ML Handbook

About this handbook

In thi handbook, we shall include '**Issue 1: Optimizing in Hyperparameters**' and '**Issue 3: Applying Ensemble Learning**' by describing two sections (Non-technical and Technical) for each issue.

Please refer to the Python notebook attached separately for more details on technical related information, like equations and codes.

Issues 1: Optimizing Hyperparameters

1. Non-Technical Report of Issue 1

Keywords: Hyperparameters, optimization, search space, objective function, cross-validation, grid search Random search, Bayesian optimization

1.1 Types of Hyperparameter Optimization

Hyperparameter optimization is the process of selecting the best hyperparameters for a given machine learning algorithm to improve its performance. There are several methods for optimizing hyperparameters, including:

- Grid Search: In this method, a range of values is specified for each hyperparameter, and the algorithm is trained and evaluated for every combination of hyperparameters in the specified range. The combination with the best performance is then selected.
- Random Search: This method involves randomly selecting values for each hyperparameter within a specified range, training and evaluating the model, and repeating the process for a certain number of iterations. The combination with the best performance is then selected.
- Bayesian Optimization: This method uses a probabilistic model to predict the performance of the model for a particular set of hyperparameters. It then selects the next set of hyperparameters to try based on the predicted performance, continuing until the best set of hyperparameters is found.

1.2 Issues of Hyperparameter Optimization

Issues may arise when optimizing hyperparameters:

- Lack of domain knowledge: Hyperparameter optimization can be highly dependent on domain knowledge, such as the expected scale of the data or the appropriate range of hyperparameters. Without sufficient domain knowledge, it can be difficult to design an appropriate hyperparameter search space and select the appropriate optimization method.
- Limited resources/limited access to resources: Hyperparameter optimization can require significant computational resources, such as high-performance computing clusters or cloud computing resources. Limited access to these resources can make it difficult to perform hyperparameter optimization at scale.
- Communication and collaboration: Hyperparameter optimization often involves multiple stakeholders, such as data scientists, domain experts, and project managers. Often there is a lack of right mix of parties to effectively evaluate which hyperparameters to consider.
- High cost: Hyperparameter optimization can be costly and time-consuming, particularly when using exhaustive methods such as grid search. Cost and time constraints can make it difficult to perform hyperparameter optimization at scale or to optimize hyperparameters for real-time applications.

One of the more challenging issues can be a lack of domain knowledge. This is because hyperparameter optimization can be highly dependent on domain knowledge, such as understanding the expected scale of the data or having knowledge of appropriate hyperparameters for a particular problem. Without sufficient domain knowledge, it can be challenging to design an appropriate hyperparameter search space, select the appropriate optimization method, or interpret the results of hyperparameter optimization.

1.3 Addressing the issues of Hyperparameter Optimization

Below are some strategies for addressing the issues mentioned above:

- Lack of Domain Knowledge: Collaborate with domain experts to gain insights into the problem and appropriate hyperparameters. Additionally, consider conducting literature reviews and attending conferences or workshops to gain knowledge about the field.

- Limited resources/limited access to resources: Use cloud computing resources, high-performance computing clusters, or parallel computing techniques to speed up hyperparameter optimization. Additionally, consider using cost-effective methods such as randomized search or reducing the size of the dataset to optimize the hyperparameters with limited resources.
- Communication and Collaboration: Establish clear communication channels and project management practices to ensure that all stakeholders are aware of the goals, constraints, and progress of the project. Additionally, consider using collaborative tools such as Google Docs or Trello to improve collaboration.
- High cost: Prioritize hyperparameters based on their expected impact on model performance and allocate resources accordingly.

1.4 Walkthrough on choosing right parameters and characteristics

Before performing the hyperparameter optimizations, first, we will need to follow these steps. The available data was first divided into training, test and validation sets. With a specific set of hyperparameters, the model is trained on the training set, and its effectiveness is assessed on the validation set. Cross-validation is a technique that aids in avoiding overfitting, which happens when a model is very complicated and fits the training data too closely, leading to inferior generalization to new data. This procedure is repeated for various sets of hyperparameters, and the model's performance for each set is compared. The best set of hyperparameters is chosen based on which set gives the highest performance on the validation set.

