MACHINE LEARNING PROJECT

Master in Data Science and Advanced Analytics

NOVA Information Management School

Universidade Nova de Lisboa

To Grant or Not To Grant

Group 48

Afonso Lopes, 20211540

Ana Isabel Duarte, 20240545

Diogo Fernandes, 20220507

Pedro Campino, 20240537

GitHub Repository

Fall Semester 2024-202

Group Member Contribution

In our group of five, we decided to split 500 points (5 x 100) among ourselves based on how much each of us contributed to the project. Based on everyone's overall contributions, we decided on the following distribution of points:

Afonso – 110 points

Diogo – 170 points

Isabel – 110 points

Pedro C. – 110 points

Pedro M. – 0 points

The project was divided into various sections, with contributions allocated as follows. It is important to note that the code and report parts for each section were completed by the same individuals:

- **Exploratory Data Analysis (EDA):** Isabel, Pedro C., and Afonso.
- **Preprocessing:** Isabel, Pedro C., and Afonso.
- **Pipeline Assembly:** Diogo and Afonso.
- **Model Development:** Diogo.
- **Open-ended Section:** Diogo.

Pedro M. didn't meaningfully participate in the project from the start. After the first handout delivery, where he made no contribution, we decided to give him another chance by assigning him a simple task to help him familiarize himself with the project's current state. Pedro M. delivered only a small portion, which was late and unusable for our work. Throughout the project, he continued to miss meetings and failed to deliver any other work that was assigned to him. His consistent lack of participation added extra pressure on the rest of us, as we had to take on additional responsibilities to cover for his absence.

Abstract

The New York Workers' Compensation Board (WCB) oversees various workers' benefits programs but relies on a manual, labor-intensive process for handling claims. This lack of automation leads to delays, increased workload, and a higher risk of errors. This project aims to address these challenges by developing a machine learning model to automate the classification of claims by injury type. Using a labeled dataset of claims from 2020 to 2022, which is inherently imbalanced due to certain types of cases being much more frequent than others, we employed a systematic approach to build, evaluate, and optimize predictive models.

We start by testing 5 models (Neural Networks, Decision Trees, K-Nearest Neighbors, Logistic Regression, and Naïve Bayes) to establish a performance. In an effort to maximize performance, we tested different types of ensemble models (Bagging, Boosting and Stacking). After selecting the model with the best performance, we tested different preprocessing techniques.

Our best-performing estimator is a Neural Network that achieved 0.41 ± 0.01 in training and 0.4 ± 0.01 in validation. It can predict very well two classes, non-compensation and temporary. But the others suffer from misclassification.

We attribute the model's limitations to the lack of meaningful features that effectively distinguish between classes. There were no performance improvements by adding more features to the model, by using various ensemble techniques, by oversampling minority classes or by using a One versus All framework.

These findings underscore the critical need for advanced feature engineering to enhance model performance. Future efforts should focus on extracting domain-specific features and exploring alternative data representations to better capture the nuances of claims data. By addressing these challenges, the WCB can significantly improve claim processing efficiency and decision accuracy.

Table of Contents

1. Introduction

The New York Workers' Compensation Board (WCB) administers and regulates workers' compensation, disability, volunteer firefighter, volunteer ambulance workers' and volunteer civil defense workers' benefits. The current process used by the WCB to identify claims is largely manual and labor-intensive. Staff members must manually process incoming claims, including reviewing and verifying documents, inputting data into the system, and performing case searches (U.S. Department of Labor, n.d.). The absence of automation means identifying valid claims requires significant time and effort, delaying resolutions and increasing the risk of errors. These inefficiencies directly impact both workers and insurers, with settlement durations in New York varying widely, sometimes taking several years (Law Offices of Mario S Crisafulli, 2023).Therefore, developing a new and more efficient approach is essential to improve the fairness and effectiveness of the claims process.

To modernize this process, adopting machine learning (ML) technologies offers a promising solution. ML has transformed the healthcare insurance sector by streamlining claim processing workflows, reducing operational costs, and facilitating the integration of data-driven approaches into healthcare insurance operations. Ensemble models such as XGBoost and Random Forest have been shown to excel in these predictive tasks, achieving high R-squared values and minimizing prediction errors (Parshuram & Joshi, 2024).

However, the inherent imbalance in claims data, where certain types of cases are much more frequent than others, presents a challenge to model performance. In an attempt to overcome this problem, we will leverage the One-vs-All (OVA) classification strategy along with oversampling techniques like SMOTE (Synthetic Minority Over-sampling Technique) and ADASYN (Adaptive Synthetic Sampling) in the open-ended section to tackle class imbalances and improve the final model predictive capabilities.

The present report is divided into six main sections. The first section, Exploratory Data Analysis, presents the key insights gathered from analysing the raw data. The second section, Pipeline Architecture, outlines the structure of pipeline developed to transform raw data into accurate predictions. In the third section, Data Preprocessing, we discuss the transformations applied, along with the decisions and assumptions made during this stage. The fourth section, Feature Selection, delves into the decision-making process and the final method chosen to select meaningful features. The fifth section, Model Development, describes the model assessment strategy adopted, compares the performance of various models and preprocessing techniques tested, and provides a critical analysis of the predictive capabilities of the final model. Finally, the report concludes with an analysis of the performance differences between the best models incorporating oversampling and the same models using the OVA strategy.

2. Exploratory Data Analysis

Throughout our exploratory data analysis, we gathered the following insights:

Invalid Entries: There are a total of 19445 claims that contain every variable missing apart from Claim Identifier and Assembly Date, including on our target variable. Those claims need to be removed from our dataset

"**Birth Year" and "Age at Injury" 0 Values as Placeholders:** Computing the Birth Year through Age at Injury and Accident Date, the obtained values only had a correspondence of 100% with the Birth Year registered column when missing or 0 values were excluded on the first two variables. Being 0 the only value for which this correspondence didn't happen; we draw the conclusion that those are placeholders for missing values.

Missing Values

IME-4 Count: It contains many missing values. However, our research led us to the conclusion that those correspond to medical examinations not counted (meaning they would be equal to 0).

Datetime Variables: More than half of the First Hearing Date values were not registered, meaning that most First Hearings were not scheduled; C-3 Date also has more than half of the values missing, yet we don't know the nature of those: it could be due to submission errors or incomplete forms, for instance. C-2 Date and Accident Date have the least number of missing values (when we exclude the Assembly Date, that has none).

Code Features, Average Weekly Wage and Birth Year: There are some missing values on these features, but they appear in small proportion (up to 5% approximately). However, it is important to keep in mind that Birth Year and Age at Injury have 0 values as placeholders, as mentioned above.

Datetime sequences: There are events that don't follow the expected time sequence (as dates registered earlier than the Accident Date).

Cardinality: Some variables have high cardinality. However, for Code Features it's possible to group their values into larger categories based on information discovered through research. OIICS Nature of Injury Description and WCB Decision have respective cardinalities of 0 (all missing values) and 1, meaning they don't provide any information for our dataset to make a distinction between results on the Claim Injury Type.

Redundancy: For Code Features and their respective Code Descriptions, they always provide the same intel, which makes them redundant. Birth Year and Age at Injury are also highly correlated.

Presence of Outliers: Especially Average Weekly Wage and IME-4 Counts have many values detected as outliers. Through extended analysis, we could also verify that those variables contained extremely high values in comparison to the others.

Pairwise Relations with Claim Injury Type

The variables with visible impact on our target variable were especially Age at Injury/Birth Year, Average Weekly Wage, IME-4 Count and Attorney/Representative.

Age at Injury/Birth Year: The older the individuals who file the claim are, the larger the tendency for the Claim Injury Type to be more severe (especially death)

Average Weekly Wage and IME-4 Count: The insurances tend to provide higher benefits and have more medical examinations done, the more severe the Claim Injury Type is.

Attorney/Representative: Most of the severe Claim Injury Types (from 5. PPD SCH LOSS to 8. DEATH) need an Attorney/Representative, while most of the others don't.

Nonetheless, some variables are likely to have a significant impact on the target variable. However, this relationship may not be apparent due to their high cardinality. For others, their impact might be more indirect.

3. Pipeline Architecture

We adopted a pipeline architecture. In addition to enhancing code clarity and improving organization through modularity, this approach offers several advantages: it prevents data leakage during model evaluation, particularly with cross-validation methods beyond the simple hold-out approach; ensures reproducibility by consistently applying the same steps across datasets and minimizing transformation errors; simplifies hyperparameter tuning by treating the entire workflow as a unified estimator, streamlining grid search and cross-validation processes for easier model comparison; and seamlessly integrates data preprocessing with final predictions. Figure 1 (**Error! Reference source not found.**) illustrates the adopted pipeline, which c omprises three main steps: data preprocessing, feature selection, and an estimator.

4. Data Preprocessing

The data preprocessing step of the pipeline consists of a *ColumnTransformer* with three sub-pipelines, each tailored to a specific feature type: numeric, categorical, and binary. This is followed by a *FeatureUnion* to ensure consistent scaling across all variables.

4.1 Numeric Pipeline

Missing Value Imputation

Missing values were imputed using domain logic informed by EDA insights:

Birth Year and Age at Injury: 0 values, identified as placeholders during EDA, were recalculated using the relationship between these variables and **Accident Date**. This improved consistency while addressing missing data.
IME-4 Count: Missing values were interpreted as 0, aligning with the EDA observation that they likely represent no recorded medical files.
Date Variables (First Hearing Date and C-2 Date): Missing dates, identified as indicating unrecorded events, were replaced with a placeholder (e.g., January 1, 2100) to retain interpretability.
Other Variables: Remaining missing values were imputed with the median to minimize bias and preserve feature distributions. The median was chosen for its robustness to outliers, ensuring a consistent and representative central value for imputation.

Anomaly Fixing

EDA revealed key anomalies that were addressed to ensure logical consistency:

Birth Year and Age at Injury: 0 values were recalculated or flagged using their relationship with **Accident Date**, reducing outlier impact.
Event Dates: Dates occurring before the **Accident Date** were corrected using median values, aligning with EDA recommendations for validating date ranges.

Feature Engineering

Temporal features were derived by calculating the time intervals between **Accident Date** and key events, capturing the progression of claims over time. Age-based features, such as the claimant's age at the time of the accident and its interactions with variables like weekly wage, were created to reflect the nuanced impact of age on claim characteristics. Additionally, **Accident Date** was decomposed into year, month, and quarter components to account for potential seasonal trends, providing the model with richer temporal insights.

Outlier Treatment

Outlier treatment was guided by EDA findings to ensure data integrity while minimizing the impact of extreme values. Key variables such as **Birth Year**, **Age at Injury**, and **IME-4 Count** exhibited outliers, often linked to placeholder values (e.g., 0). These were addressed through targeted imputation and recalculations. Additionally, two capping approaches were considered: the Interquartile Range (IQR) method and the percentile-based method. The IQR method is robust to extreme values but tends to flag moderately unusual values, which can lead to over flagging. The percentile method, while less robust and influenced by extremes, isolates only the most extreme deviations using thresholds (e.g., P1 and P99), making it suitable for targeting true extremes.

4.2 Categorical Pipeline

The categorical pipeline includes procedures for imputing missing values, fixing anomalies, engineering new features, and encoding the variables.

