FULL LEGAL NAME	LOCATION (COUNTRY)	EMAIL ADDRESS	MARK X FOR ANY NON-CONTRIBUTING MEMBER
Alejandro Piedra Alvarado	Costa Rica	1alepiedra7@gmail.com	
Peter Esekhaigbe	Nigeria	petersonese304@gmail.com	
Jude Chukwuebuka Ugwuoke	United States	jcu0005@auburn.edu	

Statement of integrity: By typing the names of all group members in the text boxes below, you confirm that the assignment submitted is original work produced by the group (excluding any non-contributing members identified with an "X" above).

Team member 1	**Alejandro Piedra Alvarado**
Team member 2	**Peter Eronmosele Esekhaigbe**
Team member 3	**Jude Chukwuebuka Ugwuoke**

Scenario

Your group has already presented a half-dozen ideas. The strategists came back with three big questions:

1. How do we know the models have the right parameters?
2. How well can the models be expected to work for predicting future cases?
3. How can the models be used together?

Responding to these tasks will surely give the strategists confidence in your ability to use these models profitably. Let's consider each of these questions as issues.

The first question asks, "How do we know the models have the right parameters?" This question requires that we know how to optimize hyperparameters. This means we'll have to explain which hyperparameters are used, how we optimize them, and how we know they are optimal. The second question asks, "How well can the models be expected to work for predicting future cases?" This question requires that our models work well both within-sample—on training data—as well as out-of-sample—on testing data. Bias measures the accuracy on training data, and variance measures the accuracy on testing data. As such, this question is motivating us to discuss the process of optimizing the bias-variance tradeoff. The third question asks, "How can the models be used together?" This question requires us to think about the various ways models can be combined. Some methods exist that combine the same type of model—homogeneous ensemble methods. Other methods exist that allow us to combine different types of models—heterogeneous ensemble methods. We want to show that, when combined, multiple models can sometimes outperform individual models. This will motivate us to show how models can be used together.

So, in summary, the three topics to be addressed are:

- Issue 1: Optimizing Hyperparameters
- Issue 2: Optimizing the Bias-Variance Tradeoff
- Issue 3: Applying Ensemble Learning—Bagging, Boosting, or Stacking

Each topic will have two sections: a technical and a non-technical section.

Step 1

As a group, all three members read through the feedback from GWP2. All group members will incorporate the suggested corrections on the three items and work together to improve the questions submitted from GWP2.

Instructor Review for GWP2

Quantitative Analysis (open-ended questions).

- Explain the terms in the equations used.

Technical and Non-technical Reports

- The report just clusters all plots together without really telling a story. Insert visuals that support a point/s you want the reader to takeaway.

Step 2

As a group, all three members plan a strategy as to what goes in the technical and nontechnical sections of each question. The deliverable from this section will be an outline.

- **Issue 1: Optimizing Hyperparameters**
    - **Technical Section:**

I. Introduction to Hyperparameter Optimization
- Definition of Terminologies.
- Classification of Hyperparameter Optimisation Methodologies
- Procedure for Optimisation of Hyperparameters
- Tuning Performance Metrics.

II. Model-Based Hyperparameter Optimization
- Model Definition
- Hyperparameters Specifications and definitions
- Model Development (Mathematical Representations—if any)
- Parameter Tuning Strategy.
- Evaluation of Tuned Model Performance (Illustrations)
- Comparison of original model and tuned model performance.

III. Challenges of Hyperparameter Tuning

- **Non-technical**

  I. Challenges of Suboptimal Model
  II. Goal of Hyperparameter Optimization.
  III. Application of Hyperparameter Tuning.

- **Issue 2: Optimizing the Bias-Variance Tradeoff**

  - **Technical**

Introduction to the Bias-Variance Tradeoff (Technical)
1.1 Definition
1.2 Classification
1.3 Consequences of Imbalance

Mathematical Formulation
2.1 Expected Test Error
2.2 Bias, Variance, and Irreducible Error
2.3 Reference Equations

Approaches to Controlling Bias and Variance
3.1 Regularization and Cross-Validation
3.2 Pruning in Decision Trees
3.3 Ensemble Methods
3.4 Feature Selection and Dimensionality Reduction
3.5 Early Stopping

Performance Evaluation
4.1 Regression Metrics
4.2 Classification Metrics
4.3 Cross-Validation Schemes
4.4 Data Preprocessing & Feature Engineering
4.5 Hyperparameter Tuning
4.6 Computational Considerations
4.7 Backtesting in Finance
4.8 Risk of Overfitting

Financial Applications
5.1 Market Forecasting
5.2 Portfolio Allocation
5.3 Credit Risk
5.4 Algorithmic Trading

- **Non-technical**
  - Understanding the problem
  - Objective of the tradeoff

- **Issue 3: Applying Ensemble Learning—Bagging, Boosting, or Stacking**

  - **Technical**

I. Introduction to Ensemble Learning (Technical)
   a) Definition.
   b) Classification.
   c) Advantages.

II. Bagging (Bootstrap Aggregating)
   a) Technical Description.
   b) Algorithm.
   c) Mathematical Formulation.
   d) Python Code.

III. Boosting
   a) Technical Description.
   b) Algorithm.
   c) Mathematical Formulation.
   d) Python Code:

IV. Stacking
   a) Technical Description.
   b) Algorithm.
   c) Mathematical Formulation.
   d) Python Code.

V. Performance Evaluation
   a) Regression.
   b) Classification.
   c) Cross-validation.

VI. Data Preprocessing & Feature Selection: Critical for maximizing model performance.
   a) Hyperparameter Tuning.
   b) Computational Cost.
   c) Backtesting.
   d) Overfitting Risk.

VII. Financial Applications.
   a) Portfolio Optimization.
   b) Credit Risk Modeling.
   c) Algorithmic Trading.
   d) Fraud Detection.

- **Non-technical**

I. Introduction to Ensemble Learning.
II. Key Benefits.
    a) Improved Prediction Accuracy and Portfolio Optimization.
    b) Robustness and Risk Reduction.
    c) Handling Complex Data and Market Adaptability.
    d) Applications in Algorithmic Trading and Credit Risk.
III. Bagging: Reducing Variability.
IV. Boosting: Improving Accuracy.
V. Stacking: Combining Strengths.
VI. Implementation Considerations.
    a) Data Preparation.
    b) Hyperparameter Tuning.
    c) Computational Trade-offs.
    d) Backtesting and Risk Management.
VII. Financial Applications.
    a) Portfolio Optimization.
    b) Credit Risk Modeling.
    c) Algorithmic Trading.
    d) Fraud Detection.
VIII. Conclusion.

Step 3

Issue 1: Optimizing Hyperparameters. Peter Esekhaigbe

Technical:

INTRODUCTION TO HYPERPARAMETER OPTIMIZATION

Hyperparameter tuning is a significant part of machine learning models. It helps to maximise the efficiency of the model and can significantly improve its accuracy. There are quite a number of methods that can be used to optimize hyperparameters of the various models that have been discussed in previous projects. Since hyperparameter optimization is model-based, we will carefully address this issue by using 3 different categories of models developed in previous projects (GWP1 and GWP2): Unsupervised learning models, supervised learning models, and Neural Networks.

Definition: Hyperparameter tuning is essentially a mathematical concept introduced introduced into the training process of a machine learning model. The model development process often requires that specific parameters are indicated before the training process can be completed and before model fitting can take place. While the modeller can deem it fit to define specific parameter arguments based on the direction of the model training process, it is important to explore a range of parameter options before making a decision on the parameter to select.

Hyperparameters are simply parameters that can influence or control the model's behaviour and performance in the learning process. They vary based on the model or algorithm in question; however, similar methodologies an strategies can be harnessed in the optimisation of these hyperparameters. Some of the common methodologies are namely:

Hyperparameter Optimization Methodologies: There are quite a number of methods for optimising hyperparameters, namely:

- **Cross-validation:** This technique simply splits the datasets and model training process into small *k*-segments often called *folds,* so as to improve the performance of the models. For example, by simply cross validating a ordinary Lasso Regression model, we can improve the accuracy score of the model significantly and consequently enhance the generalization of the model on unseen (test) data. While cross validation is already integrated into the hyperparameter tuning process for some models, (e.g *LassoCV* - which optimizes parameters in a specified search space), the *cross_validate(**model**, arguments)* is simpe way to cross validate the training process of any model.
- **Grid & Randomized Search:** Both methods practically follow the same training rules. The Grid Search algorithm implements a complete search in a grid of all the predefined hyperparameter values within a search space and select the best parameters for model fitting. Randomised Search method also does the same thing; however, it is applied to models with large search space but very few influential hyperparameter values. Hence the Randomised search selects a random subset of hyperparameter values from search space and finds the best parameter solutions. The Randomised Search algorithm is more efficient than the Grid Search - which is computationally costly.
- **Bayesian Optimization:** This technique adopts a Gaussian-based process for selection of hyperparameters. It utilises probabilistic models and objective functions in exploring and exploiting the search space in the bid to find optimal parameter solutions. It often uses libraries such *optuna, Hyperopt, BayesianOptimzation* in the learning process. The Bayesian Optimization technique is fast and more efficient method however, it could be complex to implement.
- **Grid Search (or Randomised Search) + Cross Validation:** These techniques are often indicated as *GridSearchCV or RandomizedCV* in Python libraries. Just like the *LassoCV* discussed earlier, these methods integrate cross-validation into the learning process to further optimise model parameters.
- **ML Pipelines:** This technique simply integrates two or more machine learning algorithms (e.g., an unsupervised learning like *Principal Compoent Analysis* into a supervised learning task such as *Lasso Regression*) to improve the performance of the models. While some stand-alone supervised learning models perform well, preceding unsupervised learning models and feature scaling and engineering may simplify the actual model implementation but may not necessarily improve performance. An ML pipeline is a combination of all of the aforementioned tasks.