Those above steps can be automated by using the method called Hyperparameter Optimization. Ultimately, the model with the **best performance on the validation set** is selected as the final model, and its parameters are considered to be **the right parameters** for the given problem. (Ludmila)

Hyperparameter optimization is a crucial step in developing accurate machine learning models that generalize well to new data. (Yang and Shami) The development of accurate machine learning models that generalize well to new data requires the careful selection of hyperparameters. It's particularly important to remember that hyperparameter optimization **does not ensure accurate or nearly accurate forecasts for future cases**. The model's performance will probably be good on data that is similar to the validation set, but it may not generalize well to totally unseen new data. Thus, hyperparameter optimization is **just one component of building accurate ML models**. The development of models with great generalization to new data does require careful evaluation of all necessary components.

2. Technical Report of Issue 1

This section aims to present all hyperparameter optimization methods listed above, i.e., grid search, random search and Bayesian optimization.

2.1 About Dataset and mathematical expressions

The dataset used is US Census data which is an extraction of the 1994 census data which was donated to the UC Irvine's Machine Learning Repository. The data contains approximately 32,000 observations with over 15 variables. The dependent variable in our analysis will be income level and who earns above $50,000 a year.

We use the Random Forest Classifier as our model to train and test the dataset. The key steps for the classifier are illustrated as below.

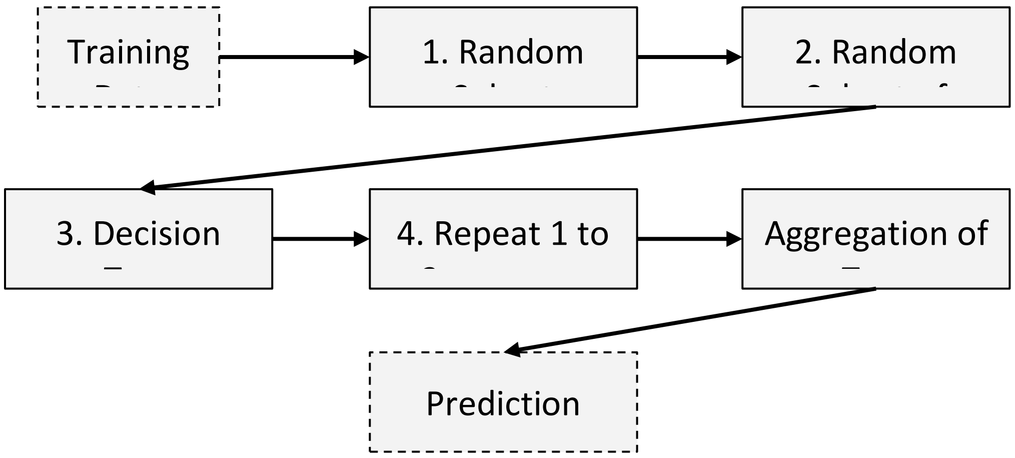

Fig 2.1: Key steps for Random Forest Classifier

The aggregation step can be done using different methods, such as taking the majority vote (in classification problems) or averaging the predictions (in regression problems). (Demir)

This equation serves as the foundation for the Random Forest classifier's final prediction:

$$y_{predicted} = argmax\,(\,1/k\ \sum f(\,x, \theta_i\,)\,)$$

where, $y_{predicted}$ is the predicted target label, k is the total numbers of decision trees in the forest, $f(\,x, \theta_i\,)$ is the function of prediction of the i^{th} decision tree on the input variable x and its respective parameters θ_i.

We choose the Random Forest algorithm because it is relatively easy to implement and does not require extensive parameter tuning. It is also robust enough to handle some complex datasets. Apart from the classifier, the above mentioned methods of Hyperparameter Optimization can be expressed as follows:

- Random Search

$$\theta_i \sim U(a_i, b_i)$$

where, θ_i is the i^{th} hyperparameter value, $U(a_i, b_i)$ is a uniform distribution between the minimum (a_i) and maximum (b_i) values of the i^{th} hyperparameter

- Grid Search

$$\theta_i = \{a_i, a_i + d_i, a_i + 2 * d_i, \ldots, b_i\})$$

where, d_i is step size of the i^{th} hyperparameter

- Bayesian Optimization

$$\theta * = argmax\, P(f(x * | x, y), D)$$

where, $\theta *$ is the corresponding set of hyperparameters to evaluate, $P(f(x * | x, y), D)$ is the posterior probability distribution of the model performance given the previous evaluations (x, y) and a new evaluation $x *$.

