Adam Chen z5363476, Ethan Fong z5363479

Introduction

Problem

As the life expectancy continues to rise whilst birth rates steadily decrease, the aging population has seen a surge in recent decades which is expected to continue. Aging is a tremendous risk factor for a variety of health issues spanning from physical such as cardiovascular diseases and arthritis to mental such as dementia. Diagnosing these medical conditions demands a long, intrusive, uncomfortable process to observe patients. A key aim is to mitigate this through predictive models which utilize the relationships between key health characteristics and susceptibility to certain conditions. The final solution to be proposed is a probabilistic predictive model which outputs the likelihood of a given patient having one of three age related conditions. [1]

Dataset

This Kaggle competition provided a dataset of 56 features along with a class label. These 56 features corresponded to an anonymized health characteristic linked to the three conditions whilst the class label indicated whether a sample was diagnosed with any of three conditions (Class 1) or was given a negative diagnosis (Class 0).

Of the 56 features, one (EJ) was interpreted as a nominal categorical feature which took up two possible arbitrary values A or B. In addition, the dataset also had a few missing values with four features missing one value, two missing two, one missing three and two missing 60.

Another dataset called ‘greeks.csv’ was also provided which Kaggle called ‘supplemental metadata’. This dataset was only provided for the training set and contained three different pieces of information:

- Alpha: Indicates the condition which is present.
    - A equates to no condition (Class 0)
    - B, D and G correspond to the three different conditions (Class 1)
    - This opened up the opportunity to train a multiclass classifier whose predictions can be transformed back to binary. For instance, a prediction of A for alpha equates to class 0 and B, D or G correspond to Class 1.
- Belta, Gamma, Delta: Kaggle described these as ‘experimental characteristics’ which indicates that they were unsure of its links to the conditions.
    - This provides an interesting task to determine whether these characteristics were linked to the conditions.
    - However, since these characteristics were exclusively provided for training, it is unclear how they can be utilised in the model for predicting from the test set.
- Epsilon: Date of data collection. This can be utilised to extract seasonal or time related trends observed within any features which relates to the time series problem explored in dealing with missing values.

Methods

In addition to the binary classifiers, multiclass classifiers were also experimented with. Multiclass classifiers were trained using the supplemental data with their predictions converted to the corresponding binary class value.

Categorical feature encoding

Most machine learning models do not accept categorical features and as such, some form of encoding is required to transform these categorical features into a numerical equivalent. Two different methods were explored for nominal categorical features: [2]

Count/Frequency Encoding: Replace the values with its frequency. For example, if ‘A’ occurs 100 times and ‘B’ occurs 200 times, 100 is assigned to ‘A’ and 200 to ‘B’.

Probability Ratio Encoding: Replace values with the probability ratio P(1)/P(0). For example, if for all observations with 'A', 100 are of class 1 and 50 are of class 2, A is substituted with 100/50 = 2.

The decision was made to use probability ratio encoding for binary classification and frequency encoding for the multiclass classifiers.

Missing values

As specified in the introduction, the given dataset had features with missing values. Kaggle's test set also likely had the same issue. Common techniques to deal with missing values are dropping affected features/observations or inputation.

Inputation can be categorized into a time series problem or a general problem. A time series problem refers to a dataset which varies over time caused by trends patterns. When trend patterns do not exist, a general problem is considered.

Since the dataset cannot confidently be categorized as a time-series problem, it will be considered a general problem. For general problems, the mean, media or mode are most commonly put in place of the missing values. Since the features were numerical, the mean was used. [3]

Imbalanced Dataset

Class imbalance is a notable concern for this dataset. Out of the 617 samples provided, 108 were of Class 1 which constituted only 17.5% of all samples. This is troublesome for classification problems since the trained model will inherently be biased towards the majority class and subsequently perform poorly for the minority class. Class Weighting and Resampling were explored to account for the imbalanced data. [4]

Class Weighting

Class weights help by assigning a higher weight to the minority class when training the model. More specifically, a greater priority is given to the minority class by influencing the cost function in a way which will result in a higher relative penalty for a misclassified minority class. [5]

The employed formula for class weights is given by:

$$w_i = \frac{n_{total}}{n_{classes} \times n_i}$$

$n_{total} = number\ of\ samples,$

$n_{classes} = number\ of\ different\ classes,$

$n_i = number\ of\ samples\ in\ class\ i$

When used on the dataset, the Class 0 has a weight of 0.61 and Class 1 has a weight of 2.86. Class weights were employed using the class weight hyperparameter when performing grid search cross validation.

