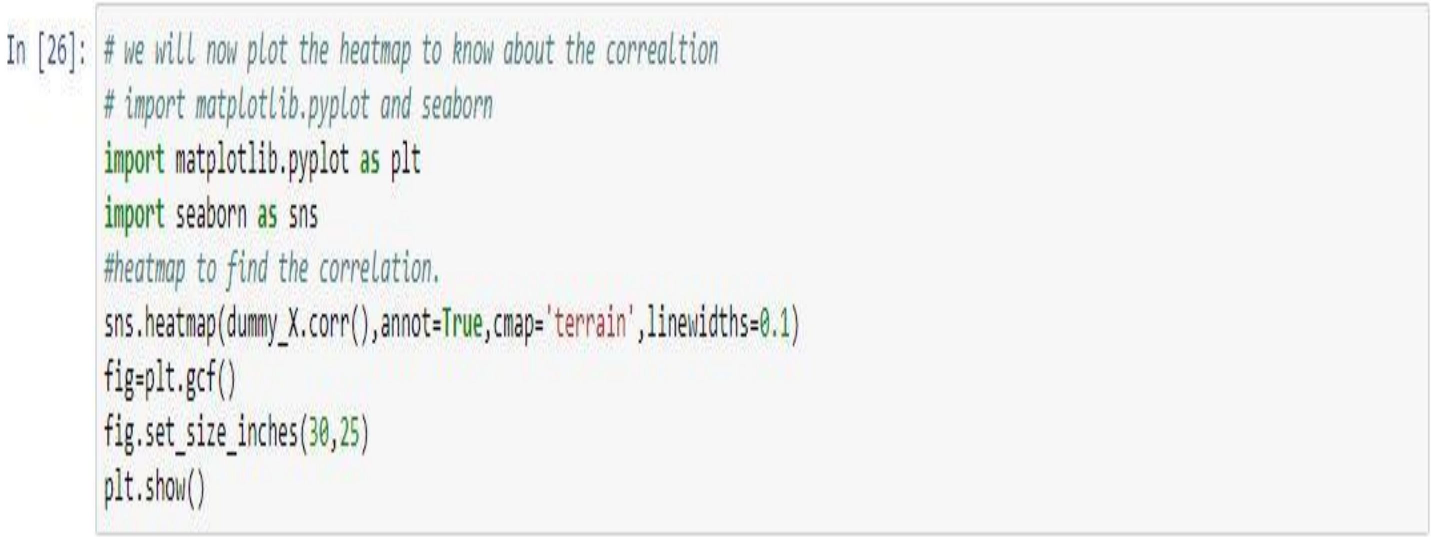
A snapshot of the dataset containing first 5 rows is as follow:

	top_left_sqr	top_middle_sqr	top_right_sqr	mid_left_sqr	mid_mid_sqr	mid_right_sqr	btm_left_sqr	btm_mid_sqr	btm_right_sqr	class
0	x	x	x	x	o	o	x	o	o	**positive**
1	x	x	x	x	o	o	o	x	o	**positive**
2	x	x	x	x	o	o	o	o	x	**positive**
3	x	x	x	x	o	o	o	b	b	**positive**
4	**x**	**x**	**x**	**x**	**o**	**o**	**b**	**o**	**b**	**positive**

Heat map:
The heat map is a 2-D representation of data in which values are represented by colours. A simple heat map provides the immediate visual summary of information. More elaborate heat maps allow the user to understand complex data.