2.2 Computation in Python

Before training the model, first of all, the following necessary libraries are imported in our python notebook.

```
`from sklearn.ensemble import RandomForestClassifier`
`from skopt import BayesSearchCV`
` `from sklearn.model_selection import GridSearchCV, RandomizedSearchCV`
`from sklearn.metrics import roc_auc_score, roc_curve`
`from sklearn.model_selection import train_test_split`
`import time`
```

After importing the dataset into the Python notebook, `` `train_test_split` `` is applied to divide the dataset into train and test. Then, we use `` `RandomForestClassifier` `` to train on our dataset and perform hyperparameter optimization by applying `` `GridSearchCV` ``, `` `RandomizedSearchCV` `` and `` `BayesSearchCV` `` respectively. In order to know the execution time for each method, we also import `` `time` `` and evaluate the model by calculating AUC score and plotting ROC curve with following commands: `` `roc_auc_score` `` and `` `roc_curve` ``.

See attached Python Notebook for more information on the examples.

Issues 3: Applying Ensemble Learning

3. Non-Technical Report of Issue 3

Keywords: ensemble methods, bagging or bootstrapping, boosting, stacking, diversity, combination, error correction, meta-learning, random forest

3.1 Types of Ensemble Learning

Ensemble learning is a method of machine learning that integrates various independent models to increase the system's overall predictive performance. It works by creating a diverse range of models and integrating their predictions in a way that is helping to maximize accuracy, decrease overfitting, and strengthen robustness to noisy data. There are three main types of ensemble learning: Bagging, Boosting, and Stacking.

- Bagging (Bootstrap Aggregating): Bootstrapping uses the random selection with replacement method and involves creating several instances of the same model on various subsets of the training data. The outputs of each separate model are averaged to provide the final prediction, which helps to minimize overfitting.
- Boosting: Multiple weak models are sequentially trained using this method, with each succeeding model being trained to fix the flaws of the previous model. The results of each individual model are combined to create the final prediction, with the better performing models receiving more weight in the process.
- Stacking: It utilizes a meta-model to combine the predictions from various models. Stacking trains a meta-model how to weigh the predictions of the various models to produce a final prediction rather than simply combining the predictions of the various models. The meta-model, which is typically a straightforward linear model, is trained using the predictions of the base models.

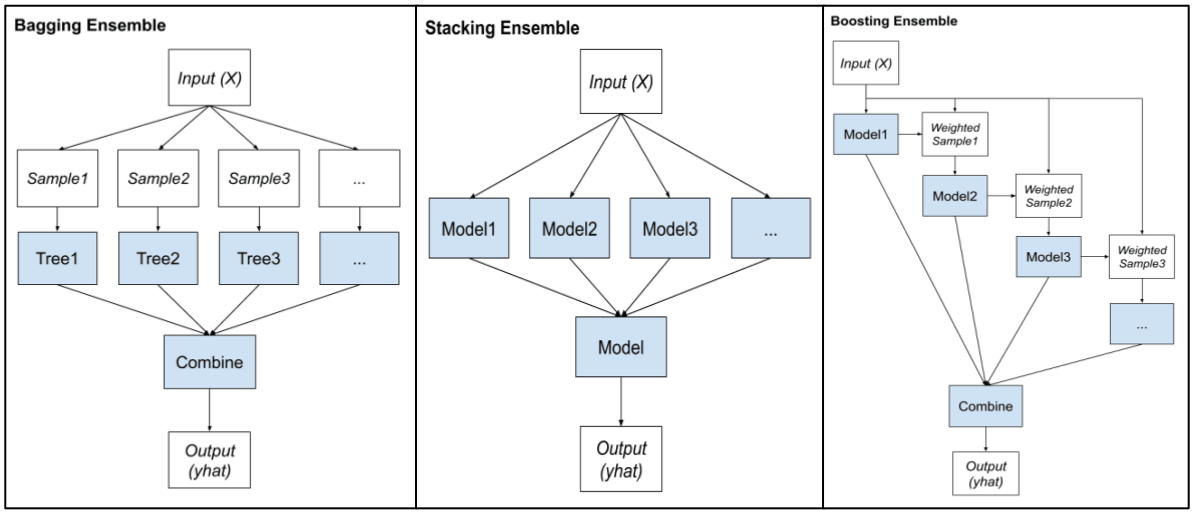

Fig 3.1: Illustrations no Ensemble Learning (Brownlee)

To summarize, bagging aims to decrease variance, boosting aims to decrease bias, and stacking aims to improve prediction accuracy. (Kalirane) Boosting and bagging combine similar weak students. Different solid learners are combined through stacking. Models are sequentially trained by boosting and simultaneously trained by bagging.

