NANYANG TECHNOLOGICAL UNIVERSITY

CZ4041 Machine Learning

Topic: Predict Health Outcomes of Horses

Group 12

Team Members
Dhanyamraju Harsh Rao (U)
Parashar Kshitij (U)
Malavade Sanskar Deepak (U)
Chen Kian Leong (U)
Mitra Ren Sachithananthan (U)

Table of Contents

1. Best Approach & Pipeline

1.1 Pipeline

This project aims to predict, to the closest degree of accuracy, the outcome of a horse's survival based on a series of features. To begin, we started our pipeline with delving deep into the area of equestrian veterinary to establish strong domain knowledge. This knowledge was crucial when it came to the feature engineering process as the prioritisation of key features was heavily dependent on how well we understood what these scientific terms meant, and how it would eventually affect the overall health of a horse. Data was then carefully curated across various datasets consisting of the medical records of the horses, ensuring that we have placed together the most holistic view of horse data relevant to its health.

Following this scientific research and data curation phase, we entered the data preprocessing phase, where we cleaned the data to rid it of any missing values, outliers and inconsistent values. We then used feature engineering to reduce the features that were less relevant and enriched the dataset with attributes that well-represented medical data. After this, we assessed various machine learning (ML) algorithms to evaluate which of them would be the most suitable for modelling, given our data and desired outcomes. We selected a few models - Random Forest, Bernoulli Naive Bayes, Nearest Centroid, XGBoost, LightGBM, HistGB, Neural Networks, Voting and Stacking Classifiers. We also utilised ensemble techniques. We also applied both one-hot encoding and ordinal encoding techniques where relevant according to the model selected. Regarding model training, we spliced the data into train and test sets using stratified methods, applying K-Fold cross-validation to ensure a strong model evaluation. Extensive tuning of hyperparameters was performed to optimise the performance of the models.

1.2 Best Approach

The various ML models were selected for specific reasons with the best interest of our dataset in mind. The random forest model is robust and has ensemble techniques, naive bayes for its textual classification ability and its handling of binary features, nearest centroid for its simple yet elegant classification, XGBoost, LightGBM and HistGB for its effectiveness with structured data and neural networks due to its ability to dissect intricate patterns and relationships.

However, it was HistGB model that performed the best, when coupled with ordinal encoding and augmented with external data in the training phase. With a Train F1 score of 0.93 and Test F1 score of 0.85, it is clear that the HistGB model with well-tuned hyperparameters is optimal for handling our small but complex dataset.

2. Introduction

2.1 Background

The purpose of this project is to predict the survival outcome of a horse based on data of its health across a range of clinical and physiological factors that might contribute to the eventual health and survival of a horse. This challenge put forth by Kaggle allows us to find novel solutions to equine science. Many use cases could be derived from this. For example, through the use of ML, we are able to identify which health factors play an influential role to the horse, determining if it lives to a ripe old age, dies prematurely or is sickly to the point of euthanisation. Having insights to these factors could pave the way for veterinarians and horse owners to tailor their healthcare approach based on data-driven methods to these horses to ensure the horses remain healthy.

In this project, we aimed to approach the problem holistically instead of merely tackling it from a purely statistical viewpoint. We sought to mimic patterns observable in real-world situations and apply that to our pipeline, hence contextualising our project to actual veterinary science and other influencing factors that are logical. With this, we hope to close the gap between healthcare and data science.

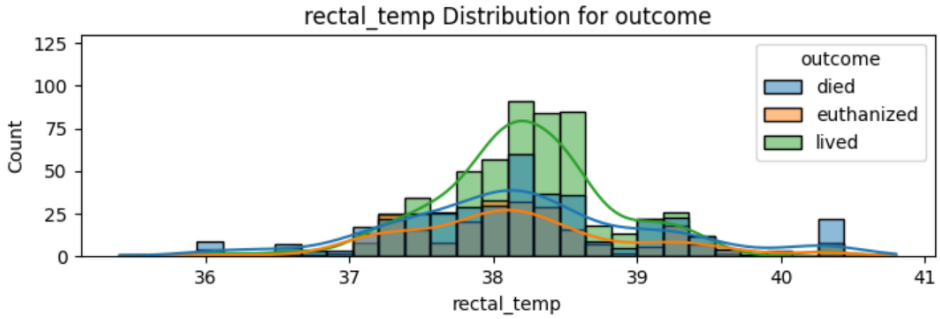
2.2 Datasets

The datasets used consists of tabular data of complex biological data. This includes a training dataset with 1235 rows and 27+1 (1 of which is the target) columns, a test dataset with 824 rows and 27 columns, as well as external data that was used to create the synthetic data for the Kaggle competition. This dataset contains 299 rows and 26+1 columns. The dataset provided was derived from a deep learning model trained using a different dataset that explores if a horse will survive based on past medical conditions. These columns represent certain medical information, such as if a horse has received surgery before, the pain it experiences, treatment results, physical condition of varying body parts, etc. Upon deeper inspection, not all of the features are relevant and contribute significantly to the overall survival outcome of the horse. This will be addressed in the data processing segment below.

3. Data Processing

3.1 Feature Engineering

For the feature ‘rectal_temp’ we found that deviation from the normal temperature of 37.8-degree Celsius results in a higher probability of the target class to be ‘died’ or ‘euthanized’ as shown in the picture below:



