BIRZEIT UNIVERSITY

Faculty of Engineering and Technology
Electrical and Computer Engineering Department

ENCS5141 INTELLIGENT SYSTEMS LAB
Assignment # 2

Student name: Motaz Faraj

Student ID: 1190553

Section: 2

Instructor: Dr. Aziz Qaroush

Date: December 25th, 2023

Abstract

The objective of this case study is to thoroughly compare two well-known ensemble learning techniques, Random Forest and XGBoost, in various scenarios using a data set that simulates real-world scenarios with noisy data, different dimensionality levels, and large datasets. This will help in understanding of the advantages and disadvantages of each technique as well as when and why one might perform better than the other.

Table of Contents

Table of Figure

Table of Tables

1. Literature Review

1.1. Random Forest

Random Forest is a powerful machine learning algorithm for both regression and classification applications. It falls under the general category of ensemble learning, which enhances overall accuracy and robustness by combining the predictions of several weak learners (often decision trees) rather than depending just on one model.

1.1.1. Core Principles

Its fundamental idea is to train many decision trees independently on arbitrary subsets of the training data. This allows it to be constructed using multiple decision trees. Because these trees differ from one another, overfitting is prevented and diversity is introduced. Furthermore, every tree in this model is trained using a bootstrapped sample, which is a random subset that replaces the original data. By doing so, variance is decreased and various facets of the data are captured. Furthermore, only a random subset of features is evaluated at each split point in a tree as opposed to all features. In addition to increasing diversity, this keeps individual trees from becoming overly dependent on particular traits.

The final prediction for classification is decided by the individual trees' majority vote. The average prediction is used in regression. As a result, the predictions stabilize and the overall error is decreased.

1.1.2. Training Methodologies and Hyper-parameters

For its training methodologies random forest uses bagging, where trees are independent and focus on different areas of the data.

For its key hyper-parameters The number of trees (n_estimators), the maximum depth of trees (max_depth), the minimum number of samples per leaf (min_samples_leaf), and the number of features considered at each split (max_features) are crucial hyper-parameters for controlling the complexity and performance of the forest. As for its learning rate random forest has a fixed learning rate for all trees. And this can be compensated by hyper-parameter tuning to adjust the number of trees.

1.1.3. Advantages and Limitations

Because of its diversity, Random Forest is a well-known algorithm for its high accuracy in both classification and regression tasks as well as its resilience to overfitting. Its feature importance is another benefit since it offers insight into feature importance, which is necessary to understand which features have the biggest influence on the predictions.

In addition to its benefits, random forests have certain drawbacks, one of which is interpretability given that some decision trees are intricate and difficult to understand directly. Another is that the forest's inner workings can be obscure, making it challenging to pinpoint the precise causes of its predictions. This contributes to the forest's nature as a "black box." The adjustment of hyper-parameters, which can be difficult and computationally costly, is another drawback.

1.2. XGBoost

Extreme Gradient Boosting, or XGBoost, is a powerful machine learning algorithm for tasks involving regression and classification. It is part of the boosting family, as opposed to Random Forest, where models are constructed one after the other, getting more accurate with each iteration.

1.2.1. Core Principles

XGBoost gradually builds an ensemble of weak learners (often decision trees) by focusing on regions where the preceding tree made an error. As a result, the total error is decreased because every tree gains understanding through the residuals of its predecessors. Moreover, XGBoost uses both bagging and L1 and L2 regularization terms to penalize model complexity in order to prevent overfitting. This stops individual trees from overfitting to particular data points and promotes sparser models with fewer features. A user-defined loss function that is customized for the given task is optimized by XGBoost. This makes it possible to precisely regulate how the model learns.

1.2.2. Training Methodologies and Hyper-parameters

For its training XGBoost uses boosting which means that the training is sequential with every tree focusing on areas the previous one missed, leading to potentially lower bias but higher variance.
As for its key hyper-parameters XGBoost has many hyper-parameters, including learning rate, number of trees, tree depth, regularization parameters, and early stopping criteria.

1.2.3. Advantages and Limitations

Due to its exceptional accuracy, which can be achieved in a variety of tasks, especially when the hyper-parameters are carefully adjusted, XGBoost is a widely used algorithm. Its ability to handle big datasets effectively through parallelization is another benefit of its scalability that allows for parallel processing across multiple cores to speed up training. One additional benefit is that, because it is centered on minimizing residuals, it is resilient to both noisy data and outliers. Additionally, because it offers insights into feature importance and is comparable to random forests in this regard, it helps to explain the behavior of the model.
However, all of these benefits come with some drawbacks, one of which is its interpretability; although feature importance provides some insights, individual trees can be complex, and the ensemble as a whole can be difficult to interpret directly, rendering it somewhat of a "black box". Additionally, it involves hyper-parameter tuning, which, particularly for complicated issues, can be difficult and expert-level. Additionally, compared to simpler algorithms, training XGBoost can be computationally expensive, particularly when dealing with large datasets.