3.2 Issues of Ensemble Learning

There are a few issues that may arise from utilizing ensemble learning in practice:

- Overfitting: If the base models or the ensemble itself are overly complex, ensemble learning may result in overfitting. Inaccurate predictions and poor generalization performance may arise as issues.
- Model Interpretability: It can be challenging to comprehend how ensemble models make predictions, which is why they are frequently referred to as "black box" models. This lack of transparency may be problematic, particularly when the model's predictions could have important real-world repercussions. (Tannor and Tannor)
- Computationally Expensive: Due to the fact that ensemble learning involves training multiple models and combining their predictions, it can be computationally expensive and time-consuming. Due to this, it may be challenging to use the method in real-time applications or to scale it up for larger datasets. (Sarkar)
- Diversity of Models: The variety of the base models used affects how well ensemble learning works. The ensemble may not function well if the models are too similar or are based on similar

algorithms. It can be difficult to ensure model diversity, especially when working with constrained resources or time.

While ensemble methods can be helpful in reducing variance and creating more robust models, there are many drawbacks to using them, such as decreased performance and a lack of explainability. It is important to keep in mind that ensembles derive their strength from their ability to combine various models that concentrate on various aspects of the problem.

3.3 Addressing the issues of Ensemble Learning

Here are some ways to address, avoid or solve the issues that arise from utilizing ensemble learning in practice:

- Model Performance: Cross-validation methods should be used during model training in order to prevent overfitting, as well as keeping the complexity of the base models to a minimum. To lessen overfitting, feature selection and regularization techniques can also be used. On the other hand, by making the base models more complex or by using more sophisticated algorithms that can capture intricate relationships between the predictors and the target variable, underfitting can be prevented. In that way, model performance can be improved.
- Model Selection: The selection of the base models and the ensemble method should be done with great care. The ensemble method should be compatible with the models chosen, as well as their individual performance, diversity, and suitability. Based on how well they can combine the base models, ensemble methods should be chosen.
- Computational space: When assembling large datasets and intricate models, it can be computationally expensive. Using parallel processing methods or more effective algorithms can solve this problem. (Maclin and Opitz)
- Interpretability: The interpretation of ensemble models can be challenging, especially when using sophisticated techniques like stacking. Transparent models should be used as base learners to solve this problem, and feature selection techniques should be applied to lessen the number of predictors. (Demir)
- Preprocessing: Since it can amplify the effects of noisy or biased data, data preprocessing, like data cleaning techniques to remove errors and outliers, can be applied. Techniques like oversampling or undersampling are used to address the imbalanceness in the class.

3.4 Characteristics and Walkthrough of Ensemble Learning

Assuming the required dataset has been already imported and explored, here is the brief information about three ensemble learning methods.

3.4.1 Walkthrough

All processes usually start with splitting the data into training and testing sets. The weak learning algorithm for boosting and for other two, the base model used shall need to be defined such as logistic regression or decision trees or linear regression. After that, train the model by applying the necessary libraries in the notebook. In order to improve the performance, the hyperparameters of the base model shall need to be tuned. Then, train the model again with tuned parameters and evaluate the performance of the model on the testing set. To be distinct,

- Bagging creates the final forecast, divides the data into many random subgroups, trains a model on each subset, then averages their predictions.
- Boosting involves training a series of models, each one designed to fix the flaws of the one before it, then combining the results to arrive at the final forecast.
- Stacking involves training a variety of varied models, using the predictions they produce as input to a meta-model, and using the output of the meta-model as the final prediction.

3.4.2 Characteristics

Before choosing the right parameters, first, we need to know how the model performs on the dataset with its default value. After that, optimization hyperparameters techniques shall be used to determine the best suitable parameters for the model on the dataset. Then, we shall use cross-validation techniques to evaluate the performance of the model by dividing the dataset into training and test datasets multiple times. Usually, t**he parameters of the model with best performance on the validation set are regarded as the right parameters to train the model**. Apart from these techniques, it is important to note that the individual domain knowledge plays crucial parts in selecting the relatable parameters.