Procedure for Hyperparameter Optimization**.**

Hyperparameter optimisation is implemented through the following steps:

1. The first step to optimising hyperparameters is to identify the model in question: Is it a supervised machine learning model (classifiers, regressors), an unsupervised learning model, or a neural network?
2. Identify all of the relevant hyperparameters associated with the model in question.
3. Specify a search range for the parameters for tuning
4. For a supervised model, apply an optimisation algorithm search (e.g., GridSearch, K-fold cross-validation) to determine the best parameters for the models. For an unsupervised model, implement a manual search function on the search range.
5. Apply the best hyperparameters in the modelling operations.
6. Evaluate the model performance by using metrics such as Mean-Squared Error, R-squared value, and Classification report, depending on the kind of model, to measures its accuracy

Tuning Performance Metrics.

Metrics used to evaluate the performance of the models are specific to the model task:

- **Regression Metrics:** Since we typically use regression models for prediction, a more efficient way to evaluate the model is to measure the error (difference) between the predicted value (*y_pred*) , and the actual target value (*y_test*). Interestingly, there are a number of methods to achieve this. The manner adopted for a particular model is dependent on the choice of the modeler. However, it is important to note that the accuracy and performance of two regression models can only be compared based on the same metric.Some of the most common regression metrics:
    1. **Mean Squared Error:** As the name implies, this is a measure of the mean of the squared errors between the predicted and actual (test) value. This error penalizes large errors. Mathematically, it can be represented as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{pred} - y_{test})^2$$

$where$:

$$MSE = Mean\ Squared\ Error$$
$$y_{test} = Test\ data$$
$$y_{pred} = Predicted\ y\ data$$

    2. **Root Mean Squared Error:** As the name implies, this is the root of the mean of the squared errors between the predicted and actual (test) value. Similar to MSE. Mathematically, it can be represented as:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{test} - y_{pred})^2}$$

$where$:

$$RMSE = Root\ Mean\ Squared\ Error$$

3. **Mean Absolute Error:** This technique takes the absolute error from the difference between the predicted value and test data. Mathematically, it can be represented as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_{test} - y_{pred}|$$

$where$:

$$MAE = Mean\ Absolute\ Error$$

Note: *The performance of a model is measured by how much these errors are minimised. Hence, the lower the error value, the better the model performance and accuracy.*

4. **R-squared:** This statistical metric ranges from 0 to 1 and measures how well a regression model explains variance. Mathematically, it is represented as the ratio of the residual sum of squares (squared-error) of the model to the total sum of squares (squared-error) of the mean model.

$$R^2 = \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^{n}(y_{test} - y_{pred})^2}{\sum_{i=1}^{n}(y_{test} - \bar{y})^2}$$

$where$:

$$SS_{res} = Residual\ sum\ of\ squares$$
$$SS_{tot} = total\ sum\ of\ squares$$
$$\bar{y} = mean\ of\ the\ test\ data$$

R^2 evaluates the goodness of fit of the model. The higher the R^2 value, the higher the model performance and goodness of fit.

- **Classification Metrics:** Classification models are mainly used to categorise dataset into multiple classes. The most predominant challenge with the performance of these models lies in their ability to offer precision and accuracy during the classification of data points. To measure the accuracy of the models, there are a number of metrics that are used. This report contains a number of relevant metrics such as Precision, Recall, F1 score, and accuracy score.

1. **Precision:** This metric measures the proportion of predictions that are accurate**.** SImply put, the ratio of correction predictions to total predictions. In actual mathematical learning calculations, it is the ratio of the number of true predicted positives to the total number of predicted positives. Mathematically,

$$Precision = \frac{TP}{TP + FP}$$

$where$:

$TP = True\ Positives\ (number\ cases\ predicted\ positive\ \ and\ actually\ positive)$

$FP = number\ of\ cases\ predicted\ positive\ but\ actually\ negative$

High model precision implies *few false positive cases*. The goal is to achieve high precision value close enough to 1. This metric is applied in detecting the cost of having a false positive (e.g detecting a email spam).

2. **Recall:** This metric measures the actual number of true positives correctly predicted by the model. Mathematically, it is represented as:

$$Recall = \frac{TP}{TP + FN}$$

$where$:

$TP = True\ Positives\ (number\ cases\ predicted\ positive\ \ and\ actually\ positive)$

$FN = number\ of\ cases\ predicted\ negative\ but\ actually\ positive$

High model recall implies *few false negative cases*. The goal is to minimize the number of false negatives. In essence, this metric assesses the cost of missing a positive.

3. **F1 Score:** This metric combines the recall and precision in the same formula. Mathematically, it is represented as:

$$F1 - Score = 2 * \frac{Precision*Recall}{Precision + Recall}$$

A high F1-score indicates a good balance between precision and recall.

4. **Accuracy Score:** This measures the ratio of correct predictions to total number of predictions. Mathematically,

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predicitions}$$

5. **Other metrics:** All of the metrics discussed overleaf are mainly particular to supervised learning models. Unsupervised learning models also utilise metrics such as accuracy score in the evaluation of model performance. Some of these include:
   - Cross-validation score: Evaluate the improved performance of the model to which is applied to
   - Early stopping performance score: To evaluate the capacity of the model to stop early once overfitting is detected
   - Model training time speaks to the complexity of the model.
   - Visual plots: Plots comparing the error of training and validation error of models; scree plot which show explained variance; and heatmaps to show the relationship between relevant parameters. AUC-ROC curves, etc.

MODEL-BASED HYPERPARAMETER OPTIMIZATION

Hyperparameters are specific to kind of model applied to a dataset; hence, parameter optimisation can only be implemented based on the kind of parameters associated with the model.

In the Python notebook, we tried to optimise the parameters of the models implemented in the previous two projects (GWP1 and GWP2), and so we will explain how the parameter optimization process is done based on the models run in those processes. Simply put, we will try to optimise the right

parameters in these models, show how to measure the optimality of these parameters, and consequently improve model performance. Same data sources used for running in the initial projects were also used for the hyperparameter tuning; same data preprocessing conditions were maintained.

For this section, we will classify the following models into 3 categories: *Unsupervised Learning models, Supervised Learning models, and Neural Networks*

A. Unsupervised Learning:

1. Hierarchical Clustering:

Hierarchical clustering is an unsupervised learning model which is used to group similar data points to form tree clusters (agglomerative hierarchical clustering) or split one cluster into partitions of smaller clusters (divisive clustering).

One key step in hierarchical clustering is to define the threshold distance between data points in the formation of clusters. Hence there are common distance measures used in hierarchical clustering. Some of which are:

- **Euclidean distance:** This distance measures the sum of squared distance between variables in the datasets. Euclidean distance metric measures the as-the-crow flies (hypotenuse) distance between to data variables. Mathematically, it is represented as:

$$E = \sqrt{\sum_{j=1}^{n} (x_j - y_j)^2}$$

$where$:
$E = Euclidean\ Distance$
$X(x_{1,}, x_2 \dots x_n) = X\ variable$
$Y(y_{1,}, y_2 \dots y_n) = Y\ variable$

- **Manhattan distance:** This distance metric measures the distance travelled from one data point to another on a grid-like path. Mathematically, it is represented as:

$$M = \sum_{j=1}^{n} |x_j - y_j|$$

$where$:
$M = Manhattan\ Distance$
$X(x_{1,}, x_2 \dots x_n) = X\ variable$
$Y(y_{1,}, y_2 \dots y_n) = Y\ variable$

Relevant Hyperparameters.

There are 3 main hyperparameters that can be tuned in a hierarchical clustering model (say, an agglomerative clustering problem); these parameters are namely:

- n_clusters: number of clusters

- linkage method: which measures how to compute distance between clusters. Linkage methods could b*e ward, complete, average, single, etc.*
  - The single linkage method defines distance between two clusters as the minimum distance between any pair of points from each cluster.

$$D(A,B) = min\, d(a,\ b)\ where\ a,\ b \in A,\ B\ respectively$$

  - The complete linkage method defines distance between two clusters as the longest distance between any pair of points from each cluster.

$$D(A,B) = max\, d(a,\ b)\ where\ a,\ b \in A,\ B\ respectively$$

  - The ward linkage method defines the distance between two clusters are the minimum increase in within-cluster variance after linking.

$$\Delta E = \sum_{x \epsilon A \cup B} ||x - \mu_{A \cup B}||^2 - (\sum_{x \epsilon A} ||x - \mu_A||^2 + \sum_{x \epsilon B} ||x - \mu_B||^2$$

- distance metric: euclidean, manhattan etc.

Model Development & Parameter Tuning Strategy.

a. Manual grid search was implemented to determine the best hyperparameters (linkage method, n_clusters, and distance metric) for the model development.
   - N_cluster search range: between 2 and 10 clusters
   - Linkage method search list: ward, complete and single
   - Distance metric search space: Euclidean and Manhattan
b. Created a for loop to determine a suitable linkage method for a specific distance metric.
c. Run a code assess the model performance using the ***Silhouette Score*** (ranging between 0 and 1)
d. Develop the model using *AgglomerativeClustering* Library as opposed to *fcluste*r library used in the initial model (in GWP 1), simply because the former is a faster model for clustering.

Model Evaluation.

Upon running the algorithm, the following e best hyperparameters were obtained:

- N_clusters: 2
- Linkage method: complete
- Distance metric: Euclidean.
- Silhouette score: 70% (indicative of excellent clustering).

Model Comparison.

