DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Intelligent Systems Lab

ENEE5141

Assignment #2

Prepared By: Ibrahim Nobani

Student ID: 1190278

Instructor: Dr. Aziz Qaroush

Section: 2

Date: 25/12/2023

Abstract:

This report gives a detailed study on two of the most popular machine learning ensemble methods, Random forrest and XGBoost, highlighting the difference between bagging ensemble methods and boosting methods.

The comparison focuses on three contexts, Imbalanced classes, Noisy data or features, and large datasets. We dive deep into how each methods accommodates into each of these problems, how they handle, and how both methods adjust to them using different combinations of hyperparameters for each.

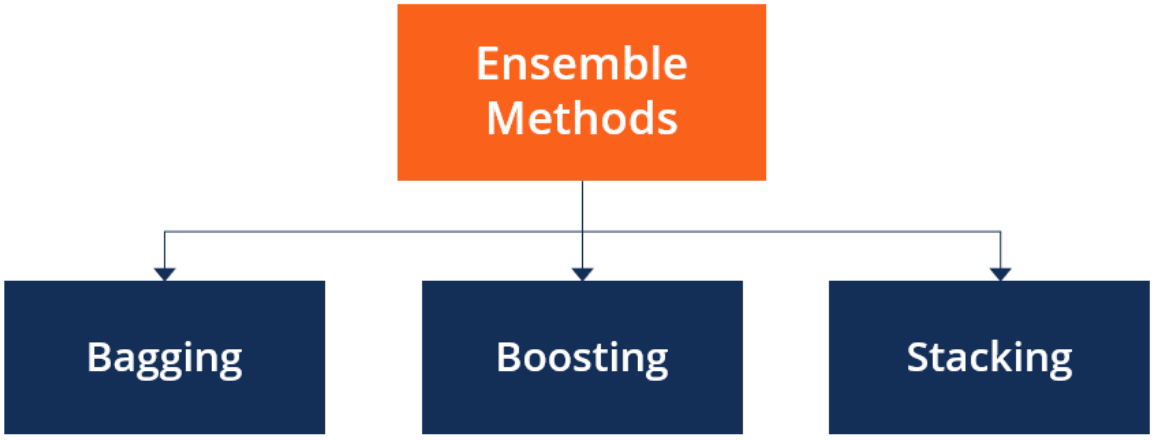
Contents

Introduction:

Ensemble Methods:

Ensemble methods are techniques that aim at improving the accuracy of results in models by combining multiple models instead of using a single model. The combined models increase the accuracy of the results significantly. This has boosted the popularity of ensemble methods in machine learning. [1]

Ensemble methods aim at improving predictability in models by combining several models to make one very reliable model. The most popular ensemble methods are boosting, bagging, and stacking. Ensemble methods are ideal for regression and classification, where they reduce bias and variance to boost the accuracy of models.



1. Bagging

Bagging, the short form for bootstrap aggregating, is mainly applied in classification and regression. It increases the accuracy of models through decision trees, which reduces variance to a large extent. The reduction of variance increases accuracy, eliminating overfitting, which is a challenge to many predictive models.

Bagging is classified into two types, i.e., bootstrapping and aggregation. Bootstrapping is a sampling technique where samples are derived from the whole population (set) using the replacement procedure. The sampling with replacement method helps make the selection

procedure randomized. The base learning algorithm is run on the samples to complete the procedure.

Aggregation in bagging is done to incorporate all possible outcomes of the prediction and randomize the outcome. Without aggregation, predictions will not be accurate because all outcomes are not put into consideration. Therefore, the aggregation is based on the probability bootstrapping procedures or on the basis of all outcomes of the predictive models.

Bagging is advantageous since weak base learners are combined to form a single strong learner that is more stable than single learners. It also eliminates any variance, thereby reducing the overfitting of models. One limitation of bagging is that it is computationally expensive. Thus, it can lead to more bias in models when the proper procedure of bagging is ignored.

2. Boosting

Boosting is an ensemble technique that learns from previous predictor mistakes to make better predictions in the future. The technique combines several weak base learners to form one strong learner, thus significantly improving the predictability of models. Boosting works by arranging weak learners in a sequence, such that weak learners learn from the next learner in the sequence to create better predictive models.

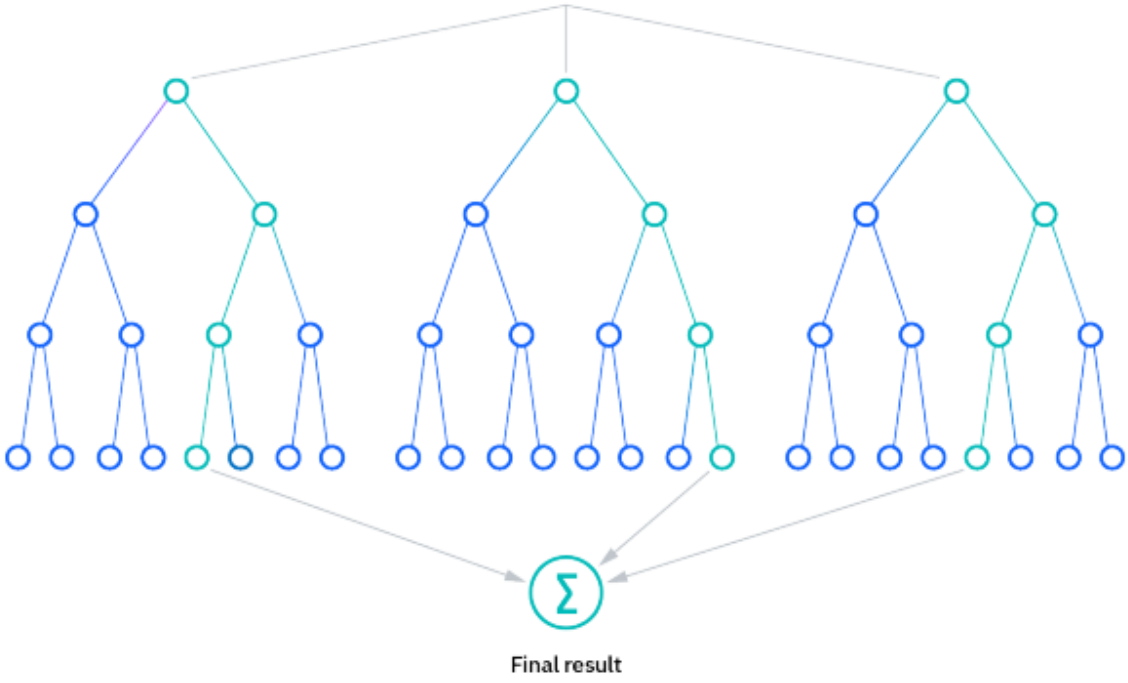
Boosting takes many forms, including gradient boosting, Adaptive Boosting (AdaBoost), and XGBoost (Extreme Gradient Boosting). AdaBoost uses weak learners in the form of decision trees, which mostly include one split that is popularly known as decision stumps. AdaBoost's main decision stump comprises observations carrying similar weights.

