Project Report

On

Horse Survival Prediction using Various Models

Submitted by

Sagar Rathore, Aman Gaur, Satyam Sahu

Under the guidance of

Dr. Bharti
Department of Computer Science
University of Delhi

November 26, 2023

Contents

Abstract

In recent years, machine learning models have demonstrated remarkable capabilities in various domains, including healthcare. This project aims to conduct a thorough comparative study of five prominent machine learning models—Random Forest, XGBoost, LightGBM, Decision Tree, and CatBoost—in predicting the survival of horses based on their previous medical conditions. The equine health domain poses unique challenges, and accurate survival prediction can significantly impact veterinary care and decision-making.

1 Introduction

In the world of Data Mining, picking the right tool for predicting things can be tricky. This project dives into different models—like Decision Trees, Random Forests, XGBoost, LightGBM, and CatBoost—to create a smart system for guessing if a horse is healthy based on its medical history.

Why does this matter? Well, in animal healthcare, having a good way to predict health outcomes is super important. It helps vets make better decisions. The dataset have many features, and we want to see which one works best for predicting if a horse is in good shape or not.

This study isn't just about building a good system; it's also about checking how well each model actually works. We'll use new data to test them out and measure things like accuracy and F1 scores. These scores tell us how good the models are at making the right calls and not missing important stuff.

The models we've picked— **Decision Trees, Random Forests, XGBoost, LightGBM, and CatBoost** —are like different tools in a toolbox. Each has its strengths, and we want to find out which one is the best for predicting horse health based on their medical past.

So, as we go along, we're not just making a horse survival predictor. We're figuring out which tool works best and why. And who knows, what we learn here might even help out in other areas beyond just horses!

2 Problem Statement

Develop a **predictive model** that accurately assesses the survival probability of horses based on their historical medical records. The model should effectively utilize the available data to identify **patterns and relationships between medical conditions and survival outcomes**. This model can be instrumental in improving equine healthcare and aiding veterinarians in making informed treatment decisions.

3 Methodology

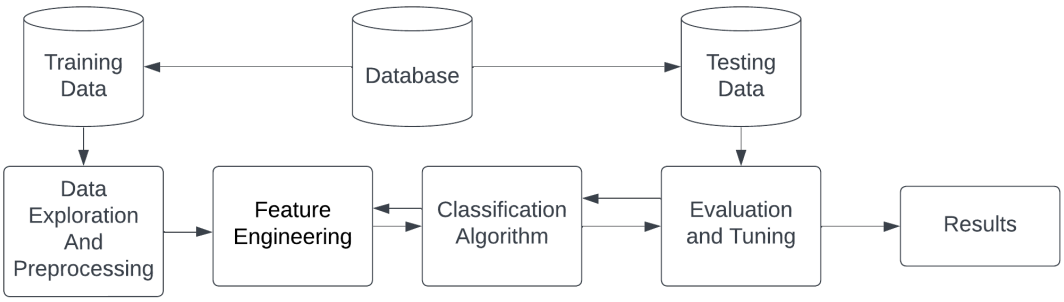


Figure 1: Flow Diagram

The methodology for depends on following phases for prediction of survival of horse.

1. **Data Preprocessing**: Getting the dataset, and cleaning the dataset to address missing values, inconsistencies, and outliers. Employ techniques such as imputation, data transformation, and error correction, if needed. And to perform Data Exploration to analyze to understand the distribution of variables, and identify potential patterns.

2. **Feature Engineering**: The process of selecting and transforming features. This includes features scaling normalization and encoding categorical attributes.

3. **Model Selection And Training**: Evaluate the performance of various machine learning models. Utilize a cross-validation approach to ensure unbiased evaluation.

4. **Model Evaluation**: Performance Metrics: Evaluate the final model using metrics such as precision, recall, and **F1-score** on an unseen test dataset. Analyze the errors made by the model to identify potential biases or limitations

3.1 Data Preprocessing

Data preparation is the crucial step of cleaning, transforming, and enriching raw data before it can be used for analysis or modeling. It involves identifying and correcting errors, inconsistencies, and missing values, as well as standardizing data formats and structures.

3.1.1 Dataset

The dataset "Horse Colic" was picked up from the UC Irvine Machine Learning Repository. The dataset is divided into 27 main features and about 300 data objects. the dataset is rich in missing values(about 30 percent) with discrete, nominal, and continuous data types. Since this was a competition dataset hosted on Kaggle,

around **1000 new instances** were created using **deep learning** in the train dataset and test dataset.

Table 1: Feature and Description for the Horse Health Dataset

Feature	**Description**
Surgery	Whether the horse had surgery or not
Age	Age category of the horse
Hospital Number	Numeric ID assigned to the horse
Rectal Temperature	Normal 37.8
Pulse	Reflects heart condition
Respiratory Rate	Subjective indication of peripheral circulation.
Peripheral Pulse	Subjective measurement
Mucous Membranes	Subjective measurement of color.
Capillary Refill Time	Clinical judgement.
Pain	Subjective judgement of the horse's pain level.
Peristalsis	indication of activity in the horse's gut.
Abdominal Distension	Important parameter
Nasogastric Tube	Refers to any gas coming out of the tube.
Nasogastric Reflux	Greater amount indicates possible obstruction
Nasogastric Reflux pH	Normal values are in the 3 to 4 range
Rectal Examination - Feces	Absent feces probably indicates an obstruction
Abdomen	Possible values
Packed Cell Volume	.The number of red cells by volume in the blood.
Total Protein	Higher value indicates greater dehydration
Abdominocentesis Appearance	Normal fluid,cloudy or serosanguinous
Abdominocentesis Total Protein	The higher the level, the more likely it is to have a compromised gut.
Outcome	Lived,Died, Was euthanized
Surgical Lesion	was the problem (lesion) surgical?
Type of Lesion	Four-part code indicating site, type, subtype, and specific code and 3 Lesions Lesion1 Lesion 2 Lesion 3
CP Data	Is pathology data present for this case?

