

NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

SC4000 PROJECT

GRP 43

Name	Matriculation Number
Jden Goh Jen Kiat	U2222711B
Ng Qi Wei	U2221323B
Lee Wei Jun Alan	U2220730A

Table of Contents

- 1. Problem Identification**
- 2. Challenges**
- 3. Exploratory Data Analysis**
- 4. Models**
- 5. Experimental Study for Effectiveness**
- 6. Conclusion**
- 7. References**

1. Problem Identification

For this Kaggle competition, we had to build a deep learning model to predict the health outcomes of horses given various medical indicators. The purpose of the model was then to determine whether the horses lived, survived or were euthanized.

2. Challenges

For this dataset, one of the challenges we face came from the categorical columns present in this dataset.

Firstly, there is a high number of categorical columns, and each category has multiple unique categories in them. Encoding these features without proper preprocessing can lead to the **curse of dimensionality**, especially when using **one-hot encoding**. Each unique category contributes a new dimension and will result in a high-dimensional feature space. We will take a look at how we can group the pre-process effectively to tackle this challenge.

There are a number of categorical columns including outlier categories, where certain labels occur very infrequently compared to others. Such categories do not contribute meaningfully to our predictive model and may instead add in noise that will affect the performance of our model. Moreover, an abundance of outlier category contributes to the above-mentioned challenge of high-dimension space when encoding our categorical features,

Some of the columns also had missing values which would pose significant challenges in evaluating the data efficiently. Missing values reduce the overall completeness of the dataset and columns with missing values further complicate the encoding process.

Hence, we explored several imputation techniques to combat this problem such as filling the missing values with the columns' corresponding mean or mode values.

3. Exploratory Data Analysis

Outcome Analysis

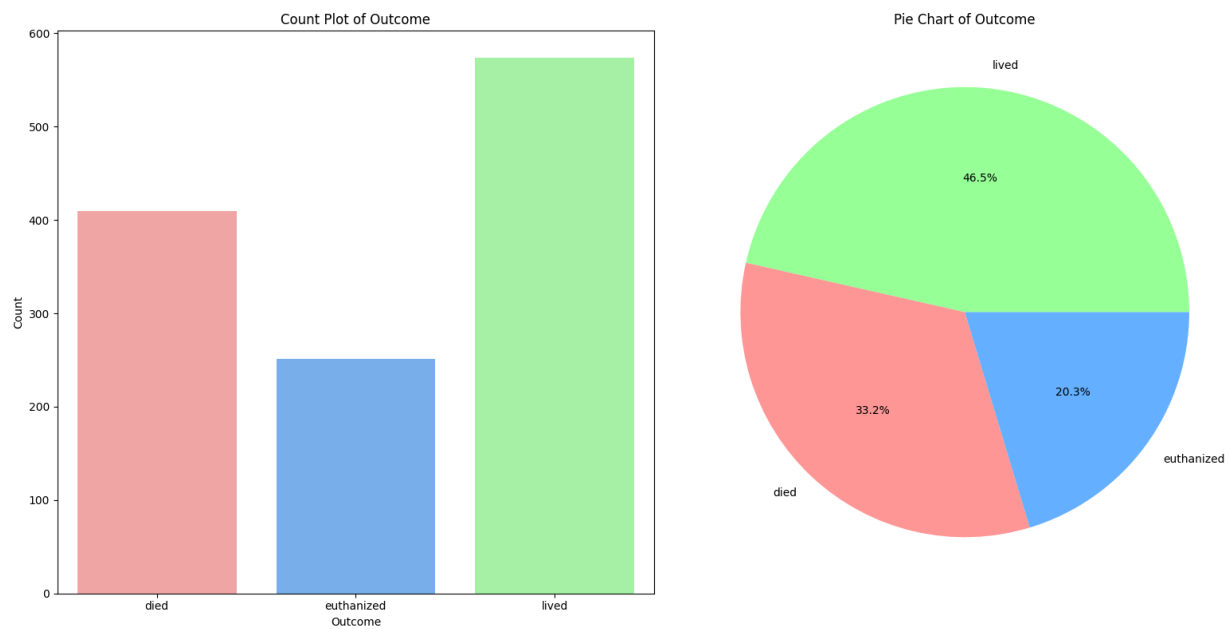


Fig.1 Outcome Analysis

Our first step was to analyze the distribution of our target **outcome**. It can be observed that:

- Majority of the horse lived (46.5%)
- Among the remaining horses, a larger portion of the horse **died** (33.2%) while the rest were **euthanized** (29.3%)

This was important for us to take note as we have undistributed outcomes and hence we will consider the use of resampling techniques later on to ensure a more balanced technique is used for training.

