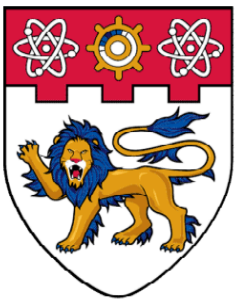


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC4000 Machine Learning
Group Project
Predict Health Outcome of Horses

Name	Matriculation Number	Contribution
Brandon Jang Jin Tian	U2220936G	LightGBM, Experimental Analysis of Chosen Solution, Conclusion
Chung Zhi Xuan	U2220300H	HistBoost, CatBoost, Ensemble Learning, Comparison
Ting Ruo Chee	U2220572C	Preprocessing, Feature Engineering, Ensemble Learning
Yau Jun Hao	U2220332F	EDA, Challenges, Random Forests
Pearlina Tan Qinlin	U2221690F	XGBoost, EDA

Abstract:

This report aims to provide a robust predictive model to predict the health outcome of horses in the Kaggle Playground Series - Season 3, Episode 22 competition. We experimented with well-known machine learning algorithms, including Random Forest, XGBoost, LightGBM, HistGradientBoosting, and CatBoost along with ensembling techniques such as stacking, to leverage the strengths of individual models. The performances of these model architectures were accessed in Kaggle to retrieve the official public leaderboard rankings and scores. Finally, to conclude the report, further insights on the evaluation of the effectiveness of different machine learning and ensembling architectures.

Table of Contents

1. Leaderboard At-a-Glance 3
1.1. Ranking Position 3
2. Competition Description 3
2.1. Dataset Description 3
2.2. Evaluation for Model Selection 4
3. Exploratory Data Analysis 4
3.1. Target Variable Analysis 4
3.2. Missing Data Analysis 5
3.3. Numerical Variable Analysis 6
3.4. Categorical Variable Analysis 7
4. Challenges of the Problem 9
5. Proposed Solution 9
5.1. Preprocessing 9
5.2. Feature Engineering 10
5.3. Methodologies 10
5.3.1. Random Forest Classifier 10
5.3.2. Light Gradient-Boosting Machine (LightGBM) 11
5.3.3. Histogram-Based Gradient-Boosting (HistBoost) 12
5.3.4. Extreme Gradient-Boosting (XGBoost) 13
5.3.5. Categorical Boosting (CatBoost) 13
5.3.6. Ensemble Learning 14
6. Experimental Study 15
6.1. Experimental Analysis of Chosen Solution 15
6.1.1. Validation Loss vs Boosting Rounds 16
6.1.2. Validation Accuracy for Each Hyperparameter Combination 16
6.1.3. Feature Importance 17
6.2. Chosen Solution Novelty 17
6.3. Comparison with Other Models 18
7. Conclusion 19
8. References 20

1. Leaderboard At-a-Glance

Evaluation Score

The leaderboard score with the Light Gradient-Boosting Machine (LightGBM) is as follows:

Figure 1: Evaluation Score

1.1. Ranking Position

Rank = 101/1543 = 6.55% (Top 6%)

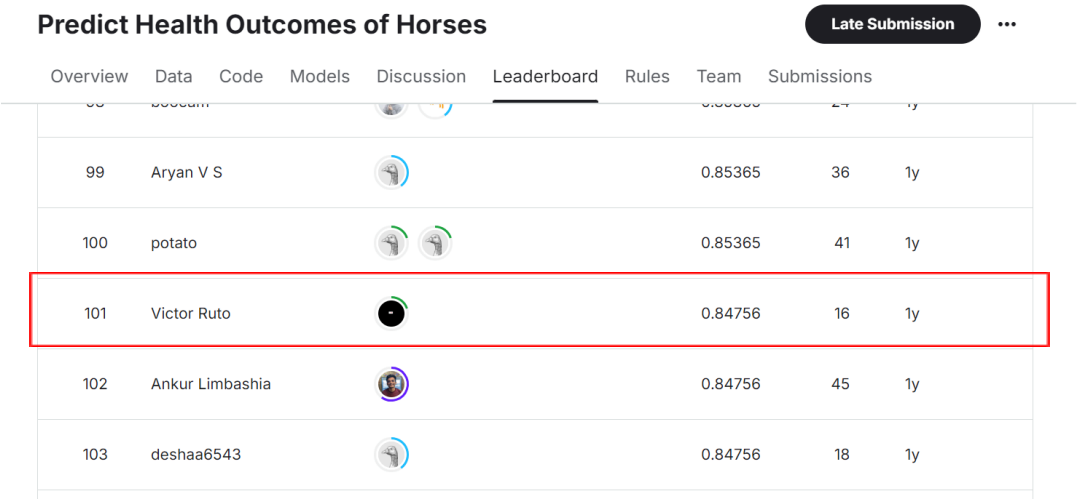


Figure 2: Ranking Position

2. Competition Description

The competition's objective is to predict horse health outcomes (lived, died, euthanized) based on various medical indicators.

2.1. Dataset Description

Feature	Description
id	The ID given to each hospital visit.
surgery	Indicates whether the horse required surgery.
age	The age of the horse.
hospital_number	The hospital case number assigned to the horse.
rectal_temp	The horse's rectal temperature in degrees Celsius.
pulse	The horse's pulse in beats per minute (bpm).
respiratory_rate	The horse's respiratory rate.
temp_of_extremities	Indicates the horse's peripheral circulation.
peripheral_pulse	The quality of circulation or perfusion of the horse.
mucuous_membrane	Measures the colour of the horse's mucous membranes.

capillary_refill_time	Indicates the quality of circulation within external capillaries. The longer the time taken, the poorer the circulation.
pain	The horse's pain level.
peristalsis	Indicates the level of activity in the horse's gut.
abdominal_distention	The level of abdominal distention within the horse.
nasogastric_tube	Measures the amount of gas released by the horse, if any.
nasogastric_reflux	Measures the amount of reflux from the horse's intestines, if any.
nasogastric_reflux_ph	The pH of the horse's nasogastric reflux, if any.
rectal_exam_feces	The amount of faeces relieved by the horse.
abdomen	Indicates the condition of the horse's abdomen.
packed_cell_volume	The number of red blood cells by volume in the horse's blood.
total_protein	The amount of protein by volume in the horse's blood.
abdomo_appearance	Appearance of fluid obtained from the horse's abdomen.
abdomo_protein	The amount of protein by volume in the horse's abdominal fluid.
surgical_lesion	Indicates whether the lesion was surgical.
lesion_1	Describes the first lesion's site, type, subtype, and description. Takes the value 0 if there is no lesion.
lesion_2	Describes the second lesion's site, type, subtype, and description.
lesion_3	Describes the third lesion's site, type, subtype, and description.
cp_data	Indicates whether pathology data was provided for this case.
outcome	States whether the horse 'lived', 'died', or was 'euthanized'; the **target variable**.

2.2. Evaluation for Model Selection

We experimented with and compared several machine learning models before selecting the one with the best performance score for the Kaggle competition. The scores are evaluated based on micro-averaged F1-score between the predicted and the actual values.

$$\text{Micro F1 Score} = \frac{\text{Net } TP}{\text{Net } TP + \frac{1}{2}(\text{Net } FP + \text{Net } FN)}$$

Figure 3: Formula for Micro-Averaged F1-Score

3. Exploratory Data Analysis

We conducted Exploratory Data Analysis (EDA) to explore the dataset.

3.1. Target Variable Analysis

We first look into the distribution of our target variable.