Random Forrest:

The random forest algorithm is an extension of the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees. Feature randomness, also known as feature bagging or "the random subspace method"(link resides outside ibm.com), generates a random subset of features, which ensures low correlation among decision trees. This is a key difference between decision trees and random forests. While decision trees consider all the possible feature splits, random forests only select a subset of those features. [2]

The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised of a data sample drawn from a training set with replacement, called the

bootstrap sample. Of that training sample, one-third of it is set aside as test data, known as the out-of-bag (oob) sample, which we'll come back to later. Another instance of randomness is then injected through feature bagging, adding more diversity to the dataset and reducing the correlation among decision trees. Depending on the type of problem, the determination of the prediction will vary. For a regression task, the individual decision trees will be averaged, and for a classification task, a majority vote—i.e. the most frequent categorical variable—will yield the predicted class. Finally, the oob sample is then used for cross-validation, finalizing that prediction.



Hyperparameters:

1) N-Estimators: The number of trees in the forest.

2) criterion: defines the function which determine the quality of the split for the trees, we will the two most common criteria: gini and entropy.

XGBoost:

XGBoost is a robust machine-learning algorithm that can help you understand your data and make better decisions. XGBoost is an implementation of gradient-boosting decision trees. It has been used by data scientists and researchers worldwide to optimize their machine-learning models.

XGBoost is a widespread implementation of gradient boosting. Let's discuss some features of XGBoost that make it so attractive. XGBoost offers regularization, which allows you to control overfitting by introducing L1/L2 penalties on the weights and biases of each tree. This feature is not available in many other implementations of gradient boosting. Another feature of XGBoost is its ability to handle sparse data sets using the weighted quantile sketch algorithm. This algorithm allows us to deal with non-zero entries in the feature matrix while retaining the same computational complexity as other algorithms like stochastic gradient descent. XGBoost also has a block structure for parallel learning. It makes it easy to scale up on multicore machines or clusters. It also uses cache awareness, which helps reduce memory usage when training models with large datasets [3].

Gradient boosting adds predictors sequentially to the ensemble, where preceding predictors correct their successors, thereby increasing the model's accuracy. New predictors are fit to counter the effects of errors in the previous predictors. The gradient of descent helps the gradient booster identify problems in learners' predictions and counter them accordingly [1].

XGBoost makes use of decision trees with boosted gradient, providing improved speed and performance. It relies heavily on the computational speed and the performance of the target model. Model training should follow a sequence, thus making the implementation of gradient boosted machines slow [1].

Hyperparameters:

1) Learning rate: it is a regularization parameter that shrinks feature weights in each boosting step.
2) N-Estimators: number of trees to be generated and boosted.

Procedure and analysis

1. Imbalanced classes.

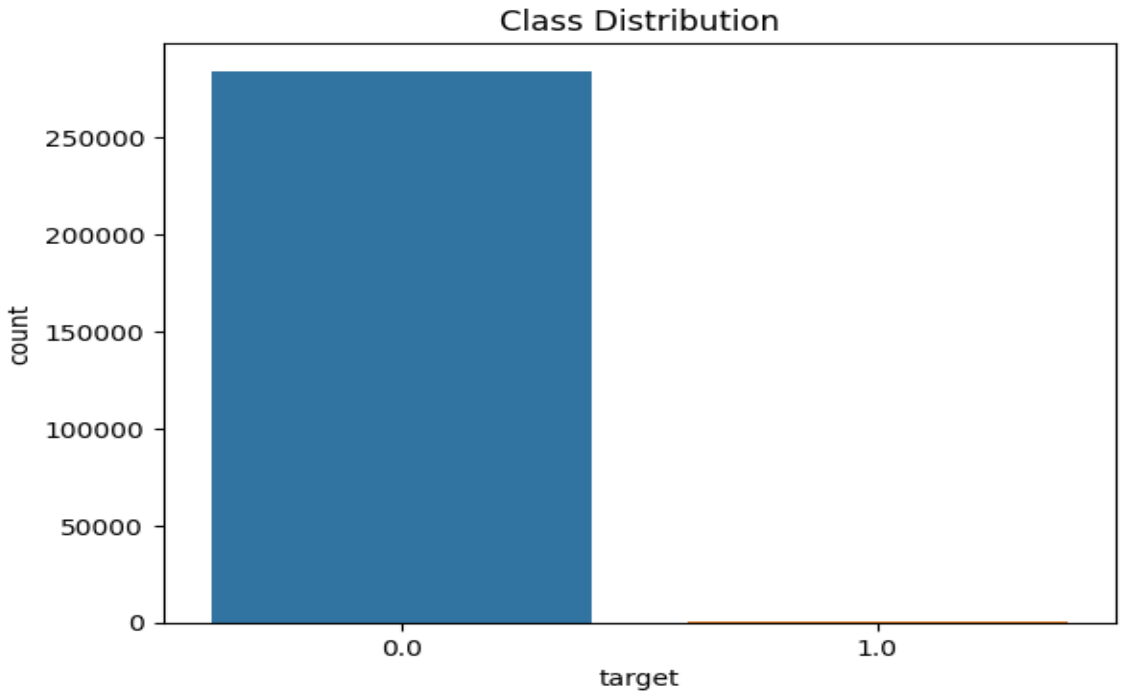
Credit Card Fraud Detection Dataset:

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions, it contains only numerical input variables which are the result of a PCA transformation [4].

The dataset is highly imbalanced, meaning that the number of legitimate transactions (non-fraudulent) significantly outweighs the number of fraudulent transactions. This characteristic is typical for fraud detection datasets, where the occurrence of fraudulent events is rare compared to non-fraudulent ones.

Because the classes are unbalanced, the accuracy metric wont be considered as the main metric, as the classes will predict the '0' value most of the time because it's the majority, other metrics will be used like precision, recall and AUC-Roc which is advised from the dataset makers.

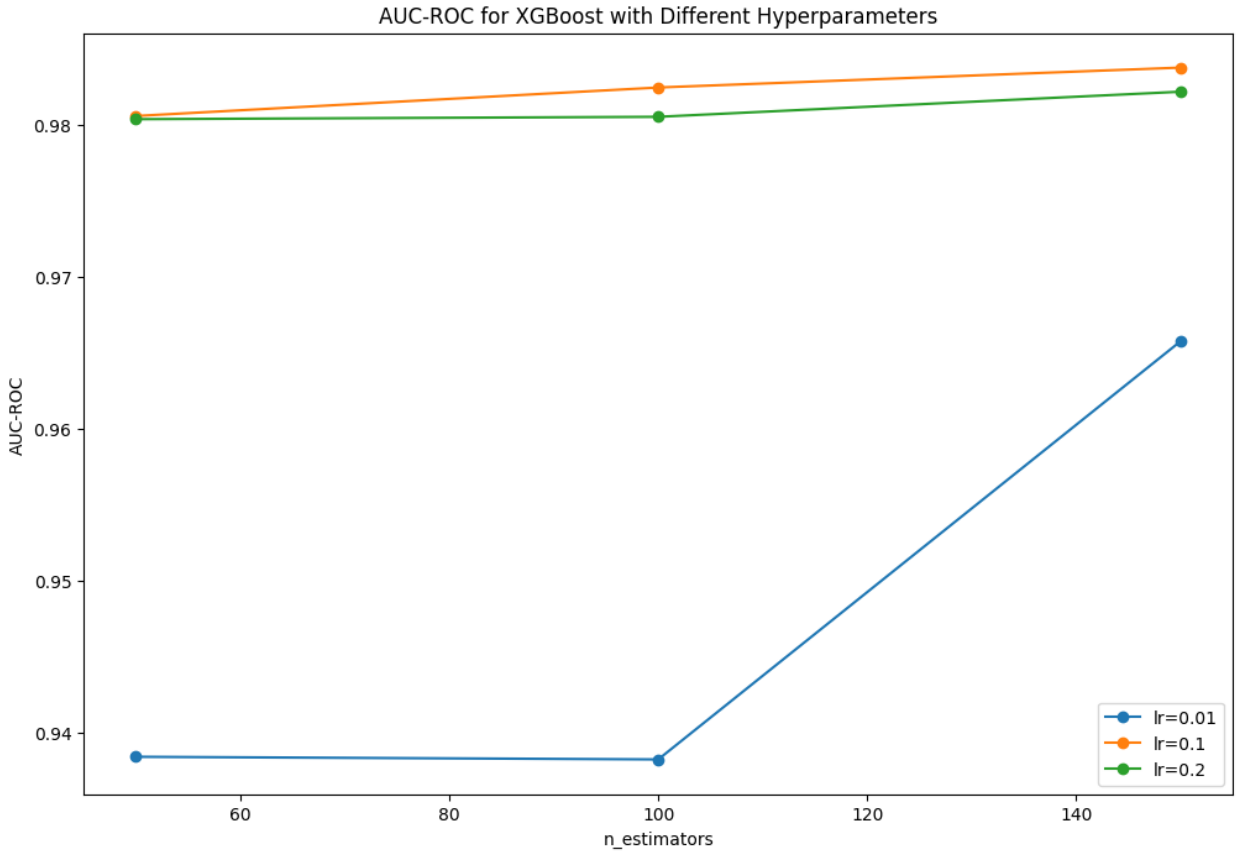
Here is a visualization of the unbalanced classes in this dataset:



First, I ran the XGBoost since its much faster, on different estimators and learning rates for regularization, the results are documented in the table below:

n_estimators	learning_rate	Training Time (seconds)	Precision (Class 1)	Recall (Class 1)	F1-Score (Class 1)	AUC-ROC
50	0.01	5.02	0.00	0.00	0.00	0.9384
50	0.1	2.61	0.98	0.81	0.88	0.9806
50	0.2	4.83	0.96	0.80	0.87	0.9804
100	0.01	3.29	0.99	0.67	0.80	0.9383
100	0.1	4.30	0.97	0.78	0.86	0.9824
100	0.2	6.59	0.96	0.79	0.87	0.9805
150	0.01	4.26	0.99	0.73	0.84	0.9658
150	0.1	8.18	0.97	0.80	0.88	0.9838
150	0.2	6.00	0.96	0.78	0.86	0.9822

