

Spring 2021

Learning to Rank Model Performance and Review with Pairwise Transformations

Ajinkya Vijay Koshti

Follow this and additional works at: <https://lib.dr.iastate.edu/creativecomponents>



Part of the [Operational Research Commons](#)

Recommended Citation

Koshti, Ajinkya Vijay, "Learning to Rank Model Performance and Review with Pairwise Transformations" (2021). *Creative Components*. 757.

<https://lib.dr.iastate.edu/creativecomponents/757>

This Creative Component is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Creative Components by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

Learning to Rank Model Performance and Review with Pairwise Transformations

by

Ajinkya Koshti

A creative component submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Industrial Engineering

Program of Study Committee:

Dr. Sigurdur Olafsson, Major Professor

Dr. Gary Mirka

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this creative component. The Graduate College will ensure this creative component is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2021

Copyright © Ajinkya Koshti, 2021. All rights reserved.

DEDICATION

I would like to dedicate this creative component to my parents, Seema, and Vijay Koshti. Without their constant encouragement throughout my engineering education, this work would not have been possible.

TABLE OF CONTENTS

	Page
LIST OF FIGURES	v
LIST OF TABLES	vi
ACKNOWLEDGMENTS	vii
ABSTRACT	viii
CHAPTER 1. INTRODUCTION	9
CHAPTER 2. LITERATURE REVIEW	14
CHAPTER 3. PAIRWISE TRANSFORMATIONS	16
The concept.....	16
CHAPTER 4. THE DATASET	18
CHAPTER 5. ALGORITHMS	20
Decision Trees	20
Recursive Binary Splitting	22
Hyperparameters	23
Random Forest.....	25
Bagging	26
XGBoost	28
CHAPTER 6. TESTING METHODOLOGIES AND EVALUATION METRICS	30
K-fold Cross-Validation	30
Stratified K-fold Cross-Validation	30
ROC Curve	31
Area Under the Curve (AUC).....	32
CHAPTER 7. FEATURE SELECTION AND HYPERPARAMETER OPTIMIZATION	34
Feature Selection	34
Grid-Search Optimization.....	35
Random-Search Optimization	35
CHAPTER 8. RESULTS	36
Decision Tree Model	36
Decision Tree	36
Baseline Model Results	37
Stratified K-fold Cross-Validation Results	37
Area Under the Curve.....	38
Feature Importance Score.....	39

Model Performance after Feature Selection	39
Stratified K-fold Cross-Validation Results after Feature Selection	40
Area Under the Curve after Feature Selection	40
Optimized Hyperparameters.....	41
Model Performance after Hyperparameter Optimization.....	41
Random Forest Model	42
Baseline Model Results	42
Stratified K-Fold Cross-Validation Results	42
Area Under the Curve.....	43
Feature Importance Score.....	44
Model Performance After Feature Selection.....	44
Stratified K-Fold Cross-Validation Results After Feature Selection	45
Area Under the Curve After Feature Selection	45
Optimized Hyperparameters.....	46
Model Performance After Hyperparameter Optimization.....	46
XGBoost Results	47
Baseline Model Performance	47
Stratified K-Fold Cross-Validation Results	47
Area Under the Curve.....	48
Feature Importance Score.....	49
Baseline Model Results After Feature Selection.....	49
Stratified K-Fold Cross-Validation Results After Feature Selection	50
Area Under the Curve After Feature Selection	50
Optimized Hyperparameters.....	51
Model Performance after Hyperparameter Optimization.....	51
Borda Count.....	52
Borda count generated list example:	53
CHAPTER 9. CONCLUSION.....	54
APPENDIX. THE PYTHON FUNCTION.....	59

LIST OF FIGURES

	Page
Figure 1. Decision Tree.....	21
Figure 2. Details of a node.....	22
Figure 3. Random Forest.....	25
Figure 4. ROC curve (figure adapted from Classification: ROC Curve and AUC, Google)	32
Figure 5. Decision tree for baseline model	36
Figure 6. Decision tree baseline model ROC and AUC	38
Figure 7. Decision tree feature importance score	39
Figure 8. Decision tree feature selection ROC and AUC	40
Figure 9: Random forest baseline model AUC	43
Figure 10: Random forest feature importance score.....	44
Figure 11: Random forest feature selection ROC and AUC.....	45
Figure 12: XGBoost baseline model AUC	48
Figure 13: XGBoost feature importance score	49
Figure 14: XGBoost feature selection AUC	50

LIST OF TABLES

	Page
Table 1. Traditional dataset.....	16
Table 2. Pairwise transform of traditional dataset	17
Table 3. Decision tree baseline model results.....	37
Table 4. Decision tree stratified k-fold cross-validation results	37
Table 5. Decision tree feature selection results	39
Table 6. Decision tree feature selection stratified k-fold cross-validation results	40
Table 7. Decision tree hyperparameter optimization results	41
Table 8: Random forest baseline model results	42
Table 9: Random forest stratified k-fold cross-validation results.....	42
Table 10: Random forest feature selection results	44
Table 11: Random forest feature selection stratified k-fold cross-validation results	45
Table 12: Random Forest hyperparameter optimization results	46
Table 13: XGBoost baseline model results.....	47
Table 14: XGBoost stratified k-fold cross-validation results	47
Table 15: XGBoost feature selection results	49
Table 16: XGBoost feature selection stratified k-fold cross-validation results	50
Table 17: XGBoost hyperparameter optimization results	51
Table 18: Borda count results (top 5)	53

ACKNOWLEDGMENTS

I would like to thank my committee chair, Dr. Sigurdur Olafsson, and my committee member Dr. Gary Mirka for their guidance and support throughout the course of this research.

In addition, I would also like to thank my friends, colleagues, the department faculty and staff for making my time at Iowa State University a wonderful experience.

ABSTRACT

Learning to rank is an application of machine learning that is finding increasing use in information retrieval systems. Various applications such as document retrieval, webpage ranking, sentiment analysis, and online advertising use one or the other kind of learning algorithm to return a list of instances ranked in order of the quality of fit. Learning to rank also finds application in sports analytic; especially in activities like football scouting, where club representatives attend football games worldwide to collect intelligence on potential recruits.

In this project, learning to rank method is applied to automate the process of football scouting by analyzing the player database to generate a list of desired number of footballers that could be recruited based on given criteria. This becomes an important application as player recruiting is a competitive business and having a list of options ranked on given criteria can help in making faster and better decisions. Learning to rank technique is used with pairwise-approach that makes it possible to use state-of-the-art supervised algorithms for generating predictions. Borda count technique is applied to convert the pairwise predictions into a ranked list.

CHAPTER 1. INTRODUCTION

Analytics has played an essential role in the growth of many organizations in different sectors. Business analytics includes different branches of computer science like data engineering, predictive analytics, and business intelligence at a high level. Each of these processes has importance in its own business functions but is often used together to tackle challenges and unlock new avenues. Data engineering deals with creating a pipeline for collecting, organizing, and storing data as per relevance and requirement. The data is analyzed according to the business goals and it often has a common theme of predicting an unknown and uncertain business phenomenon. This activity of predicting unknown parameters from historical data is termed predictive analytics. Predictive analytics enables companies to tackle variety of problems ranging from improving existing processes to anticipating customer churn [1].

Various companies have been using predictive analytics for different business functions for many years. Historically, companies always relied on human intelligence to make decisions. Persons with subject-matter expertise were entrusted to decide the strategy of the company and improvise if needed. The human knowledge and understanding of the field enabled them to anticipate market changes, uncertain challenges, and unprecedented difficulties. Hence, depending on the personnel available, companies could navigate these challenges. But as the scale of business increased and inventions picked up the pace, the industry realized the value of historical data. The historical data contained different patterns that were good indicators of success, and unearthing these patterns unlocked many potential opportunities for businesses.

At a high level, predictive analytics is a function of data, algorithms, and cost function. The quality of a predictive model is defined by these three factors – data, algorithm, and the cost function. Predictive analytics includes experimental techniques that considers practical scenarios

rather than an established theory to draw its predictions [2]. Since the real value is in the data, making sure the collected data is consistent, appropriate, valid becomes the first step towards building any model. The second building block of a predictive model is an algorithm. These algorithms are designed for common as well as special purposes and are innovated consistently by researchers all over the globe. The last aspect of the model is its cost function. Cost function also called an evaluation metric, is the parameter used to evaluate the model's quality. It depends on the business goal the model is used for. Some common examples of cost functions are accuracy, precision, and F1 score.

Various studies have indicated a strong connection between a company's data science strategy and its success in terms of revenue, profit, or shareholder returns [3]. Companies that have a well-planned data strategy tend to stay ahead of the curve. A study suggests big organizations have a three times better data strategy than the smaller firms [4]. A good data strategy adds a predictive layer to business functions and anticipates potential problems ahead of time. The value of building a good data strategy lies not in terms of money saved in the present but in opportunities generated for the future. With the creative use of data, many companies have been able to change their fortunes. Retail businesses use predictive analytics for predicting product demand, optimizing display shelves, and managing inventory. Various practices like just-in-time or lean can help enterprises to optimize their inventories and product orders by integrating predictive analytics and generating intelligent predictions backed by data. The airline industry uses historical data to predict busier days and busier networks to change their pricing accordingly. They use these data-generated hotspots on networks and schedules to adjust their pricing. Hotels and restaurants or other hospitality industry businesses use predictive analytics to predict the demand or occupancy and plan accordingly. They can use high occupancy periods to

organize special events or concerts and generate maximum revenue or plan the maintenance work during periods of lean occupancies. Thus, integrating predictive analytics in business strategy allows organizations flexibility to operate.

Predictive analytics also has social applications. It can be used to prevent criminal activities. Bank frauds, credit card scams, email spams are now easily detectable with advanced algorithms. Although these algorithms err on the side of caution and generate a significant number of false positives, they have very well succeeded in reducing the number of such crimes. Face detection technologies have helped identify criminals and speed up the investigation process. Advanced algorithms can now help physicians classify skin cancers and anticipate the progression of age-related conditions like osteoarthritis in healthcare. Though these technologies are far from reliable substitutes, they are now more than reliable decision support systems. With the pace of innovation and research interest today, these technologies can soon become viable alternatives to human expertise.

