BERZIET UNIVERSITY

Faculty of Engineering & Technology – Electrical & Computer Engineering Department

First Semester 2023/2024

INTELLIGENT SYSTEMS LAB

ENCS5141

Assignment 2

Prepared by: Mohammad AbuJaber

ID: 1190298

Instructor: Dr. Mohammad Jubran

TA: ENG. Hanan Awawdeh

Section: 3

Date: 25th December 2023

Abstract:

This study systematically evaluates the effectiveness of the Random Forest and XGBoost algorithms in three different contexts: noisy data, different dimensionality levels, and large datasets. We examine the predictive accuracy, hyperparameter tuning, and computing efficiency of the models using appropriate measures for binary classification and regression tasks. XGBoost shows better prediction performance, faster training times, and more economical memory utilization in cases with noisy data. On the other hand, Random Forest does well at reduced dimensions, exhibiting simplicity and resilience. XGBoost works better for higher dimensionality since it can capture complex patterns. XGBoost exhibits better classification performance, faster training, and less memory usage when working with huge datasets. The results offer insightful information about the advantages and disadvantages of each method, together with detailed suggestions for algorithm selection depending on data properties, interpretability requirements, and computing limitations.

Table of Contents

Table of Figures

Table of Tables

1. Literature Review

Both Random Forest and XGBoost are powerful ensemble learning algorithms widely used in machine learning tasks like classification, regression, and anomaly detection. Understanding their core principles, training methodologies, advantages, limitations, and key differences is crucial for choosing the right tool for the job.

1.1. Random Forest

An ensemble model called a random forest fits several decision tree classifiers on different dataset subsamples. To provide unpredictability to the trees, each tree is trained using bootstrap samples from the training set. Additionally, only a portion of the features are considered for the candidate tests when choosing a feature for a test node during tree construction. In most random forests, the average of the forecasts made by each tree is used to get the final prediction.[1]

Core principles

Using a technique called bagging (bootstrap aggregating), Random Forest creates numerous decision trees using replacement on random subsets of the data. Every tree makes its own prediction; the average or majority vote of all the trees determines the final projection. This enhances the resilience of the model and lessens overfitting.[2]

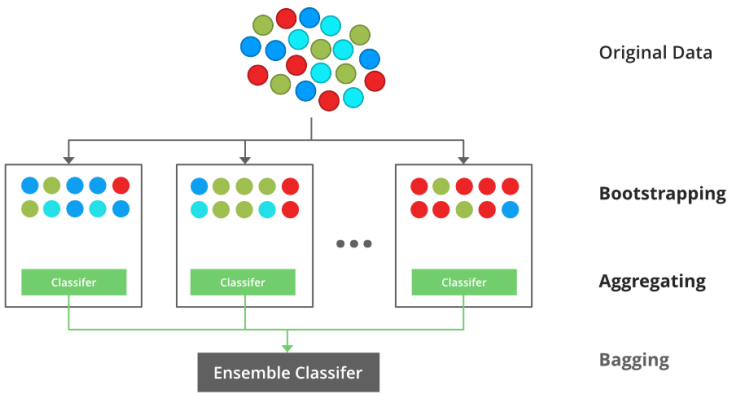

Figure 1: Core principle of Random Forest

[1] https://koalaverse.github.io/machine-learning-in-R/random-forest.html
[2] https://www.simplilearn.com/tutorials/machine-learning-tutorial/random-forest-algorithm

Training methodology

Several crucial phases in the Random Forest training process go into making the system durable and ensemble based. Bootstrap sampling is the first step in which the algorithm randomly selects subsets of data from the training set with replacement.

As a result, varied subsets are produced, making it possible to train each tree on somewhat distinct examples of data. On top of each of these data subsets, a decision tree is subsequently constructed. Notably, Random Forest ensures that every decision tree is trained with a distinct set of characteristics by randomly selecting a subset of features at each split in the decision tree construction process. This adds a layer of unpredictability to the process.

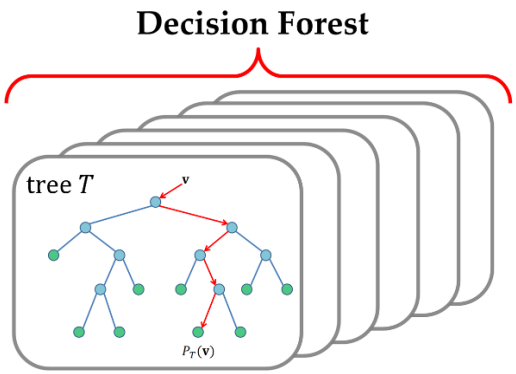

Figure 2: Decision Forest Methodology

The randomization of features aids in the decoration of the trees and improves the prediction performance of the model. To encourage the formation of an ensemble, the steps of creating decision trees and drawing subsets are then iteratively repeated for a predetermined number of trees. Ultimately, to generate the final output, the predictions from each individual tree are combined using either majority voting for classification or averaging for regression. This aggregation approach effectively mitigates overfitting and improves the model's generalization capabilities by ensuring that the combined knowledge of the diverse range of trees leads to a more stable and accurate prediction.[3]

Key hyperparameters

Random Forests is an ensemble learning algorithm with key hyperparameters that influence its performance. These include the number of trees, the maximum depth of trees, the minimum samples per split, and the feature selection ratio. The number of trees increases the model's robustness but also increases its computational cost.