The exact dataset and problem being addressed can have an impact on how well an ensemble learning model performs. In a variety of scenarios, ensemble learning has generally been found to increase prediction accuracy relative to individual models. However, it's crucial to remember that no model is perfect and that **predicting upcoming cases will always involve some level of uncertainty.**

Therefore, careful testing and validation procedures should be used to assess how well an ensemble learning model predicts future cases. Sometimes, simpler models are more effective. To maximize performance, it's crucial to carefully select and adjust the model's hyperparameters.

3.4.3 Models Usage Comparison

In ensemble learning, models can be combined together in different ways to improve the overall performance of the system. The most common ways to use models together in ensemble learning are as mentioned above; bagging, boosting and stacking. Depending on the purpose of usage, users can choose which model to be used for the issue. Please refer to the below table for brief characteristics comparison on each model.

Features	**Bagging**	**Boosting**	**Stacking**
Purpose of usage	Reduce Variance	Reduce Bias	Improve Accuracy
Type of Base Learner	Homogeneous	Homogeneous	Heterogeneous (Different Types)
Base Learner Training Style	Parallel / Independent	Sequential	Meta-model | Parallel and/or Sequential
Aggregation	Averaging / Max-voting	Weighted-averaging	Weighted-averaging
Training Time	Typically faster than boosting	Typically slower than bagging, faster than stacking	Typically the slowest, due to stacking layers

Table 3.1: Comparison on each models (Kalirane)

It can also be used in hybrid forms such as Bagging + Boosting or Stacking + Bagging. (Ribeiro and Coelho) The choice of technique depends on the specific problem and the nature of the dataset.

4. Technical Report of Issue 3

This section aims to present all three ensemble learning methods listed above, i.e., bagging, boosting and stacking.

4.1 About Dataset and some mathematical expressions

The dataset used is taken from the WQU data from VM about Luxembourg index and other $MSCI$ indices. The strategy for this issue is to take a long position for anticipated daily returns of 2.5% on $LUXXX$. In addition to the $MSCI$ indices from other countries, we also add technical indicators like Simple Moving Average (SMA) ratio and Relative Strength Index (RSI) ratio to our dataset.

The equation for SMA is as follows:

$$SMA = (\text{sum of prices for } n \text{ periods}) / n$$

where: n is the number of periods, (sum of prices for n periods) is the sum of closing prices over the specified number of periods. In this issue, we use 15-SMA (15 days) and 5-SMA (5 days), dividing them to get required SMA ratio.

The equation behind for RSI is as follows:

$$RSI = 100 - (100 / (1 + RS))$$

where: RS is the average gain of the up periods divided by the average loss of the down periods over a given period of time. The formula for RS is:

$$RS = (Average\ Gain / Average\ Loss)$$

To calculate the average gain, the sum of all gains over a given period is divided by the number of periods. To calculate the average loss, the sum of all losses over a given period is divided by the number of periods. The formula for calculating the average gain and average loss is:

$$Average\ Gain = [(Previous\ Average\ Gain * (n-1)) + Current\ Gain] / n$$
$$Average\ Loss = [(Previous\ Average\ Loss * (n-1)) + Current\ Loss] / n$$

Where n is the number of periods over which the calculation is being performed. The first average gain and average loss are calculated using the simple moving average (SMA) of the gains and losses over the first n periods. After the first n periods, the average gain and average loss are calculated using the previous average gain and loss values.

In this issue, we use 5-RSI (5 days) and 15-RSI (15 days) to determine the another technical indicator RSI ratio.

Unlike other methods like DecisionTree and Linear Regression, there are no specific equations for these methods (Bagging, Boosting, Stacking) as they are implemented through various algorithms and techniques. Thus, we will describe the Python code used to build these models in the notebook.

4.2 Computation in Python

Before training the model, first of all, the following necessary libraries are imported in our python notebook.

```
`from sklearn.ensemble import (
      AdaBoostClassifier,
      GradientBoostingClassifier,
      RandomForestClassifier,
      StackingClassifier
)`
`from sklearn.linear_model import LogisticRegression`
`from sklearn.naive_bayes import GaussianNB`
`from sklearn.svm import SVC`
`from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor`
`import xgboost as xgb`
```

After importing the dataset into the Python notebook, we perform some basic EDA and transform the "Date" data from String to datetime. Instead of focusing on prices, we use daily returns. This is achieved by applying `` `pct_change()` `` from the pandas package. Then, we define the target column by regarding any return of greater than 2.5% to be 1 and otherwise 0. For the features indices, we use following countries indices; `` `"MSCI CHINA", "MSCI SINGAPORE", "MSCI RUSSIA", "MSCI DENMARK", "MSCI NORWAY"` ``. Next, we append the technical indicators, SMA ratio and RSI ratio to our dataset.