We primarily talk about two technologies – machine learning and deep learning when we talk about predictive analytics. Machine learning is a technique enabling machines to learn from historical data and make decisions independently. These machines learn from their own mistakes and become smarter with each iteration. This operation of machines learning from the data and being penalized for their mistakes is defined as an algorithm. There are multitudes of machine learning algorithms that only get better with time and application. Another positive with machine learning is that these algorithms have not only got better at predicting but also have reduced their requirements. You no longer even need a high-speed processor to implement any of these advanced algorithms. Cloud computing technology enables you to build your models entirely on the cloud and deploy them using virtual machines. Thus, as a science, machine learning has

grown astronomically in last two decades [5]. Deep learning is another technique that works on the same principle but with different architecture. Deep learning implements neural networks, a data architecture that replicates the human brain's neural nodes and decision-making power. These algorithms differ in complexity, but at the high level, the premise is the same – to minimize the error evaluated by the cost function.

Machine learning primarily sees two kinds of problems – supervised learning and unsupervised learning. As the name suggests, supervised learning is 'supervised' or 'guided' by the true labels. When the available historical data contains true values of the datapoints, and the algorithm simply needs to find the patterns in data relating to those labels, the process used is called supervised learning. Supervised learning is an easier and more straight-forward approach, and the error is easier to evaluate. The main task of supervised learning is to generate insight into which attributes correlate strongly with the given true label. Unsupervised learning is used when the historical data does not contain any true labels. Here, we do not know the true values of datapoints, and hence, the task is to group similar datapoints in the form of a cluster based on their closeness in terms of available features. Since true labels are unknown, this process does not help in evaluating individual datapoints but in assessing the proximity of datapoints and clustering them together. Some applications of unsupervised learning are clustering together potential customers or separating a population based on demographics.

These practices are helpful when the objective is to classify observations or predict a specific datapoint's numerical value. There are often applications where datapoints are necessary to be arranged rank-wise in a list; this could be due to the intrinsic nature of the problem at hand or due to the limitations in the system. For example, let us consider a search engine optimization task. In this task, when a user enters a search term, the search engine should return some results

to those queries. Here, the result that is the closest match to the search terms should be displayed first, while the second closest match is displayed second. This is an ordered task where results are required in the form of an ordered list. Let us consider another example where ordered lists are necessary due to a limitation in the system. For example, consider a credit card fraud detection system. If there are five cybersecurity experts in a team, only five credit card fraud alerts could be handled at a time. In such situations, our model must return five instances that are at the most risk. The list of ordered five instances is what we need from our algorithm. An algorithm that can perform this task is called learning to rank.

The thesis contributes by proposing a learning to rank model to capture the value in the football scouting and recruiting business. The learning to rank approach to football scouting will aid in decision making and better squad planning which has significant commercial value. The thesis introduces and implements the unique idea of using three different techniques – learning to rank pairwise approach, state-of-the-art supervised learning algorithm like XGBoost, and a Borda count method to generate a ranked list of instances based on given criteria, in this case, player attributes. Currently, a variety of heuristics and probability functions are used to generate a ranking from predictions. Borda count is a straight-forward approach that gives equal weight to all pairwise comparisons and considers all pairwise match-ups equally in ranking evaluation. Last but not least, the thesis contributes to the Python library by creating an innovative function to generate pairwise transformations that make the foundation of ranking analysis with one line of code. The function is reproducible and can be applied to any new analysis.

CHAPTER 2. LITERATURE REVIEW

Learning to rank (LTR) is growing in both popularity and application. Of all the approaches used in LTR, pairwise transformation is the one used and applied most on the document retrieval tasks [6]. Shashua & Levin (2002) tackled a multi-class learning problem arising in visual recognition and information retrieval. They introduced a large-margin criterion to come up with a solution that is biased towards pairs farthest apart from each other [7]. The thesis builds on this premise of applying a traditional machine learning algorithm on pairs of instances. Lebanon & Lafferty (2002) build on Mallows Model, a generative model calculating distribution of permutations, to combine multiple input rankings to create a model that optimizes ranking instead of classification. They built probability distributions over the label rankings obtained from each input model and built permutation group models [8].

Herbrich et al. (2002) used support vector machines (SVM) to create an ordinal regression model based on margin rank boundaries. The ordinal regression model outperformed support vector classification and support vector regression for document retrieval [9]. Similarly, RankBoost was created as an ordinal regression extension to the AdaBoost algorithm [10]. A single linear ordering of instances was produced by combining a set of given linear orderings called the ranking features. Burges et al (2005) proposed ranking using gradient descent approach [11]. They used a neural network on top of a probabilistic cost function called as RankNet.

Learning to rank requires training data containing labels generated from pre-defined conditions or from manual judgement of relevance. Joachims (2002) used clickthrough data to present a method for document retrieval. This method worked well in risk minimization framework and was shown to be feasible with larger queries and datasets [12]. Olafsson and Li

(2005) used decision trees to create dispatching rules for a production scheduling task [13]. One problem was to prevent the model from being biased towards queries containing large number of relevant documents. This results from giving equal penalties to incorrect ranking among higher ranked and lower ranked documents, in tasks where accuracy in top-ranked results was desired [14]. Cao et al. (2006) proposed modifying hinge loss function in ranking SVM to deal with this problem. The loss function was optimized using gradient descent and quadratic programming [14]. All popular methods thus far encounter a common problem – using an appropriate probability function to create a list and propose a unique solution for it.

CHAPTER 3. PAIRWISE TRANSFORMATIONS

The concept

In pairwise approach, we compare all possible pairs of observations in the dataset to identify which observation ranks higher than the other one in the pair. Pairwise method performs better than the pointwise method; the listwise approach outperforms all but is also more complicated. [15] These outcomes are then analyzed together to form a cohesive ranked list. When we say we compare all pairs of observations, we pit all their features against each other. Consider an example of three observations with three features.

Table 1. Traditional dataset

ID	Feature1	Feature2	Feature3
1			
2			
3			

Then our pairwise transformations will first return a cartesian product of observations and then compare the features for each of them. The resultant transformation would look like the table below. The Feature1_Higher returns if the value of Feature1 of the first observation in the pair is greater or not.

Table 2. Pairwise transform of traditional dataset

ID (Pair)	Feature1_Higher?	Feature2_Higher?	Feature3_Higher?
1-2	Yes/no	Yes/no	Yes/no
1-3	Yes/no	Yes/no	Yes/no
2-3	Yes/no	Yes/no	Yes/no

Creating pairwise transformations is not a straight-forward job in R or Python. There are no available packages that can create these transformations from scratch. ‘Itertools’ package in python provides an option to create the cartesian product but it has no option of comparing the attribute values. Hence, it was of great incentive to create a robust Python function which can return the pairwise transformations with a simple one line of code and without having to worry about any supplementary work. The created Python function (see appendix) only takes the dataset as input and requires identification of ID column. Post that it would automatically create the transformations and return a new dataframe with pairwise transformations.

CHAPTER 4. THE DATASET

To test the learning to rank approach, FIFA19 dataset was used. The dataset was obtained from Kaggle and contained detailed statistics and attributes of all the soccer players in the EA Sports FIFA 19 video game. [16] The dataset contains attributes like player information, player attributes, player ratings, player net worth and in-game skill ratings given by EA Sports. The main attribute here is the ‘Overall Rating’ which decides how good a player is in the gameplay. The overall rating is ascribed on a scale of 1-100 with 1 being the lowest and 100 being the highest.

In soccer world, player transfer from one club to another is a common phenomenon. Clubs often look for the best talent in the market they can recruit for their club within the available price range. The market value of player depends upon number of factors like – Player ratings, player position, current form and whether the player holds European passport. Top clubs of each league compete for the best players in each position every year with one of them emerging as a winner. The transfer business though is not as straightforward as it sounds. There is a great deal of uncertainty involved and often, a player deemed as a future star fails to shine whereas a player overlooked in youth leagues goes on to become a world class player. This uncertainty is more serious as the club is investing a large amount in a future of a player, and if the player fails to make his mark, the club must face losses. Hence, clubs are wary while spending on unproven talents while happy to break their bank on proven youngsters – a strategy that places more financial risk on the given player living up to his expectations. One such example is when FC Barcelona bought Phillipe Coutinho from Liverpool FC at a high price of \$159.5 million when his market value was \$99.0 million (source: transfermarkt). The player though failed to make his mark at the new club and soon his market value dropped to \$61.6

million. FC Barcelona are now trying to sell the player but, in any case, they will have to bear huge financial losses owing to the drop in market value. The top 5 clubs have spent \$4.21 billion in transfers in 2018 and the sum is only increasing. The total transfer industry is a business of over \$10 billion a year. Hence there is a huge potential for data mining and analytics to aid in the decision process.

The premise of this project lies on the idea of automating the player scouting aspect – which is – finding the best player for a club to pursue his transfer. When we say finding the best player, we actually want three or four more players as the backup in case we lose our first-choice player to another club. Hence, this becomes a perfect application of ranking algorithm. We will seek to return a list of top 5 alternatives ranked for a given player profile. Currently, the process of scouting players is carried out manually with expert scouts, often the former players or coaches, observe the young players and decide if a player is worth pursuing or not. In this project, the scouting process will be automated based on the skill ratings given by FIFA. These skill ratings denote how good a player is in each department of the sport. We will use these skill ratings to predict the overall rating and rank the players in contention. An alternative application could also be ranking players for individual, and team of the season awards based on the year's statistics.

CHAPTER 5. ALGORITHMS

One prime advantage of using pairwise transformations for ranking is any machine learning algorithm can be used. Since every datapoint becomes a direct comparison between two records, we can use any algorithm to predict whether the first record in comparison ranks higher. To directly test the ranking algorithm, we used the algorithm on the pairwise transformed dataset to get comparison results and rank them in order using the Borda count method. The objective was to generate a list based on given requirements.