Anomaly Fixing: Using the custom transformer, *AnomalyFixer_Categorical*, we removed the column “OIICS Nature Of Injury Description”, because it contained 100% of the values missing.

Feature Engineering

During EDA, we identified some categorical variables with high cardinality and suggested grouping them in order to reduce it, if possible. We created new variables keeping that in mind.

Supra Code Features (using respective Code Features): These features have larger groups (discovered through our research in EDA) which allowed us to aggregate them effectively.

USA Region (using Zip Code): Looking up the geographical regions in the USA, we found out that the Zip Codes have the first number corresponding to a scale from eastern (0) to western (9) regions. So, to reduce even further the cardinality, we grouped them into 3 regions: Eastern (0-4), Center (5-8) and Western (9). Any Zip Code that didn’t have the American format or wasn’t present in the American Zip Codes database that we found, we attributed the category ‘Not USA’. We also decided to make the missing values its own category: “Missing”, as we considered it could be relevant to our target variable (so this variable isn’t included in the topic: Missing Value Imputation)

Season of Accident (using Accident Date): Since datetime variables cannot be directly used by algorithms, we decided to extract the month from the 'Accident Date' and, to reduce cardinality, group the months by season. We considered that the time of year when the accident occurred could influence the claim and the overall procedure.

Missing Value Imputation

We replaced the missing values on the code features with their respective modes because we didn’t find any underlying pattern on such cases and they represent a small percentage of our dataset, so in doing this we preserve the distribution of the data with simplicity.

Variable Encoding

On the first version of the pipeline, we used frequency encoding because it efficiently captures the distribution of categories in the dataset in a simple and efficient way. Then, we proceeded to create an alternative version of the pipeline where, besides frequency encoding, the normalized frequency encoding and ordinal encoding are also tested so we can verify which one works the best for our predictions.

4.3 Binary Pipeline

The binary pipeline effectively transforms different variables into binary features, creates additional informative binary flags, and handles missing values, all of which are essential steps to prepare the data for the different machine learning models that will be used later on.

Binary Variable Transformation: The *BinaryFeatureTransformer* is responsible for converting categorical variables into binary representations. Specifically, the transformer handles variables such as Gender, where it creates a binary column for "Male" (1 for "M" and 0 for other values like "F", "X", or "U"). In this case, the decision to create a binary column for "Male" is based on the distribution of the different categories seen on the EDA, where "M" and "F" are dominant. Given the limited representation of smaller categories "X" and "U," combining them ensures sufficient statistical significance while preserving key patterns in the data. For other

categorical variables with binary-like responses, such as 'Y' and 'N,' the transformer generates binary flags (1 for 'Y' and 0 for 'N,' with NaN for any other values), ensuring consistency across binary features.

Feature Engineering: The *Binary_Feature_Engineering* step focuses on creating new binary features from key date and age-related variables. It derives binary flags for missing dates, such as the presence or absence of values in columns like Accident Date and C-2 Date. It also creates flags indicating whether certain dates fall on weekends (e.g., Accident Date or C-2 Date) and whether Accident Date occurred during the COVID-19 period.

Missing Value Imputation: Based on the EDA, the *SimpleImputer* with the "most_frequent" strategy is the optimal choice for handling missing values in the binary pipeline. This strategy is particularly relevant given the unbalanced nature of the binary features. For example, the COVID-19 Indicator variable is highly unbalanced, with most values being 0 and a much smaller proportion being 1.

4.4 Scaling

Using the pipeline framework and cross-validation prevented us from examining the distributions of features and outliers across folds, making it difficult to select an appropriate scaling method a priori. As a result, we applied the *StandardScaler* in the baseline pipeline, given its general applicability and ability to standardize features by centering them around zero and scaling to unit variance.

During model optimization, we explored alternative scaling methods that could potentially better align with the characteristics of the data. We tested the *MinMaxScaler*, which scales features to a fixed range and is particularly suited for algorithms sensitive to feature ranges, and the *RobustScaler*, which uses the median and interquartile range to handle outliers effectively. The latter was included in this project, because, despite incorporating outlier treatment in the pipeline, we cannot definitively confirm whether outliers remain in the data.

5. Feature Selection

To ensure the feature selection process remains independent of specific model types or assumptions about model behavior, and to better integrate with the pipeline workflow, we opted for a model-agnostic approach. Given the dataset size, we discarded wrapper methods due to their high computational cost and reliance on models like RFE, which require feature importance measures. Embedded methods were also excluded to avoid limiting the analysis to models with built-in regularization techniques. Instead, we employed filter methods.

Initially, we considered statistical methods such as ANOVA for numerical features and Cramér's V (For more details, refer to Appendix D) for categorical features, given the categorical target. However, the chi-squared assumption underlying Cramér's V, requiring observed values of at least 5, was violated. We then explored Shapely values for feature importance (For more details, refer to Appendix D), but the need to fit the model during preprocessing and again during training, proved to be computationally expensive.

Therefore, we opted for mutual information. This method provides a measure of the amount of information that one variable has about another variable (Vergara & Estévez, 2014). It has three main advantages: it captures both linear and non-linear relationships (Ross, 2014); makes no assumptions on data distributions; and quantifies the shared information between variables, enabling the use of a threshold to select the most relevant features for the target (For more details, refer to Appendix D).

The scikit-learn library (Pedregosa et al., 2012) implementation has two important parameters, the number of neighbours and specification of inputted discrete features. To estimate the best number of neighbors, we performed a grid search optimizing for the F1 macro score, testing values from 5 to 150 in increments of 5. The optimal choice was 35 neighbors, determined by the highest validation score and a closely matching training score, with no evidence of overfitting (Figure 2, Appendix A). Due to some pre-coded

transformers used during preprocessing operating with NumPy arrays and losing column name information, we were unable to pass the column names of our discrete features to the feature selection transformer. As a result, the discrete variables parameter was set to "auto," which discriminates between features based on density and sparseness.

To select features, we had two options: selecting the top k features with the highest mutual information values or selecting features based on a specified percentile of values. We opted for the latter, considering it less restrictive and more flexible, as it allows the number of selected features to adjust dynamically based on the dataset's characteristics in each fold, making the process more robust. As we will see below, multiple percentile thresholds were explored during model selection to identify the optimal balance between feature inclusion and model performance.

6. Model Development

6.1 Model Assessment Strategy

To rigorously evaluate model performance and ensure reliable results, we used grid search with embedded cross-validation. This approach enables hyperparameter optimization while avoiding data leakage and providing robust performance estimates. *StratifiedKFold* was chosen as the cross-validation method to ensure that the class distribution is preserved across folds. Although in literature the most common numbers of folds are 5 or 10 (Aning et al., 2021), we opted for 3 folds. We acknowledge that this is not optimal because it forces the train dataset to be ≈66.7% and validation ≈33.3%. Providing less training data can limit the model's ability to learn complex patterns, potentially reducing generalizability. This trade-off, while not ideal, was necessary to balance computational constraints and ensure a feasible cross-validation strategy. The optimization scoring chosen was the F1 macro, as it provides a balanced evaluation of precision and recall across all classes attributing the same weight to each. This makes it particularly suitable for imbalanced multiclass classification problems like the one presented by this project.

Our approach, summarized in Figure 3 (Appendix A), consisted of four main steps. First, we evaluated various baseline models and identified suitable candidates for ensemble methods. In the second step, we tested and optimized the most promising ensemble techniques. Third, we implemented stacking ensembles by combining optimized baseline and ensemble models in different configurations to enhance performance. Finally, we selected the top two models and refined them further by experimenting with various preprocessing techniques to maximize their performance.

The model selection process prioritized the highest mean F1 validation score, ensuring that training performance remained comparable to avoid overfitting. In the event of a tie, the simplest model was chosen to optimize computational efficiency, adhering to the principle of Occam's razor.

6.2 Candidate Models Comparison and Selection Process

We start by testing 5 different models with basic hyperparameters to obtain a performance baseline. The models included were Neural networks (NN), Logistic regression (LR), K-nearest neighbors (KNN), Decision Trees (DT) and Naïve Bayes (NB). For LR we set the class weight to balanced because this mode uses the values of the target variable to automatically adjust weights inversely proportional to class frequencies in the input data, potentially improving performance in unbalanced datasets. For KNN we used the best K selected during testing for feature selection. We used Manhattan distance because it measures absolute differences between feature values, which aligns well with categorical data where features can only be equal or different. We also applied distance-based weighting to increase the influence of closer neighbors, which helps address class imbalance by allowing minority class samples to have stronger voting power when they are the nearest

neighbors. For DT we also used the class weights set to balanced for the same reason in LR and set a max depth of 15 to prevent overfitting. The best models were NN (0.39 ± 0.02 train, 0.37 ± 0.01 validation) and DT (0.46 ± 0.01 train, 0.34 ± 0.03 validation, fig 4 and table 1, Appendix B). Even with the prepruning, DT exhibits moderate overfitting, suggesting that a lower max depth might be beneficial. During this phase we also tested the impact of the percentile on the feature selection method. Testing feature selection thresholds (50, 70, and 90 percentiles) showed no significant performance differences for most models, except for NB, which performed best with 50. This possibly hints that the current features are not providing important information for the models, since adding more doesn't have an impact on performance. For this reason and because it's less computationally expensive to train models with fewer features, we proceeded further testing by setting the 90-percentile threshold.

Based on the best-performing models identified in the previous experiment, we proceeded to evaluate their performance within ensemble frameworks, specifically Random Forest (RF), Gradient Boosting (GB), Bagging, and AdaBoost, to leverage ensemble methods' ability to improve predictive performance and reduce overfitting by combining multiple models. For RF, we tested different combinations of maximum depth (5 and 10), minimum samples required for a split (2 and 5), and the number of estimators (100 and 200). For GB, we explored various combinations of learning rates (0.05 and 0.1), maximum depths (5 and 10), and the number of estimators (50 and 100). For Bagging, we used a Multi-layer Perceptron (MLP) as the base estimator with the same parameters as before. For AdaBoost, we tested the base stump approach with learning rates of 0.05 and 0.1, as well as 50 and 100 estimators. The performance of the ensemble frameworks varied significantly across methods (fig 6, Appendix B). RF showed better performance with deeper trees (10), but there was no difference with the rest of the parameters tested. The GB exhibited similar validation scores independent of the parameter configuration, although the models with deeper trees (10) overfitted. Bagging, using MLP as the base estimator, achieved similar results to GB without overfitting. Nevertheless, these scores haven't surpassed the scores obtained with the regular MLP configuration, indicating limited improvement from ensemble aggregation. In contrast, AdaBoost consistently underperformed across all tested configurations (table 2, Appendix B).

Since Bagging with MLP estimator and GB showed the most promising results (0.4 ± 0.01 train, 0.38 ± 0.02 validation and 0.52 ± 0.03 train, 0.4 ± 0.01 validation, respectively), we decided to optimize the hyperparameters of the base MLP estimator hoping to improve the Bagging results and expand the search space of the GB hyperparameters, to keep validation performance while controlling overfitting. The regular MLP achieved the highest validation performance observed to date, with a training score of 0.41 ± 0.01 and a validation score of 0.4 ± 0.01, as reported in Table 3 (Appendix B), following hyperparameter optimization. The parameters tested were, activation function, network architecture, initial learning rate, learning rate decaying function and solver.