We also analysed the distribution of each column to identify which categories had an imbalance of data. E.g the categories `peripheral_pulse` and `temp_of_extremities` had data imbalances

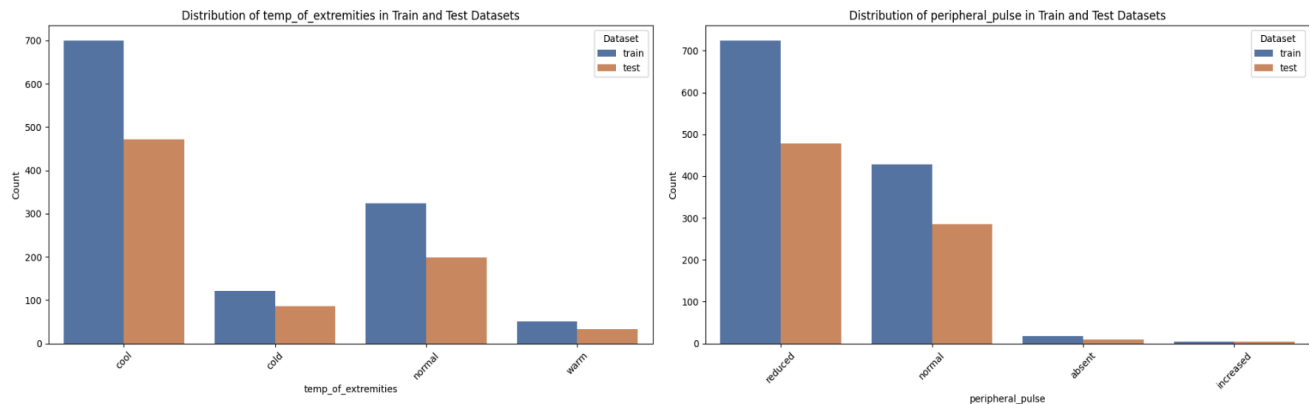


Fig.2 Examples of Skewed Categorical Data

As observed, **warm** and **cold** `temp_of_extremities` and **absent** and **increased** `peripheral_pulse` are severely underrepresented in their respective categories. This could be an issue when training our model as our model will be unable to learn much from these underrepresented category labels for their respective class label, which will lead to poor performance when trying to predict the outcome of horses with absent and increased peripheral pulse for example. There are a number of imbalanced categories in our dataset which we will have to take into account.

Correlation Matrix

Our next step was to take a look at the correlation between the numerical features as well as outcome.