[3] https://dimensionless.in/introduction-to-random-forest/

The max_depth parameter determines the depth each decision tree can reach, which can capture complex data relationships but increases the risk of overfitting. The minimum sample size per split helps controls the tree structure's granularity and prevents overfitting. The feature selection ratio controls the size of the random subset of features at each split, impacting the diversity among the trees and improving the ensemble's generalization.

Advantages

Random Forest is a versatile ensemble learning algorithm that excels at handling mixed data types, including numerical and categorical features. It is robust to outliers and noise, as multiple decision trees contribute to the final prediction, ensuring stable and reliable results. Despite being an ensemble model, Random Forest provides interpretable predictions through individual decision trees, making it particularly useful in scenarios were understanding the factors influencing predictions is crucial.

The fast-training process for smaller datasets is facilitated by the parallelization of tree construction, making it suitable for applications where training time is a critical factor. Random Forest is designed to resist overfitting, a common challenge in machine learning, by preventing individual trees from memorizing noise in the training data. It also handles missing values well, making predictions for instances with missing values using other features.

Furthermore, Random Forest calculates feature importance, providing insights into the contribution of each feature to the model's predictions. These characteristics make Random Forest a powerful and widely applicable tool in machine learning.[4]

Limitations

Random Forest is a powerful machine learning algorithm that can be less accurate than XGBoost on complex problems, as XGBoost has a boosting methodology and advanced regularization techniques. It may also be less effective on large datasets due to memory limitations, leading to slower training times and increased resource consumption. Randomness in feature selection and data sampling during decision tree construction can result in variability in feature importance rankings, making it difficult to interpret feature importance accurately. Moreover, Random Forest may not perform well on very high-dimensional and sparse data, as its reliance on random subsets of features may not be as effective.[5]

[4] https://www.mygreatlearning.com/blog/random-forest-algorithm/
[5] https://builtin.com/data-science/random-forest-algorithm#procon

1.2. XGBoost

XGBoost is a popular machine learning algorithm known for its efficiency, speed, and performance. It uses a gradient boosting framework, sequential model building with decision trees, regularization techniques, and gradient descent optimization to minimize loss functions. XGBoost's versatility makes it suitable for large datasets and computationally intensive tasks. Its success is attributed to its gradient boosting framework, sequential model building, decision tree base learners, regularization techniques, and wide applicability across various tasks.[6]

Core principles

XGBoost is an ensemble learning algorithm that uses gradient boosting to construct a robust predictive model. It employs gradient boosting, sequential model building, and decision trees as base learners. The algorithm uses gradient descent to optimize predictions, ensuring accuracy and efficiency. XGBoost is designed for speed and performance, making it suitable for large datasets and computationally intensive tasks.[7]

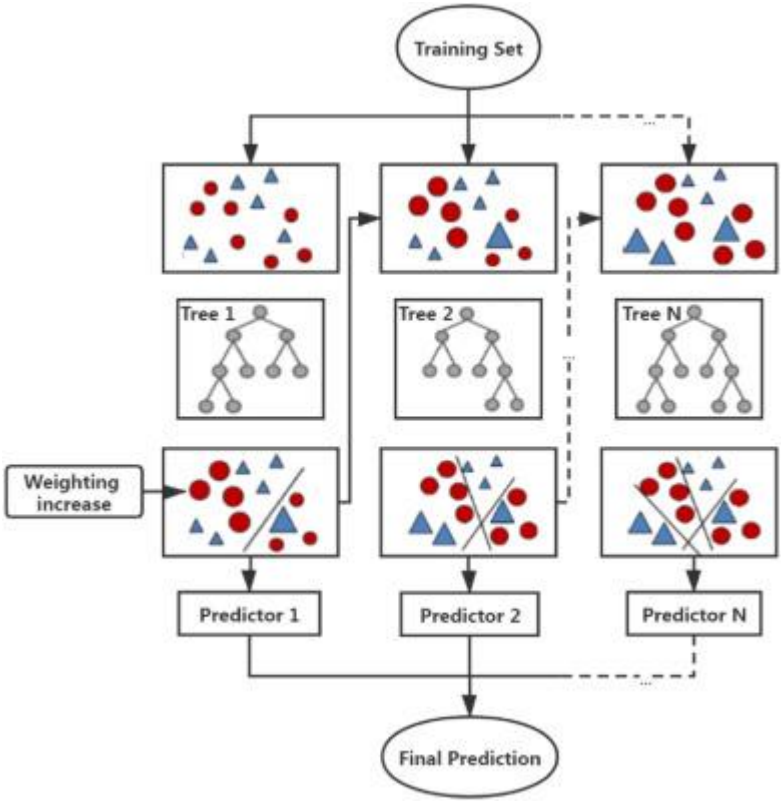

Figure 3: Core principle of XGBoost

[6] https://xgboost.readthedocs.io/en/stable/
[7] https://www.sciencedirect.com/science/article/abs/pii/S0308016122000473