3.1.2 Data Summary

	dtypes	count	#unique	#missing	%missing
surgery	object	1235	2	0	0.000000
age	object	1235	2	0	0.000000
hospital_number	int64	1235	255	0	0.000000
rectal_temp	float64	1235	43	0	0.000000
pulse	float64	1235	50	0	0.000000
respiratory_rate	float64	1235	37	0	0.000000
temp_of_extremities	object	1196	4	39	3.157895
peripheral_pulse	object	1175	4	60	4.858300
mucous_membrane	object	1214	6	21	1.700405
capillary_refill_time	object	1229	3	6	0.485830
pain	object	1191	6	44	3.562753
peristalsis	object	1215	5	20	1.619433
abdominal_distention	object	1212	4	23	1.862348
nasogastric_tube	object	1155	3	80	6.477733
nasogastric_reflux	object	1214	4	21	1.700405
nasogastric_reflux_ph	float64	1235	26	0	0.000000
rectal_exam_feces	object	1045	5	190	15.384615
abdomen	object	1022	5	213	17.246964
packed_cell_volume	float64	1235	49	0	0.000000
total_protein	float64	1235	83	0	0.000000
abdomo_appearance	object	1187	3	48	3.886640
abdomo_protein	float64	1235	54	0	0.000000
surgical_lesion	object	1235	2	0	0.000000
lesion_1	int64	1235	57	0	0.000000
lesion_2	int64	1235	4	0	0.000000
lesion_3	int64	1235	2	0	0.000000
cp_data	object	1235	2	0	0.000000
outcome	object	1235	3	0	0.000000

(a) Train Data

	dtypes	count	#unique	#missing	%missing
surgery	object	824	2	0	0.000000
age	object	824	2	0	0.000000
hospital_number	int64	824	210	0	0.000000
rectal_temp	float64	824	34	0	0.000000
pulse	float64	824	49	0	0.000000
respiratory_rate	float64	824	38	0	0.000000
temp_of_extremities	object	789	4	35	4.247573
peripheral_pulse	object	777	4	47	5.703883
mucous_membrane	object	811	6	13	1.577670
capillary_refill_time	object	818	3	6	0.728155
pain	object	795	6	29	3.519417
peristalsis	object	805	4	19	2.305825
abdominal_distention	object	802	4	22	2.669903
nasogastric_tube	object	760	3	64	7.766990
nasogastric_reflux	object	810	3	14	1.699029
nasogastric_reflux_ph	float64	824	29	0	0.000000
rectal_exam_feces	object	699	4	125	15.169903
abdomen	object	670	5	154	18.689320
packed_cell_volume	float64	824	48	0	0.000000
total_protein	float64	824	72	0	0.000000
abdomo_appearance	object	793	3	31	3.762136
abdomo_protein	float64	824	50	0	0.000000
surgical_lesion	object	824	2	0	0.000000
lesion_1	int64	824	54	0	0.000000
lesion_2	int64	824	4	0	0.000000
lesion_3	int64	824	1	0	0.000000
cp_data	object	824	2	0	0.000000

(b) Test Data

Figure 2: Train and Test Data Summary

In the figure 2, we have 4 columns representing datatype, count, unique values, and missing values. The darker the columns, the more the value.

We can observe that there are lots of missing values in our dataset around 28 columns and Shape of Train Data: (1235, 28) Shape of Test Data: (824, 27).

There are around **11 numerical features** and **16 categorical features**. From observing it closely we can say that all missing values are in categorical Features. We need to predict the outcome variable. It is classified into 3 parts: Lived, Euthanized, and Dead.

Outcome distribution

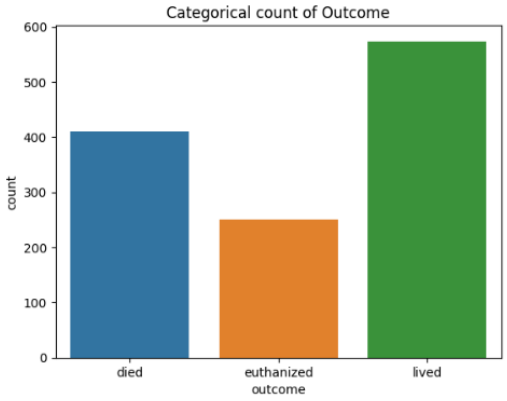


Figure 3: Outcome Distribution

We can observe that there is also a huge class imbalance which can lead to high bias and poor generalization.