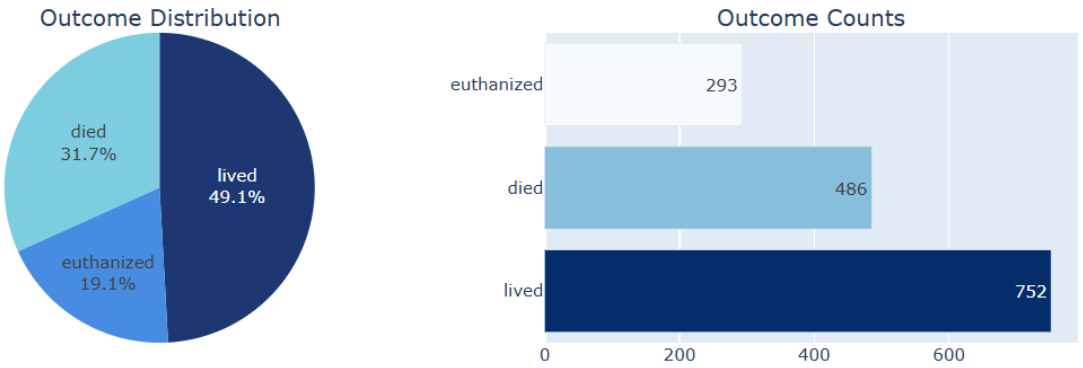


Figure 4: Distribution of Target Variable

The pie chart indicates a slight imbalance between the three classes, with approximately 50% of the data belonging to the 'lived' class, 30% to the 'died' class, and 20% to the 'euthanized' class.

3.2. Missing Data Analysis

We assessed the missing data for each feature, and the table below presents the top 12 features with the highest percentage of missing values:

	dtypes	missing#	missing%		dtypes	missing#	missing%
abdomen	object	329	0.214892	**peripheral_pulse**	object	129	0.084259
rectal_exam_feces	object	292	0.190725	**nasogastric_reflux**	object	127	0.082952
nasogastric_reflux_ph	float64	246	0.160679	**pain**	object	99	0.064664
abdomo_appearance	object	213	0.139125	**temp_of_extremities**	object	95	0.062051
abdomo_protein	float64	198	0.129327	**abdominal_distention**	object	79	0.051600
nasogastric_tube	object	183	0.119530	**mucous_membrane**	object	68	0.044415

Figure 5: Table of Missing Values

It is notable that both categorical and numeric features constrain missing values. Hence, we conducted further analysis on the features to determine the most appropriate strategy for imputing these missing values.

By plotting the correlation matrix, several valuable insights are revealed:

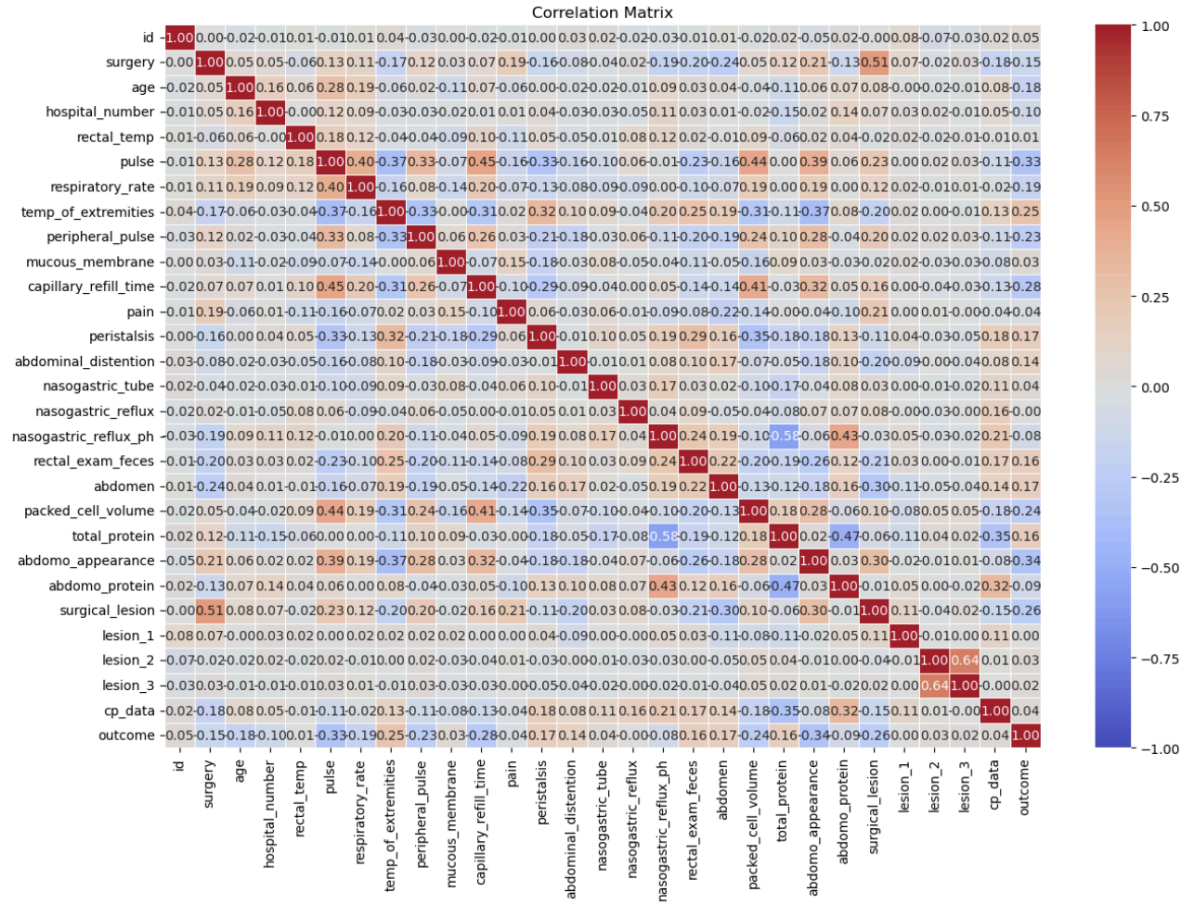
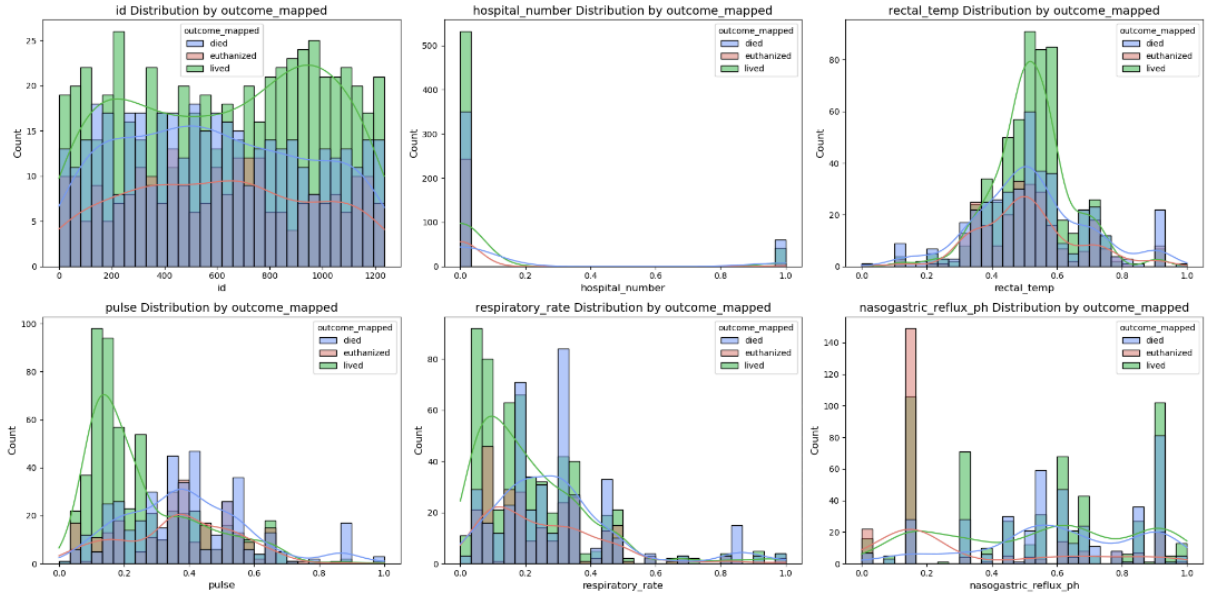


