Project Report

Comparative Analysis of Machine Learning Models for Classification of Signals Based on Higgs-Boson Experiments

Akshat Tyagi (at3761)
Hamza Mirza (hm1800)
Pulkit Aneja (pa1304)
Vishal Prabhu (vp1179)

May 3, 2018

Instructor: Professor Juan Rodriguez

ABSTRACT

The discovery of the Higgs-Boson particle was a massive event in the scientific community. This is because Higgs-Boson was the last piece and a major piece of the Standard Model of Particle Physics yet to be discovered. Scientists believe this discovery could prove many theories in Physics, that explain how the universe works.

This project determines whether a Higgs-Boson particle was involved in an experiment or if was just some background noise. As this is a very important topic for physicists to determine the behavior of these particles, we attempt to find the best classification model among the five most popular, that predicts most accurately whether the particle was involved in the experiment. The resulting best classification model can be actually used by the physicists to observe and determine the behavior of these particles.

We analyzed the data with the help of Apache Spark, specifically the Spark ML library. In this report, we have addressed the steps that we performed to analyze the data, and compared the classification models based on various metrics to see which one performs best and is the best fit for this dataset.

ACKNOWLEDGEMENT

We owe a debt of deepest gratitude to our Professor Juan C Rodriguez for his constant guidance, support, motivation, and encouragement throughout the period of this work carried out. His readiness for consultation at all times, his educative comments, his concern and assistance even with practical things have been invaluable. Professor Juan C Rodriguez has been a great source of inspiration for us and we thank him from the bottom of our hearts and we are grateful to him for providing us with the necessary knowledge and guidance for the completion of our project.

Not to forget, we also thank our Teaching Assistants - Shaivi Kochar and Sneha Arkery, for their continuous support and for helping us whenever required.

–

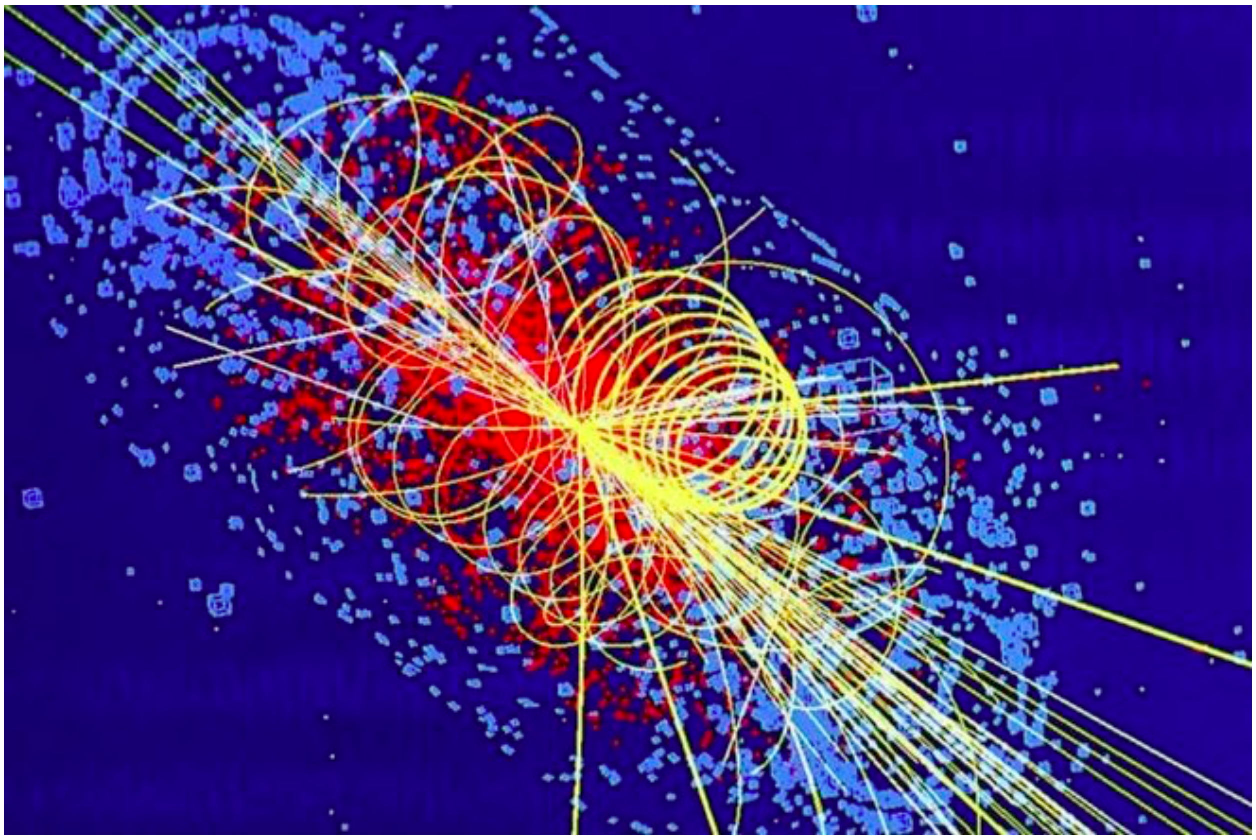
Akshat Tyagi
Hamza Mirza
Pulkit Aneja
Vishal Prabhu

CONTENTS

1 Introduction

The discovery of the Higgs-Boson particle, also called the God Particle, was a massive event in the scientific community. This is because Higgs-Boson was the last, and a major piece of the Standard Model of Particle Physics yet to be discovered. This discovery helps us understand how particles and forces interact in the universe.