3.1.3 Exploratory data analysis

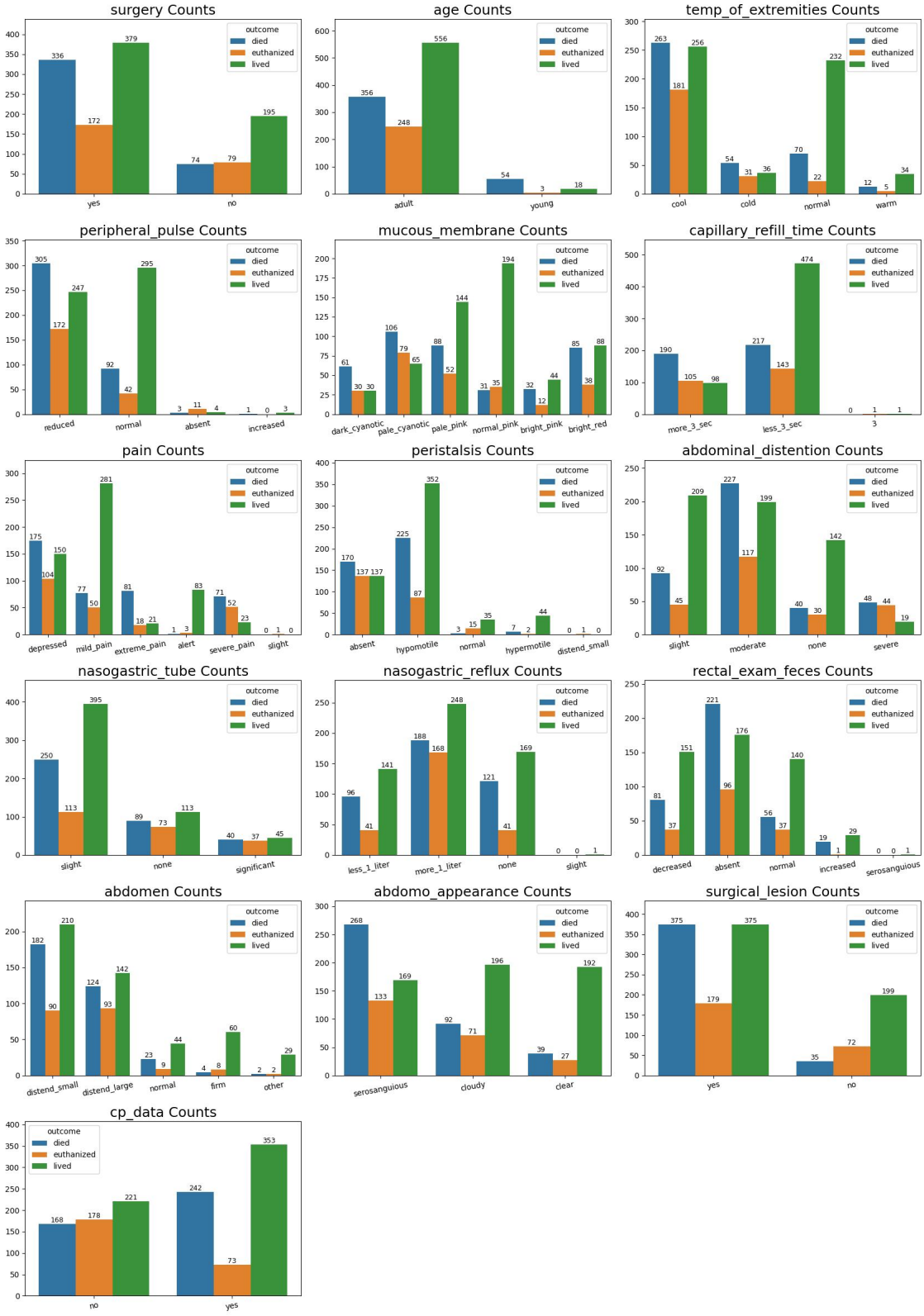


Figure 4: Count Plot Train

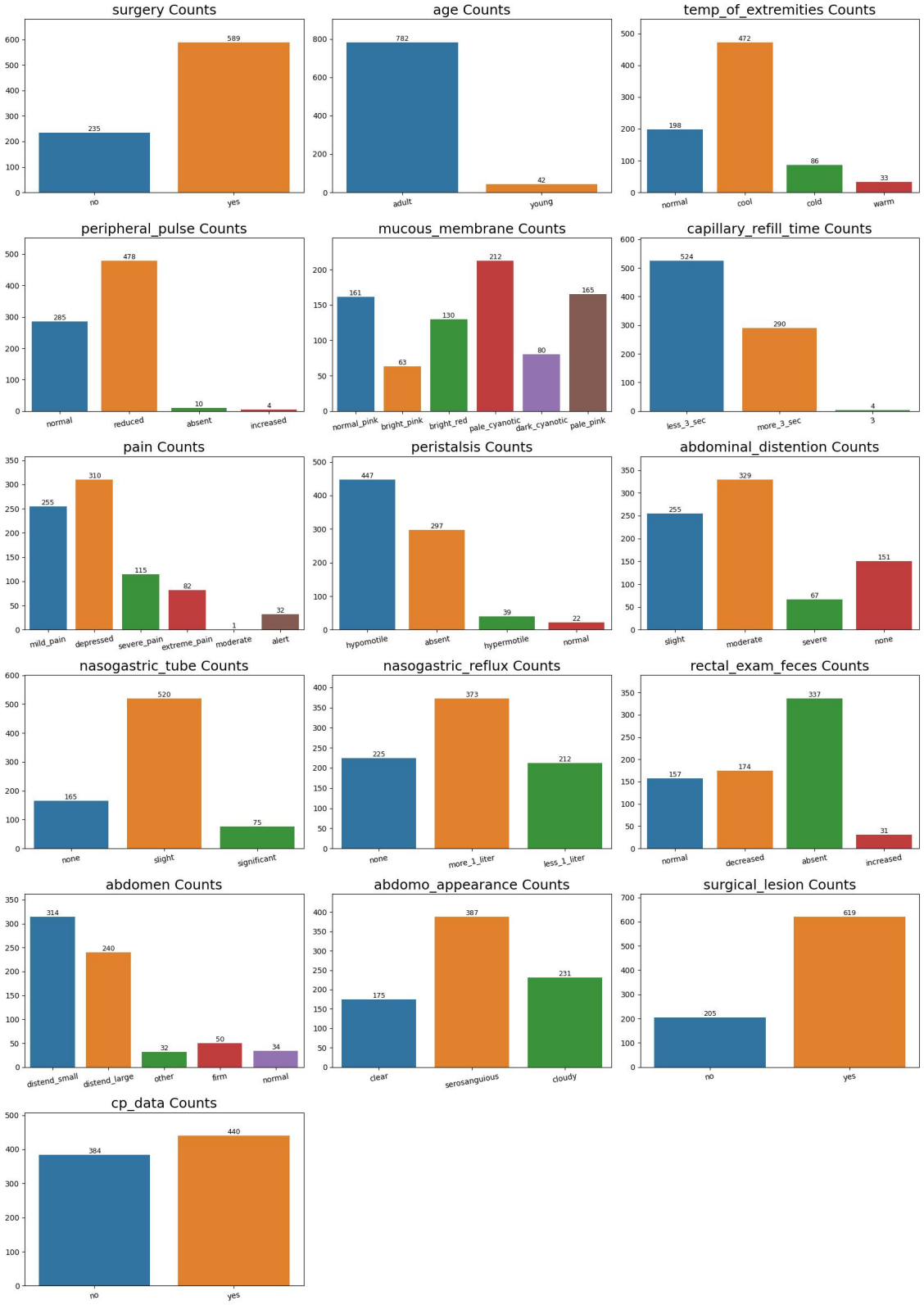


Figure 5: Count Plot Test Data

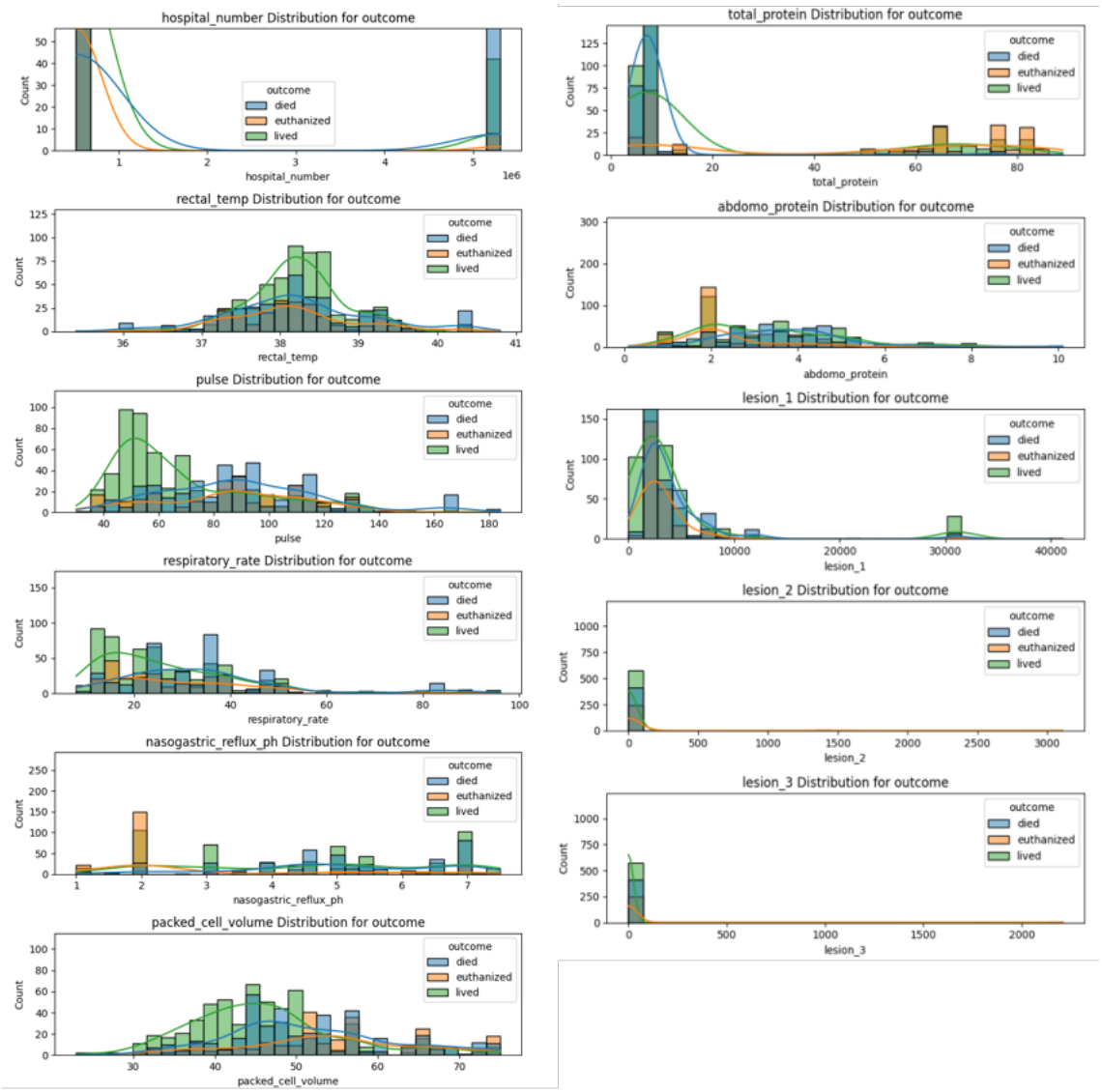


Figure 6: Numerical Feature Hist Plot