Decision Trees

Decision tree learning is one of the most popular classification algorithms. It is a machine learning algorithm which uses an everyday structure of a tree to organize its predictive algorithm. It can cover both kinds of predictive tasks – classification and regression; and algorithms are named as such. In decision tree classification, a decision-making process is structured like a tree where every piece of information enables you to choose a branch. A node is the core building block of the tree. A node with no incoming edges is called a root, a node with no outgoing edges is called a leaf, while all other nodes with are called internal or test nodes [17]. The branch is one of the labels that you are looking for to classify the instance. Each internal node separates the datapoints into multiple sub-spaces based on the proximity of values [17]. The separation is based on cardinality in categorical features, and on the range of values in numerical ones.

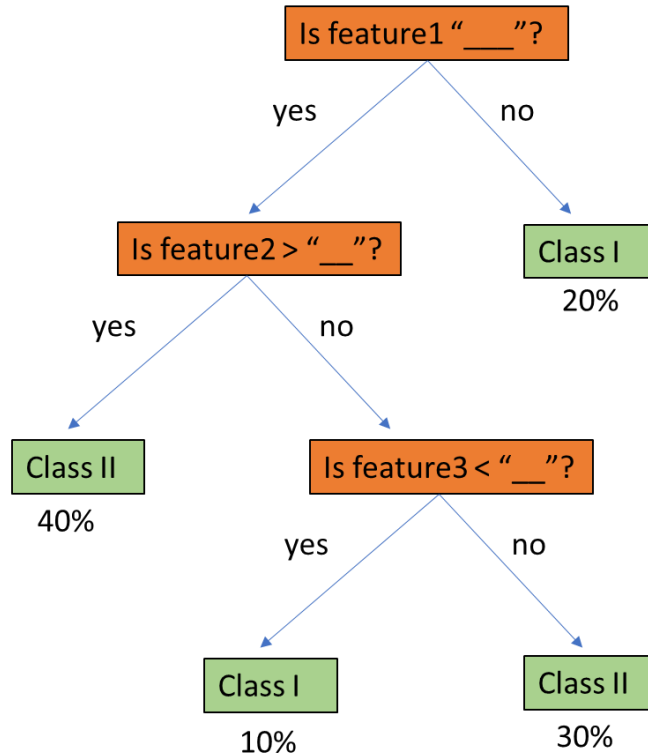
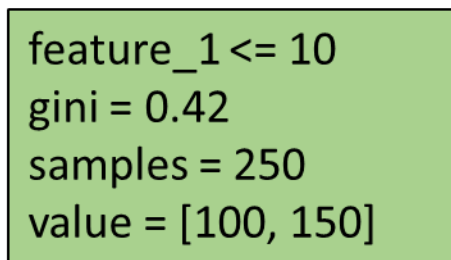


Figure 1. Decision Tree

A decision tree is drawn upside down and is traversed in similar fashion. For a classification tree, we classify the instance into a leaf at each node. The leaves at the bottom represent each separate class, holding a target value or a probability vector representing the probability of an instance belonging to that class. Refer the picture above, the orange boxes represent the conditions also called as nodes. These nodes are further split into branches based on the condition given. The process is continued until all the instances are classified. The terminal nodes which cannot be split further are called as decision leaves. The green boxes represent the class the instances have been classified into.

A practical dataset with a real-world problem will have many features and thus will result into bigger trees with deeper branches. The methodology of the algorithm can be clearly understood from this simple example though. The importance of features and their statistical

significance is clearly evident, and it can also be seen that features don't influence the decision-making standalone; their relative values with other features is what influences splitting of the nodes. Thus, this algorithm won't be the best choice for applications with independent features. But since, all real-world features are dependent, decision tree classifier becomes the most popular choice for getting the baseline predictions. A detailed node looks like the figure shown below.



```
feature_1 <= 10
gini = 0.42
samples = 250
value = [100, 150]
```

Figure 2. Details of a node

The first line indicates the condition of the feature, in this case the feature_1 being less than or equal to 10. The second line indicates the gini index which is the measure of impurity. Samples indicate the number of instances classified in this node while the values indicate their distribution in various classes.

Recursive Binary Splitting

The process of considering all the features with different split points (to split the nodes) is called as recursive binary splitting. The split points are important as changing one split point can change the entire tree. They are decided using a cost function used to evaluate the error, thus each node is split at a point that minimizes the overall error.

The two functions used to evaluate the split are the Gini index and Entropy. Gini Index measures impurity of the instance space represented by probability distributions of the target values [17]. In simpler terms, Gini index indicates how often an instance would be labelled

incorrectly was it to be labelled randomly based on the distribution of the labels. Thus, if the labels are distributed evenly among all classes, there will be perfect impurity as no one class dominates another, and the Gini index would be 1.

The second function used to evaluate the split is entropy. It is also a measure of randomness of the class distribution. Higher the randomness, higher the entropy.

$$Gini = 1 - \sum_{i=1}^n p^2(c_i)$$

$$Entropy = \sum_{i=1}^n -p(c_i) \log_2(p(c_i))$$

Where $p(c_i)$ is the probability/percentage of class c_i in the node.

Gain(S, a) is defined as the information provided by an attribute 'a' that contributes to the quality of the classification. Each attribute will provide varying information gain based on its entropy which is the measure of randomness of the class distribution of that attribute [18].

Information gain will be calculated as:

$$Gain(S, a) = Entropy(S) - \sum_{v \in Values(a)} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$S_v = \{s \in S : a(s) = v\}$$

Hyperparameters

- i) **Max_depth**: max_depth is used to control the depth of the tree, i.e., how deep the model will go on splitting the instance space. When max depth is set to a certain value, no other criteria (like entropy or Gini index) will be considered for splitting. The tree will simply stop at the set value and the leaf nodes at that value will represent final classification.
- ii) **Min_samples_leaf**: min_samples_leaf indicate the minimum number of instances required to be in that leaf node after the classification is made. If a split is made

such that a leaf node represents only a few samples, it indicates that we are overfitting the model and not generalizing well-enough for the whole dataset. Doing so also reduces the complexity of the model as it naturally controls the depth of the tree.

- iii) `Min_samples_split`: `Min_samples_split` works exactly like `min_samples_leaf`, i.e., it governs the split to reduce the overfitting. The difference is that the split is decided based on the number of instances available in the node after the previous split was made.
- iv) `Criterion`: `Criterion` represents the criterion used to evaluate the quality of the split (Gini or entropy)

Random Forest

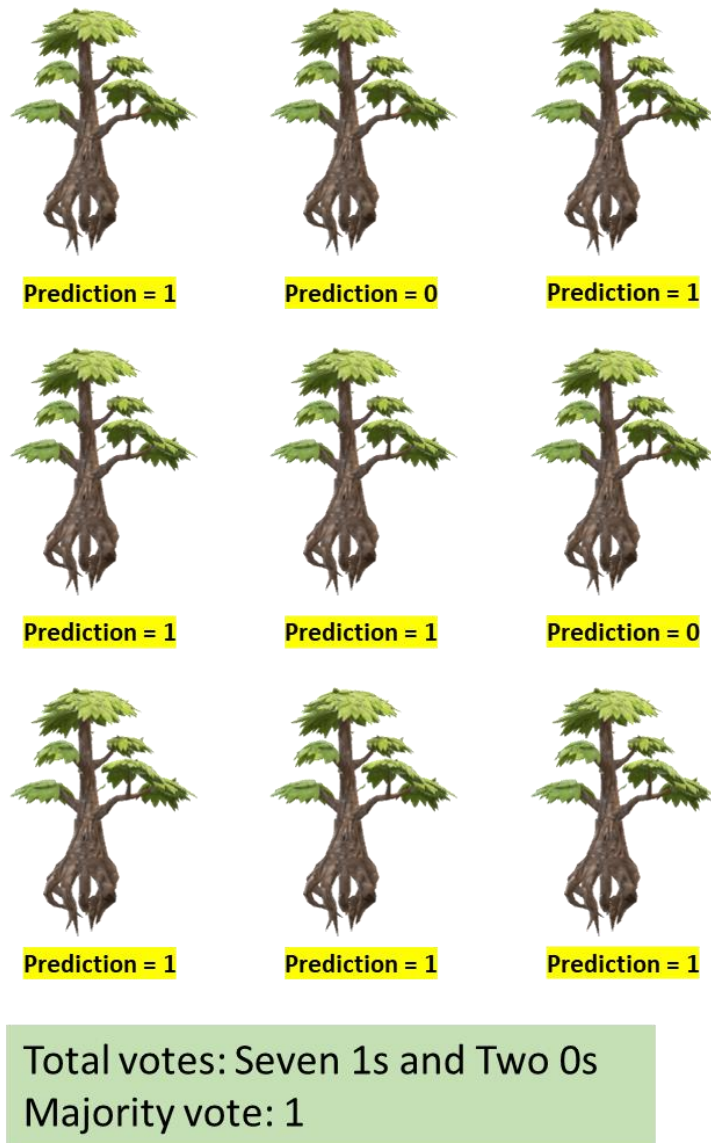


Figure 3. Random Forest

Random forest is a popular algorithm that is an extension of decision trees and the basis of advanced bagging and boosting machines. Like the name suggests, random forest classifier is a classifier combining a random set of tree classifiers from all possible tree classifiers. This randomness in selection of trees determines the quality of random forest model. The tree classifiers in the set are generated using a random vector that has been sampled independently of

the input vector. Each generated tree will classify the instance space on its own and the classification will count as one vote towards making the final classification [19]. Thus, final classification is a majority vote from all individual tree classifiers.

The underlying concept at the back of random forest is a simple but compelling one — the wisdom of crowds. A group of people is likelier to make a correct decision than any one individual picked at random.

One key thing while using random forests is the low correlation between the individual trees. If we are using trees with high correlation between them, we are essentially using the same tree twice thus increasing the inherent bias of that one tree. Uncorrelated models inculcate the concept of wisdom of crowds in true sense and yield more accurate predictions than any individual tree. The reason for this wonderful result is that as long as the models replicate the diversity of the crowd, i.e., they do not correlate highly with each other essentially causing the duplicates, the error of individual models shall be overcome by collective prediction of the models. The uncorrelated models make an error, but it is very unlikely that they all err in the same direction. Hence, as long as majority of the trees are right, the random forest will predict correctly.

Bagging

Decision trees are quite sensitive to the underlying data i.e., a small change in the underlying data can end up in a totally different tree and consequently totally different predictions. Random forest puts this to advantage by randomly sampling from the dataset with replacement and training an individual tree on it [19]. This process is called bagging or bootstrap aggregating. Remember, we are not reducing the size of the dataset, we are just picking a sample from it with replacement.