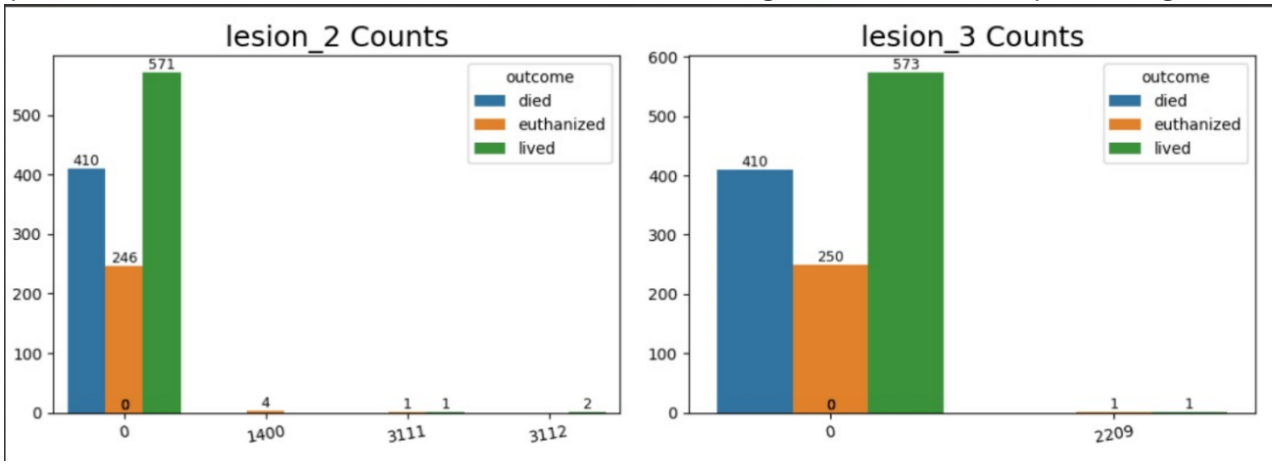
Hence, we created a new feature called 'dev_from_normal' which replaces every rectal temperature with its difference from normal temperature. This new feature captures extent to which the horse's rectal temperature deviates from baseline.

We also observed that if lesion_2 is not zero, outcome is 'lived' or 'euthanized' (see the figure in data cleaning section). Thus, we can simplify lesion_2 to be binary (1 if its value is > 0, else 0).

3.2 Data Cleaning

We've chosen to omit the 'hospital_number' feature from our analysis. While it signifies the hospital where the horse received treatment, our examination revealed its lack of meaningful association with the horse's survival status. Additionally, hospital_number, while having numerical values, is a categorical feature. But treating 'hospital_number' as a categorical feature presents challenges. With 255 unique values, employing one-hot encoding would introduce an impractically large number of features to the dataset.

We have also omitted lesion_3 because of its similar distribution to lesion_2. Moreover, a chi-squared test also reveals that lesion_3 does not have a significant relationship with target variable.

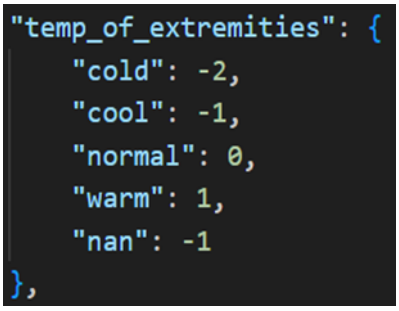


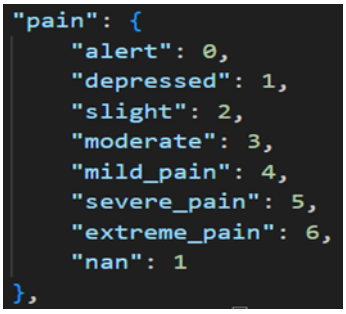
The train dataset has a lot of missing values in the categorical features which is dealt with using mode imputation. We are also using external dataset which contains missing values in the numerical features, which is dealt with using median imputation.

3.3 Encoding Used

We opted for ordinal encoding for certain categorical variables, such as 'temp_of_extremeties', 'peripheral_pulse','capillary_refill_time','pain','peristalsis','abdominal_distension','nasogastric_reflux', 'rectal_exam_feces', and 'abdomo_appearance'. This decision was driven by the desire to reduce the feature set, which would have otherwise been expanded considerably if we had used one-hot encoding.

Furthermore, we considered the inherent order among the categories within these variables. For instance, we assigned ordinal values based on their semantic meaning. For instance, 'normal' values were assigned a numerical value of 0, while values indicating 'less than normal' were assigned negative values, and those indicating 'more than normal' were assigned positive values.
Additionally, we chose to encode other features in a manner where all categories were represented by non-negative values, as we felt this better captured their meaning. For example, comparing 'temp_of_extremities' and 'pain':





For null values, we followed the policy of mode imputation, where the missing values are filled with the most commonly occurring category.
The features 'abdomen' and 'mucous_membrane' need to be one-hot encoded because we couldn't observe any order among the categories. Binary features such as 'age', 'surgical_lesion', 'surgery', and 'cp_data' are simply encoded as 0 or 1 depending on their meaning.

3.4 K-Fold Cross Validation & External Dataset

To maximize the utility of the limited training data at our disposal, we employed a five-fold cross-validation technique. This method involved partitioning the training data into five subsets or folds, wherein four folds were utilized for training while the remaining fold was reserved for validation. This iterative process yielded five unique combinations of training and validation sets.
Furthermore, to enhance the robustness of our model training, we augmented each training set with supplementary data extracted from an external dataset. This external dataset constitutes the foundational data utilized in the context of the Kaggle competition under consideration. Particular attention was given to ensure that this supplementary data was exclusively appended to the training set of each cross-validation combination. This precautionary measure aimed to preserve the integrity of the validation set, thereby facilitating an accurate evaluation of the model's generalization performance across the competition dataset.
External Dataset source: https://www.kaggle.com/datasets/yasserh/horse-survival-dataset

3.5 Hyperparameter Tuning

Utilizing Optuna for hyperparameter optimization across all models within our project, we pursued the maximization of the average validation F1 score as the objective function for each combination in K-Fold Cross Validation. Notably, we incorporated the random seed as a hyperparameter in this optimization process. Although this approach deviates from the conventional assumption underlying Optuna's methodology, wherein it typically assumes a linear or logarithmic variation in hyperparameter values, we deemed it necessary to expedite the exploration of diverse random seed values alongside different combinations of hyperparameters. Despite its departure from the ideal optimization scenario, this compromise was deemed pragmatic and worthwhile for our objectives.