After dividing the dataset into training and testing sets, we train the model respectively using `` `GridSearchCV` `` to find the best parameters so that the model gets trained to get optimal value. Different models like Adaptive Boosting, Gradient Boosting, XG boosting, Bagging (Random Forest) and Stacking (Decision Tree + SVM + Gaussian Naive Bayes -> Logistic Regression) are trained by using the following commands: `` `"AdaboostClassifier", "GradientBoostingClassifier", "XGBClassifier", "RandomForestClassifier", "StackingClassifier"` ``.

After re-training the model with tuned parameters, these models are compared by using the `` `roc_auc_score` `` and `` `roc_curve` ``.

See attached Python Notebook for more information on the examples.

5. Marketing Characteristics

In this section, the computed results are mentioned in accordance with each individual report.

5.1 For Issue 1

After training the model with three different types of hyperparameter optimization methods, the generated results are as follows.

Generated Parameters	GRID SEARCH	RANDOM SEARCH	BAYES SEARCH
max_depth	5	5	4
min_samples_leaf	2	4	5
n_estimators	200	188	247
best_score	0.81012	0.80974	0.80951
Execution time (second)	89.35	53.84	184.42

Table 5.1: Comparing generated results of Hyperparameter Optimization

Based on the generated results, we can notice that the random search method is the fastest one among the three while the Bayesian Optimization method is the slowest. The best score generated is not very much different. Only the execution time is noticeably different.

We also use the generated parameters and try to calculate AUC score and plot the ROC curve to evaluate the model with different hyperparameter optimization methods. We use the Random Forest Classifier as our base model. The generated AUC scores are as follows:

```
No Skill: ROC AUC=0.50000
RFC with grid search: ROC AUC=0.82674
RFC with random search: ROC AUC=0.82751
RFC with Bayesian Optimization: ROC AUC=0.82674
```

The ROC curve plot is -

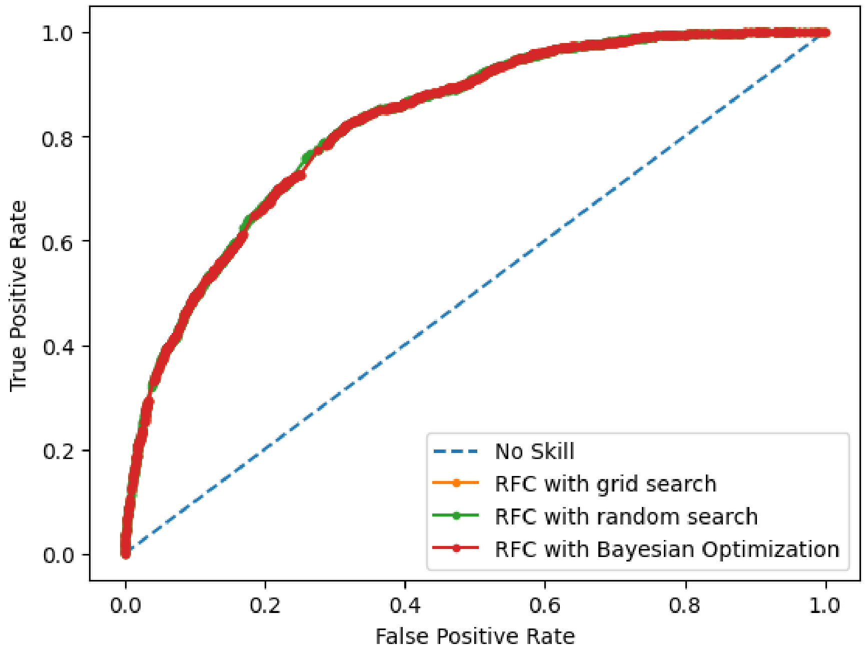

Fig 5.1: ROC curve plot for Issue 1 (Python Notebook)

Since we are using the same base model, the generated ROC curves seem almost identical to each other. We may find the only difference in AUC score.