According to physicists, this could help prove some long-held theories in Physics, for example, the theory that Higgs-Bosons are responsible for the mass of matter and that mass is not implicit. If proven, such theories could possibly change the way we see and perceive the universe. This also takes us one step closer to one of the major goals of Physics - “The Theory of Everything”.



2 Methodology

2.1 Dataset

The dataset was obtained from a Kaggle competition held in 2014. This dataset is a subset of a larger dataset provided by the University of Irvine Machine Learning Repository. This dataset was provided by actual physicists working at the Large Hadron Collider (LHC), the world's largest particle accelerator at CERN Laboratory in Geneva, Switzerland. The dataset contains data from hundreds of thousands of experiments conducted.

The dataset includes:

- 30 features corresponding to 30 columns
- 1 label column that specifies whether the experiment included a Higgs-Boson particle or was it background noise
- 1 column corresponding to the event-id

Link to dataset:
Kaggle dataset (smaller dataset): https://www.kaggle.com/c/higgs-boson/data
Actual dataset: https://archive.ics.uci.edu/ml/datasets/HIGGS

2.2 Tech Stack

We implemented the project using the following technologies:

- Apache Spark
- Spark MLlib
- Tableau
- Apache Zeppelin Notebook
- Docker

2.3 Data Preprocessing and Cleaning

The data provided was pretty much clean and did not need a lot of preprocessing. This was a huge advantage as it saved us a lot of time to focus on the actual implementation. However, there were some things we needed to modify.

1. **Label Encoding:** The labels provided by were characters, and because machine learning only works on numerical values, we needed to encode the labels. We changed
   s -> 1
   b -> 0

2. **Removing Missing Values:** The dataset contained some missing values that needed to be filled. We changed them to -999, indicating the experiment did not include that feature. We also dropped the Event-id column as it was of no use to us.

3. **Normalization and Feature Scaling:** In order to make sense of the data, it needs to be in a similar format and range. To do this, we normalized and scaled our data so that they are in the same range and outliers do not drastically affect it.

4. **Data Split:** In order to train models, we need to split data into training data and testing data that are, as the name suggests, used for training and testing data for our model. We 70-30, where 70% was used for training and 30% for testing.

2.4 Classification Models Used

- **Multivariate Logistic Regression:** We used a simple logistic regression model, which uses a cost function -ylog(h(x)) - (1-y)log(1- h(x)) for each term to predict for binary classes. To avoid overfitting due to high variance, we have used a regularization term appended to the cost function which penalizes the hypothesis if it overfits. This term is usually ||theta||^2 for all the parameters in the original cost function.

- **Decision Tree:** A decision tree, also called a classification tree, provides classification for a dataset by implementing a tree structure developed using change/gain in entropy through information gain for all the features. Here, we have used information gain as the criterion for split rather than gini. The default maximum depth is 5.

- **Random Forest Algorithm:** a random forest works by creating many weak learning trees which only use m features where m<=p(total features) to predict. These trees use bootstrapping to create bootstrapped datasets with or without replacement. Then, the data which wasn’t used to fit the weak learners is used to train the data. The technique used is bagging. Later, it uses an ensemble technique like majority vote classification to determine a final prediction.

- **Gradient Boosting Tree:** A gradient boosting tree creates residuals from its predictions, which is nothing but the difference between its prediction and the real values. It then uses these residuals to fit the original data using the original features. Based on this, it adds the fitting predictions to the original predictions. It continues to do this till the residuals stop changing in value or some stopping criteria is met. These trees have been doing very well in kaggle competitions. They could use forward stepwise fitting or square error loss fitting. Adaboost is another example, which gives certain weight to all the data points, with misclassified points receiving extra weight.

- **Multilayer Perceptron:** A multilayer perceptron is a neural network consisting of some hidden layers which can be used to perform both supervised and unsupervised learning tasks. It uses certain activation functions which non-linearly map the outputted probability between 0 and 1(like in sigmoid) or -1 and 1(like in hyperbolic tangent). Each layer has some weights associated with it, which, together with the nodal values of the layer, computes some cost function z, which is then passed to the activation function. The output is the input of the next layer. After a forward pass, back propagation is performed which computes the gradients used to correct the weights. In our experiment, we used three layers as hidden layers, having the same number of nodes as the features.