4. Models Used

4.1 Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) or the average prediction (regression) of the individual trees. It's an extension of bagging that builds many uncorrelated decision trees and then averages them to improve accuracy and control over-fitting.
Key Features:

- Ensemble Method: Combines multiple decision trees to improve predictive performance.
- Random Feature Selection: Each tree in the forest is trained on a random subset of features, adding randomness and reducing correlation between trees.
- Bootstrap Sampling: Each tree is trained on a bootstrap sample (randomly sampled with replacement) from the training data.
- Voting Mechanism: For classification tasks, the final prediction is determined by a majority vote of all trees. For regression tasks, it's the average prediction of all trees.
- Effective for Large Datasets: Random Forests can handle large datasets with high dimensionality.

4.2 Bernoulli Naive Bayes

Bernoulli Naive Bayes is a variant of the Naive Bayes classifier that assumes features are binary-valued (e.g., presence or absence of a feature). It's particularly useful for text classification tasks, where each term's presence or absence in a document is used as a feature.
Key Features:

- Naive Bayes Assumption: Assumes that features are conditionally independent given the class label.
- Binary Features: Works well with binary features, such as in text classification with bag-of-words representation.
- Simple and Efficient: Computationally efficient and easy to implement.

- Requires Feature Independence: Performs best when the features are indeed conditionally independent.
- Can Handle Large Datasets: Suitable for large datasets with high-dimensional feature spaces.

4.3 Nearest Centroid

Nearest Centroid is a simple and efficient classification algorithm that represents each class by the centroid (mean) of its members in the feature space. During prediction, the class of a test sample is determined by the nearest centroid based on a distance metric (typically Euclidean distance).
Key Features:

- Centroid Representation: Each class is represented by the mean vector of its training samples.
- Distance Metric: Uses a distance metric (e.g., Euclidean distance) to determine the closest centroid to a test sample.
- Classification Rule: Assigns the class label of the nearest centroid as the predicted class for the test sample.
- Interpretability: Nearest Centroid is straightforward and interpretable, making it suitable for quick exploratory analysis.
- Sensitive to Outliers: Can be sensitive to outliers due to its reliance on mean vectors.

4.3 XGBoost

XGBoost, short for Extreme Gradient Boosting, is a popular and highly efficient implementation of gradient boosting machines. It was developed by Tianqi Chen and is known for its scalability and performance. XGBoost has become a go-to choice for structured/tabular data problems and has been widely used in machine learning competitions due to its ability to produce high-quality predictions.
Key Features:

- Regularization: XGBoost provides options for regularization to prevent overfitting, such as L1 (Lasso) and L2 (Ridge) regularization.
- Parallelization: It supports parallel and distributed computing, making it efficient for large datasets.
- Tree Pruning: Implements advanced techniques like tree pruning to control model complexity and enhance performance.
- Handling Missing Values: XGBoost has built-in capabilities to handle missing values within the dataset.
- Cross-validation: Includes tools for cross-validation to help with parameter tuning and model evaluation.

4.4 LightGBM

LightGBM is another popular gradient boosting framework developed by Microsoft. It is designed for efficiency and optimized for large datasets. LightGBM uses a histogram-based approach for

binning continuous feature values into discrete bins, which speeds up training and reduces memory usage.
Key Features:

- Histogram-based Splitting: Uses histogram-based algorithms for finding the best split, which leads to faster training times.
- Leaf-wise Tree Growth: LightGBM grows trees leaf-wise (as opposed to level-wise), which can lead to better accuracy.
- Gradient-based One-Side Sampling (GOSS): Optimizes the training process by focusing on instances with large gradients.
- Exclusive Feature Bundling: Efficiently handles categorical features by bundling them into exclusive feature groups.

4.5 HistGB

HistGB, or Histogram-based Gradient Boosting, is an umbrella term that can refer to any gradient boosting implementation that utilizes histogram-based algorithms for splitting data points in decision trees. Both XGBoost and LightGBM fall under this category due to their use of histogram-based methods.
Key Features:

- Histogram-based Splitting: Utilizes histograms to bin and split feature values, making tree construction more efficient.
- Reduced Memory Footprint: Histogram-based methods often reduce memory usage compared to traditional tree-based methods.
- Scalability: Well-suited for large datasets and distributed computing environments.

4.6 Neural Networks

Neural Networks are Artificial Intelligence models that mimic the human brain's structure and function through the interlink of neurons (artificial neurons in Neural Networks), with layer analysing data input to map an output. Intricate patterns and relationships can be found through implementation of neural networks. Backpropagation are applied in neural networks to update internal parameters and minimize the differences between predicted and observed results.

To predict the class label of horse health status (three-class classification), we utilize SoftMax activation functions, and further trained using Adam optimizer and Sparse Categorical Cross-Entropy Loss Function. Overall f1 score through neural network did not yield significant improvement as compared to other models, the limited size of the training dataset and the high dimensionality of the feature space may have hindered its ability to achieve significant improvements in the f1 score.

4.7 Autoencoder Neural Networks

Autoencoder Neural Networks represent a class of artificial neural networks employed predominantly in unsupervised learning tasks, with a focus on dimensionality reduction and feature learning. Comprising both an encoder and a decoder component, the autoencoder framework operates by first reducing input data into a lower-dimensional representation, known as encoding, through the encoder module. This encoding encapsulates essential characteristics of the input data while reducing its complexity. Subsequently, the decoder module endeavours to reconstruct the data back into its original format. The primary objective of the autoencoder paradigm revolves around minimizing reconstruction error, serving as a metric for assessing the disparity between the initial data and its reconstructed counterpart.

While conventional neural networks also perform dimensionality reduction prior to making predictions, autoencoders possess the capability to train on both the training and test data as they do not require the final labels for reconstruction loss computation. Leveraging this characteristic, we endeavoured to employ autoencoders to address the challenge posed by the small training set in our study. However, despite our efforts, the efficacy of the autoencoder was limited by the diminutive size of the dataset and the disparate nature of its columns. Consequently, the autoencoder failed to provide a substantial improvement over the performance achieved by a regular neural network, yielding comparable accuracy outcomes.