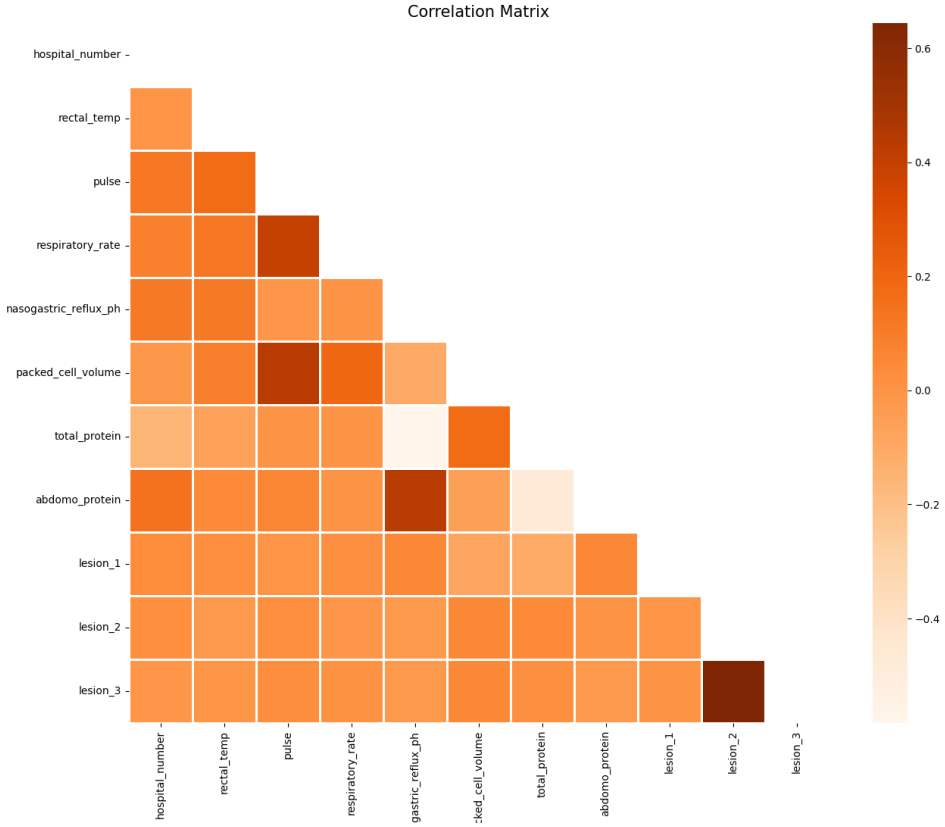
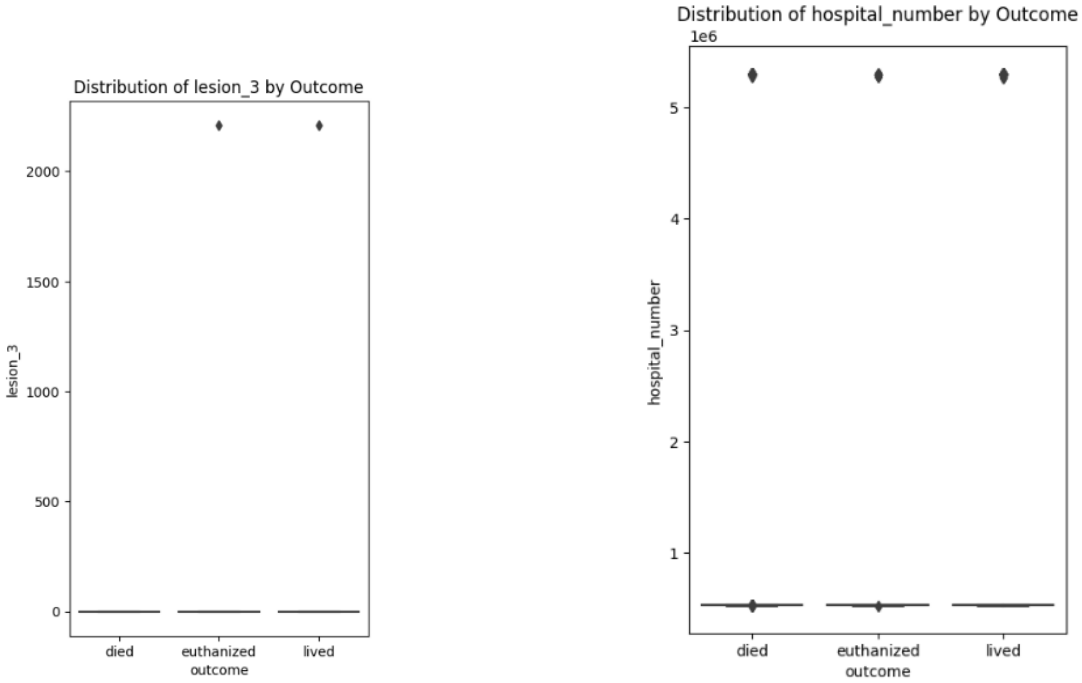


Figure 7: Numerical Feature Correlation Matrix



(a) lesion 3

(b) Hospital Number

Figure 8: Box Plot

We can make these inferences from the data analysis.