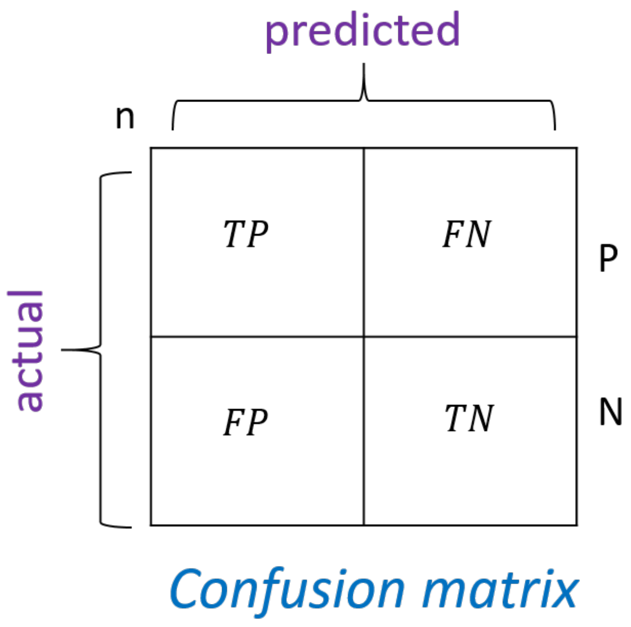
2.5 Evaluation Metrics

- **True Positive Rate/ Precision:** TP / (TP+FP).
- **False Positive Rate:** FP / (FP + TN)
- **Recall/Sensitivity:** TP/(TP+FN).
- **F1 Score:** 2*((Precision*Recall)/(Precision+Recall))
- **Area Under ROC Curve:** Area under the Receiver Operating Characteristic Curve
- **Area Under PR Curve:** Area under Precision-Recall Curve
- **Accuracy:** Ratio of correct classifications to total number of classifications.

3 Results and Discussion

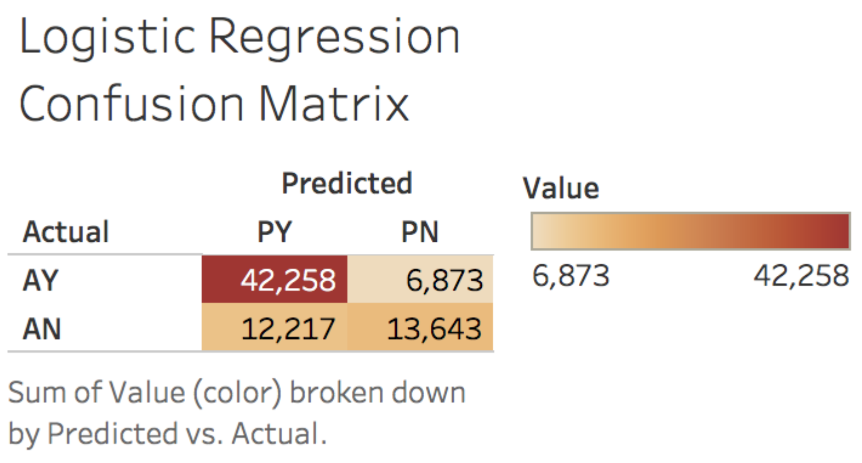
3.1 Confusion Matrix Comparison

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.
Example confusion matrix for a binary classifier:



Here, TP = True Positive, TN = True Negative, FP = False Positive, and FN = False Negative. There are two possible predicted classes: "yes" (1) and "no" (0). If we were predicting the presence of a higgs boson particle, for example, 1 would mean Higgs-Boson particle was a part of the experiment, and 0 would mean it was just background noise.

Let us look at the results now:



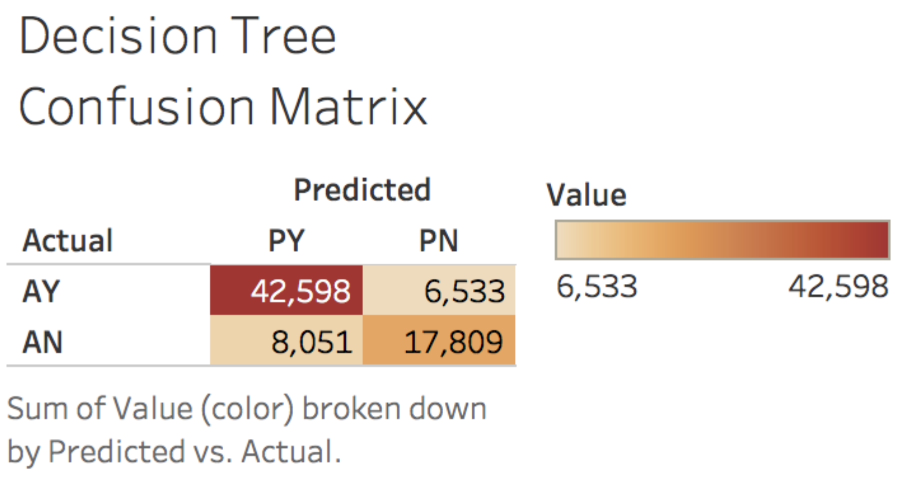
Logistic Regression Confusion Matrix

Actual	Predicted PY	Predicted PN
AY	42,258	6,873
AN	12,217	13,643

Value
6,873
42,258

Sum of Value (color) broken down by Predicted vs. Actual.

The Logistic Regression classifier made the correct prediction for around 75% of values. This is a decent rate, but there are still a lot of wrongly predicted values.