4.8 Self-Attention Neural Networks

Distinguished by their self-attention mechanisms, these models excel in capturing intricate dependencies within input sequences. Through self-attention mechanisms, each element within the input sequence iteratively evaluates the significance of all other elements, assigning varying weights based on their contextual relevance. This real-time computation of attention weights at each layer enables the model to discern salient segments within the input sequence while attenuating extraneous or redundant information. Moreover, the inherent parallelization capabilities afforded by self-attention mechanisms facilitate expedited training on large-scale datasets and enhance inference speed vis-à-vis conventional sequential models.

While the integration of self-attention mechanisms promises to enhance the model's capacity to capture nuanced relationships between input elements, the attendant computational demands pose a formidable challenge. The extensive data requirements for training transformer models present a notable impediment, rendering their application impractical for datasets of limited size. As anticipated, empirical findings corroborated this assertion, with the self-attention neural network emerging as the least performant model among those evaluated in our study.

5. Ensemble Technique

5.1 Voting Classifier

The voting classifier is a commonly used method in ensemble learning, involving training several base models separately and making predictions based on a majority vote or weighted average of their predictions. This method is especially beneficial in classification assignments, with each foundational model serving as a "voter" that aids in making the prediction. The voting classifier can merge different classifiers like decision trees, support vector machines, or logistic regression to form a varied ensemble. By combining the predictions of various models, the voting classifier typically outperforms individual base models in terms of both training and validation F1 scores.

5.2 Stacking Classifier

Stacking, also referred to as stacked generalization, represents an ensemble learning approach that amalgamates the predictions of multiple base models through the utilization of a meta-model. During the stacking process, diverse base models such as decision trees, support vector machines, or neural networks are individually trained on the dataset. Subsequently, the predictions generated by the base models along with the original features are employed as input for the meta-model. The meta-model is then trained to amalgamate the predictions of the base models while potentially incorporating additional features, culminating in the production of the final prediction.

In our implementation, we employed the Stacking Classifier from the scikit-learn ensemble module to instantiate the stacking classifier technique. This method amalgamates the predictions of various base estimators, specified in the estimator's parameter, utilizing a meta-estimator delineated in the final_estimator parameter. In this context, the base models are predefined classifiers, while the top-level model alternated between a Decision Tree Classifier and XGBoost. Initially, a decision tree classifier was selected to facilitate interpretability, with XGBoost subsequently explored due to its renowned efficacy in handling small datasets. Subsequently, the stacking classifier underwent training on the training data and was deployed to predict outcomes for both the training and validation datasets.

In addition to leveraging the predictions of the base models for the stacked classifier, we incorporated supplementary columns of data with the aim of enabling the stacked ensemble model to discern which classifier's result to prioritize under varying circumstances. The selection of these additional columns was informed by an explainability analysis conducted using Shap on our preceding models.

5.3 Neural Network as Final Estimator

To utilize the ensemble technique using neural network, different base models like random forest, nearest centroid, and Bernoulli naive Bayes classifiers are first trained. These various models, possessing different architectures, training subsets, or hyperparameters, combine to enhance the

ensemble's predictive power. After being trained, their results are combined to create the ensemble model, using techniques such as averaging forecasts or majority voting. Making sure there is a variety in the base models, obtained through various structures, training methods, or data samples, is important to optimize the performance of the ensemble and lower the chances of models making the same mistakes.

5.4 Observation with Ensemble Technique

In our investigation into the application of ensemble techniques on our training dataset, we observed a consistent trend of diminished performance across all ensemble methodologies compared to the performance of the original base model. This decline in performance raises concerns regarding potential overfitting, particularly given the constrained size of the training dataset. Furthermore, the diminutive size of the testing dataset exacerbated the challenge by yielding unpredictable test F1 scores in contrast to the more stable validation F1 scores. Consequently, we opted to prioritize the adoption of simpler models guided by the principle of Occam's Razor, to mitigate the complexities associated with overfitting and the volatility of test F1 scores.

	Training F1 Score	*Validation F1 Score*
Voting Classifier	0.73178	0.70040
Stacking Classifier	0.73178	0.70040
Decision Tree with Other Data	0.76417	0.7125
Neural Network as Final Estimator	0.7831	0.6841