Following these adjustments, the retrained Bagging model, now utilizing the optimized MLP estimator, obtained the exact same score as the previous version, as detailed in Table 4 and figure 7 (Appendix B). Similarly, the expanded Gradient Boosting hyperparameter search version obtained a slightly lower validation score (0.38 ± 0.01) but successfully controlled overfitting (0.4 ± 0.03 on the training set), primarily by limiting the amount of data used to train each tree to 50% (table 5, Appendix B). Ensemble methods are expected to improve predictive performance by combining multiple models. However, the lack of improvement observed in the scores, coupled with the earlier finding that increasing the number of features does not affect performance, likely reflects a deficiency of additional meaningful information in the current feature set, limiting the potential benefits of ensemble techniques.

To further explore ensemble techniques and test the hypothesis above, we implemented stacking methods, combining predictions from multiple base models and the best ensembles. The goal was to leverage

the complementary strengths of these models by using a meta-learner to optimize the final predictions. Logistic regression was chosen as the meta-learner because its simplicity complements the complexity of the base learner models. The C values [0.1, 1.0, 10.0] were selected to explore the trade-off between underfitting and overfitting, with smaller values promoting stronger regularization to avoid overfitting and larger values allowing greater flexibility to reduce underfitting. *Saga* was added as an alternative solver because it achieves faster convergence on larger datasets. The selection of base learners was guided by combining top-performing models (NN and DT) with the best ensemble methods (Bagging with NN and GBoost), while also including LR and NB in some configurations to introduce diversity due to their distinct characteristics (and as respectful nod to the fans). A comparison of the validation scores is presented in figure 8 and table 6 (Appendix B). On average, all combinations performed worse than the previous models, the best stack was Bagging with NN as an estimator (0.33 ± 0.08 on train and 0.31 ± 0.07 on validation). These results support our claim that the issue lies in the lack of meaningful features that can effectively distinguish between classes, rather than the models' ability to capture complex patterns in the data. If the issue were the latter, we would expect ensemble models to outperform the base models, and stacking models to achieve at least comparable performance to other ensemble methods.

The last step we took was to attempt improving our two best-performing models, NN and GBoost, by experimenting with different preprocessing techniques. These included feature scaling, alternative methods of outlier detection, and testing various encoders for categorical features. The scalers tested were those previously discussed in their respective subsection. The other outlier method tested was the percentile approach, which could be more suitable than the IQR depending on the data distributions because it identifies outliers based on specified percentiles, making it more adaptable to skewed distributions and less influenced by extreme values. For the encoders, we tested ordinal encoding and a variation of frequency encoding, where the values were normalized instead of using absolute frequency counts. The best combination was NN with Minmax scaler, percentile method for outlier detection and frequency encoder (0.41 ± 0.00 train, 0.40 ± 0.02 validation, table 7, Appendix B). Some combinations with GBoost achieved the same validation score but showed signs of overfitting, as evidenced by significantly higher training scores compared to validation scores.

In conclusion, the NN model with 2 layers of 30 neurons, a tanh activation function, a constant learning rate starting at 0.01, and the SGD solver, combined with the MinMax scaler, the percentile method for outlier detection (1st and 99th percentiles), and the frequency encoder, proved to be the best-performing estimator. This model reached 0.41 ± 0.01 in training and 0.4 ± 0.01 in validation and it was refitted on the whole dataset.

6.3 Final Model Evaluation

The classification report (Table 8, Appendix B) reveals significant discrepancies in the model's performance across the different classes. The model performs well on Non-Compensation and Temporary claims, the two largest classes in the dataset, likely due to their size providing the model with sufficient examples during training. The F1 scores for these categories are among the highest, reflecting a balance between precision and recall. For PPD SCH LOSS, the model shows average performance, with relatively balanced precision and recall, indicating moderate success in predicting this class. The Cancelled and Death classes present notable challenges. While these categories achieve reasonable precision, their recall values are much lower, indicating that the model correctly predicts some cases but fails to capture the majority. This suggests a high number of false negatives. The most critical failures occur with the PPD NSL and PTD classes, where the model achieves a recall of 0. This indicates that these classes are not being identified at all, resulting in a very high number of false negatives. These failures are likely due to the severe class imbalance, with these categories having the smallest representation in the dataset. Notably, despite having the second lowest number of cases, the Death class performs better than PPD NSL, which has the third lowest representation. A possible justification for this could be that the Death class exhibits more distinct features or patterns that make it easier to separate from

other classes, whereas PPD NSL might share feature overlaps with other categories, making it more challenging for the model to correctly identify.

Considering the confusion matrix (Figure 9, Appendix B), Cancelled and Non-Compensation, show considerable overlap, as 52% of Cancelled cases are misclassified as Non-Compensation. Similarly, the Med Only class is predominantly misclassified as Non-Compensation (53%) and Temporary (35%). The Death class is frequently confused with Temporary (47%) and Non-Compensation (14%). PPD SCH LOSS also exhibits challenges, with 38% of its cases being misclassified as Temporary. The worst-performing classes, as previously discussed, are PPD NSL and PTD, which are overwhelmingly misclassified as Temporary, with 86% and 90% of their cases respectively being assigned to this class. These results suggest that the model defaults to this class when uncertain.

During the model selection phase, we hypothesized that the lack of improvement in model performance when testing various models might be attributed to the absence of meaningful features capable of effectively distinguishing between classes. The current analysis corroborates this hypothesis, demonstrating that the model struggles to discriminate between certain classes and that class imbalance significantly impacts the predictive capabilities of the models. Notably, the Death class achieves a higher F1 score despite having fewer cases than the PPD NSL class, further supporting our initial hypothesis. These findings suggest that the suboptimal performance likely results from a combination of these two factors.

7. Open-Ended Section - Overcoming Data Imbalance

As previously stated, we have identified two main factors that may potentially hinder the model's performance: data imbalance, which can lead to biased predictions favoring the majority class, and the absence of meaningful features capable of effectively distinguishing between classes. The main goal of this section is to address the data imbalance problem, ensuring better classification of underrepresented classes and ultimately enhancing the model's overall performance. We tested two different techniques: One-versus-all (OVA), which simplifies multi-class problems into multiple binary classifications that better handle imbalances in each class individually, and oversampling with SMOTE and ADASYN, which artificially generates synthetic samples to increase the representation of minority classes (for detailed information, refer to Appendix D). For testing, we evaluated our best-performing model both on its own and in an OVA configuration, with and without oversampling. In both oversampling techniques we set the sampling strategy to *not majority*, which resample all classes but the majority class. We tested two different numbers of neighbors, 5 (the default) and 10 to evaluate whether the augmentation improved performance.

The results indicate no overall performance improvement using either oversampling technique (Table 9, Appendix C). The NN model without OVA performed better without oversampling than with it (Figure 11, Appendix C). With SMOTE, a lower number of neighbors yielded better results, whereas the choice of neighbors made no significant difference with ADASYN. The NN model with OVA demonstrated similar performance with and without oversampling (Figure 10, Appendix C), although it scored slightly lower with ADASYN when using 5 neighbors. Regardless of oversampling, the OVA model was prone to overfitting, with an average score difference of more than 0.1 between the training and validation sets.

The results of this experiment suggest that oversampling techniques and the OVA configuration did not provide substantial benefits in addressing the challenges posed by data imbalance in our dataset. Consequently, it is reasonable to conclude that the primary limitation preventing significant improvement in the model's performance during development and selection lies in the absence of features capable of effectively distinguishing between classes. This emphasizes the critical role of feature engineering in creating

a more robust and discriminative model, as addressing class imbalance alone appears insufficient to overcome the underlying challenges.

8. Conclusion

This project addressed the challenge of automating claim classification for the WCB by systematically exploring various machine learning models and preprocessing techniques, culminating in a model designed to classify claims based on key characteristics of the bureaucratic process.

Our analysis revealed the inherent difficulty in achieving high predictive performance due to significant class imbalances and possible absence of features capable of effectively distinguishing between claim types. Despite testing a variety of preprocessing techniques, feature selection methods, and advanced ensemble approaches, our final model—a Neural Network with optimized preprocessing—achieved modest success, particularly for the largest claim categories. However, it struggled to generalize across underrepresented classes, highlighting the critical role of data quality and feature engineering and the importance of integrating domain-specific knowledge to enhance the feature set.

Regarding the project limitations, the model selection process could have been more robust by increasing the number of cross-validation folds to five, aligning with the heuristic of an 80% training and 20% validation split. Expanding the hyperparameter search space to examine interactions between parameters and their influence on model performance might have also provided deeper insights, albeit at the cost of greater computational effort. While adopting the pipeline architecture streamlined experimentation and ensured data safety, it introduced certain challenges. Specifically, it restricted access to split indexes, making it difficult to generate classification reports and confusion matrices during model selection. These tools could have provided valuable insights into model performance and patterns of misclassification. Moreover, debugging and controlling data transformations within the pipeline proved more complex.

Another potential improvement would have been to switch the project framework, given that our best model is a neural network. Frameworks like PyTorch or Keras offer greater flexibility in configuring network architectures and more efficient methods for hyperparameter optimization, such as Hyperband. Hyperband dynamically allocates resources based on model performance: it starts by training all configurations with minimal resources, eliminates underperforming ones early, and allocates more resources to promising candidates. Additionally, these frameworks support GPU acceleration, significantly speeding up training and allowing for the exploration of a broader range of configurations.

Finally, after obtaining the final model results, a further improvement would have been to revisit the exploratory data analysis (EDA) phase. By examining data patterns that could help distinguish between misclassified classes—especially between temporary and other underrepresented classes—we could have refined data preprocessing and feature engineering. This iterative approach would have created more information-rich features, enhancing the model's ability to differentiate between classes effectively.

Future work should focus on creating features that better capture the complexities of claims data while addressing the trade-off between precision and recall. Higher precision reduces false positives, ensuring cases aren’t overestimated in severity and avoiding unnecessary overcompensation, thus conserving resources. However, overly prioritizing precision risks missing legitimate claims. In contrast, higher recall aims to identify all valid claims, minimizing the chance of overlooking legitimate cases but increasing false positives. This could lead to wasted resources investigating or compensating less severe claims. Striking a balance is crucial to ensure fairness, accurately assess claims, and comply with the board's mandate while minimizing resource misuse. Additionally, incorporating Shapley values for model interpretability would be invaluable, allowing us to explain the model's predictions and provide actionable insights to the council.

Bibliography

Aning, J., Nti, I. K., & Nyarko-Boateng, O. (2021). Performance of Machine Learning Algorithms with Different K Values in K-fold CrossValidation. *International Journal of Information Technology and Computer Science*, *13*(6), 61–71. https://doi.org/10.5815/ijitcs.2021.06.05

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, *16*, 321–357. https://doi.org/10.1613/jair.953

Haibo He, Yang Bai, Garcia, E. A., & Shutao Li. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 1322–1328. https://doi.org/10.1109/IJCNN.2008.4633969

Howitt, D., & Cramer, D. (Eds.). (2004). Cramer's V. In *The SAGE Dictionary of Statistics*. SAGE Publications, Ltd. https://doi.org/10.4135/9780857020123.n125

Law Offices of Mario S Crisafulli. (2023, October 17). *How Long Does A Workers Comp Settlement Take In NY?* https://injuredatworkalbany.com/how-long-does-a-workers-comp-settlement-take-in-ny/

Lock, R. H., Lock, P. F., Morgan, K. L., Lock, E. F., & Lock, D. F. (2020). *Statistics: Unlocking the power of data* (Third edition). Wiley.