There is some sort of disparity in the type of parameters that were used in the original model (run in GWP1). In the previous project the modeler specified and ran the clustering algorithm the *ward linkage_method.* However, our results here show that the ***complete*** *linkage_method* is the right parameter for the model implementation.

The Silhouette score is a measure of how well data points fit within a cluster and the distinction between separate clusters. Mathematically, the silhouette score of a single point is:

$$s(i) = \frac{b(i) - a(i)}{max\ (a_i, b_i)}$$

$$where:$$
$$s(i) = Silhouette\ Score\ of\ point\ i$$

$$a(i) = average\ distance\ of\ point\ i\ to\ all\ other\ points\ in\ the\ same\ cluster$$
$$b(i) = average\ distance\ of\ point\ i\ to\ all\ other\ points\ in\ the\ nearest\ cluster$$

2. Principal Component Analysis (PCA):

Principal Component Analysis is the dimensionality reduction algorithm that reduces datasets into components that preserve the valuable information of the overall dataset.

Relevant Hyperparameters: The main hyperparameter in the PCA algorithm is the number of components (*n_components)* obtained after mode implementation.

Model Development and Parameter Tuning Strategy.
In the Jupyter Notebook, three approaches were adopted for hyperparameter tuning:

- Specify Standard Explained variance: This helps tailor the model to only work with optimal number of components required for dimensionality reduction. A standard explained variance for a principal component should be greater than or equal to 95%.
- Implement a scree plot to visualise number of components with explained variance greater than a threshold value (preferably 95%)
- Run a grid search, randomised search, or search specific to the model in question to determine the optimal number of components. Used when PCA is integrated in a pipeline with supervised learning tasks. For this case, we'll make use of LassoCV (as this was one of the models run in GWP 1).

Model Evaluation.

- **Running the PCA algorithm using the first approach:** number of components (n_components: 2).
- **Scree plot:** The scree plot also shows the visual representation of number of components obtained from the model.

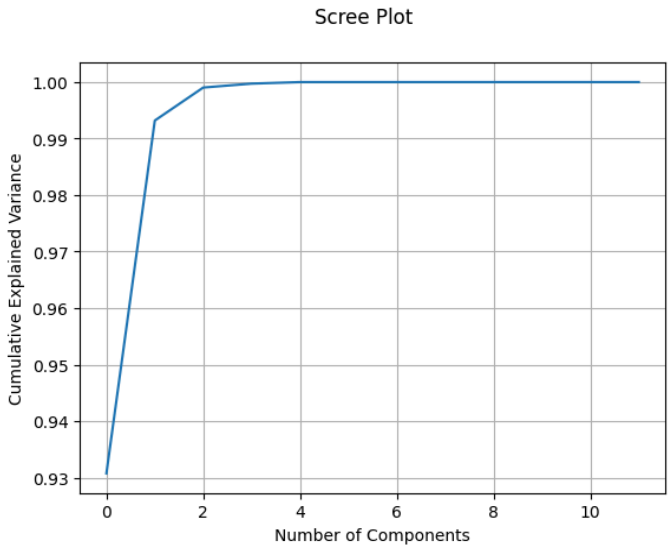

Fig 1. Scree Plot

- **ML Pipeline (PCA + LassoCV):** For this approach, it was noticed that running PCA as preceding algorithm in a supervised learning task like Lasso Regression with cross validation somewhat reduced the model performance of the single Lasso Regression model. The main hyperparameter in the LassoCV algorithm is the *alpha,* which is the regularization parameter that penalises overfitting and underfitting the Lasso Regression model.

  On comparing the single LassoCV (Lasso Regression with cross validation) model in the GWP2 and our PCA + LassoCV implementation, our results are as follows:
  - R-squared: 93% (which is less than the RMSE in single LassoCV)
  - RMSE: 0.003 (good but larger than what was obtained in the previous project)
  - Optimal Alpha: 1 * 10^-6 (similar to what was obtained in the previous project)

  In conclusion, an unsupervised learning model may not improve the performance of a well-optimised supervised learning model.

B. Supervised Learning Models:

1. Linear Discriminant Analysis (LDA):

Linear Discriminant Analysis is a supervised learning model that classifies data points and helps with dimensionality reduction of the dataset.

Relevant Hyperparameters.
There are 2 main hyperparameters associated with Linear Discriminant Analysis:

- **Shrinkage:** this is the regularization parameter for the covariance matrix. It takes either "*auto" (to automate the regularization process)* or the search space range of *float (0.1, 0.5, and 0.9 used in the model implementation).*
- ***Solver:*** identifies the kind of transformation approach applied on the covariance matrix; It could take either of the following:
  I. *Lsqr* (Least Squares Iterative Algorithm): this uses least squares to solve complex linear system problems
  II. *Eigen* (Eigenvalue decomposition): this controls the model performance using eigen decomposition.
  III. *Svd (*Single Value decompostion): applies to only well-organised data.
- **N_components:** number of components obtained after dimensionality reduction can be optimised if the decision boundary (number of splits) is not defined.

Model Development and Parameter Tuning Strategy.
In the Jupyter notebook, we optimised each of the aforementioned hyperparameters (except for n_components, which was specified as 2 via a split_date) by:

- Run a grid search with validation (GridSearchCV) on a parameter grid that contains:
  1. Solver search list: (*svd, lsqr, eigen)*
  2. *Shrinkage* search list: (*auto, (0.1, 0.5, 0.9)*

- Fit the validation model to the training dataset and print the best parameters.
- Fit the LDA (*LinearDiscriminantAnalysis library)* model with the best parameters and evaluate model performance
- Evaluate the model performance using a confusion matrix and a classification report.

Model Evaluation. The best parameters obtained are as follows:

- Shrinkage: *auto (similar to previous model results)*
- Solver: *eigen (similar to previous model results)*

Classification report and confusion matrix results are the same as those of the previous model implementation in (GWP2), and therefore there is no need for further optimization.

2. Support Vector Machines:

Support Vector Machine is a supervised learning model used for classification and regression tasks. For the case study, we will only consider Support Vector Classifiers.

Relevant Hyperparameters. Main hyperparameters that can be tuned in SVMs modelling:

- C: Regularization parameter
- kernel type: 'linear" "poly" , "rbf", "sigmoid," etc.
- gamma: kernel coefficients for non-linear kernel functions
- epsilon: measures margin tolerance of the decision boundary and support vectors
- degree: controls the "poly" kernel function degree.

Model Development & Parameter Tuning Strategy.

- Implement the usual data preprocessing and train-test split on the data.
- Define the parameter search space under *param_grid.* In our model choose to tune only hyperparameters to limit training time of the model
    a. C search list: (0.01, 0.1, 1, 10, 100)
    b. Kernel search list: ('linear', "poly" , "rbf", "sigmoid,")
- Run a Grid Search with cross validation algorithm on the parameter grid; specify the number of cross validation steps (*cv=5)*
- Fit model of the grid search on the training data and print best parameters and estimator
- Further analyse the cross-validation results: get the score and identify relevant parameters from the output dataframe.
- Visualise the relationship between best parameters on the heatmap.
- Develop model with best parameters
- Fit tuned-model on the train set.
- Make predictions on the test set
- Evaluate model performance

Model Evaluation.

a. Best parameters obtained:
   C: 0.1, kernel: "poly"

b. Heatmap:

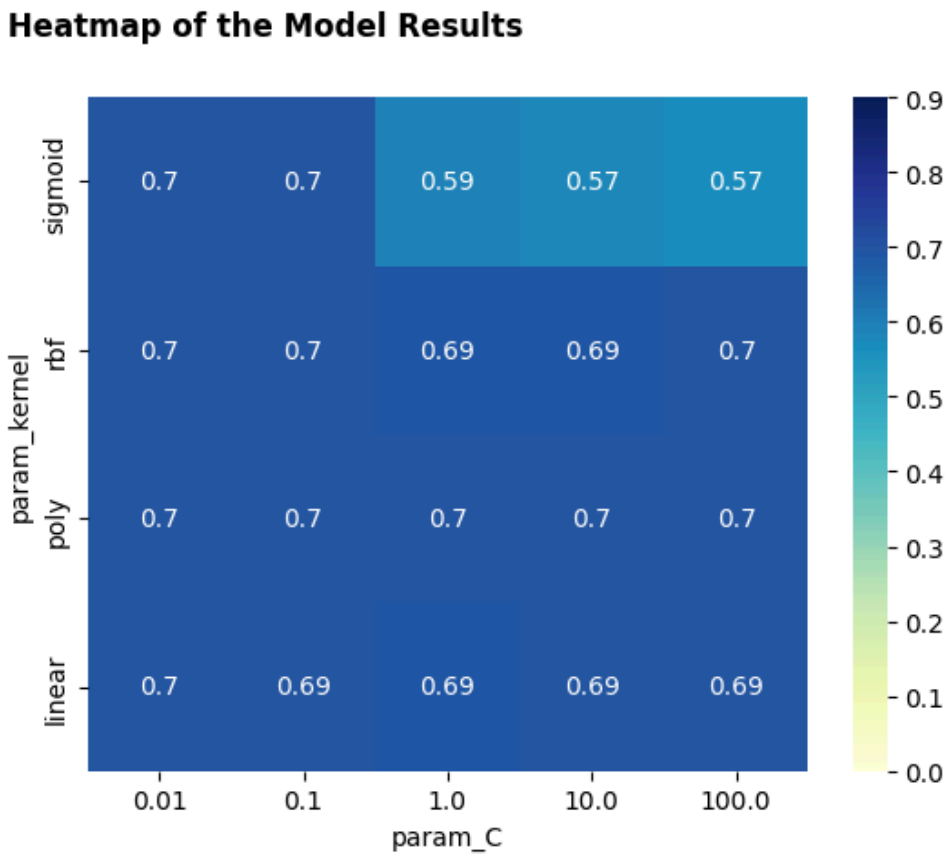

Fig 2. Heatmap

- Accuracy score of the tuned model: 65%.