2. Scenarios designed and analysis

For this part different scenarios of data will be used such as noisy data or features, varying degrees of dimensionality and large datasets to compare between the two models.

2.1. Large datasets

For this scenario the effect of large datasets will be studied on both random forest and XGBoost using the datasets TeePublic_review that classify a set or reviews to a positive number between 0 and 5 this dataset contain a total of 247590 data point or review. As a metric for this scenario the training time for both models will be used to see how the model handles this load of data computationally as well as the accuracy, precision, recall, and F1 score to see the performance of the models.

The first thing was to find the optimal parameter using grid search so that the training condition for the two models will be at its best. The following are the optimal parameters for the two models.

Best random forest hyper parameters: {'criterion': 'gini', 'max_depth': 30, 'min_samples_leaf': 4, 'min_samples_split': 5, 'n_estimators': 300}

Best XGBoost Hyper parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200, 'sampling_method': 'uniform', 'subsample': 1}

After the hyper parameters tuning, the training time for each model was calculated on different run and the following figure shows the training time for more than one run for each model.

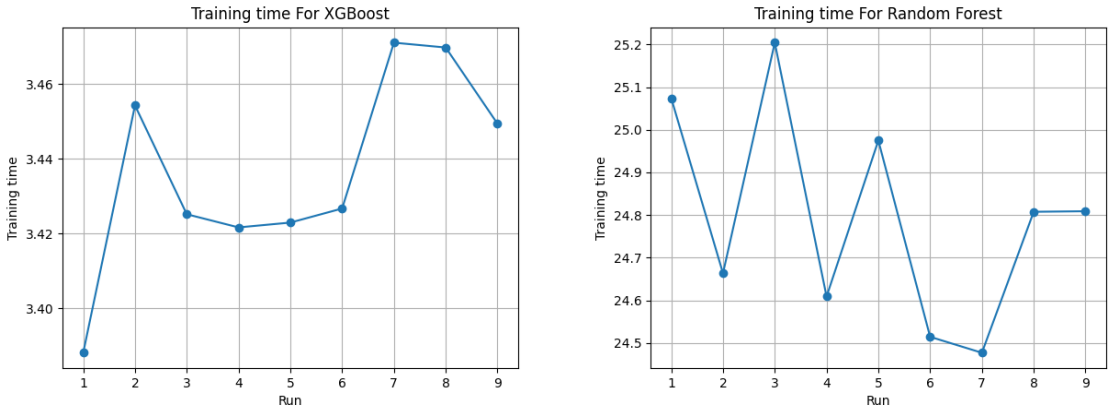

Figure 1: Training time for different runs for both random forest and XGBoost

It is evident that XGBoost outperforms random forest on large datasets due to its use of gradient boosting, an iterative method that creates trees one after the other. Its main goal is to fix the mistakes made by earlier trees, resulting in faster convergence and more effective learning. Additionally, it uses regularization techniques like L1 and L2 to manage tree complexity and avoid overfitting. This frequently leads to faster training of simpler models. Because XGBoost is specifically made to handle sparse data which is common in large datasets it performs exceptionally well when compared to random forest, which is another reason for its exceptional performance. It uses sparse matrix techniques to cut down on memory usage and optimize computations, which speeds up training times and allows for parallel processing across multiple cores enabling it to take full advantage of modern hardware and significantly speed up training.

As for the performance the two models preformed will or approximately the same on large datasets in terms of precision, recall, and F1 score. The following figures shows the results of this training.

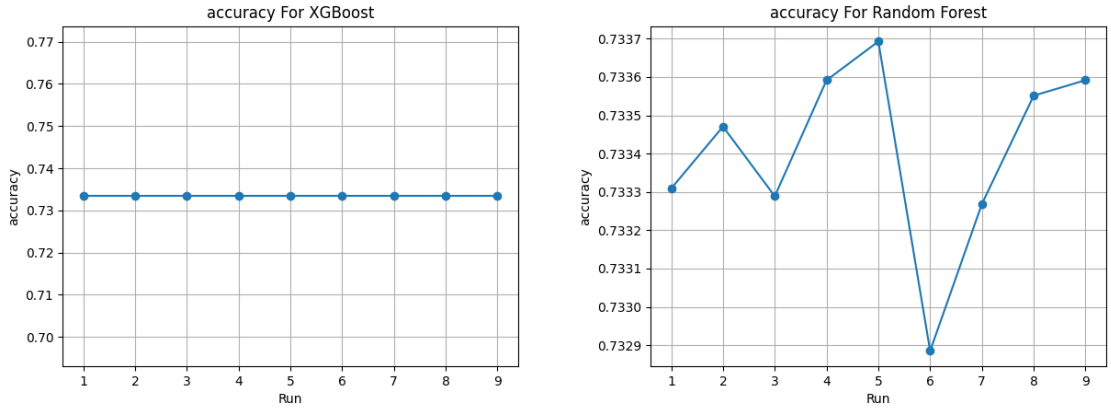

Figure 2: Training results on large dataset for both random forest and XGBoost on the accuracy metric