Another factor in bagging is the use of random subset of features. As we use all features in our decision tree, in random forests, each tree uses a random collection of features. This makes each tree unique and less correlated to each other.

XGBoost

XGboost is an advanced machine learning algorithm that became popular on platforms like Kaggle. It supports classification, regression, as well as ordinal regression problems and is available in all popular frameworks including R, Python, and Julia. XGBoost stands for eXtreme Gradient Boosting. It was created by Tianqi Chen and later developed by other contributors. In his own words, *"The name XGboost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms. Which is the reason why many people use XGboost."* [20]

The XGBoost library implements the gradient boosting decision tree algorithm. For multiclass classification problems, tree boosting has shown to yield better results than other classification algorithms, and almost as good as the results from deep learning [21]. It is also referred as gradient boosting, multiple additive regression trees, stochastic gradient boosting, and gradient boosting machines. Boosting is simply an ensemble process where models clustered together work on residues of previous models. That is, they correct the error made by previous models thus minimizing the loss. The loss is minimized by using a gradient descent function and this is where the name gradient boosting comes from.

XGboost was developed to get the best in terms of both computing time and memory resources since this is a primary problem with large datasets. The algorithm yields prediction ten times faster than existing algorithms and is scalable to distributed systems as well; this provides the dual advantage of using XGBoost [22].

In previous studies, XGBoost outperformed several algorithms in terms of speed and accuracy. Python's XGBoost package in scikit library was found to be more accurate than R's GBM package as the former works on expanding entire tree while the latter works on just one branch. The result is faster but comes at the cost of accuracy. The algorithm while using column

subspace too, speeds up the process even further, but at the cost of the accuracy. Thus, it can be useful only on truly big datasets [22]. The interesting and relevant result for this thesis is the use of column subsampling while working on a ranking problem [22]. Subsampling tends to reduce overfitting and thus increases accuracy on ranking problems, which is further supported by the experimental results.

Some key algorithm implementation features include: [22]

- i) **Sparse Aware** implementation to deal with missing values. When a sparse matrix is missing values, the instance is classified in the default space ignoring those values. Sparsity aware is a unique algorithm used to find the best values in place of missing data instead of using simple averages.
- ii) **Block Structure** to fully utilize the resources available like processor, memory, and the disk space. Block structure allows use of independent threads for computing results and reading the data thus enabling the parallelization to make efficient use of computing resources.
- iii) **Continued Training** so that you can use pre-trained models and further boost them.
- iv) **Highly scalable** and end-to-end system. The portability and robust distribution of algorithm enables it to be run on any system.

CHAPTER 6. TESTING METHODOLOGIES AND EVALUATION METRICS

K-fold Cross-Validation

Cross-validation is a statistical method used to test the machine learning models by dividing the data into number of folds, training on all but one fold and then testing on the remaining one. This enables the model to use entire dataset as the test data for same model.

In this method, 'k' number of partitions are created in the data. In the first iteration, first partition is reserved for test, and the model is trained on remaining partitions. Similarly, in the second iteration, the second partition is reserved for test while remaining partitions are used for training. The process is repeated until all the partitions have been used for testing. The error is then estimated by using the average of error obtained in each partition.

It is easy to implement in python and can be used to reduce the bias in the model. Even if it has some bias, the bias will be pessimistic i.e., it will overestimate the error.

Here, scikit-learn package was used to implement the k-fold cross-validation.

Stratified K-fold Cross-Validation

Stratified K-fold cross-validation is an extension of k-fold cross validation used to reduce bias in imbalanced data. In stratified k-fold, all folds have equal strata of target classes i.e., the ratio between target classes is the same in each fold. This ensures the data is distributed evenly among all folds and we are getting a less biased prediction.

Stratified k-fold was useful in this case as you could see that the accuracy was yielding to 100% in our training data. This happened so because of the class imbalance; due to which the model predicted only one target label, which naturally turned out to be correct. Thus, stratified k-fold would give us the best results of all possible methods.

ROC Curve

A classification model gives a set of probabilities of an instance belonging to each class. In a binary classification, the probability threshold is usually considered as 0.5 to make the classification. Thus, if we change this threshold, we might get an entirely different classification for several instances, thus changing the value of cost-function. An ROC curve (receiver operating characteristic curve) is used to represent this change in classification with respect to different thresholds. An ROC curve plots two evaluation metrics: [23]

- i) True Positive Rate
- ii) False Positive Rate

True positive rate or Recall is defined as the ratio of classified true positives to the actual positives in the dataset.

$$TPR = \frac{TP}{TP + FN}$$

False positive rate is defined as the ratio of falsely classified positives to the actual negatives in the dataset.

$$FPR = \frac{FP}{FP + TN}$$

An ROC curve plots these two metrics (TPR and FPR) at different probability thresholds for classification. If we consider '1' as a label for positives, lowering the threshold of classification will classify more instances as positive and vice versa, increasing the threshold will classify more instances as negative. Conventionally, to calculate these points on the ROC curve, we would have needed to fit separate logistic regression models many times covering complete range (0 to 1) of threshold values. But there is a sorting algorithm making this task easier.

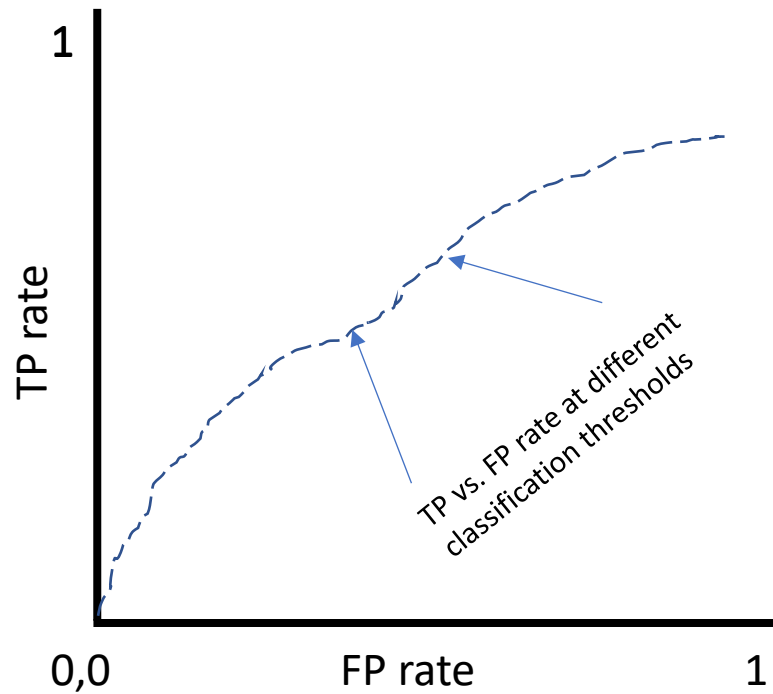


Figure 4. ROC curve (figure adapted from Classification: ROC Curve and AUC, Google)

Area Under the Curve (AUC)

AUC stands for the area under the curve. It measures the area engulfed by the ROC curve. AUC is an aggregate of the performance of the models at all classification thresholds. The value of AUC is represented as a fraction of 1; higher the value more accurate the model in classifying both positives and negatives [23]. Since AUC calculates the curve plotted at different classification thresholds, AUC really measures the impact of these thresholds. It indicates the possibility of a random positive ranked higher than a random negative. Additionally, AUC becomes an attractive metric to use for a couple more reasons – it is independent of the range of the label values, and of the classification threshold.

AUC is a key metric to evaluate for smaller datasets and datasets with class imbalance. A point to remember is since AUC works on identifying both erroneous positive and negative

classifications, it will be undesirable to use when one label has more cost attached to it than the other. For example, in medical diagnosing tests, a false negative is costlier than a false positive. Hence, a better objective function would be to minimize false negatives.

CHAPTER 7. FEATURE SELECTION AND HYPERPARAMETER OPTIMIZATION

Feature Selection

Feature selection, also known as attribute selection, dimensionality reduction, or variable subset selection, is a method of trimming down number of features in the dataset for building the model. It is widely used in machine learning, both to simplify the algorithm, and to reduce overfitting. Some of the practical applications of feature selection are:

- i) Faster processing
- ii) Avoiding curse of dimensionality
- iii) Reduce overfitting
- iv) Using only statistically significant variables
- v) Incorporate extra information on features in the model

The thought process behind using feature selection technique is that in a dataset with large number of features, there is a high probability that many of those features are redundant, irrelevant or contain too much noise. These features can be removed without losing any data thus making model simpler. Features can also be removed when two or more features are highly correlated. In that case, even though the second feature is relevant, it becomes redundant. One example is features like date of birth and age. Both features provide similar information and hence one can be removed.

Feature selection is also used when no features are irrelevant or redundant, but the size of data is too large to process. Hence, in such cases, features can be trimmed to increase the processing time.

Grid-Search Optimization

Grid search is simply a process where we select best hyperparameters for the model. We provide a list of hyperparameters in the form of the range or absolute values. The grid-search will then fit a decision tree using every possible combination of the provided hyperparameter values to give us the best possible fit. The advantage of using grid-search is you can test for a wide range of values and get the best parameters for your model, but the disadvantage is the model will take longer time to run.