  Despite including the kernel "sigmoid" function in this new classification as opposed to the three kernel functions used in the previous project (GWP2), we can deduce the results for models are the same and the parameters are optimal. However, implementing additional data preprocessing (or feature scaling) steps may improve model performance.

  Secondly, we can deduce that the grid-search-validation model training time increases when the categories of parameters in the grid search space increase—we isolated *the "gamma" parameter to limit training time.*

C. Neural Networks:

A neural network is a machine learning model inspired by the structure and function of the human brain. It's made up of layers of nodes ("neurons"), where each node is connected to others and processes information using weights, biases, and activation functions. For this case study we will be optimising hyperparameters of the multi-layer perceptron used in the previous project.

- Multilayer Perceptron: This neural network takes a certain number of inputs into a multi-hidden layer for processing an produces an output. It is used to perform classification tasks

Relevant Hyperparameters.

Some of the most common hyperparameters of MLPClassifier are as follows:

- Solver: specifies the optimizers that to be used. *"Adam (Adpative Moment Estimation)"* is the most commonly used optimizer for training neural networks, however, there are others such as
    I. *rmsprop (Root mean square propagation):* adjust the learning rate for each parameter individually based on the size of the gradient descent.
    II. *lbfgs (Limited-memory Broyden–Fletcher–Goldfarb–Shanno):* a quasi-Newton algorithm optimizer for minimizing loss functions.
    III. *sgd (Stochastic gradient descent):* useful for optimizing the algorithm stp by step using a random subset of data.
- Activation: defines the activation function for the model
- batch_size : defines the number of mini batches for stochastic optimization
- Learning rate: rate of gradient-based optimisation. It is often constant for MLPClassifiers etc.

For this case study, we will only consider the first 3 hyperparameters for tuning.

Model Development and Parameter Tuning Strategy.

- Implement the usual data preprocessing and train-test split on the data.
- Define the parameter search space under *param_grid.* In our model choose to tune only hyperparameters to limit training time of the model
    a. *solver* list: (*'adam', 'rmsprop', 'sgd', 'lbfgs'*)
    b. *Activation* search list: ('relu', "tanh" , "softmax," "sigmoid,")
- Run a Grid Search with cross validation algorithm on the parameter grid; specify the number of cross validation steps (*cv=5)*
- Fit a model of the randomized search on the training data and print the best parameters and estimator. You can proceed to confirm results using a grid search
- Fit tuned-model on the train set.
- Make predictions on the test set
- Evaluate tuned model performance.
- Compare original and tuned model performance

Model Evaluation.

To evaluate the performance of the tuned model, we developed a confusion matrix, classification report and accuracy score of the model and then proceeded to compare results with the original models:

Results obtained are:

Best parameters:

- Activation: *"tanh" (as opposed to "relu" activation function applied in the previous project)*
- Solver: *"sgd" (as opposed to "adam" used in the previous project)*

Confusion matrix:

- Number of *true positives i*ncreased from 4 (in the original model) to 5 (in the tuned model)

Classification report:

- Accuracy score remained the same.
- The F1 score increased slightly.

Challenges of Hyperparameter Tuning:

1. Computational Efficiency: Hyperparameter tuning often times complicates the model training process and delays model implementation time. For example, tuning a neural network with parameters such as learning rate, batch size, and activation functions, yet with many hidden layers, could be computationally expensive.
2. Scalability: Automating machine learning pipelines - with a number of models - manually may not scale for large datasets. In fact, the output of individual models in the pipeline may impact the overall accuracy performance of the pipeline algorithm (as shown in our pipeline implementation of PCA and Lasso Regression with cross validation).
3. Parameter and Metrics Selection : Selection of the right parameters and metrics to evaluate the performance of the models after tuning can be a hassle sometimes. A model may appear well-tuned but yet fail in real-world applications.

Non-Technical**:**

Challenges of Suboptimal Models.

1. Poor Generalization: a model may fit and perform well on a training set but fail to generalize unseen test data appropriately. This is indicative of a low accuracy score and high error. Suboptimality of a model is often a result of poorly defined parameters. Optimization of hyperparameters will solve this challenge and improve the performance of the model
2. Significant financial losses: Financial institutions could incur significant losses just by simply missing a true positive and maximising the number of false positives and negatives in the confusion matrix obtained from algorithms. Hence, there is a significant need to consider all possible alternatives in *the parameter search space* and also ensure that they consider only the right parameters before training the model and testing it on unseen data.
3. Black box models: Improper hyperparameter tuning may result in interpretability issues for the analyst. A bad model is simply a black box model, which makes financial justification difficult

Goal of Hyperparameter tuning.

The essence of hyperparameter tuning is to present the right optimal parameters to a model for training and testing. This effectively improves the model's accuracy score, reduces overfitting, and improves generalisation.

Application of Hyperparameter Tuning in Finance.

1. Algorithmic trading: Fine-tuning of the hyperparameters improves model forecast ability in trading. In addition, traders could benefit from new trading strategies, minimise losses, and minimise transaction costs. Fine tuning models like Support Vector and Random Classifiers could come in handy for traders

2. **Credit Scoring Assessments**: Models such as decision trees and SVM regressors are used to predict predict whether a borrower will default on a loan. By optimizing hyperparameters (such as tree depth, or regularization strength), these models can more accurately identify high-risk borrowers and reduce defaults. Tuning hyperparameters could also enhance precision in risk assessment and loan approval decisions.
3. Customer Segmentation: Models like hierarchical clustering are used by financial institutions to group customers according to their behaviours, financial status, and preferences. Tuning the hyperparameters of these models can lead to accuracy in segmentation and market studies.

Issue 2: Optimizing the Bias-Variance Tradeoff. Jude Chukwuebuka Ugwuoke.

Technical

Introduction to the Bias-Variance Tradeoff.

Definition: The bias-variance trade-off essentially delineates the equilibrium between two unique error types that models experience while they are being trained (Belkin et al.; Yang et al.). Bias represents the error caused by the use of highly limiting assumptions, where a model oversimplifies and thus cannot capture very complex data relationships. Variance is the error caused by being too sensitive to the training sample, which leads to a great fit of the model for training instances and poor performance for the test set.

Classification.

- **High Bias/Underfitting:** A straightforward model not capable of capturing the complex relationship among the data is said to be high bias; it has a high training error (Briscoe and Feldman).
- **High Variance/Overfitting:** A model that is highly sensitive to idiosyncrasies in the training data is referred to as high variance; it is too complex and does not work well with new data.

Consequences of Imbalance.

A model with too much bias fails to notice useful signals, whereas a model with too much variance not only captures the noise but also overfits the training data. It is, therefore, important to find the point that is just right to achieve strong out-of-sample results.

Mathematical Formulation.

Expected Test Error.

Let $\hat{f}(x)$ be a fitted model for a point x. Then, the expected test error at x can be captured by:

$$\left(E(\mathrm{y} - \hat{f}(x_0))\text{^}2\right) = Var\left(\hat{f}(x_0)\right) + \left[Bias\left(\hat{f}(x_0)\right)\right]^2 + Var(\epsilon)$$

Bias, Variance, and Irreducible Error.

- Variance quantifies how sensitive $\hat{f}(x)$ is to the training data.
- Bias reflects how far $\hat{f}(x)$ deviates from the correct solution on average.
- Irreducible Error (Var(ϵ) is the noise that is inherent in the data.

Reference Equation.

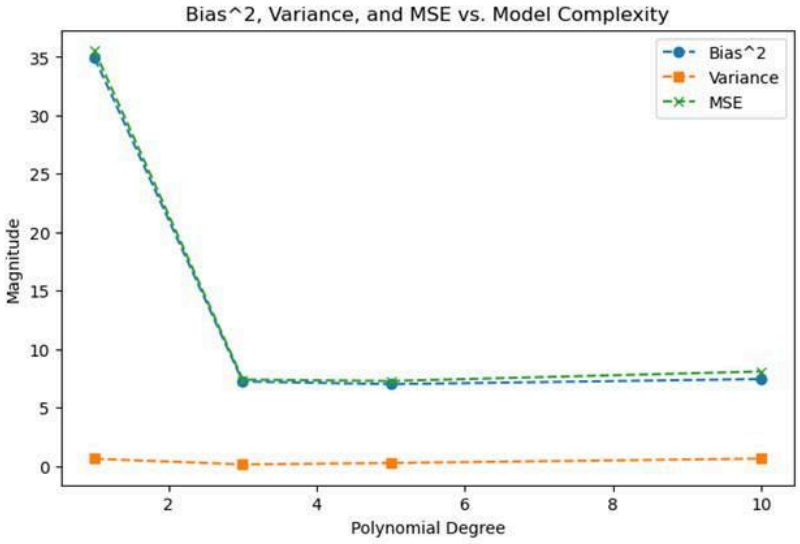

Fig 3: Bias-Variance Trade-off

From the figure above, Degree 1 yields a very large bias^2 and a low variance, which results in a high MSE due to significant underfitting. Increasing the polynomial degree initially lowers bias^2, which sharply reduces the MSE around Degree 3, but climbing further to Degree 10 begins to inflate variance again, which causes a slight uptick in MSE, showing an early sign of overfitting. This pattern reflects the reference equation below:

$$\text{MSE} = \text{Var}\left(\hat{f}(x)\right) \left(+\left[\text{Bias}\hat{f}(x)\right]\text{^}2 + \text{Var}(\epsilon)\right.$$

This indicates that while increasing complexity can reduce bias, it also increases variance, thus validating that a certain model complexity gives the optimum generalization.