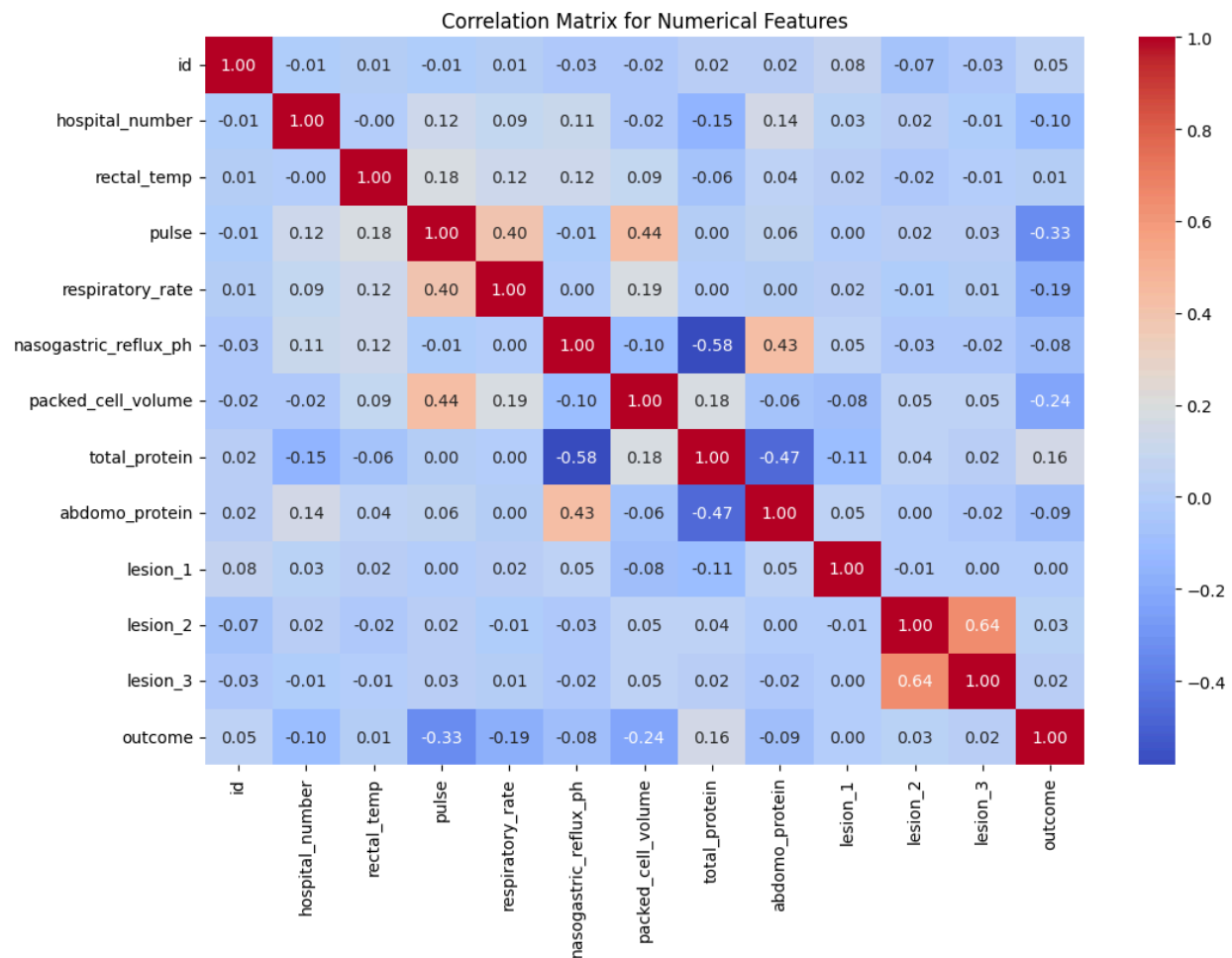


Fig.3 Correlation Matrix

We can observe that `hospital_number` has a very weak correlation with correlation with outcome, which should be expected, and we will be removing it later on as it has minimal predictive value.

Interestingly, for `lesion_2` and `lesion_3`, we observed a strong positive correlation, which we should take a look at further.

Pulse, packed_cell_volume and respiratory_rate have the strongest negative correlation with respect to outcome. We should look into these features more closely when trying to determine the outcome of a horse.

Lesion_2 and Lesion_3

We dived into the value_counts for `Lesion_2` and `Lesion_3`. They are categorical columns with each number representing certain information of the lesion.