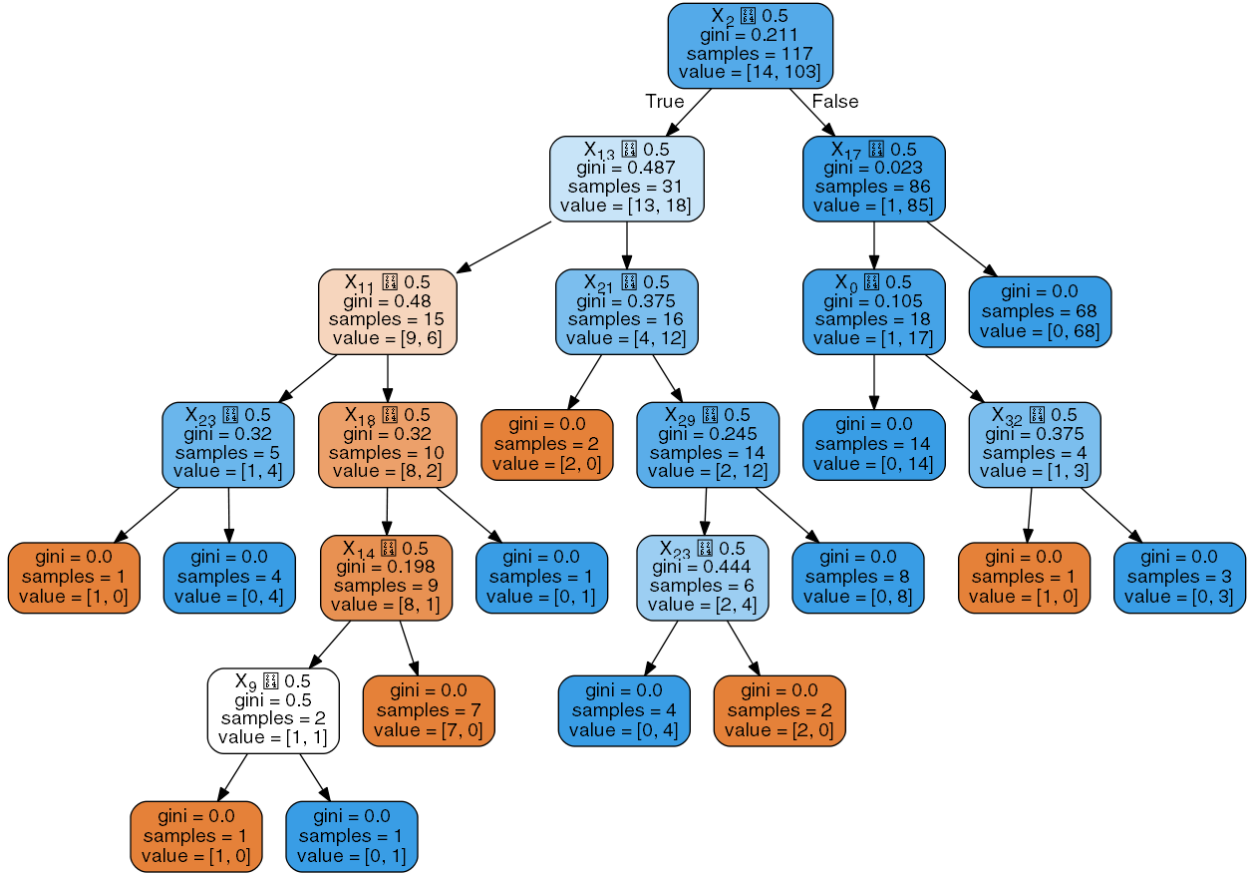
Random-Search Optimization

Random search optimization is similar to grid-search optimization except that instead of testing for every possible combination of hyperparameters, the algorithm will only test for random combinations. This saves computing effort while still optimizing the decision tree.

CHAPTER 8. RESULTS

Decision Tree Model

Decision Tree



Baseline Model Results

Table 3. Decision tree baseline model results

Evaluation Metric	Value
Training Accuracy	100%
Validation Accuracy	87.06%
Testing Accuracy	74.74%
10-fold Cross-validation	85.87%
Stratified cross-validation	89.37%

Stratified K-fold Cross-Validation Results

Table 4. Decision tree stratified k-fold cross-validation results

Evaluation Metric	Value
Maximum accuracy	94.01%
Minimum accuracy	85.59%
Overall accuracy	89.37%
Standard deviation	0.023

Area Under the Curve

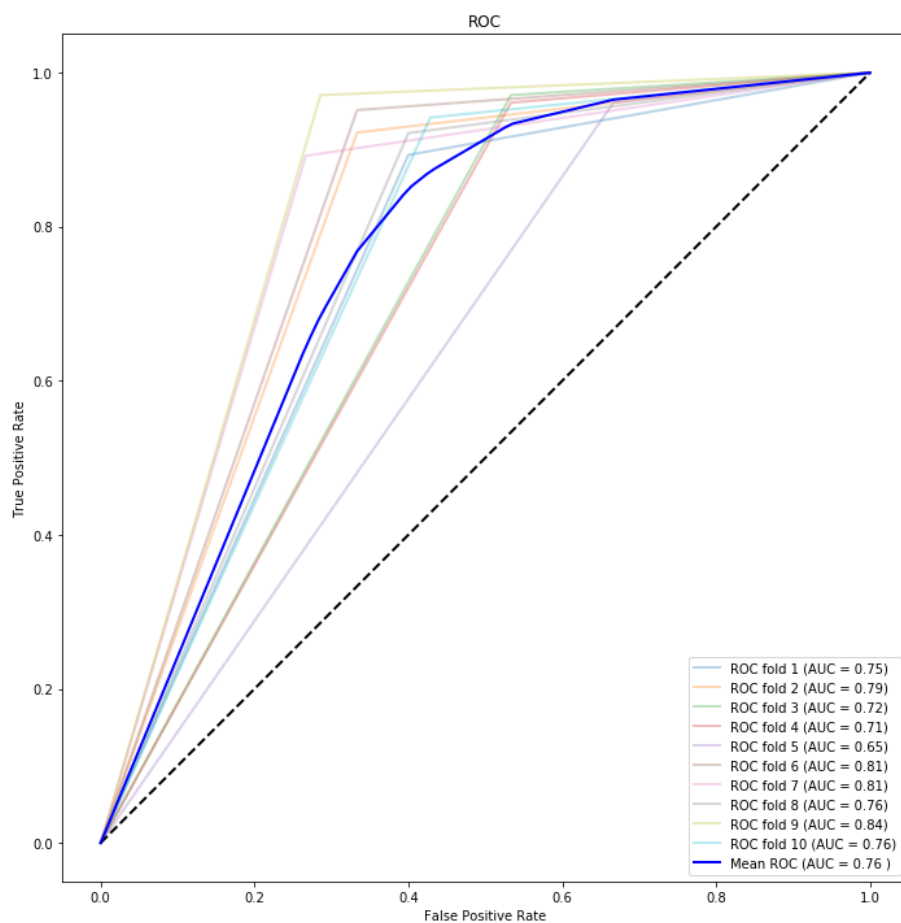


Figure 6. Decision tree baseline model ROC and AUC

Feature Importance Score

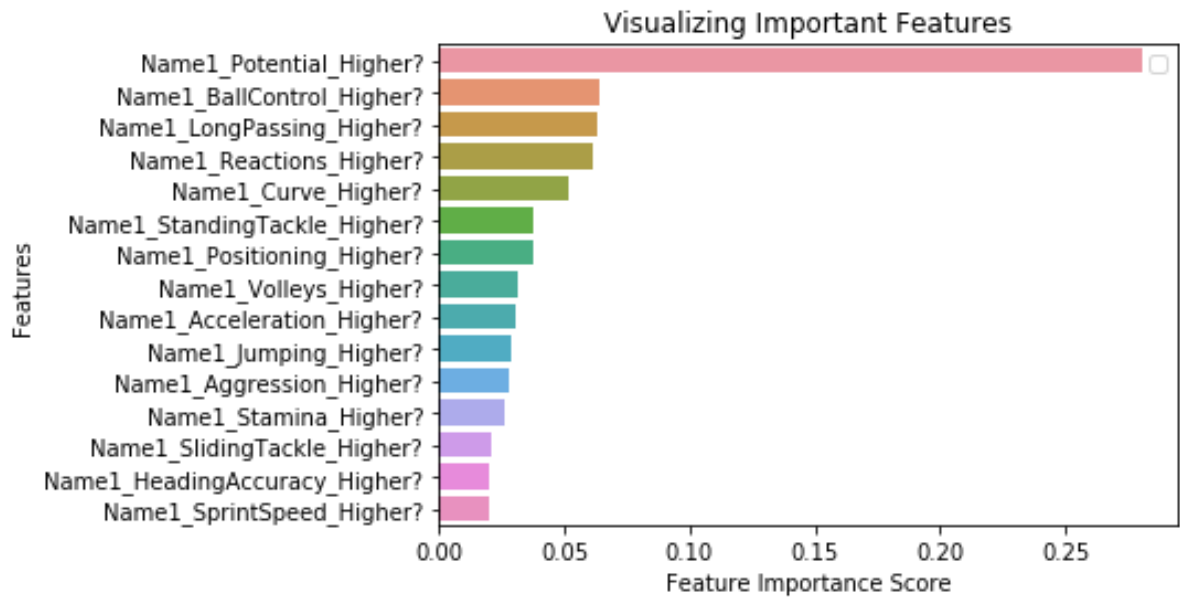


Figure 7. Decision tree feature importance score

Model Performance after Feature Selection

Table 5. Decision tree feature selection results

Evaluation Metric	Value
Training Accuracy	99.14%
Validation Accuracy	87.72%
Testing Accuracy	87.17%
10-fold Cross-validation	86.21%
Stratified cross-validation	89.28%

Stratified K-fold Cross-Validation Results after Feature Selection

Table 6. Decision tree feature selection stratified k-fold cross-validation results

Evaluation Metric	Value
Maximum accuracy	91.52%
Minimum accuracy	86.44%
Overall accuracy	89.28%
Standard deviation	0.016

