Exercise Report: Units 1–3 and 8

Joan Fernández Navarro

STATISTICAL LEARNING
FOR DATA SCIENCE

MUCEIM

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Contents

List of Figures **2**

List of Tables **3**

1 Unit 1. Ensemble Learning **4**
1.1 Bias and Variance . . . 4
1.2 Ensemble Learning . . . 5
1.2.1 Model Diversity . . . 6
1.3 Voting Mechanisms . . . 7
1.4 Stacking . . . 8

2 Unit 2. Bagging Techniques - RF and ET **10**
2.1 Bootstrap . . . 10
2.2 Random Forest and Extra Trees . . . 12
2.3 Hiperparameter Optimization with Cross-Validadtion (RF) . . . 13
2.4 Project - Customer Classification . . . 14
2.4.1 Data Processing . . . 14
2.4.2 Model Training . . . 16
2.4.3 Results and Discussion . . . 16

3 Unit 3. Boosting Methods - Ada Gradient XGB **18**
3.1 Boosting Algorithms Comparison . . . 18
3.2 XGBoost . . . 19
3.3 Income Classifier - XGBoost . . . 20
3.4 Project - Losses Prediction . . . 23
3.4.1 Data Processing . . . 23
3.4.2 Model Training . . . 25
3.4.3 Results and Discussion . . . 26

4 Unit 8. Time Series Analysis **29**
4.1 Store Sales Time Series . . . 29
4.1.1 Preprocessing . . . 29
4.1.2 Decomposition . . . 30
4.1.3 Forecasting . . . 31
4.2 ARIMA and SARIMAX . . . 33
4.2.1 Autocorrelation . . . 33
4.2.2 ARIMA Models . . . 34
4.2.3 Seasonal ARIMA (SARIMA) Models . . . 35

List of Figures

1 Three polynomial regressions of different degrees with non-linear data ($y = sin(2X) + \varepsilon$). 5
2 Decision boundaries of the individual models showing varying complexity and adaptation to the data. 6
3 Decision boundary of the ensemble model illustrating improved separation and generalization. 7
4 Comparison of decision boundaries using hard, soft, and weighted voting ensembles, illustrating differences between binary and probabilistic cluster assignments. 8
5 Decision boundaries of the stacking ensemble using Logistic Regression as the meta-learner, showing probabilistic cluster assignments. 9
6 Distribution comparison between the full population and a small sample of 50 elements. 11
7 Bootstrap distributions of the sample mean and standard deviation across $1,000$ resamples, showing the 95% confidence intervals. 11
8 Feature importance comparison between Random Forest and Extra Trees models, showing similar relevance patterns. 13
9 Relative improvement in accuracy over the baseline Random Forest model for both grid search and random search hyperparameter optimization methods. 15
10 Distribution of customer profitability based on the 'BMA_corregido' variable. Profitable customers (values > 0) represent 19.3% of the total. . . 15
11 Feature importance distribution for the baseline model trained with all variables. 16
12 Confusion matrix of the final nine-variable model for customer classification. 17
13 Accuracy and training time comparison of AdaBoost, Gradient Boosting, XGBoost, and LightGBM on a synthetic two-cluster dataset. 19
14 Distribution of accuracy scores across 30 hyperparameter combinations for XGBoost. 20
15 Class distribution of the target variable in the Adult dataset. 20
16 Convergence of the XGBoost model during training, showing the evolution of log loss across iterations for the training and validation sets. . . 22
17 Feature importance derived from the trained XGBoost model. 22
18 Model performance visualization showing the confusion matrix (left) and ROC curve (right). 23
19 Distribution of hurricane damage classes for the hotel. Class 0 (Non-payable), Class 1 (Partially Payable) and Class 2 (Fully Payable). . . . 24
20 Convergence of the XGBoost classification and regression models. . . . 25
21 Feature importance derived from the trained XGBoost model. 26
22 Confusion matrix for the classification model for losses prediction. . . . 27
23 Predicted vs. actual losses and residuals for the regression model for losses prediction. 27

24 Daily time series of total sales across all stores. 29
25 Smoothing of the daily series using moving averages of 7, 30, and 90 days. 29
26 Monthly series of aggregate sales. 30
27 Decompositions of the monthly sales series. 31
28 Forecast using Single Exponential Smoothing with various α values. . . 32
29 Forecast using Double Exponential Smoothing (Holt's Method) with various α and β values 33
30 Forecast using Triple Exponential Smoothing (Winters Method). 33

List of Tables

1 Description of the variables included in the Adult dataset. 21
2 Description of the custom variables created for hurricane characterization. 24

The code developed to complete all the proposed exercises for this report is available in the online GitHub repository: https://github.com/JoanFer030/statistical_exercises

1 Unit 1. Ensemble Learning

1.1 Bias and Variance

In the field of machine learning, the trade-off between bias and variance is one of the fundamental concepts for understanding the behavior of predictive models. This section presents an experimental analysis of this phenomenon through the evaluation of polynomial regression models applied to data with nonlinear patterns, with the aim of illustrating how different levels of complexity model the relationship between variables differently.

- **Bias** Represents the inherent error that occurs when an overly simplified model attempts to approximate complex relationships present in the data. A model with high bias tends to make systematic errors, as it fails to adequately capture the patterns existing in the training data. This behavior is known as underfitting. In these cases, the simplicity of the model prevents it from reflecting the actual complexity, resulting in inaccurate predictions and poor generalizations.
- **Variance** Measures the sensitivity of the model to small variations in the training data. A model with high variance tends to overfit the patterns in the training set, including noise or outliers. This behavior is known as overfitting, in which the model achieves a very low error rate in the training data but performs poorly when evaluated on new data. In these cases, the excessive complexity of the model causes a loss of generalization ability.

To illustrate the concepts of bias and variance, an experiment was developed using synthetic data with a nonlinear relationship between the independent variable X and the dependent variable y. The data were generated according to the following expressions:

$$\begin{aligned} X &= [-2, 2]; \quad \text{divided into 100 equal subintervals} \\ y &= sin(2 \cdot X) + \varepsilon; \quad \varepsilon \sim N(0, 0.4^2) \end{aligned}$$

where ε represents a Gaussian noise term with a mean of zero and a standard deviation of 0.4. This dataset allows us to observe how different levels of model complexity affect fit and generalization.
Next, three polynomial regression models with different degrees of complexity (degrees 1, 3, and 20) were trained for the purpose of comparatively analysing the effects of underfitting and overfitting as a function of the degree of the polynomial.
As shown in Figure 1, the first-degree polynomial model (linear regression) exhibits a high level of underfitting. This model fails to capture any patterns in the training data, showing virtually no generalization ability. Numerically, it achieves a mean square error (MSE) of 0.6526 in the training set and 0.6287 in the validation set, which highlights its excessive simplicity and limited ability to represent the nonlinear relationship between

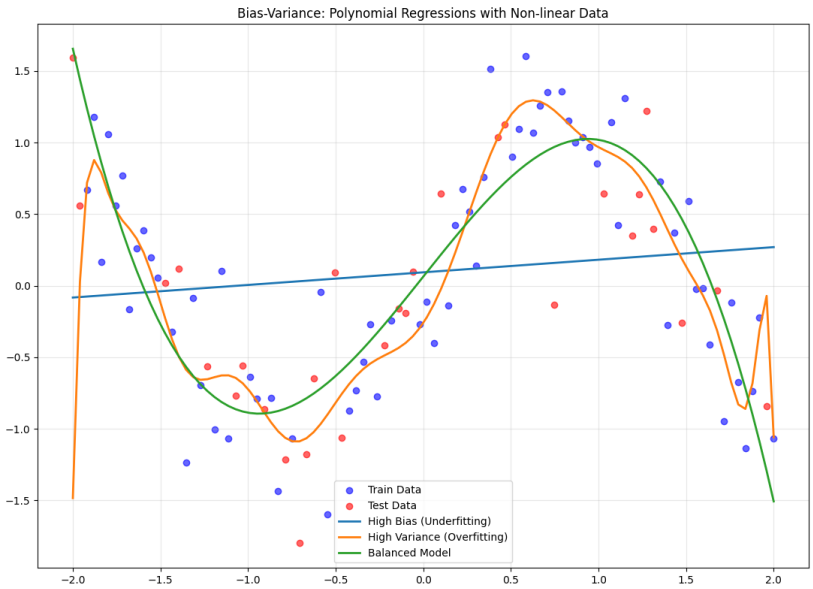


Figure 1: Three polynomial regressions of different degrees with non-linear data ($y = sin(2X) + \varepsilon$).

the variables.

At the opposite extreme, the 30th degree polynomial model shows a clear case of overfitting. As can be seen in the figure, the model reproduces the behavior of the training set almost exactly, even following the data pattern point by point. Numerically, it has an MSE of 0.1092 in training (notably low), but an MSE of 0.5336 in validation, which is considerably higher. This difference reflects the excessive complexity captured by the model, which is why it ends up adjusting to the noise in the training set and losing virtually all of its generalization capacity.

Finally, the third-degree polynomial model achieves an adequate balance between bias and variance. This model manages to appropriately represent the general trend of the training data without overfitting to it, which translates into good performance on the validation set. The MSE values are relatively similar in both cases, 0.1679 for training and 0.2236 for validation, confirming its ability to correctly generalize the behavior of the data.

1.2 Ensemble Learning

Ensemble methods have become essential tools for improving the performance and robustness of predictive models. These approaches combine multiple individual models to produce a final prediction that is typically more accurate and stable than that of any single model. In this section, an experiment is conducted using several individual models and one ensemble model, with the goal of analysing how model diversity affects prediction accuracy and generalization capability.

- **Bagging (Bootstrap Aggregating)** is an ensemble technique designed to reduce variance and improve model stability. It works by generating multiple versions of a training dataset through bootstrap resampling (randomly sampling with

replacement) and training a separate model on each of these samples. The final prediction is obtained by aggregating the outputs of all models, typically through averaging in regression tasks or majority voting in classification tasks. Bagging reduces overfitting by smoothing out the fluctuations caused by high-variance models, such as decision trees.

- **Boosting** unlike bagging, it aims to reduce bias by sequentially training models in a way that each new model focuses on correcting the errors made by the previous ones. In boosting, models are added iteratively, and the predictions are combined through a weighted sum, where the weights depend on each model's accuracy. Boosting transforms a collection of weak learners, that are models that perform only slightly better than random guessing, into a strong learner with significantly improved predictive performance.