Resampling

Undersampling was not considered to account for the already underloaded sample size so oversampling was explored. There were two main methods of oversampling of interest which were random oversampling (ROS) and synthetic minority oversampling (SMOTE). [4]

ROS duplicates existing examples through random selection with replacement, aiming to replicate the effect of increasing samples in the minority class.

SMOTE generates new instances by randomly selecting points on lines between existing examples.
Oversampling was only performed on the training set. Since grid search was employed, care was taken to ensure that the validation set in each fold was not being oversampled. This was accomplished through pipelines in python's sklearn library.

Stratified sampling
Another consideration of imbalanced datasets is the distribution of each class in the training and test sets. Ideally, the training set should be a perfect representation of the test set, void of sampling bias. Stratified sampling solves this by partitioning the total population into subgroups corresponding to each class before selecting instances from each group to ensure that the training and test set accurately represent the proportion of the entire dataset. In the case of the provided dataset, since 17.5% of the samples were of class 1 then the train test split utilised stratified sampling to ensure that the train and test sets will also inherit that same class distribution. [4]

Tuning hyperparameters

All classifiers which were considered were Logistic Regression, K Nearest Neighbors, Gaussian Naive Bayes, Decision Trees, Support Vector Classifiers along with ensemble methods Random Forest and Adaptive Boosting. After running preliminary grid search on these models, the two best performing were Random Forest and Adaptive Boosting whilst Logistic Regression was deemed the most computationally efficient relative to performance. These models were the eventual focus of hyperparameter tuning. Hyperparameters were chosen based on preliminary testing of each parameter and gauging their impact on performance.

Learning Model	**HyperParameters**	**Effects**	**Values**
Logistic Regression	C	Corresponds to the complexity of the model. A smaller value specifies stronger regularization and leads to less overfitting.	1, 0.1, 0.01
	Fit intercept	Determines whether a constant to represent the bias should be added to the model.	True or False
	Class weights	Class weights determine how strongly samples of each class contribute to training. Used to address class imbalance. Class weights are inversely proportional to class frequency.	'Balanced' or None
Random Forest	N estimators	The number of estimators in the ensemble	100, 200, 400 , 800
	Criterion	The method used for node splitting in the decision trees	Gini, Entropy, Log Loss
	Max depth	The max depth of each tree in the ensemble	1, 3, 5, 7

	Class weights	Used to address class imbalance similar to logistic regression	'Balanced' or None
Adaptive Boosting	N estimators	The number of estimators in the ensemble	50, 100, 250, 500
	Base models	The base models that compose the ensemble	Decision Trees with balanced class weights and max depth of one of: 1, 2, 3 and criterion of one of: Gini, Entropy, Log Loss

Grid Search Cross Validation

Hyperparameters were tuned through Grid Search Cross Validation (GridSearchCV). Even though GridSearchCV is a computationally exhausting process having to test every possible permutation of hyperparameters, the relatively small dataset mitigated this. GridSearch will also be performed with different ways of oversampling to investigate the most optimum hyperparameters along with the ideal way of dealing with the imbalanced data.

K-Fold Cross Validation

K-Fold cross validation was also performed during this GridSearchCV using five folds. This means that for every model, there was a 80:20 split between the training and validation set. This provided an accurate evaluation of each model's performance, taking the average of the model's scores across five arbitrary datasets.