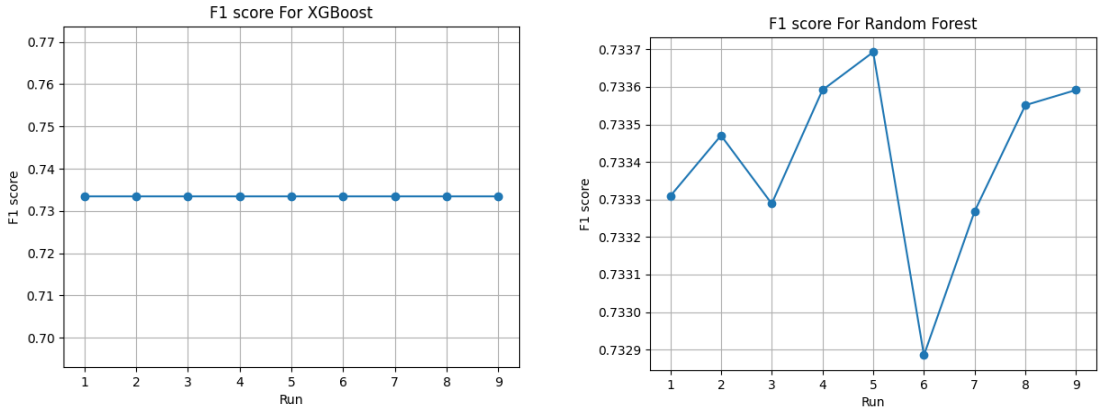

Figure 3: Training results on large dataset for both random forest and XGBoost on the F1 score metric

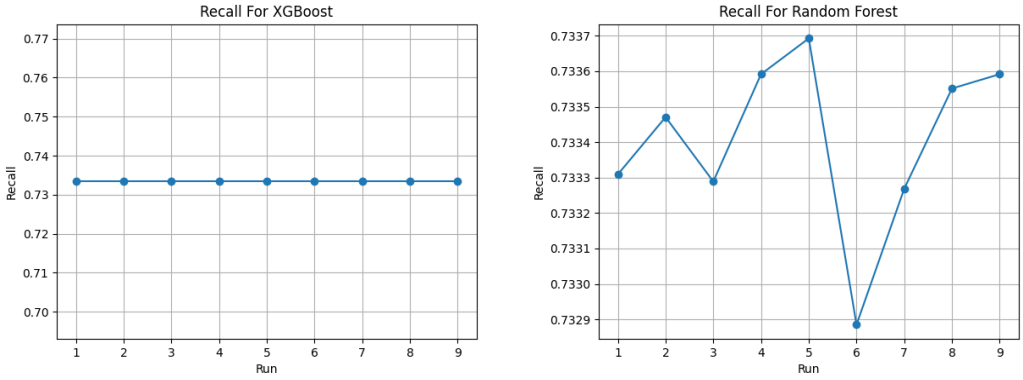

Figure 4: Training results on large dataset for both random forest and XGBoost on the recall metric

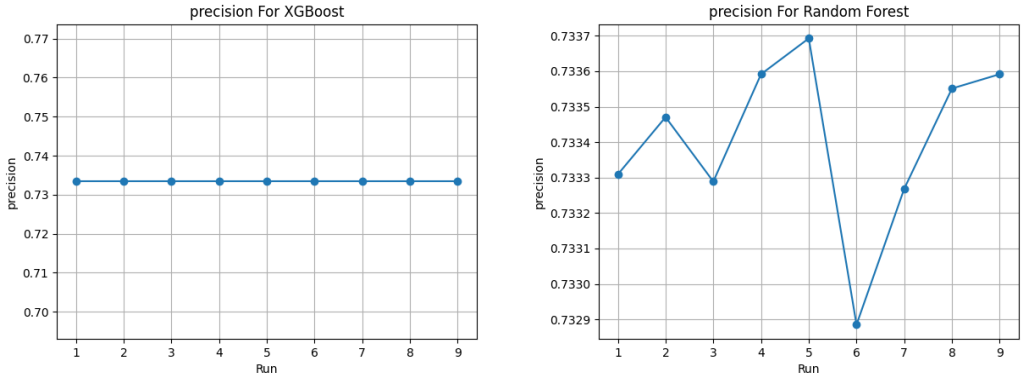

Figure 5: Training results on large dataset for both random forest and XGBoost on the precision metric