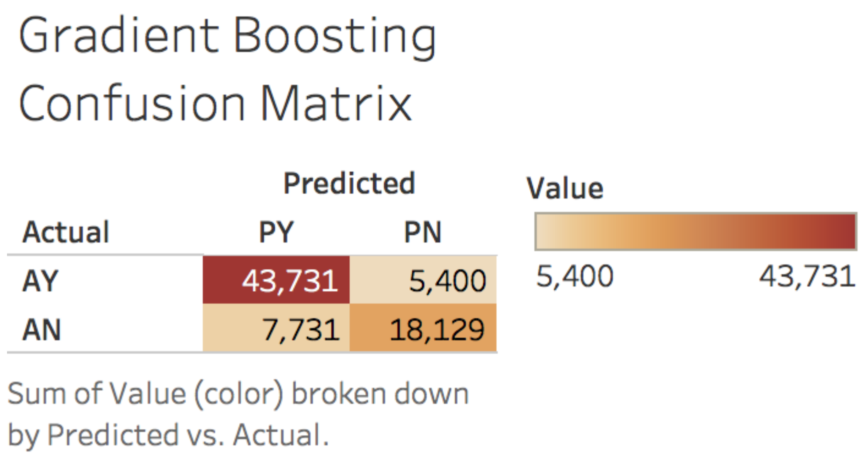
Decision Tree Confusion Matrix

Actual	Predicted PY	Predicted PN
AY	42,598	6,533
AN	8,051	17,809

Value
6,533
42,598

Sum of Value (color) broken down by Predicted vs. Actual.

Decision Tree Classifier did much better than the Logistic Regression Classifier with almost 80% correct predictions.



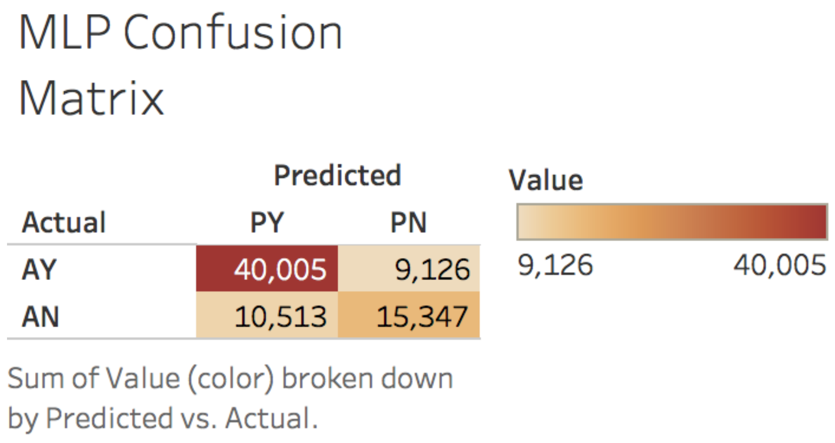
Gradient Boosting Confusion Matrix

	Predicted	
Actual	PY	PN
AY	43,731	5,400
AN	7,731	18,129

Value: 5,400 – 43,731

Sum of Value (color) broken down by Predicted vs. Actual.

The Gradient Boosting Tree Classifier was the best performing classifier with around 83% correctness.



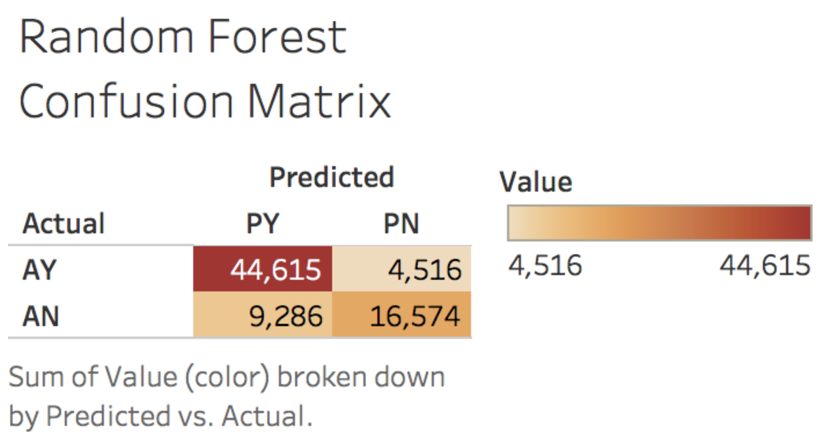
MLP Confusion Matrix

	Predicted	
Actual	PY	PN
AY	40,005	9,126
AN	10,513	15,347

Value: 9,126 – 40,005

Sum of Value (color) broken down by Predicted vs. Actual.

The MultiLayer Perceptron, unexpectedly, had the worst performance even with 3 hidden layers, with just 73% accuracy.