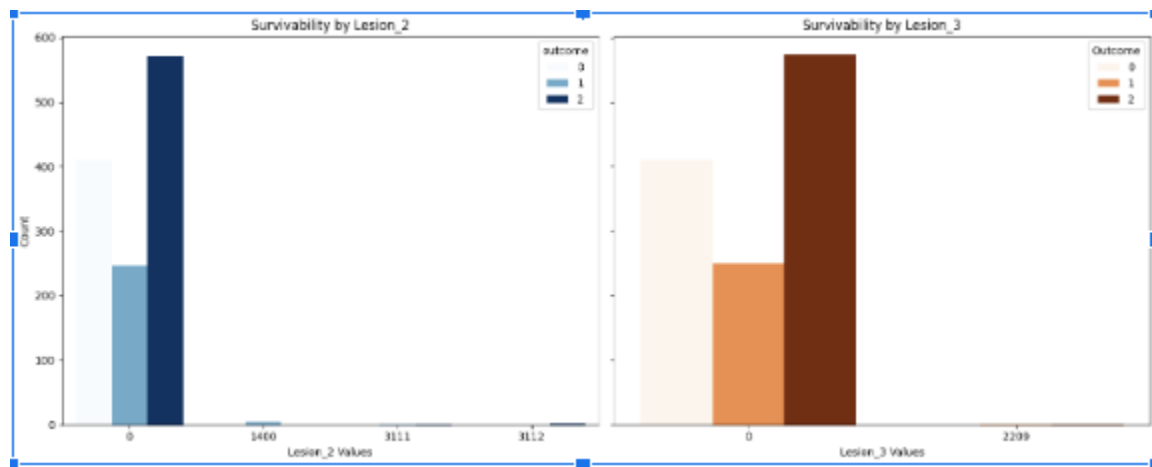


Fig.4 Lesion_2 and Lesion_3 Data Visualisation

As observed, data is highly skewed, with the majority of the values being 0. The existence of a few non-zero values is not enough to influence our prediction meaningfully. As such, with this data providing little meaningful value, we decided to drop these 2 columns for our training.

Pre-processing

Chi-Squared test of independence

For our categorical dataset, we decided to carry out a **Chi-Squared test of independence** to identify categorical variables that are significantly associated with the target variable `outcome`.

Our main reason for choosing is because we have a rather significant number of categorical features, and the Chi-Squared test will help us prioritise features that are relevant and remove any irrelevant or noisy features that will affect our model generalisability.

Output:

Feature: `lesion_3`

Chi-Square Statistic: 1.5384347269455936

P-Value: 0.4633755810904284

`lesion_3` does not have a significant relationship with outcome.

Based on our test, `lesion_3` was identified to not have a significant relationship with the outcome, with a p-value of `0.46` (>0.05). This reinforces our previous decision to drop the column.

Missing Entries

We moved on to handle our missing values in our dataset (both train and test).

Columns with Missing Values:

	0
temp_of_extremities	39
peripheral_pulse	60
mucous_membrane	21
capillary_refill_time	6
pain	44
peristalsis	20
abdominal_distention	23
nasogastric_tube	80
nasogastric_reflux	21
rectal_exam_feces	190
abdomen	213
abdomo_appearance	48

Fig.5 Missing Values

For our missing entries, we filled in with the:

1. **Mean**, for numerical features
2. **Mode**, for categorical features

Categorical Encoding

As mentioned, we have decided to analyze and select individual columns for **one-hot encoding** or **label-encoding**.

One-hot encoding is used to transform our **nominal categorical features**. This ensures that the model does not mistakenly interpret a relationship between categories, for example, the **surgery** column where its a binary “yes” and “no”. However, we must be careful with adding too many dimensions to our dataset when we are doing this.

Label encoding, on the other hand, is applied to **ordinal categorical features**, where the categories have a meaningful order. We are able to keep the inherent meaning behind the ranking of the categories when converting to numerical numbers. This includes features such as **pain**, where different rows reflect the level of pain (e.g., none < mild < moderate < severe for pain).

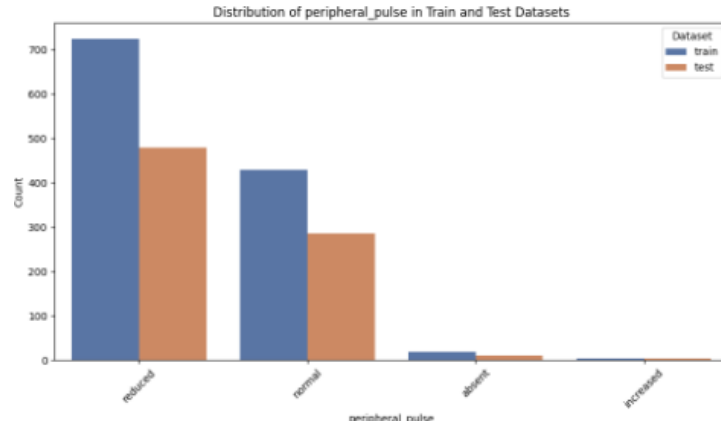


Fig6. Example of Categorical Feature with rare categories

Additionally, we addressed rare categories by grouping them into a unified **others** category. This step was necessary to handle cases with little data points in specific categories, ensuring consistency across both train and test datasets.