Because the problem is a multiclass classification problem, these metrics were selected. Regarding their performance, it's evident that both produced results that were roughly comparable. XGBoost, on the other hand, shows consistency throughout all runs, and random forest varies slightly, but both stay relatively close to one another, with none falling below 0.73 or rising above 0.74. This might be because they use decision trees as their basic building blocks, which are good at handling complex data and capturing non-linear relationships, making them suitable for large datasets. It might also be because hyper parameter tuning, when done correctly, can lead to similar levels of accuracy.

Both algorithms used a significant amount of memory, but it was found that, when it comes to large datasets, XGBoost uses more memory than random forest. Specifically, XGBoost used between 4.8 and 5 GB of system memory, while random forest used between 3.6 and 4 GB. This is because XGBoost builds trees sequentially and stores them in memory for later prediction, which can result in significant memory consumption, especially for large ensembles. Additionally, due to its dense feature representation, the boosting process necessitates explicitly storing all values and adding trees iteratively to correct errors in prior trees leading to the increase in memory consumption

2.2. Noisy data or features

The objective of this scenario is to show and understand which of the two models can deal and minimize the effect of noisy data on the trained model. And to show this the glass was used as this dataset contains a total of 214 data point with 10 features, this dataset was used due to the noticeable amount of outliers in it.