Meepaganithage, A., Rath, S., Nicolescu, M., Nicolescu, M., & Sengupta, S. (2024). Feature Selection Using the Advanced Shapley Value. *2024 IEEE 14th Annual Computing and Communication Workshop and Conference (CCWC)*, 0207–0213. https://doi.org/10.1109/CCWC60891.2024.10427665

Parshuram, H. P., & Joshi, S. G. (2024). Machine Learning Advancements in Healthcare Insurance: A Comprehensive Review and Future Directions. *International Journal of Advanced Research in Science, Communication and Technology*, 283–291. https://doi.org/10.48175/IJARSCT-17843

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Müller, A., Nothman, J., Louppe, G., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2012). *Scikit-learn: Machine Learning in Python*. https://doi.org/10.48550/ARXIV.1201.0490

Ross, B. C. (2014). Mutual Information between Discrete and Continuous Data Sets. *PLoS ONE*, *9*(2), e87357. https://doi.org/10.1371/journal.pone.0087357

U.S. Department of Labor. (n.d.). *Procedure Manual*. DOL. Retrieved December 21, 2024, from https://www.dol.gov/agencies/owcp/FECA/regs/compliance/DFECfolio/FECA-PT1

Vergara, J. R., & Estévez, P. A. (2014). A Review of Feature Selection Methods Based on Mutual Information. *Neural Computing and Applications*, *24*(1), 175–186. https://doi.org/10.1007/s00521-013-1368-0

Appendix A – Pipeline and Workflow Diagrams

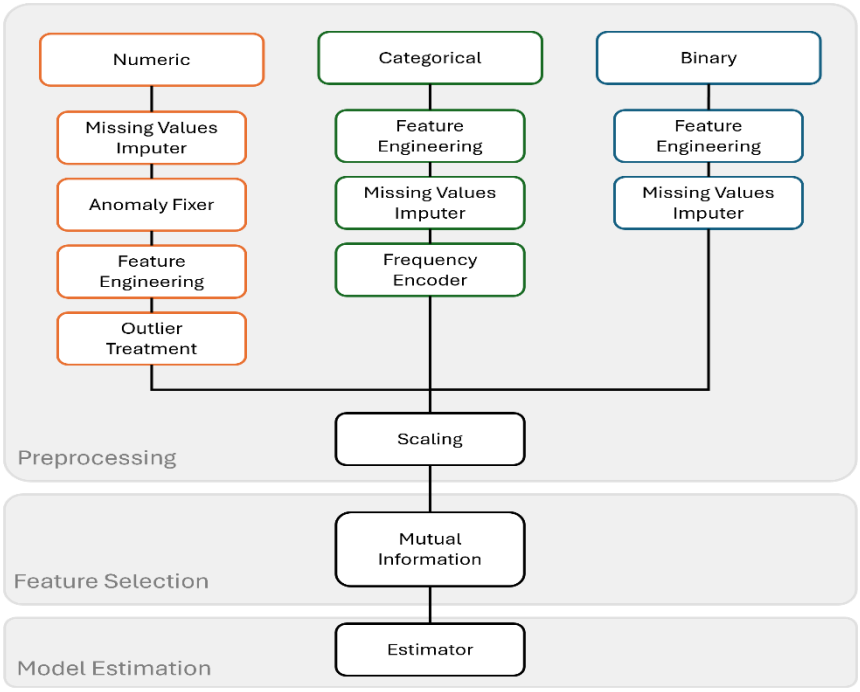

Figure 1 - Adopted pipeline architecture for data preprocessing, feature selection and model estimation.

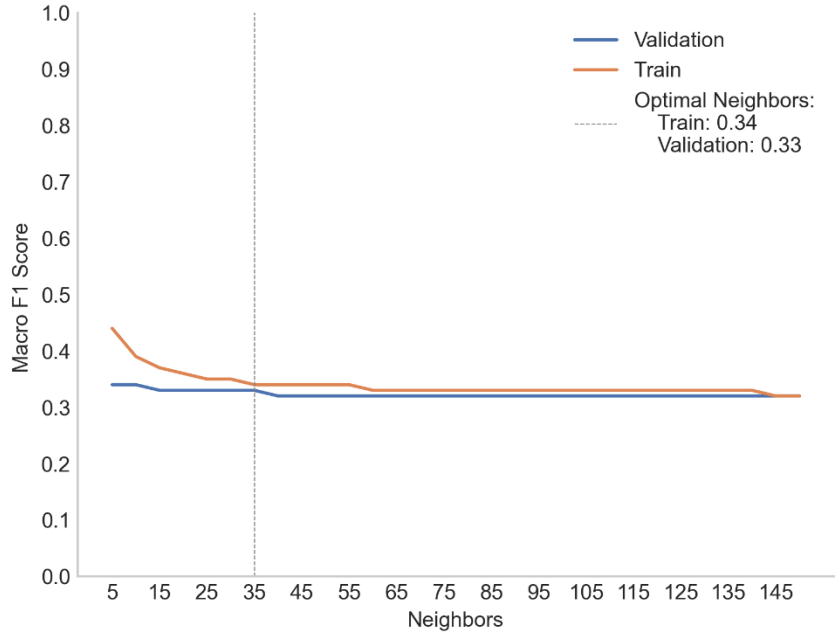

Figure 2 - Estimation of the best number of neighbors for feature selection.

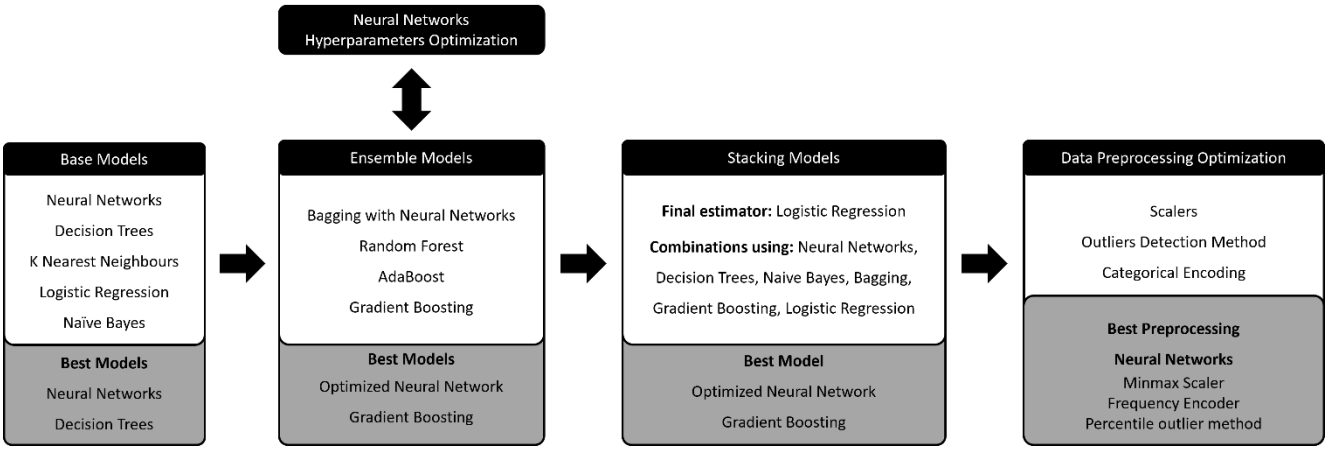

Figure 3- Overview of the model development and optimization process.

Appendix B – Candidate Models Comparison

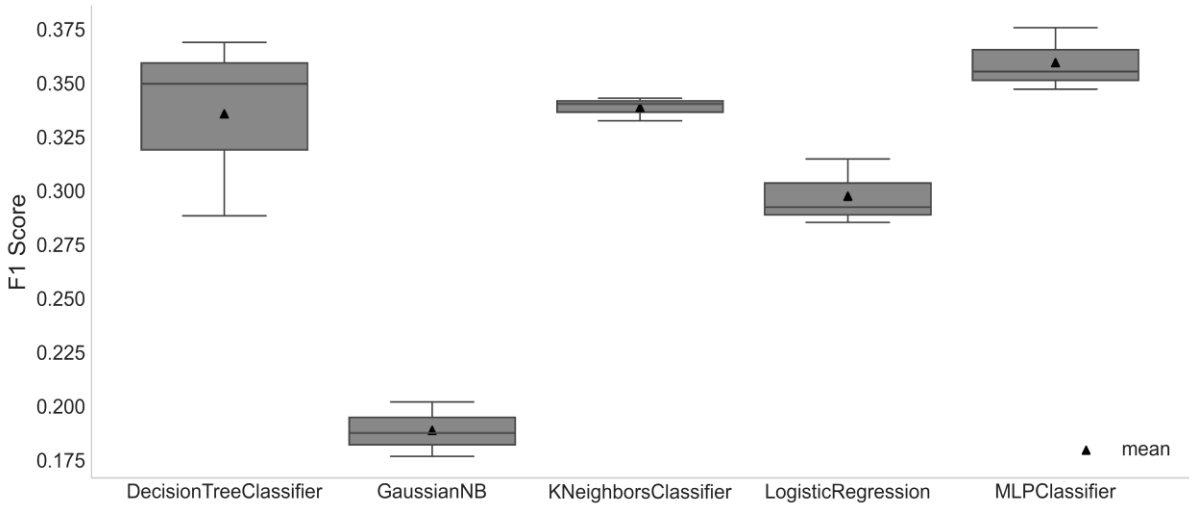

Figure 4 - Distribution of Macro F1 validation scores for the baseline models across 3-fold cross-validation.

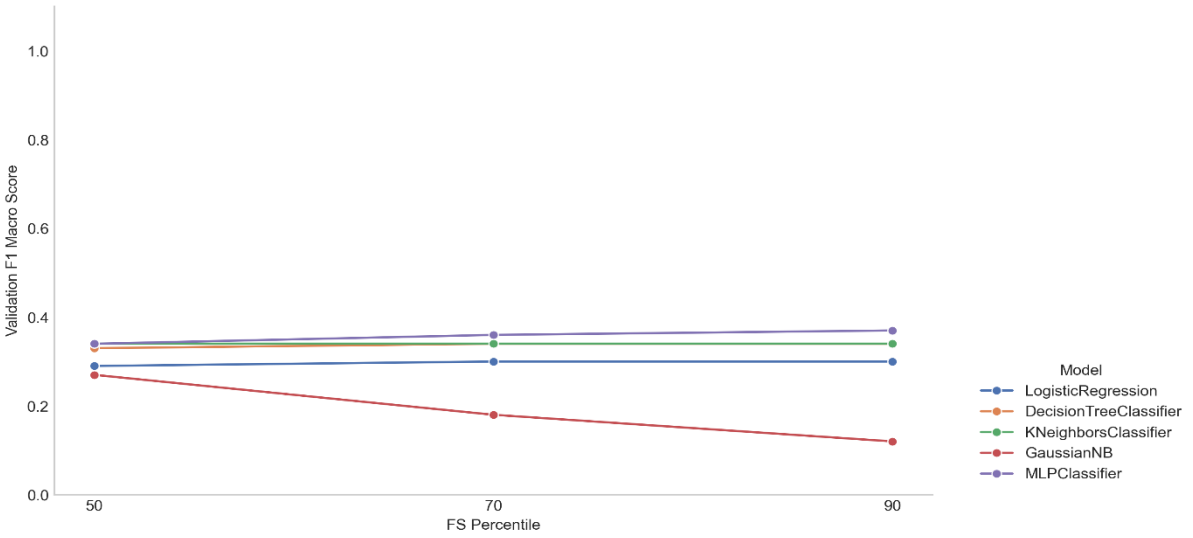

Figure 5 – Macro F1 validation Scores for baseline models across different FS (Feature Selection) percentiles.