F-1 Score

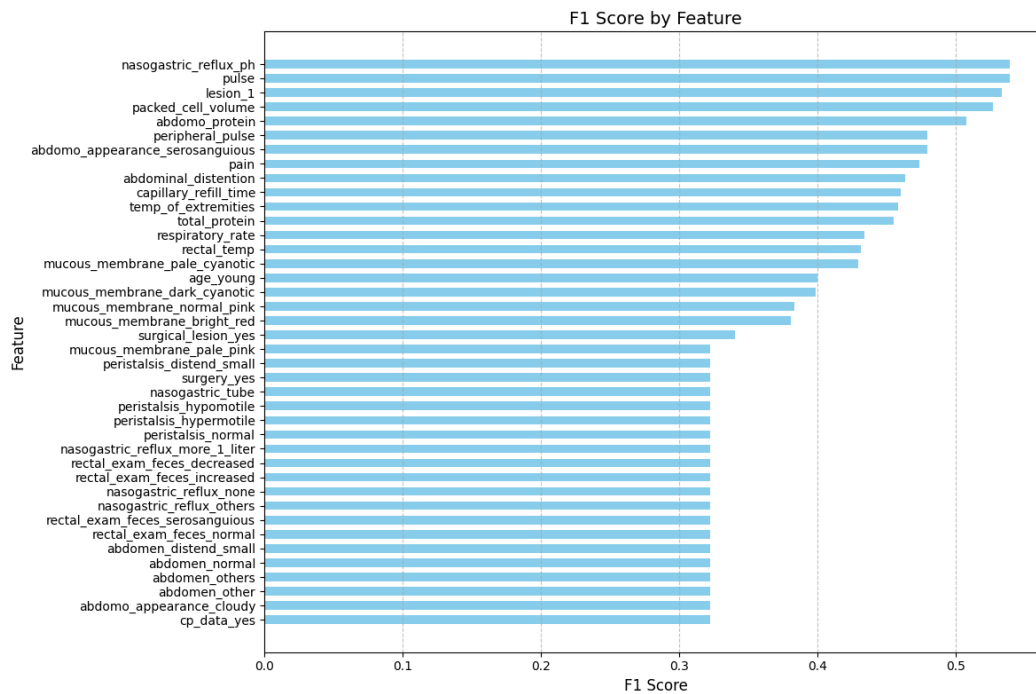


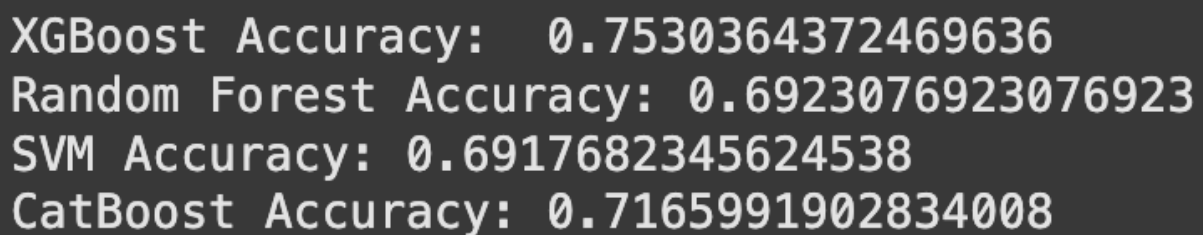
Fig7. F-1 Score

Lastly, we calculated and plotted the **F1 Score by Feature** to visualise the importance of each individual feature in predicting outcome. This helped us to gain better understanding on the features we have, especially after performing our encoding, and we can use it for our feature selection later on.

4. Models

Proposed Model Implementation

For this Machine Learning project, the kaggle competition we chose largely consisted of categorical and numerical values in its dataset. With some data processing techniques, we could then convert all the dataset into purely numerical values. This gave us the flexibility of choice to select from a wide plethora of machine learning algorithms such as Random Forest, XGBoost, CatBoost and Support Vector Machine. We learnt in SC4000 class about the use of ensemble learning so we planned to select the best 3 machine learning algorithms and utilise ensemble learning to finalise our machine learning model. By comparing the 4 machine learning algorithms on the raw datasets without pre-processing or feature engineering techniques, we then evaluated which 3 models would be the most accurate for our task.



```
XGBoost Accuracy: 0.7530364372469636
Random Forest Accuracy: 0.6923076923076923
SVM Accuracy: 0.6917682345624538
CatBoost Accuracy: 0.7165991902834008
```

Fig8. Accuracy of Base Models

As can be seen in Fig 8, Random Forest, CatBoost and XGBoost had the highest accuracy scores. Thus, we decided to use them for our ensemble learning.