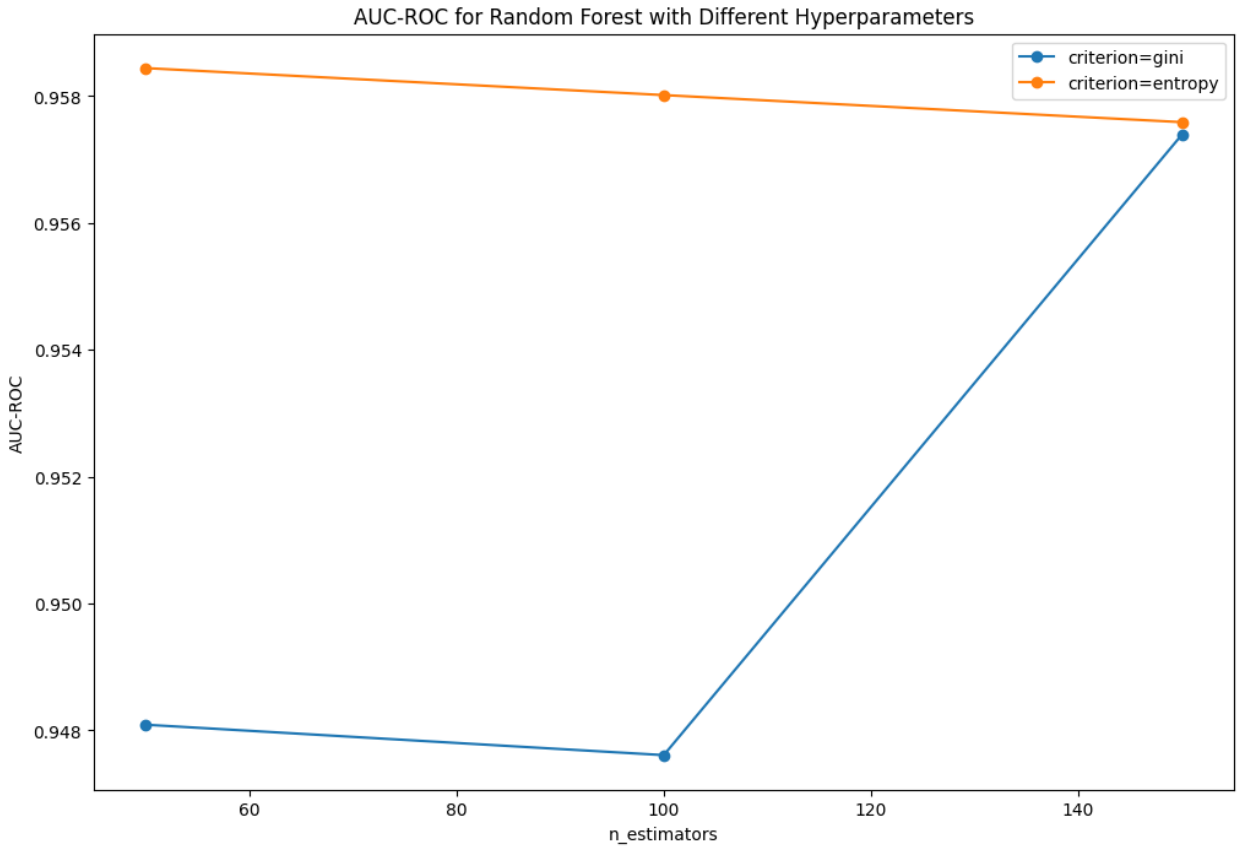
As seen, there is a difference between each iteration, what we can observe is the following The AUC-ROC values generally increase with higher values of n_estimators. Higher learning rates like 0.1 and 0.2 tend to have better AUC-ROC values compared to a lower learning rate 0.01. There is also a trade-off between AUC-ROC performance and training with higher learning rates and n_estimators leading to longer training times. Check the image below for more visualization about the AUC-ROC for different hyperparameters:



Precision is generally high across different hyperparameter configurations its always 1 for the majority class and varies between 0.96 and 0.99 for the minority class, Recall varies across models, with higher values observed for the minority class in configurations with higher learning_rate, The best F1-Score is observed for n_estimators=150 and learning_rate=0.1, same as the AUC-ROC.

As for Random Forest, I also tried different combinations of values and produced the following results:

n_estimators	criterion	Training Time (seconds)	Precision (Class 1)	Recall (Class 1)	F1-Score (Class 1)	AUC-ROC
50	gini	153.68	0.97	0.76	0.85	0.9481
50	entropy	97.00	0.97	0.79	0.87	0.9584
100	gini	290.64	0.97	0.77	0.86	0.9476
100	entropy	191.21	0.97	0.80	0.88	0.9580
150	gini	438.73	0.97	0.77	0.86	0.9574
150	entropy	309.09	0.97	0.80	0.88	0.9576



As expected the training time is longer for more number of estimators, also the entropy criterion is relatively faster than the default gini criterion.

Precision for the minority class (Class 1) is consistently high, indicating a low rate of false positives. Recall for the minority class shows a slight decrease with higher n_estimators. It remains relatively high across configurations. F1-Score for the minority class remains stable across configurations, with a slight decrease for n_estimators=150. AUC-ROC values are consistently high across different hyperparameter configurations, the highest F1 Score and AUC-ROC is for n_estimators=50 and criterion=entropy.