Random Forest
Confusion Matrix

Actual	Predicted PY	Predicted PN
AY	44,615	4,516
AN	9,286	16,574

Value: 4,516 – 44,615

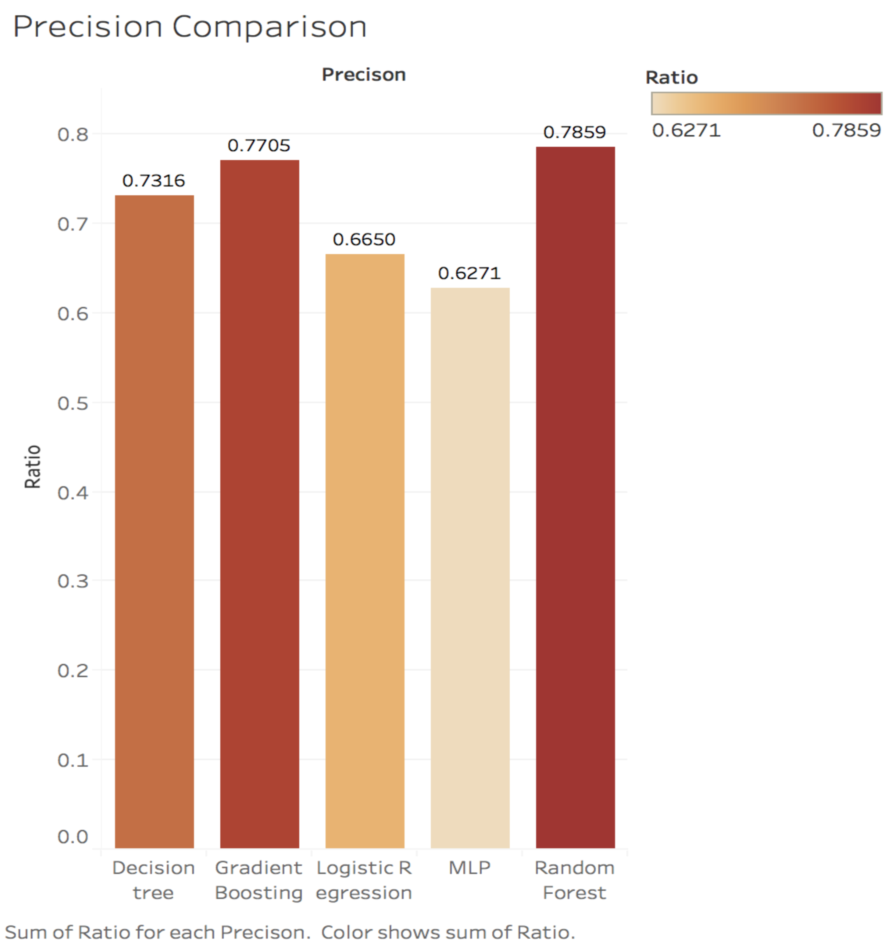
Sum of Value (color) broken down by Predicted vs. Actual.

Random Forest was the second best performer with around 60000 correct predictions out of 75000.

3.2 Precision Comparison

Precision is the fraction of relevant instances among the retrieved instances. It is calculated as the ratio of true positives and the sum of true positives and false positives.

It is calculated as:
TP / (TP+FP)

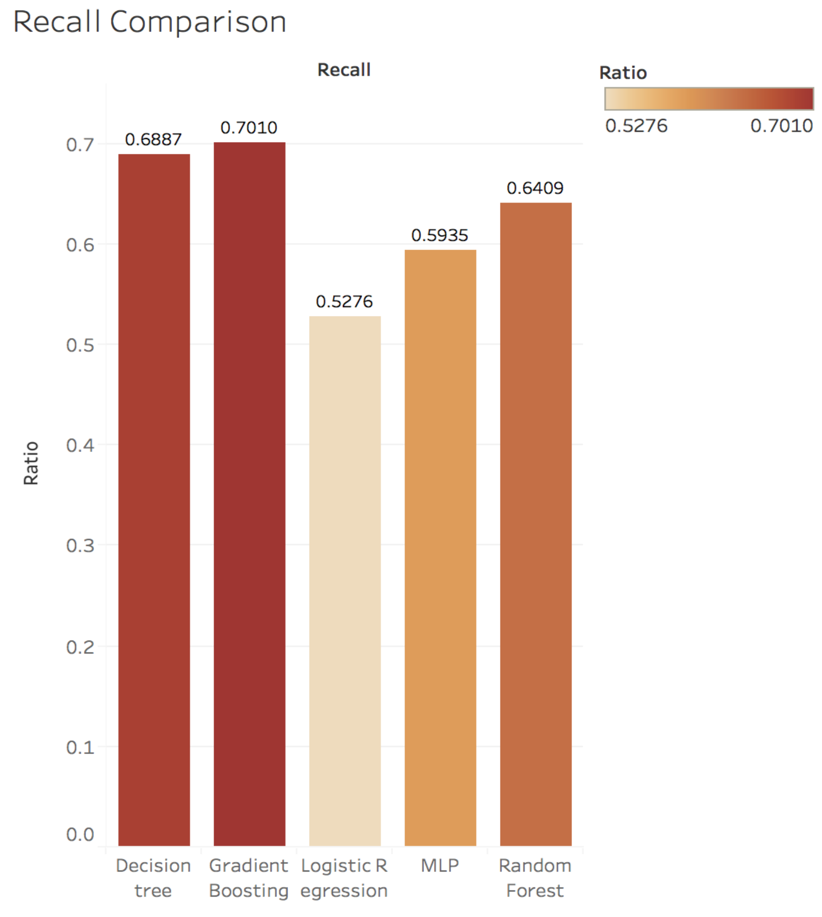


As we can see, Random Forest (78.5%) returned the highest precision rate closely followed by Gradient Boosting Tree (77%) and Decision Tree (73%). Logistic Regression and MultiLayer Perceptron did poorly in comparison with just 66.5% and 62.7% respectively.

3.3 Recall Comparison

Recall is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Both precision and recall are therefore based on an understanding and measure of relevance.