Approaches to Controlling Bias and Variance.

Regularization and Cross-Validation.

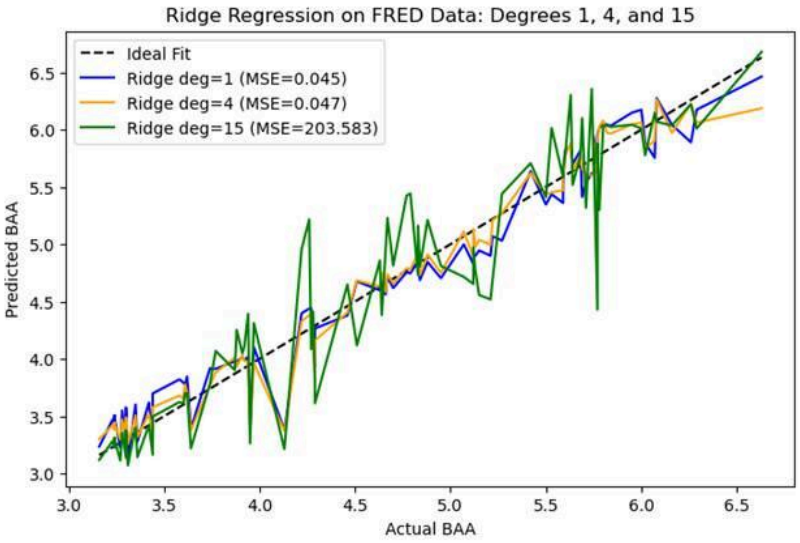

Fig 4: Ridge Regression

Regularization reduces overfitting by constraining the level of regression coefficients and improving generalization to unseen data. In ridge regression, which applies L2 regularization, there is a penalization of large weights that discourages overly complex models. While stronger penalties may slightly increase bias, they can drastically reduce variance, making predictions more robust across different datasets.

Cross-validation complements this approach by partitioning the data into training and validation folds so as to assess model stability and guide the hyperparameter tuning. The figure above demonstrates Ridge regression that is applied to FRED macroeconomic data (FEDFUNDS and AAA) to predict BAA bond yields. In the figure, the polynomial models of degrees 1, 4, and 15 were compared using cross-validation, and degree 15 overfits the data, which results in erratic predictions and a significantly higher MSE. Degrees 1 and 4 provide smoother fits and nearly identical, low MSE values, evidencing that simpler models with proper regularization achieve optimal bias-variance tradeoff for forecasting credit spreads.

Pruning in Decision Trees.

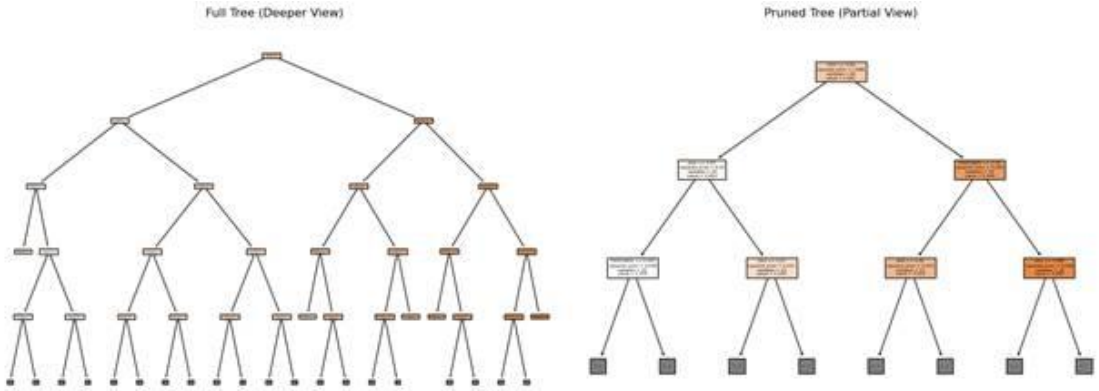

Fig 5: Full vs Pruned Tree

Decision trees, while being powerful, are also prone to overfitting when allowed to grow without constraints on noisy or limited macroeconomic datasets. They capture not only the true signal but also spurious fluctuations that are specific to the training period. Pruning alleviates this by limiting the tree's complexity, trimming branches that do not significantly reduce prediction error. This simplifies the model and increases its ability to generalize to unseen data.

The figure above compares two decision trees built on FRED macroeconomic data (FEDFUNDS and AAA used to predict BAA yields). The full tree on the left is grown to maximum depth and fits the training data closely, but also models anomalie,s which increases the risk of overfitting. The pruned tree, on the right, restricts depth to 3 levels. Though this slightly increases bias, it reduces variance. The mean squared error (MSE) confirms this, as we have 0.049 for the full tree vs. a lower 0.044 for the pruned version.

Ensemble Methods.

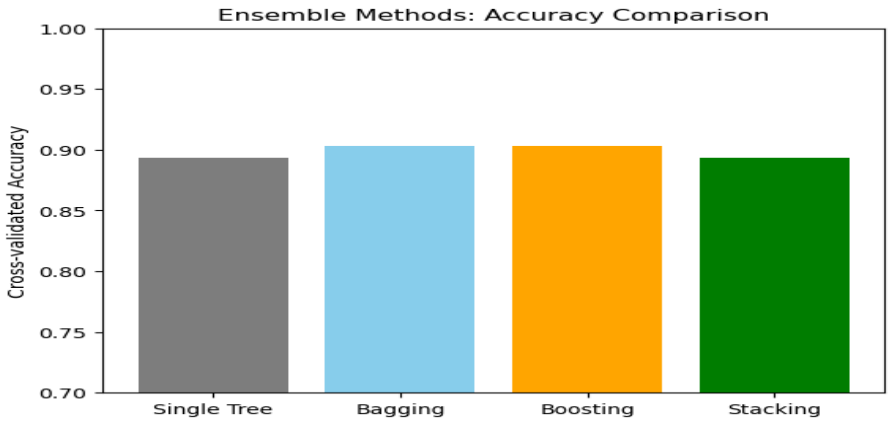

Fig 6: Ensemble Methods: Accuracy Comparison

Ensemble methods combine models to improve prediction accuracy and reduce the limitations of individual learners. While a single decision tree may overfit the training data (due to high variance), ensemble methods aggregate multiple learners to balance this effect. For example, bagging, such as in Random Forests, reduces variance by training each tree on different subsets of the data and averaging the results at the end. Boosting, on the other hand, builds models sequentially—each new learner focuses on the errors of its predecessors, effectively lowering bias. Stacking integrates diverse model types and uses a meta-model to refine predictions, enhancing both bias and variance handling.

The figure above compares the cross-validated accuracy of four classifiers trained on the synthetic "make_moons" dataset: a standalone decision tree, bagging (Random Forest), boosting (Gradient Boosting), and a stacking ensemble. All ensemble methods slightly outperform the base decision tree. Bagging and Boosting both achieve over 90% accuracy, whereas stacking performs similarly but offers architectural flexibility.

Feature Selection.

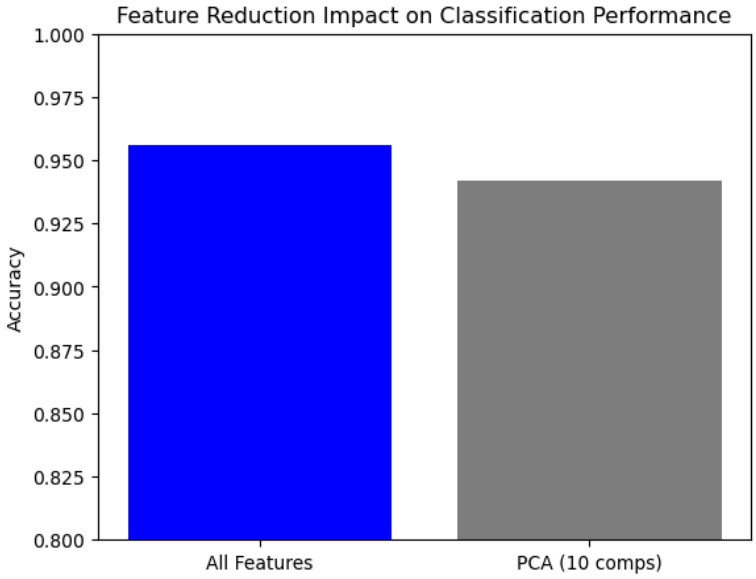

Fig 7: Feature Reduction Impact on Classification Performance

The bar chart above shows how using all the features in a dataset can lead to slightly better accuracy compared to reducing the dataset with PCA. Although the model with PCA used only 10 components, it still performed nearly as well as the full-feature model. This suggests that most of the information was captured by those components, but some detail was lost. This means that reducing the number of features helps make the model simpler and less prone to overfitting, which can be especially helpful when working with noisy or complex data.

Early Stopping in Neural Networks.

Neural networks are powerful tools for modeling complicated relationships in economic data. They can easily become too specialized to the training set, especially when allowed to train for too long. This overfitting results in high variance and weak generalization. Early stopping offers a practical remedy: it keeps track of performance on a validation set and halts training once improvements plateau, preventing the model from chasing noise rather than the true signal. In this case, we use a multi-layer perceptron regressor to predict BAA bond rates from FEDFUNDS and AAA rates in FRED macroeconomic data. The network includes two hidden layers and uses ReLU activation and Adam optimization. Instead of training blindly for a fixed number of epochs, we enable early_stopping = True in the model configuration.

The plot below shows the model's loss curve during training, which rapidly reduces error capturing the major trends in the data. After around 120–150 epochs, the curve flattens, and early stopping kicks in to freeze training. This ensures the final model is neither underfit nor overfit, and is well-suited for forecasting in a noisy financial environment.