Both methods perform well on the given dataset with unbalanced classes, achieving high precision for the minority class. XGBoost tends to provide higher AUC-ROC values and, in some cases, better recall for the minority class.

In conclusion, XGBoost appears to be a slightly better choice for unbalanced classes on this dataset, considering its ability to achieve higher AUC-ROC. Random forest as well can be seen to be more computationally expensive looking at the high training times compared to XGBoost.

“In XGBoost, when the model fails to predict the anomaly for the first time, it gives more preferences and weightage to it in the upcoming iterations thereby increasing its ability to predict the class with low participation; but we cannot assure that random forest will treat the class imbalance with a proper process [5].”

2. Noisy data or features:

Breast Cancer Dataset:

The Breast Cancer dataset is a widely used benchmark in machine learning, particularly in the context of binary classification tasks. It consists of features computed from digitized images of fine needle aspirates (FNA) of breast masses, with the goal of distinguishing between malignant and benign tumors. The dataset encompasses 30 real-valued features, representing characteristics such as mean radius, texture, and smoothness, among others. Each instance is labeled as either malignant or benign [6].

For this part I used the breast cancer dataset, I added some random noise to its data and features in order to study it from there.

First, I started by evaluating the dataset before adding noise and produced the following results:

Random Forrest:

```
Best Hyperparameters for Random Forest: {'criterion': 'entropy',
'n_estimators': 150}

AUC-ROC on Test Set (Random Forest): 0.9983622666229938

Precision on Test Set (Random Forest): 0.958904109589041

Recall on Test Set (Random Forest): 0.9859154929577465

F1 Score on Test Set (Random Forest): 0.9722222222222222
```

XGBoost:

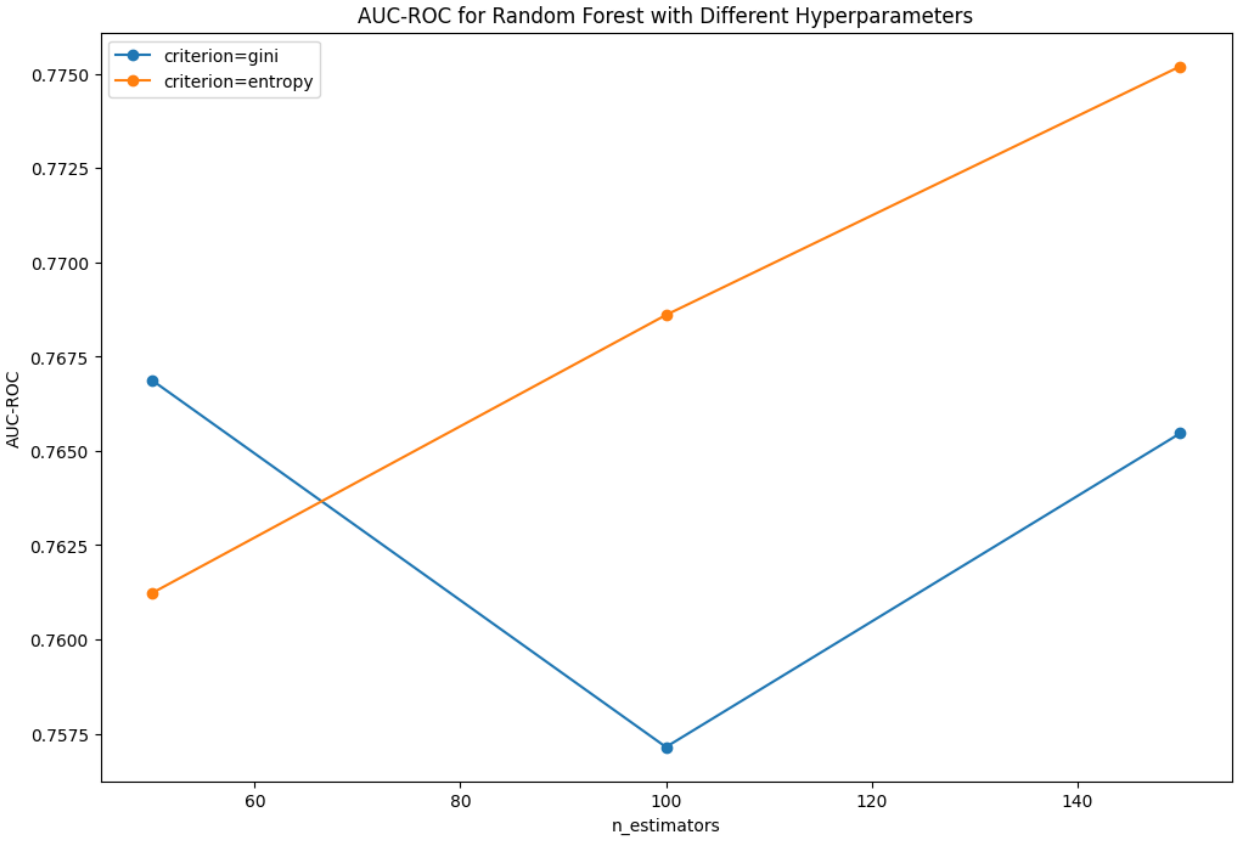
```
Best Hyperparameters for XGBoost: {'learning_rate': 0.2, 'n_estimators': 150}
AUC-ROC on Test Set (XGBoost): 0.9927939731411726
Precision on Test Set (XGBoost): 0.9583333333333334
Recall on Test Set (XGBoost): 0.971830985915493
F1 Score on Test Set (XGBoost): 0.965034965034965
```

Now I am introducing noise on the dataset to see which model performs better, clearly Random Forest does best on this dataset as we have seen, also according to [6].