1.2.1 Model Diversity

Model diversity refers to the degree of difference between the individual models that make up an ensemble. In other words, it measures how independently the models learn and make their predictions. Theoretically, high diversity implies that the models commit different types of error on the data, which allows the ensemble to combine their outputs in a complementary way. Therefore, the main goal of a high model diversity is to reduce the overall generalization error.
To analyse this effect, a synthetic dataset was generated consisting of 800 samples distributed in two distinct clusters. This setup allows us to evaluate how various models (both individual and ensemble) capture the underlying structure of the data and how model diversity contributes to improving predictive accuracy and generalization.
The models used in this experiment were a Logistic Regression, a Decision Tree, a Support Vector Machine (SVM) with an RBF kernel, and a k-Nearest Neighbors (kNN) classifier with $k = 5$ neighbors. Each of these models was trained and evaluated individually. Subsequently, they were combined into an ensemble model using a majority voting scheme to analyse how the aggregation of diverse classifiers affects predictive performance.

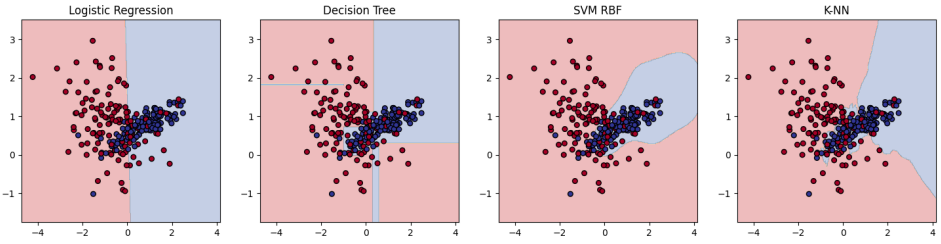


Figure 2: Decision boundaries of the individual models showing varying complexity and adaptation to the data.

As shown in Figure 2, the decision boundaries of the individual models vary considerably. In the cases of Logistic Regression and Decision Tree, the boundaries are more

linear, which is consistent with the inherent characteristics of these algorithms. Consequently, both models achieved lower accuracy scores, 0.7750 and 0.8125, respectively. On the other hand, the SVM and kNN models produced decision boundaries that more closely follow the non-linear shape of each cluster, resulting in higher accuracies of 0.8167 for SVM and 0.8333 for kNN.

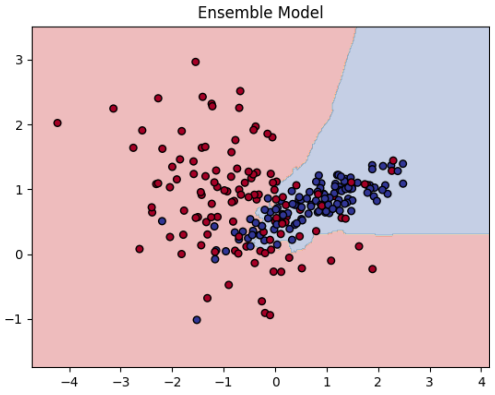


Figure 3: Decision boundary of the ensemble model illustrating improved separation and generalization.

Figure 3 illustrates the decision boundary of the ensemble model, which forms a more precise contour around both clusters, particularly around the second (blue) cluster. This boundary successfully captures the underlying structure of the data with greater accuracy, as reflected by the ensemble's overall performance, achieving an accuracy of 0.8392. These results demonstrate how combining diverse models through majority voting can enhance generalization and yield more robust predictions compared to individual classifiers.

1.3 Voting Mechanisms

Within bagging-based ensemble models, different voting strategies can be employed to combine the predictions of individual classifiers. These voting methods determine how the final class label is assigned based on the outputs of all base models. In this section, three common approaches are analyzed: hard voting, soft voting, and weighted voting.

- **Hard voting** assigns the final prediction based on the majority class selected by the individual models. Each model contributes one vote, and the class receiving the highest number of votes becomes the ensemble's output. This method is simple and effective when all models have similar performance, but it may be suboptimal if some classifiers are significantly more accurate than others.

- **Soft voting** takes into account the predicted class probabilities rather than just the final class labels. The probabilities produced by each model for every class are averaged, and the class with the highest aggregated probability is chosen. This approach often leads to better performance, as it considers the confidence of each prediction rather than treating all votes equally.

- **Weighted voting** extends the idea of soft voting by assigning different weights to the models according to their performance, typically based on validation accuracy or another evaluation metric. Models with higher accuracy have greater influence on the final decision.

Following the same experimental framework described earlier, an ensemble was trained using the same four classifiers, but this time applying each of the three voting strategies independently. A new synthetic dataset was generated for this purpose, allowing for a comparative analysis of how different voting methods affect classification accuracy and generalization performance.

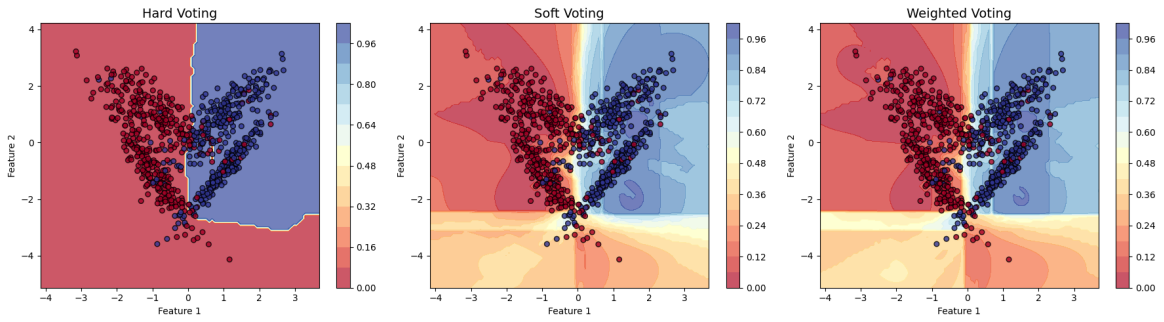


Figure 4: Comparison of decision boundaries using hard, soft, and weighted voting ensembles, illustrating differences between binary and probabilistic cluster assignments.

As shown in Figure 4, the decision boundaries exhibit a similar overall pattern across all four voting mechanisms, forming a quadrant in the upper right region where samples of the second cluster (blue) are classified. In the case of hard voting, the cluster assignments are strictly binary: each sample either belongs to the cluster or it does not, resulting in decision boundaries drawn as a single line.
In contrast, the decision boundaries produced by soft and weighted voting are probabilistic rather than fixed. Each sample is assigned a probability of belonging to one cluster or another, with the probability decreasing as the sample moves away from the cluster until reaching the boundary where both classes have equal probability. This probabilistic representation allows for a smoother separation between clusters.
Numerical results confirm the benefits of these probabilistic boundaries. The ensemble using soft voting achieved an accuracy of 0.9137, and the weighted voting ensemble reached 0.9150, both outperforming the hard voting ensemble, which obtained an accuracy of 0.9012. These results demonstrate that incorporating probability information and model weighting can enhance the ensemble's predictive performance and generalization.

1.4 Stacking

Stacking is another ensemble learning technique that aims to combine the strengths of diverse base models by learning how to best integrate their predictions. The main difference between stacking and the other two techniques (bagging and boosting) lies in the way the final prediction is made.

In stacking, multiple base models (also called level-0 models) are trained independently on the same dataset. The predictions of these base models are then used as input features for a higher-level model, known as the meta-learner or level-1 model. The meta-learner is responsible for learning the optimal way to combine the outputs of the base models, effectively correcting their individual weaknesses and producing a final prediction that is typically more accurate than any single base model.
Unlike bagging and boosting, which usually combine models using simple aggregation rules such as averaging or weighted voting, stacking leverages the predictive power of a machine learning model to determine how to merge the predictions.
Following the experimental framework described in the previous sections, a new synthetic dataset was generated. Using this data set, four base learners, the same models used previously, were trained independently. A Logistic Regression model was then employed as the meta-learner, which takes the predictions of the base learners as input features to produce the final ensemble predictions. This approach allows the meta-learner to learn how to optimally combine the outputs of the diverse base models.

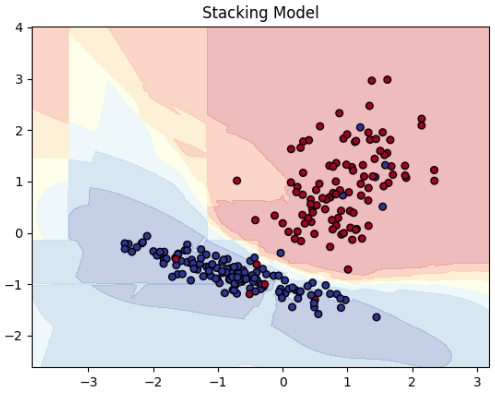


Figure 5: Decision boundaries of the stacking ensemble using Logistic Regression as the meta-learner, showing probabilistic cluster assignments.

As shown in Figure 5, one of the most notable characteristics of the stacking ensemble is the presence of probabilistic decision boundaries. By using Logistic Regression as the meta-learner, the model produces soft, rather than binary, cluster assignments, resulting in more nuanced and flexible boundaries. The ensemble effectively captures the underlying patterns of each cluster, as illustrated in the figure. This improved representation is also reflected in the numerical results, with the stacking model achieving an accuracy of 0.9458, outperforming the individual base learners and previous ensemble methods.

2 Unit 2. Bagging Techniques - RF and ET

2.1 Bootstrap

In this section, we introduce and analyze the bootstrap resampling method, a non-parametric technique widely used in ensemble learning. The concept of bootstrapping plays a central role, particularly in methods based on bagging (Bootstrap Aggregating). Bootstrapping in this context refers to the process of creating multiple training datasets by sampling with replacement from the original dataset. Each of these samples, called bootstrap samples, typically has the same size as the original data, but contains repeated observations and leaves out others.
The main goal of this resampling procedure is to introduce diversity among the base models. Since each model in the ensemble is trained on a slightly different subset of the data, their individual errors tend to be less correlated. When the predictions of all base learners are later aggregated, this diversity reduces the overall variance of the model and leads to better generalization performance. In essence, bootstrapping allows bagging to stabilize models that are highly sensitive to small changes in the training data, without increasing bias significantly.