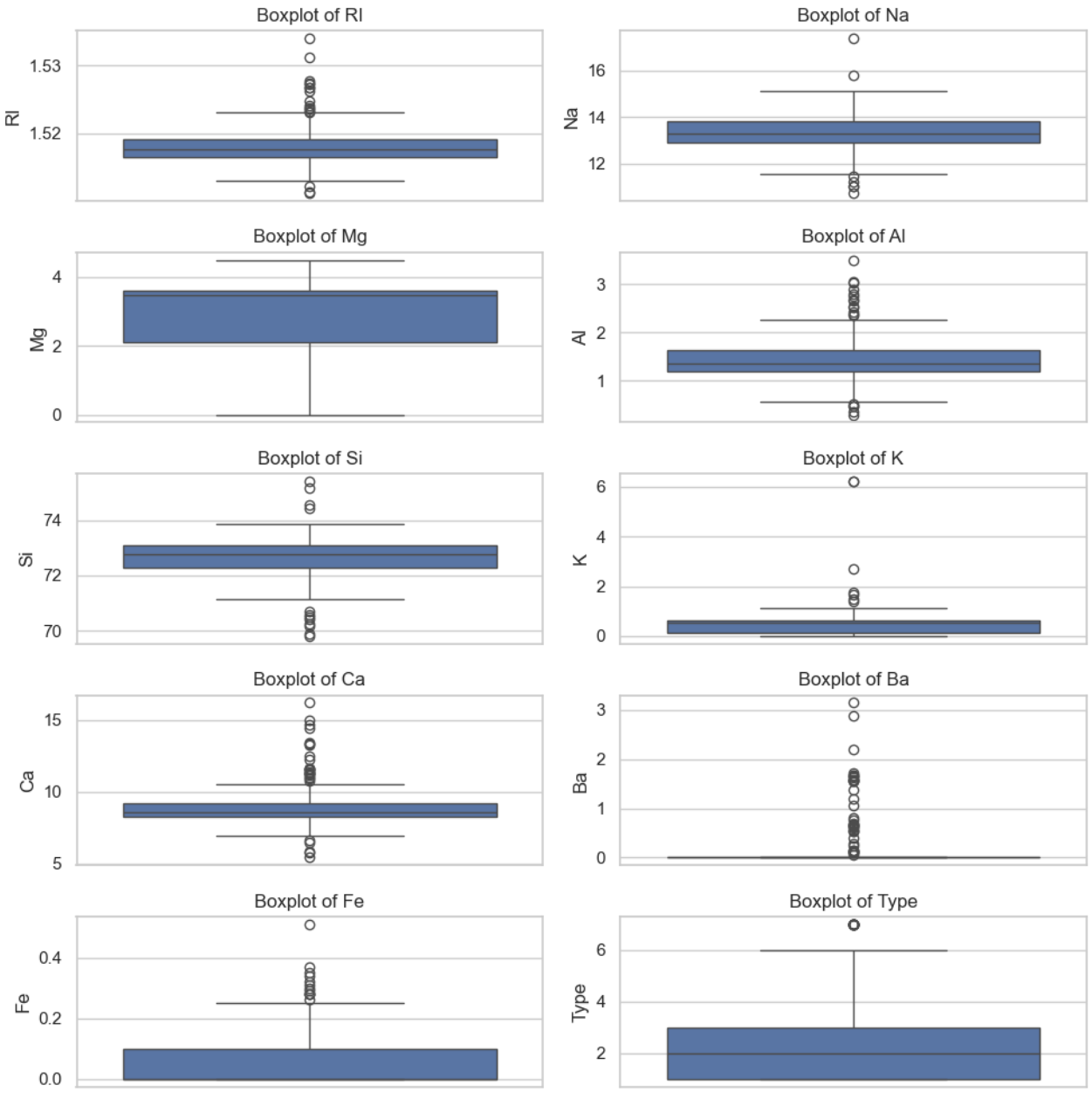

Figure 6: detecting the outliers in the dataset

This scenario will be a classification scenario on which the glass type will be determined by the amount of each element in that glass type. For the evaluation metrics precision, recall, and F1 score as well as acurecy

For the hyper parameters grid search was used to obtain the best case scenario for them with the following parameters.
Best XGBoost Hyper parameters: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 100, 'sampling_method': 'uniform', 'subsample': 1}

Best Random forest Hyper parameters: {'criterion': gini, 'max_depth': 30, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100}

When both models are trained the XGBoost model outperformed random forest due to its iterative learning and sequentially structure of building weak trees so that each tree will try to minimize the error of the previous ones as well as its regularization techniques that often makes it more robust to noise than Random Forest. The following figure shows the Precision, Recall and F1 Score of each model.

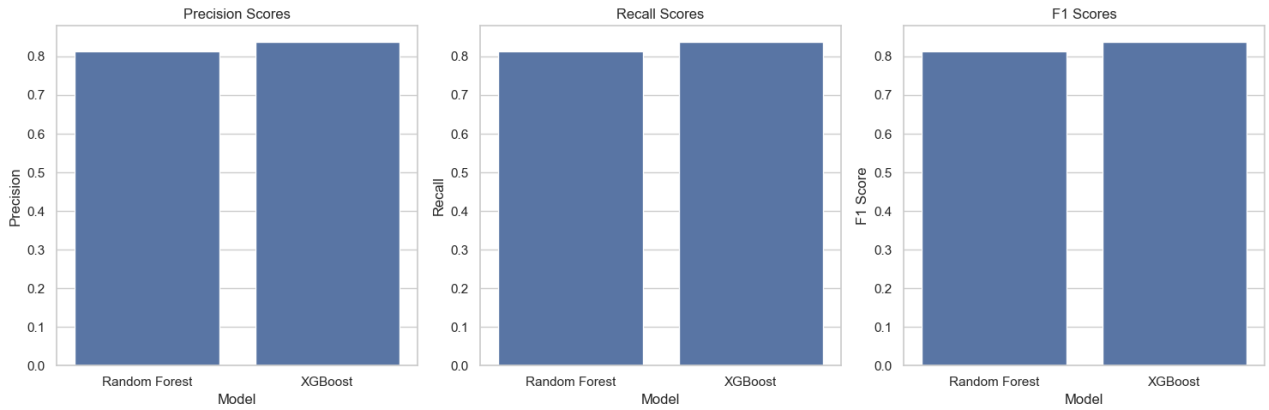

Figure 7: Precision, Recall and F1 Score of random forest and XGBoost with noisy data

The following summarizes the metrics for both random forest and XGBoost

Model\metric	Accuracy	Precision	Recall	F1-Score	Train time
Random forest	0. 8139	0. 8139	0. 8139	0. 8139	0.06726
XGBoost	0. 8372	0. 8372	0. 8372	0. 8372	0.07354

Table 1: Random forest and XGBoost performance on noisy data

As for the training time for both models, it was observed to be less with random forest than XGBoost due to its working mechanism of building multiple decision trees in parallel and combining their predictions, which can lead to efficient training. However, the training time also depends on the depth of the trees and the number of trees in both random forest and XGBoost as well as the size of the tree as XGBoost may perform

better on larger datasets than random forest. The following figure shows the training time for the two models with the best parameters using grid search.