Table 1 - Mean and Standard Deviation of Macro F1 Scores for Train and Validation Across Baseline Models (3-Fold Cross-Validation).

Model	Parameters	FS Percentile	F1 Train Score	F1 Validation Score
MLPClassifier	hidden_layer_sizes=(20, 20), learning_rate_init=0.01, max_iter=1000	90	0.39 ± 0.02	0.37 ± 0.01
MLPClassifier	hidden_layer_sizes=(20, 20), learning_rate_init=0.01, max_iter=1000	70	0.37 ± 0.02	0.36 ± 0.02
MLPClassifier	hidden_layer_sizes=(20, 20), learning_rate_init=0.01, max_iter=1000	50	0.36 ± 0.02	0.34 ± 0.01
LogisticRegression	class_weight='balanced', max_iter=1000	90	0.32 ± 0.0	0.3 ± 0.01
LogisticRegression	class_weight='balanced', max_iter=1000	70	0.31 ± 0.01	0.3 ± 0.01
LogisticRegression	class_weight='balanced', max_iter=1000	50	0.3 ± 0.01	0.29 ± 0.02
KNeighborsClassifier	n_neighbors=35, p=1, weights='distance'	50	1.0 ± 0.0	0.34 ± 0.01
KNeighborsClassifier	n_neighbors=35, p=1, weights='distance'	70	1.0 ± 0.0	0.34 ± 0.01
KNeighborsClassifier	n_neighbors=35, p=1, weights='distance'	90	1.0 ± 0.0	0.34 ± 0.0
GaussianNB	base model	50	0.27 ± 0.01	0.27 ± 0.01
GaussianNB	base model	70	0.18 ± 0.04	0.18 ± 0.03
GaussianNB	base model	90	0.13 ± 0.01	0.12 ± 0.01
DecisionTreeClassifier	class_weight='balanced', max_depth=15	90	0.46 ± 0.01	0.34 ± 0.03
DecisionTreeClassifier	class_weight='balanced', max_depth=15	70	0.46 ± 0.01	0.34 ± 0.03
DecisionTreeClassifier	class_weight='balanced', max_depth=15	50	0.45 ± 0.01	0.33 ± 0.04

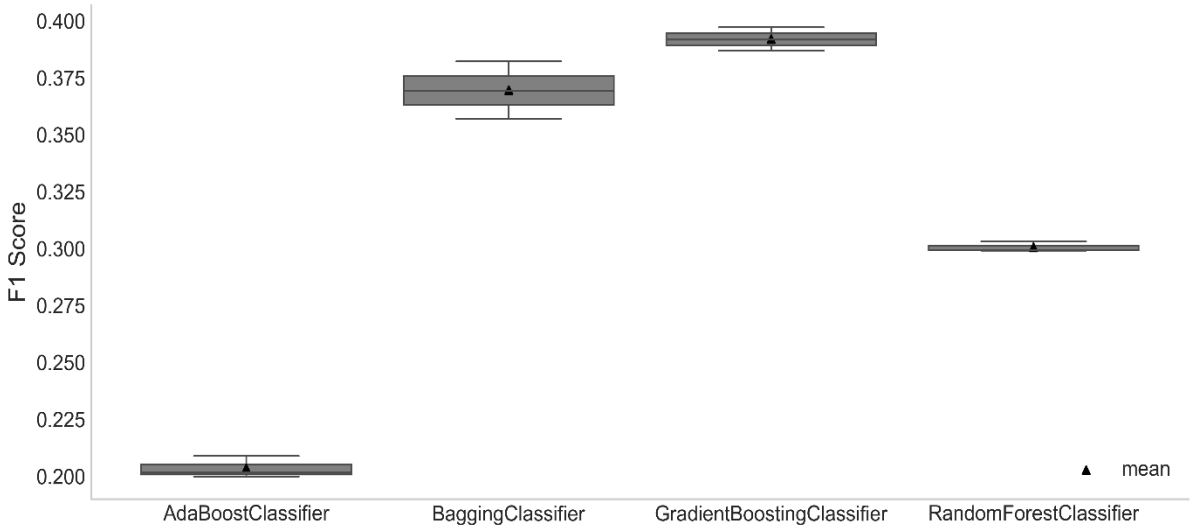

Figure 6 - Distribution of Macro F1 validation scores for the baseline ensemble models across 3-fold cross-validation.

Table 2 - Mean and Standard Deviation of Macro F1 Scores for Train and Validation Across Baseline Ensemble Models (3-Fold Cross-Validation).

Model	Parameters	F1 Train Score	F1 Validation Score
RandomForestClassifier	max_depth: 10, min_samples_split: 2, n_estimators: 100	0.36 ± 0.01	0.35 ± 0.0
RandomForestClassifier	max_depth: 10, min_samples_split: 2, n_estimators: 200	0.36 ± 0.01	0.35 ± 0.0
RandomForestClassifier	max_depth: 10, min_samples_split: 5, n_estimators: 100	0.36 ± 0.0	0.35 ± 0.0
RandomForestClassifier	max_depth: 10, min_samples_split: 5, n_estimators: 200	0.36 ± 0.01	0.35 ± 0.0
RandomForestClassifier	max_depth: 5, min_samples_split: 2, n_estimators: 200	0.25 ± 0.01	0.25 ± 0.01
RandomForestClassifier	max_depth: 5, min_samples_split: 5, n_estimators: 100	0.25 ± 0.01	0.25 ± 0.01
RandomForestClassifier	max_depth: 5, min_samples_split: 5, n_estimators: 200	0.25 ± 0.01	0.25 ± 0.01
RandomForestClassifier	max_depth: 5, min_samples_split: 2, n_estimators: 100	0.25 ± 0.01	0.25 ± 0.0
GradientBoostingClassifier	learning_rate: 0.05, max_depth: 5, n_estimators: 100	0.52 ± 0.03	0.4 ± 0.01
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 5, n_estimators: 50	0.46 ± 0.01	0.4 ± 0.01
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 5, n_estimators: 100	0.48 ± 0.01	0.4 ± 0.01
GradientBoostingClassifier	learning_rate: 0.05, max_depth: 5, n_estimators: 50	0.49 ± 0.02	0.39 ± 0.01
GradientBoostingClassifier	learning_rate: 0.05, max_depth: 10, n_estimators: 50	0.7 ± 0.01	0.39 ± 0.01
GradientBoostingClassifier	learning_rate: 0.05, max_depth: 10, n_estimators: 100	0.75 ± 0.01	0.39 ± 0.01
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 10, n_estimators: 50	0.72 ± 0.01	0.39 ± 0.01
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 10, n_estimators: 100	0.8 ± 0.01	0.38 ± 0.02
BaggingClassifier	estimator=MLPClassifier(hidden_layer_sizes=(20, 20), learning_rate_init=0.01 max_iter=1000, max_features: 1.0, max_samples: 0.8, n_estimators=10	0.4 ± 0.01	0.38 ± 0.02
BaggingClassifier	estimator=MLPClassifier(hidden_layer_sizes=(20, 20), learning_rate_init=0.01 max_iter=1000, max_features: 0.8, max_samples: 0.8, n_estimators=10	0.38 ± 0.01	0.37 ± 0.01
BaggingClassifier	estimator=MLPClassifier(hidden_layer_sizes=(20, 20), learning_rate_init=0.01 max_iter=1000, max_features: 1.0, max_samples: 1.0, n_estimators=10	0.39 ± 0.02	0.37 ± 0.0
BaggingClassifier	estimator=MLPClassifier(hidden_layer_sizes=(20, 20), learning_rate_init=0.01 max_iter=1000, max_features: 0.8, max_samples: 1.0, n_estimators=10	0.36 ± 0.01	0.36 ± 0.01
AdaBoostClassifier	learning_rate: 0.05, n_estimators: 50	0.2 ± 0.0	0.2 ± 0.0
AdaBoostClassifier	learning_rate: 0.05, n_estimators: 100	0.2 ± 0.0	0.2 ± 0.0
AdaBoostClassifier	learning_rate: 0.1, n_estimators: 50	0.2 ± 0.0	0.2 ± 0.0
AdaBoostClassifier	learning_rate: 0.1, n_estimators: 100	0.2 ± 0.0	0.2 ± 0.0

Table 3 - Mean and Standard Deviation of Macro F1 Scores for Train and Validation of the Multi-Layer Perceptron optimization (3-Fold Cross-Validation).

Activation Function	Architecture	Learning Rate	Initial Learning Rate	Solver	F1 Train Score	F1 Validation Score
relu	(30, 30)	adaptive	0.02	sgd	0.42 ± 0.01	0.4 ± 0.02
relu	(30, 30)	constant	0.02	sgd	0.42 ± 0.01	0.4 ± 0.01
tanh	(30, 30)	constant	0.01	sgd	0.41 ± 0.01	0.4 ± 0.01
relu	(20, 20, 20)	adaptive	0.02	sgd	0.41 ± 0.01	0.39 ± 0.02
relu	(30, 30)	adaptive	0.01	adam	0.41 ± 0.01	0.39 ± 0.01
tanh	(20, 20, 20)	adaptive	0.00	sgd	0.41 ± 0.01	0.39 ± 0.01
relu	(30, 30)	adaptive	0.01	sgd	0.41 ± 0.01	0.39 ± 0.02
relu	(30, 30)	adaptive	0.00	sgd	0.41 ± 0.01	0.39 ± 0.01
relu	(30, 30)	constant	0.00	sgd	0.41 ± 0.01	0.39 ± 0.01
tanh	(20, 20, 20)	constant	0.01	sgd	0.41 ± 0.01	0.39 ± 0.02
tanh	(20, 20, 20)	constant	0.02	sgd	0.41 ± 0.01	0.39 ± 0.02
relu	(30, 30)	constant	0.01	sgd	0.41 ± 0.01	0.39 ± 0.01
logistic	(30, 30)	adaptive	0.00	adam	0.41 ± 0.01	0.39 ± 0.01
tanh	(30, 30)	adaptive	0.00	sgd	0.41 ± 0.01	0.39 ± 0.02
tanh	(30, 30)	adaptive	0.02	sgd	0.42 ± 0.0	0.39 ± 0.01