Central Limit Theorem (CLT)
The Central Limit Theorem (CLT) establish that the sampling distribution of the sample mean tends to follow a normal distribution as the sample size increases, regardless of the shape of the original population distribution, provided that the samples are independent and identically distributed. Mathematically, if $X_1, X_2, ..., X_n$ are random samples from a population with mean μ and finite variance σ^2, then the standardized sample mean

$$Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$$

approaches a standard normal distribution as $n \to \infty$.
In the context of bootstrapping, the CLT provides the theoretical foundation for estimating the sampling distribution of a statistic when only one sample is available. By repeatedly resampling (with replacement) from the observed data and computing the statistic of interest (such as the mean or standard deviation) for each resample, the resulting distribution of these bootstrap estimates approximates the true sampling distribution. Thanks to the CLT, as the number of bootstrap samples grows, this empirical distribution converges toward a normal distribution centered around the population parameter.
This connection allows bootstrapping to be used as a practical method for assessing the variability, bias, and confidence intervals of estimators and also in ensemble methods like bagging, where resampling is used to stabilize model predictions.

To illustrate these concepts empirically, we generated a large synthetic population and performed a bootstrap experiment. The population was sampled from a Normal distribution with mean 30 and standard deviation 5, consisting of $50,000$ observations:

$$X_i \sim N(30, 5^2); \quad i = 1, ..., 50000$$

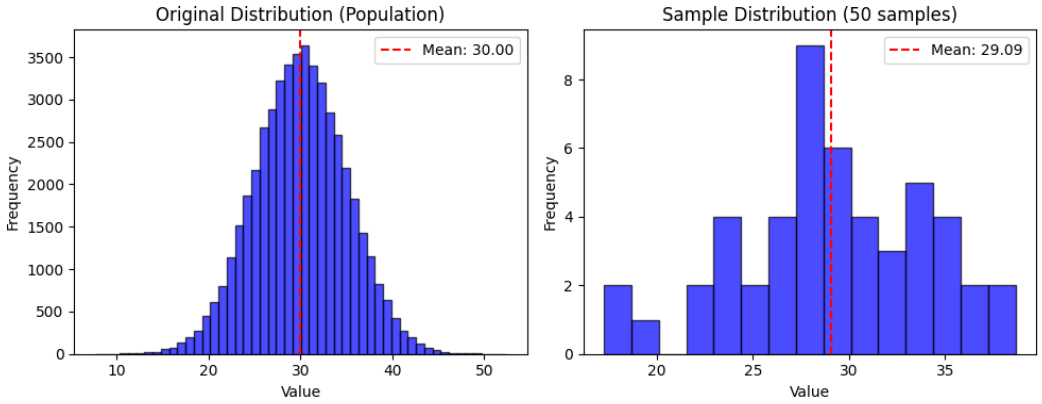


Figure 6: Distribution comparison between the full population and a small sample of 50 elements.

As shown in Figure 6, the generated population follows a clear normal distribution centered around a mean value of 30. However, when a small sample of only 50 elements is drawn from this population, the distribution of the sample no longer appears perfectly normal, and its mean slightly deviates from the population mean, taking a value of 29.09. This illustrates how limited samples can introduce sampling variability and deviate from the theoretical properties of the underlying distribution, a concept that bootstrap resampling seeks to mitigate by repeatedly drawing new samples from the available data, as will be shown in the following example.

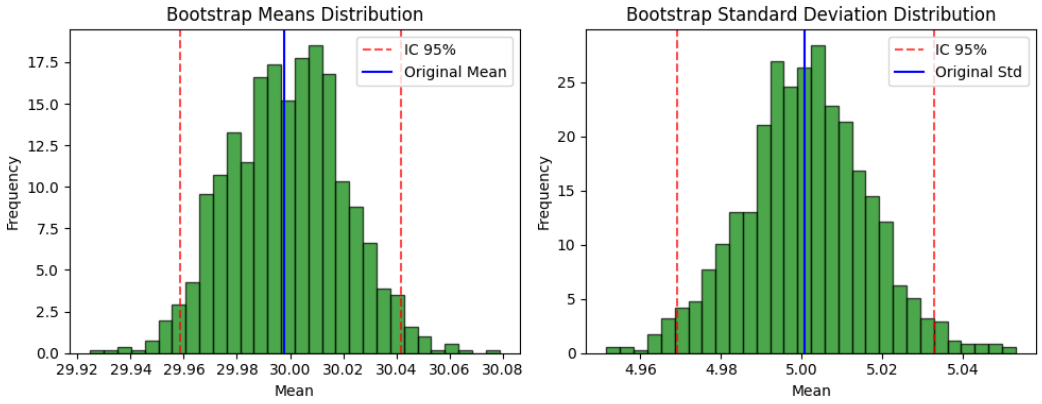


Figure 7: Bootstrap distributions of the sample mean and standard deviation across 1,000 resamples, showing the 95% confidence intervals.

Following with the same population and to ilustrate the functioning of the bootstrapping process, 1,000 samples of the same size as the original population were generated with replacement. For each of these samples, the mean and standard deviation were computed. As shown in Figure 7, the distributions of both statistics approximately follow a normal shape, which aligns with the theoretical expectations derived from the Central Limit Theorem.

When calculating the 95% confidence intervals, the results show that for the mean, the interval ranges from 29.96 to 30.04, clearly including the true population mean.

A similar pattern can be observed for the standard deviation, where the confidence interval extends from 4.97 to 5.03, once again covering the population's true standard deviation. These results confirm that the bootstrap successfully estimates the underlying population parameters, even when relying on repeated resampling from a single dataset.

2.2 Random Forest and Extra Trees

In this section, two tree-based ensemble methods are explored and compared; they are commonly referred to as randomization-based approaches. These methods aim to improve predictive performance by combining multiple weak learners into a single, stronger model. By introducing controlled randomness and iterative learning strategies, they seek to reduce both bias and variance while enhancing the model's ability to generalize to unseen data. These two models are Random Forest and Extremly Randomized Trees, also known as Extra Trees. An experiment is conducted to evaluate and compare their performance on a synthetic dataset, with the objective of analyzing how the level of randomness introduced during training affects model accuracy and stability.

- **Random Forest** is an ensemble learning method that constructs multiple decision trees using the bagging approach. Each tree is trained on a random bootstrap sample of the data, and at each split, a random subset of features is considered. This controlled randomness reduces correlation among trees and prevents overfitting, leading to improved generalization. The final prediction is obtained through averaging (for regression) or majority voting (for classification).
- **Extra Trees (Extremely Randomized Trees)** extend the idea of Random Forests by introducing an additional source of randomness. Instead of searching for the best possible split at each node, Extra Trees select split thresholds randomly for each feature and then choose the best among these random splits. This results in higher variance reduction and faster training times, often yielding slightly better generalization performance on noisy data. However, Extra Trees can sometimes be less stable in small or highly imbalanced datasets.

Following the same experimental setup as in the previous unit, both models were trained on a newly generated synthetic dataset, with $1,000$ samples, and evaluated using accuracy as the main performance metric. The comparison allows us to assess how the degree of randomness in tree construction influences the ensemble's ability to generalize and capture complex patterns in the data.

The results obtained from both models are highly comparable in terms of predictive performance, with accuracy values exceeding 0.8. Specifically, the Random Forest achieved an accuracy of 0.8233, while the Extra Trees model obtained 0.8167. The most remarkable difference between the two methods lies in their training efficiency. As theoretically expected, Extra Trees exhibited a considerably faster training process, finishing in 0.191 seconds compared to 0.414 seconds for Random Forest, an improvement of

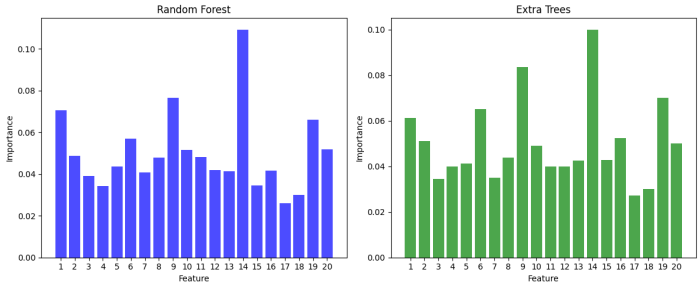


Figure 8: Feature importance comparison between Random Forest and Extra Trees models, showing similar relevance patterns.

approximately 116.8%. This highlights the computational advantage introduced by the additional randomization in the Extra Trees algorithm.
To complement the performance analysis, we further examined the feature importance assigned by both models. As illustrated in Figure 8, the two algorithms display a very similar distribution of feature relevance, even at the quantitative level. In both cases, feature 14 stands out with a relative importance of around 0.1. Additionally, features 1, 6, 19 and 20 consistently emerge as the most influential variables in both models, suggesting that both ensembles capture comparable structural patterns within the data set.

2.3 Hiperparameter Optimization with Cross-Validadtion (RF)

In this section, we focus on the optimization of hyperparameters, a critical step in building machine learning models. Hyperparameters are configuration settings that govern the learning process itself, such as the depth of a decision tree, the regularization strength in a logistic regression, or the number of neighbors in a k-Nearest Neighbors classifier. Unlike model parameters, which are learned directly from the training data, hyperparameters must be set prior to training. Optimizing these values is essential because they can significantly affect model performance, generalization, and the risk of overfitting or underfitting.
A common approach to evaluating hyperparameter choices is cross-validation, which involves partitioning the training data into multiple folds, training the model on a subset of folds, and validating it on the remaining fold(s). This process is repeated so that each fold serves as a validation set, providing a robust estimate of model performance. Additionally, early stopping is often employed to prevent overfitting during training, by monitoring validation performance and halting the learning process once improvements become negligible.
Two widely used strategies for hyperparameter search are described below:

- **Grid Search:** This method exhaustively evaluates all possible combinations of a predefined set of hyperparameter values. Although it guarantees that the global optimum within the grid is tested, it can be computationally expensive, especially for large search spaces.

- **Random Search:** Instead of testing all combinations, random search samples hyperparameter configurations randomly from specified distributions. This method often finds near-optimal solutions faster than grid search, particularly when only a few hyperparameters significantly influence model performance.

To illustrate these concepts, an experiment was conducted using a synthetic classification dataset composed of two classes. Both grid search and random search were applied to optimize hyperparameters of a Random Forest model, employing cross-validation to evaluate performance and early stopping to prevent overfitting during training. This setup allows a direct comparison of the two optimization methods in terms of efficiency and predictive performance.
After performing hyperparameter optimization with both techniques, several clear observations can be made. First, the number of combinations explored by each method differs substantially: grid search evaluated a total of 243 combinations, whereas random search tested only 50 configurations. Consistently with theoretical expectations, the search time required to find the best combination is also considerably lower for random search, taking 33.24 seconds compared to 78.99 seconds for grid search. These results might suggest that predictive performance (accuracy) would be substantially worse for random search; however, this is not the case. Grid search achieved an average accuracy of 0.7436 across the three folds for its best hyperparameter combination, while random search, despite requiring less than half the search time, reached an accuracy of 0.7371, differing only by a few hundredths.
When comparing both methods to a baseline model, as illustrated in Figure 9, the relative improvements over the baseline are modest, and there is no significant difference between the two optimization approaches. These findings highlight the practical advantage of random search for hyperparameter optimization, as it achieves near-equivalent predictive performance with substantially lower computational and temporal cost.