Standardization

To prevent data leakage, standardization was fitted to the training set before being applied to the validation and test sets. This includes the training and validation splits inside each fold of grid search cross validation.

Evaluation Metrics

When determining the best performing model during grid search cross validation, a metric is supplied. As mentioned previously, the imbalance nature of the dataset unveils a flaw with the default metric of accuracy which is simply correct predictions divided by total predictions. Since class 0 samples account for 82.5% of total samples, a model which predicts class 0 will be able to achieve an accuracy of 82.5% assuming equal distribution in the test set. As such, metrics beyond accuracy had to be considered and favored for evaluation.

F1 Score [7]

The F1 score is typically used to evaluate performance for imbalanced datasets and returns the harmonic mean of precision and recall. Precision refers to how many positive predictions were correct whilst recall refers to how many of the positive samples were correctly predicted.

In other words,

$$Precision = \frac{TP}{TP + FP}, \; Recall = \frac{TP}{TP + FN}$$

$$TP = True\ Positives, \; FP = False\ Positives, \; FN = False\ Negatives$$

Hence, the formula for F1 score is given by

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

The F1 score applied on the previous model which predicts 0 for every input obtains an F1 score of 0 since there were no positives being predicted which means $TP = 0$. Thus, the F1 score is a more suitable measure than pure accuracy when evaluating performance on imbalance datasets. In this context where False Negatives are prioritized, another metric called F2 which weighs the precision and recalls differently can be used although it wasn't considered for this project. For multiclass classifiers, the macro F1 score will be used which is the sum of F1 scores for all the classes divided by the number of classes. This metric is preferred over the alternative F1 micro [6].

Balanced Logarithmic Loss (BLL) [8]
The competition required that the predictions be made in the form of probabilities. In that sense, F1 score falls short and another metric is required. The metric which Kaggle used to evaluate the predictions is the balanced logarithmic loss. The formula is given below:

$$\text{Log Loss} = \frac{-\frac{1}{N_0}\sum_{i=1}^{N_0} y_{0i} \log p_{0i} - \frac{1}{N_1}\sum_{i=1}^{N_1} y_{1i} \log p_{1i}}{2}$$

Where N_c is the number of samples in class C,

y_{ci} is 1 if sample i is in class C and 0 otherwise,

p_{ci} is the predicted probability of sample i belonging to class C.

Note that the natural logarithm is used here and to avoid extremities with the log function, p was replaced with $\max(\min(p, 1 - 10^{15}), 10^{15})$.

This loss function is somewhat of an adaptation on the prototypical log loss function with consideration of class imbalance, ensuring that each class contributes an equal amount to the loss.

Since balanced logarithmic log loss was the criteria which Kaggle prioritized, it was decided to evaluate the models using the balanced logarithmic loss during grid search. Nonetheless, the F1 score remained an insightful metric for final model evaluation.

Feature selection

Generally speaking, if a model is provided with more features to work with, there is more information to work with which culminates in a more accurate prediction. However, data is commonly inundated with noise. The more features present, the more potential noise which the model will misinterpret as information. On the surface, it was deemed plausible that certain features of the dataset deserved to be dropped, considering that some features were missing values with two missing 60 as well as the categorical feature 'EJ' which was arbitrarily encoded.

The method of choice was **forward feature selection** which operates by selecting features based on the performance of the model trained using those features.

The algorithm is as follows [9]:

1) Train n models where n is the number of features and take the feature which provides the best performance.
2) Train n - 1 models which combine the previous best feature and the remaining n - 1 features and take the feature which led to the highest improvement and repeat until the desired number of features is reached or until there is no more significant improvement or perhaps even a decline in performance.

Since the number of trained models is quadratic, it is ideal to use a relatively low computationally exhaustive model. Hence, logistic regression was utilized here using tuned hyperparameters. Feature selection was only applied on the best performing model in the interest of time.