Here we will know the correlation the attributes like top left square, top middle square ,top right square , middle left square ,middle middle square , middle right square, bottom left square , bottom middle square ,bottom right square , class

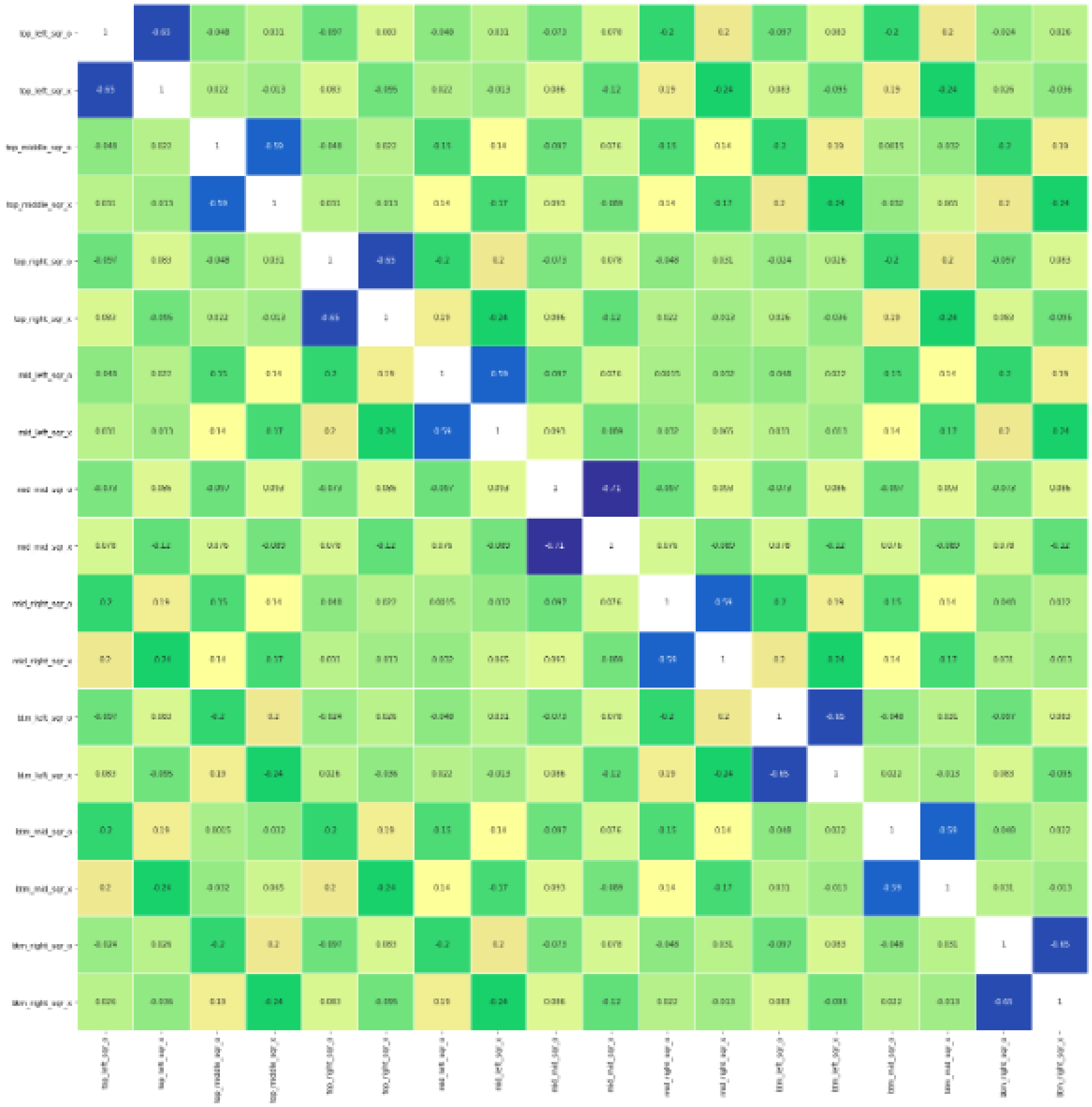
By using the below code we will generate the heat map between the attributes and try to deduce the correlation between the attributes. We will use the following code to do so.
By



In [26]:

```
# we will now plot the heatmap to know about the correaltion
# import matplotlib.pyplot and seaborn
import matplotlib.pyplot as plt
import seaborn as sns
#heatmap to find the correlation.
sns.heatmap(dummy_X.corr(),annot=True,cmap='terrain',linewidths=0.1)
fig=plt.gcf()
fig.set_size_inches(30,25)
plt.show()
```

By executing the above code we get the following heat map:



(For a more clear heat map refer to the Code shown in the above that was executed in the project)

From the above heat map we can clearly say that there is no much correlation between the attributes.