2.4 Project - Customer Classification

The objective of this project is to predict if a customer is going to generate profit over their lifetime, in order to determine whether the company should accept them as client or not. To achieve this, one Random Forest model will be developed, classification model to predict one of two categories, accept or deny.

2.4.1 Data Processing

The first step in the data preprocessing phase involved removing all columns containing information about the insurance policy itself, since these variables explicitly indicate whether a customer will generate profit or not, potentially introducing data leakage into the model. After this cleaning step, all rows with a value of 2 in the 'SINCO' column were removed. This filtering process resulted in a dataset composed of $51,618$ rows and 196 columns, ready for modeling.
Next, the target variable was defined to identify customers who generate profit for the company. This variable was derived from the 'BMA_corregido' field, where customers with values greater than zero were labeled as profitable.

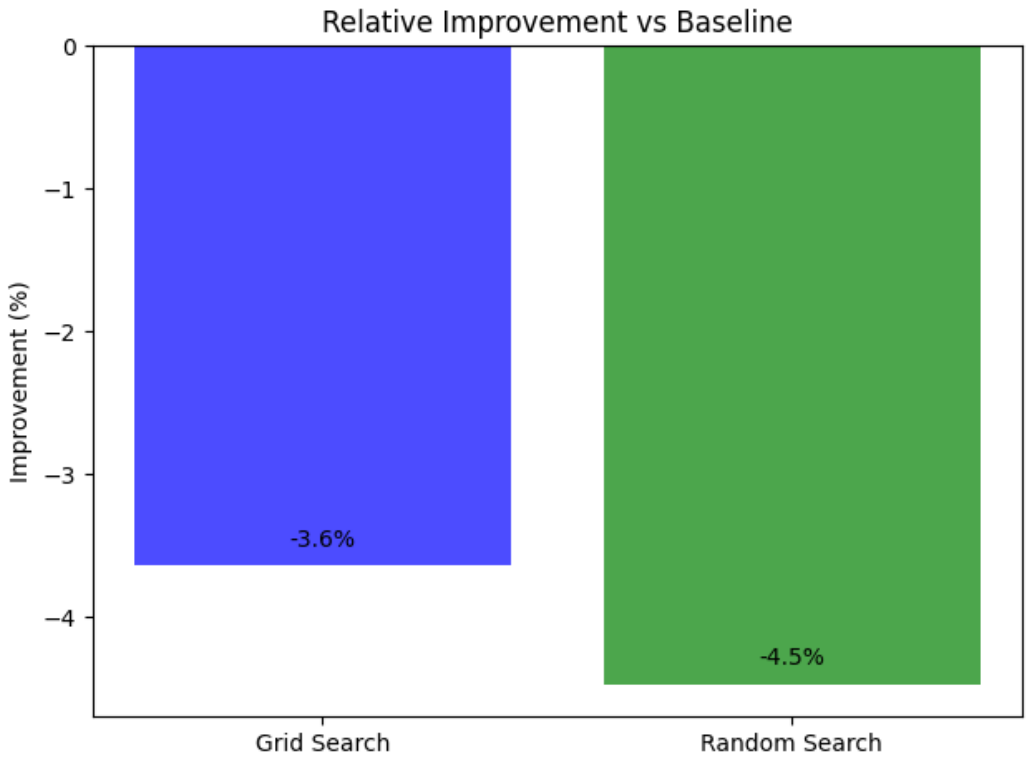


Figure 9: Relative improvement in accuracy over the baseline Random Forest model for both grid search and random search hyperparameter optimization methods.

As illustrated in Figure 10, the range of profits is considerably narrower than that of losses, and the number of profitable customers is also substantially smaller. Specifically, these customers represent only 19.3% of the total population, yet they generate an average profit of 17.63 euros per customer, resulting in a total profit of approximately 909, 881 euros, equivalent to about 7% of the total potential profit.

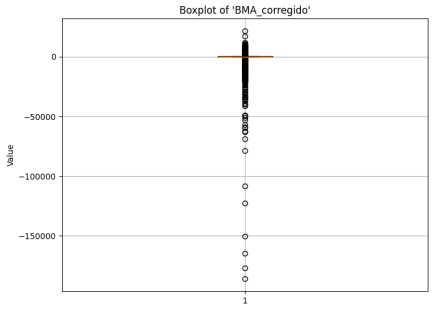


Figure 10: Distribution of customer profitability based on the 'BMA_corregido' variable. Profitable customers (values > 0) represent 19.3% of the total.

2.4.2 Model Training

For the training, validation, and testing process, the dataset was divided into three stratified subsets to preserve the original class distribution across all partitions. The resulting sample sizes were as follows: Training set with $32,518$ samples, Validation set with $15,486$ samples, and Test set with $3,614$ samples.

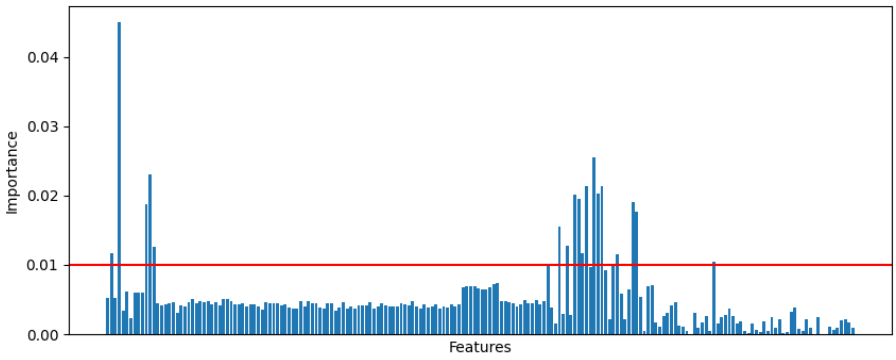


Figure 11: Feature importance distribution for the baseline model trained with all variables.

As a first step, a baseline model was trained using all available variables. This initial model served both as a benchmark for comparison and as a tool to evaluate feature importance. As shown in Figure 11, the relative importance of the features varies considerably, some variables exhibit minimal or even negligible contribution to the model's predictive performance. Moreover, the distribution of importance values is highly uneven: while most features contribute only marginally, a small subset accounts for a substantial proportion of the total importance.
Given this observation, the decision was made to isolate and retain only the most influential features—specifically, those within the top 5% of importance scores, resulting in a refined subset of nine key variables. A new model was then trained using only these selected features to assess whether predictive performance could be maintained or even improved with a more parsimonious representation of the data.

2.4.3 Results and Discussion

The model trained using all available variables required 31.02 seconds of training time and achieved a ROC AUC of 0.684 on the test set. In contrast, the model trained with only the nine most important features required just 11 seconds to train, while obtaining a slightly lower but comparable ROC AUC of 0.667. Given this small trade-off in predictive performance and the substantial reduction in computational cost, the nine-variable model was selected as the final version.
When examining the Figure 12 that contains the confusion matrix reveals that the model successfully classifies most observations belonging to Class 0, although it shows a tendency to over-classify in this dominant category. As a result, some instances that truly belong to Class 1 are incorrectly assigned to Class 0.

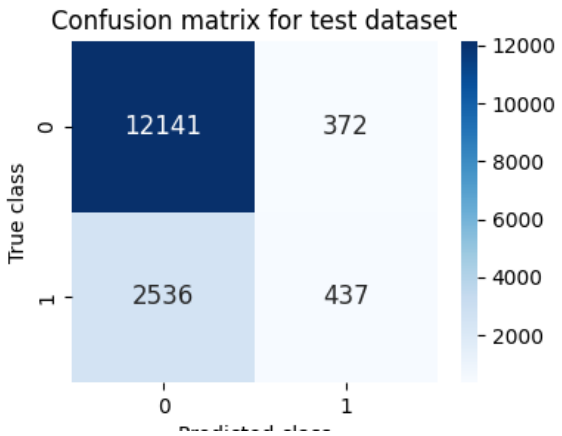


Figure 12: Confusion matrix of the final nine-variable model for customer classification.

From an economic standpoint, the model demonstrates strong practical performance: the average profit per client rises to 50.84 euros, and the overall profit margin increases to 21%, representing a 14-percentage-point improvement compared to the baseline scenario. The only noteworthy misclassification corresponds to a client erroneously accepted by the model, resulting in a loss of nearly $200,000$ euros.
In summary, despite minor classification imbalances, the final model effectively identifies profitable clients, enhancing both the average profit per client and the overall profitability margin.

3 Unit 3. Boosting Methods - Ada Gradient XGB

3.1 Boosting Algorithms Comparison

In this section, we focus on boosting, a powerful ensemble learning technique designed to improve predictive performance by sequentially combining multiple weak learners. Unlike bagging, which trains models independently on bootstrap samples, boosting trains each model in sequence, with each subsequent learner focusing on the errors made by the previous ones. By iteratively correcting the mistakes of prior models, boosting reduces bias and produces a strong overall predictor, often achieving higher accuracy than individual learners or simple ensembles.

- **AdaBoost (Adaptive Boosting):** Builds an ensemble of weak learners, typically decision stumps, by iteratively reweighting the training samples. Misclassified samples receive higher weights in subsequent iterations, forcing the next learner to focus on harder examples.
- **Gradient Boosting:** Sequentially fits new models to the residual errors of the ensemble constructed so far, effectively performing gradient descent in function space. This approach allows for the optimization of arbitrary differentiable loss functions.
- **XGBoost (Extreme Gradient Boosting):** An efficient and scalable implementation of gradient boosting that includes regularization, parallel processing, and handling of missing values, providing state-of-the-art performance in many machine learning tasks.
- **LightGBM (Light Gradient Boosting Machine):** A gradient boosting framework optimized for efficiency and scalability. It uses a leaf-wise growth strategy, histogram-based splitting, and other techniques to reduce training time while maintaining high predictive performance.