Figure 6: Correlation Matrix

Pain, rectal temperature, and surgical lesions emerge as strong predictors of health outcomes. Moderate correlations between features like pulse and respiratory rate or total protein and packed cell volume suggest potential redundancies that can be addressed through feature engineering. Weak correlations across many variables imply that the dataset likely benefits from including multiple independent features in the model to capture nuanced patterns.

3.3. Numerical Variable Analysis

We further analyse the numeric variables in the dataset. By plotting pairplot for numeric variables we can observe key insights into the relationships between the numerical features and the health outcomes:



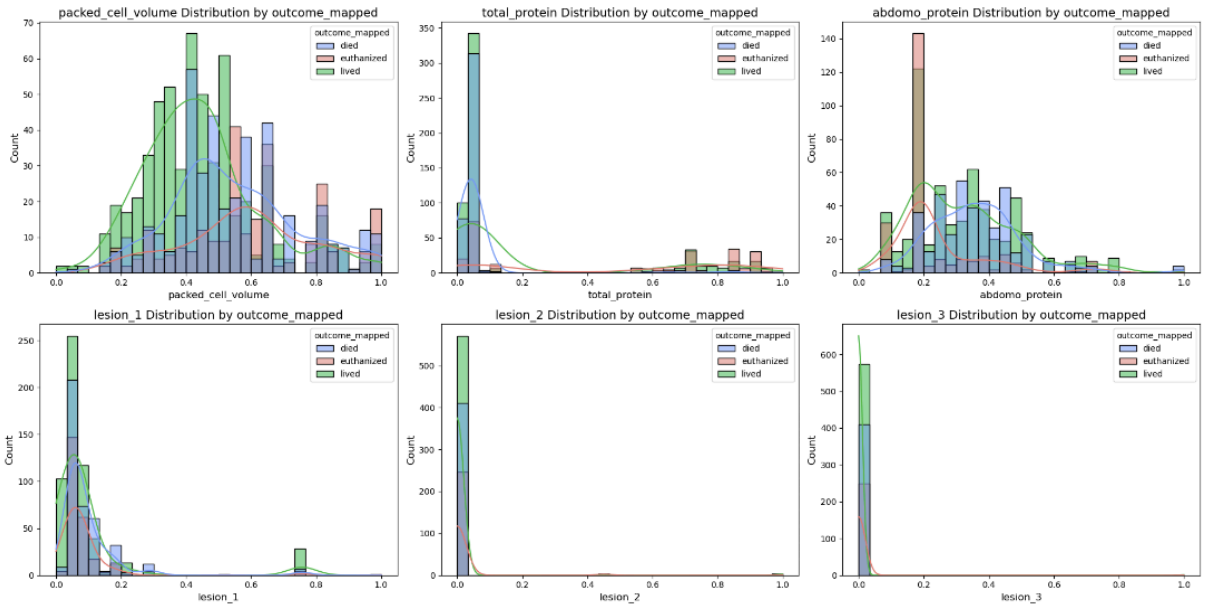


Figure 7: Pair Plot for numerical variables

Variables like rectal_temp, pulse, respiratory_rate, packed_cell_volume, and abdomo_protein show meaningful differences among health outcomes, suggesting they could be strong predictors.

Some variables, such as hospital_number and id, do not appear to provide significant distinctions between the outcome groups and may not be useful for model training.

Pulse and respiratory_rate consistently show higher values for poor health outcomes (died, euthanized), indicating their importance in predicting distress or fatal outcomes.

All three lesion variables show similar distributions, with most values being concentrated at low levels. There are no significant distinctions in outcomes, implying that lesion variables might have limited direct influence on prediction but could contribute when used with other variables.

3.4. Categorical Variable Analysis

We visualised the count plots for categorical features in both the training and test sets. Some notable features are highlighted below:

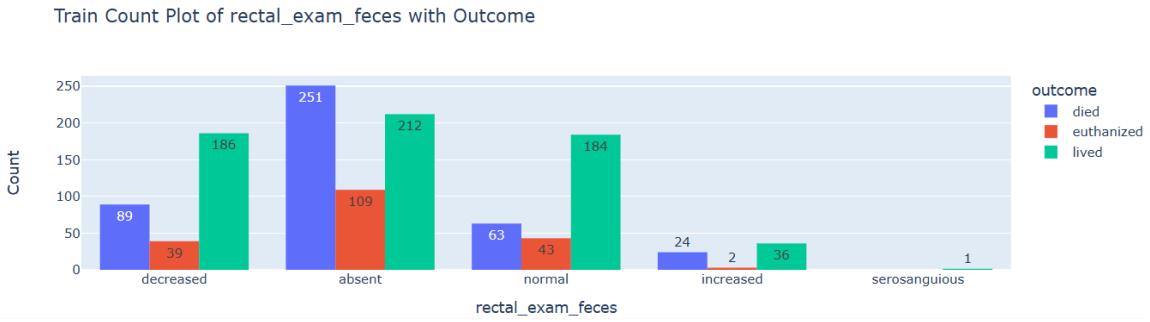


*Figure 8: Distribution of **rectal_exam_feces** by Target Classes (Train)*

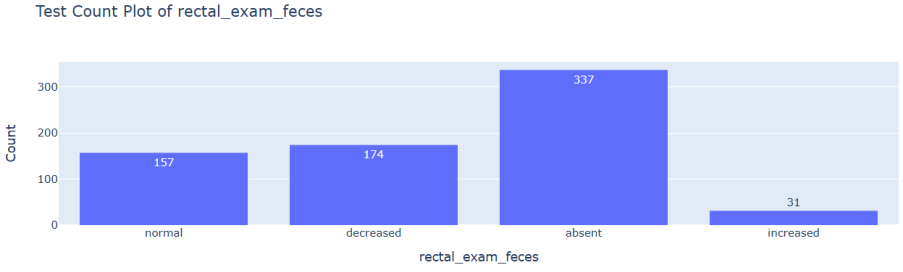


*Figure 9: Distribution of **rectal_exam_feces** (Test)*

We observed an extreme class in the **rectal_exam_feces** feature that appears exclusively in the training set but not the test set. There is also a slight imbalance between the classes.