Results

Hyperparameter tuning

Optimal hyperparameters (scores were obtained by taking an average of 5)

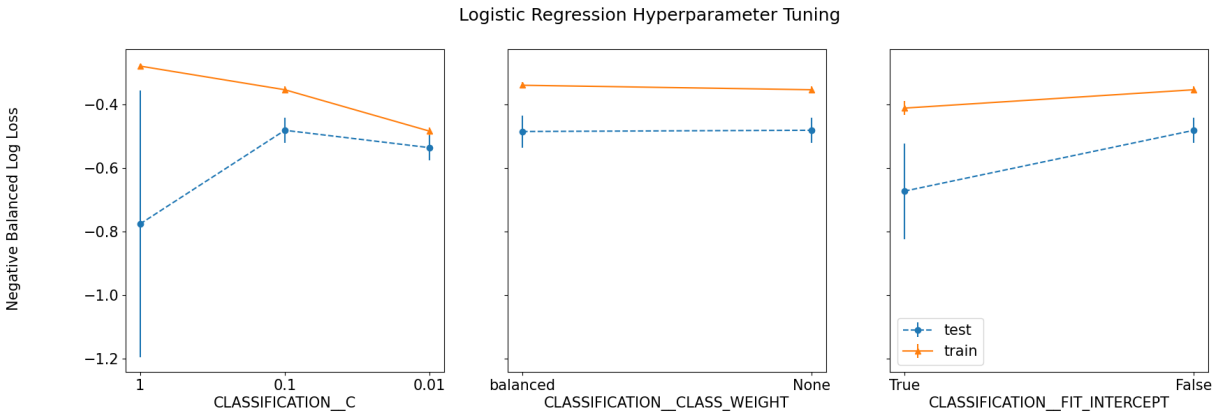
Binary

Model	Optimal Hyperparameters	Oversampling	BLL score	F1 Score	Accuracy
Logistic Regression	C: 0.1, class_weight: None, fit_intercept: False	No	0.380	0.716	0.876
Random Forest	criterion: log loss, max_depth: 7, N_estimators: 800	SMOTE	0.306	0.843	0.942
Adaptive Boosting	Estimator: (Decision Tree using criteria of entropy, max depth of 3), n_estimators: 500	SMOTE	0.314	0.837	0.941