To illustrate the capabilities of these methods, an experiment was conducted using a synthetic dataset of 2,000 individuals divided into two clusters. This setup allows for a detailed analysis of both training time and predictive performance (accuracy) across the different boosting algorithms, providing insights into their relative strengths and computational efficiency.
As shown in Figure 13, the predictive performance of the four boosting models is generally good. Except for AdaBoost, the differences in accuracy among Gradient Boosting, XGBoost, and LightGBM are minimal, ranging from approximately 0.78 to 0.79. In contrast, AdaBoost achieves a notably lower accuracy of 0.717, representing a difference of more than 10% compared to the other methods.
Regarding training times, a higher degree of variability is observed, with some surprising results. AdaBoost completes training in 0.522 seconds, slightly below the average of the four models (1.591 seconds). Gradient Boosting requires 1.174 seconds, which is relatively close to the mean. The most striking values are found for XGBoost, which

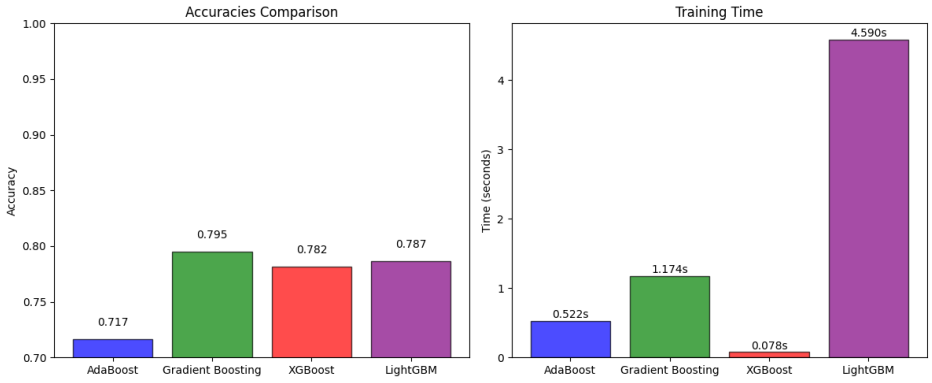


Figure 13: Accuracy and training time comparison of AdaBoost, Gradient Boosting, XGBoost, and LightGBM on a synthetic two-cluster dataset.

completes training in only 0.078 seconds — an unexpectedly short time given the complexity of the method, although this can be partially attributed to its highly optimized implementation. On the other hand, LightGBM requires more than 4 seconds to train, which is noteworthy considering that it is generally recognized for its fast training times. These observations highlight that while accuracy differences among most boosting methods are minor, training times can vary significantly depending on implementation details and algorithmic optimizations, emphasizing the importance of considering both performance and efficiency when selecting a boosting model. Taken together, these results demonstrate the remarkable superiority of XGBoost, which achieves comparable predictive performance to other boosting techniques while drastically reducing training time.

3.2 XGBoost

In this section, we focus exclusively on XGBoost, specifically we will evaluate its performance when combined with hyperparameter optimization using random search. To evaluate the effectiveness of hyperparameter tuning, random search is employed to explore different configurations of key XGBoost parameters, including the number of estimators, maximum tree depth, learning rate and subsampling ratios. Random search is particularly suitable in this context because it can efficiently explore high-dimensional hyperparameter spaces and often identifies near-optimal settings with fewer iterations compared to exhaustive grid search.

The experimental setup uses a synthetic data set specifically generated to test XGBoost under controlled conditions. This data set consists of two distinct clusters, allowing us to analyse both predictive performance (accuracy) and computational efficiency (training time).

The hyperparameter optimization process was carried out by evaluating a total of 30 parameter combinations using 3-fold cross-validation, resulting in the training of 90 models in total. Despite this extensive search, the total optimization time was only 22.42 seconds, which is remarkably low given the wide hyperparameter space explored. The best-performing configuration achieved an accuracy of 0.7576, demonstrating the

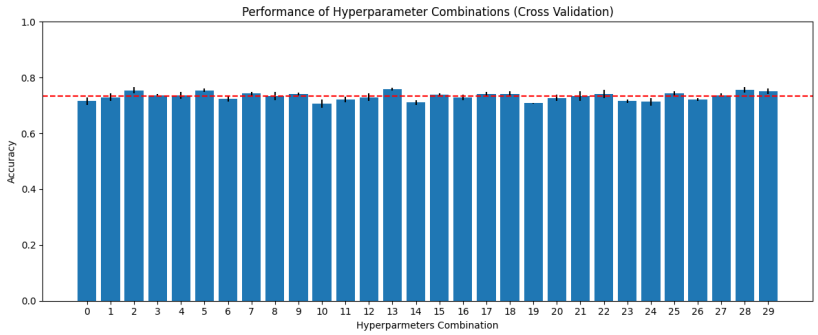


Figure 14: Distribution of accuracy scores across 30 hyperparameter combinations for XGBoost.

strong balance between efficiency and predictive performance. In addition, all tested configurations were analyzed to further assess the stability of the model. As shown in Figure 14, the performance scores are tightly clustered around the mean, indicating very limited variability across combinations. Moreover, the standard deviation across folds is minimal, confirming the high stability and robustness of XGBoost, which consistently maintains strong generalization capabilities regardless of the training subset used.

3.3 Income Classifier - XGBoost

In this section, we are going to use XGBoost in a more realistic context, we will employ the well-known "Adult" dataset from scikit-learn, also referred to as the Census Income dataset. This dataset contains information collected from the 1994 U.S. Census, the objective classification is to predict whether an individual's income exceeds \$50,000 per year based on demographic and employment attributes.
The dataset includes $48,842$ observations and 14 input features (both numerical and categorical), along with a binary target variable indicating the income class.

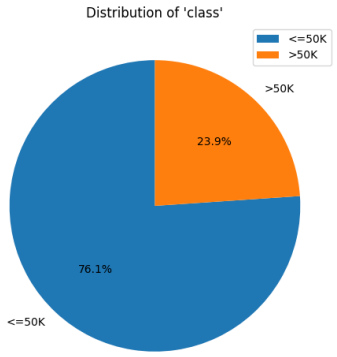


Figure 15: Class distribution of the target variable in the Adult dataset.

The first step in the analysis involves examining the class distribution of the target variable. As shown in Figure 15, the population earning less than $\$50,000$ represents

Table 1: Description of the variables included in the Adult dataset.

Variable	Type	Description
`age`	Categorical	Age of the individual.
`workclass`	Categorical	Type of employer.
`fnlwgt`	Numerical	Final sampling weight assigned to each observation.
`education`	Categorical	Highest level of education attained.
`education-num`	Numerical	Encoded numeric representation of education level.
`marital-status`	Categorical	Marital status.
`occupation`	Categorical	Type of occupation or job category.
`relationship`	Categorical	Relationship within the household.
`race`	Categorical	Race of the individual.
`sex`	Categorical	Gender of the individual.
`capital-gain`	Numerical	Monetary gain from capital investments.
`capital-loss`	Numerical	Monetary loss from capital investments.
`hours-per-week`	Categorical	Average number of working hours per week.
`native-country`	Categorical	Country of origin.
`income`	**Binary**	Indicates whether income $>$ \$50K or $\leq$ \$50K.

76.1% of the total, while only 23.9% earn more than $\$50,000$. Although this distribution indicates a certain degree of imbalance, it is not severe enough to warrant specific corrective measures. In practice, XGBoost is known to handle moderate class imbalances effectively through its internal weighting mechanisms.

In the second step, and as presented in Table 1, the dataset contains a considerable number of categorical variables. Since most machine learning algorithms—including XGBoost—require numerical input, these variables were transformed using the dummy encoding technique. This approach converts categorical variables into a series of binary columns, allowing the model to interpret them correctly without imposing ordinal relationships.

Additionally, a significant number of missing values were identified in the variables 'workclass', 'occupation', and 'native-country'. Given that all three are categorical, the mode imputation method was applied to replace missing values with the most frequent category in each feature. Finally, the variables 'capital-gain' and 'capital-loss', representing income from capital investments, were removed from the dataset due to their strong correlation with the target variable, which could otherwise lead to data leakage and an overestimation of model performance.

Model Training

For the training, validation, and testing process, the dataset was divided into three stratified subsets to preserve the original class distribution across all partitions. The resulting sample sizes were as follows: Training set with $27,351$ samples, Validation set with $14,653$ samples, and Test set with $6,838$ samples.

During training, early stopping was applied with a patience of 30 iterations to prevent overfitting by halting the optimization process once the model's performance on the

validation set stopped improving. Given that this is a binary classification problem, the logarithmic loss (log loss) was used as the evaluation metric, as it provides a continuous measure of the model's confidence in its predictions.

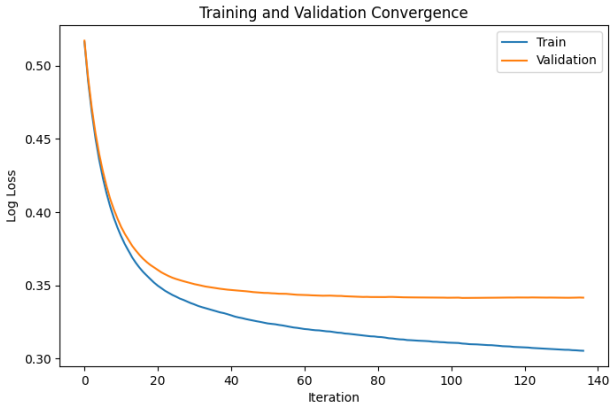


Figure 16: Convergence of the XGBoost model during training, showing the evolution of log loss across iterations for the training and validation sets.

As illustrated in Figure 16, the training process converged after 136 iterations, completing in a total of 2.8 seconds. The final results indicate a log loss of 0.30539 for the training set and 0.34166 for the validation set, reflecting a good balance between model fit and generalization capability.

Results and Discussion

The final results of the model demonstrate a high generalization capacity and predictive accuracy, achieving an accuracy of 0.8557 on the training set and 0.8443 on the test set. The minimal difference between both values indicates that the model does not suffer from overfitting, confirming a well-balanced learning process.

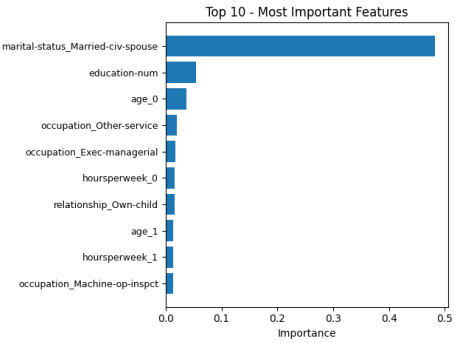


Figure 17: Feature importance derived from the trained XGBoost model.