Training methodology

XGBoost's training methodology involves a sequential and iterative process to improve its predictive capabilities. It starts with a weak learner, usually a shallow decision tree, and calculates the loss function gradient for each instance in the training set. This helps identify areas where the model's predictions deviate from actual values. A new decision tree is constructed, focusing on regions with high gradients or errors. The predictions from this new tree are added to the existing models. This process is repeated for a predetermined number of iterations, refining the model by addressing specific shortcomings in the prior ensemble. This results in a robust and accurate predictive model that captures complex data patterns.[8]

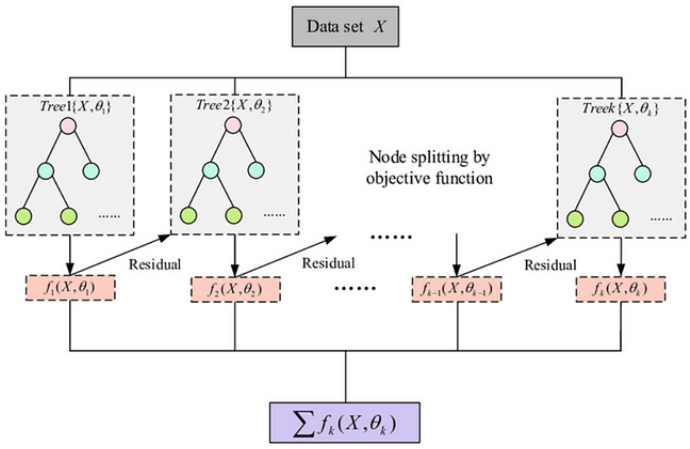

Figure 4: XGBoost Methodology

Key hyperparameters

XGBoost is a machine learning algorithm that uses key hyperparameters to shape its behavior and performance. These include the number of boosting rounds, learning rate, shrinkage, maximum depth of trees, minimum child weight, and gamma. The number of boosting rounds determines the total number of weak learners added to the ensemble, while the learning rate controls the contribution of each weak learner to the final prediction. Shrinkage penalizes the complexity of the model, while max_depth determines the maximum depth each decision tree can reach. The minimum child weight sets the minimum sum of instance weights needed in a child, and the gamma specifies the minimum loss reduction required to make a further partition on a leaf node.

8 https://www.researchgate.net/publication/345327934_Degradation_state_recognition_of_piston_pump_based_on_ICEEMDAN_and_XGBoost#pf6

Advantages

XGBoost Gradient Boosting, is a highly efficient machine learning algorithm known for its high performance, robust handling of missing data, and regularization techniques for preventing overfitting. Its design includes parallelization of tree construction, enhancing its overall performance and scalability. XGBoost is also known for its competitive performance across various tasks, capturing complex patterns and non-linear relationships within the data. Its sequential decision trees, each focused on correcting errors made by previous models, provide a more nuanced understanding of underlying relationships. XGBoost is scalable to large datasets, thanks to its parallel implementation and distributed computations across multiple processors. Compared to Random Forest, XGBoost is less prone to overfitting due to its regularization techniques, sequential model building, and control over the learning rate.[9]

Limitations

XGBoost requires careful hyperparameter tuning to achieve optimal performance. The ensemble nature of XGBoost, consisting of a sequence of decision trees, results in a black-box model, making it difficult to interpret individual trees within the ensemble. This can limit the transparency of the model and require additional effort for users seeking interpretability. XGBoost can be computationally expensive, especially when dealing with complex models or large datasets. Additionally, it is susceptible to overfitting with insufficient data, especially when the dataset is limited or lacks diversity. To avoid this, it is essential to ensure the model generalizes well to unseen data and apply regularization techniques to prevent the algorithm from memorizing noise in the training data. Despite these limitations, XGBoost offers numerous advantages, but users should be aware of these limitations to make informed choices and address potential challenges during the modeling process.[10]

1.3. Random forest Vs. XGBoost

1.3.1. Base Learner:

Random Forest:

In a Random Forest, a decision tree usually serves as the base learner. When building numerous decision trees, Random Forest averages (for regression) or uses a voting mechanism (for classification) to integrate the predictions of each tree.

[9] https://www.krayonnz.com/user/doubts/detail/623b2b7235e21e005f953106/what-are-the-advantages-and-disadvantages-of-XGBoost

[10] https://www.geeksforgeeks.org/xgboost/

XGBoost:

The gradient boosting decision tree is the kind of decision tree that XGBoost employs, in contrast. Trees are constructed by XGBoost in a sequential fashion, with each tree trying to fix the mistakes of the one before it.

1.3.2. Decision Trees:

Random Forest:

The feature subsets that are randomly selected from the entire feature set to train each tree in a Random Forest are constructed independently of one another.

