FACULTY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING

INTELLIGENT SYSTEMS LAB
ENCS5141

Title: Comparative study between Random Forest and XGBoost
Assignment #2

Prepared by: Ali Mohammed
ID: 1190502

Instructor: Dr. Yazan Abu Farha
Teacher Assistant: Eng. Hanan Awawdeh

Section: 1
BIRZEIT
December 2023

Abstract

This assignment aims to study two models, Random Forest and XGBoost, by comparing them when handling different types of datasets and using precise evaluation metrics along with studying their computational efficiency in terms of training time and memory usage.

Table of Contents

List of Figures

1 Literature Review

Random Forest and XGBoost are both popular machine learning algorithms used for classification and regression tasks. They belong to the ensemble learning family but employ different techniques - Random Forest utilizes bagging while XGBoost utilizes boosting.

1.1 Random forest

Random Forest is a machine learning algorithm that can be used for both classification and regression problems. It is an ensemble method that combines multiple independently trained decision trees and then undertakes majority voting on the output of all these trees to produce a final prediction.

1.1.1 Training mythology

Random Forest uses a method called bagging (bootstrap aggregation). It creates multiple subsets of the original dataset, with replacement, and then trains a separate decision tree on each subset. The final prediction is the average of the predictions from all the trees.

1.1.2 Main hyperparameters

- **Number of Trees** (**n_estimators**): This is the number of trees you want to build before taking the maximum voting or averages of predictions. Higher number of trees gives you better performance but makes your code slower.
- **Maximum Depth of Each Tree** (**max_depth**): This is the maximum depth of each tree. The deeper the tree, the more splits it has and it captures more information about the data.
- **Minimum Samples Split** (**min_samples_split**): This is the minimum number of samples required to split an internal node. It can be an integer or a float representing the fraction of the total number of samples.
- **Minimum Samples Leaf** (**min_samples_leaf**): The minimum number of samples required to be at a leaf node. It can be an integer or a float representing the fraction of the total number of samples.
- **Maximum Features** (**max_features**): The number of features to consider when looking for the best split. It can be an integer, float, or a string from {“sqrt”, “log2”, None}.
- **Bootstrap Method**: Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

1.1.3 Advantages

- **Robustness**: Random Forest is a robust algorithm that can handle noisy data and outliers. It is less likely to overfit the data, which means it can generalize well to new data.
- **Accuracy**: Random Forest is one of the most accurate machine learning algorithms. It can handle both classification and regression problems and can work well with both categorical and continuous variables.
- **Speed**: Despite being a complex algorithm, Random Forest is fast and can handle large datasets. It can also be easily parallelized to speed up training.
- **Feature Importance**: Random Forest provides a measure of feature importance, which can help in feature selection and data understanding.

1.1.4 Limitations

- **Overfitting**: Although Random Forest is less prone to overfitting than a single decision tree, it can still overfit the data if the number of trees in the forest is too high or if the trees are too deep.
- **Interpretability**: Random Forest can be less interpretable than a single decision tree because it involves multiple trees. It can be difficult to understand how the algorithm arrived at a particular prediction.
- **Training Time**: The training time of Random Forest can be longer than other algorithms. Especially if the number of trees and the depth of the trees are high.
- **Memory Usage**: Random Forest requires more memory than other algorithms because it stores multiple trees. This can be a problem if the dataset is large.

1.2 XGBoost

XGBoost stands for Extreme Gradient Boosting. It is a machine learning algorithm that can also be used for both classification and regression problems. It is an ensemble model, but it differs by training many decision trees sequentially. Training steps:

1. Draw random samples with replacement from the training data (bootstrapping).
2. For each sample, build a decision tree:
   - At each node, randomly select a subset of features to consider for splitting.
   - Split the node using the best feature among the selected subset.
3. Repeat steps 1-2 to create multiple trees.
4. Aggregate predictions from all trees to get the final prediction.

1.2.1 Training mythology

In XGBoost, each decision tree is shallow and adjusted with the error from the previous tree, resulting in many weak classifiers which when combined create a highly performant mode. The training steps:

1. Initialize a model with a single tree.
2. For each subsequent tree:
   - Calculate the residuals (errors) of the current model.
   - Fit a new tree to those residuals.
   - Add the new tree to the model, with a learning rate that controls its contribution.
3. Repeat step 2 until a stopping criterion is met.