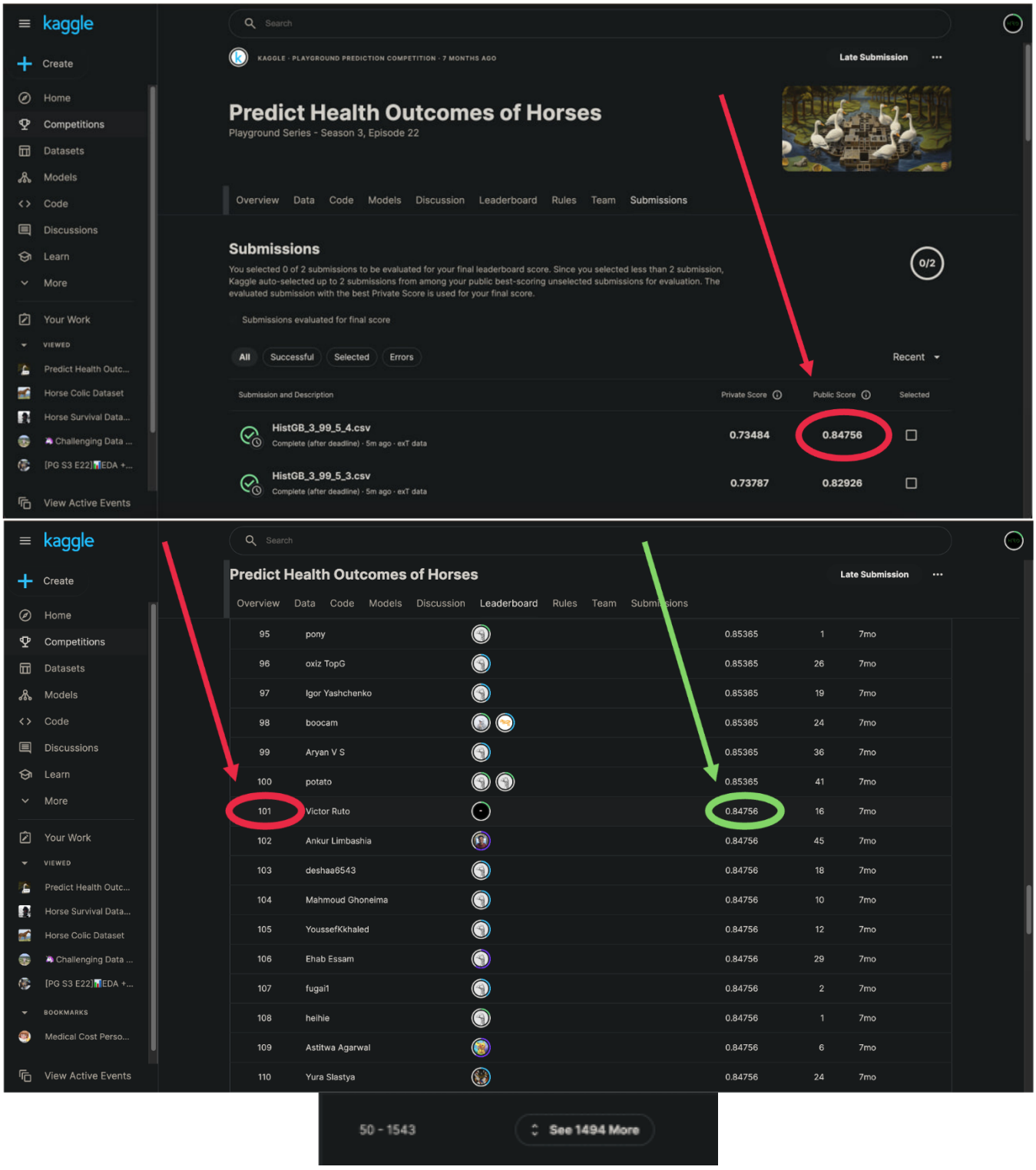
6. Result

In broad terms, the ensemble models based on tree algorithms, including Random Forest, Hist Gradient Boosting, XGBoost, and LightGBM, exhibited comparable F1 scores during both training and validation phases. Nevertheless, due to the limited size of the public test dataset, constituting 20% of the overall test dataset, equating to 164 instances, considerable disparities emerged in the test F1 scores among these models. Notably, the test F1 scores exhibited notable fluctuations even within iterations of identical models across diverse experiments. The attainment of our ultimate test F1 score necessitated the execution of multiple experiments employing varied model combinations.

Conversely, our proprietary ensemble models demonstrated analogous validation F1 scores to those observed in the individual tree-based models aforementioned. However, their performance on the test dataset registered a decline, aligning with the principle of Occam's Razor. A comprehensive breakdown of the outcomes from each experiment is provided in the appendix.

The optimal model identified in our analysis was a Hist Gradient Descent Model configured with specific hyperparameters: Max Depth set to 3, Max Iterations set to 99, Min Samples per Leaf set to 5, L2 Regularization set to 4.2e-6, and Learning Rate set to 0.1. Remarkably, this model attained a commendable Test F1 score of **0.84756** on the Public Test Set. Consequently, our performance

yielded a rank of 101 out of 1543 total participants, positioning us within the **top 6.7%** of the competition cohort.



7. Limitations and Challenges

There are three main challenges we faced during this project: small training dataset, small test dataset, and the large number of nan values.

1. **Small Training Dataset** – The training dataset comprised a modest 1235 observations distributed across 27 variables. Following one hot encoding, the feature space expanded to over 80 columns, thereby engendering a substantial search space for the model. In response, efforts were undertaken to curtail the dimensionality, resulting in a refinement to 35 columns through the judicious elimination of irrelevant features and the adoption of Binary Encoding and Ordinal Encoding techniques in lieu of one hot encoding wherever feasible. However, the restricted size of the dataset precluded the utilization of sophisticated deep learning methodologies such as Neural Networks and transformers. Furthermore, this constraint precipitated a tendency towards overfitting within various tree-based ensemble models, as evidenced by instances wherein the models exhibited a proclivity to memorize the training dataset in its entirety, thereby yielding a Train F1 Score of 1, rather than discerning the underlying patterns. Mitigation of this phenomenon was achieved through the imposition of constraints on tree depth and the number of estimators employed, supplemented by the integration of Optuna for hyperparameter optimization. Furthermore, the application of k-fold cross-validation techniques alongside an external dataset proved instrumental in managing this challenge.

2. **Large Number of NaN Values** – Both the training and test datasets exhibited a notable prevalence of NaN (Not a Number) values distributed throughout their respective variables. Dropping all rows containing NaN values would result in a reduction of the training dataset from 1235 to 771 rows, a diminution deemed unacceptable given its potential exacerbation of the aforementioned challenge. To address this issue, we opted to replace NaN values with the mode of their respective column. This approach was grounded in the rationale that the mode, representing the most frequently occurring value within a column, would diminish the likelihood of the model attributing undue significance to these imputed values. Notably, the columns containing NaN values exclusively comprised categorical variables. Although the primary dataset did not feature numerical columns with NaN values, such instances were encountered in the external dataset. In these cases, we substituted NaN values with the median value of their respective column. This choice was motivated by the sensitivity of the mean to outliers and the failure of the mode to account for the distribution of numerical values.

3. **Small Test Dataset** - The test dataset consists of a modest 824 rows, with the public test set on Kaggle utilizing approximately 20% of this dataset, resulting in an approximate subset of 164 rows. The limited size of this dataset poses a significant challenge, as it leads to substantial variability in the public test F1 score with only a small number of samples. Consequently, the choice of random seed employed in the dataset has a notable impact on the test F1 score. Given the competitive nature of the leaderboard, this dependency on the random seed significantly affects the ranking position. To address this issue, we conducted numerous experiments

employing various models and techniques. Notably, we incorporated the random seed as a parameter in the Optuna framework for hyperparameter tuning, seeking to mitigate the influence of random seed variability on model performance and leaderboard ranking stability.