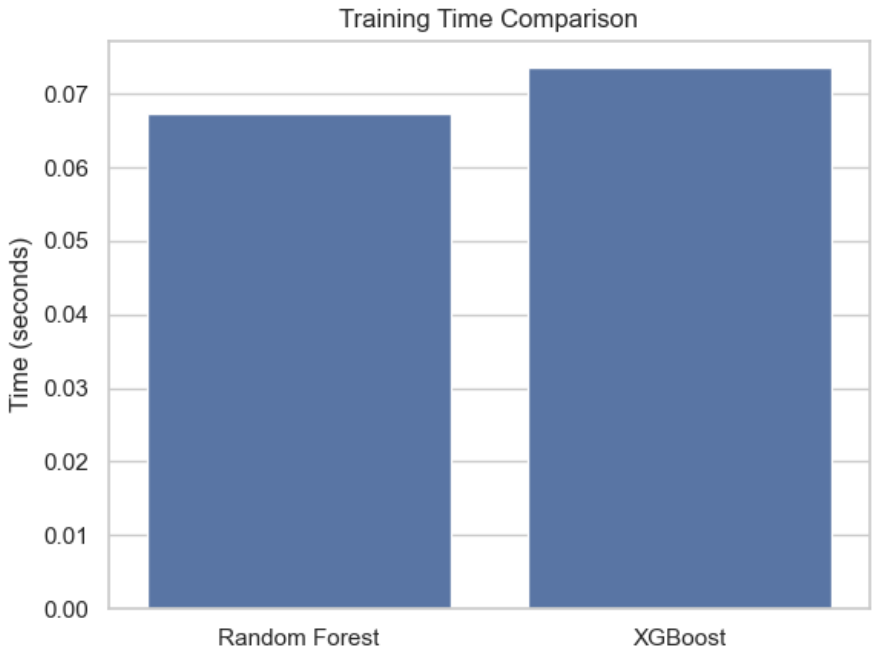

Figure 8: Training time for XGBoost and random forest before hyper parameter tuning

Their memory usage was found to be nearly identical, with random forest utilizing roughly 350MB of memory as opposed to random forest's 100 MB. This could be because of XGBoost's regularization features, which aid in managing noise and result in a little higher memory usage.

2.3. Varying degrees of dimensionality

This scenario uses a dataset with a total of 14 features, the dataset possums include 104 examples. The goal of this scenario is to examine the effects of datasets with large numbers of features, or variables, and what happens when the number of features is varied on the trained models. And to accomplish this the PCA technique will be used in this part.

On the start of each training iteration the best or optimal parameters were calculated using grid search as these parameters will be used for training the data at that same number of n_components or features. The following table shows the optimal parameters used for each feature level.

Feature level\model	Random forest	XGBoost
1	{'criterion': 'absolute_error', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 100}	{'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 100, 'sampling_method': 'uniform', 'subsample': 0.8}
2	{'criterion': 'absolute_error', 'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 300}	{'colsample_bytree': 1.0, 'learning_rate': 0.01, 'max_depth': 8, 'n_estimators': 100, 'sampling_method': 'uniform', 'subsample': 0.5}
3	{'criterion': 'absolute_error', 'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 100}	{'colsample_bytree': 1.0, 'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 100, 'sampling_method': 'uniform', 'subsample': 0.5}
4	{'criterion': 'squared_error', 'max_depth': 30, 'min_samples_leaf': 4, 'min_samples_split': 5, 'n_estimators': 300}	{'colsample_bytree': 1.0, 'learning_rate': 0.01, 'max_depth': 7, 'n_estimators': 200, 'sampling_method': 'uniform', 'subsample': 0.5}
5	{'criterion': 'absolute_error',	{'colsample_bytree': 1.0,

	'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100}	'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 100, 'sampling_method': 'uniform', 'subsample': 0.5}
6	{'criterion': 'absolute_error', 'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100}	{'colsample_bytree': 1.0, 'learning_rate': 0.01, 'max_depth': 8, 'n_estimators': 100, 'sampling_method': 'uniform', 'subsample': 0.5}
7	{'criterion': 'absolute_error', 'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 300}	{'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 100, 'sampling_method': 'uniform', 'subsample': 0.5}
8	{'criterion': 'absolute_error', 'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}	{'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 100, 'sampling_method': 'uniform', 'subsample': 0.5}
9	{'criterion': 'absolute_error', 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}	{'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 100, 'sampling_method': 'uniform', 'subsample': 0.5}
10	{'criterion': 'absolute_error', 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}	{'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 6, 'n_estimators': 100, 'sampling_method': 'uniform', 'subsample': 0.5}
11	{'criterion': 'absolute_error', 'max_depth': 30,	{'colsample_bytree': 1.0, 'learning_rate': 0.2,

	'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}	'max_depth': 6, 'n_estimators': 100, 'sampling_method': 'uniform', 'subsample': 0.5}
12	{'criterion': 'absolute_error', 'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}	{'colsample_bytree': 0.8, 'learning_rate': 0.01, 'max_depth': 8, 'n_estimators': 200, 'sampling_method': 'uniform', 'subsample': 0.5}

Table 2: Hyper parameters at each feature level from 1 to 12

Since this is a regression problem, to find the age of the possums based on its physical property's, so the metric for this problem will be the Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R2) that are commonly used to evaluate the performance of regression models. The following figures shows these values at each feature level.