Random Forest

Random Forest is a machine learning algorithm which constructs multiple decision trees through a process called bagging or bootstrap aggregating. The algorithm creates numerous different subsets from the original datasets through randomly sampling with replacement or otherwise known as bootstrapping. Each subset is used to train a separate decision tree, introducing diversity among the trees due to different data and feature subsets. When making predictions, the Random Forest aggregates the outputs of all individual trees. This ensemble technique reduces overfitting and improves predictive accuracy by balancing out individual trees' errors.

Why Random Forest?

Random forest is a robust model that combines multiple trees in a Random Forest , which can further reduce sensitivity to noise and outliers. Our dataset presents outliers in some categories and numerical columns, and we believe Random Forest will be able to handle them and generalise.

Methodology and Results from Random Forest

```
#Find Optimal Parameters
rf_random = RandomizedSearchCV(
    estimator=rf_model,
    param_distributions=param_grid,
    n_iter=50,
    cv=3,
    verbose=2,
    random_state=42,
    n_jobs=-1
)
```

Fig.9 Random Forest Hyperparameter optimization

We conducted hyperparameter optimization for our Random Forest model using RandomizedSearchCV. This process enabled us to identify the optimal parameter

configuration, ensuring improved model performance. These parameters were then utilised for training the model, which we subsequently used to make predictions on the dataset.

We managed to attain an accuracy of 0.72.

Categorical Boosting

Categorical Boosting(CatBoost) is a gradient boosting algorithm that builds an ensemble of decision trees sequentially, where each new tree aims to correct the errors of the combined previous trees. By using techniques like ordered boosting to prevent target leakage and overfitting, CatBoost provides high-performance models that work well out-of-the-box, especially on datasets with numerous categorical variables.

Why CatBoost?

CatBoost works well out of the box and requires minimum parameter tunings. It is able to handle categorical data efficiently, which is ideal for us given the high-dimensionality of our categorical features.

It is also a robust technique that reduces overfitting on small datasets, which will be beneficial for our dataset and it will help the model generalise better.

Methodology and Results from CatBoost

```
# Initialize RandomizedSearchCV
cat_random = RandomizedSearchCV(
    estimator=cat_model,
    param_distributions=param_grid,
    n_iter=50,
    cv=3,
    verbose=1,
    random_state=42,
    n_jobs=-1
)
```

Fig.10 Catboost Hyperparameter optimization

We repeated this process for CatBoost which was much faster and easier as compared to Random Forest. This allowed us to optimise the model further before we finally used it to make predictions.

We managed to attain an accuracy of 0.73.

Extreme Gradient Boosting

Extreme Gradient Boosting (XGBoost) is an optimised gradient boosting framework which creates an ensemble of decision trees sequentially, where each tree corrects the errors of its predecessors.

Why XGBoost?

XGBoost has built-in regularisation parameters like L1 regularisation and L2 regularisation. This can further help in feature selection by reducing the contribution of irrelevant or redundant features in these high-dimensional datasets.

XGBoost can also capture complex interaction between our features given the high-dimensionality of our data. We believe this model can uncover hidden patterns that might influence the target outcome that other models may not be able to pick out.

Of course, XGBoost is also known for its robustness in handling a mix of categorical and numerical data, and hence we decided to pick XGBoost as well.

Methodology and Results from XGBoost

```
# Perform Optuna optimization
study_xgb = optuna.create_study(direction='maximize')
optuna.logging.set_verbosity(optuna.logging.WARNING)
study_xgb.optimize(objective_xg, n_trials=50, show_progress_bar=True)
```

Fig.11 XGBoost Hyperparameter optimization

For XGBoost, we used Optuna optimization which is a state-of-the-art hyperparameter optimization framework that is designed to automate the search for optimal hyperparameters. This allowed us to maximise the model's performance by finding the best parameter settings. Thus, the optimised model achieved better accuracy and robustness, leading to improved prediction results.