1.2.2 Main hyperparameters

- **Number of Trees (n_estimators)**: This is the number of gradient boosted trees. Equivalent to number of boosting rounds.
- Learning Rate (eta): This makes the model more robust by shrinking the weights on each step.
- **Maximum Depth of Each Tree (max_depth):** The maximum depth of a tree, same as GBM. Used to control over-fitting as higher depth will allow the model to learn relations very specific to a particular sample.
- **Minimum Child Weight (min_child_weight):** Defines the minimum sum of weights of all observations required in a child. Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.

- **Subsample (subsample):** Denotes the fraction of observations to be randomly sampled for each tree. Lower values make the algorithm more conservative and prevent overfitting but too small values might lead to under-fitting.
- **Gamma (gamma):** A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.
- **Regularization Parameters (alpha, lambda):** Can be used in case of very high dimensionality so that the algorithm runs faster when implemented.

1.2.3 Advantages

- **Excellent Predictive Powers**: XGBoost is precise, adapts well to all types of data and problems, and is known for its high accuracy.
- **Efficiency**: XGBoost is the fastest gradient-boosting library for R, Python, and C++.
- **Ease of Use**: XGBoost has excellent documentation, making it very easy to use.
- **Works Well on Limited Datasets**: XGBoost performs very well on medium, small, data with subgroups and structured datasets with not too many features.
- **Resilient and Robust**: XGBoost is a resilient and robust method that prevents and curbs overfitting quite easily.

1.2.4 Limitations

- **Handling Categorical Features**: XGBoost cannot handle categorical features while other similar packages like LightGBM and CatBoost can.
- **Memory Usage**: XGBoost can be more memory-hungry than LightGBM3.
- **Speed**: XGBoost can be slower than LightGBM.
- **Overfitting**: XGBoost, like other tree algorithms, can over-fit the data, especially if the trees are too deep with noisy data.

1.3 Key differences

- **Ensemble Technique**: Random Forest uses bagging (parallel ensemble), while XGBoost uses boosting (sequential ensemble).
- **Speed**: Random Forest can be faster to train compared to XGBoost, especially on large datasets.
- **Training Methodology**: The decision trees in XGBoost are trained sequentially with adjustments made from the prior tree's error, while in Random Forest they are created in parallel and independently
- **Overfitting**: Random Forest models are prone to overfitting, while XGBoost models counter this by creating shallower trees.
- **Parameter Tuning**: XGBoost has many more parameters that can be optimized through parameter tuning, compared to Random Forest.
- **Handling of Datasets**: XGBoost often works better with imbalanced datasets
- **Decision Trees:** Random Forest uses fully-grown trees, while XGBoost uses trees with regularized depth.
- **Regularization**: Random Forest has limited regularization, while XGBoost has built-in regularization techniques.
- **Learning Rate**: XGBoost has a learning rate to control overfitting, while Random Forest does not.

2 Scenarios designed and analysis

For each scenario the objectives are described, and the selected dataset. Appropriate evaluation metrics are used based on each scenario. The computational efficiency of Random forest and XGBoost are compared in terms of training time and memory usage.

2.1 Imbalanced classes: Breast Cancer Wisconsin (Diagnostic) Dataset

2.1.1 Scenario and objective

- **Scenario**: Predicting whether a breast mass is malignant or benign. In this dataset, the majority of masses are benign, so the malignant cases represent a minority class.
- **Objective**: The goal is to accurately identify the rare instances of malignancy while minimizing the false positives.

2.1.2 Dataset

The Breast Cancer Wisconsin (Diagnostic) Data Set is a popular dataset used in machine learning and data science. The dataset is used to predict whether the cancer is benign or malignant It was donated to the UCI Machine Learning Repository on October 31, 1995. The features describe characteristics of the cell nuclei present in the image. It contains 569 instances with 30 features