When analyzing feature importance, as illustrated in Figure 17, it becomes evident that one variable stands out significantly from the rest: marital-status, specifically the

category married-civ-spouse. This label refers to individuals who are married to a civilian spouse, that is, not married to a member of the armed forces. Interestingly, this group exhibits a strong positive association with higher income levels, likely reflecting the socioeconomic stability associated with civilian households. The second most relevant feature is the education level, which also shows a clear positive relationship with income, confirming the model's ability to capture meaningful socioeconomic patterns in the data.

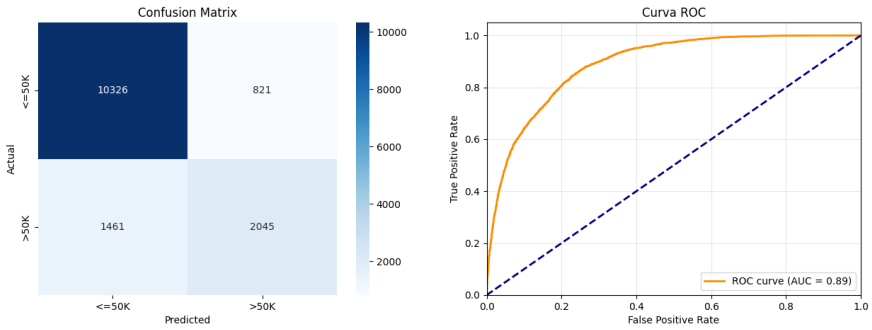


Figure 18: Model performance visualization showing the confusion matrix (left) and ROC curve (right).

From a performance perspective, the model achieves an AUC of 0.89, confirming its strong discriminative power. As shown in Figure 18, the ROC curve reveals a solid trade-off between sensitivity and specificity, while the confusion matrix indicates that most observations are correctly classified. Nevertheless, a slight tendency remains for the model to underpredict high-income individuals, classifying some of them as belonging to the lower-income group due to the moderate imbalance present in the dataset.

3.4 Project - Losses Prediction

The objective of this project is to predict the damage a hurricane might cause to a hotel in Kingston, Jamaica, in order to determine whether insurance should cover the damage, and if so, whether the payment should be partial or full. To achieve this, two XGBoost models will be developed: the first is a classification model to predict one of three categories (non-payable, partially payable, fully payable), and the second is a regression model to estimate the monetary value of the damage in dollars.

3.4.1 Data Processing

The available dataset includes various characteristics of past hurricanes affecting the region and the damaged caused to the hotel. To better reflect real-world impacts, enhanced features were generated using domain-specific knowledge. The following metrics were calculated:
These enhanced features were then crossed with the hotel damage records to build the dataset used for modeling.
Before developing the models, it is crucial to analyze both the distribution of classes and the distribution of numeric losses. The three classes were defined based on two

Table 2: Description of the custom variables created for hurricane characterization.

Variable	Description
`inverse_dist`	Inverse distance to the hotel (closer hurricanes have higher values)
`mws_weighted`	Maximum wind speed weighted by proximity
`mslp_weighted`	Minimum sea-level pressure weighted by proximity
`pressure_deficit`	Deviation of central pressure from mean sea-level pressure (1013.25 hPa)
`severity_index`	Ratio of wind intensity to distance, reflecting potential for damage
`risk_exposure`	Combined measure of wind and pressure deficits relative to distance
`wind_pressure`	Dynamic wind pressure based on wind speed
`rmw_ratio`	Relative size of the hurricane eye
`compound_impact`	Combination of severity index and wind pressure
`proximity_intensity`	Combined effect of proximity and wind intensity

thresholds: $\$15,000$ for partially payable and $\$115,000$ for fully payable damages. This categorization allows the models to distinguish between events that do not require insurance coverage, those that require partial compensation, and those that are fully payable.

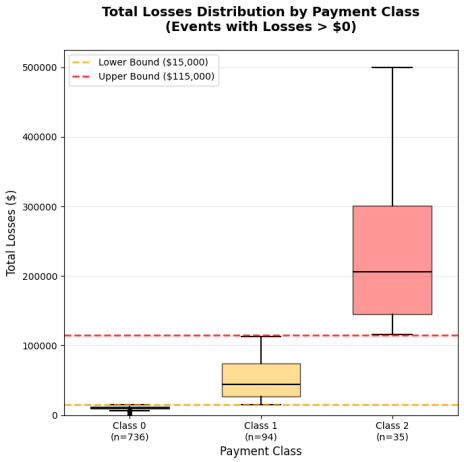


Figure 19: Distribution of hurricane damage classes for the hotel. Class 0 (Non-payable), Class 1 (Partially Payable) and Class 2 (Fully Payable).

As observed in Figure 19, the distribution of classes is highly imbalanced. The majority class, Non-payable (Class 0), contains 736 events with positive losses, representing almost all cases where no payment is needed. The losses in this class are relatively low, ranging from as little as $\$7.20$ up to $\$14,930.80$, with an average of approximately $\$10,330$ and a median around $\$10,496$. The narrow range of losses in this class indicates a well-defined set of minor events that typically do not trigger insurance claims.
The Partially Payable class (Class 1) includes 94 events with losses between $\$15,327$

and $112,655, showing significantly higher variability compared to Class 0. The average loss in this category is around $51,383, while the median is approximately $44,303. This discrepancy between the average and median reflects the presence of a few events with extremely high damages, which increases the overall variability. The distribution in this class suggests that the insurance payout may vary substantially depending on the intensity and proximity of the hurricane, making this a challenging category for classification.
Finally, the Fully Payable class (Class 2) represents the rarest events, with only 35 instances. Losses in this category are extreme, ranging from roughly $115,835 up to $500,087, with an average of $237,933 and a median of $205,758. The very high variance in this class highlights the unpredictability of catastrophic hurricanes and the potential for severe damage to the hotel. Despite the small number of events, accurately predicting this category is crucial for risk management and insurance planning.
Overall, this analysis demonstrates the significant imbalance and variability present in the dataset. Class 0 dominates the observations, while Classes 1 and 2 contain progressively fewer but more variable losses. These characteristics must be carefully considered when training both the classification and regression models, as they directly impact model performance, metric selection, and evaluation strategies.

3.4.2 Model Training

For the training, validation, and testing process, the data set was divided into three stratified subsets to preserve the original class distribution across all partitions. The resulting sample sizes were as follows: Training set with 80,354 samples, Validation set with 43,047 samples, and Test set with 20,089 samples.
During training, early stopping was applied with a patience of 50 iterations to prevent overfitting by halting the optimization process once the model's performance on the validation set stopped improving. Given that this is a multiclass classification problem, multi logarithmic loss was used as the evaluation metric, as it provides a continuous measure of the model's confidence in its predictions. In the case of the regression model the mean absolute error was used as the evaluation metric.

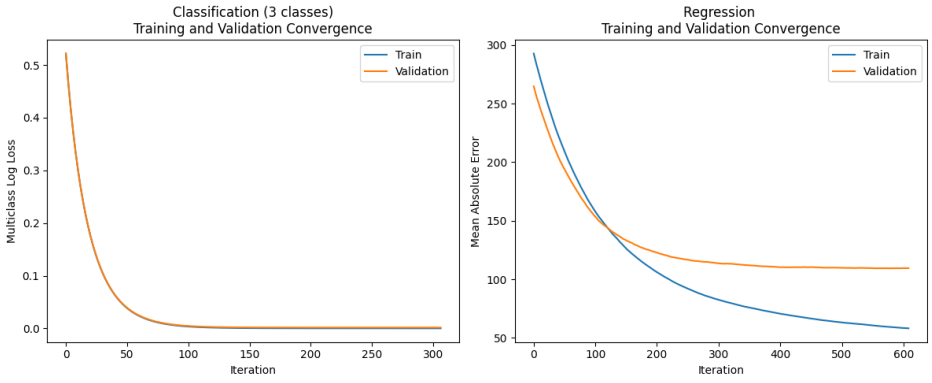


Figure 20: Convergence of the XGBoost classification and regression models.

As observed in the Figure 20 the convergence of the models differs substantially. The

classification model reaches convergence much earlier than the regression model, specifically at iteration 305 compared to iteration 607 for regression. Examining the convergence in detail, we see that for the classification model, the training and validation curves remain very close throughout the process. This indicates that the model generalizes effectively to the validation data, as confirmed by the final multi-class log-loss values: 0.00026 for training and 0.00181 for validation.
In contrast, the regression model shows a different pattern. The training and validation error curves are not as parallel. Initially, the model predicts the validation data better than the training data, and this behavior persists until approximately iteration 150. After this point, the reduction in validation error slows down considerably compared to the training error, resulting in an increasing gap between the two curves with each iteration. The final errors for the regression model are 58.1779 for training and 109.5144 for validation, reflecting the greater difficulty in accurately predicting continuous loss values and the presence of more variability in the regression task compared to the classification task.

3.4.3 Results and Discussion

The final performance of the classification model demonstrates that it is capable of making correct predictions on both the training and test sets. The model achieved a macro F1 score of 0.9883 on the training set and 0.5168 on the test set. While there is a notable difference between these values, this discrepancy can largely be attributed to the highly imbalanced number of samples across classes. Examining the F1 score per class confirms this: for Class 0, the F1 score is nearly 1, whereas for Class 1 and Class 2 the results are much lower, at 0.21 and 0.33 respectively, reflecting the challenge of accurately predicting the rare, high-damage events.
In the regression model, the differences between training and test performance are evident but less pronounced. The mean absolute error (MAE) is $60.15 on the training set and $106.94 on the test set, indicating that while the model struggles more on unseen data, it maintains reasonable generalization for continuous damage predictions.

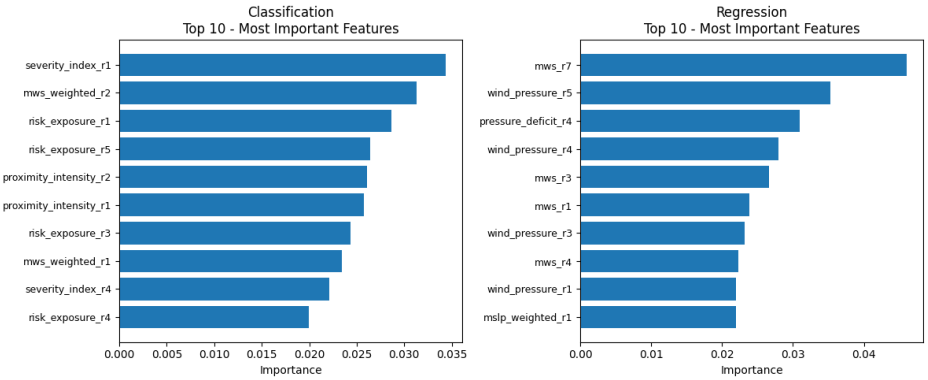


Figure 21: Feature importance derived from the trained XGBoost model.