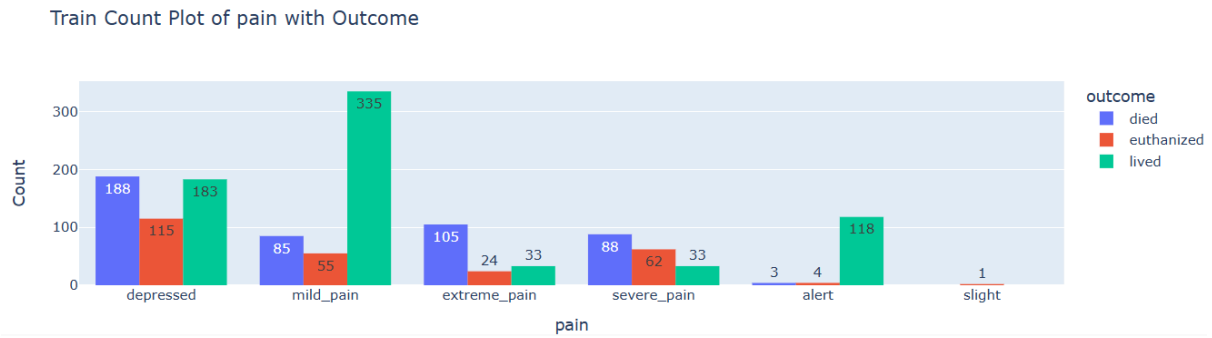


*Figure 10: Distribution of **pain** by Target Classes (Train)*

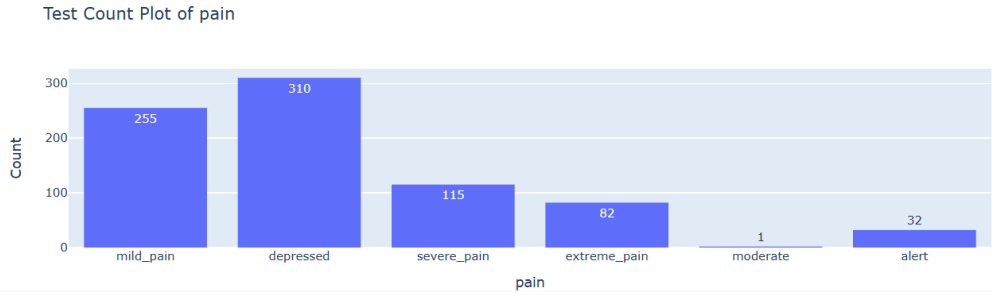


*Figure 11: Distribution of **pain** (Test)*

The **pain** feature presents a different situation, where both of the datasets contain an extreme class with a similar meaning, but the class names differ. This suggests that we should standardise the class label during preprocessing.

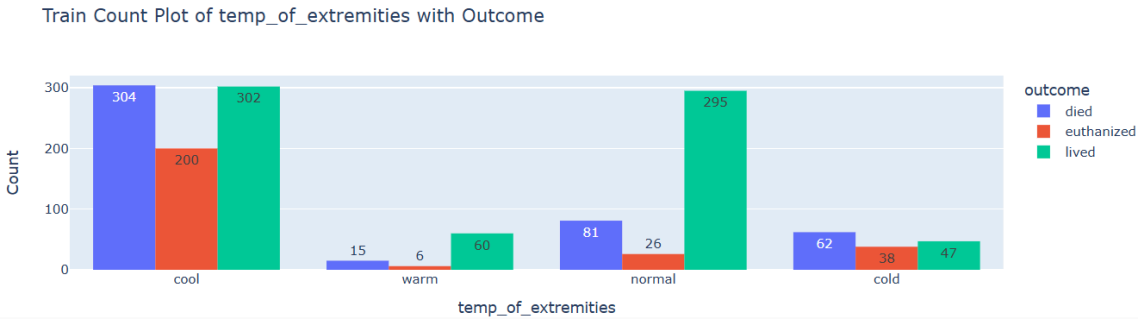


*Figure 12: Distribution of **temp_of_extremities** by Target Classes (Train)*

Temp_of_extremities is an ordinal categorical variable, so it would be more appropriate to manually encode it.

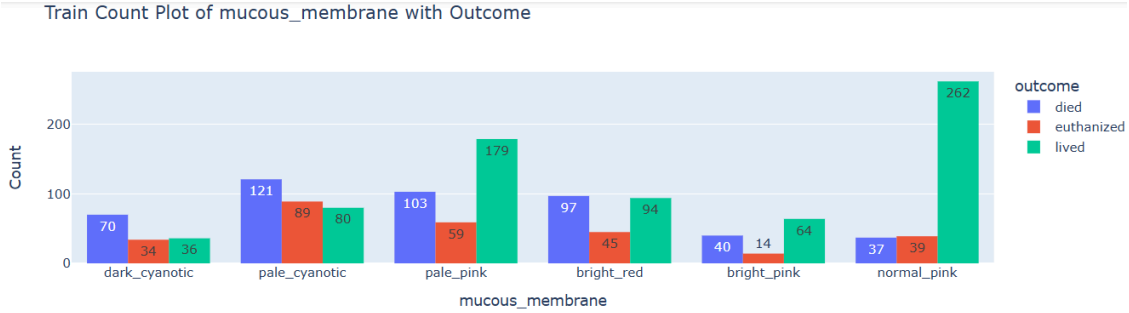


*Figure 13: Distribution of **mucous_membrane** by Target Classes (Train)*

For the **mucous_membrane** feature, there is no inherent order among the categories, which suggests that One-Hot Encoding should be used to encode this variable.

4. Challenges of the Problem

After preliminary data analysis, we observed that the target variable exhibits a slight class imbalance. Therefore, for model evaluation, we should prioritise the **F1 score** over the **accuracy score**, as the F1 score provides a better evaluation for imbalanced classes.

Another challenge we faced was the presence of missing values in some features. Hence, we performed manual imputation, filling missing categorical values with the mode and numerical values with the median.

5. Proposed Solution

5.1. Preprocessing

The train and test datasets were combined for preprocessing. Firstly, we handled the numerical variables. We drew box plots using Plotly to investigate the distribution of each feature and identify outliers.

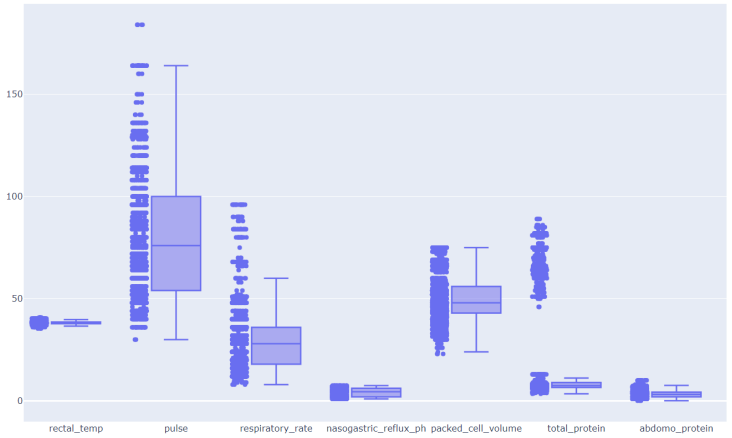


Figure 14: Box Plot of Numerical Variables in Combined Dataset

It appears that two of them have a significant number of outliers, respiratory_rate and total_protein. To investigate whether they will harm model performance, we plot the distributions of each feature by health outcome.

Figure 15: Box Plots of respiratory_rate and total_protein Grouped By Outcome (Train)

Judging by both plots and applying domain knowledge, we conclude that the outliers in each feature have realistic values despite being higher than normal (Liburt et al., 2016). Furthermore, in the case of total_protein, there are obvious differences in the distributions across different outcomes. Since they are likely to provide valuable information to machine learning models, we decided not to conduct any scaling methods or remove outliers. The final step for preprocessing the numerical variables was to impute null values with the median, as it is robust towards outliers.