Algorithms and techniques:

Here mainly we will use three algorithms.

1.Naive Bayes .(Benchmark Model)

2. Logistic Regression.

3.Decision Tree.

4. Random Forest.

Naive Bayes

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.
The fundamental Naive Bayes assumption is that each feature makes an:

- independent
- equal contribution to the outcome.

Advantages:

- Very simple, easy to implement and fast.
- If the NB conditional independence assumption holds, then it will converge quicker than discriminative models like logistic regression.
- Even if the NB assumption doesn't hold, it works great in practice.
- Need less training data.
- Highly scalable. It scales linearly with the number of predictors and data points.
- Can be used for both binary and multiclass classification problems.
- Can make probabilistic predictions.
- Handles continuous and discrete data.
- Not sensitive to irrelevant features.

Disadvantages:

1 .The first disadvantage is that the Naive Bayes classifier makes a very strong assumption on the shape of your data distribution, i.e. any two features are independent given the output class. Due to this, the result can be potentially very bad - hence, a “naive” classifier. This is not as terrible as people generally think, because the NB classifier can be optimal even if the assumption is violated, and its results can be good even in the case of sub-optimality.
2. Another problem happens due to data scarcity. For any possible value of a feature, you need to estimate a likelihood value by a frequents approach. This can result in probabilities going towards 0 or 1, which in turn leads to numerical instabilities and worse results. In this case, you need to smooth in some way your probabilities, or to impose some prior on your data, however you may argue that the resulting classifier is not naive anymore. 3. A third problem arises for continuous features. It is common to use a binning procedure to make them discrete, but if you are not careful you can throw away a lot of information.

Parameters:
alpha : float, optional (default=1.0), fit_prior : boolean, optional (default=True), class_prior : array-like, size (n_classes,), optional (default=None), X : {array-like, sparse matrix}, shape = [n_samples, n_features], y : array-like, shape = [n_samples], sample_weight : array-like, shape = [n_samples], (default=None), deep : boolean, optional, classes : arraylike, shape = [n_classes] (default=None)

Logistic Regression:

Logistic Regression is a Machine Learning classification algorithm that is used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.). In other words, the logistic regression model predicts P(Y=1) as a function of X.

Advantages:
Because of its efficient and straightforward nature, doesn't require high computation power, easy to implement, easily interpretable, used widely by data analyst and scientist. Also, it doesn't require scaling of features. Logistic regression provides a probability score for observations.

Disadvantages:

Logistic regression is not able to handle a large number of categorical features/variables. It is vulnerable to over fitting. Also, can't solve the non-linear problem with the logistic regression that is why it requires a transformation of non-linear features. Logistic regression will not perform well with independent variables that are not correlated to the target variable and are very similar or correlated to each other.

Parameters:

Class sklearn.linear_model.LogisticRegression (penalty=’l2’, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver=’warn’, max_iter=100, multi_class=’warn’, verbose=0, warm_start=False, n_jobs=None)
The application is classification oriented . So, techniques that are used are taken from Classification techniques

Decision Trees:

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predict the value.