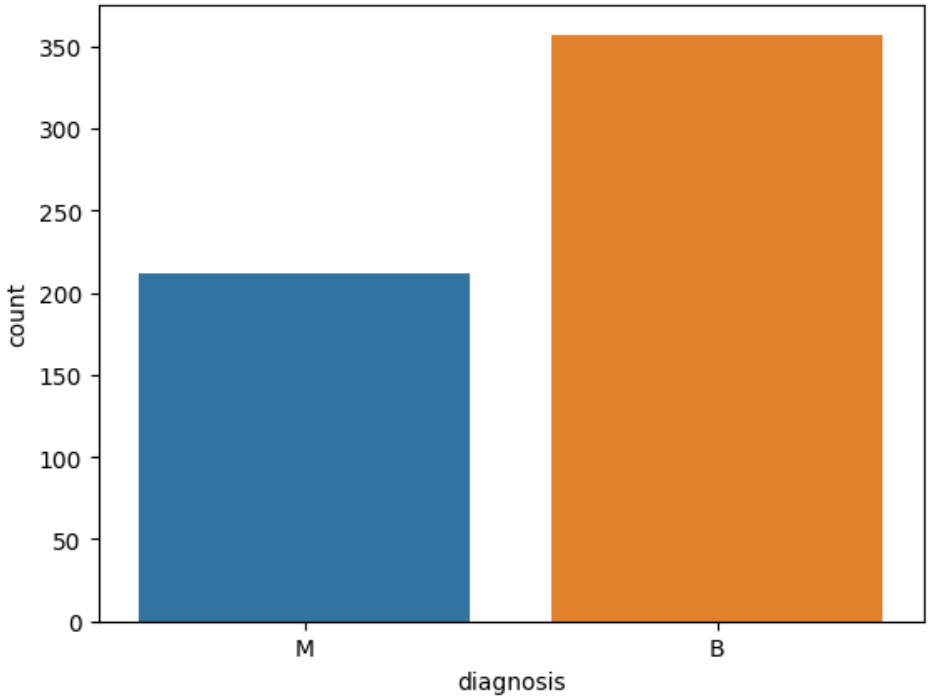

Figure 1: Distribution of target feature (diagnosis)

In the notebook file, the labels are (0 for M) and (1 for B).
The proportion of each class:

- Class B: 357 (62.74165%)
- Class M: 212 (37.25835%)

In general, a dataset is considered imbalanced if it follows the table:

Degree of imbalance	**Proportion of Minority Class**
Mild	20-40% of the dataset
Moderate	1-20% of the dataset
Extreme	<1% of the dataset.

So, the chosen dataset is considered imbalanced with mild degree.

2.1.3 Hyperparamter tuning

2.1.3.1 Random forest

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_features': ['sqrt', 'log2', None],
    'max_depth': [5, 9, None],
    'max_leaf_nodes': [3, 7, None],
}

```

2.1.3.2 XGBoost

```
params = {
    "colsample_bytree": [0.3, None],
    "gamma": [0.5, None],
    "learning_rate": [0.5, None],
    "max_depth": [5, None],
    "n_estimators": [25, 50, 100, None],
    "subsample": [0.5, None]
}

```

2.1.4 Evaluation metric

For imbalanced datasets, accuracy is not a good metric as it can be misleading. Instead, **F1 score** metric takes into account the performance for each class.

2.1.5 Results

Metric	Random forest		XGBoost	
	Before tuning	After tuning	Before tuning	After tuning
F1 score	0.9721	0.9775	0.9775	0.9831

2.1.6 Computational efficiency

Computational efficiency	Random forest		XGBoost	
	Before tuning	After tuning	Before tuning	After tuning
Training time	0.321 seconds	0.512 seconds	0.430 seconds	0.628 seconds
Memory usage	113.325 MB	145.380 MB	85.931 MB	109.795 MB

2.2 Noisy data or features: UCI Adult Income Dataset

2.2.1 Scenario and objective

- **Scenario**: Predicting whether income exceeds $50K/yr based on census data. Some features may contain errors, outliers, or may be irrelevant.
- **Objective**: The goal is to create a model that can accurately predict income level while being robust to noise in the data.

2.2.2 Dataset

The UCI Adult Income Dataset, also known as the “Census Income” dataset, is a popular dataset used in machine learning for classification tasks. The prediction task is to determine whether a person makes over $50K a year. There are 48,842 instances in the dataset and 14 features.

Z-score is used to detect outliers in a dataset, the following results are the Z-score for each feature.

```
Column [Age] number of outliers is: 143 (0.4391757009919843%)
Column [Workclass] number of outliers is: 9865 (30.296981050950524%)
Column [Education-Num] number of outliers is: 1198 (3.679248180338442%)
Column [Marital Status] number of outliers is: 0 (0.0%)
Column [Occupation] number of outliers is: 0 (0.0%)
Column [Relationship] number of outliers is: 0 (0.0%)
Column [Race] number of outliers is: 4745 (14.572648260188569%)
Column [Sex] number of outliers is: 0 (0.0%)
Column [Capital Gain] number of outliers is: 2712 (8.328982525106722%)
Column [Capital Loss] number of outliers is: 1519 (4.665090138509259%)
Column [Hours per week] number of outliers is: 9008 (27.664998003746813%)
Column [Country] number of outliers is: 3391 (10.414299315131599%)
```

Figure 2: Results of Z-score

It is noticeable that the data set is noisy since the number of outliers (noise) is high in many features.

2.2.3 Hyperparamter tuning

2.2.3.1 Random forest