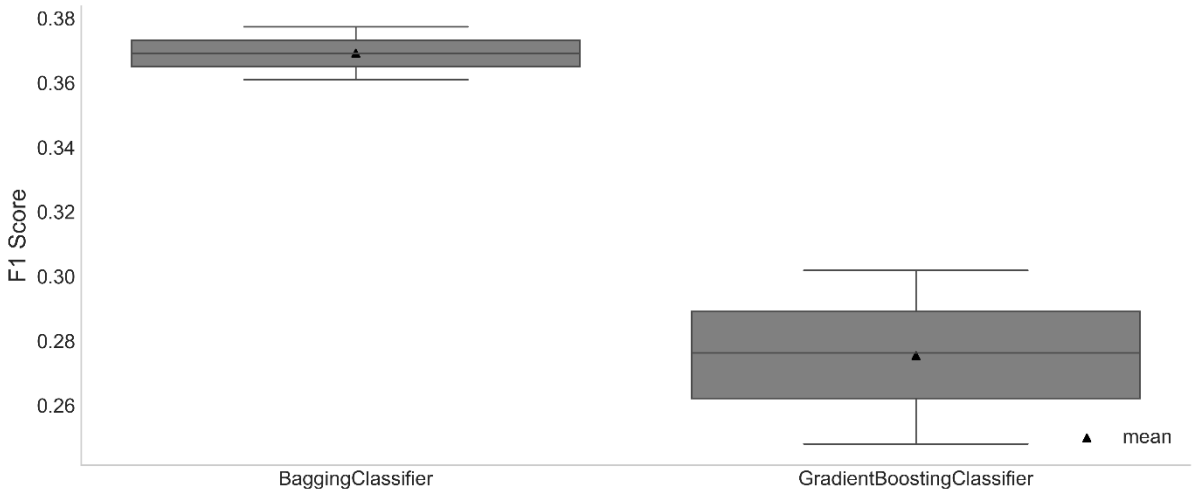

Figure 7 - Distribution of Macro F1 validation scores for the optimized selected ensemble models across 3-fold cross-validation.

Table 4 - Mean and Standard Deviation of Macro F1 Scores for Train and Validation of Bagging model with optimized Multi-Layer Perceptron (3-Fold Cross-Validation).

Model	Parameters	F1 Train Score	F1 Validation Score
BaggingClassifier	estimator=MLPClassifier(activation=tanh, hidden_layer_sizes=(30, 30), learning_rate_init=0.01 max_iter=1000, solver=sgd , max_features: 1.0, max_samples: 0.2, n_estimators: 10, n_estimators=10	0.4 ± 0.01	0.38 ± 0.01
BaggingClassifier	estimator=MLPClassifier(activation=tanh, hidden_layer_sizes=(30, 30), learning_rate_init=0.01 max_iter=1000, solver=sgd , max_features: 1.0, max_samples: 0.4, n_estimators: 30, n_estimators=10	0.4 ± 0.01	0.38 ± 0.01
BaggingClassifier	estimator=MLPClassifier(activation=tanh, hidden_layer_sizes=(30, 30), learning_rate_init=0.01 max_iter=1000, solver=sgd , max_features: 1.0, max_samples: 0.4, n_estimators: 50, n_estimators=10	0.41 ± 0.01	0.38 ± 0.01
BaggingClassifier	estimator=MLPClassifier(activation=tanh, hidden_layer_sizes=(30, 30), learning_rate_init=0.01 max_iter=1000, solver=sgd , max_features: 0.8, max_samples: 0.4, n_estimators: 10, n_estimators=10	0.39 ± 0.0	0.37 ± 0.01
BaggingClassifier	estimator=MLPClassifier(activation=tanh, hidden_layer_sizes=(30, 30), learning_rate_init=0.01 max_iter=1000, solver=sgd , max_features: 0.8, max_samples: 0.4, n_estimators: 30, n_estimators=10	0.39 ± 0.01	0.37 ± 0.01
BaggingClassifier	estimator=MLPClassifier(activation=tanh, hidden_layer_sizes=(30, 30), learning_rate_init=0.01 max_iter=1000, solver=sgd , max_features: 1.0, max_samples: 0.4, n_estimators: 10, n_estimators=10	0.4 ± 0.01	0.37 ± 0.01
BaggingClassifier	estimator=MLPClassifier(activation=tanh, hidden_layer_sizes=(30, 30), learning_rate_init=0.01 max_iter=1000, solver=sgd , max_features: 0.8, max_samples: 0.4, n_estimators: 50, n_estimators=10	0.38 ± 0.02	0.37 ± 0.0
BaggingClassifier	estimator=MLPClassifier(activation=tanh, hidden_layer_sizes=(30, 30), learning_rate_init=0.01 max_iter=1000, solver=sgd , max_features: 1.0, max_samples: 0.2, n_estimators: 30, n_estimators=10	0.39 ± 0.02	0.37 ± 0.0
BaggingClassifier	estimator=MLPClassifier(activation=tanh, hidden_layer_sizes=(30, 30), learning_rate_init=0.01 max_iter=1000, solver=sgd , max_features: 1.0, max_samples: 0.2, n_estimators: 50, n_estimators=10	0.39 ± 0.01	0.37 ± 0.0
BaggingClassifier	estimator=MLPClassifier(activation=tanh, hidden_layer_sizes=(30, 30), learning_rate_init=0.01 max_iter=1000, solver=sgd , max_features: 0.8, max_samples: 0.2, n_estimators: 30, n_estimators=10	0.37 ± 0.01	0.36 ± 0.0
BaggingClassifier	estimator=MLPClassifier(activation=tanh, hidden_layer_sizes=(30, 30), learning_rate_init=0.01 max_iter=1000, solver=sgd , max_features: 0.8, max_samples: 0.2, n_estimators: 50, n_estimators=10	0.37 ± 0.01	0.36 ± 0.0
BaggingClassifier	estimator=MLPClassifier(activation=tanh, hidden_layer_sizes=(30, 30), learning_rate_init=0.01 max_iter=1000, solver=sgd , max_features: 0.8, max_samples: 0.2, n_estimators: 10, n_estimators=10	0.36 ± 0.0	0.35 ± 0.0

Table 5 - Mean and Standard Deviation of Macro F1 Scores for Train and Validation of the optimized Gradient Boost model (3-Fold Cross-Validation).

Model	Parameters	F1 Train Score	F1 Validation Score
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 5, max_features: 0.5, n_estimators: 50, subsample: 1	0.47 ± 0.01	0.39 ± 0.01
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 5, max_features: 0.5, n_estimators: 100, subsample: 1	0.45 ± 0.05	0.38 ± 0.03
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 5, max_features: 0.5, n_estimators: 50, subsample: 0.5	0.4 ± 0.03	0.38 ± 0.01
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 10, max_features: 0.5, n_estimators: 50, subsample: 1	0.74 ± 0.01	0.37 ± 0.03
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 10, max_features: 0.5, n_estimators: 100, subsample: 1	0.77 ± 0.03	0.37 ± 0.02
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 5, max_features: 0.5, n_estimators: 100, subsample: 0.5	0.43 ± 0.01	0.37 ± 0.01
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 10, max_features: 0.5, n_estimators: 50, subsample: 0.5	0.53 ± 0.0	0.37 ± 0.01
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 5, max_features: 1, n_estimators: 100, subsample: 1	0.43 ± 0.04	0.36 ± 0.01
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 10, max_features: 1, n_estimators: 50, subsample: 1	0.59 ± 0.05	0.36 ± 0.01
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 10, max_features: 1, n_estimators: 100, subsample: 1	0.55 ± 0.12	0.36 ± 0.01
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 10, max_features: 1, n_estimators: 100, subsample: 0.5	0.45 ± 0.0	0.34 ± 0.02
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 10, max_features: 1, n_estimators: 50, subsample: 0.5	0.42 ± 0.01	0.34 ± 0.01
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 5, max_features: 1, n_estimators: 50, subsample: 1	0.37 ± 0.02	0.33 ± 0.0
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 10, max_features: 0.5, n_estimators: 100, subsample: 0.5	0.52 ± 0.05	0.31 ± 0.06
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 5, max_features: 1, n_estimators: 50, subsample: 1	0.34 ± 0.03	0.31 ± 0.04
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 5, max_features: 1, n_estimators: 50, subsample: 0.5	0.35 ± 0.02	0.31 ± 0.01
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 10, max_features: 1, n_estimators: 50, subsample: 1	0.42 ± 0.03	0.29 ± 0.03
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 5, max_features: 0.5, n_estimators: 50, subsample: 1	0.28 ± 0.08	0.27 ± 0.07
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 10, max_features: 1, n_estimators: 50, subsample: 0.5	0.33 ± 0.01	0.26 ± 0.03
GradientBoostingClassifier	learning_rate: 0.1, max_depth: 5, max_features: 1, n_estimators: 100, subsample: 0.5	0.27 ± 0.15	0.25 ± 0.15
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 10, max_features: 1, n_estimators: 100, subsample: 1	0.33 ± 0.01	0.25 ± 0.03
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 5, max_features: 1, n_estimators: 100, subsample: 1	0.27 ± 0.06	0.24 ± 0.04
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 5, max_features: 0.5, n_estimators: 50, subsample: 0.5	0.22 ± 0.12	0.21 ± 0.11
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 10, max_features: 1, n_estimators: 100, subsample: 0.5	0.26 ± 0.05	0.2 ± 0.02
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 5, max_features: 1, n_estimators: 50, subsample: 0.5	0.22 ± 0.06	0.18 ± 0.05
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 10, max_features: 0.5, n_estimators: 50, subsample: 1	0.21 ± 0.04	0.18 ± 0.01
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 10, max_features: 0.5, n_estimators: 50, subsample: 0.5	0.2 ± 0.12	0.16 ± 0.06
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 5, max_features: 1, n_estimators: 100, subsample: 0.5	0.19 ± 0.08	0.16 ± 0.04
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 10, max_features: 0.5, n_estimators: 100, subsample: 0.5	0.17 ± 0.03	0.15 ± 0.04
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 5, max_features: 0.5, n_estimators: 100, subsample: 1	0.14 ± 0.11	0.14 ± 0.1
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 5, max_features: 0.5, n_estimators: 100, subsample: 0.5	0.13 ± 0.03	0.12 ± 0.05
GradientBoostingClassifier	learning_rate: 0.5, max_depth: 10, max_features: 0.5, n_estimators: 100, subsample: 1	0.14 ± 0.03	0.12 ± 0.01

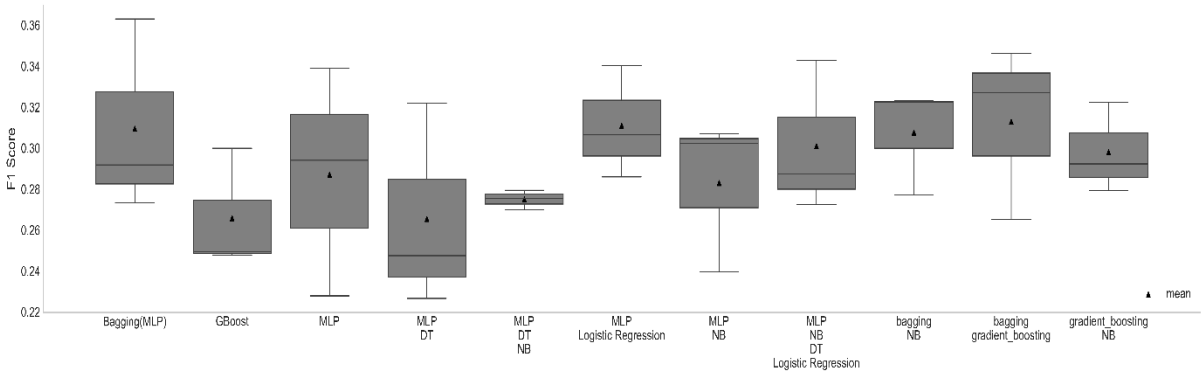

Figure 8 - Distribution of Macro F1 Validation Scores for Stacking Models across 3-fold cross-validation. The figure displays the performance distribution of the base learners included in the stacking ensemble, with Logistic Regression used as the meta-learner.