The categorical variables required extensive preprocessing to make them compatible with machine learning models. The steps taken are summarised as follows:

- **Label encoding:** Ordinal categorical features, such as temp_of_extremities and age, were either passed through a Label Encoder or encoded manually to capture the inherent order among the classes.
- **One-hot encoding:** The remaining categorical features were passed through a One-Hot Encoder to convert them to integers, as machine learning models usually require numerical input rather than string or object types.
- **Normalising inconsistent columns:** The discrepancy in class names between the train and test datasets were standardised, preventing trained models from handling unexpected values when making predictions on the test set.

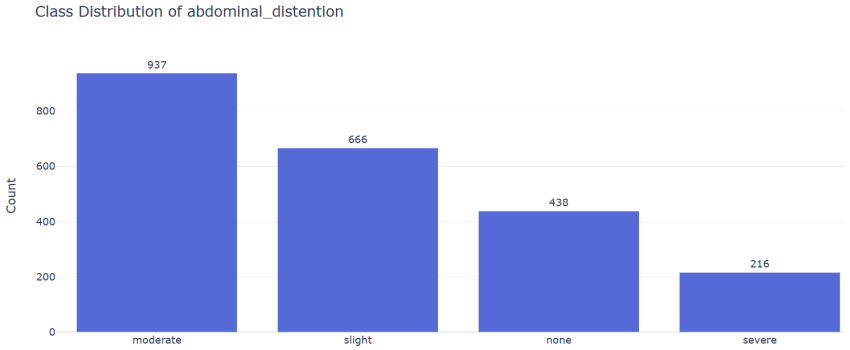


Figure 17: Class Distribution of abdominal_distention in Combined Dataset

Again, we applied domain knowledge while imputing missing values for selected categorical variables. For example, it makes more sense to impute missing values in abdominal_distention with 'none' rather than assuming the mode, 'moderate'. We then imputed the remaining categorical variables that contained missing values with the mode.

5.2. Feature Engineering

We continued using the combined dataset for feature engineering. Our goal was to reduce redundancies found in lesion_2 and lesion_3.

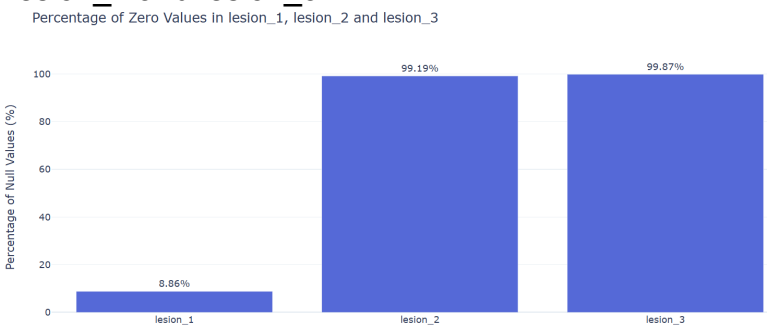


Figure 18: Percentage of zero values in lesion_1, lesion_2 and lesion_2 in Combined Dataset

We observed that lesion_2 and lesion_3 each contained 0's for over 90% of the data. Since these columns will likely not provide any meaningful input to machine learning models, we decided to remove them from the dataset. After splitting the data back into train and test sets, we are now ready for the model training phase.

5.3. Methodologies

5.3.1. Random Forest Classifier

Random Forest Classifier is an ensemble learning method that generates and combines multiple decision trees to classify a target variable. Each tree is constructed using different subsets of training data and a random selection of features. These processes diversify the trees and help

reduce overfitting. The final prediction is typically obtained through majority voting among the trees (GeekForGeeks, 2024).

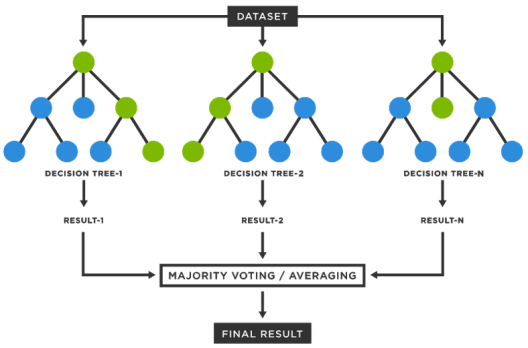


Figure 19: Random Forest Classifier (Spotfire, n.d.)

Key Parameters

n_estimators	The number of trees in the forest.	300
max_features	The number of features to consider when looking for the best split.	sqrt
min_samples_leaf	The minimum number of samples per leaf.	1
min_samples_split	The minimum number of samples required to split an internal node.	10
max_depth	The maximum depth of each tree.	100

5.3.2. Light Gradient-Boosting Machine (LightGBM)

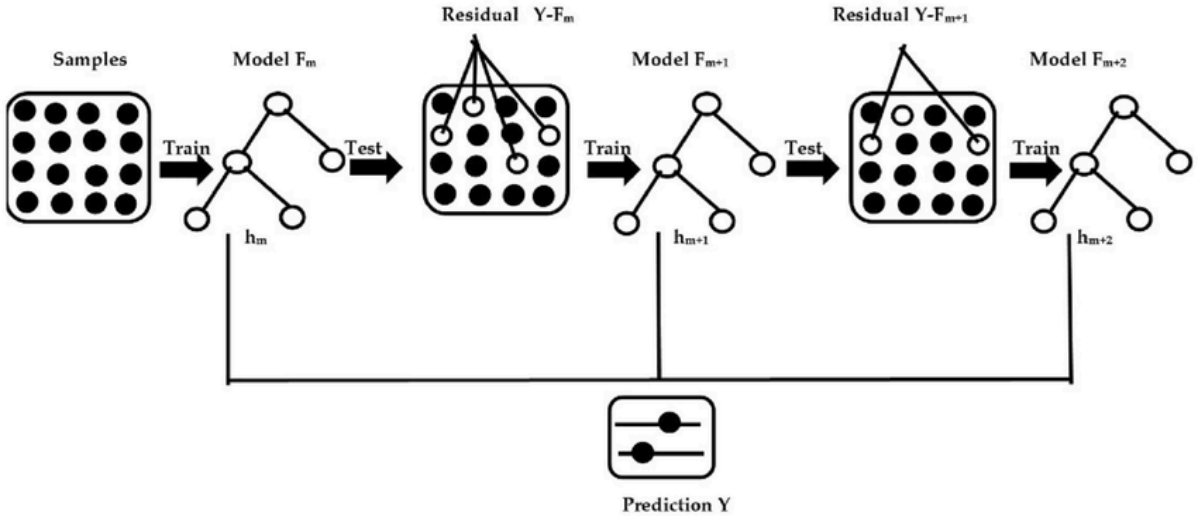


Figure 20: Overview of LightGBM's Approach (Hameed et al., 2022)

LightGBM is a highly efficient and fast implementation of gradient boosting for decision tree algorithms where it is designed for performance and speed in handling large datasets (Ke et al., 2017).

➔ **Handles Class Imbalance:** LightGBM can internally handle class imbalance via parameters like scale_pos_weight or through techniques like SMOTE as applied before. By training on the SMOTE-augmented data, your LightGBM classifier ensures that minority classes (died, euthanized) are considered more heavily during model training, improving the ability to classify them correctly.