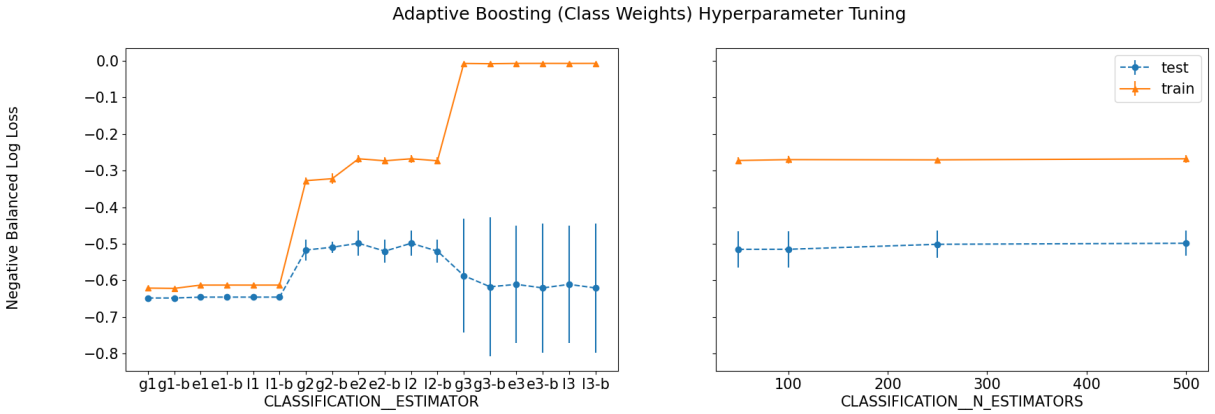
MultiClass

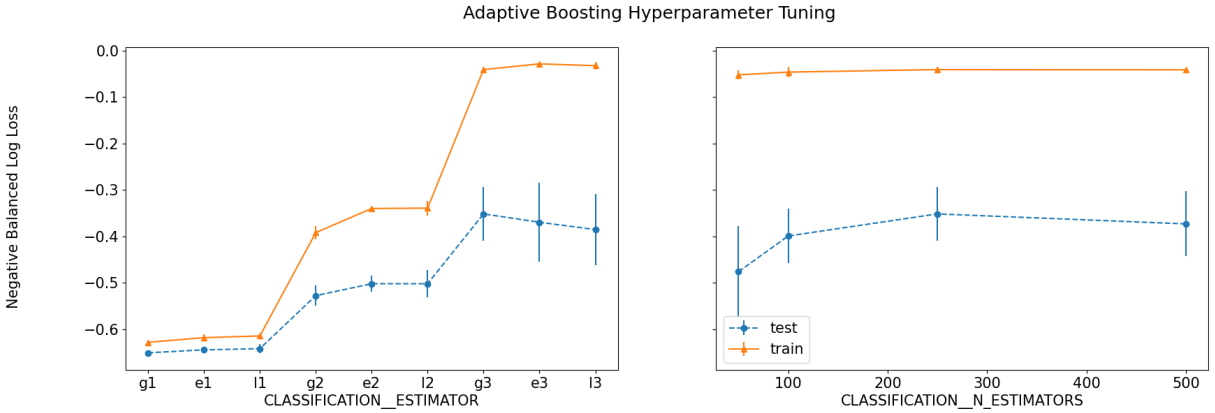
Model	Optimal Hyperparameters	Oversampling	BLL score	F1 Score	Accuracy
Logistic Regression	C: 0.01, class_weight: Balanced, fit_intercept: False	No	0.457	0.63	0.846
Random Forest	criterion: entropy, max_depth: 7, N_estimators: 200	ROS	0.377	0.712	0.907
Adaptive Boosting	Estimator: (Decision Tree using criteria of entropy, max depth of 3), n_estimators: 500	SMOTE	0.640	0.584	0.832

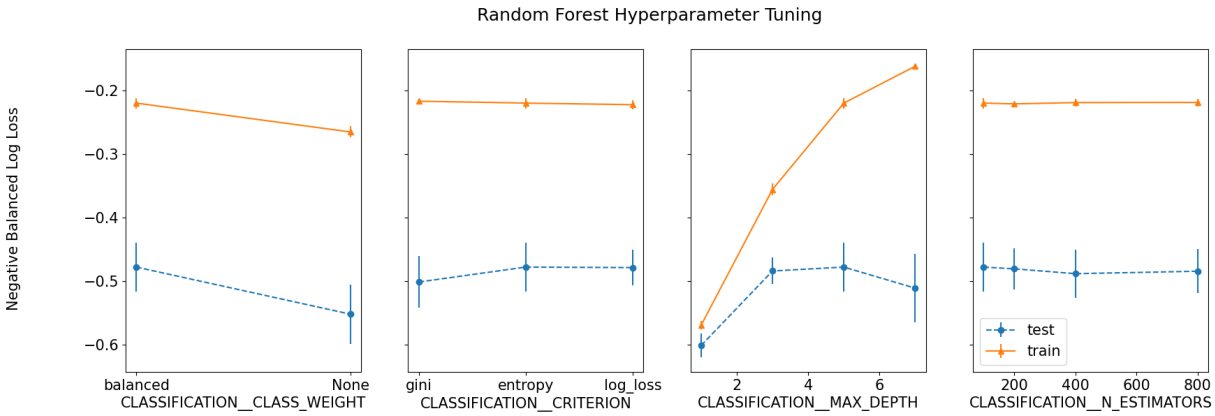
The effects of each hyperparameter on the mean score of all splits is graphed below:



For the Estimator hyperparameter, the letter in the x-axis labels refer to the criterion being used (g=Gini, e=Entropy, l=Log Loss). The number refers to the max depth decision trees that make up the ensemble. The b is present only on the *Adaptive Boosting (Class Weights)* graph and is present when the data was balanced with class weights, as opposed to being left unbalanced.