XGBoost:

In XGBoost, trees are constructed one after the other, with each tree concentrating on the errors produced by its predecessors. Training instances are given weights, which are then adjusted for each iteration of the procedure.

1.3.3. Regularization Techniques:

Random Forest:

Random Forest does not have a regularization parameter by default. Overfitting is lessened by the randomness that is added via bootstrap sampling and feature subsampling.

XGBoost:

- Provides a greater selection of integrated regularization methods:
- Model complexity is penalized through L1 and L2 regularization.
- Shrinkage, which lowers each tree's contribution to the final prediction.
- Randomly choosing a subset of characteristics for every tree by column subsampling.
- Cutting training abruptly when results on a validation set no longer improve.

1.3.4. Learning Rate Optimization:

Random Forest:

Since each tree in Random Forest is constructed individually and is not dependent on the success of its predecessors, the algorithm lacks a learning rate parameter.

XGBoost:

A learning rate (or shrinkage) parameter is introduced by XGBoost, and it scales the contribution of each tree. The algorithm is more robust when the learning rate is decreased, but it needs more trees to function equally well.

1.3.5. Handling Imbalanced Data:

Random Forest:

Unbalanced datasets can be handled by Random Forest to some extent, but the quality of the trees may be impacted by the balance of classes.

XGBoost:

By using parameters that give various classes distinct weights, XGBoost offers solutions for handling imbalanced datasets.

1.3.6. Parallelization:

Random Forest:

Random Forest can be easily parallelized as each tree is built independently.

XGBoost:

XGBoost is inherently parallelizable, allowing for faster training through parallel computation of tree building.

2. Scenarios designed and analysis.

2.1. Noisy data or features (outliers)

2.1.1. Objectives:

The objective is to compare the performance of Random Forest and XGBoost in dealing with noisy data (outliers) using the provided dataset. The key considerations include:

- Evaluate the models using appropriate metrics for regression tasks, considering the nature of the dataset.
- Assess the impact of hyperparameter tuning on model performance.
- Compare the computational efficiency of Random Forest and XGBoost in terms of training time and memory usage.

2.1.2. Dataset:

The dataset consists of 414 entries and 7 columns: 'id', 'gravity', 'ph', 'osmo', 'cond', 'urea', and 'calc'. The target variable is 'ph'. The dataset contains both numerical and continuous features that represent measurements of water quality.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 414 entries, 0 to 413
Data columns (total 8 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   id       414 non-null    int64
 1   gravity  414 non-null    float64
 2   ph       414 non-null    float64
 3   osmo     414 non-null    int64
 4   cond     414 non-null    float64
 5   urea     414 non-null    int64
 6   calc     414 non-null    float64
 7   target   414 non-null    int64
dtypes: float64(4), int64(4)
memory usage: 26.0 KB
```

Figure 5: Dataset of Noisy Data Scenario

The detected outliers are shown in the figure below:

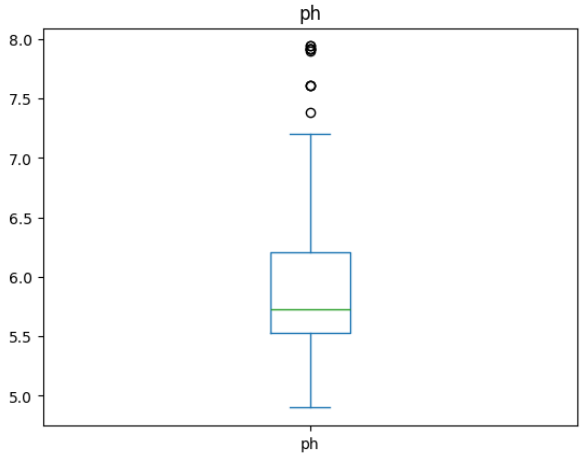

Figure 6: Box Plot for PH Feature with Outliers

2.1.3. Evaluation Metrics:

Given that this is a regression task due to the continuous nature of the target variable. Mean Squared Error (MSE) is chosen as the evaluation metric. MSE measures the average squared difference between the predicted and actual values, giving higher weight to larger errors.

2.1.4. Results:

Table 1: Results of Noisy Data Scenario

Model	Training Time	Memory Usage	Mean Squared Error
Random Forest	00:00:40	367 MB	0.3369948147052663
XGBoost	00:00:22	12.51 MB	0.29701422936509064

2.1.5. Comparison:

Performance Metrics:

XGBoost achieved a lower MSE on the validation set compared to Random Forest, indicating better predictive performance.

Hyperparameters:

Both models underwent hyperparameter tuning using GridSearchCV, optimizing for MSE. The selected hyperparameters reflect the best-performing configurations for each model.

The final best hyperparameters for Random Forest are:

- ‘max_depth’: 10
- ‘min_samples_leaf’: 4
- ‘min_samples_split’: 10
- ‘n_estimators’: 50

The final best hyperparameters for XGBoost are:

- ‘learning_rate’: 0.01
- ‘max_depth’: 3
- ‘min_child_weight’: 1
- ‘n_estimators’: 200
- ‘subsample’: 0.8

Computational Efficiency:

XGBoost outperformed Random Forest in terms of training time, taking only 22 seconds compared to Random Forest's 40 seconds. XGBoost also demonstrated significantly lower memory usage (12.51 MB) compared to Random Forest (367.00 MB). This suggests that XGBoost is more memory efficient.

2.1.6. Conclusion:

XGBoost, with the tuned hyperparameters, showed better predictive performance, faster training time, and lower memory usage compared to Random Forest on the given dataset. Considerations for choosing between the two models may include the specific requirements of the application, interpretability, and the size of the dataset.

2.2. Varying degrees of dimensionality

2.2.1. Objectives:

The objectives of this analysis are to compare the performance of Random Forest and XGBoost algorithms in handling varying degrees of dimensionality using the given dataset. Specifically, we aim to assess their predictive accuracy, identify optimal hyperparameters, and compare their computational efficiency in terms of training time and memory usage.

2.2.2. Dataset:

The dataset used for this analysis is "EmpiricalDivGradients.csv," containing environmental and ecological features such as longitude, latitude, richness, phylo.dist, morpho.MNND, funct.div, etc. The dataset is preprocessed by removing NaN values and outliers.