Some advantages of decision trees are:

Simple to understand and to interpret. Trees can be visualized. Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values. The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree. Able to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of variable. See algorithms for more information. Able to handle multi-output problems. Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret. Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model. Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

Decision-tree learners can create over-complex trees that do not generalize the data well. This is called over fitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem. Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble. The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement. There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems. Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Parameters:

Class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

Random Forest

Tree models are known to be high variance, low bias models. In consequence, they are prone to over fit the training data. This is catchy if we recapitulate what a tree model does if we do not prune it or introduce early stopping criteria like a minimum number of instances per leaf node. Well, it tries to split the data along the features until the instances are pure regarding the value of the target feature, there are no data left, or there are no features left to spit the dataset on. If one of the above holds true, we grow a leaf node. The consequence is that the tree model is grown to the maximal depth and therewith tries to reshape the training data as precise as possible which can easily lead to over fitting. Another drawback of classical tree models like the (ID3 or CART) is that they are relatively unstable. This instability can lead to the situation that a small change in the composition of the dataset leads to a completely different tree model.

Parameters:
Class sklearn.ensemble.RandomForestClassifier (n_estimators=’warn’, criterion=’gini’, max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=’auto’, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None

Advantages:
1. Reduction in over fitting: by averaging several trees, there is a significantly lower risk of over fitting.
2. Less variance: By using multiple trees, you reduce the chance of stumbling across a classifier that doesn’t perform well because of the relationship between the train and test data.

Disadvantages:
1. It takes more time to train samples.

Benchmark Model:

For this problem we will choose Naive Bayes model as the benchmark model. By applying this Naive Bayes we achieved an accuracy of 0.718 that is 71.8%. Now we will try and achieve better accuracy than this model by using the above mentioned classification models.

III. Methodology

Data Pre-processing:

In this step we will pre process the data. Data pre-processing is considered to be the first and foremost step that is to be done before starting any process. We will read the data by using read_csv. Then we will know the shape of the data. After that we will know the unique class attributes by using unique (). And by using the info () we will know the information of the attributes. Then we will check whether there are any null values by using isnull ().

LabelEncoder can be used to normalize labels. We will import LabelEncoder from sklearn.preprocessing we will also use fit_transform(y) for Fit label encoder and return encoded labels. After doing that we will use le.transform () for Transform labels to normalized encoding.

After that we will divide the whole data into training and testing data. We will assign 70% of the data to the training data and the remaining 30% of the data into testing data. We will do this by using train_test_split from sklearn.model_selection.

Implementation

Out of the chosen algorithms we will first take Random Forest classification model. We will take a classifier and fit the training data. After that we will predict that by using predict (X_train). Now we will predict the accuracy of the testing data by using accuracy_score (y_test, pred).

By doing so for the Naive bayes which is the benchmark will give us the accuracy of 0.718.

We will now choose the algorithm which will give us the better score than the benchmark model out of Decision tree ,Logistic Regression and Random Forest . By following the same procedure above that is fitting, predicting and finding the accuracy score we will get the accuracy score as bellow.
Logistic Regression:0.982
Decision Tree:0.954
Random Forest: 0.986
From the above reports Random Forest seems to be performing well.

Complications:

While building the Random Forest Classifier I faced a complication while tuning the parameters. Firstly we should come to a conclusion about the parameters we are going to tune. Then by varying the range of values of the chosen parameter we should be able to get the parameter value which gives us the better accuracy than the unturned value. So it is complicated to chose the parameter that is to be tuned in this case n_estimator.

Refinement

I found out random forest as the best classifier out of the chosen classifiers. Now we will perform tuning of random forest classifier in order to achieve the better accuracy. For this we will use GridSearchCV.
For doing refinement we will just tune the parameter. Here we will assign n_estimators=[450,500] and 'criterion': ['gini', 'entropy']. Now will find out the new accuracy. The new accuracy will be 0.986. This tuned accuracy will be a little bit more than the untuned accuracy.