```
1. param_grid = {
2.     'n_estimators': [100, 150, 200],
3.     'max_features': ['sqrt', 'log2', None],
4.     'max_depth': [3, 9, None],
5.     'max_leaf_nodes': [5, 9, None],
```

2.2.3.2 XGBoost

```
1. params = {
2.     "colsample_bytree": [0.5, None],
3.     "gamma": [0.2, 0.5, None],
4.     "learning_rate": [0.5, 1, None],
5.     "max_depth": [2, None],
6.     "n_estimators": [25, 50, 100, None],
7.     "subsample": [0.6, None]
```

2.2.4 Evaluation metric

For noisy datasets, it is important to use a metric that can handle the noise effectively. So, **precision** measures the model's ability to correctly identify true positives, minimizing false positives caused by noise.

2.2.5 Results

Metric	Random forest		XGBoost	
	Before tuning	After tuning	Before tuning	After tuning
Precision	0.7212	0.7846	0.7580	0.7713

2.2.6 Computational efficiency

Computational efficiency	Random forest		XGBoost	
	Before tuning	After tuning	Before tuning	After tuning
Training time	2.096 seconds	5.954 seconds	3.228 seconds	6.540 seconds
Memory usage	428.397 MB	581.092 MB	338.972 MB	412.805 MB

2.3 Large dataset: Forest cover type dataset

2.3.1 Scenario and objective

- **Scenario**: Predicting forest cover type from cartographic variables only. The dataset is quite large with over half a million instances.

- **Objective**: The goal is to accurately predict the forest cover type while efficiently handling the large size of the dataset.

2.3.2 Dataset

The Forest Cover Type Dataset is a popular dataset used in machine learning and data analysis. The dataset is used for the classification of pixels into 7 forest cover types based on attributes such as elevation, aspect, slope, hillshade, soil type, and more. The dataset has 581012 samples and 54 features.

Classes	7
Samples total	581012
Dimensionality	54
Features	int

Figure 3: General information about dataset

2.3.3 Hyperparamter tuning

2.3.3.1 Random forest

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_features': ['sqrt', 'log2'],
    'max_depth': [2, 3, None],
    'max_leaf_nodes': [2, 3, None]
}
```

2.3.3.2 XGBoost

```
params = {
    "colsample_bytree": [0.5, None],
    "gamma": [0.2, None],
    "learning_rate": [0.9, None],
    "max_depth": [15, None],
    "n_estimators": [100, 150, None],
    "subsample": [0.9, None]
}
```

2.3.4 Evaluation metric

In the case of large datasets, computational efficiency becomes crucial. Models need to be scalable and capable of handling big data efficiently. **Accuracy** is often used as it is relatively straightforward and efficient to compute.

2.3.5 Results

Metric	Random forest		XGBoost	
	Before tuning	After tuning	Before tuning	After tuning
Accuracy	0.9538	0.9544	0.8716	0.9700

2.3.6 Computational efficiency

Computational efficiency	Random forest		XGBoost	
	Before tuning	After tuning	Before tuning	After tuning
Training time	89.722 seconds	177.328 seconds	150.224 seconds	221.326 seconds
Memory usage	935.174 MB	1183.146 MB	363.845 MB	445.943 MB

3 Conclusion and recommendations

Both Random Forest and XGBoost have their own strengths and weaknesses.

- **Imbalanced Classes**: Both algorithms can handle imbalanced classes, but XGBoost is more memory-efficient due to its handling of sparse data and its inbuilt parameter for class imbalance. On the other hand, Random Forest does not have a specific mechanism to handle imbalanced classes.
- **Noisy Data or Features:** Random Forest can handle noisy data well and might be faster to train as it does not require iterative optimization like XGBoost. However, XGBoost might use less memory as it builds trees one at a time, unlike Random Forest which builds all trees independently.
- **Large Datasets**: XGBoost is often favored for large datasets because it is computationally efficient. However, it might take longer to train due to the need for iterative optimization. Random Forest might be faster to train on large datasets, but it could use more memory as it needs to store all the trees in memory.

Table 1: Performance summary

Dataset issue	Evaluation Metric	Random forest	XGBoost
Imbalanced dataset	F1 score	0.9775	0.9831
Noisy dataset	Precision	0.7846	0.7713
Large dataset	Accuracy	0.9546	0.9700

First of all, the differences between two models are small.

- When the dataset is imbalanced, the XGBoost performs better.
- Random forest performs better than XGBoost when the data is noisy.
- In case of large datasets, XGBoost is preferred.