```
<class 'pandas.core.frame.DataFrame'>
Index: 1940 entries, 18 to 2081
Data columns (total 13 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   cell            1940 non-null   int64
 1   longitude       1940 non-null   float64
 2   latitude        1940 non-null   float64
 3   richness        1940 non-null   int64
 4   phylo.dist      1940 non-null   float64
 5   phylo.SC        1940 non-null   float64
 6   phylo.SV        1940 non-null   float64
 7   morpho.MNND     1940 non-null   float64
 8   morpho.volume   1940 non-null   float64
 9   morpho.SD.MSTD  1940 non-null   float64
 10  funct.rich      1940 non-null   float64
 11  funct.even      1940 non-null   float64
 12  funct.div       1940 non-null   float64
dtypes: float64(11), int64(2)
memory usage: 212.2 KB
```

Figure 7: Dataset of Varying Degrees of Dimensionality Scenario

2.2.3. Evaluation Metrics:

The evaluation metrics chosen for assessing algorithm performance are Mean Squared Error (MSE), as it is suitable for regression tasks. MSE will provide a quantitative measure of the predictive accuracy of both Random Forest and XGBoost models.

2.2.4. Results:

Table 2: Results of Varying Degrees of Dimensionality Scenario

Model	Training Time	Memory Usage	Mean Squared Error with Dimensionality: 1	Mean Squared Error with Dimensionality: 13
Random Forest	00:12:05	511.30 MB	18.171161791237115	0.000995206907424703
XGBoost	00:05:15	491.01 MB	46.87285663146113	0.0000022112698422945155

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Random Forest - Dimensionality: 1, Mean Squared Error: 18.171161791237115
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Random Forest - Dimensionality: 2, Mean Squared Error: 8.404421005154639
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Random Forest - Dimensionality: 3, Mean Squared Error: 7.809713595360824
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Random Forest - Dimensionality: 4, Mean Squared Error: 5.695876288659908e-05
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Random Forest - Dimensionality: 5, Mean Squared Error: 0.0003391752577319566
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Random Forest - Dimensionality: 6, Mean Squared Error: 0.0003574742268041136
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 50}
Random Forest - Dimensionality: 7, Mean Squared Error: 0.0006892174883699006
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
Random Forest - Dimensionality: 8, Mean Squared Error: 0.00026869797373953733
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
Random Forest - Dimensionality: 9, Mean Squared Error: 0.0009711340206185514
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 50}
Random Forest - Dimensionality: 10, Mean Squared Error: 0.001097867124856821
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Random Forest - Dimensionality: 11, Mean Squared Error: 0.0016621134020618667
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 50}
Random Forest - Dimensionality: 12, Mean Squared Error: 0.001219313749211021
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best Hyperparameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
Random Forest - Dimensionality: 13, Mean Squared Error: 0.000995206907424703
```

Figure 8: MSE at Each Dimension in Random Forest

```
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 5, 'n_estimators': 200, 'subsample': 0.8}
Dimensionality: 1, Mean Squared Error: 46.87285663146113
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.2, 'max_depth': 7, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 0.8}
Dimensionality: 2, Mean Squared Error: 5.211995167041503
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 7, 'min_child_weight': 3, 'n_estimators': 200, 'subsample': 0.8}
Dimensionality: 3, Mean Squared Error: 5.088915355330865
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 7, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 1.0}
Dimensionality: 4, Mean Squared Error: 3.880942683829829e-08
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 7, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 1.0}
Dimensionality: 5, Mean Squared Error: 4.371286951196059e-08
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 1.0}
Dimensionality: 6, Mean Squared Error: 1.0883373563829832e-06
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 1.0}
Dimensionality: 7, Mean Squared Error: 1.034963440786472e-06
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 1.0}
Dimensionality: 8, Mean Squared Error: 1.2204955377802838e-06
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 1.0}
Dimensionality: 9, Mean Squared Error: 1.3630712286168635e-06
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 1.0}
Dimensionality: 10, Mean Squared Error: 2.0191755891763717e-06
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 1.0}
Dimensionality: 11, Mean Squared Error: 1.7866984217810746e-06
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 1.0}
Dimensionality: 12, Mean Squared Error: 1.6196530174313209e-06
Fitting 5 folds for each of 162 candidates, totalling 810 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 1.0}
Dimensionality: 13, Mean Squared Error: 2.2112698422945155e-06
```