Logistic Regression and Random Forest showed similar hyperparameter trends both when the data was balanced with class weights and when the data was balanced with oversampling. Class weights seem to have little to no effect on Adaptive Boosting, and shows signs of overfitting for max tree depth of 3 when the data is left unbalanced or balanced with class weights. When Adaptive Boosting is run on oversampled data, the overfitting for max tree depth 3 is no longer present.

ROS and SMOTE oversampling

Binary (scores were obtained by taking an average of 5 for the best performing models)

Model	No sampling	Score with ROS	Score with SMOTE
Logistic Regression	0.413 (no class weights)	0.425	0.421
Random Forest	0.395 (no class weights)	0.355	0.326
Adaptive Boosting	0.392 (class weights used by base estimator decision trees)	0.397	0.324

Multi Class (scores were obtained by taking an average of 5 for the best performing models)

Model	Score with class weights/no sampling	Score with ROS	Score with SMOTE
Logistic Regression	0.457 (balanced class weights)	0.473	0.461
Random Forest	0.496 (no class weights)	0.381	0.377
Adaptive Boosting	0.645 (class weights used by base estimator decision trees)	0.640	0.645

For the binary models, logistic regression performed best with no sampling and not even class weights which was unexpected. The two ensemble methods on the other hand performed best with SMOTE oversampling by quite a large margin in terms of BLL.

The multiclass models performed noticeably poorly compared to the binary models. Logistic regression performed best using class weights although there was not a huge difference compared to oversampling. Meanwhile, Random Forest performed best with oversampling whilst Adaptive Boosting was largely neutral. There was no discernible difference between ROS and SMOTE.

Feature Selection

Feature selection was performed on Adaptive Boosting with its optimized hyperparameters using the SMOTE oversampling method.

Number of features	BLL (Mean over 10)	F1 (Mean over 10)	Accuracy (Mean over 10)
5	0.427	0.712	0.887
10	0.347	0.731	0.894
15	0.343	0.771	0.902
20	0.339	0.768	0.918
25	0.333	0.786	0.921
30	0.326	0.798	0.924
35	0.328	0.802	0.929
40	0.322	0.825	0.941
45	0.314	0.849	0.939
50	0.311	0.847	0.942
All features retained (56)	0.317	0.844	0.941

From the results, the dataset appears to contain several redundant and perhaps even harmful features since the performance of the model which includes all 56 features consistently performs worse than some of those with lower dimensions. BLL scores remained relatively steady across the entire dataset with the differences more prevalent regarding the F1 score and accuracy. Ultimately, feature size of 50 was deduced to be most ideal based on the results.

Submission to Kaggle

To summarize, the model submitted to Kaggle was a binary Adaptive Boosting classifier using the optimal hyperparameters concluded from grid search. Categorical features were encoded using probability ratio encoding, missing values were replaced with the feature mean, the data was standardized and then forward feature selection was used to select the best 50 features. SMOTE oversampling was used to account for the imbalance.

Discussion

Comparison of different methods, their features and performances

Logistic regression takes the least amount of time to train, however produces the worst results in all metrics.

Random Forest and AdaBoost were very comparable in their results. They both achieved similar scores and both took approximately the same amount of time to train. In general, AdaBoost typically tends to be more accurate in classification than Random Forest, but in this case, their scores were very comparable. The decision to choose AdaBoost as the final submission to Kaggle was based on the marginal superiority in BLL score.

Binary vs Multiclass
The results comprehensively indicated that the multiclass classifiers performed worse than the binary classifiers overall. This can be attributed to the exacerbation of an already imbalanced dataset. The distribution of the dataset was 509, 61, 18 and 29 which equates to 82.5%, 9.9%, 2.9% and 4.7%. Stratified sampling of the training and test set yielded a regular margin of 0.2% for each class so in some instances, the class with 2.9% of samples had a 2.7% or 3.1% share in the splits which led to poor performance as well as huge variation in results. In addition, a lack of samples for the smallest samples likely increased the probability of noise being interpreted as information.