Random Forrest:

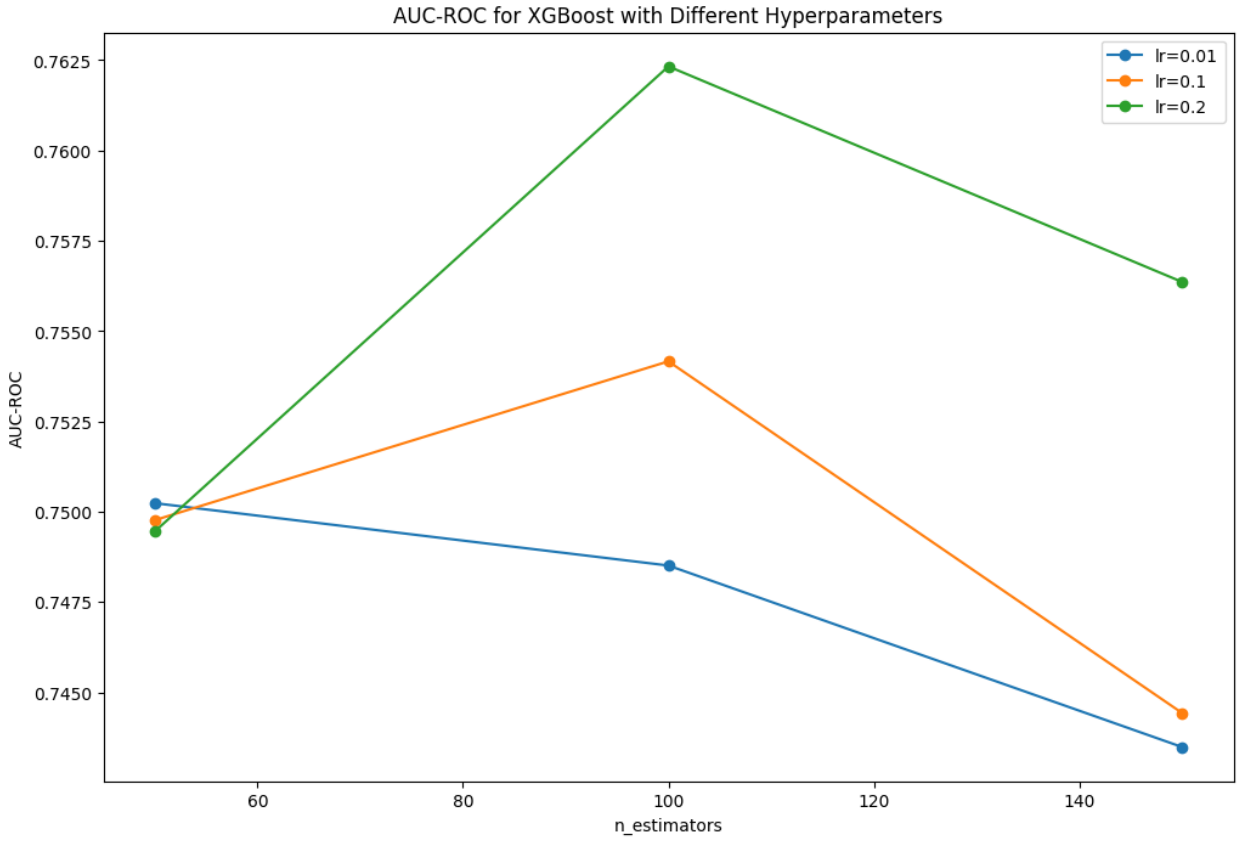
n_estimators	criterion	Training Time (s)	Precision (0)	Precision (1)	Recall (0)	Recall (1)	F1-score (0)	F1-score (1)	Accuracy	AUC-ROC
50	gini	0.17	0.70	0.75	0.65	0.78	0.67	0.77	0.73	0.7669
50	entropy	0.18	0.72	0.75	0.63	0.82	0.67	0.78	0.74	0.7612
100	gini	0.31	0.70	0.74	0.63	0.80	0.67	0.77	0.73	0.7571
100	entropy	0.34	0.78	0.76	0.63	0.86	0.70	0.81	0.76	0.7686
150	gini	0.46	0.78	0.76	0.63	0.86	0.70	0.81	0.76	0.7655
150	entropy	0.50	0.77	0.75	0.61	0.86	0.68	0.80	0.75	0.7752

Overall, Random Forest performs reasonably well in handling noise, we can notice that increased estimators increase training time, the entropy criterion is relatively better when the estimator is (100, 150), for class 0 we can see better metrics values. Despite noise, the F1-score remained relatively stable across different hyperparameter settings, indicating a consistent trade-off between precision and recall. The following plot is the AUC ROC for each parameter:



XGBoost:

n_estimators	learning_rate	Training Time (s)	Precision (0)	Precision (1)	Recall (0)	Recall (1)	F1-score (0)	F1-score (1)	Accuracy	AUC-ROC
50	0.01	2.24	0.75	0.72	0.55	0.86	0.64	0.78	0.73	0.7502
50	0.1	2.91	0.75	0.74	0.61	0.85	0.67	0.79	0.75	0.7498
50	0.2	1.27	0.76	0.76	0.65	0.85	0.70	0.80	0.76	0.7495
100	0.01	4.53	0.78	0.76	0.63	0.86	0.70	0.81	0.76	0.7485
100	0.1	3.12	0.76	0.76	0.65	0.85	0.70	0.80	0.76	0.7542
100	0.2	0.81	0.77	0.77	0.67	0.85	0.72	0.81	0.77	0.7623
150	0.01	1.56	0.76	0.74	0.59	0.86	0.67	0.79	0.75	0.7435
150	0.1	0.46	0.74	0.75	0.63	0.83	0.68	0.79	0.75	0.7444
150	0.2	0.35	0.75	0.77	0.67	0.83	0.71	0.80	0.76	0.7564



XGBoost achieved the highest AUC-ROC of 0.7623 with 100 estimators and a learning rate of 0.2., learning rate played a role in balancing precision and recall. Higher learning rates tended to result in improved recall, F1 Score remained consistent across different hyperparameters meaning there is a balance between recall and precision.