5.2 For Issue 3

After transforming the imported dataset and ensuring that there is no null value in the data columns, we use a total of five models to train the dataset to work on the Ensemble Learning. For the boosting algorithms (Adaptive Boosting, Gradient Boosting and XG Boosting), we regard the date range of `n_estimators` as $[\ 10, 20, 50, 100\ ]$ and `learning_rate` as $[\ 0.1, 0.25, 0.5, 1.0\ ]$. After performing the Grid Search cross-validation technique on these boosting algorithm, the best parameters are generated as below:

Generated Parameters	AdaBoost	GradientBoosting	XGBoost
n_estimators	10	10	50
learning_rate	0.25	0.1	0.25

Table 5.2: Generated best parameters for Boosting algorithms

Based on the results, we can notice that even though they are of the same type of algorithm, the best parameters generated can be different.

For the bagging model (RandomForestClassifier), after performing the Grid Search method, the best parameter generated are `max_depth` as 5, `min_samples_split` as 4, `n_estimators` as 10, and `random_state` as 0. As for the stacking model, we do not perform any hyperparameter tuning. We just took the default parameter value to train the model. After that, we calculated AUC score and plotted the ROC curve to compare these five models. The ROC curve plot is as below.

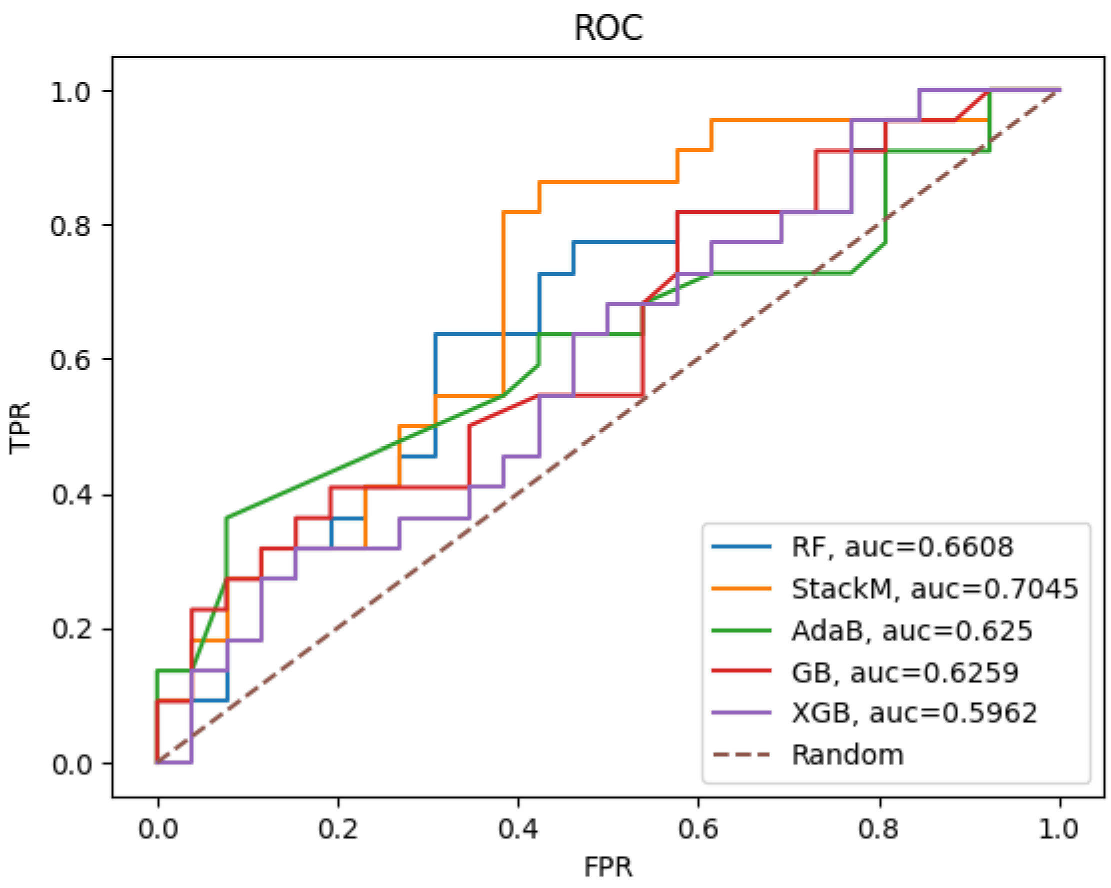

Fig 5.2: ROC Curve plot for Issue 3 (Python Notebook)