IV. Result

Model evaluation and validation

The final model we have chosen is tuned random forest which gave us more accuracy that is 0.986. In order to achieve this accuracy we assigned n_estimators=[450,500] and 'criterion': ['gini', 'entropy'] but without using the n_estimators and criterion we achieved the accuracy of only 0.972 which is a bit

less than the tuned value. Here we can say that the solution is reasonable because we are getting much more less accuracy while using other models.
The final model that is tuned random forest has been tested with various inputs to evaluate whether the model generalises well. This model is also robust enough for the given problem. We can say this by testing it over different random sates. Here we used the random sates 100,105,110 and obtained nearly same accuracy for each model and the mean accuracy is equal to the accuracy when random sate is 100. From this we can say that Small changes in the training data will not affect the results greatly. So the results found from this model can be trusted.

Justification

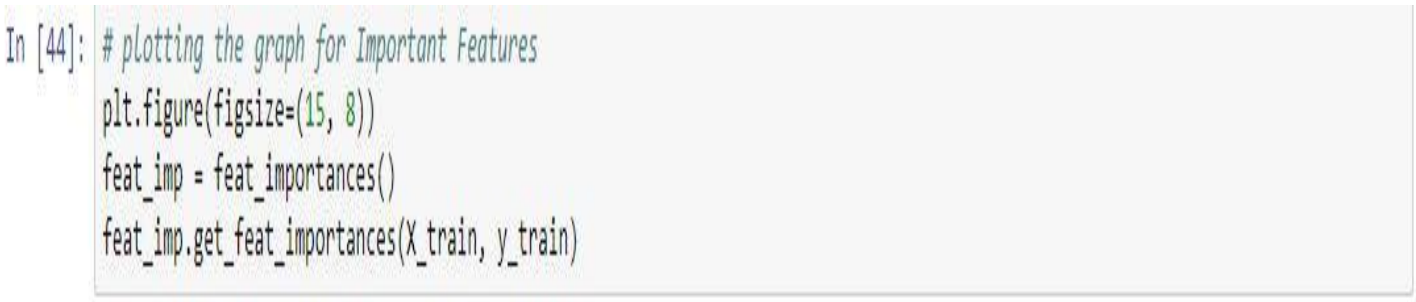
My final model's solution is better than the benchmark model.

Tuned	random forest	Benchmark model
Accuracy	0.986	0.718

From the above we can conclude that the results for the final model are stronger than the benchmark model.
Hence we can say that tuned random forest provides the significant to solve the problem of predicting the outcome of the endgame Tic-Tac-toe.

V. Conclusion

We will plot the important features according to their rank by using the bar graph.