We managed to attain an accuracy of 0.75.

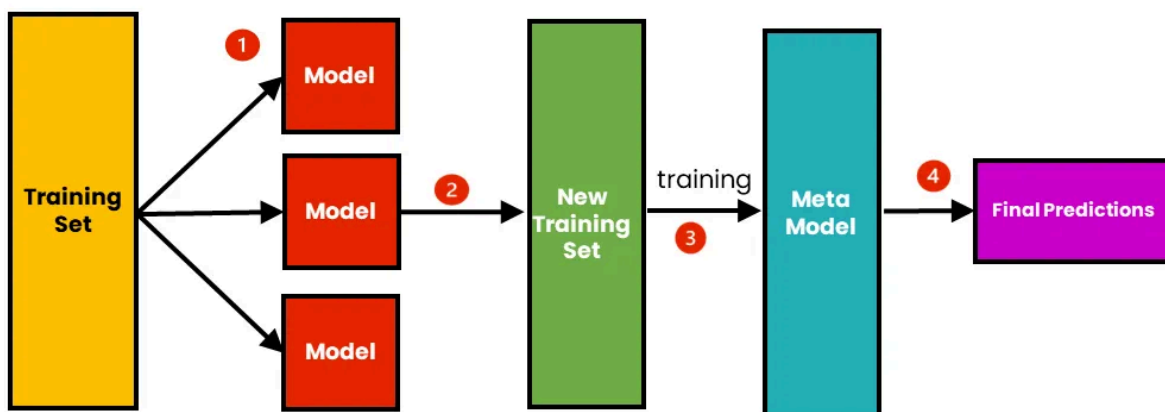
Ensemble Learning

In order to combine the results from these 3 base models, we used ensemble learning. Ensemble learning aims to leverage the strengths of different classifiers, which ideally should be independent and each exhibit an error rate of less than 50%. Since our three models meet these criteria, they are suitable candidates for ensemble learning. By combining the predictions from these classifiers, ensemble learning makes a final, more accurate decision. For this project, we have chosen to implement stacking.

Stacking is a powerful ensemble learning technique where multiple base models are trained to predict the output, and their predictions are then used as input for a final model known as a meta model. The meta model then learns an optimal way to combine these predictions before arriving at a final output. The idea behind stacking is to leverage the strengths of various models and combine them in a way that reduces overall error.

For this project, we also adjusted the weights of each base model to optimise their contributions to the final predictions. This allowed us to give more importance to base models which performed better.

The Process of Stacking



This stacking ensemble learning model which combined the results from Random Forest, CatBoost and XGBoost was then our final model used to generate predictions for the competition.

This ultimately helped us achieve our target result of 0.79268.