Based on the AUC score generated, we can find the stacking model performs very well compared to other models. Because it combines the results of various models and trains a new model using these results, the stacking model's AUC score can be higher than that of other models. This new model might be able to include the advantages of the different models while minimizing their disadvantages, resulting in enhanced performance. As the base models are trained on various subsets of the data and their outputs are combined using a meta-learner, stacking can also lessen the effects of overfitting and improve generalization to new data. Therefore, the stacking model can achieve higher accuracy and AUC score compared to the individual models if the base models are well-trained and diverse and the meta-learner is effective in combining their outputs. However, it is crucial to note that the success of the stacking model depends on a number of variables, including the standard of the base models, their diversity, and the choice of the meta-learner.

6. Conclusion

To summarize, Ensemble learning has shown enormous promise in a variety of real-world applications, including fraud detection, market prediction, and healthcare diagnostics, and may increase the accuracy, stability, and resilience of the final prediction by merging different models, which can lead to better decision-making and resource allocation. Besides, Ensemble learning is predicted to play an increasingly crucial role in tackling difficult issues in a variety of sectors in the future, including finance, marketing, and cybersecurity. With the emergence of big data, deep learning, and cloud computing, ensemble learning may take use of these advancements to attain even greater performance and scalability. Yet, there are some difficulties with ensemble learning, such as model selection, parameter tuning, and model interpretability. As a result, thoroughly designing and evaluating ensemble models is critical to ensuring their efficacy and efficiency in practice.

Cited Journals and Learn More

Below are the cited references and journal articles that are referenced in this handbook in MLA format.

1. Brownlee, Jason. “A Gentle Introduction to Ensemble Learning Algorithms - MachineLearningMastery.com.” *Machine Learning Mastery*, 19 April 2021, https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/. Accessed 2 May 2023.
2. Brownlee, Jason. “What Is Argmax in Machine Learning? - MachineLearningMastery.com.” *Machine Learning Mastery*, 3 April 2020, https://machinelearningmastery.com/argmax-in-machine-learning/. Accessed 2 May 2023.
3. Demir, Necati. “Ensemble Methods in Machine Learning.” *Toptal*, https://www.toptal.com/machine-learning/ensemble-methods-machine-learning. Accessed 2 May 2023.
4. Kalirane, Mbali. “Ensemble Learning Methods: Bagging, Boosting and Stacking.” *Analytics Vidhya*, 20 January 2023, https://www.analyticsvidhya.com/blog/2023/01/ensemble-learning-methods-bagging-boosting-and-stacking/. Accessed 2 May 2023.
5. Ludmila, Shumilov. “Why Do We Need a Validation Set in Addition to Training and Test Sets?” *Towards Data Science*, 11 April 2022, https://towardsdatascience.com/why-do-we-need-a-validation-set-in-addition-to-training-and-test-sets-5cf4a65550e0. Accessed 2 May 2023.
6. Maclin, R., and D. Opitz. “[1106.0257] Popular Ensemble Methods: An Empirical Study.” *arXiv*, 1 June 2011, https://arxiv.org/abs/1106.0257. Accessed 2 May 2023.
7. Ribeiro, Matheus Henrique Dal Molin, and Leandro dos Santos Coelho. “Ensemble approach based on bagging, boosting and stacking for short-term prediction in agribusiness time series.” *Applied Soft Computing*, ScienceDirect, 29 July 2022,

https://www.sciencedirect.com/science/article/abs/pii/S1568494619306180. Accessed 2 May 2023.

8. Sarkar, Priyankur. "An Intro to Ensemble Learning in Machine Learning | by Priyankur Sarkar." *Medium*, 16 August 2018, https://medium.com/@priyankur.sarkar/an-intro-to-ensemble-learning-in-machine-learning-5ed8792af72d. Accessed 2 May 2023.
9. Tannor, Philip, and Shlomo Tannor. "When You Shouldn't Use Ensemble Learning." *Deepchecks*, 5 May 2021, https://deepchecks.com/when-you-shouldnt-use-ensemble-learning/. Accessed 2 May 2023.
10. Yang, Li, and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice." *Neurocomputing*, ScienceDirect, 29 July 2022, https://www.sciencedirect.com/science/article/abs/pii/S0925231220311693. Accessed 2 May 2023.