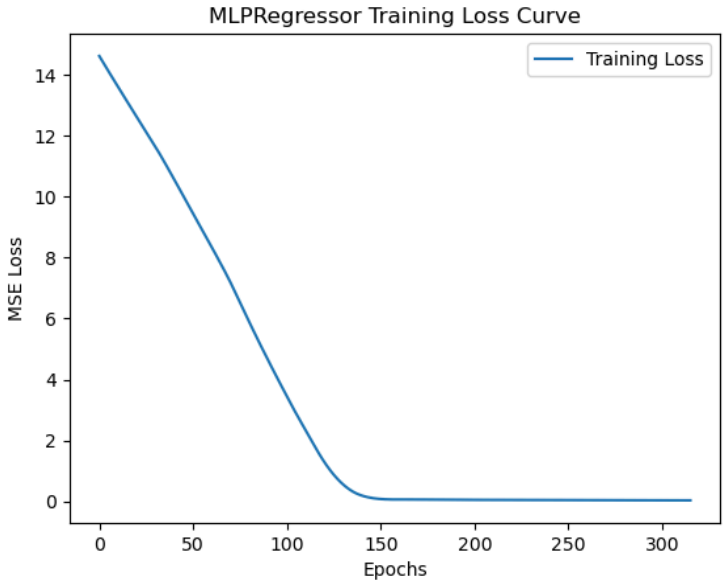

Fig 8: MLPRegressor Training Loss Curve

Performance Evaluation.

- Evaluating Regression Models. When working with models that predict numbers such as asset prices or interest rates, we use metrics like Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) to measure how far off the predictions are from actual values. Mean Absolute Error (MAE) is also another useful metric that gives a straightforward average of the prediction errors, irrespective of the direction.

- Evaluating Classification Models. For classification tasks such as predicting if a stock will go up or down, we look at accuracy, precision, recall, and the F1 score to get a full picture of how well the model is performing. The ROC AUC score evaluates how the model performs when the data is not balanced.
- Cross-Validation Techniques. To test how reliable a model is, we often use cross-validation. The most common method is K-Fold, where the data is split into parts and rotated between training and validation. For time-based financial data, using a time-series split is always better because it respects the sequence of events and avoids leaking future data into the past.
- Preparing the Data. Before modeling is performed, data needs to be cleaned and prepared by identifying and handling outliers, filling in missing values, and applying standardization or normalization.
- Tuning Model Parameters. To find the best settings for a model, we have to use methods like grid search, random search, or Bayesian optimization to balance the trade-off between underfitting and overfitting. Key parameters that need to be adjusted include model depth, learning rate, and regularization strength.
- Computational Load. More complex models need more memory and processing power. Here, it is important to keep an eye on how long training takes and weigh that against any improvements in model performance for easy comparisons.
- Backtesting for Financial Models. In finance, we often use historical data to simulate how a model would have performed in the past—this is called backtesting.
- Overfitting Problem. Financial data is noisy, making overfitting a big risk. The best safeguard is to validate models using new, unseen data and watch for sharp drops in performance when switching from training to testing sets.

Financial Applications.

- Market Forecasting. When forecasting markets, you risk making your model too simple or complicated. If it is too simple, you miss key trends. If it is too complex, you are just fitting noise. The best spot is in the middle, enough to catch useful patterns without overreacting to every market move.
- Portfolio Allocation. Allocating assets depends on stable estimates of return and risk. It is hard to make solid decisions if the model changes too much. But if it ignores necessary signals, one will have a bad mix. We need a setup that gives you confidence without being blind to the data.
- Credit Risk Assessment. In lending, the model needs to spot who is risky and who is not. If it overlooks details, one will credit the wrong people. If it over-analyzes, we decline good borrowers. Balance matters, so we are not flying blind or bogged down by noise.
- Algorithmic Trading. Trading models have to be sharp but not twitchy. You leave money on the table if they miss real opportunities because they are too cautious. If they jump at every blip, you rack up losses. The goal is to stay responsive, but grounded, especially when speed matters.

Non-Technical

Understanding the problem.

The concept of the bias-variance trade-off in machine learning (ML) addresses the equilibrium between two distinct errors that a model may experience when trained on data. Bias is an error caused by simplifying assumptions within the learning algorithm. A model with a high bias is too simple to capture the data patterns and disregards the training set meant to be trained. As a result, its performance is poor on both newly encountered data and the training set, thereby signifying underfitting. Variance is an error that is introduced by excessive difficulty in the learning algorithm. A highly variant model is overly responsive to the training set's noise. Though it achieves strong results on the training set, it underperforms on untested data since it learns both the noise and the true patterns. This is known as overfitting. If we reduce bias by increasing complexity, we inadvertently inflate the variance, and vice versa. When we strike an ideal bias-variance balance, it is best to construct an ML model capable of robust generalization to unseen data.

The Objective.

Achieving the optimal bias-variance trade-off entails finding the "spot" where the model is neither too simplistic nor overly sophisticated in fitting the training data. Typical solutions to this balancing act use regularization, cross-validation, decision tree pruning, ensemble methods, feature engineering or selection, hyperparameter refinements, and early stopping for neural networks. Ultimately, these processes minimize bias and variance simultaneously so that the final model retains strong performance in training and excels out-of-sample.

Issue 3: Applying Ensemble Learning: Bagging, Boosting, or Stacking. Alejandro Piedra Alvarado.

Technical

I. Introduction to Ensemble Learning.

Ensemble learning is a technique that combines multiple machine learning models to enhance predictive accuracy and robustness. It is classified into two types: homogeneous ensembles, which use the same type of base model (e.g., Random Forest), and heterogeneous ensembles, which integrate different models (e.g., SVM and Neural Networks). The key advantages of ensemble learning include variance reduction (by averaging predictions) and bias reduction (by iteratively refining models) (Dietterich).

II. Bagging (Bootstrap Aggregating).

Bagging mitigates variance and improves stability by training multiple models on different bootstrap samples of the data and aggregating their outputs. The most well-known bagging-based algorithm is the Random Forest (Breiman).

Algorithm.

1. Generate multiple bootstrap samples from the training dataset.
2. Train an independent model on each sample.
3. Aggregate the predictions via averaging (for regression) or majority voting (for classification).

Mathematical Formulation.

Given a dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, bagging train B models on bootstrap samples D_b

and aggregates predictions: $\hat{y} = \frac{1}{B}\sum_{b=1}^{B} f_b(X)$.

Python Code. Refer to "MLiF_GWP3_g8507.ipynb", 3: Applying Ensemble Learning - Bagging.

III. Boosting.

Boosting sequentially trains weak models, each focusing on the errors of its predecessor. Popular boosting methods include AdaBoost, Gradient Boosting, and XGBoost (Friedman).

Algorithm.

1. Train a weak learner on the dataset.
2. Assign higher weights to misclassified observations.
3. Train the next weak learner to correct previous mistakes.
4. Combine weak learners into a strong model.

Mathematical Formulation.

At each iteration t, boosting fits a weak learner $f_t(x)$ to the pseudo-residuals:

$$r_i^{(t)} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F^{(t-1)}}$$

Then it updates the ensemble model. $F^{(t)}(x) = F^{(t-1)}(x) + \eta \cdot f_t(x)$

η being the learning rate.

This method focuses on learning where previous models failed, which is especially useful in detecting rare events like defaults or fraud (Chen & Guestrin; Friedman).

Python Code. Refer to "MLiF_GWP3_g8507.ipynb", 3: Applying Ensemble Learning - Boosting.

IV. Stacking.

Stacking combines multiple base models through a meta-learner that optimally integrates their predictions (Wolpert), improving generalization across asset classes and horizons.

Algorithm.

1. Algorithm: Train multiple base models on the training data.
2. Use their predictions as input features for a meta-learner.
3. The meta-learner combines base model predictions for improved accuracy.

Mathematical Formulation.

Given base models $f_1, f_2, \ldots, f_n$, stacking uses a meta-learner g

to optimize predictions: $\hat{y} = g(f_1(X), f_2(X), \ldots, f_n(X))$

Python Code. Refer to "MLiF_GWP3_g8507.ipynb", 3: Applying Ensemble Learning - Stacking.

V. Performance Evaluation.

1. Regression: The main regression metrics were already mentioned in Issue 1: Optimizing Hyperparameters: Tuning Performance Metrics: Regression Metrics (MSE, RMSE, and MAE, and R-squared).

2. Classification: As in the previous point, the main regression metrics have also been mentioned in Issue 1: Optimizing Hyperparameters: Tuning Performance Metrics: Classification Metrics (Precision, Recall, F1 Score, Accuracy Score), yet here is relevant to dive deeper into one of the "other metrics": the ROC AUC..

   There is not a single equation that directly represents the ROC curve itself. Yet, the Area Under the ROC Curve (AUC) can be represented mathematically as an integral:

$$AUC = \int_0^1 TPR(FPR^{-1}(f))df$$

   Where:

   - $TPR(t)$ is the True Positive Rate at a given threshold t.
   - $FPR(t)$ is the False Positive Rate at a given threshold t.
   - $FPR^{-1}(f)$ is the inverse function of the FPR, giving the threshold t at which the FPR is f.

   Such was completed in MLiF_GWP3_g8507.ipynb, 3: Applying Ensemble Learning, where the figures represent the AUC for Bagging, Boosting, and Stacking models, respectively.

   These figures, on the left side, show the AUC for each given class using the trapezoidal rule on the (FPR, TPR) pairs, which is a numerical approximation of the integral:

$$AUC = \int_0^1 TPR(FPR_i^{-1}(f))df$$

   Where the subscript i denotes the class.