It is calculated as:
TP/(TP+FN)



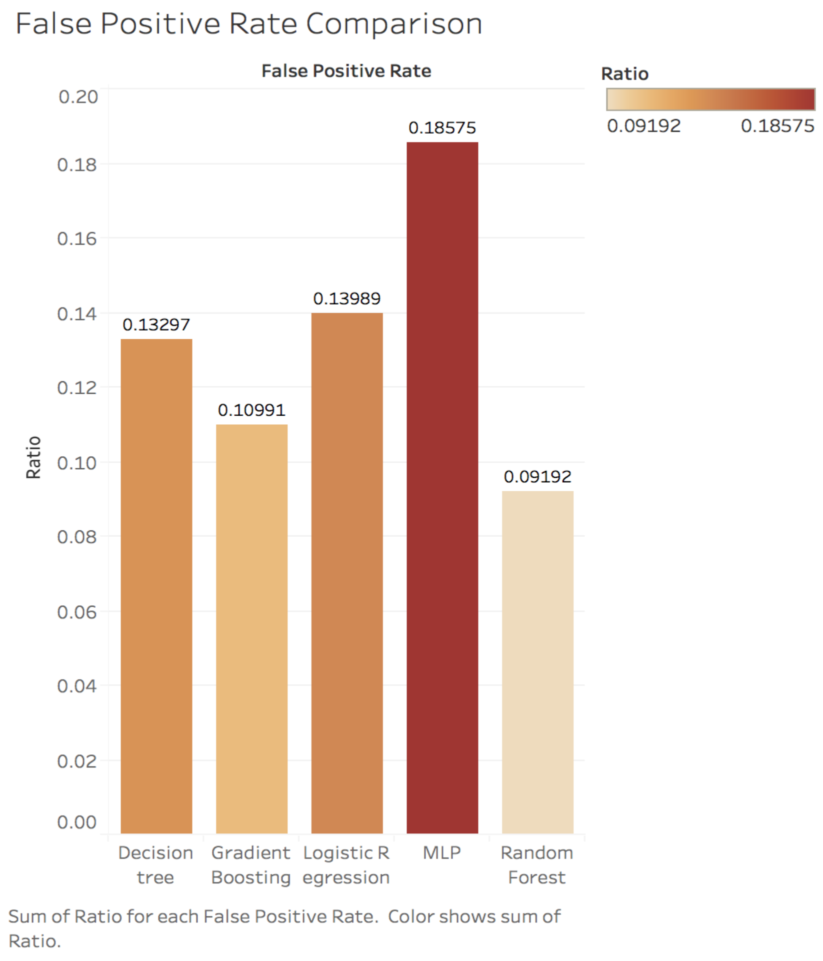
Sum of Ratio for each Recall. Color shows sum of Ratio.

Here we see the Gradient Boosting Tree (70%) provides the best performance followed by Decision Tree (~69%). Random Forest does much worse than it did with precision, with 64% recall rate.

3.4 False Positive Rate Comparison

It is the probability of falsely rejecting the null hypothesis for a particular test. The false positive rate is calculated as the ratio between the number of negative events wrongly categorized as positive (false positives) and the total number of actual negative events.

It is calculated as:
FP / (FP + TN)

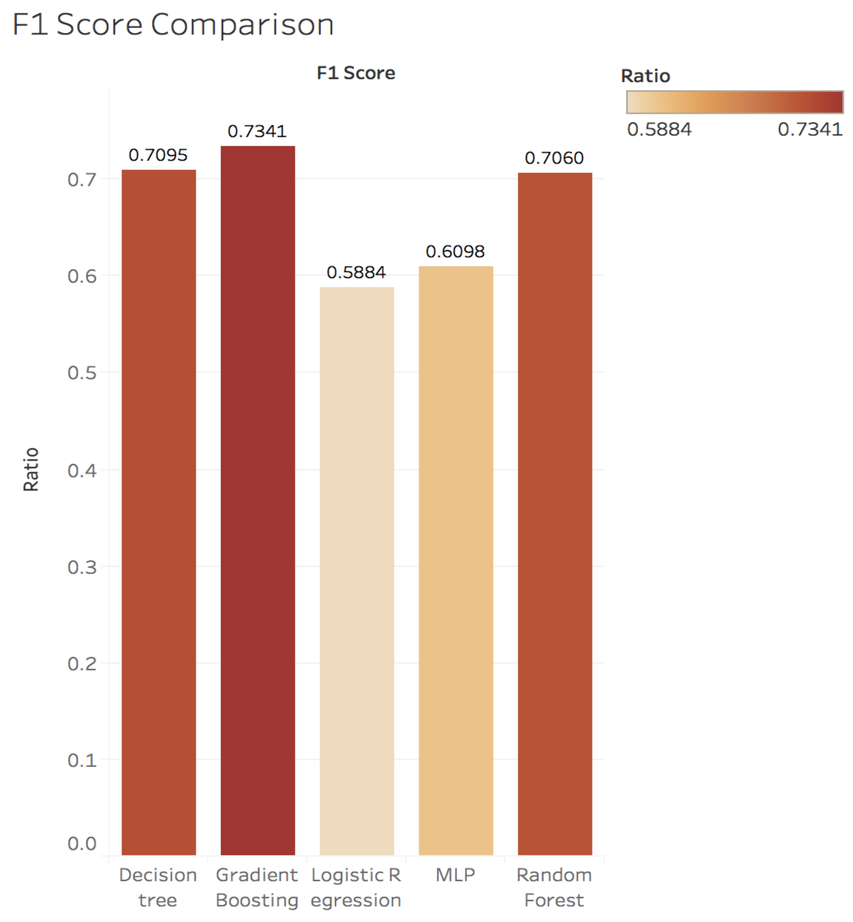


The False Positive Rate is the highest for MultiLayer Perceptron with 18.5% indicating the worst performance. Random Forest and Gradient Boosting Tree do well again with the two lowest False Positive Rates of 9% and 11% respectively.