8. Conclusion

In wrapping up our project, we've gained invaluable insights into the world of machine learning model selection and optimisation. We've discovered that while neural networks have their strengths, tree-based ensemble models often outshine them when it comes to deciphering complex patterns in tabular data. Our experiments with Auto-ML have shown us its limitations in preventing overfitting, highlighting the need for hands-on fine-tuning. Diving into the documentation of XGBoost, LightGBM, and HistGB has been eye-opening, giving us a deeper understanding of these powerful algorithms and how to use them effectively. And perhaps most importantly, we've learned that when it comes to preventing overfitting, setting constraints on tree depth proves more fruitful than restricting the number of estimators.

In conclusion, through the journey of our project, although we have understood what ML models work best for our given situation, what we took back is more valuable as it transcends all the various contexts - we learnt how to learn and that sometimes the key to a problem is to not overcomplicate things. In line with Occam's Razer, the more explicit but yet concise a problem and its pipeline is, the less is left to the unknown. This was evident when we used overengineered models but the key was in simplicity. This project has not only advanced our understanding of machine learning but also set a foundation for not just this horse health outcomes but in further research as well.

9. Contribution

Team Members	Contribution
Dhanyamraju Harsh Rao (U[illegible])	Model Implementation, Report and Video
Parashar Kshitij (U[illegible])	Model Implementation, Report
Malavade Sanskar Deepak (U[illegible])	Data Processing, Report
Chen Kian Leong (U[illegible])	Model Implementation, Report
Mitra Ren Sachithananthan (U[illegible])	Feature Engineering, Report

10. Appendix

Dataset					Result		
Ordinal Encoding Used ?	**External Data Used**	**EnsembleModel**	**With some data**	**Models Used with Hyperparameters**	**Train F1**	**Valid F1**	**Test F1 Public**
No	-	Decision Tree	Yes	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=23), Nearest Centroid (metric='correlation'), Bernoulli Naive Bayes	0.76	0.71	0.74
No	-	-	-	Random Forest (max_depth=23, random_state=123, criterion='entropy', n_estimators=28)	1	0.74	0.71
Yes (All Positive)	-	Decision Tree	Yes	Random Forest (max_depth=7, random_state=123, criterion='gini', n_estimators=4), Nearest Centroid (metric='correlation'), Bernoulli Naive Bayes	0.79	0.7	0.725
Yes (All Positive)	-	-	-	NN(35,12,6,3) All Sigmoid	-	0.63	0.71
No	-	Decision Tree	Yes	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=4), Nearest Centroid (metric='correlation'), Bernoulli Naive Bayes	0.74	0.72	0.701
No	-	Decision Tree	Yes	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=4), Nearest Centroid (metric='correlation'), Bernoulli Naive Bayes, XGBoost (n_estimators=117, max_depth=4, learning_rate=0.25993)	0.996	0.733	0.78
No	-	Voting Classifier (weights optimised by Optuna)	No	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=4), Nearest Centroid (metric='correlation'), Bernoulli Naive Bayes, XGBoost (n_estimators=117, max_depth=4, learning_rate=0.25993)	0.828	0.741	0.75
No	-	Voting Classifier (weights optimised by Optuna)	No	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=4), Nearest Centroid (metric='correlation'), Bernoulli Naive Bayes, XGBoost (n_estimators=117, max_depth=4, learning_rate=0.25993), LightGBM ('n_estimators': 99, 'max_depth': 8, 'learning_rate': 0.07516061619130993), HistGBClassifier ('learning_rate': 0.030335798022501945, 'max_iter': 140, 'max_depth': 10, 'min_samples_leaf': 16, 'l2_regularization': 0.0002309029429926762)	1	0.741	0.7561
No	-	Decision Tree	Yes	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=4), Nearest Centroid (metric='correlation'), Bernoulli	1	0.741	0.7744

				Naive Bayes, XGBoost (n_estimators=117, max_depth=4, learning_rate=0.25993), LightGBM ('n_estimators': 99, 'max_depth': 8, 'learning_rate': 0.07516061619130993), HistGBClassifier ('learning_rate': 0.030335798022501945, 'max_iter': 140, 'max_depth': 10, 'min_samples_leaf': 16, 'l2_regularization': 0.0002309029429926762)			
No	-	Voting Classifier (weights optimised by Optuna)	No	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=4), Nearest Centroid (metric='correlation'), Bernoulli Naive Bayes, XGBoost {'n_estimators': 110, 'max_depth': 3, 'learning_rate': 0.11846833545955368}, LightGBM ('n_estimators': 99, 'max_depth': 8, 'learning_rate': 0.07516061619130993), HistGBClassifier ('learning_rate': 0.030335798022501945, 'max_iter': 140, 'max_depth': 10, 'min_samples_leaf': 16, 'l2_regularization': 0.0002309029429926762)	0.9989	0.757	0.7683
Yes (Positive & Negative with nan values replaced by mode)	-	Voting Classifier (weights optimised by Optuna)	No	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=7), Nearest Centroid (metric='braycurtis'), Bernoulli Naive Bayes, XGBoost {'n_estimators': 16, 'max_depth': 5, 'learning_rate': 0.129813}, LightGBM ('n_estimators': 9, 'max_depth': 6, 'learning_rate': 0.232923621)	0.82	0.777	0.75
Yes (Positive & Negative with nan values replaced by mode)	-	Voting Classifier (weights optimised by Optuna)	No	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=7), Nearest Centroid (metric='braycurtis'), Bernoulli Naive Bayes, XGBoost {'n_estimators': 16, 'max_depth': 5, 'learning_rate': 0.129813}, LightGBM ('n_estimators': 9, 'max_depth': 6, 'learning_rate': 0.232923621), HistGBClassifier ('learning_rate': 0.02312436, 'max_iter': 249, 'max_depth': 4, 'min_samples_leaf': 10, 'l2_regularization': 2.119e-6)	0.833	0.7813	0.7439
Yes (Positive & Negative with nan values replaced by mode)	-	Decision Tree	Yes	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=7), Nearest Centroid (metric='braycurtis'), Bernoulli Naive Bayes, XGBoost {'n_estimators': 16, 'max_depth': 5, 'learning_rate': 0.129813}, LightGBM ('n_estimators': 9, 'max_depth': 6, 'learning_rate': 0.232923621), HistGBClassifier ('learning_rate': 0.02312436, 'max_iter': 249, 'max_depth': 4, 'min_samples_leaf': 10, 'l2_regularization': 2.119e-6)	0.99	0.757	0.78
Yes (Positive & Negative with nan values replaced by mode)	-	Decision Tree	No	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=7), Nearest Centroid (metric='braycurtis'), Bernoulli Naive Bayes, XGBoost {'n_estimators': 16, 'max_depth': 5,	0.83	0.7813	0.7138