Class 0 represents negative returns.	**Class 1 represents neutral returns [-1% - +1%]**	**Class 2 represents positive returns.**

   Then, on the right side of the figures, the Macro-average AUC is shown. This is accomplished by interpolating the TPR values for each class at a common set of FPR values. Then it averages the interpolated TPR values to get the mean_tpr. Finally, the macro-average ROC AUC is calculated as the AUC of this macro-average ROC curve:

$$AUCmacro = auc(fpr_{macro}, tpr_{macro})$$

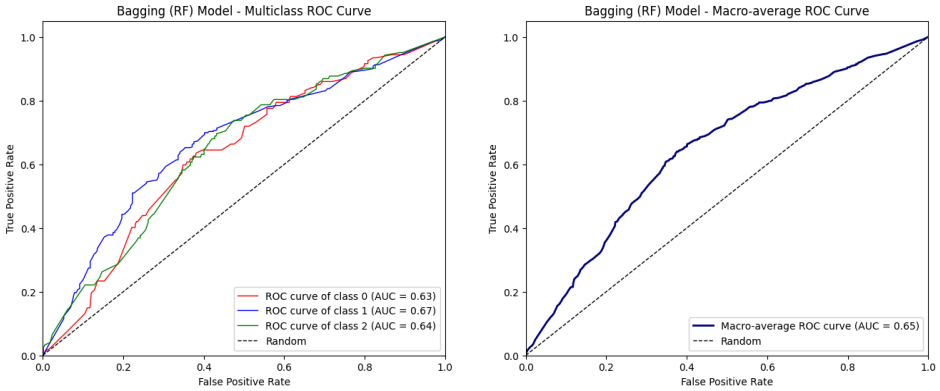

Fig 9: Bagging Model.

Boosting Model.

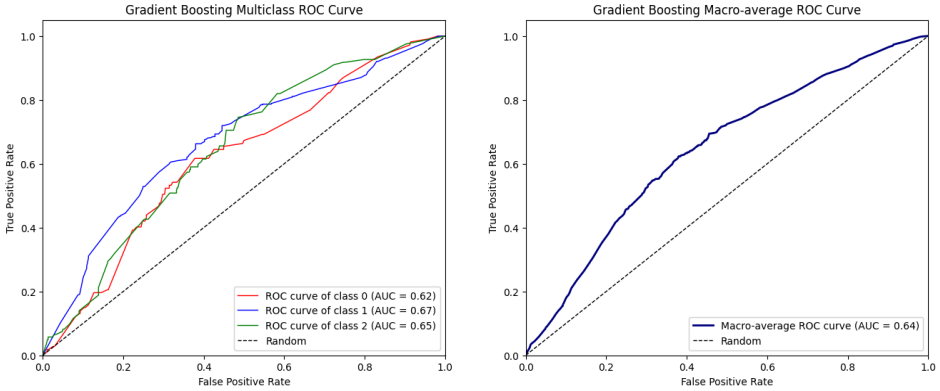

Fig 10: Boosting Model.

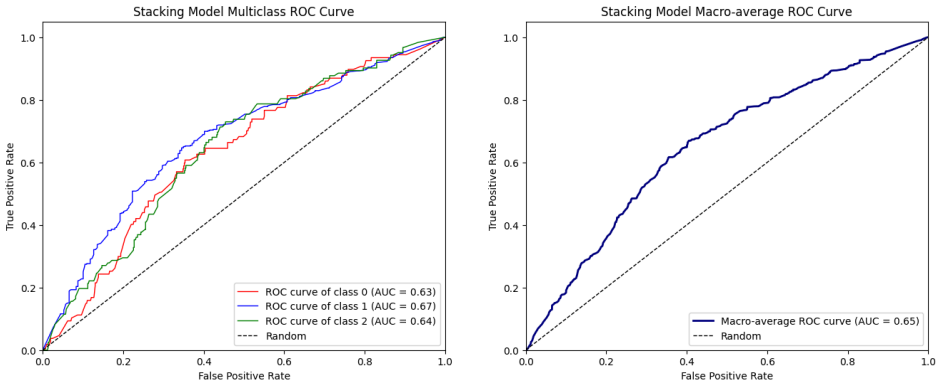

Fig 11: Stacking Model.

VI. Implementation Considerations.

Data Preprocessing & Feature Selection. Data preprocessing ensures that the raw financial data is cleaned and standardized. Techniques like imputation, normalization, and scaling help stabilize model performance. Feature selection is crucial for focusing on the most relevant financial indicators, reducing noise, and improving prediction accuracy by eliminating irrelevant features.

Hyperparameter Tuning. Essential for optimizing ensemble models. More on this in "Issue 1: Optimizing Hyperparameters" from this same report.

Computational Cost. Ensemble methods, especially those with many base models, can be computationally intensive. For large datasets, this can increase processing time and resource consumption. Techniques like parallel processing and model pruning help mitigate the computational load, allowing models to scale effectively for applications like algorithmic trading or risk management.

Backtesting. Is key to evaluating an ensemble model's robustness, simulating its performance on historical data. It helps identify potential overfitting and ensures that the model can adapt to real-world conditions. Proper backtesting considers transaction costs and slippage, which can affect model performance in practice, and out-of-sample performance to avoid look-ahead bias (Lopez De Prado, 2018).

Overfitting Risk. While ensemble methods help reduce overfitting, careful model monitoring is essential. Overfitting occurs when a model captures noise rather than underlying patterns, hindering its performance on new data. Cross-validation and regularization techniques help ensure that the model generalizes well to unseen financial data, which is crucial for maintaining accuracy in dynamic market conditions.

Non-Technical

I. Introduction to Ensemble Learning.

Ensemble learning is a powerful technique that combines multiple models to enhance the accuracy and reliability of predictions. In essence, it's like consulting a group of experts to make better-informed decisions. For example, just as a team of doctors with different specialties would offer a more reliable diagnosis, ensemble methods aggregate the insights from different models to improve the decision-making process.

For portfolio strategists, ensemble learning is akin to consulting multiple experts before making investment decisions. By using various methods, such as bagging, boosting, and stacking, ensemble techniques provide more robust and accurate financial modeling, which is essential in today's unpredictable markets.

II. Key Benefits.

Ensemble learning offers significant advantages for portfolio strategists, enabling more robust and accurate financial decision-making.

- Improved Prediction Accuracy. By combining multiple models, ensemble methods enhance the reliability of predictions, leading to better investment decisions. This is particularly valuable in portfolio optimization, where accurate predictions are crucial for effective asset allocation, especially during volatile market periods.
- Robustness and Risk Reduction. Ensemble methods reduce the risk of relying on a single, potentially flawed model. This robustness is essential for minimizing risks in financial portfolios. Reduced

overfitting ensures that models generalize well to new data, improving the stability of investment strategies.

- Handling Complex Data and Market Adaptability. Ensemble techniques are adept at processing high-dimensional and noisy financial data, which is common in real-world scenarios. Their flexibility allows them to adapt to changing market conditions by capturing complex patterns that single models may miss. This adaptability is vital for navigating dynamic financial landscapes.
- Applications in Algorithmic Trading and Credit Risk. Stacking methods can be used to combine technical and fundamental analysis models to create better trading signals. Boosting techniques such as XGBoost also improve credit risk assessments by focusing on correcting previous model errors, enhancing the ability to predict loan defaults.

III. Bagging (Reducing Variability).

Bagging, or Bootstrap Aggregating, involves creating multiple models from different subsets of the data and averaging their predictions. This reduces variability and helps stabilize predictions. A practical example is the use of Random Forest in credit scoring, where bagging helps to smooth out predictions and reduce the risk of extreme fluctuations in outcomes, improving portfolio stability.

IV. Boosting (Improving Accuracy).

Boosting works by iteratively refining models and focusing on mistakes made by earlier versions. This is akin to learning from past errors to improve performance over time. Models like XGBoost, often used in stock prediction, focus on correcting previous errors, making them particularly useful for predicting default risks and refining financial models.

V. Stacking (Combining Strengths).

Stacking combines multiple models to create a "super-model" that leverages the strengths of each individual model. It's like having a team of specialists working together under a manager's guidance. In financial forecasting, stacking might involve combining decision trees with time-series models, offering a more comprehensive approach to predicting market trends.

The figure below is a great way to visualize the three models [Bagging (RF), Boosting (GB), and a combined Stacking method] and their predictive power on whether the S&P 500's next day return will be negative, neutral (not much change), or positive.

The plot shows how well each method can distinguish between a positive return (+1%) and a non-positive return (-1%). The bottom of the plot shows how often the method incorrectly predicts a positive return (false alarm). The left side shows how often it correctly predicts a positive return. The lines for each method show this trade-off. The higher and further to the left a line is, the better the method is at identifying positive returns, minimizing false alarms.

The number next to each name (the AUC) is a score for how good the method is overall at this. Closer to 1 is better than closer to 0.5 (which is just guessing, shown by the dashed line). This model comparison shows the "Stacking" method having a higher score (0.6611) than the others, suggesting it's slightly better at predicting positive returns accurately with fewer wrong guesses, at least compared to the individual "Random Forest" and "Gradient Boosting" methods.