Area Under the Curve after Feature Selection

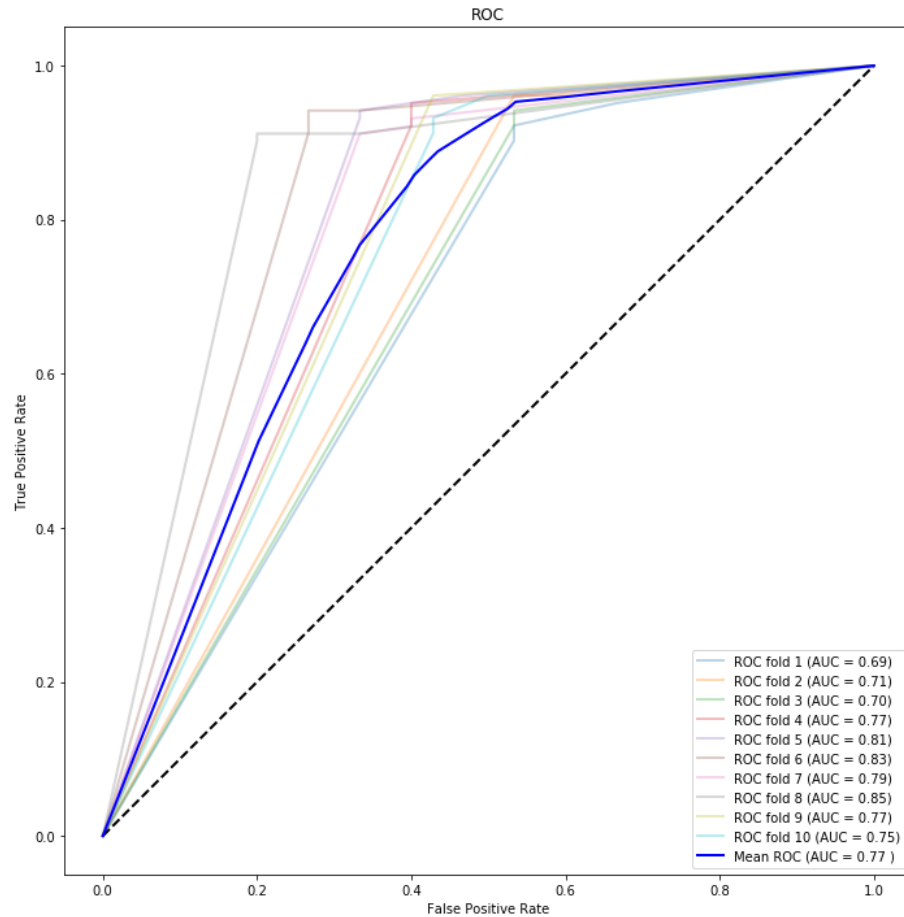


Figure 8. Decision tree feature selection ROC and AUC

Optimized Hyperparameters

- ccp_alpha=0.0
- class_weight=None
- criterion='gini'
- max_depth=4
- max_features=None
- max_leaf_nodes=None
- min_impurity_decrease=0.0
- min_impurity_split=None
- min_samples_leaf=1
- min_samples_split=12,
- min_weight_fraction_leaf=0.0

Model Performance after Hyperparameter Optimization

Table 7. Decision tree hyperparameter optimization results

Metric	Grid-search	Random-search
Training accuracy	95.65%	94.52%
Validation accuracy	94.41%	93.15%
Testing accuracy	93.26%	92.23%
Cross-validation	93.67%	92.63%

Random Forest Model

Baseline Model Results

Table 8: Random forest baseline model results

Evaluation Metric	Value
Training accuracy	100%
Validation accuracy	88.38%
Testing accuracy	63.85%
10-fold cross-validation	89.08%
Stratified k-fold	92.43%

Stratified K-Fold Cross-Validation Results

Table 9: Random forest stratified k-fold cross-validation results

Evaluation Metric	Value
Maximum accuracy	95.76%
Minimum accuracy	89.83%
Overall accuracy	92.43%
Standard deviation	0.0185

Area Under the Curve

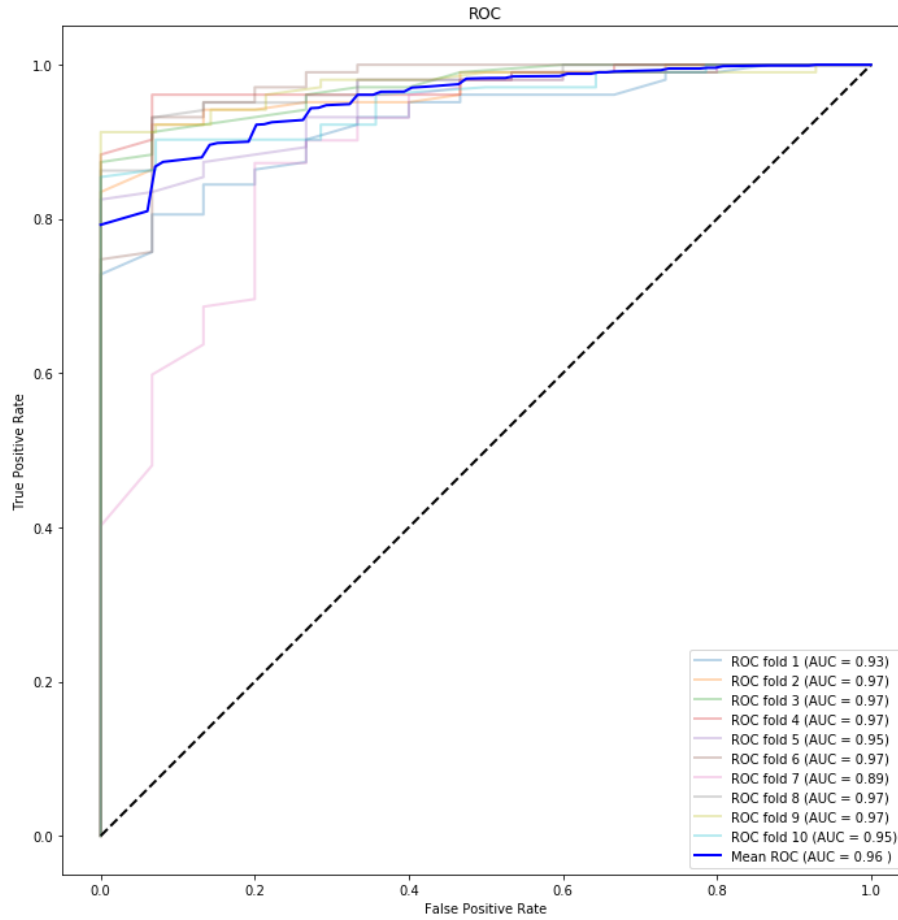


Figure 9: Random forest baseline model AUC

Feature Importance Score

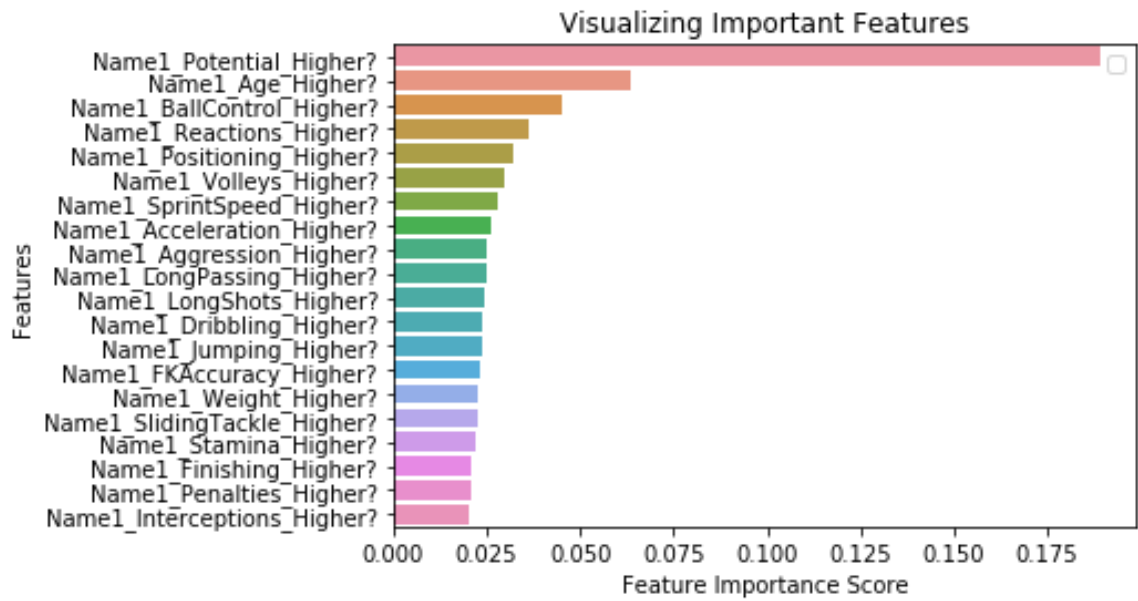


Figure 10: Random forest feature importance score

Model Performance After Feature Selection

Table 10: Random forest feature selection results

Evaluation Metric	Value
Training Accuracy	100%
Validation Accuracy	87.44%
Testing Accuracy	87.17%
10-fold Cross-validation	89.53%
Stratified cross-validation	91.75%

Stratified K-Fold Cross-Validation Results After Feature Selection

Table 11: Random forest feature selection stratified k-fold cross-validation results

Evaluation Metric	Value
Maximum accuracy	94.07%
Minimum accuracy	89.83%
Overall accuracy	91.75%
Standard deviation	0.0144