3.5 F1 Score Comparison

It is a measure of a test's accuracy. It considers both the precision and the recall of the test to compute the score. The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

It is calculated as:
2 * ((Precision*Recall) / (Precision+Recall))

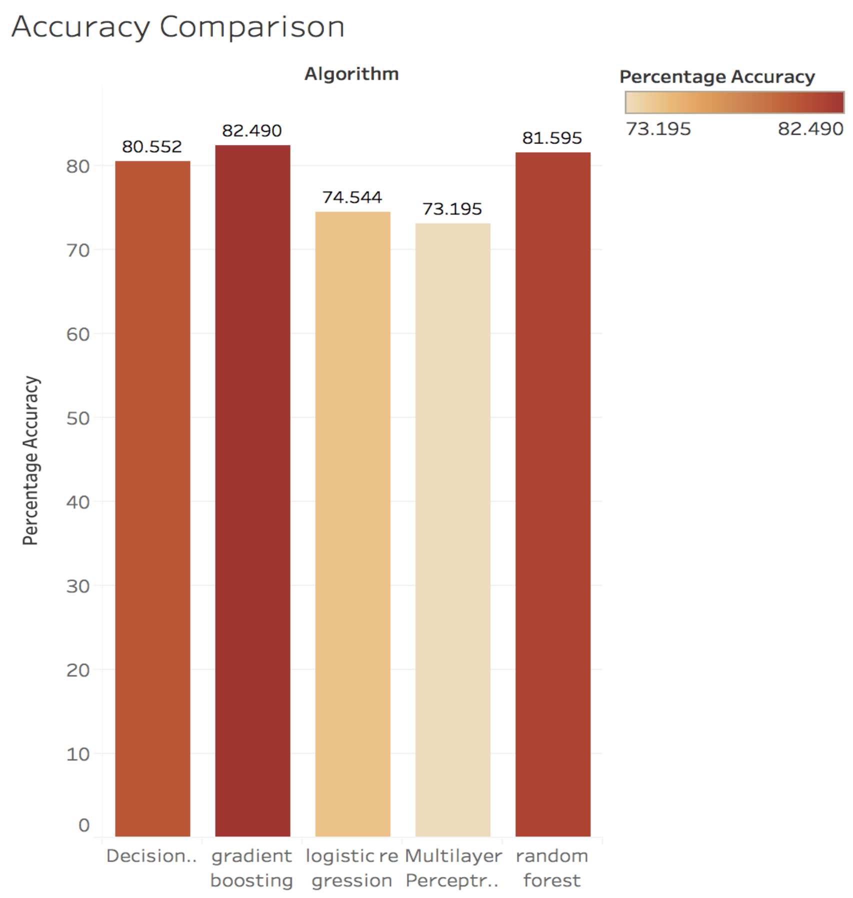


Sum of Ratio for each F1 Score. Color shows sum of Ratio.

The best F1 Score performance was given by Gradient Boosting Tree (73.5%) closely followed by Decision Tree (71%) and Random Forest (70.5%). Logistic Regression had the worst performance in this case with just 58%.

3.6 Accuracy Comparison

Accuracy is simply the ratio of correct classifications to total number of classifications.

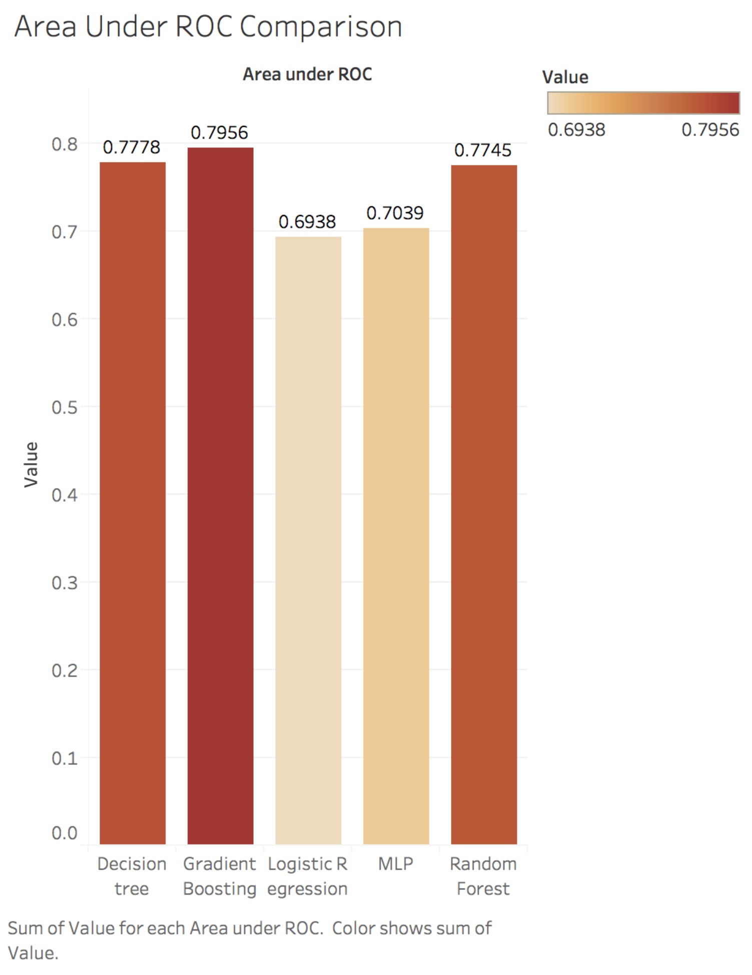


Sum of Percentage Accuracy for each Algorithm. Color shows sum of Percentage Accuracy.