➔ **Leaf-Wise Growth:** LightGBM grows trees leaf-wise instead of level-wise. This means that it will continue to grow deeper on the most promising branches (leaves) of the tree, where more improvements in prediction accuracy can be made.

```
[LightGBM] [Info] Training started with 200 estimators, learning_rate=0.05
[LightGBM] [Info] Boosting round 50: training accuracy: 72.50%
[LightGBM] [Info] Boosting round 100: training accuracy: 75.10%
[LightGBM] [Info] Boosting round 200: training accuracy: 77.60%
```

Figure 21: LightGBM Boosting Output

Key Parameters

max_depth	Controls the depth of each tree to avoid overfitting.	6
min_child_samples	Minimum number of data points a node should have before splitting.	10
n_estimators	The number of trees to build.	200
learning_rate	Controls how much to shrink the contribution of each tree.	0.05

➔ **Gradient-Based Learning:** LightGBM builds each tree by learning the residual errors from the previous trees. This means that each tree tries to correct the mistakes of the earlier ones.

- ◆ For each boosting round, our model tries to reduce the residual error from previous trees, optimising the performance of the model.

➔ **Regularisation:** LightGBM has built-in L1 (reg_alpha) and L2 (reg_lambda) regularisation to prevent overfitting by penalising the size of the model. These regularisation techniques help to ensure that our model generalises well to new data.

```
[LightGBM] [Info] Applying L1 regularization (alpha = 0.1)
[LightGBM] [Info] Applying L2 regularization (lambda = 0.1)
[LightGBM] [Info] Final training accuracy: 78.90%
```

Figure 22: LightGBM Regularisation Output

➔ **Prediction:** Once trained, our LightGBM model uses the ensemble of decision trees to make predictions on new, unseen data. It uses the cumulative votes from all the trees to predict the class.

```
[LightGBM] [Info] Test prediction accuracy: 72.80%
```

Figure 23: LightGBM Train-Test Prediction Accuracy Output

5.3.3. Histogram-Based Gradient-Boosting (HistBoost)

HistBoost is an optimised version of Gradient Boosting that enhances speed and efficiency. Instead of evaluating each feature value individually, it divides continuous features into discrete bins or histograms. This reduces the number of split points the algorithm needs to consider, enabling faster computation and lesser memory usage, which makes it suitable for large datasets.

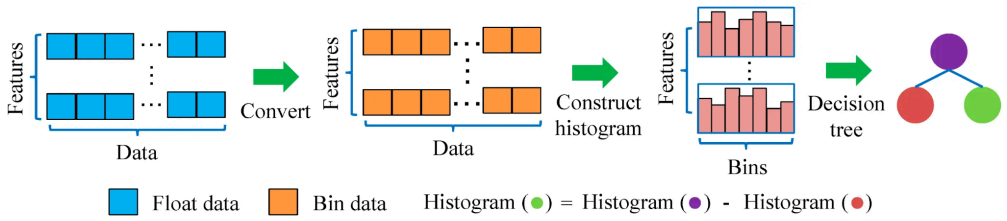


Figure 24: Histogram-Based Gradient Boosting (Liang et al., 2020)

Randomised search was conducted to determine the optimal set of parameters for the model. The best combination found is as follows:

Key Parameters

l2_regularization	The L2 regularisation parameter penalises leaves with small hessians.	0.5247
learning_rate	The learning rate, also known as shrinkage.	0.0705
max_bins	The maximum number of bins to use for non-missing values.	10
max_depth	The maximum depth of each tree.	5
max_iter	The maximum number of iterations of the boosting process, i.e. the maximum number of trees for binary classification.	100
max_leaf_nodes	The maximum number of leaves for each tree.	21
min_samples_leaf	The minimum number of samples per leaf.	2

5.3.4. Extreme Gradient-Boosting (XGBoost)

XGBoost is an optimised version of the Gradient Boosting framework. It offers superior performance and scalability due to its use of efficient algorithms such as regularisation and parallelized execution. This makes it ideal for large-scale data and competitions.

Key Parameters

n_estimators	The number of boosting rounds or trees.	500
max_depth	Controls the maximum depth of each decision tree, preventing overfitting.	6
learning_rate	The step size at each iteration while moving toward a minimum of the loss function.	0.1
gamma	The minimum loss reduction required for a split to be made.	0.2
colsample_bytree	Controls the fraction of features used to grow each tree.	0.8
subsample	The fraction of the training set used to fit each tree to control overfitting.	0.9
reg_lambda (L2 regularisation)	Prevents overfitting by penalising large coefficients.	1

5.3.5. Categorical Boosting (CatBoost)

CatBoost is a variant of Gradient Boosting that is designed to handle both categorical and numerical features. It does not require any feature encodings techniques like One-Hot Encoder or

Label Encoder to convert categorical features into numerical features. This significantly simplifies the data preparation process.

Key Parameters

cat_features	The list of indices of categorical features, allowing the model to handle them appropriately.	-
iterations	The maximum number of trees to build in the boosting process.	500
learning_rate	The step size at each iteration while moving toward a minimum of the loss function.	0.0936
bagging_temperature	Controls the strength of bagging, a technique where random subsets of data are used for training the model.	0.8116
l2_leaf_reg	The L2 regularisation coefficient to reduce overfitting by penalising large weights.	4.5594
random_strength	The strength of randomness when choosing splits.	7
border_count	The maximum number of borders used to split continuous features into discrete bins.	50
depth	The depth of the trees; deeper trees can model more complex relationships but may lead to overfitting.	5

5.3.6. Ensemble Learning

Ensemble methods are powerful techniques in machine learning that combine multiple learning algorithms to achieve improved predictive performance compared to using any single model. The idea is that by aggregating the predictions from diverse models, we can create a more robust and accurate predictor. Ensemble methods help to reduce variance and bias by leveraging the strengths of each model involved. There are several key techniques used for aggregation in ensemble learning:

- **Max Voting**: Each model in the ensemble makes a prediction, and the final output is the class label that receives the most votes, i.e. the model that the majority of algorithms agree upon becomes the final prediction.
- **Averaging**: The predictions from multiple models are averaged to make the final prediction.
- **Weighted Averaging**: This is a refined version of averaging, where different models are assigned different weights based on their performance or importance. The final prediction is made by taking a weighted average of the predictions from each model. This method allows for models that perform better to have a stronger influence on the final result.

We experimented with the **Weighted Average** aggregation technique. Specifically, we combined predictions from multiple gradient boosting models such as **XGBoost**, **LightGBM**, **CatBoost** and **HistBoost**. By assigning different weights to each model's output probabilities, we derived a final prediction. This allows more reliable models to contribute more to the final decision.

Experiments

Several combinations were experimented to determine the optimal combination. All models were first tuned using randomised search to optimise the hyperparameters based on accuracy scores obtained from cross-validation. For each fold of the training data, the out-of-fold predictions were stored, and the models were evaluated on the validation set using the F1 micro score as the performance metric. The OptunaWeights class from Optuna is then used to determine the optimal weights of each model.

Based on the weights obtained, the final prediction is a weighted average of the outputs from multiple models. The table below shows the weight distribution of each model and the corresponding accuracy:

Combination	Models	Weights	Accuracy
1	XGBoost	0.34035	0.78048
	LightGBM	0.22045	
	CatBoost	0.22459	
	HistBoost	0.21461	
2	XGBoost	0.28588	0.79268
	XGBoost (tuned)	0.17707	
	LightGBM	0.29040	
	CatBoost	0.24664	

Note: In combination 2, two XGBoost models with different hyperparameters were used.

The final ensemble predictions were obtained by calculating a weighted average of the base models' outputs, allowing for improved robustness and accuracy in classifying the test data.