Table 6 - Mean and Standard Deviation of Macro F1 Scores for Train and Validation Across Stacking Models (3-Fold Cross-Validation) with Logistic Regression used as the meta-learner.

Base Learners	F1 Validation Score	F1 Train Score
Bagging(MLP)	0.31 ± 0.07	0.33 ± 0.08
Bagging(MLP), GBoost	0.31 ± 0.06	0.33 ± 0.07
Bagging(MLP), NB	0.31 ± 0.04	0.33 ± 0.04
MLP, NB, DT, Logistic Regression	0.30 ± 0.04	0.33 ± 0.03
GBoost, NB	0.30 ± 0.03	0.32 ± 0.04
MLP, Logistic Regression	0.31 ± 0.03	0.32 ± 0.03
MLP	0.29 ± 0.07	0.30 ± 0.08
MLP, DT, NB	0.27 ± 0.04	0.30 ± 0.05
MLP, DT	0.27 ± 0.07	0.29 ± 0.08
GBoost	0.27 ± 0.06	0.29 ± 0.07
MLP, NB	0.28 ± 0.05	0.29 ± 0.05

Table 7 - Mean and Standard Deviation of Macro F1 Scores for Train and Validation across two selected best models with different preprocessing techniques combinations (3-Fold Cross-Validation).

Model	Encoding Strategy	Outlier Detection	Scaler	F1 Train Score	F1 Validation Score
MLPClassifier	Frequency Encoder	Percentile	MinMaxScaler	0.41 ± 0.00	0.40 ± 0.02
MLPClassifier	Frequency Encoder	Percentile	StandardScaler	0.42 ± 0.02	0.39 ± 0.01
MLPClassifier	Frequency Encoder	Percentile	RobustScaler	0.28 ± 0.00	0.28 ± 0.01
MLPClassifier	Frequency Encoder	IQR	StandardScaler	0.41 ± 0.01	0.39 ± 0.02
MLPClassifier	Frequency Encoder	IQR	MinMaxScaler	0.40 ± 0.02	0.39 ± 0.01
MLPClassifier	Frequency Encoder	IQR	RobustScaler	0.36 ± 0.01	0.35 ± 0.00
MLPClassifier	Frequency Encoder Normalized	Percentile	StandardScaler	0.42 ± 0.01	0.40 ± 0.01
MLPClassifier	Frequency Encoder Normalized	Percentile	MinMaxScaler	0.42 ± 0.01	0.40 ± 0.01
MLPClassifier	Frequency Encoder Normalized	Percentile	RobustScaler	0.32 ± 0.02	0.31 ± 0.02
MLPClassifier	Frequency Encoder Normalized	IQR	StandardScaler	0.41 ± 0.01	0.39 ± 0.02
MLPClassifier	Frequency Encoder Normalized	IQR	RobustScaler	0.41 ± 0.01	0.39 ± 0.02
MLPClassifier	Frequency Encoder Normalized	IQR	MinMaxScaler	0.41 ± 0.01	0.39 ± 0.01
MLPClassifier	Ordinal Encoder	Percentile	MinMaxScaler	0.42 ± 0.01	0.40 ± 0.02
MLPClassifier	Ordinal Encoder	Percentile	StandardScaler	0.42 ± 0.01	0.40 ± 0.01
MLPClassifier	Ordinal Encoder	Percentile	RobustScaler	0.31 ± 0.02	0.31 ± 0.02
MLPClassifier	Ordinal Encoder	IQR	StandardScaler	0.41 ± 0.01	0.39 ± 0.02
MLPClassifier	Ordinal Encoder	IQR	MinMaxScaler	0.41 ± 0.01	0.39 ± 0.02
MLPClassifier	Ordinal Encoder	IQR	RobustScaler	0.41 ± 0.01	0.39 ± 0.02
GradientBoostingClassifier	Frequency Encoder	Percentile	RobustScaler	0.46 ± 0.02	0.40 ± 0.00
GradientBoostingClassifier	Frequency Encoder	Percentile	StandardScaler	0.45 ± 0.02	0.39 ± 0.01
GradientBoostingClassifier	Frequency Encoder	Percentile	MinMaxScaler	0.46 ± 0.03	0.39 ± 0.01
GradientBoostingClassifier	Frequency Encoder	IQR	RobustScaler	0.43 ± 0.03	0.39 ± 0.01
GradientBoostingClassifier	Frequency Encoder	IQR	StandardScaler	0.44 ± 0.01	0.38 ± 0.01
GradientBoostingClassifier	Frequency Encoder	IQR	MinMaxScaler	0.43 ± 0.02	0.35 ± 0.04
GradientBoostingClassifier	Frequency Encoder Normalized	Percentile	MinMaxScaler	0.46 ± 0.01	0.40 ± 0.01
GradientBoostingClassifier	Frequency Encoder Normalized	Percentile	StandardScaler	0.48 ± 0.03	0.39 ± 0.01
GradientBoostingClassifier	Frequency Encoder Normalized	Percentile	RobustScaler	0.45 ± 0.02	0.39 ± 0.01
GradientBoostingClassifier	Frequency Encoder Normalized	IQR	MinMaxScaler	0.43 ± 0.01	0.38 ± 0.01
GradientBoostingClassifier	Frequency Encoder Normalized	IQR	RobustScaler	0.44 ± 0.04	0.38 ± 0.01
GradientBoostingClassifier	Frequency Encoder Normalized	IQR	StandardScaler	0.43 ± 0.02	0.38 ± 0.00
GradientBoostingClassifier	Ordinal Encoder	Percentile	StandardScaler	0.46 ± 0.01	0.38 ± 0.02
GradientBoostingClassifier	Ordinal Encoder	Percentile	MinMaxScaler	0.47 ± 0.04	0.38 ± 0.00
GradientBoostingClassifier	Ordinal Encoder	Percentile	RobustScaler	0.45 ± 0.05	0.37 ± 0.04
GradientBoostingClassifier	Ordinal Encoder	IQR	StandardScaler	0.45 ± 0.04	0.38 ± 0.02
GradientBoostingClassifier	Ordinal Encoder	IQR	RobustScaler	0.45 ± 0.02	0.38 ± 0.01
GradientBoostingClassifier	Ordinal Encoder	IQR	MinMaxScaler	0.43 ± 0.01	0.37 ± 0.01

Table 8 – Classification report of the best model selected.

	precision	recall	f1-score	support
1. CANCELLED	0.72	0.43	0.54	12477.00
2. NON-COMP	0.85	0.98	0.91	291078.00
3. MED ONLY	0.46	0.07	0.13	68906.00
4. TEMPORARY	0.73	0.86	0.79	148507.00
5. PPD SCH LOSS	0.67	0.60	0.63	48280.00
6. PPD NSL	1.00	0.00	0.00	4211.00
7. PTD	1.00	0.00	0.00	97.00
8. DEATH	0.60	0.31	0.41	470.00
accuracy	0.79	0.79	0.79	0.79
macro avg	0.75	0.41	0.43	574026.00
weighted avg	0.75	0.79	0.74	574026.00

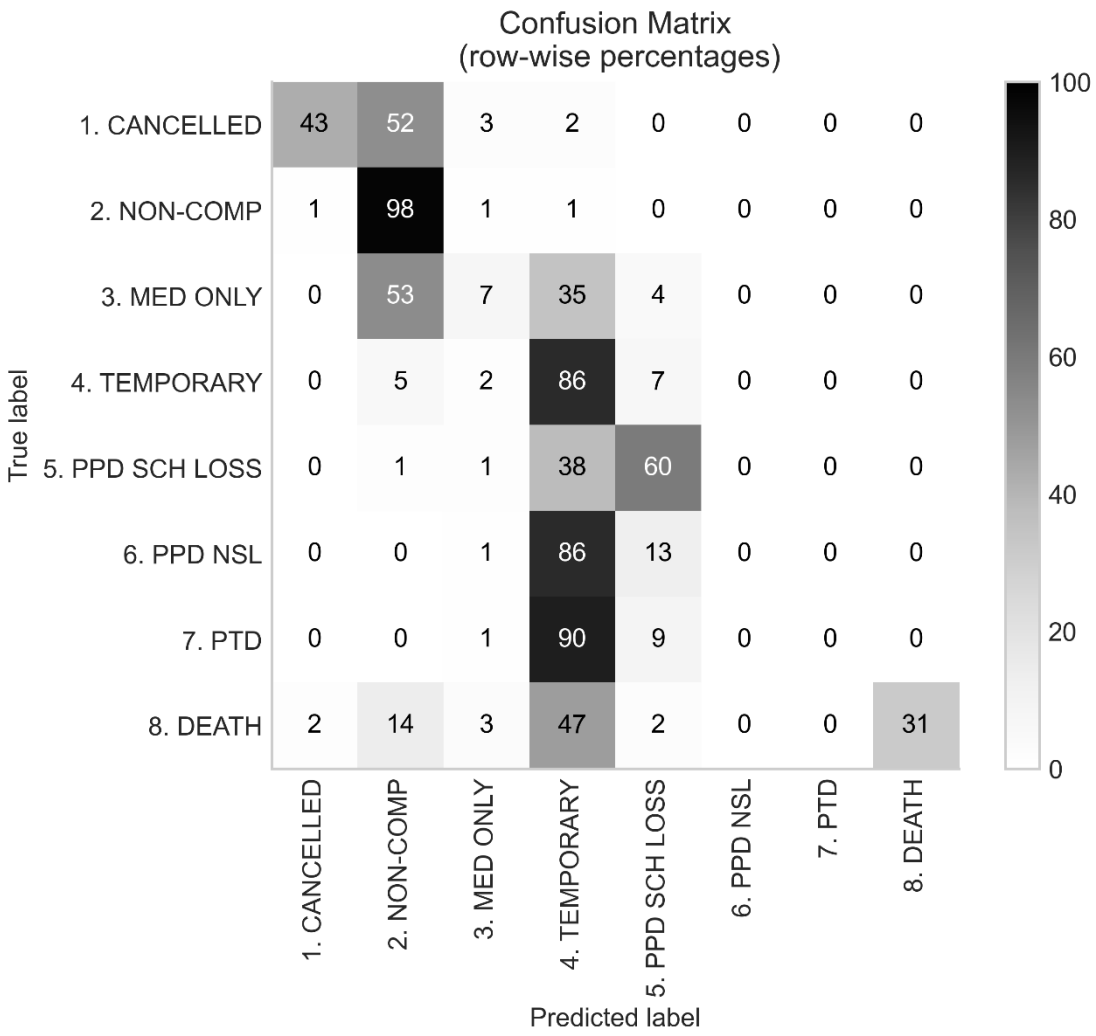

Figure 9 Confusion Matrix of the best model selected.