Area Under the Curve After Feature Selection

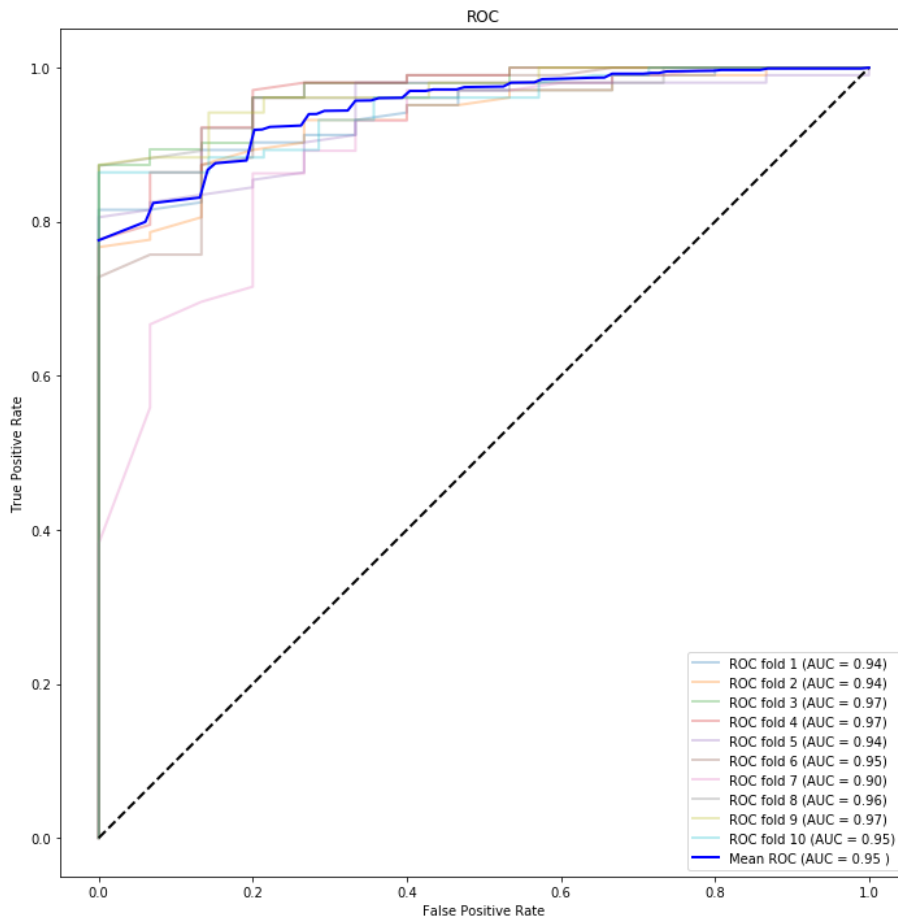


Figure 11: Random forest feature selection ROC and AUC

Optimized Hyperparameters

- bootstrap=False
- ccp_alpha=0.0
- class_weight=None
- criterion='gini'
- max_depth=60
- max_features='sqrt'
- max_leaf_nodes=None
- max_samples=None
- min_impurity_decrease=0.0
- min_impurity_split=None
- min_samples_leaf=1
- min_samples_split=5
- min_weight_fraction_leaf=0.0
- n_estimators=1800

Model Performance After Hyperparameter Optimization

Table 12: Random Forest hyperparameter optimization results

Metric	Random-search
Training accuracy	100 %
Validation accuracy	93.54%
Testing accuracy	93.39%
Cross-validation	94.85%

XGBoost Results

Baseline Model Performance

Table 13: XGBoost baseline model results

Evaluation Metric	Value
Training accuracy	99.14%
Validation accuracy	88.47%
Testing accuracy	69.77%
10-fold cross-validation	89.61%
Stratified k-fold	92.68%

Stratified K-Fold Cross-Validation Results

Table 14: XGBoost stratified k-fold cross-validation results

Evaluation Metric	Value
Maximum accuracy	95.76%
Minimum accuracy	88.88%
Overall accuracy	92.68%
Standard deviation	0.02

Area Under the Curve

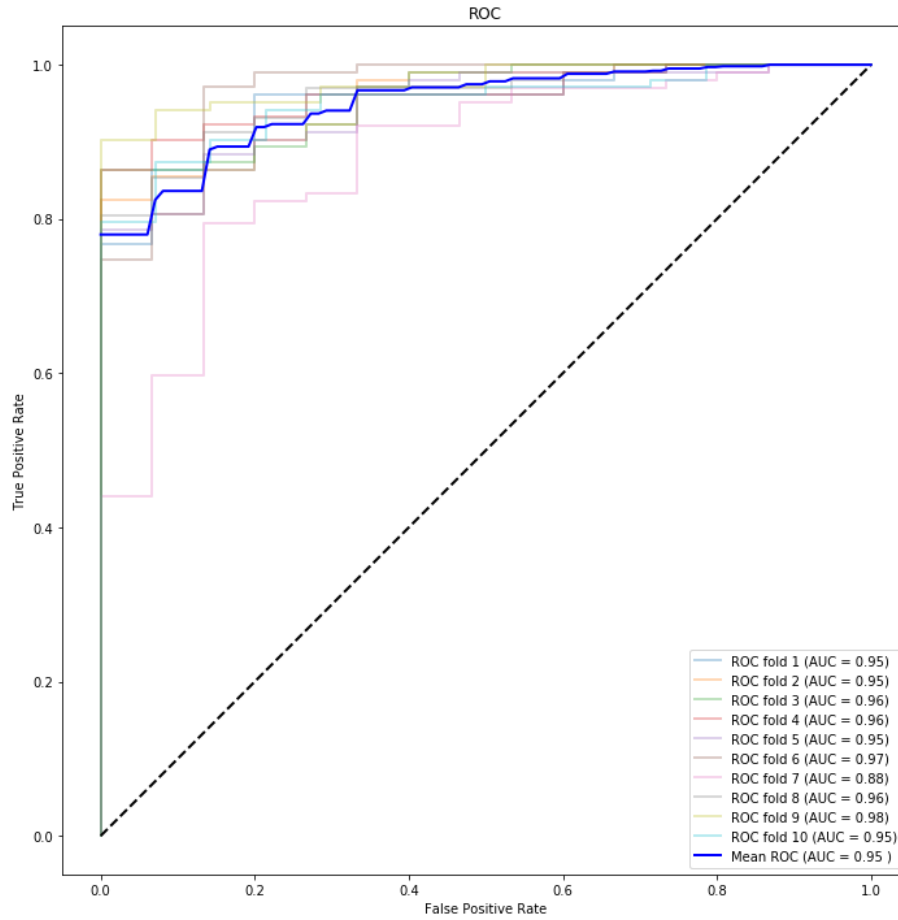


Figure 12: XGBoost baseline model AUC

Feature Importance Score

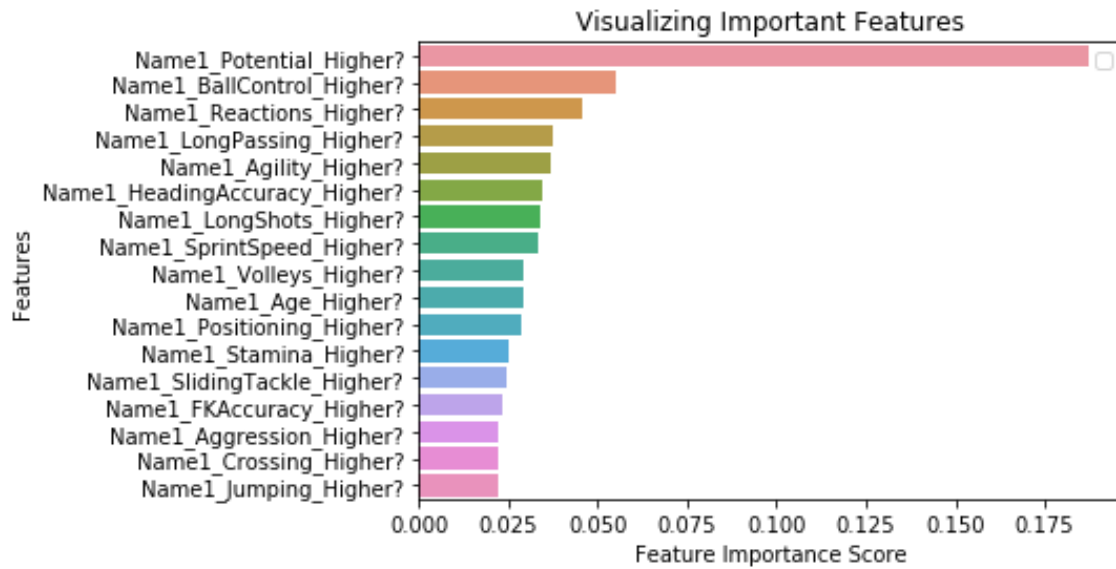


Figure 13: XGBoost feature importance score

Baseline Model Results After Feature Selection

Table 15: XGBoost feature selection results

Evaluation Metric	Value
Training Accuracy	97.43%
Validation Accuracy	87.72%
Testing Accuracy	87.17%
10-fold Cross-validation	90.38%
Stratified cross-validation	92.60%

Stratified K-Fold Cross-Validation Results After Feature Selection

Table 16: XGBoost feature selection stratified k-fold cross-validation results

Evaluation Metric	Value
Maximum accuracy	97.45%
Minimum accuracy	89.74%
Overall accuracy	92.60%
Standard deviation	0.022

Area Under the Curve After Feature Selection

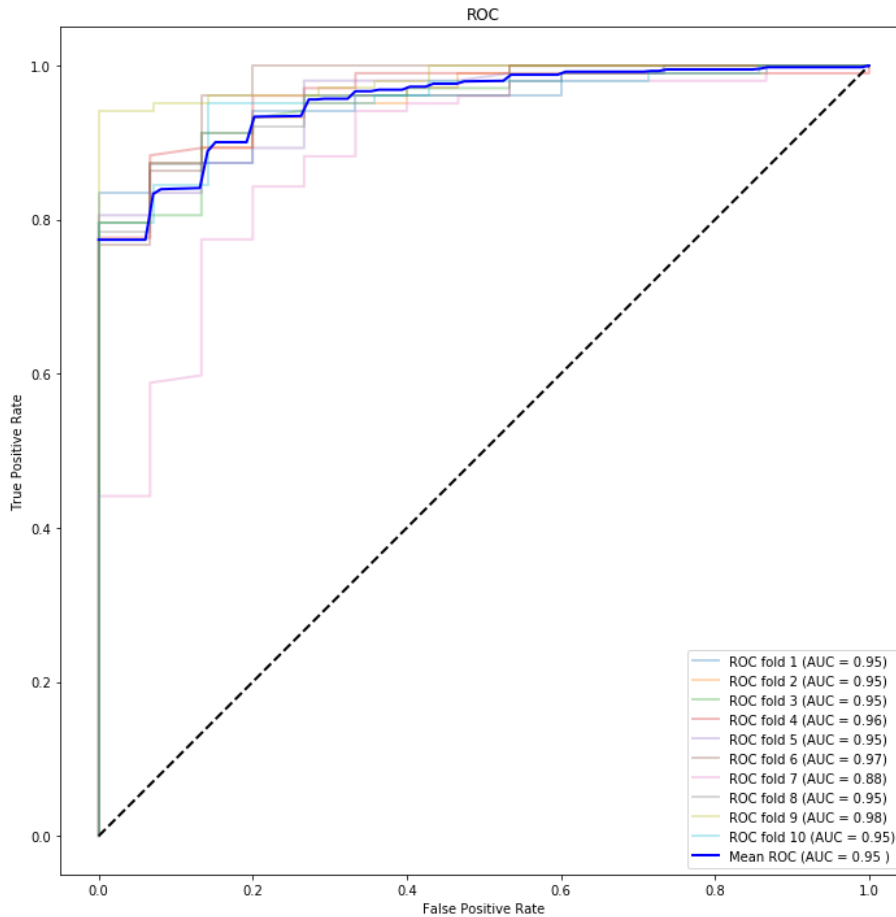


Figure 14: XGBoost feature selection AUC

Optimized Hyperparameters

- base_score=0.5
- booster='gbtree'
- colsample_bylevel=1
- colsample_bynode=1
- colsample_bytree=1
- eta=0.03
- gamma=0
- learning_rate=0.1
- max_delta_step=0
- max_depth=10
- min_child_weight=1
- missing=None
- n_estimators=1600

Model Performance after Hyperparameter Optimization

Table 17: XGBoost hyperparameter optimization results

Metric	Random-search
Training accuracy	98.56 %
Validation accuracy	93.39%
Testing accuracy	92.01%
Cross-validation	95.69%

Borda Count

The final step in implementing this project is generating a list from predictions obtained in the last step. We implemented the algorithm, evaluated different metrics to identify the best and least biased algorithm for a ranking task on an imbalanced data. After we decide on the algorithm of choice, the final step remains to generate a list from those predictions. There are different heuristics and probability functions to generate a ranked list, but the one that is simple, optimal and robust is the borda count.