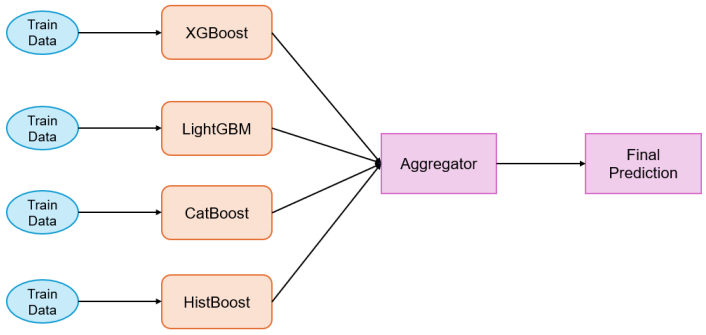


Figure 25: Weighted Average Ensemble Classifier for Combination 1

6. Experimental Study

6.1. Experimental Analysis of Chosen Solution

LightGBM yields one of the highest validation accuracy and micro-averaged F1-score, prompting us to further experiment on the hyperparameter tuning.

Parameters	n_estimators	[100, 200, 300]
	max_depth	[4, 6, 8, 10]
	learning_rate	[0.01, 0.05, 0.1]
	subsample	[0.6, 0.8, 1.0]
	colsample_bytree	[0.6, 0.8, 1.0]
	min_child_samples	[10, 20, 30]
	reg_alpha	[0, 0.1, 0.5]

	reg_lambda	[0, 0.1, 0.5]

6.1.1. Validation Loss vs Boosting Rounds

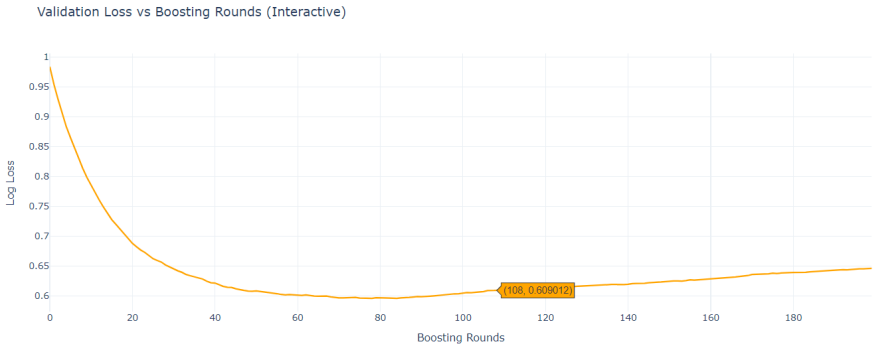


Figure 26: Validation Loss vs Boosting Rounds in LightGBM

The validation log loss decreases steeply during the first 20-40 boosting rounds, then flattens out and stabilises around 100 boosting rounds. After about 120 boosting rounds, the log loss starts to increase again, which suggests that **the model might be overfitting beyond this point**. The final log loss stabilises at around 0.6, but the model may have benefitted from early stopping to prevent the slight increase in loss beyond 100 rounds. Since the LightGBM starts overfitting, we might have to Introduce early stopping (e.g., after 100 rounds) to prevent overfitting and further improve model generalisation.

6.1.2. Validation Accuracy for Each Hyperparameter Combination

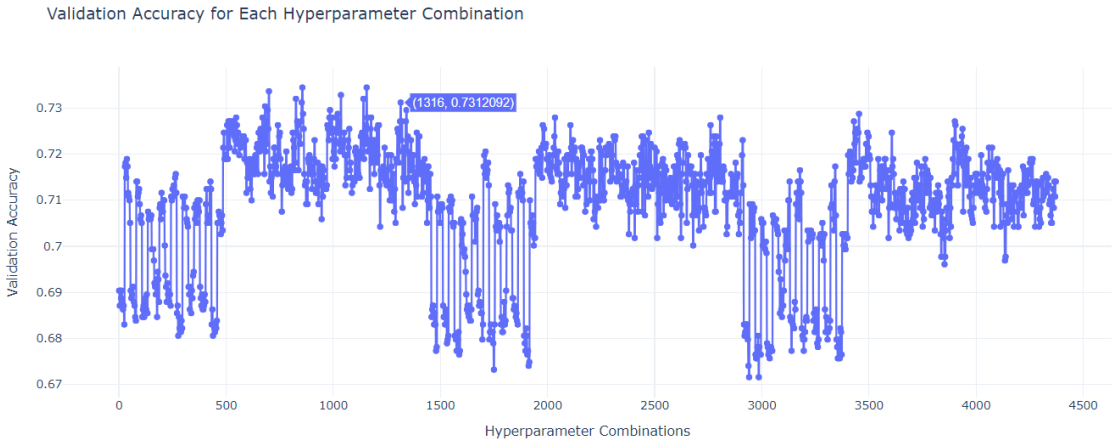


Figure 27: Validation Accuracy for Each Hyperparameter Combination

The plot shows the variation in validation accuracy across different hyperparameter combinations used. There are regions where the accuracy is clustered around 0.72-0.73, indicating that some hyperparameter combinations were more effective than others. There are noticeable dips in validation accuracy at certain hyperparameter combinations, which highlights the importance of proper tuning. In addition, the range of accuracies varies between approximately 0.67 and 0.73, showing that the model is quite sensitive to hyperparameters. Upon further running of the code, the optimal parameter that yield the highest accuracy:

Optimal Parameters	n_estimators	100
	max_depth	4
	learning_rate	0.1
	subsample	0.6
	colsample_bytree	0.6
	min_child_samples	10
	reg_alpha	0
	reg_lambda	0.1

6.1.3. Feature Importance

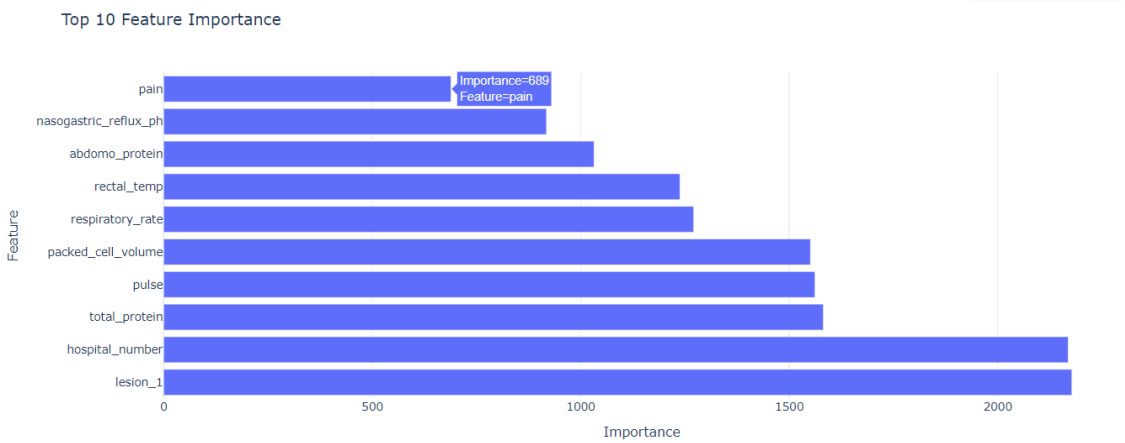


Figure 28: Feature Importance in LightGBM

Lesion_1 and hospital_number are the two most important features and these features provide crucial predictive power for the model. Pain has relatively lower importance compared to the top-ranked features, which might indicate that while it is useful, it is not as decisive as other clinical variables. This is useful for understanding which medical factors play the biggest role in predictions especially in LightGBM since we can potentially improve our predictive score.

6.2. Chosen Solution Novelty

To obtain the best results in terms of the micro-averaged F1-score, we have deployed several models discussed above. LightGBM, among the rest, was one of the most promising models. However, purely deploying the machine learning models was not enough without any preprocessing. Going beyond standard preprocessing by introducing domain-specific feature engineering was considered but upon desktop research, we have chosen certain techniques to improve our results.