Accuracy is arguably the most important metric among these as it is a direct indicator of the best performing algorithm overall. Again, we see Gradient Boosting Tree acing the metric with 82.5% accuracy followed by Random Forest with 81.5%. MultiLayer Perceptron had the worst accuracy among all with ~73.2%.

3.7 Area Under ROC Curve Comparison

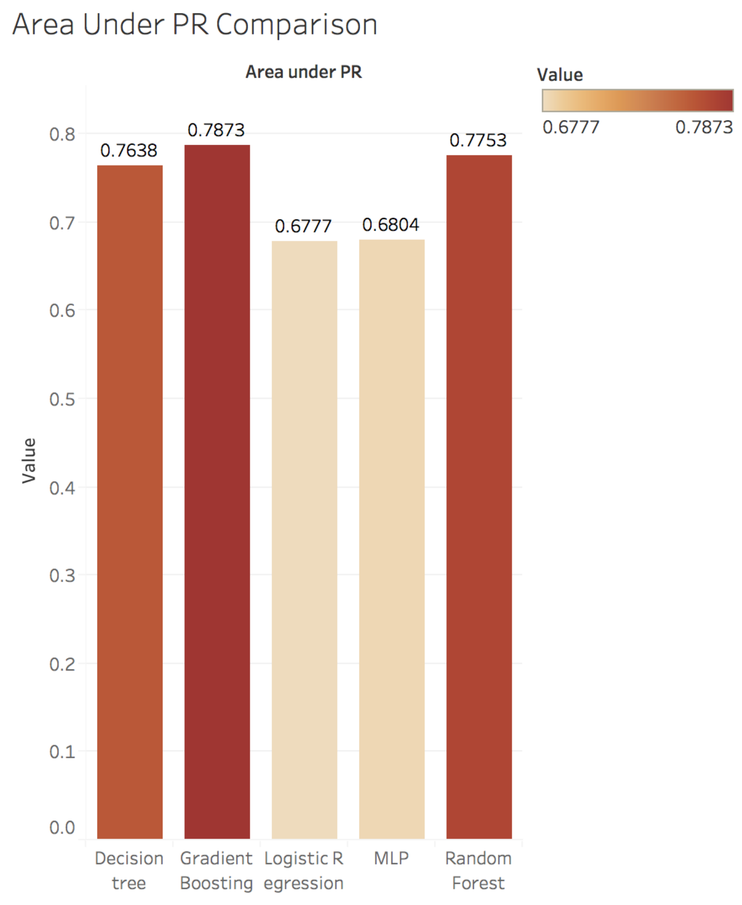
A Receiver Operating Characteristic Curve, i.e. ROC Curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The area under this curve is called the Area Under ROC Curve.



As expected, the Area under ROC value was the highest for Gradient Boosting Tree at 79.5%, closely followed by Decision Tree (77.7%) and Random Forest (77.4%). Logistic Regression had the lowest value with 69%.

3.8 Area Under PR Curve Comparison

Precision-Recall Curve is the curve resulting from plotting the Precision along the y-axis and Recall along the x-axis. The area under that curve is called the Area Under PR Curve.



Sum of Value for each Area under PR. Color shows sum of Value.

Again, Gradient Boosting Tree (78.7%) did the best followed by Random Forest (77.5%). Logistic Regression and MultiLayer Perceptron had almost equally low performances of around 68%.

4 Conclusion

We observed that Gradient Boosting Tree had the most best performances among all the metrics, followed by Random Forest. The worst performer among all the metrics combined was Logistic Regression closely followed by MultiLayer Perceptron. Overall, the results met our expectations except MultiLayer Perceptron did a lot worse than we had anticipated. It would, probably, have done much better with a few more hidden layers.

Tree-based classifiers did the best overall, mostly because of the number of features to work on. The higher the number of features, the better is the performance of tree-based classifiers and the worse the performance of Logistic Regression.

Hence, we conclude that for this particular dataset, tree-based classifiers or a modification of tree-based classifiers does the best simply because of the number of features to be considered.

5 Code and References

5.1 Code

The code is provided as five json files, one file for a model each, along with this report and the presentation.

5.2 Dataset

UCI ML Datasets (https://archive.ics.uci.edu/ml/datasets/HIGGS). File size is about 8GB. For the demonstration, we used a part of this dataset, available on Kaggle (https://www.kaggle.com/c/higgs-boson/data).

5.3 References

- https://www.kaggle.com/c/higgs-boson/data
- https://archive.ics.uci.edu/ml/datasets/HIGGS
- https://spark.apache.org/mllib/
- https://spark.apache.org/sql/
- http://stackoverflow.com