Focusing in the Figure 21, the feature importance differ considerably between the two models. First, the varibles that each model consider most influential are almost com-

pletly different. For classification, the most important feature is the severity index at point R1, whereas for regression it is the maximum wind speed (MWS) at point R7. Despite these differences in ranking, the overall scale of importance is similar between the two models, and no single feature dominates excessively, suggesting that both models rely on a combination of multiple features to make predictions.

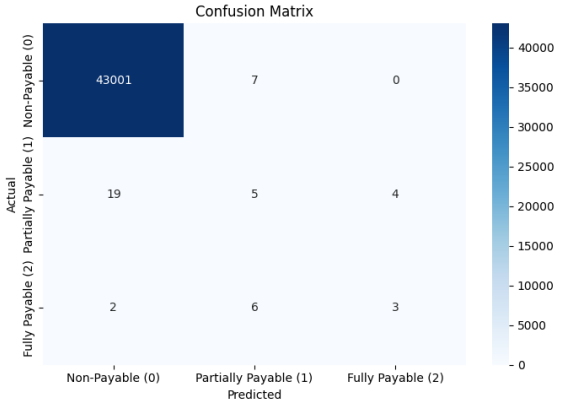


Figure 22: Confusion matrix for the classification model for losses prediction.

A closer analysis of the Figure 22, classification model predictions reveals behavior that aligns with expectations given the highly imbalanced nature of the dataset. The model tends to overwhelmingly predict the Non-payable class, which is the majority category, even in cases where this classification is not correct. This tendency is a direct consequence of the large disparity in sample sizes across the classes. While the model achieves high overall accuracy, its ability to correctly identify the rare events—Partially Payable and Fully Payable damages—is limited. In fact, only a very small number of predictions outside the majority class are made correctly, highlighting the challenges inherent in imbalanced classification tasks and emphasizing the need for careful metric selection and potentially for strategies such as class weighting or oversampling when training future models.

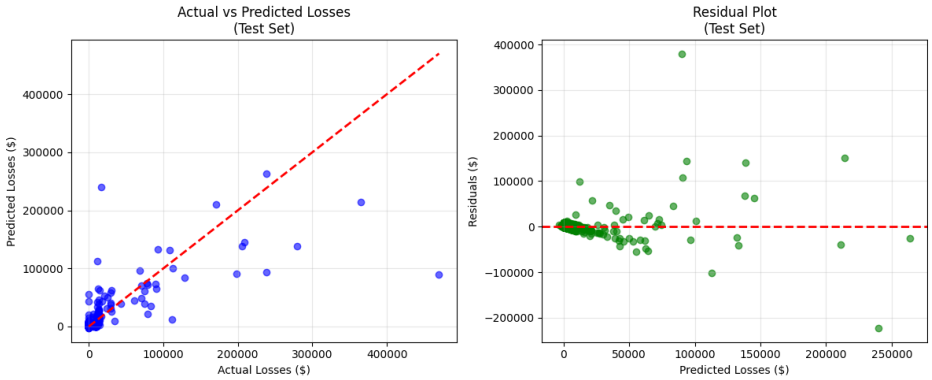


Figure 23: Predicted vs. actual losses and residuals for the regression model for losses prediction.

In contrast, the regression model exhibits a different pattern of errors, reflecting the continuous nature of the target variable. The lowest prediction errors occur for cases

with minimal losses, which are by far the most common in the dataset. As the magnitude of the losses increases, the model tends to produce larger deviations from the true values. An examination of the residuals shows that the errors are spread both above and below the actual values, with no systematic bias in one particular direction. However, extreme cases illustrate the limitations of the model: some predictions overestimate the actual losses by as much as $\$400,000$, while others underestimate them by more than $\$200,000$. These results underscore the difficulty of modeling rare and extreme loss events, which are inherently harder to predict due to their high variability and low frequency in the data.

4 Unit 8. Time Series Analysis

4.1 Store Sales Time Series

4.1.1 Preprocessing

The data set used consists of daily sales of different product categories across multiple stores over a five-year period, totaling 1,684 daily records. Each record corresponds to the total sales in dollars recorded on a specific day for a given store and category. Since the objective of this section is to perform a time series analysis, the daily data from all stores and categories were aggregated into a single daily time series, considering only the total sales per day across all stores. This transformation allows for the analysis of the overall sales dynamics over time and facilitates the identification of general patterns and trends.

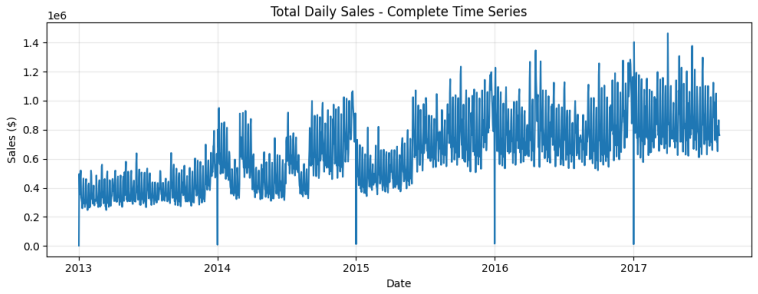


Figure 24: Daily time series of total sales across all stores.

As can be seen in Figure 24, the daily series presents data recorded for each day, showing a high density of measurements. However, the complexity and intrinsic variability of the data make it difficult for simple models to accurately capture the underlying behaviour. The daily series shows considerable daily fluctuations due to factors such as variations in demand, promotions, holidays, and specific events for each store or category, which generates noise in the analysis.

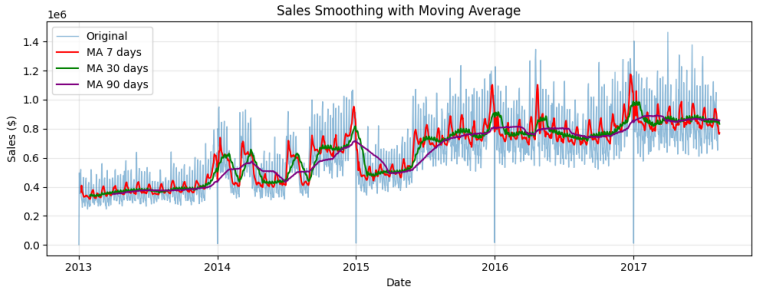


Figure 25: Smoothing of the daily series using moving averages of 7, 30, and 90 days.

To explore the structure of the data, smoothing was applied using a moving average, as shown in Figure 25. This procedure helps reduce short-term variability and highlight broader trends in the data. Despite the smoothing, the daily series still presents peaks and valleys that reflect the complex nature of daily commercial activity.

In order to simplify the analysis and facilitate modeling, it was decided to aggregate the daily series to a monthly level, calculating the total sales per month. This aggregation offers several advantages: it reduces daily noise, highlights more stable behavioral patterns, and allows models to identify trends and cycles more easily. As observed in Figure 26, the monthly series shows a smoother and more predictable pattern, with less abrupt cycles that reflect sales seasonality and sustained growth trends over time.

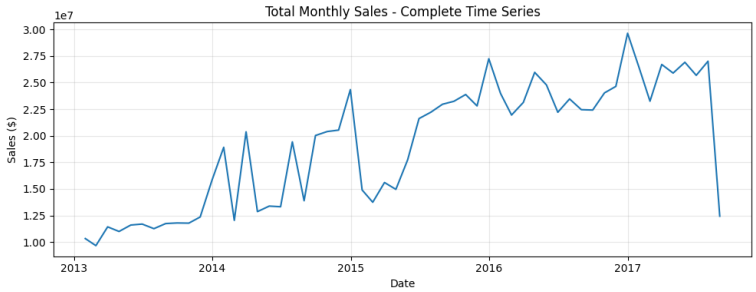


Figure 26: Monthly series of aggregate sales.

Additionally, an analysis of the time series trend was performed using linear regression methods applied to the monthly series. This analysis reveals sustained sales growth over the studied period, with an estimated slope of $293,677.37$ additional dollars per month, indicating that, on average, sales increase steadily each month. This information is crucial, as it allows for anticipating the future behavior of the series and serves as a reference for forecasting models that incorporate trend components.

4.1.2 Decomposition

Time series decomposition is a technique used to break down a series into its fundamental components in order to better understand its underlying patterns. By separating the observed series into trend, seasonal, and residual components. This decomposition is useful as it allows to address each component individually and improve forecast accuracy.

- **Additive Decomposition**: In an additive model, the observed value of the series is assumed to be the sum of its components:

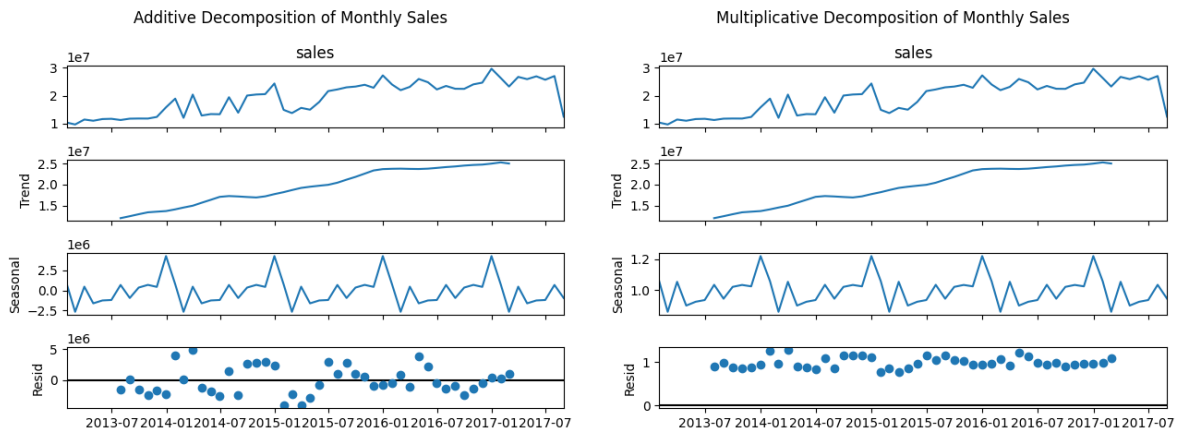
$$y_t = T_t + S_t + R_t,$$

  where T_t represents the trend component, S_t the seasonal component, and R_t the residual (or random) component.