1. Handling Imbalanced Classes Using Advanced Techniques

```
# Handle class imbalance using SMOTE
smote_tomek  = SMOTETomek(random_state=42)
X_train_smote_tomek, y_train_smote_tomek = smote_tomek .fit_resample(X_train, y_train)
```

```
grid_search = GridSearchCV(estimator=LGBMClassifier(), param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_rfe, X_train_smote_tomek)

# Best parameters from Grid Search
best_params = grid_search.best_params_

# Train the LightGBM model with optimal parameters
best_lgbm = LGBMClassifier(**best_params)
best_lgbm.fit(X_train_rfe, y_train_smote_tomek)
```

Figure 29: Snippets of SMOTE + Tomek Links in LightGBM Code Implementation

Instead of standard SMOTE for class imbalance, we have used SMOTE + Tomek Links - a combination of oversampling and cleaning the dataset to remove Tomek links, which are noisy samples. This reduces bias and improves class separability (Viadinugroho, 2021).

2. Dynamic Feature Selection

```
rfe_selector = RFE(estimator=LGBMClassifier(), n_features_to_select=20)
X_rfe = rfe_selector.fit_transform(X_opt, Y_opt)
```

Figure 30: Snippets of RFE in LightGBM Code Implementation

Instead of static feature selection, we implemented Recursive Feature Elimination (RFE) with LightGBM to dynamically choose the best features based on importance. As LightGBM provides feature importance metrics, we have to set a threshold for removing the least important features during training iterations and reevaluate the model after each iteration.

3. Early Stopping with Custom Thresholds

```
# Custom early stopping callback
def early_stopping_custom(booster, metric_name, threshold=0.01):
    if booster.best_iteration > 10 and abs(booster.best_score['valid_0']['multi_logloss']
                                           - booster.best_score['training']['multi_logloss']) < threshold:
        return True
    return False

best_lgbm = LGBMClassifier(**best_params)
best_lgbm.fit(X_train_rfe, y_train_smote_tomek, eval_set=[(X_val, Y_opt)], callbacks=[early_stopping_custom])
```

Figure 31: Snippets of Early Stopping in LightGBM Code Implementation

Instead of relying on a fixed number of rounds for early stopping, we use a custom threshold to decide when to stop training. For example, stop training when the difference between training and validation loss is minimal or when validation accuracy does not improve after a certain percentage of boosting rounds. This also helps us save time in training LightGBM.

6.3. Comparison with Other Models

Model	Validation Accuracy	Training Time inc. Tuning (mins)	Complexity
Random Forest Classifier	72.63%	5	Low
LightGBM	79.94%	150	Medium
Histogram-Based Gradient Boosting	74.63%	50	Medium
XGBoost	73.80%	90	High

CatBoost	74.23%	110	High

Comparison of LightGBM and Ensemble Learning

While the optimal ensemble learning model generally outperforms individual models such as XGBoost, CatBoost, and HistBoost by leveraging the strengths of multiple algorithms, in this specific experiment, LightGBM alone achieved a higher validation accuracy of 79.94%. The ensemble of models (XGBoost, LightGBM, CatBoost, and HistBoost), which were expected to provide better performance due to model diversity, underperformed relative to LightGBM in this case, achieving a validation accuracy of 75.20%.

Training Time: LightGBM alone is more efficient in terms of training time (150 minutes), while ensembles (180 minutes) tend to take longer because multiple models need to be trained and combined.

Why the Ensemble Fails to Outperform LightGBM:

- **Data Characteristics**: LightGBM might suit this dataset better as LightGBM is highly efficient at handling large datasets with many features. Our dataset consists of 28 features, which is a relatively high number given the dataset size. In contrast, models like XGBoost or CatBoost might not be optimised as well for this specific dataset.
- **Overfitting**: The dataset used in this competition is relatively small, with only 1,235 rows but many features (28 columns). In such cases, combining multiple models in an ensemble can increase the risk of overfitting. While ensemble learning is generally known for its ability to reduce overfitting, this is not always the case when the dataset is small. In this case, the ensemble model may have overfitted to noise or irrelevant patterns in the training data, resulting in poorer generalisation on unseen data. A single, well-tuned model like LightGBM can often avoid this issue and generalise better due to simpler modelling and fewer opportunities to fit noise.
- **Redundancy of Models**: If LightGBM already captures the most important data patterns, adding other models like XGBoost or HistBoost that perform similarly may not add significant new information. This can lead to a situation where the ensemble does not improve over a single well-tuned model, and may even dilute the accuracy by averaging predictions that are less well-suited.

This experiment demonstrates that a well-tuned single model can outperform a more complex ensemble in certain scenarios. While ensemble methods are often praised for their ability to combine the strengths of multiple models and reduce prediction errors, they may not always be the best solution. In this case, LightGBM was able to capture the relevant data patterns and generalise better than the ensemble learning model.

7. Conclusion

Methodology	Public Score
Random Forest Classifier	0.78048
LightGBM	0.84756
HistBoost	0.77439
XGBoost	0.78048
CatBoost	0.75609
Ensemble Learning	0.79268

In conclusion, we developed a robust predictive model for forecasting horse health outcomes by exploring various machine learning algorithms, including Random Forest, XGBoost, LightGBM, HistGradientBoosting, and CatBoost, each offering unique advantages. While models like XGBoost and LightGBM are better in handling complex and high-dimensional data, Random Forest provides valuable interpretability. To enhance accuracy and mitigate individual model limitations, we applied stacking, enabling a balanced combination of strengths across models and achieving improved predictive performance.

This project has been an insightful journey, deepening our understanding of model trade-offs between complexity, interpretability, and accuracy. The insights gained have laid a strong foundation for our future work in predictive modelling and machine learning applications.

8. References

GeeksforGeeks, (n.d.). Random Forest Classifier using Scikit-learn.
Random Forest Classifier using Scikit-learn - GeeksforGeeks

Hameed, A., Yap, W.-S., Morris, E., & Kasim, B. (2022). A practical intrusion detection system based on denoising autoencoder and LightGBM classifier with improved detection performance. Journal of Ambient Intelligence and Humanized Computing, 14(6), 7427–7452.
https://doi.org/10.1007/s12652-022-04449-w

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree.
https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf

Liang, W., Luo, S., Zhao, G., & Wu, H. (2020). Predicting Hard Rock Pillar Stability Using GBDT, XGBoost, and LightGBM Algorithms. Mathematics, 8(5), 765–765.
https://doi.org/10.3390/math8050765

Liburt, N., Malinowski, K., Williams, C. (2016). Measuring Temperature, Pulse & Respiration (TPR): What's Normal For My Horse? New Jersey Agricultural Experiment Station.
https://njaes.rutgers.edu/fs1262/

Predict Health Outcomes of Horses. (2024). Kaggle.
https://www.kaggle.com/competitions/playground-series-s3e22/overview

Spotfire, (n.d.). Demystifying the Random Forest Algorithm for Accurate Predictions.
Spotfire | Demystifying the Random Forest Algorithm for Accurate Predictions

Viadinugroho, R. A. A. (2021, April 18). Imbalanced Classification in Python: SMOTE-Tomek Links Method. Medium.
https://towardsdatascience.com/imbalanced-classification-in-python-smote-tomek-links-method-6e48dfe69bbc

Yasser, M. H. (n.d.). Horse Survival Dataset. Kaggle.
https://www.kaggle.com/datasets/yasserh/horse-survival-dataset