In the comparison between Random Forest (RF) and XGBoost, both shown robustness in handling noise, with RF maintaining stable precision and recall levels and XGBoost showing adaptability to noisy features, especially after tunning the learning rate. RF, being an ensemble method, tends to be more interpretable and is less sensitive to hyperparameter tuning. On the other hand, XGBoost, with its boosting technique, can capture complex relationships and adapt to noise more flexibly.

3. Large Datasets:

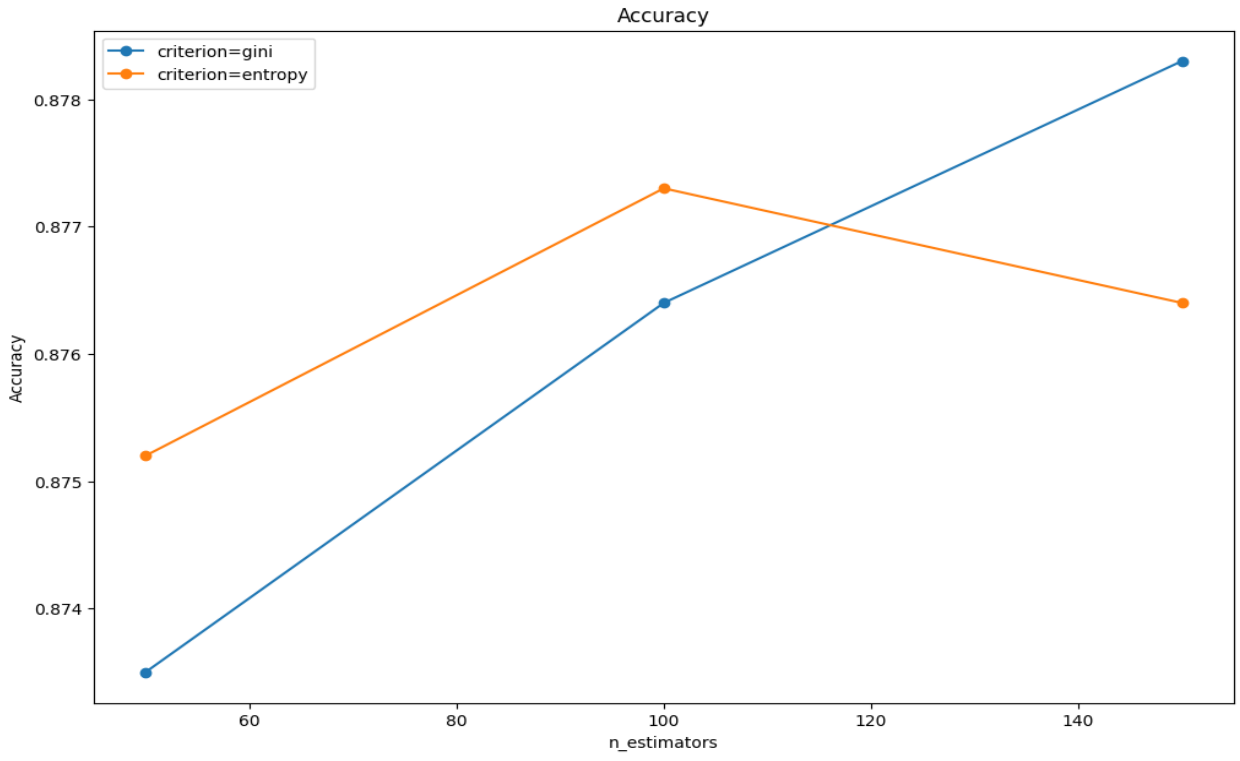
MNIST-Fashion Dataset:

Fashion-MNIST is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. I loaded the dataset from the TensorFlow library here [7].

First, I started by evaluating the dataset on the random forest:

Here I will be using the accuracy as the main metric

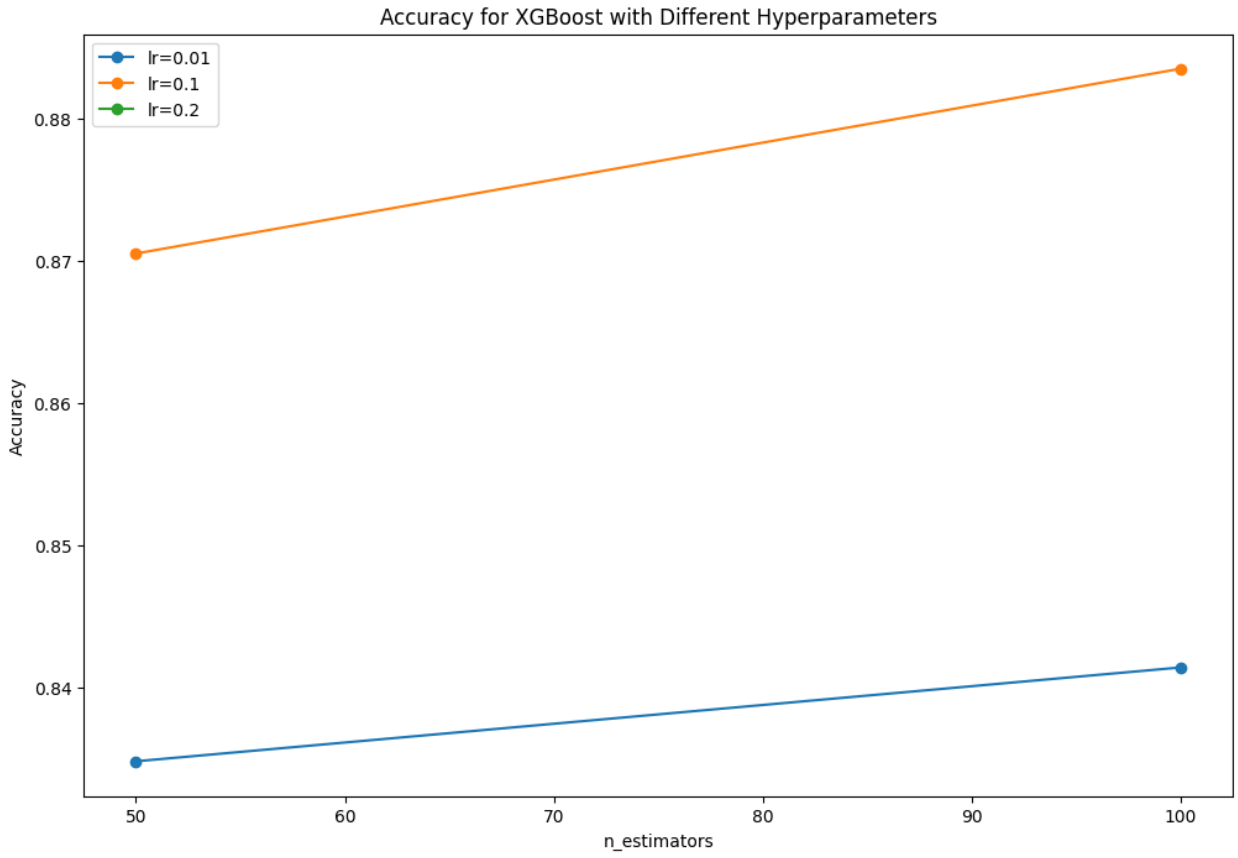
n_estimators	Criterion	Training Time (seconds)	Accuracy	AUC-ROC
50	Gini	47.56	0.8735	0.9885
50	Entropy	54.42	0.8752	0.9887
100	Gini	94.40	0.8764	0.9893
100	Entropy	101.86	0.8773	0.9896
150	Gini	142.59	0.8783	0.9896
150	Entropy	152.63	0.8764	0.9899



The model consistently achieves high accuracy and AUC-ROC scores across different hyperparameter settings, we can see them both increase with the estimators, also with the criterion change, but the criterion does not seem to increase the accuracy by much. The most efficient thing about random forest in the context of large datasets is its high speed in training this is because of building multiple decision trees independently, resulting in better computation.

XGBoost:

n_estimators	Learning Rate	Training Time (seconds)	Accuracy	AUC-ROC
50	0.01	266.21	0.8348	0.9813
50	0.1	271.09	0.8705	0.9888
100	0.01	525.90	0.8414	0.9832
100	0.1	512.39	0.8835	0.9910
100	0.2	502.75	0.8942	0.9921



Higher learning rates (0.1) tend to result in better accuracy and AUC-ROC compared to lower learning rates (0.01). Training time increases with the number of estimators and higher learning rates, indicating a trade-off between model complexity and training time.

I also tried using a learning rate of 0.2 to further see how the model goes, it produced better accuracy, resulting in the best results for any of the other combinations whether it was random forest or XGBoost with an accuracy of 0.8942. Further proving our initial claim.

As we notice from the results of both methods, random forest produces good results with better training time, while XGBoost has better results once the parameter are tuned correctly, but the training time is much more, therefore it has a big cost computationally.

The choice between either of them in the context of large datasets depends on the computational resources, if available XGBoost produces better results, but if not, random forest gives great results as well with less training time. So, the choice comes down the trade-off someone is willing to give between computations and accuracy.

Conclusion:

This report gives a detailed study and comparison between two of the most popular machine learning ensemble methods, Random Forest and XGBoost by highlighting the difference between bagging and boosting methods.

First a study into how both these methods can handle unbalanced classes, the dataset used was the credit fraud dataset and presented as a scenario for this topic, we saw the result of both methods using a wide combination of hyperparameters, XGBoost appears to be a slightly better choice for unbalanced classes on this dataset, considering its ability to achieve higher AUC-ROC.

Then we saw how both methods perform on noisy data, we started by loading the breast cancer dataset, we ran a test on it to see the performance before introducing noise to the data, after introducing the noise, different hyperparameters were done and noticed, as a result we saw that both methods show robustness in handling noise,

For the last part, we tried both algorithms on large datasets, particularly the multi-class Mnist-Fashion dataset which contains 60k images, both of our models achieve great results, but there is a huge trade-off, random forest has much faster training times that XGBoost, while XGBoost provides more accurate results as we tune in the parameters to an accepted result, so it depends on the computational resources and the trade-off with accuracy someone is willing to do.

References

[1] [Online]. Available: https://corporatefinanceinstitute.com/resources/data-science/ensemble-methods/.

[2] [Online]. Available: https://www.ibm.com/topics/random-forest.

[3] [Online]. Available: https://www.simplilearn.com/what-is-xgboost-algorithm-in-machine-learning-article.

[4] [Online]. Available: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud.

[5] [Online]. Available: https://medium.com/geekculture/xgboost-versus-random-forest-898e42870f30.

[6] [Online]. Available: https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic.

[7] [Online]. Available: https://www.tensorflow.org/datasets/catalog/fashion_mnist.