				'learning_rate': 0.129813}, LightGBM ('n_estimators': 9, 'max_depth': 6, 'learning_rate': 0.232923621), HistGBClassifier ('learning_rate': 0.02312436, 'max_iter': 249, 'max_depth': 4, 'min_samples_leaf': 10, 'l2_regularization': 2.119e-6)			
No (used K-Fold)	-	Voting Classifier (weights optimised by Optuna)	No	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=43), Nearest Centroid (metric='braycurtis'), Bernoulli Naive Bayes, XGBoost {'n_estimators': 14, 'max_depth': 5, 'learning_rate': 0.4964}, LightGBM ('n_estimators': 20, 'max_depth': 2, 'learning_rate': 0.44189), HistGBClassifier ('learning_rate': 0.2321, 'max_iter': 69, 'max_depth': 2, 'min_samples_leaf': 19, 'l2_regularization': 6.6e-5)	0.866	0.7812	0.768
No (used K-Fold)	-	Decision Tree	Yes	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=43), Nearest Centroid (metric='braycurtis'), Bernoulli Naive Bayes, XGBoost {'n_estimators': 14, 'max_depth': 5, 'learning_rate': 0.4964}, LightGBM ('n_estimators': 20, 'max_depth': 2, 'learning_rate': 0.44189), HistGBClassifier ('learning_rate': 0.2321, 'max_iter': 69, 'max_depth': 2, 'min_samples_leaf': 19, 'l2_regularization': 6.6e-5)	0.9585	0.7287	0.78
No (used K-Fold)	-	Decision Tree	Yes	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=43), Nearest Centroid (metric='braycurtis'), Bernoulli Naive Bayes, XGBoost {'n_estimators': 14, 'max_depth': 5, 'learning_rate': 0.4964}, LightGBM ('n_estimators': 20, 'max_depth': 2, 'learning_rate': 0.44189), HistGBClassifier ('learning_rate': 0.2321, 'max_iter': 69, 'max_depth': 2, 'min_samples_leaf': 19, 'l2_regularization': 6.6e-5), SVC(kernel='rbf')	0.9585	0.7287	0.78
No (used K-Fold)	-	Voting Classifier (weights optimised by Optuna)	No	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=43), Nearest Centroid (metric='braycurtis'), Bernoulli Naive Bayes, XGBoost {'n_estimators': 14, 'max_depth': 5, 'learning_rate': 0.4964}, LightGBM ('n_estimators': 20, 'max_depth': 2, 'learning_rate': 0.44189), HistGBClassifier ('learning_rate': 0.2321, 'max_iter': 69, 'max_depth': 2, 'min_samples_leaf': 19, 'l2_regularization': 6.6e-5), SVC(kernel='rbf')	0.817	0.777	0.78
No (used K-Fold)	-	XGBOOST	Yes	Random Forest (max_depth=5, random_state=123, criterion='gini', n_estimators=43), Nearest Centroid (metric='braycurtis'), Bernoulli Naive Bayes, XGBoost {'n_estimators': 14, 'max_depth': 5, 'learning_rate': 0.4964}, LightGBM ('n_estimators': 20, 'max_depth': 2,	0.957	0.7287	0.78

				'learning_rate': 0.44189), HistGBClassifier ('learning_rate': 0.2321, 'max_iter': 69, 'max_depth': 2, 'min_samples_leaf': 19, 'l2_regularization': 6.6e-5), SVC(kernel='rbf')			
Yes (Positive & Negative with nan values replaced by mode) and K-Fold	-	-	-	RandomForest(max_depth=7, random_state=1963, criterion='entropy', n_estimators=37)	0.836	0.7287	0.719
Yes (Positive & Negative with nan values replaced by mode) and K-Fold	-	-	-	XGBoost(n_estimators=19, max_depth=2, learning_rate=0.257, objective='multi:softmax', num_class=3, random_state=1607338919)	0.7287	0.7489	0.75
Yes (Positive & Negative with nan values replaced by mode) and K-Fold	-	-	-	LightGBM ('n_estimators': 14, 'max_depth': 4, 'learning_rate': 0.3312371944460501, random_state=2559945896)	0.85	0.76	0.78658
Yes (Positive & Negative with nan values replaced by mode) and K-Fold	-	-	-	HistGB('learning_rate': 0.2574262422842679, 'max_iter': 16, 'max_depth': 3, 'min_samples_leaf': 11, 'l2_regularization': 0.0004212419957410775, 'random_state': 362310	0.82	0.755	0.80487
Yes (Positive & Negative with nan values replaced by mode) and K-Fold	-	Voting Classifier (weights optimised by Optuna)	No	RandomForest(max_depth=7, random_state=1963, criterion='entropy', n_estimators=37) XGBoost(n_estimators=19, max_depth=2, learning_rate=0.257, objective='multi:softmax', num_class=3, random_state=1607338919) LightGBM ('n_estimators': 14, 'max_depth': 4, 'learning_rate': 0.3312371944460501, random_state=2559945896) HistGB('learning_rate': 0.2574262422842679, 'max_iter': 16, 'max_depth': 3, 'min_samples_leaf': 11, 'l2_regularization': 0.0004212419957410775, 'random_state': 362310	0.79	0.7611	0.76291
Yes (Positive & Negative with nan values replaced by mode) and K-Fold	-	Decision Tree	Yes	RandomForest(max_depth=7, random_state=1963, criterion='entropy', n_estimators=37) XGBoost(n_estimators=19, max_depth=2, learning_rate=0.257, objective='multi:softmax', num_class=3, random_state=1607338919) LightGBM ('n_estimators': 14, 'max_depth': 4, 'learning_rate': 0.3312371944460501, random_state=2559945896) HistGB('learning_rate': 0.2574262422842679, 'max_iter': 16, 'max_depth': 3, 'min_samples_leaf': 11, 'l2_regularization': 0.0004212419957410775, 'random_state': 362310	0.836	0.7287	0.7256
Yes (Positive & Negative with nan values replaced by mode) and K-Fold	-	Decision Tree	Yes	RandomForest(max_depth=7, random_state=1963, criterion='entropy', n_estimators=37) XGBoost(n_estimators=19, max_depth=2, learning_rate=0.257, objective='multi:softmax', num_class=3, random_state=1607338919)	0.846	0.72	0.7856