Kaggle Results

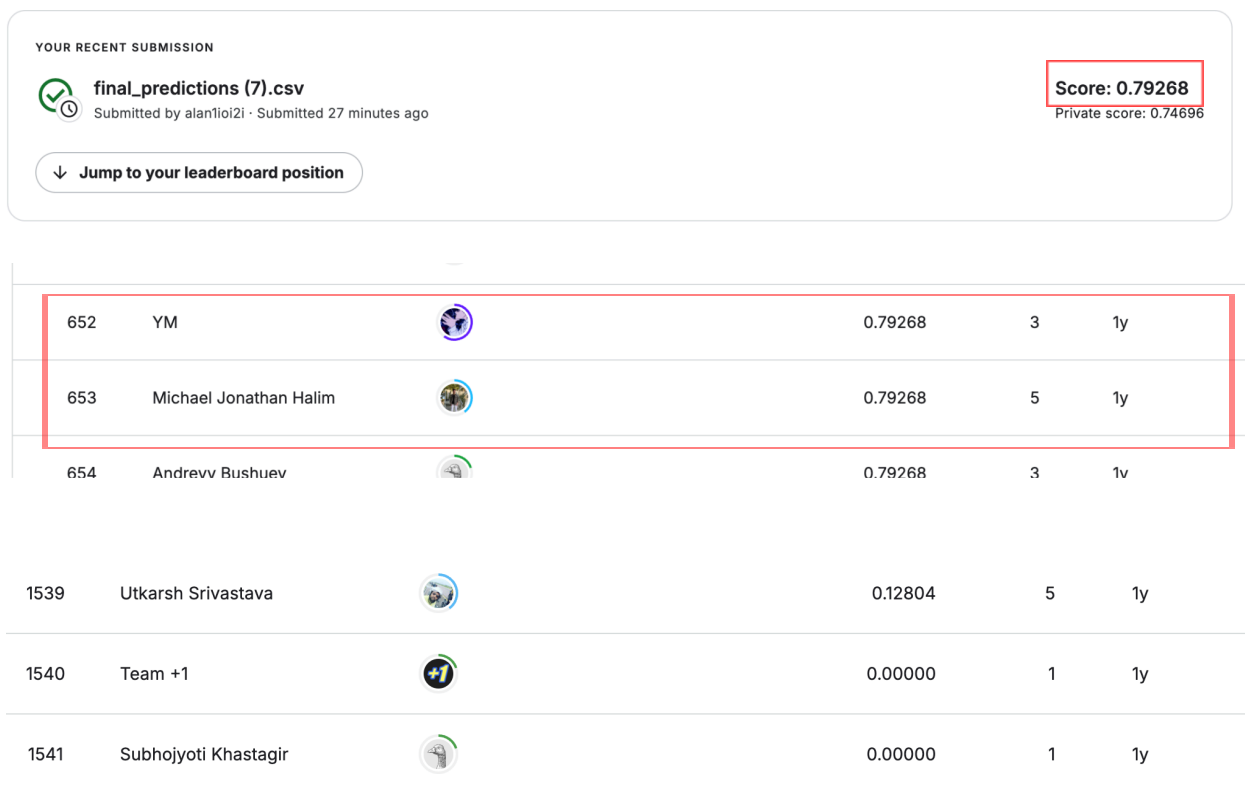


FIG 12. Competition Score

As seen from Fig 12, we obtained a final public score of 0.79268 where we would finish 642/1541 on the public leaderboard or top 41%.

5. Experimental Study for Effectiveness

SMOTE

During our pre-processing, we tried using **SMOTE** to balance our dataset, as a larger portion of the data belonged to horses that lived, and we were afraid our model would not be able to learn the features of the other 2 classes effectively.

However, training our models using our oversampled data gave us an average decrease of 0.3 in our accuracy across the 3 models we used. This can be attributed to the introduction of noise into our data due to the high variance of our minority class.

Also, our dataset is not large enough, hence our model is more sensitive to noisy data and may perform worse instead.

Encoding

Our initial encoding choice was only label encoders, which gave use a rather robust model that was able to achieve an accuracy of 0.77. However, upon deeper understanding of our dataset, we took a step back and adjusted the way we encode our different categorical data based on our understanding of the features.

This gave our model accuracy an average increase of 0.2, which meant that the model is able to derive better insights into our problem and generalise better.

6. Conclusion

In conclusion, this project has allowed us to gain a deeper insight into handling complex machine learning tasks involving high-dimensional data, imbalanced target classes, and a mix of categorical and numerical features.

After our careful pre-processing, feature engineering and model selection, we were able to address the challenges we faced in this dataset and work towards improving the predictive performance.

The use of our ensemble learning was impactful as we were able to combine different base models and architecture to boost our score. We were able to capitalise on the strengths of each algorithm. Additionally, introducing novel ideas like adjusting the weights of the base models allowed us to prioritise the contributions of the most effective models, leading to a significant boost in performance.

Overall, we were able to systematically explore, validate and further refine our solutions as we went through the tasks, which further helped boost the validity and convincingness of our project as we employ the above techniques to overcome the challenges we faced.

7. References

<https://medium.com/@denizgunay/random-forest-af5bde5d7e1e>

https://www.researchgate.net/figure/The-flow-diagram-of-the-CatBoost-model_fig3_370695897

<https://www.nvidia.com/en-us/glossary/xgboost/>

https://medium.com/@brijesh_soni/stacking-to-improve-model-performance-a-comprehensive-guide-on-ensemble-learning-in-python-9ed53c93ce28