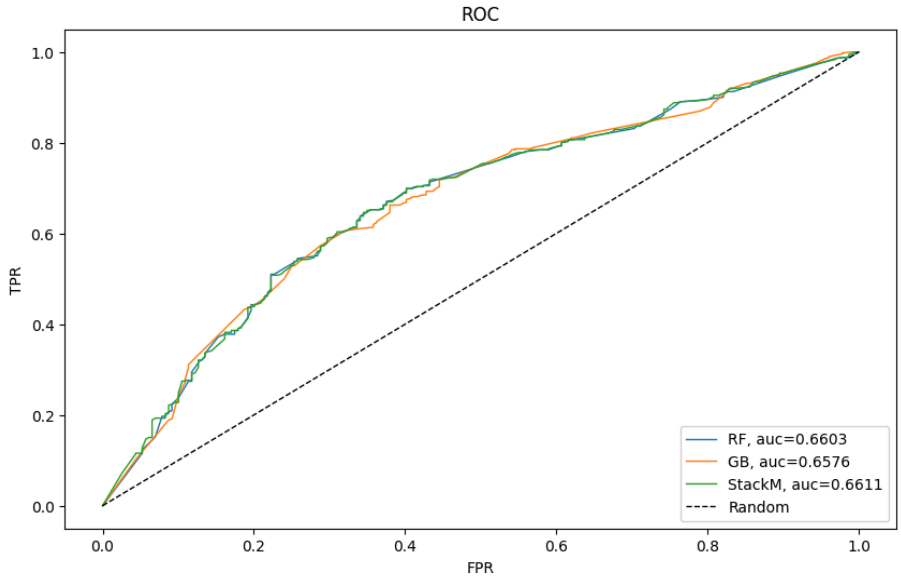

Fig 12: ROC - Stacking Model.

VI. Financial Applications.

Portfolio Optimization: Ensemble methods enhance asset allocation by combining models, leading to more robust investment strategies, especially during periods of market volatility.

Credit Risk Modeling: Boosting techniques like AdaBoost and XGBoost improve credit risk assessments by focusing on previous model errors, making them effective in identifying high-risk borrowers and predicting defaults.

Algorithmic Trading: Stacking integrates both technical and fundamental models, improving the generation of trading signals and enhancing market predictions.

Fraud Detection: By combining multiple models, ensemble learning reduces false positives, improving the accuracy of fraud detection while minimizing errors.

VII. Real-World Examples and Case Studies.

Credit Scoring: Financial institutions use Random Forest to improve credit scoring models, reducing the risk of loan defaults. A key study, "Consumer Credit Scoring Using Logistic Regression and Random Forest" (Sen Roy) demonstrates the effectiveness of this method.

Hedge Fund Strategies: Hedge funds have adopted ensemble methods to enhance trading strategies.

- The study "Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy" (Yang et al.) shows how combining different models improves trading performance.
- Ensemble methods like boosting and stacking outperform standalone models by accounting for non-stationarities in financial time series and reducing noise-driven overfitting, as demonstrated in *"Advances in Financial Machine Learning"* (Lopez De Prado).

Industry Endorsement: Jaffray Woodriff, CEO of Quantitative Investment Management, has highlighted the importance of ensemble methods in trading. In his foreword to Ensemble Methods in Data Mining, he shared, "I stumbled upon the winning concept of creating one 'super-model' from a large and diverse group of base predictive models," showcasing how ensemble strategies have transformed trading approaches.

VIII. Conclusion.

Ensemble learning has become a valuable tool in the financial sector by enhancing prediction accuracy and reliability through the combination of multiple models. These methods help financial professionals make better-informed decisions in today's complex and unpredictable markets. By reducing the risk of overfitting and improving the ability to handle noisy or incomplete data, ensemble techniques enable decision-makers to adopt more robust and adaptable strategies. As the financial world grows more and more complex, ensemble methods will remain essential in refining forecasting and risk management. Their ability to merge different models into a unified, high-performance "super-model" makes them an indispensable resource for professionals looking to navigate rapidly changing conditions with confidence and precision.

Step 4

	Wrote	Reviewed
Student A: Alejandro Piedra Alvaardo	Issue 3	Issue 2
Student B: Peter Esekhaigbe	Issue 1	Issue 3
Student C: Jude Chukwuebuka Ugwuoke	Issue 2	Issue 1

Step 5: Revised Marketing Alpha

In this section, we highlight the enhanced strengths of our machine learning framework and how it contributes to the development of robust, data-driven trading strategies. These refinements not only improve predictive accuracy but also position our models to generate meaningful alpha, that is, returns exceeding a market benchmark, in dynamic financial environments.

The Power of Machine Learning in Finance: A Strategic Evolution.

Machine learning continues to transform quantitative finance by revealing patterns and signals that traditional methods often miss. In previous reports, a powerful set of tools were introduced: LASSO Regression for feature selection, Hierarchical Clustering for market segmentation, Principal Component Analysis (PCA) for dimensionality reduction, Linear Discriminant Analysis (LDA) for classification, Support Vector Machines (SVMs) for nonlinear separation, and Neural Networks for modeling complex relationships.

In this report, a layout to further elevate those models (and an array of others) through strategic enhancements that address some of the main questions raised by portfolio strategists was presented.

Key Advancements: A Machine Learning Framework.

- **Precision Parameter Tuning.** Systematic hyperparameter optimization techniques, such as grid search and cross-validation, were applied to refine model configurations. This ensures each model operates efficiently, reducing reliance on ad hoc adjustments and improving overall stability.
- **Robust Predictive Performance.** Emphasis was placed on models that not only perform well on historical data but also generalize effectively to out-of-sample scenarios. Managing the bias-variance tradeoff helps prevent overfitting and improve out-of-sample reliability, which is critical for live market applications.
- **Strategic Model Integration.** Ensemble techniques, such as bagging, boosting, and stacking, were employed to combine diverse models in a way that amplifies strengths and mitigates individual weaknesses. This integration demonstrated improved performance over single-model approaches across testing scenarios.

Investor Benefits.

- **Deeper Insights.** The improved framework enhances the ability to extract actionable signals from complex, high-dimensional financial data, enabling sharper insights into market behavior.
- **Confident Decision-Making.** Enhanced predictive capabilities provide investment professionals with more reliable inputs for strategic decision-making.
- **Dynamic Market Adaptation.** The machine learning techniques are built to learn from evolving data patterns, improving resilience in response to shifting market regimes.
- **More Resilient Risk Management.** Through robust validation and stress testing, model risk can be reduced, which helps build more stable and predictable portfolios.
- **Enhanced Alpha Generation.** The combination of tuning and ensemble techniques materially improves the potential to deliver consistent excess returns across varying market conditions.

Conclusion.

The framework presented here reflects a significant advancement from prior stages. Through optimized model parameters, improved generalization, and ensemble strategies, a suite of tools has been laid out that not only meets the demands of portfolio strategists but also offers investment managers a durable edge in today's complex and competitive markets

References

Belkin, Mikhail, et al. 'Reconciling Modern Machine-Learning Practice and the Classical Bias–Variance Trade-Off'. *Proceedings of the National Academy of Sciences*, vol. 116, no. 32, Aug. 2019, pp. 15849–54. *pnas.org (Atypon)*, https://doi.org/10.1073/pnas.1903070116.

Breiman, Leo. "Random Forests." *Berkeley.edu*, Jan. 2001, www.stat.berkeley.edu/~breiman/randomforest2001.pdf.

Belkin, Mikhail, et al. 'Reconciling Modern Machine-Learning Practice and the Classical Bias–Variance Trade-Off'. *Proceedings of the National Academy of Sciences*, vol. 116, no. 32, Aug. 2019, pp. 15849–54. *pnas.org (Atypon)*, https://doi.org/10.1073/pnas.1903070116.

Briscoe, Erica, and Jacob Feldman. 'Conceptual Complexity and the Bias/Variance Tradeoff'. *Cognition*, vol. 118, no. 1, Jan. 2011, pp. 2–16. *ScienceDirect*, https://doi.org/10.1016/j.cognition.2010.10.004.

Chen, Tianqi, & Carlos, Guestrin. "XGBoost: A Scalable Tree Boosting System." *arXiv*, 10 June 2016, https://arxiv.org/pdf/1603.02754.

Dietterich, Thomas. "Ensemble Methods in Machine Learning." *Oregonstate.edu*, 2000, web.engr.oregonstate.edu/~tgd/publications/mcs-ensembles.pdf.

Friedman, Jerome H. "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics*, vol. 29, no. 5, 2001, pp. 1189–232, www.jstor.org/stable/2699986.

Hastie, Trevor, et al. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed., Springer, 2009. https://www.sas.upenn.edu/~fdiebold/NoHesitations/BookAdvanced.pdf.

Lopez de Prado, Marcos. *Advances in Financial Machine Learning*. Wiley, 2018. *agorism.dev*, https://agorism.dev/book/finance/ml/Marcos%20Lopez%20de%20Prado%20-%20Advances%20in%20Financial%20Machine%20Learning-Wiley%20%282018%29.pdf.

Sen Roy, Hirak. *Consumer Credit Scoring Using Logistic Regression and Random Forest*. 2016, es.slideshare.net/slideshow/project-65739883/65739883.

Wolpert, David H. "Stacked Generalization." *Neural Networks*, vol. 5, no. 2, Jan. 1992, pp. 241–59, https://doi.org/10.1016/s0893-6080(05)80023-1.

Woodriff, Jaffray. "Ensemble Methods in Data Mining - Foreword." *Synthesis Lectures on Data Mining and Knowledge Discovery*, Springer International Publishing, 2010, https://doi.org/10.1007/978-3-031-01899-2. Foreword by Jaffray Woodriff. Giovanni Seni (Author), John Elder (Author), Robert Grossman (Series Editor).

Yang, Hongyang, et al. *Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy*. 11 Sept. 2020, papers.ssrn.com/sol3/papers.cfm?abstract_id=3690996.

Yang, Zitong, et al. 'Rethinking Bias-Variance Trade-off for Generalization of Neural Networks'. *Proceedings of the 37th International Conference on Machine Learning*, PMLR, 2020, pp. 10767–77. *proceedings.mlr.press*, https://proceedings.mlr.press/v119/yang20j.html.