- **Multiplicative Decomposition**: In a multiplicative model, the observed value is assumed to be the product of its components:

$$y_t = T_t \times S_t \times R_t.$$

  This approach is appropriate when the seasonal variations or residuals increase or decrease proportionally with the level of the series.



(a) Additive decomposition of the monthly sales series.

(b) Multiplicative decomposition of the monthly sales series.

Figure 27: Decompositions of the monthly sales series.

Following the previous data, both additive and multiplicative decompositions were applied to the monthly sales series. As shown in Figure **??**, the additive decomposition clearly separates the series into trend, seasonal, and residual components, capturing the underlying growth pattern and the repeating seasonal fluctuations. On the other hand, the multiplicative decomposition, illustrated in Figure **??**, models the series by considering proportional effects, which highlights how seasonal variations increase with the overall level of sales.
Despite the differences in approach, both decompositions successfully break down the time series into its main components, allowing a clearer understanding of the trend, seasonality, and residual variability. This confirms that either method can be effectively used to analyze and model the underlying structure of the sales data.

4.1.3 Forecasting

Exponential smoothing is a widely used technique for forecasting time series, which assigns exponentially decreasing weights to past observations. This approach allows the model to give more importance to recent data while still considering historical patterns. There are several variants of exponential smoothing, depending on whether the model accounts for trend and seasonality:

- **Single Exponential Smoothing**: This method only captures the level of the series and does not account for trend or seasonality. The forecast is calculated using a smoothing parameter α $(0 < \alpha < 1)$, which determines how quickly the weights of past observations decrease. High values of α give more weight to recent observations, while low values smooth the series more heavily.

- **Double Exponential Smoothing (Holt's Method)**: This method extends single exponential smoothing by capturing trend in addition to level. It introduces a second smoothing parameter, β $(0 < \beta < 1)$, which controls the smoothing of

the trend component. Like α, β balances sensitivity to recent changes versus stability of the trend estimate.

- **Triple Exponential Smoothing (Holt-Winters Method)**: This method further extends double exponential smoothing to account for both trend and seasonality It can be applied in additive or multiplicative form, depending on whether seasonal effects are roughly constant (additive) or proportional to the level of the series (multiplicative).

Following the approach used with the monthly sales data, the series was split into training and test sets to evaluate forecasting performance. The last twelve months of data (from September 2016 to August 2017) were reserved as the test set, while the first 44 months (from January 2013 to August 2016) were used for training. This split ensures that the model is evaluated on the most recent data while being trained on a sufficiently long historical period to capture trends and seasonal patterns.

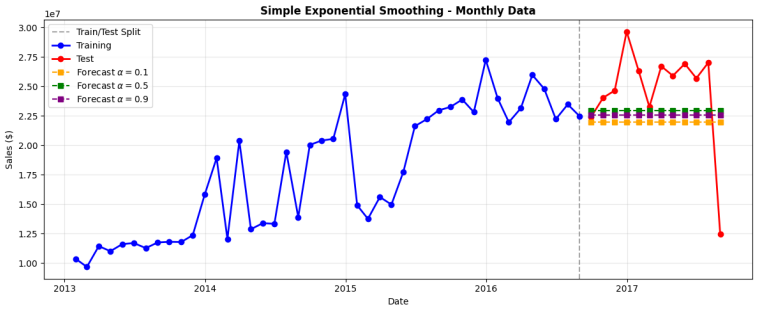


Figure 28: Forecast using Single Exponential Smoothing with various α values.

As shown in Figure 28, the use of Single Exponential Smoothing (SES) produces a relatively simple forecast, essentially a straight line starting from different points depending on the value of α. When α is high, the forecast is closer to the last observed point of the original series. For instance, with $\alpha = 0.9$ the prediction starts almost exactly where the original series ends. Conversely, with $\alpha = 0.1$, the forecast lies further below, reflecting a smoother and more conservative estimate that places less weight on the most recent observation.

In Figure 29, the results from Double Exponential Smoothing (Holt's Method) highlight the effect of accounting for trend. The predictions now incorporate the series' trend, and depending on the values of α and β, the forecast can have a positive or negative slope. Observing the prediction with lower α and β values, the slope is slightly positive, reproducing the overall trend of the series. While the other predictions exhibit a negative slope, this behavior occurs because the trend in the last five observed points (train set) is negative, even though the overall series trend is positive. This demonstrates how Holt's method responds not only to the long-term trend but also to recent changes in the series.

For the Holt-Winters Method, illustrated in Figure 30, the forecast captures both trend and seasonality, providing a more realistic prediction of the series. While the prediction is not perfect, as reflected in the evaluation metrics (MAPE: 17.39%, MAD:

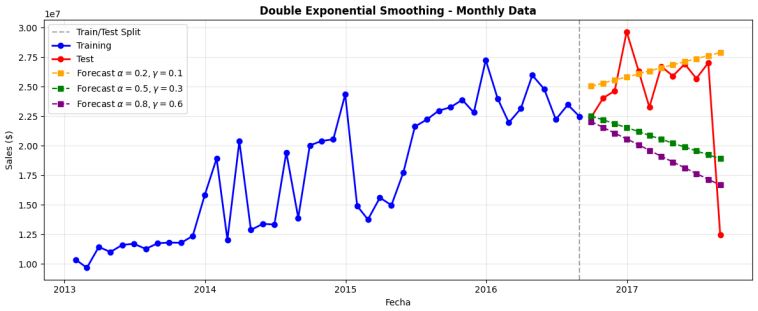


Figure 29: Forecast using Double Exponential Smoothing (Holt's Method) with various α and β values

$\$3,108,348.35$, RMSE: $\$4,807,563.2$), it already reflects the general pattern of the data. The elevated errors are primarily due to a sudden drop in the last point of the test series, which the model could not anticipate because this behavior was not observed in previous data. This anomaly is likely caused by incomplete data for the final month of the series.

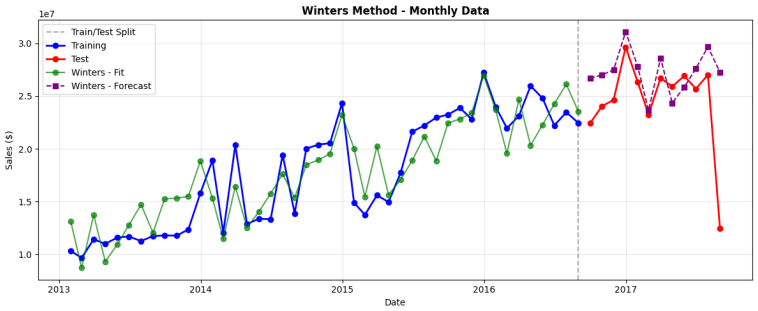


Figure 30: Forecast using Triple Exponential Smoothing (Winters Method).

As expected, the results for the Holt-Winters Method show the model's ability to capture both the trend and seasonal components of the series. The forecast provides a realistic representation of the underlying patterns, smoothing out short-term fluctuations while following the general upward trend and periodic seasonal variations.

4.2 ARIMA and SARIMAX

In this section, we introduce the analysis of time series data using ARIMA and SARIMAX. Time series analysis aims to model temporal dependencies in data, allowing us to forecast future values based on past observations. Unlike standard regression, time series models explicitly account for autocorrelation, trends, and seasonality, which are inherent in temporal data.

4.2.1 Autocorrelation

Autocorrelation Function (ACF) and Correlograms

Autocorrelation measures the degree of similarity between a time series and a lagged

version of itself over successive time intervals. It is a fundamental concept because most time series are not independent over time, past values often influence future values. For example, in financial data, yesterday's stock price is often correlated with today's price.

$$\rho_k = \frac{\text{Cov}(y_t, y_{t-k})}{\sigma^2}$$

The Autocorrelation Function (ACF) plots ρ_k for different lags k. Visualizing the ACF with a correlogram helps identify key patterns in the series. By examining the correlogram, we can detect whether seasonal components must be included in the model.

Partial Autocorrelation Function (PACF)
While the ACF shows the overall correlation between the series and its lags, it cannot differentiate between direct and indirect correlations. For example, if y_t is correlated with y_{t-2} because of the intermediate correlation with y_{t-1}, the ACF will reflect this combined effect. The Partial Autocorrelation Function (PACF) solves this by measuring the correlation between y_t and y_{t-k} after removing the effect of all intermediate lags. The PACF is particularly useful for selecting the order p in an autoregressive (AR) model, as it shows where the direct influence of past values cuts off.

4.2.2 ARIMA Models

Autoregressive (AR) Models
Autoregressive models predict the current value of a series as a linear combination of its previous values plus a stochastic error. An AR model of order p ($AR(p)$) is defined as:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2).$$

Here, the coefficients $(\phi_1, \ldots, \phi_p)$ quantify the influence of past observations. AR models assume that past behavior contains information about the future.

Moving Average (MA) Models
Moving Average models represent the current value as a function of past forecast errors, capturing short-term shocks in the series:

$$y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2).$$

MA models are particularly effective when the series is affected by sudden but temporary disturbances. The MA coefficients $(\theta_1, \ldots, \theta_q)$ determine how past errors influence current observations.

ARIMA Models
ARIMA models combine three key components to model time series data. First, the autoregressive (AR) component uses past values of the series to predict the current value. Second, the moving average (MA) component accounts for past forecast errors to adjust predictions. Finally, differencing (I for Integration) is applied to remove trends and achieve stationarity in non-stationary series.

This combination allows ARIMA models to capture both short-term fluctuations and longer-term patterns. The choice of the model parameters (p, d, q) is guided by analyzing autocorrelation and partial autocorrelation plots and testing for stationarity.

4.2.3 Seasonal ARIMA (SARIMA) Models

SARIMA models extend ARIMA to handle seasonal patterns in time series data. In addition to the autoregressive, moving average, and differencing components of ARIMA, SARIMA incorporates seasonal autoregressive, seasonal moving average, and seasonal differencing terms. These seasonal components allow the model to capture patterns that repeat over fixed intervals. By including seasonality explicitly, SARIMA improves forecast accuracy for series with predictable repetitive behavior.

SARIMAX Models
SARIMAX further extends SARIMA by allowing the inclusion of exogenous variables, which are external factors that can influence the series. By incorporating these additional predictors, SARIMAX can account for both the temporal and seasonal structure of the series as well as external influences, providing more accurate and robust forecasts for real-world applications.