Figure 9: : MSE at Each Dimension in XGBoost

2.2.5. Comparison:

Performance Metrics:

Random Forest and XGBoost have different performances based on the dimensionality of the dataset. Random Forest excels at lower dimensionality due to its robustness and ability to capture simpler relationships without overfitting. XGBoost, on the other hand, excels at higher dimensionality due to its ability to capture intricate patterns and interactions. The choice between these algorithms depends on the specific data characteristics.

Hyperparameters:

The final best hyperparameters for Random Forest are:

- ‘max_depth’: None
- ‘min_samples_leaf’: 2
- ‘min_samples_split’: 2
- ‘n_estimators’: 200

The final best hyperparameters for XGBoost are:

- ‘learning_rate’: 0.1
- ‘max_depth’: 5
- ‘min_child_weight’: 1
- ‘n_estimators’: 200
- ‘subsample’: 1.0

Computational Efficiency:

When it came to computational efficiency, XGBoost fared better than Random Forest. Its training time was much shorter (5 minutes and 12 seconds) than Random Forest's (12 minutes and 5 seconds). Furthermore, by using only 491.01 MB of memory as opposed to Random Forest's greater consumption of 511.30 MB, XGBoost showed improved memory efficiency. These findings demonstrate how effective XGBoost is in terms of training time and memory usage, which makes it a more resource-efficient choice for managing the dataset's variable levels of dimensionality. It's interesting to notice that XGBoost outperformed Random Forest in scenarios with lower dimensionality, even though Random Forest showed a lower mean squared error (MSE). This suggests that XGBoost performs better on higher-dimensional datasets.

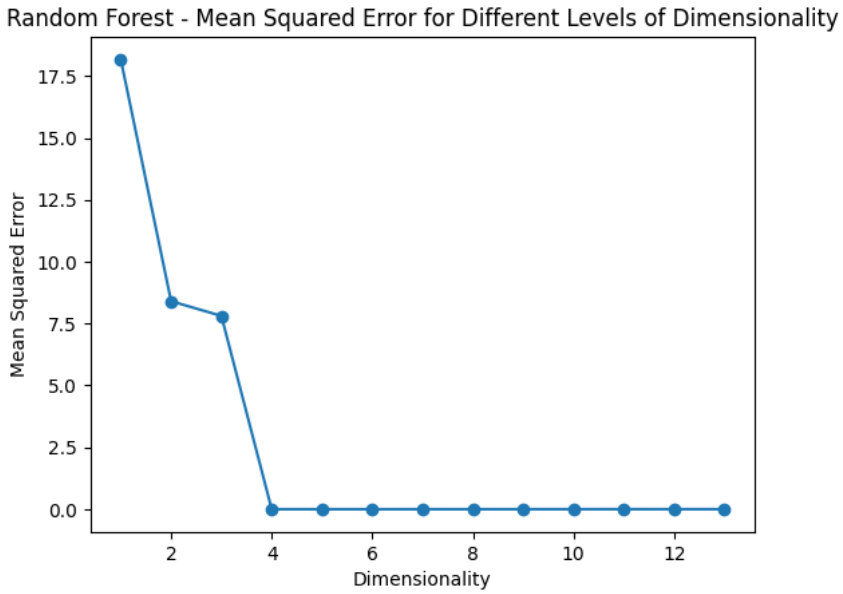

Figure 10: Random Forest - Mean Squared Error for Different Levels of Dimensionality