Appendix C – Oversampling & One-versus-all

Table 9 - Mean and Standard Deviation of Macro F1 Scores for Train and Validation using OneVsRestClassifier with MLPClassifier and MLPClassifier with and without oversampling (SMOTE and ADASYN) under varying neighbor parameters (3-Fold Cross-Validation).

Model	Oversampling	Neighbours	F1 Train Score	F1 Validation Score
OneVsRestClassifier	None	-	0.39 ± 0.01	0.37 ± 0.0
MLPClassifier	None	-	0.42 ± 0.0	0.4 ± 0.01
OneVsRestClassifier	SMOTE	5	0.52 ± 0.01	0.37 ± 0.04
OneVsRestClassifier	SMOTE	10	0.52 ± 0.01	0.37 ± 0.03
MLPClassifier	SMOTE	5	0.46 ± 0.01	0.36 ± 0.02
MLPClassifier	SMOTE	10	0.44 ± 0.02	0.34 ± 0.05
OneVsRestClassifier	ADASYN	5	0.52 ± 0.02	0.35 ± 0.03
OneVsRestClassifier	ADASYN	10	0.52 ± 0.03	0.38 ± 0.02
MLPClassifier	ADASYN	5	0.44 ± 0.01	0.33 ± 0.03
MLPClassifier	ADASYN	10	0.43 ± 0.02	0.33 ± 0.06

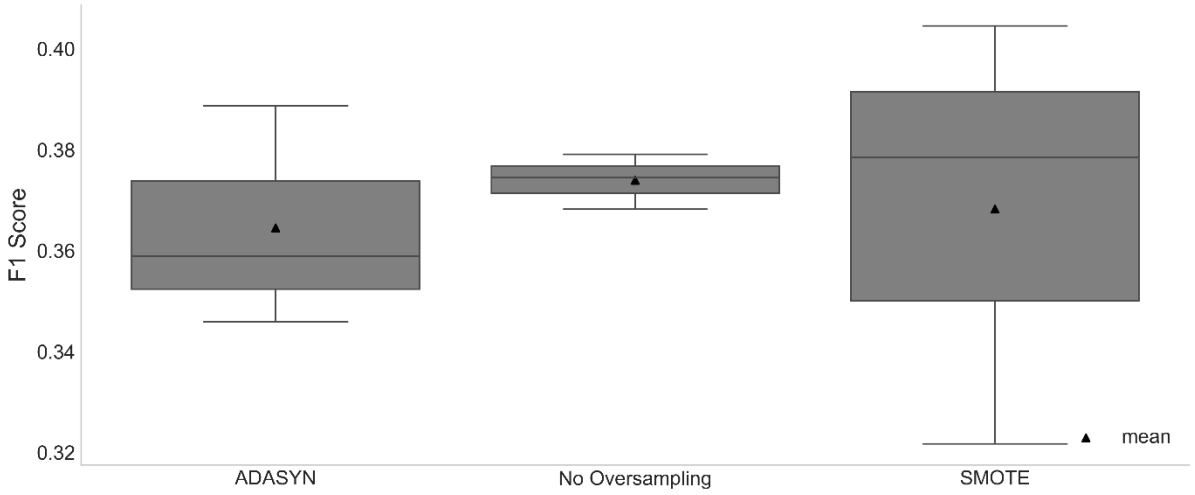

Figure 10 - Distribution of Macro F1 Validation Scores using OneVsRestClassifier with MLPClassifier estimator with and without oversampling (SMOTE and ADASYN) across 3-fold cross-validation

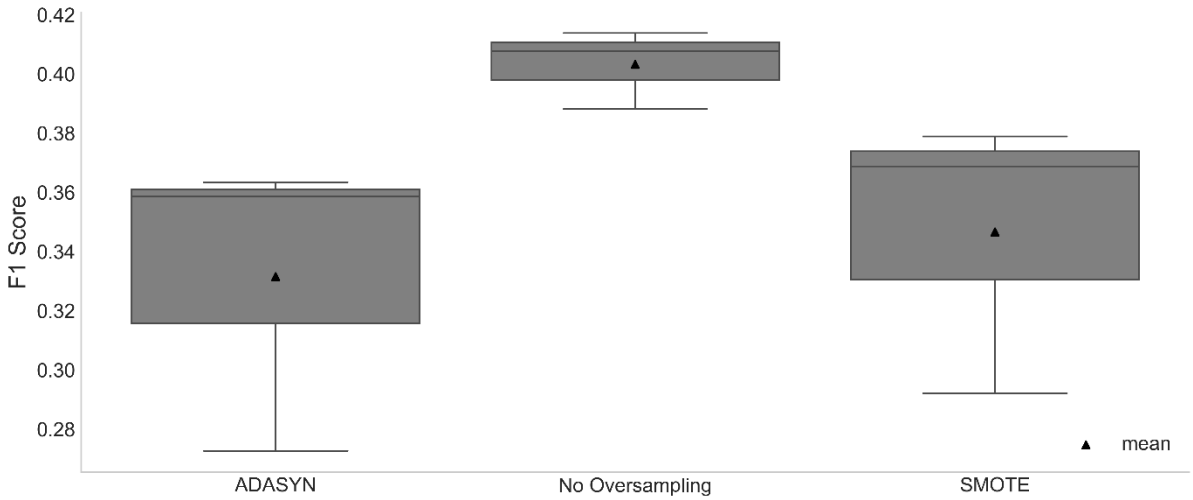

Figure 11 - Distribution of Macro F1 Validation Scores using MLPClassifier with and without oversampling (SMOTE and ADASYN) across 3-fold cross-validation

Appendix D – Extra Curricular Methods

Robust Scaler

Robust Scaler is a preprocessing technique used to standardize numeric features by removing the median and scaling based on the interquartile range (IQR). Unlike standard scaling, it is resistant to outliers, as it focuses on the central data distribution rather than extreme values (scikit-learn documentation, version 1.6.0, consulted on 18/12/24, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html).

ANOVA

ANOVA is a statistical procedure for testing differences in means among groups. It produces a p-value that indicates whether the observed differences in means among the groups are statistically significant. A small p-value (typically less than 0.05) suggests that at least one group mean is significantly different from the others, rejecting the null hypothesis that all group means are equal. ANOVA assumes that the data in each group are normally distributed, all groups have equal variances, and observations within and across groups are independent of one another (Lock et al., 2020).

In this project, the target variable would have been used as the independent variable, with its classes representing the groups, while the numeric continuous features would have acted as the dependent variables. A feature would be selected for training if the p-value from the ANOVA test exceeded the significance level; otherwise, it would be excluded.

Crammer's V

Cramér's V is a statistical measure used to assess the strength of association between two categorical variables. It provides a value between 0 and 1, where 0 indicates no association and 1 indicates a perfect association. Unlike chi-square tests, which only determine whether an association exists, Cramér's V quantifies the degree of association. It is based on the chi-square statistic and is adjusted to account for the size of the contingency table, making it suitable for comparisons across different datasets (Howitt & Cramer, 2004).

In this project, Cramér's V would have been used to evaluate the relationship between the categorical target variable and other categorical features. Feature selection could have been implemented in two ways: either by selecting the top k features with the highest Cramér's V values or by selecting features based on a specified percentile of values.

Shapley Values

Shapley values are a concept from cooperative game theory. In the machine learning context, Shapley values calculate the contribution of a feature by checking how much it improves the model's prediction when added to different groups (coalitions) of other features. This is done by averaging the feature's marginal impact across all possible combinations of features, ensuring every order and group is fairly

considered. Additionally, Shapley values could also be used to explain feature importance for less interpretable models like neural networks.

Advanced adaptations, such as the *Advanced Shapley Value*, adjust the calculation to better suit feature selection tasks by combining the mean and minimum contributions of each feature across all samples, ensuring robustness and improved accuracy (Meepaganithage et al., 2024).

In this project, Shapley values would have been used to evaluate the importance of each feature relative to the target variable. The decision to include or exclude features would be implemented similarly to Cramér's V implementation.

Mutual Information

Mutual information is calculated using the joint probability distribution of two variables and their marginal distributions. It is based on entropy, where entropy measures the uncertainty of a variable, and MI reduces uncertainty about one variable given knowledge of another. For continuous variables, the k-nearest neighbors method is commonly used to estimate entropy by determining distances to the k-th closest neighbor, providing a robust and efficient approximation without requiring binning. For categorical variables, probabilities are derived directly from the frequency of observations (Ross, 2014; Vergara & Estévez, 2014).

One-vs-All

One-vs-All (OvA), also known as One-vs-Rest (OvR), is a classification strategy used to adapt binary classifiers for multiclass problems. In OvA, separate binary classifiers are trained for each class, treating it as the positive class while grouping all other classes as the negative class. Each classifier outputs a probability, and the class corresponding to the classifier with the highest probability is selected as the final prediction (scikit-learn documentation, version 1.6.0, consulted on 18/12/24, https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html#sklearn.multiclass.OneVsRestClassifier).

SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE is a method used to address class imbalances in datasets, where the minority class or classes are underrepresented. Unlike traditional oversampling techniques that replicate instances, SMOTE generates synthetic data points by interpolating between existing minority class samples and their nearest neighbours. SMOTE begins by selecting a minority class sample and one of its k-nearest neighbours at random. It then computes the difference between the feature vectors of the chosen neighbour and the original sample. This difference vector is scaled by a random value drawn uniformly from the range [0,1] and the scaled difference is added to the original sample's feature vector. The resulting point lies on the line segment connecting the two samples and serves as the synthetic data point (Chawla et al., 2002).

This method ensures that synthetic examples are diverse yet confined within the minority class region, thereby expanding its decision boundaries. As a result, SMOTE helps models generalize better and reduces the risk of overfitting.

ADASYN (Adaptive Synthetic Sampling)

ADASYN is a method designed to address class imbalance similar to SMOTE. Unlike SMOTE, ADASYN focuses on generating more synthetic samples for minority class instances that are harder to learn based on their distribution relative to the majority class. To generate synthetic data, ADASYN first computes the degree of imbalance in the dataset and calculates a weight for each minority instance based on the number of majority instances among its k-nearest neighbours. The weights are normalized to form a distribution, which determines how many synthetic samples each minority instance will generate. For each minority instance, ADASYN randomly selects one of its k-nearest neighbours and computes the difference vector between the two. A random value from [0,1] scales this difference vector, which is then added to the original instance's feature vector to create a synthetic data point (Haibo He et al., 2008).

This adaptive method ensures that synthetic samples are concentrated in regions where the minority class overlaps more with the majority class, helping models focus on these difficult areas.

Appendix E - Disclosure of Generative AI usage

GPT4o was used to improve text fluidity, enhance sentence readability, and correct grammatical errors in the present report. The prompts were designed to preserve the original content and ideas while only improving their presentation form.

The same model was used as an auxiliary tool to generate code to help create the pipeline and custom classes, and for LATEX code manipulation to customize output tables.