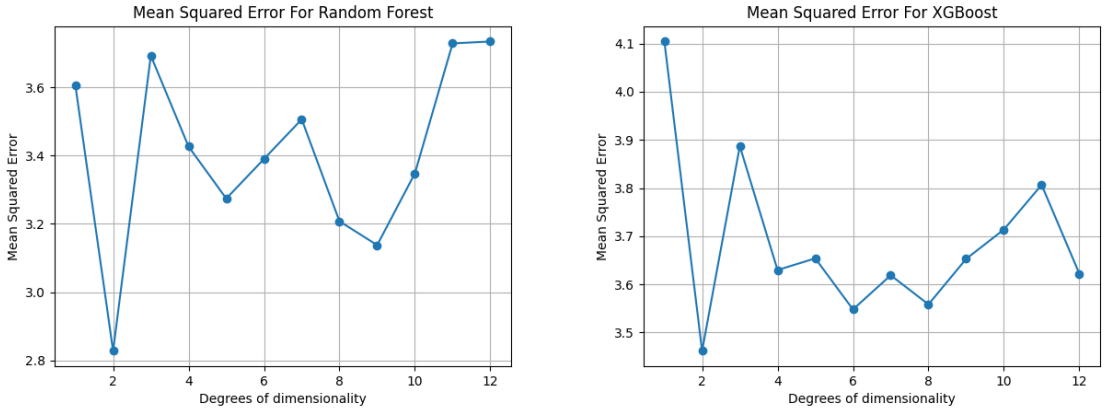

Figure 9: MSE at each feature level

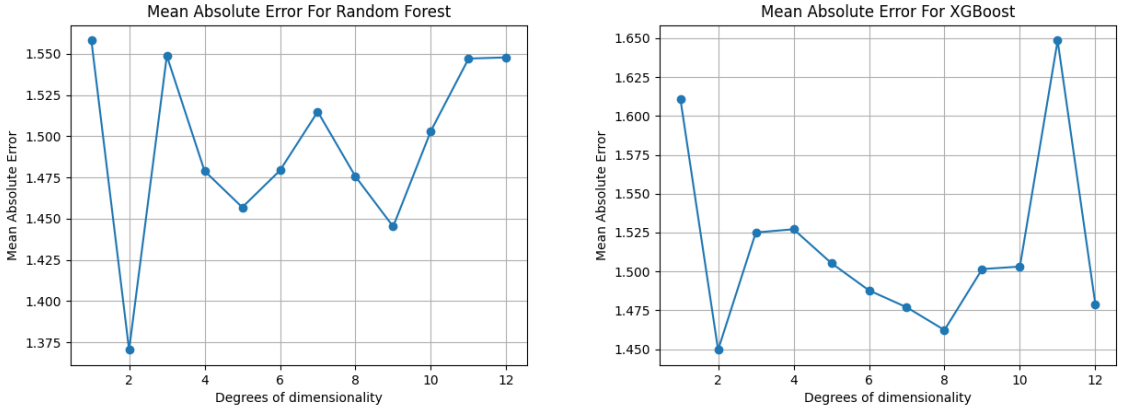

Figure 10: MAE at each feature level

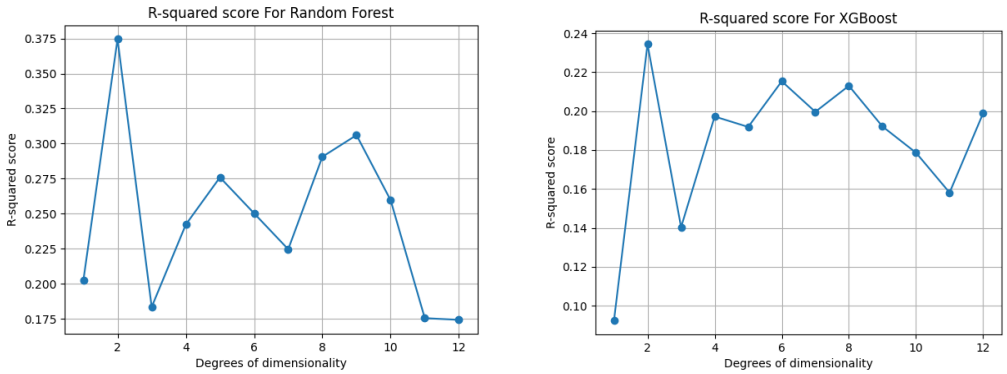

Figure 11: R-squared score at each feature level

As can be seen in the figures, both XGBoost and random forest performed similarly for varying degrees of dimensionality. This could be because both ensemble methods are based on trees and share similar underlying principles, which could explain why they perform similarly on some datasets. However, examining the curve reveals that random forest performs better at low dimensionality, and that as degree increases, as demonstrated by degrees 5 and 8, its MSE increases. Conversely, XGBoost also improves with degree, as evidenced by the curve, which indicates that higher degrees outperform lower ones as shown by the transition from degree 3 to 4 and by going from 5 to 6. This can also be noted in the MAE curve where random forest MAE increases when the degree increases as seen at degrees 7 to 12 and the same goes for XGBoost as it decreases by going from 4 to 8.

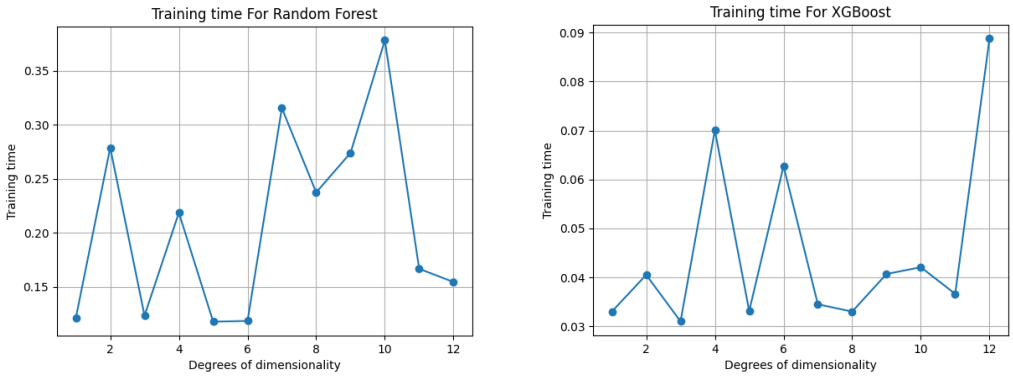

Figure 12: Training time at each feature level

It's clear that XGBoost is faster than Random Forest on the same conditions, hyper parameter tuning with the same dataset, due to XGBoost efficient handling of sparse data while Random Forest treats all features equally, potentially making it slower with sparse datasets. Also XGBoost builds trees sequentially focusing on correcting the errors of the previous tree which leads to faster convergence and fewer total trees needed compared to Random Forest's independent tree building.

3. Conclusion and recommendations

In the first scenario, which involved a large dataset, Random Forest demonstrated effective parallelization capabilities that scaled well to large datasets, but its training could be slow for very large datasets. In contrast, XGBoost, which is highly optimized for speed and performance, trained large datasets faster than Random Forest, but it needed more computation power. Hence, the suitability of any model for this situation depends on its computational capabilities, time, and interpretability. If computational resources are available and speed is crucial, then XGBoost is the best option, if resources are limited or interpretability is crucial, then random forest is the best option.