- Total protein appears to be very highly related to the target.
- There is no obvious linear relationship between numerical features.
- The age Counts column contains a large number of adults.
- When "cpdata Counts is Yes, the target variable is euthanized with a small probability.
- Lesion 3 has very less instances.
- Hospital Id is case id which can be dropped.

Further data analysis be revealed in Feature Engineering.

3.1.4 Data Preprocessing

Data preparation is the crucial step of cleaning, transforming, and enriching raw data before it can be used for analysis or modeling. It involves identifying and correcting errors, inconsistencies, and missing values, as well as standardizing data formats and structures. In order to train our machine learning model effectively, we need to transform the categorical features in our dataset into numerical data, this process is called encoding.

The are many types of Encoding here are some:

- One hot Encoding : with one-hot, we convert each categorical value into a new categorical column and assign a binary value of 1 or 0 to those columns
- Ordinal Encoding. We assign an index to a categorical value in the categorical column.

We found out that for our dataset ordinal encoding gave better results than one hot encoding during model evaluation.

Missing Values: To address the missing values in categorical features, we filled them with their mode values.

3.2 Feature Engineering

Feature engineering is a crucial step in the machine learning pipeline where you create new features or modify existing ones to improve the performance of your model. Effective feature engineering can lead to better model interpretability, generalization, and overall predictive performance . Here are some common techniques and ideas for feature engineering:

To enhance the model's performance, we made several adjustments to the dataset:

- **Feature Modification:** Due to only one instance of "slight" pain in the training dataset and "moderate" pain in the test dataset, we merged them into one. Similarly, there are many instances like this in datasets like peristalsis, rectal_exam_feces, and nasogastric_reflux.
- **Dropped Feature:** Based on boxplot analysis (see Figure 8), we identified that the features "hospital_number" and "lesion 3" did not effectively differentiate between classes. Consequently, we removed these features from the training dataset.

3.2.1 Feature Selection

Feature selection is the process of choosing a subset of relevant features from the original set of features to improve model performance, reduce computational cost, and enhance interpretability. Here are some common feature selection methods:

Filter Methods:

Variance Thresholding: Remove features with low variance as they may not provide much information. Correlation-based Selection: Remove features that are highly correlated with each other. Wrapper Methods:

Recursive Feature Elimination (RFE): Recursively remove features and build a model until the optimal subset of features is selected. Forward Selection: Iteratively add features to the model based on performance. Backward Elimination: Iteratively remove features from the model based on performance. Embedded Methods:

LASSO (L1 Regularization): Penalize the absolute values of the coefficients, encouraging sparsity and leading to automatic feature selection.

Lasso (L1 Regularization

The L1 regularization method is a technique used in machine learning to prevent overfitting and improve the generalization of a model. It does this by adding a regularization term to the objective function that the model is trying to minimize. The regularization term penalizes large values of the model's parameters, which can help to reduce the complexity of the model and prevent it from overfitting to the training data.

In terms of feature selection, L1 regularization can be useful because it can help to identify the most important features in the data. When L1 regularization is applied, the model will automatically assign small values to the weights of less important features, effectively "zeroing out" these features and removing them from the model. This can help to reduce the dimensionality of the data and improve the model's performance.

L1 Regularization technique wasn't very helpful as it showed every feature is important for predicting outcome variable.

```
total features: 29
selected features: 29
features with coefficients shrank to zero: 5
```

Figure 9: Lasso Regularization Result

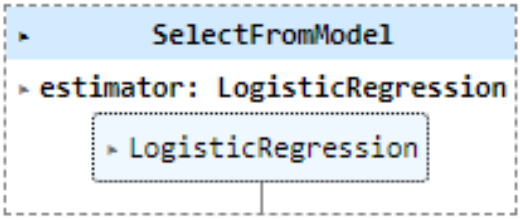


Figure 10: Estimator

Backward Elimination Backward Elimination is a feature selection technique that involves starting with all the features in your model and iteratively removing the least significant ones until a stopping criterion is met. Here's a step-by-step guide on how to perform Backward Elimination:

1. Fit the Model with All Features: Train your model using all the available features.
2. Evaluate Feature Significance: Assess the significance of each feature's contribution to the model. Common methods like feature importance scores in tree-based models.
3. Identify the Least Significant Feature:
4. Remove the least significant feature
5. Re-fit the Model:
6. Train your model again, but this time with the reduced set of features. Repeat Steps 2-5:
7. Stop Criterion:Decide on a stopping criterion, such as reaching a certain number of features or until all remaining features are statistically significant.

We applied 2 methods Backward Elimination and Lasso (L1 Regularization) and found that only **lesion2, hospital_number, lesion3** can be removed from the dataset as it has less feature importance in all models. Thus this also proved our data analysis.

3.3 Model Selection And Training

In the process of model selection and training, the first crucial step is to clearly define the problem at hand, whether it be classification, regression, clustering, or another type of task. Following this, a comprehensive understanding of the dataset is essential, involving analysis of feature types, distributions, and relationships, as well as addressing any missing values or outliers. The dataset is then split into training, validation, and test sets. Choosing an appropriate algorithm comes next, ranging from linear models and tree-based methods to support vector machines and neural networks. Hyperparameters are fine-tuned through techniques like grid search or random search. The model is trained on the training set and evaluated on the validation set, iterating through hyperparameter adjustments if necessary. Once satisfied, the final model is trained on both the training and validation sets and evaluated on the test set to ensure robust generalization to unseen data.

Our problem is classifcation problem and we have tabular data, for these types of problem tree based models perfoms better than neural nets and classifier like logistic ,SVM.

We have used 5 tree-based models for this project.

1. Decision Tree Classifier

2. Random Forest Classifier
3. XGboost Classifier
4. Catboost Classifier
5. LGBM Classifier

Before we get into evaluation lets understand each model first.

Decision Tree Classifier Decision trees are used to solve classification problems and categorize objects depending on their learning features. They can also be used for regression problems or as a method to predict continuous outcomes from unforeseen data. The decision tree algorithm belongs to the supervised learning algorithm family.

3.3.1 Ensemble Learning

Ensemble learning is a supervised learning technique that combines multiple models to build a more powerful and robust model and then to combine their predictions for the classification of unseen instances using some form of voting (such as majority voting. . .).

There are mainly two types of ensemble classifiers.