Table 2: Computational efficiency summary

Dataset issue	Random forest		XGBoost	
	Training time	Memory usage	Training time	Memory usage
Imbalanced dataset	0.512	145.380	0.628	109.795
Noisy dataset	5.954	581.092	6.540	412.805
Large dataset	177.328	1183.146	221.326	445.943

There are another considerations must be taken into account which are training time and memory usage since they may be limited.

If the model is sensitive to time, then:

- Use random forest if the dataset is imbalanced since it is faster.
- Consider random forest for noisy dataset.
- XGBoost is faster in the case of large dataset.

If the resources are limited, especially memory, then consider using XGBoost since it is memory-efficient.

The mentioned tables and notes above summarized the objective of this assignment. So, it is clearly mentioned when to use each model.

References

(1) XGBoost vs Random Forest, which is better? - Stephen Allwright. https://stephenallwright.com/xgboost-vs-random-forest/.
(2) What is Random Forest? | IBM. https://www.ibm.com/topics/random-forest.
(3) Random Forests(TM) in XGBoost — xgboost 2.0.3 documentation. https://xgboost.readthedocs.io/en/stable/tutorials/rf.html.
(4) Random Forest vs XGBoost | Top 5 Differences You Should Know - EDUCBA. https://www.educba.com/random-forest-vs-xgboost/.
(5) XGBoost - GeeksforGeeks. https://www.geeksforgeeks.org/xgboost/.
(6) A Gentle Introduction to XGBoost for Applied Machine Learning. https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/.
(7) Understanding XGBoost Algorithm In Detail - Analytics India Magazine. https://analyticsindiamag.com/xgboost-internal-working-to-make-decision-trees-and-deduce-predictions/.
(8) XGBoost Algorithm | Brief Guide to XGBoost Algorithm - EDUCBA. https://www.educba.com/xgboost-algorithm/.
(9) Introduction to XGBoost and its Uses in Machine Learning. https://www.aiplusinfo.com/blog/introduction-to-xgboost-and-its-uses-in-machine-learning/.
(10) A Guide on XGBoost hyperparameters tuning | Kaggle. https://www.kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning.
(11) How to Develop Random Forest Ensembles With XGBoost. https://machinelearningmastery.com/random-forest-ensembles-with-xgboost/.
(12) Random Forests in XGBoost — xgboost 0.90 documentation - Read the Docs. https://federated-xgboost.readthedocs.io/en/latest/tutorials/rf.html.
(13) How To Train A Random Forest With XGBoost | Forecastegy. https://forecastegy.com/posts/xgboost-random-forest/.
(14) Comparing Decision Tree Algorithms: Random Forest vs. XGBoost. https://www.activestate.com/blog/comparing-decision-tree-algorithms-random-forest-vs-xgboost/.
(15) How do we decide between XGBoost, RandomForest and Decision tree?. https://datascience.stackexchange.com/questions/78542/how-do-we-decide-between-xgboost-randomforest-and-decision-tree.
(16) What Is Random Forest? A Complete Guide | Built In. https://builtin.com/data-science/random-forest-algorithm
(17) Random Forest - Definition, Algorithms With Examples. https://corporatefinanceinstitute.com/resources/data-science/random-forest/.
(18) An introduction to random forests - univ-toulouse.fr. https://perso.math.univ-toulouse.fr/motimo/files/2013/07/random-forest.pdf.
(19) A Simple Introduction to Random Forests - Statology. https://www.statology.org/random-forests/.
(20) en.wikipedia.org. https://en.wikipedia.org/wiki/XGBoost.
(21) Random Forest Pros & Cons | HolyPython.com. https://holypython.com/rf/random-forest-pros-cons/.

(22) What are the advantages and disadvantages of random forest?. https://www.rebellionresearch.com/what-are-the-advantages-and-disadvantages-of-random-forest.
(23) What are the disadvantages of random forest? - Rebellion Research. https://www.rebellionresearch.com/what-are-the-disadvantages-of-random-forest.
(24) Advantages and disadvantages of Random Forest | AIML.com. https://aiml.com/what-are-the-advantages-and-disadvantages-of-random-forest/.
(25) XGBoost: Everything You Need to Know - neptune.ai. https://neptune.ai/blog/xgboost-everything-you-need-to-know.
(26) XGBoost Simply Explained (With an Example in Python) - Springboard. https://www.springboard.com/blog/data-science/xgboost-explainer/.
(27) What are the limitations while using XGboost algorithm?. https://datascience.stackexchange.com/questions/44169/what-are-the-limitations-while-using-xgboost-algorithm.
(28) XGBoost Regressor - APMonitor. https://apmonitor.com/pds/index.php/Main/XGBoostRegressor.
(29) en.wikipedia.org. https://en.wikipedia.org/wiki/XGBoost.