In the second scenario, which involved noisy data or features, Random Forest showed its robustness to noise due to averaging which can be seen in this evaluation metrics that shows that its performance is not that far off from XGBoost. But it also shows that it can still be influenced by outliers. While XGBoost showed its powerful regularization techniques which can handle noise effectively leading to outperforming random forest in noisy data situations. So in the noisy data scenario and if performance is crucial and computational resources are available then it is recommended to use XGBoost while it is recommended to use random forest if interpretability or robustness to noise.

In the last scenario, which involved varying degrees of dimensionality, random forest showed that it can perform well in both low and high dimensions but performance can decrease in very high dimensions while XGBoost improves its performance at high dimensions compared to the lower dimensions due to feature sparsity techniques but it may underperform in very low dimensions. So based on these it would be safe to say that XGBoost is preferred for high-dimensional datasets while random forest is better suited for low to moderate dimensionality.

4. References

[1] Leo Breiman: "Random Forests" Machine Learning.

Random Forests | Machine Learning (springer.com)

[2] Introduction to Random forest

https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest-simplified/

[3] What is random forest?

What is Random Forest? | IBM

[4] Random Forest Algorithm

Machine Learning Random Forest Algorithm - Javatpoint

[5] Random forest

Random forest - Wikipedia

[6] XGBoost: A Scalable Tree Boosting System

XGBoost | Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining

[7] A Gentle Introduction to XGBoost for Applied Machine Learning

A Gentle Introduction to XGBoost for Applied Machine Learning - MachineLearningMastery.com

[8] XGBoost Documentation

XGBoost Documentation — xgboost 2.0.3 documentation

[9] XGBoost

XGBoost - Wikipedia