- Bagging (Bootstrap Aggregation) - Bagging is a type of ensemble learning in which multiple base(weak) models are trained independently in parallel on different subsets of the training data. subset is generated using bootstrap sampling (Randomly 'n' subsets of original training data are sampled with replacement.). In bagging classifier final prediction using majority voting. In bagging regression final prediction is made by averaging.

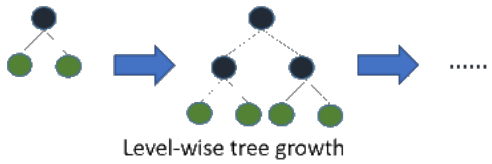
- Boosting - Boosting is a machine learning technique that combines multiple weak learners into a single strong learner.Boosting works by sequentially improving the performance of individual weak learners by assigning more weight to the data points that are misclassified or have high error.

Random Forest Classifier A random forest model is a extension of bagging technique that combines multiple decision trees to create a more accurate and robust prediction. It works by selecting random subsets of the data and features, also known as feature bagging, generates a random subset of features, which ensures low correlation among decision trees, and building a decision tree for each subset. Then, it aggregates the predictions of all the trees to produce the final output

Extreme Gradient Boosting

In XGBoost decision trees are created in sequential form. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. Each decision tree is trained on a random subset of the data features and tries to minimize the loss function. These individual classifiers/predictors then ensemble to give a strong and

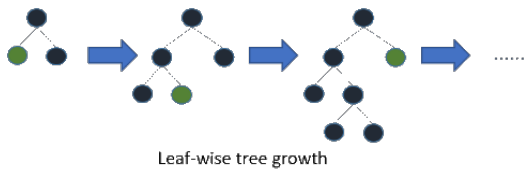
more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.



Used Parameters

- 'objective': 'multi:softmax': This parameter specifies the learning task and the objective function to be used. In this case, it's set to 'multi:softmax', indicating that the model is configured for multi-class classification.
- 'num_class': 3: This parameter defines the number of classes in the classification task.
- 'max_depth': 10:Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit
- 'learning_rate': 0.1: This parameter determines the step size at each iteration while moving towards a minimum of the loss function. It scales the contribution of each tree. A lower learning rate typically requires more trees in the ensemble but can lead to better generalization.

Light Gradient Boosting Model LGBM model is a gradient boosting technique that uses decision trees as weak learners and improves their performance by adjusting their weights based on the error of the previous learner. LGBM model has several features that make it faster and more accurate than other gradient boosting algorithms, such as histogram-based algorithm, leaf-wise growth, exclusive feature bundling, and parallel and distributed learning.



Used Parameters

- 'n_estimators':number of boosting iterations
- 'max_depth': limit the max depth for tree model.
- 'learning_rate': shrinkage rate, Lower learning rates generally lead to more robust models by making the model learning more conservative.
- 'subsample': Specifies the fraction of samples (observations) to be used for training each individual tree. Setting it to 0.20 means only 20

- 'colsample_bytree': Determines the fraction of features (columns) to be randomly selected for each tree. This value of 0.56 means that 56

- 'reg_alpha': This parameter represents regularization terms that penalize the complexity of the model.

CatBoost Model CatBoost is a powerful gradient-boosting technique, it uses a number of techniques to improve the accuracy and efficiency of gradient boosting, including feature engineering, decision tree optimization and a novel algorithm called ordered boosting.One of the main advantages of CatBoost is that it can handle categorical features without any feature encoding.

Used Parameters

- loss_function = 'MultiClass': Specifies the loss function used for training. In this case, it's set for multi-class classification.

- iterations : Determines the number of boosting iterations or trees to build during training.

- learning_rate : Sets the step size or rate at which the model learns from the data.

- depth : Specifies the maximum depth of each individual tree in the ensemble.

- random_strength : Controls the magnitude of randomness for feature permutations during tree construction

Aspect	Learning Approach	Handling of Data and Features	Overfitting and Regularization	Computational Efficiency
Decision Trees	Builds a single tree with minimal impurity.	Not adept at handling missing data or categorical features without preprocessing.	Prone to overfitting, especially with deeper or more complex trees.	Quick to build but may become computationally expensive with large datasets or complex structures.
Random Forest	Ensemble of trees with bagging.	Manages missing data by averaging across trees. Doesn't handle categorical features inherently.	Reduces overfitting compared to a single decision tree.	Slower due to building multiple trees but provides better generalization.
XGBoost	Sequentially builds trees correcting errors.	Handles missing values internally, supports categorical features, and is robust with imbalanced data.	Implements regularization parameters.	Time-consuming, especially with large datasets, but offers high accuracy.
LightGBM	Gradient boosting with leaf-wise growth.	Handles missing values, supports categorical features, and uses exclusive feature bundling.	Reduces overfitting with leaf-wise growth, potentially requiring fewer splits.	Faster due to leaf-wise growth, reducing computational cost, more efficient with large datasets.
CatBoost	gradient boosting with a specialized handling of categorical features.	Efficiently handles categorical features without the need for one-hot encoding or label encoding.	Has algorithm to prevent overfitting, including built-in support for depth-wise tree growth.	Efficiently handles large datasets due to its handling of categorical features and optimization techniques.

Table 2: Comparision table between models

Splitting Data

- Train-Test Split:

  Description: The dataset is divided into two subsets: a training set used to train the model and a test set used to evaluate its performance. Use Case: Commonly used when there is a sufficient amount of data. A typical split might be 80% for training and 20% for testing.

- K-Fold Cross-Validation: Description: The dataset is divided into k subsets (folds). The model is trained k times, each time using k-1 folds for training and one fold for testing. Performance metrics are averaged over the k iterations. Use Case: Effective when dealing with a limited amount of data to ensure thorough evaluation.

For splitting the data into train and validation sets we initially used train-test split. Due to small size of dataset, it performed badly. For example, for 1 instance it was giving **0.75 f1 score** for lgbm and for 2 instances it was giving **0.69 f1 score** in LGBM classifier.