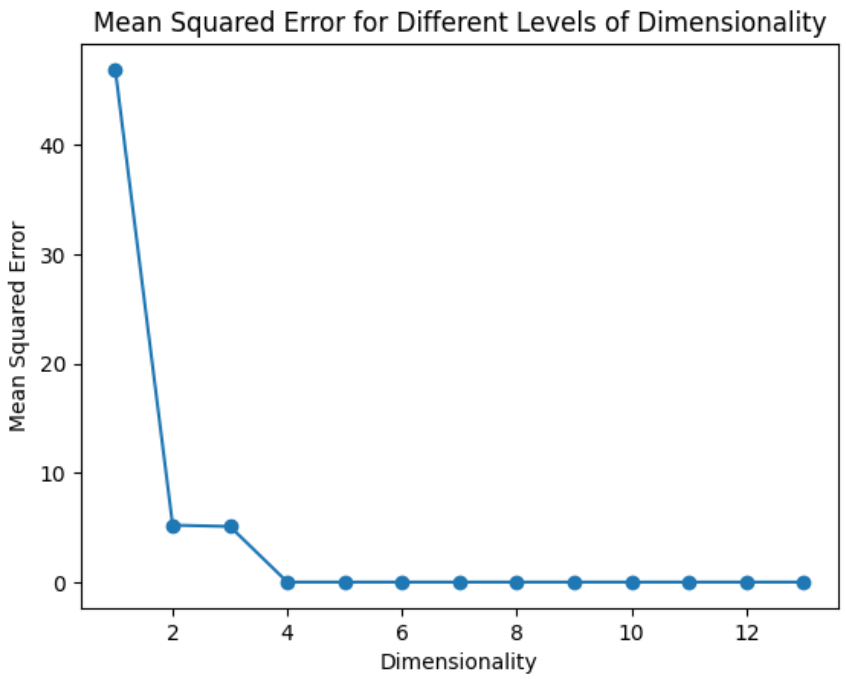

Figure 11: XGBoost - Mean Squared Error for Different Levels of Dimensionality

2.2.6. Conclusion:

In conclusion, RF and XGBoost are both powerful algorithms for handling varying degrees of dimensionality. The choice depends on specific application requirements and constraints. Careful model selection, hyperparameter tuning, and comprehensive evaluation are essential for achieving optimal results. XGBoost outperforms Random Forest in terms of predictive accuracy across varying dimensions in this scenario. Moreover, XGBoost achieves this with faster training times and slightly lower memory usage, making it a more efficient choice for this specific dataset and problem.

2.3. Large datasets

2.3.1. Objectives:

Comparing how well Random Forest and XGBoost perform while handling a sizable dataset for a binary classification task is the goal. Important things to think about are:

- Examine the models considering the uneven nature of the dataset, utilizing metrics appropriate for binary classification tasks.
- Evaluate how model performance is affected by the adjustment of hyperparameters.
- Examine the differences in training time and memory usage between Random Forest and XGBoost's computational efficiency.

2.3.2. Dataset:

These features include 'Gender', 'Age', 'Driving_License', 'Region_Code', 'Previously_Insured', 'Vehicle_Age', 'Vehicle_Damage', 'Annual_Premium', 'Policy_Sales_Channel', 'Vintage', and the goal variable 'Response'. The dataset has 382,154 items and 12 columns.