				LightGBM ('n_estimators': 14, 'max_depth': 4, 'learning_rate': 0.3312371944460501, random_state=2559945896) HistGB('learning_rate': 0.2574262422842679, 'max_iter': 16, 'max_depth': 3, 'min_samples_leaf': 11, 'l2_regularization': 0.0004212419957410775, 'random_state': 362310			
Yes (Positive & Negative with nan values replaced by mode) and K-Fold	-	XGBoost	Yes	RandomForest(max_depth=7, random_state=1963, criterion='entropy', n_estimators=37) XGBoost(n_estimators=19, max_depth=2, learning_rate=0.257, objective='multi:softmax', num_class=3, random_state=1607338919) LightGBM ('n_estimators': 14, 'max_depth': 4, 'learning_rate': 0.3312371944460501, random_state=2559945896) HistGB('learning_rate': 0.2574262422842679, 'max_iter': 16, 'max_depth': 3, 'min_samples_leaf': 11, 'l2_regularization': 0.0004212419957410775, 'random_state': 362310	0.84	0.76	0.7856
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	Random Forest(max_depth=4, random_state=8, criterion='log_loss', n_estimators=38)	0.726	0.7449	0.737
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	Bernoulli Naive Bayes	0.67	0.67	0.725
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	Bernoulli Naive Bayes	0.658	0.71	0.737
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	XGBClassifier(n_estimators=12, max_depth=3, learning_rate=0.340699296313225, objective='multi:softmax', num_class=3, random_state=1276008712)	0.76	0.74	0.774
Yes (Positive & Negative with nan values replaced by mode for categorical	Added External Data in Training	-	-	XGBClassifier(n_estimators=12, max_depth=3, learning_rate=0.340699296313225, objective='multi:softmax', num_class=3, random_state=1276008712)	0.76	0.73	0.762

and median for numerical) and K-Fold							
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	LightGBM('n_estimators': 19, 'max_depth': 3, 'learning_rate': 0.37055955822948267, 'random_state': 405834505)	0.83	0.72	0.8047
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	LightGBM('n_estimators': 19, 'max_depth': 3, 'learning_rate': 0.37055955822948267, 'random_state': 405834505)	0.84	0.74	0.79268
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	LightGBM('n_estimators': 19, 'max_depth': 3, 'learning_rate': 0.37055955822948267, 'random_state': 405834505)	0.84	0.72	0.78048
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	LightGBM('n_estimators': 19, 'max_depth': 3, 'learning_rate': 0.37055955822948267, 'random_state': 405834505)	0.84	0.72	0.82926
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	HistGB(learning_rate': 0.11891318344205706, 'max_iter': 99, 'max_depth': 3, 'min_samples_leaf': 5, 'l2_regularization': 4.22221435671052e-06, 'random_state': 20296})	0.92	0.76	0.82926
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	HistGB(learning_rate': 0.11891318344205706, 'max_iter': 99, 'max_depth': 3, 'min_samples_leaf': 5, 'l2_regularization': 4.22221435671052e-06, 'random_state': 20296})	0.94	0.72	0.823
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	HistGB(learning_rate': 0.11891318344205706, 'max_iter': 99, 'max_depth': 3, 'min_samples_leaf': 5, 'l2_regularization': 4.22221435671052e-06, 'random_state': 20296})	0.94	0.7	0.7865

Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	HistGB(learning_rate': 0.11891318344205706, 'max_iter': 99, 'max_depth': 3, 'min_samples_leaf': 5, 'l2_regularization': 4.22221435671052e-06, 'random_state': 20296})	0.93	0.73	0.82926
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	HistGB(learning_rate': 0.11891318344205706, 'max_iter': 99, 'max_depth': 3, 'min_samples_leaf': 5, 'l2_regularization': 4.22221435671052e-06, 'random_state': 20296)	0.93	0.7	**0.84756**
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	HistGB('learning_rate': 0.165132095750564, 'max_iter': 38, 'max_depth': 3, 'min_samples_leaf': 4, 'l2_regularization': 2.1615110163664448e-06, 'random_state': 3814)	0.86	0.75	0.81
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	HistGB('learning_rate': 0.165132095750564, 'max_iter': 38, 'max_depth': 3, 'min_samples_leaf': 4, 'l2_regularization': 2.1615110163664448e-06, 'random_state': 3814)	0.87	0.75	0.81
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	HistGB('learning_rate': 0.165132095750564, 'max_iter': 38, 'max_depth': 3, 'min_samples_leaf': 4, 'l2_regularization': 2.1615110163664448e-06, 'random_state': 3814)	0.88	0.71	0.792
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	HistGB('learning_rate': 0.165132095750564, 'max_iter': 38, 'max_depth': 3, 'min_samples_leaf': 4, 'l2_regularization': 2.1615110163664448e-06, 'random_state': 3814)	0.87	0.7	0.804
Yes (Positive & Negative with nan values replaced by mode for categorical and median for numerical) and K-Fold	Added External Data in Training	-	-	HistGB('learning_rate': 0.165132095750564, 'max_iter': 38, 'max_depth': 3, 'min_samples_leaf': 4, 'l2_regularization': 2.1615110163664448e-06, 'random_state': 3814)	0.88	0.71	0.84146