Borda count is a consensus-based voting system that is used in popular decision-making tasks like political elections and sports awards. In this method, voters rank candidates in order of preference. The borda count is suitable in tasks where a ranking of consensus is important over the majority vote. For this case, we consider each pairwise matchup as an instance. The datapoint that wins the matchup is awarded one point while the one that loses is awarded a negative. The total counts are aggregated to get the final scores for each instance which is used to generate the ranking.

The borda count has following characteristics [24].

- **Simplicity:** The algorithm is simple, as it only orders the items by the number of pairwise comparisons won and those lost. This reduces the order of computation by several degrees as compared to other heuristics.
- **Optimality:** The algorithm derives conditions under which the ranked list is generated optimally by considering all matchups of all instances.
- **Robustness:** The algorithm is quite robust when working with binary predictions of a pairwise comparisons.

Borda count generated list example:

Table 18: Borda count results (top 5)

Top 5 Players from Predictions	Borda Count	Price Rank
P. Galdames	96	1
B. Halimi	91	19
Allan	89	11
Z. Youssouf	89	7
E. Rexhepaj	86	5

Thus, we can see from the list above how we can obtain a top-n list for desired criteria. In this example, a list of midfielders under €3 million was generated. These are termed as utility players in soccer and are quite useful in certain situations of the game. We can also see that the algorithm uncovers some underrated and undervalued players, e.g., the second observation (B. Halimi) is the 19th most expensive player from the pool but is ranked second based on his attributes. Thus, this can be a potential use case of this technique. We can apply it on broader datasets to identify such undervalued players and backtest them against the data. It can be a great extension for this project.

CHAPTER 9. CONCLUSION

All three algorithms show a high-level of accuracy (85%+) on test data when cross-validation is employed. The training models tend to overfit due to what can be a small sample and high bias of the data. Cross-validations tend to reduce this bias with stratified k-fold cross-validation yielding best results. It was expected as the data shows high amount of class imbalance, which is tackled the best in stratified k-fold cross-validation. Feature selection further helps in reducing the noise by eliminating irrelevant features but that is not enough to stop the model from overfitting. Hyperparameter optimization helps in increasing the cross-validation accuracy but not the training bias.

Decision tree model shows 74.74% testing accuracy, which is improved by 12.43 percentage points to 87.17% with feature selection. Hyperparameter optimization increases it further by 6.09 percentage points to 93.26%. There is not much difference between accuracy of grid-search vs. random-search. The grid-search improves the accuracy by 1.1% at the expense of 14,592% increase in processing time. This makes it clear to avoid using grid-search. The stratified cross-fold validation shows good accuracy right from the baseline model. It changes from 89.37% with baseline model to 89.28% with feature selection while increasing to 92.63% with random-search hyperparameter optimization. The mean AUC shows a minimal change from 0.76 to 0.77 reflecting the class imbalance in the dataset.

Random-Forest model shows 63.85% of testing accuracy, which is quite poor compared to the decision tree model. This signals the model overfitting even more. This is further improved 23.32 percentage points to 87.17% with feature selection. Hyperparameter Optimization further increases it by 6.22% to 93.39% with random-search. The stratified k-fold cross-validation model shows a good accuracy right since the start. It shows a high accuracy of

92.43% with baseline model which is slightly reduced to 91.75% with feature selection. This signals that random forest does not work as well with trimmed data. Although we get the best result so far with hyperparameter optimization; an accuracy of 94.85%. The AUC is 0.97 for baseline model whereas 0.96 for feature selection model, which reflects the significant reduction in bias with stratified k-fold cross-validation.

The XGboost is the best model in terms of reducing bias. Since, XGboost iterates over the residuals, it was expected to show lesser bias as compared to other algorithms. The testing accuracy with XGboost is 69.77%, which increases by 17.4 percentage points to 87.17% with feature selection. It is further improved to 92.01% with random-search hyperparameter optimization. The stratified k-fold cross-validation yields the best accuracy; 92.68% with the baseline model, 92.6% with feature selection and 95.69% with hyperparameter optimization. This shows that the XGboost performs best among the models tested.

All models face the issue of class imbalance; with some dealing with it better than the others. It would be of further interest to see how the models work with big data.

REFERENCES

- [1] W. W. Eckerson, "Predictive Analytics: Extending the Value of Your Data Warehousing Investment," TDWI Research, Chatsworth, 2007.
- [2] G. Shmueli and O. R. Koppius, "Predictive Analytics in Information Systems Research," *MIS Quarterly*, vol. 35(3), pp. 553-572, 2011.
- [3] J. R. Evans and C. H. Lindner, "Business Analytics: The Next Frontier for Decision Sciences," Decision Science Institute, Houston , 2012.
- [4] T. H. Davenport, "Competing on Analytics," Harvard Business Review, 2006.
- [5] M. Jordan and T. Mitchell, Machine Learning: Trends, Perspectives, and Prospects, Science, 2015.
- [6] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *24th International Conference on Machine Learning*, Corvallis, 2007.
- [7] A. Shashua and A. Levin, "Taxonomy of large margin principle algorithms for ordinal regression problems," in *NIPS*, 2002.
- [8] G. Lebanon and J. Lafferty, "Cranking: Combining rankings using conditional probability models on permutations," in *19th International Conference on Machine Learning*, 2002.
- [9] R. Herbrich, T. Graepel and K. Obermayer, "Support vector learning for ordinal regression.," in *Ninth International Conference on Artificial Neural Networks*, Edinburgh, 1999.
- [10] Y. Freund, R. Iyer, R. E. Schapire and Y. Singer, "An Efficient Boosting Algorithm for Combining Preferences," *Journal of Machine Learning Research* , vol. 4, pp. 933-969,

2003.

- [11] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton and G. Hullender, "Learning to rank using gradient descent," in *22nd International Conference on Machine Learning*, 2005.
- [12] T. Joachims, "Optimizing search engines using clickthrough data," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.
- [13] X. Li and S. Olafsson, "Discovering Dispatching Rules Using Data Mining," *Journal of Scheduling*, vol. 8, no. 6, p. 515–527, 2005.
- [14] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang and H.-W. Hon, "Adapting ranking SVM to document retrieval," in *ACM SIGIR Int. Conf. Information Retrieval (SIGIR'06)*, 2006.
- [15] H. Liu, Z. Wu and X. Zhang, "CPLR: Collaborative pairwise learning to rank for personalized recommendation," *Knowledge-Based Systems*, vol. 148, pp. 31-40, 2018.
- [16] "Kaggle datasets," [Online]. Available: <https://www.kaggle.com/karangadiya/fifa19>. [Accessed 01 02 2020].
- [17] L. Rokach and O. Maimon, "Decision Trees," in *Data Mining and Knowledge Discovery Handbook*, Springer, 2005, pp. 165-192.
- [18] R. J. Quinlan, "Simplifying decision trees," *International Journal of Man-Machine Studies*, vol. 27, no. 3, pp. 221-234, 1987.
- [19] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24(2), pp. 123-140, 1996.
- [20] J. Brownlee, "Machine Learning Mastery," [Online]. Available: <https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/>. [Accessed 06 02 2021].

- [21] P. Li, "Robust Logitboost and adaptive base class (ABC)," in *Annual Conference on Uncertainty in Artificial Intelligence*, Catalina Island, 2010.
- [22] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *22nd ACM SIGKDD International Conference*, 2016.
- [23] "Classification: ROC Curve and AUC," Google, [Online]. Available:
<https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>.
[Accessed 06 03 2021].
- [24] N. B. Shah and M. J. Wainwright, "Simple, Robust and Optimal Ranking from Pairwise Comparisons," *Journal of Machine Learning Research*, vol. 18, pp. 1-38, 2015.

APPENDIX. THE PYTHON FUNCTION

```

def ptran(data, id):

    df = pd.DataFrame()

    listofnames = []

    k = data.columns.get_loc(id)

    for i, j in itertools.combinations(range(data.shape[0]), r=2):

        listofnames.append((data.iloc[i,k],data.iloc[j,k]))

    df['datapoint_1'] = [datapoint[0] for datapoint in listofnames]
    df['datapoint_2'] = [datapoint[1] for datapoint in listofnames]
    df['id'] = df['datapoint_1'].map(str) + '-' + df['datapoint_2'].map(str)

    df = df.drop(['datapoint_1','datapoint_2'],axis=1)

    df = df.set_index('id')

    del listofnames

    for column in data.columns:

        if column==id:

            continue

        else:

            listofvalues=[]

            for i, j in itertools.combinations(range(data.shape[0]), r=2):

                if data['Name'][i]==data['Name'][j]:

                    continue

                else:

                    listofvalues.append((data[column][i],data[column][j]))

```

```
df[str(id)+"1_"+str(column)+"_Higher?"] = [a[0]>=a[1] for a in listofvalues]
```

```
del listofvalues
```

```
df *= 1
```

```
return df
```