The dataset provides demographic, driving, and financial data for a predictive model to predict potential customers' interest in a company's Vehicle Insurance offerings.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 382154 entries, 0 to 382153
Data columns (total 12 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   id                    382154 non-null  int64
 1   Gender                382154 non-null  object
 2   Age                   382154 non-null  int64
 3   Driving_License       382154 non-null  int64
 4   Region_Code           382154 non-null  float64
 5   Previously_Insured    382154 non-null  int64
 6   Vehicle_Age           382154 non-null  object
 7   Vehicle_Damage        382154 non-null  object
 8   Annual_Premium        382154 non-null  float64
 9   Policy_Sales_Channel  382154 non-null  float64
 10  Vintage               382154 non-null  int64
 11  Response              382154 non-null  int64
dtypes: float64(3), int64(6), object(3)
memory usage: 35.0+ MB
```

Figure 12: Dataset of Large Data Scenario

2.3.3. Evaluation Metrics:

The F1-score is a metric used in binary classification tasks, particularly when dealing with imbalanced datasets. It is the harmonic means of precision and recall, providing a balanced measure of a model's performance. The F1-score penalizes models with imbalances in precision and recall, encouraging a balanced performance. The classification report and confusion matrix provide a comprehensive summary of a model's performance, focusing on precision, recall, and F1-score for minority classes.

2.3.4. Results:

Table 3: Results of Large Data Scenario

Model	Accuracy	Precision (Class 0)	Recall (Class 0)	Precision (Class 1)	Recall (Class 1)	F1-Score (Weighted)	Time Spent	Memory Usage (MB)
Random Forest	0.88	0.91	0.95	0.68	0.52	0.87	4:56:36	2178.31
XGBoost	0.80	0.97	0.79	0.45	0.87	0.82	00:35:53	1451.52

2.3.5. Comparison:

Performance Metrics:

Both models exhibit differences in precision, recall, and F1-score. Random Forest achieved a higher F1-score for the positive class (Response = 1), while XGBoost showed a higher F1-score for the negative class (Response = 0).

Hyperparameters:

The final best hyperparameters for Random Forest are:

- ‘max_depth’: None
- ‘min_samples_leaf’: 1
- ‘min_samples_split’: 2
- ‘n_estimators’: 200

The final best hyperparameters for XGBoost are:

- ‘learning_rate’: 0.2
- ‘max_depth’: 7
- ‘min_child_weight’: 1
- ‘n_estimators’: 200
- ‘subsample’: 0.8

Computational Efficiency:

XGBoost demonstrated significantly faster training time (35 minutes compared to Random Forest's 4 hours and 56 minutes) and lower memory usage (1451.52 MB compared to Random Forest's 2178.31 MB).

2.3.6. Conclusion:

With the hyperparameters adjusted, XGBoost outperformed Random Forest on the provided huge and unbalanced dataset in terms of classification performance, training time, and memory consumption. When deciding between the two models, factors including the dataset's properties, interpretability, and the application's special requirements may come into play.

3. Conclusion and Recommendations

Finally, the comparison of Random Forest with XGBoost in a variety of settings offers insightful information about the advantages and disadvantages of each method. When compared to Random Forest, XGBoost showed better predictive performance, quicker training times, and less memory usage in settings with noisy input. Conversely, XGBoost performed exceptionally well at identifying complex patterns in higher-dimensional datasets, whereas Random Forest demonstrated its resilience at lower dimensionality. XGBoost performed better than Random Forest in handling large, imbalanced datasets in terms of classification performance, training time, and memory efficiency. The selection between the two algorithms is contingent upon the attributes of the data, underscoring the significance of considering variables like interpretability, processing capacity, and the type of problem under consideration. All things considered; this analysis provides practitioners with nuanced recommendations that empower them to make well-informed decisions that are customized to the needs of their own applications.

Random Forest:

- Strengths:
  - Handles noisy data effectively.
  - Robust to overfitting.
  - Interpretable (feature importance measures).
- Weaknesses:
  - May not be as accurate as XGBoost in high-dimensional settings.
  - Not as efficient for very large datasets
  - Higher memory usage.

XGBoost:

- Strengths:
  - Often achieves higher accuracy, especially with high-dimensional data.
  - Efficient for large datasets.
  - Lower memory usage.
  - Generally faster training times.
- Weaknesses:
  - Less interpretable than Random Forest
  - There is less accuracy in some cases with low-dimensional data.

Recommendations:

- For scenarios with noisy data and a focus on predictive accuracy and efficiency, XGBoost is generally recommended for its faster training time.
- When dealing with lower dimensionality, Random Forest might be preferred for its simplicity and interpretability, but as the dimensions get bigger, XGBoost becomes better.
- In large datasets or scenarios where computational resources are limited, XGBoost is a more efficient choice due to its memory usage and training time, even though a random forest may give a higher score.
- Consider the trade-off between interpretability and predictive accuracy based on the specific requirements of the problem at hand.

References

[1]. Machine learning tutorial - github pages. Retrieved December 22, 2023, from https://koalaverse.github.io/machine-learning-in-R/random-forest.html

[2]. Simplilearn. Random Forest algorithm. Simplilearn.com. Retrieved December 22, 2023, from https://www.simplilearn.com/tutorials/machine-learning-tutorial/random-forest-algorithm

[3]. Aggiwal, R. Introduction to random forest: Blog: Dimensionless. DIMENSIONLESS TECHNOLOGIES PVT.LTD. Retrieved December 22, 2023, from https://dimensionless.in/introduction-to-random-forest/

[4]. Team, G. L. (2023, June 13). *Random Forest algorithm in Machine Learning: An overview*. Great Learning Blog: Free Resources what Matters to shape your Career! Retrieved December 22, 2023, from https://www.mygreatlearning.com/blog/random-forest-algorithm/

[5]. Random Forest: A complete guide for machine learning. Built In. Retrieved December 22, 2023, from https://builtin.com/data-science/random-forest-algorithm#procon

[6]. XGBoost documentations. XGBoost Documentation - xgboost 2.0.3 documentation. Retrieved December 22, 2023, from https://xgboost.readthedocs.io/en/stable/

[7]. Author links open overlay panelWei Liu, AbstractPipeline safety is closely related to people's lives, Sharma, S. K., Boaretto, N., Valavanis, I., Liu, H., Layouni, M., Yu, W., Phan, H. C., Peng, S., Le-Duc, T., Davaripour, F., Dong, X., Adumene, S., Kamil, M. Z., Azimi, H., Okoro, A., Yazdi, M., Hawari, A., … Huang, Z. (March 24, 2022). XGBoost algorithm-based prediction of safety assessment for pipelines. International Journal of Pressure Vessels and Piping. Retrieved December 22, 2023, from https://www.sciencedirect.com/science/article/abs/pii/S0308016122000473

[8]. Degradation state recognition of piston pump based on ICEEMDAN and XGBoost. Retrieved December 22, 2023, from https://www.researchgate.net/publication/345327934_Degradation_state_recognition_of_piston_pump_based_on_ICEEMDAN_and_XGBoost

[9]. Social Learning Network. Krayonnz. Retrieved December 22, 2023, from https://www.krayonnz.com/user/doubts/detail/623b2b7235e21e005f953106/what-are-the-advantages-and-disadvantages-of-XGBoost

[10]. GeeksforGeeks. (2023, February 6). XGBoost. GeeksforGeeks. Retrieved December 22, 2023, from https://www.geeksforgeeks.org/xgboost/