Then we switched to K-Fold Cross Validation, setting k=5. here the average f1 score was **0.73** in lgbm

3.3.2 Feature Importance's of Xgboost, LGBM and Catboost

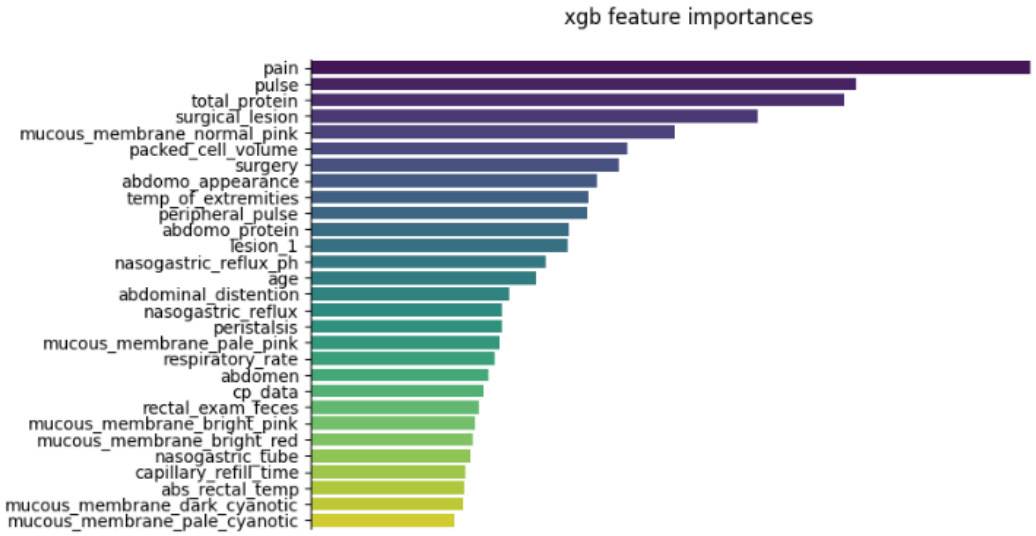


Figure 11: XGB Feature Importance

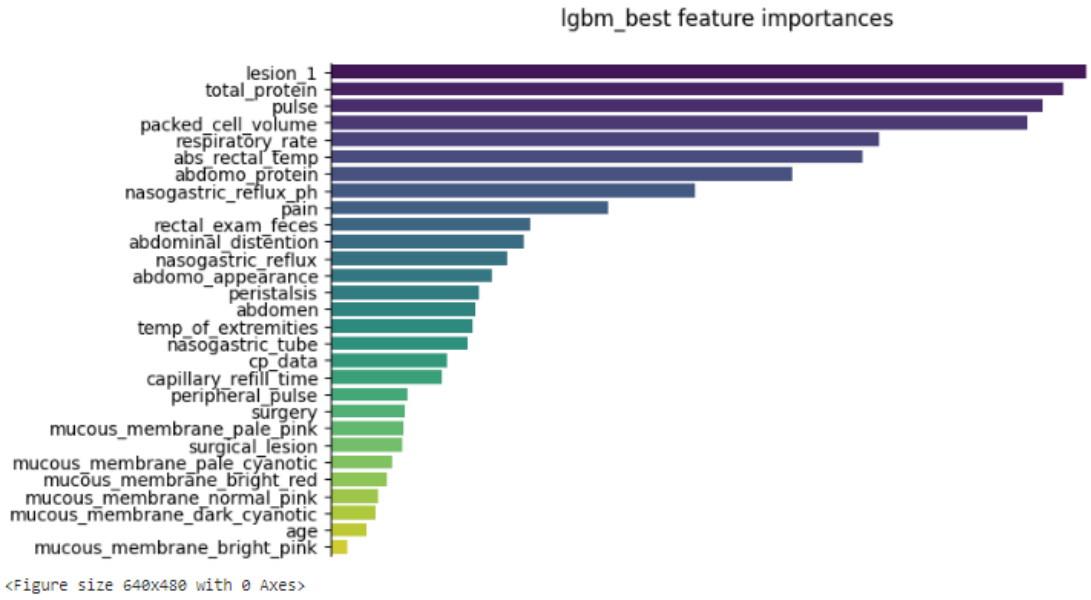


Figure 12: LGB Feature Importance

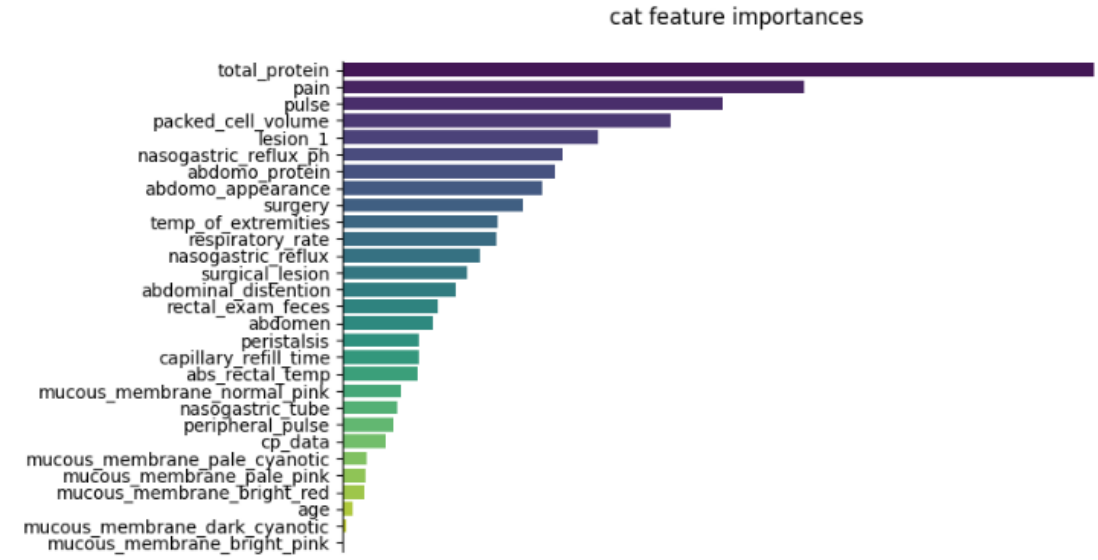


Figure 13: CAT Feature Importance

3.4 Model Evaluation

Model evaluation ensures the effectiveness and generalisability of the developed models. It involves assessing the performance of a model using various metrics and techniques to determine its ability to make accurate predictions on unseen data.