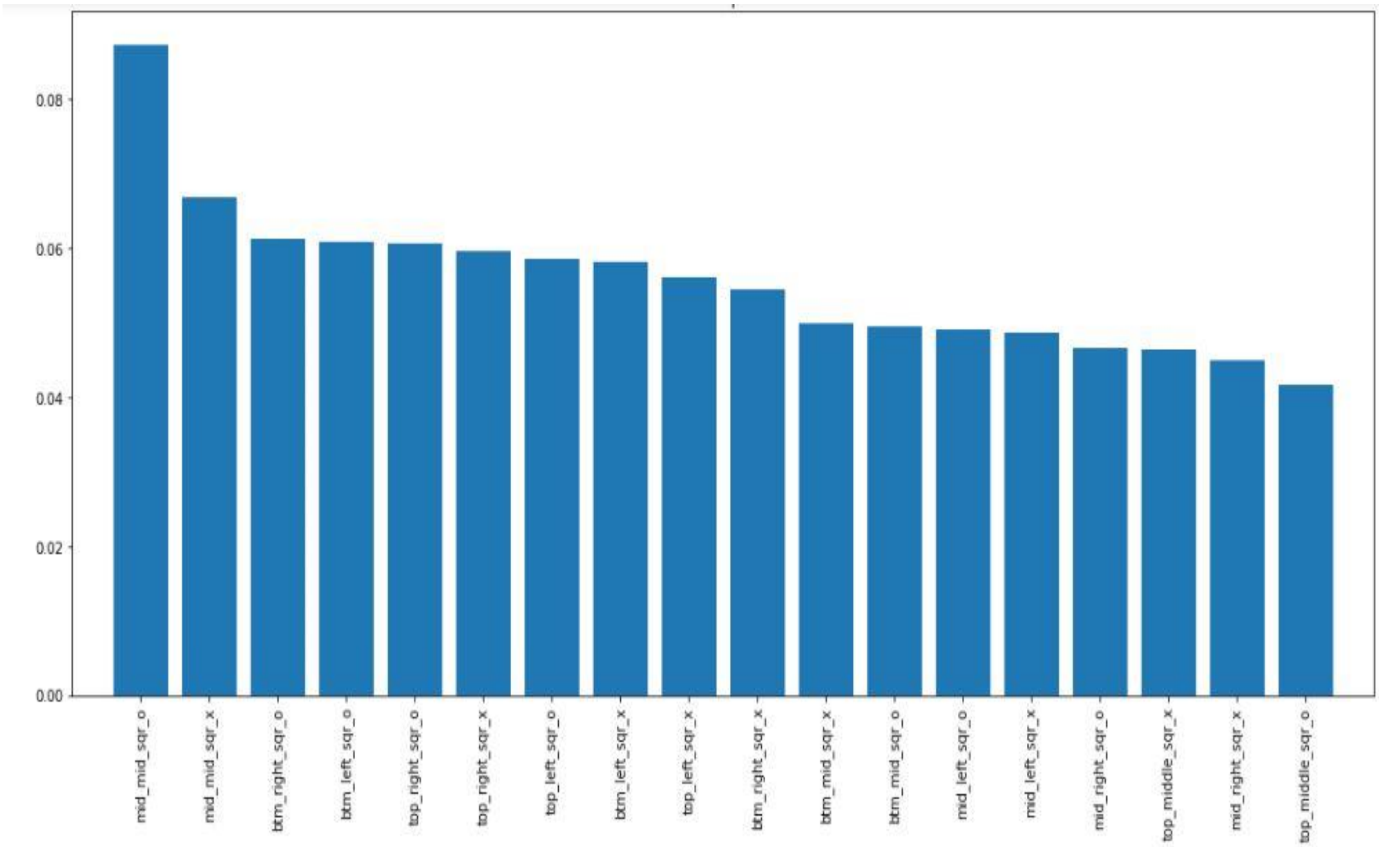


```
In [44]: # plotting the graph for Important Features
         plt.figure(figsize=(15, 8))
         feat_imp = feat_importances()
         feat_imp.get_feat_importances(X_train, y_train)
```

We will plot the graph by using the above code. We can find this in the coding section.

By plotting the bar diagram with features on x-axis and their importance on y-axis we will get the below bar diagram showing the ranks of the features according to their importance.

Plot for the feature importance



Feature

From the above graph we conclude the important features according to their rank that is wrr_4 is the most important feature and wkr_4 is the least important feature.

Reflection:

1. I have learnt how to visualize and understand the data.

2. I have learnt that the data cleaning place a very vital role in data analytics.

3. Removing the data features which are not necessary in evaluating model is very important.

4. I got to know how to use the best technique for the data using appropriate ways

5. I got to know how to tune the parameters in order to achieve the best score.

6. On a whole I learnt how to graph a dataset and applying cleaning techniques on it and to fit the best techniques to get best score.

Improvement:

The process which i have followed can be improved to classify not only for the Tic-Tac-Toe Endgame. This can also be applied for the all the other endgames in chess like increasing the size of the board to 4 x 4, 5 x 5, or even up to a 20 x 20 grid. This model is a part of my researching in Endgames. This application can be taken to next level with the many more applications. As we can say that there has never been an end in the machine learning there will be many more models to learn. By taking the more amount of dataset which consists the information about all the endgames in Tic-Tac-Toe this can be more and more be generalized to all the endgames in the Tic-Tac-Toe of grids.