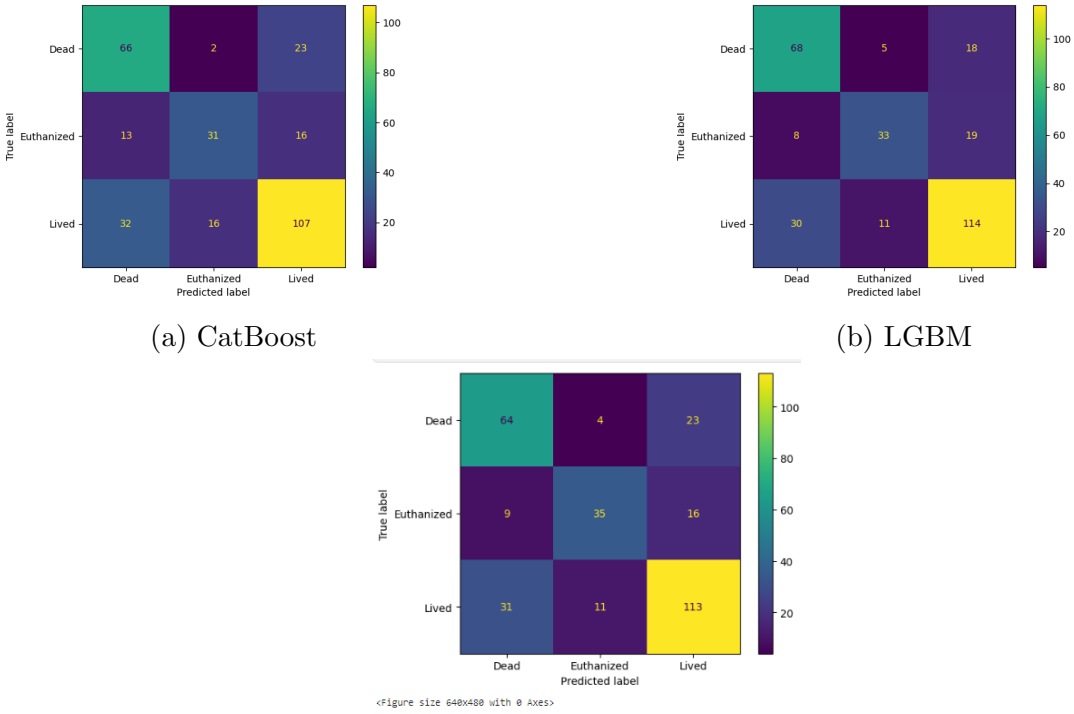
For this dataset, we decided to use the Micro Averaged F1 Score metric for Evaluation as our dataset is multi-classified and high imbalanced.

F1 Score The micro-averaged F1 score is a metric used to evaluate the performance of a multi-class classification model. It is particularly useful when dealing

with imbalanced datasets where some classes may have significantly fewer instances than others. Here's an explanation of the micro-averaged F1 score:

- F1 Score: The F1 score is the harmonic mean of precision and recall. It is a single metric that balances both false positives and false negatives.
- Micro-Averaging: Micro-averaging is a technique where you aggregate the individual true positives, false positives, and false negatives across all classes and then calculate the F1 score. This approach gives equal weight to each instance, regardless of its class.
- F1 Score: The F1 score is the harmonic mean of precision and recall. It is a single metric that balances both false positives and false negatives.
- Calculation:

$$F1_{\text{micro}} = \frac{2 \cdot \text{TP}_{\text{total}}}{2 \cdot \text{TP}_{\text{total}} + \text{FP}_{\text{total}} + \text{FN}_{\text{total}}}$$



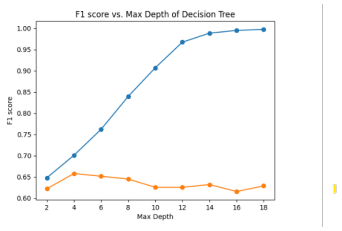
(a) CatBoost

(b) LGBM

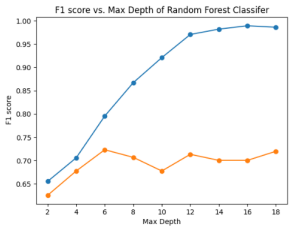
(c) XGB

Figure 14: Confusion Matrix

Each Model is evaluated with F1 score with the training dataset and to measure the optimal depth of the decision tree in the model we plot a graph to find a balance where F1 score is maximized without overfitting. Given are some of the graphs we plotted for the same reason. Blue line indicating train f1 score and orange line test score.



(a) Decision Tree



(b) Random Forest

Figure 15: F1 Score vs Increasing Depth of DT and Random Forest

4 Result

F1 Scores of all the models are assessed across 5-fold cross-validation with parameter tuning.

Model	F1 Score
Decision Tree	0.649
Random Forest	0.694
Light GBM	0.714
Cat boost	0.732
XGB	0.711

Table 3: average F1 Score 5 fold cross-validation

We can observe that Light GBM performed slightly better than XGBOOST and Catboost, while Catboost and Xgboost performed almost similarly, and while decision tree performed worse. We initially thought Catboost would give the best score because there were lots of categorical features and it uses target statistic ordered boosting but Lightgbm performed better. In all, we can say that gradient-boosted algorithms perform better than bagging and decision tree in this type of problem. Furthermore, the hyper-parameter tuning played a very important role here rather than feature selection. As tuning these parameters sufficiently increased f1 score.

In the competition on unseen test data, lgbm performed best giving 0.74696 F1 score in private score

5 Conclusion

In conclusion, in competitions with small datasets like this one, it seems more appropriate to find and apply precisely needed methods rather than engaging in extensive and complex data preprocessing and feature engineering to boost the CV Score.

In such competitions where collecting more data is not a feasible option, designing ensemble techniques to be more effective is necessary

6 Further Works

The current study demonstrates the potential of Optuna for hyperparameter optimization Optuna is an open-source hyperparameter optimization framework for machine learning models. It's designed to make it easy for researchers and practitioners to find the best hyperparameters for their models in an efficient manner.

Also a more diverse external dataset which explores horses genetic factors and also influence of environmental factors such as climate, location, and access to veterinary care can also be beneficial to horse health and welfare as horses are generally susceptible to a variety of illnesses.

7 References

- Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: a highly efficient gradient boosting decision tree. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 3149–3157.

- Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). Association for Computing Machinery, New York, NY, USA, 785–794.

- Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: unbiased boosting with categorical features. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18). Curran Associates Inc., Red Hook, NY, USA, 6639–6649.

- N. S. Rajliwall, R. Davey and G. Chetty, "Cardiovascular Risk Prediction Based on XGBoost," 2018 5th Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE), Nadi, Fiji, 2018, pp. 246-252, doi: 10.1109/APWConCSE.2018.00047.

- et. al., B. S. A. . (2021). Prediction of Type- 2 Diabetes using the LGBM Classifier Methods and Techniques. Turkish Journal of Computer and Mathematics Education (TURCOMAT), 12(12), 223–231.

- M. S. Oughali, M. Bahloul and S. A. El Rahman, "Analysis of NBA Players and Shot Prediction Using Random Forest and XGBoost Models," 2019 International Conference on Computer and Information Sciences (ICCIS), Sakaka, Saudi Arabia, 2019, pp. 1-5, doi: 10.1109/ICCISci.2019.8716412.

- Liu J, Wu J, Liu S, Li M, Hu K, Li K. Predicting mortality of patients with acute kidney injury in the ICU using XGBoost model. PLoS One. 2021 Feb 4;16(2):e0246306. doi: 10.1371/journal.pone.0246306. PMID: 33539390; PMCID: PMC7861386.