Handling of Imbalanced dataset
Logistic regression rejecting class weights may be a result of the grid search favoring a graph with no intercept added to the decision function. Since unbalanced training data predominantly affects the intercept of the model, balancing the data with class weights on a logistic regression model with a fixed intercept would result in no noticeable difference. Therefore, GridSearch rejected class weights since they were only slightly less performant, and from the hyperparameter graphs, had little to no effect. [10]

The results remained fairly neutral on SMOTE or ROS for the multiclass case. A possible reason for this is since SMOTE generates samples between two points of the same class, the points generated using two points from class 1 could now be from different classes in the multiclass situation meaning that there was less variation between the generated samples using ROS and SMOTE.

Feature Selection
Feature selection marginally improved upon performance. There was an ideal region between 45 and 50 which yielded the best results with a steady decrease in performance from 40 features down to 5. In saying this, the performance of even the lowest number of features was by no means awful and if computational exertion or time was a priority which wasn't the case here, lower feature counts could certainly be applied.

Future improvements
In dealing with missing values, another viable strategy is treating the features with missing data as a target variable and training models (usually simpler such as linear or logistic regression) with the remaining features to predict for them. Since missing data did not occupy too much of the dataset, encoding was employed instead.

Only out of the box models were explored for this project. Briefly observing the discussions of other contestants, more complex solutions such as meta models were brought up frequently. Given more time and exposure, experimenting with self-engineered solutions would hopefully improve performance as well as provide a more insightful learning experience.

Conclusion (2 marks)
A variety of different predictive models, hyper-parameters, dimension reduction and oversampling techniques were tried and tested to produce a model for ICR to identify age-related conditions. Optimal hyper-parameters were found using grid search, and the resultant models were further refined using feature selection.

Oversampling, especially SMOTE oversampling, was shown to be effective in increasing the accuracy of the ensemble methods, however did not seem to impact Logistic regression at all. Feature selection was marginally effective in increasing the efficacy of Adaptive Boosting, the chosen model.

Despite the variety of methods researched and implemented to address the class imbalance found in the training data,

there are still many more avenues left unexplored, and more nuance and refinement that could be applied to our techniques applied here.

Unfortunately, the final submission made to Kaggle ran into an unexpected error which is likely to do with missing values with Kaggle's hidden test set. Due to Kaggle's policy of one submission per day, the outcome of this final submission will not be included here. A previous successful attempt featuring more crude methods achieved a score of 0.38 which placed approximately 5000th on the leaderboard.

References

[1] InVitro Cell Research. 2023. "ICR - Identifying Age-Related Conditions: Overview". *Kaggle*, May 12. https://www.kaggle.com/competitions/icr-identify-age-related-conditions/overview

[2] Chaturvedi, Saket. 2020. "Categorical Encoding Methods". *Analytics Vidhya* (blog), May 10. https://medium.com/analytics-vidhya/categorical-encoding-methods-535b5c289512#:~:text=Probability%20Ratio%20Encoding%20%E2%80%94%20This%20method,the%20target%20being%20False%20%2F%200
.

[3] Nawale, Tejashree . 2022. "How to deal with Missing Values in Machine Learning". *Geek Culture* (blog), May 18, 2022. https://medium.com/geekculture/how-to-deal-with-missing-values-in-machine-learning-98e47f025b9c

[4] Rutecki, Marcin. 2022. "Best techniques and metrics for Imbalanced Dataset". *Kaggle* (blog), December 29. https://www.kaggle.com/code/marcinrutecki/best-techniques-and-metrics-for-imbalanced-dataset#4.-Data-pre-processing

[5] Singh, Kamaldeep. 2023. "How to Improve Class Imbalance using Class Weights in Machine Learning?". *Analytics Vidhya* (blog), July 6 2023. https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/

[6] Narasimhan, Harikrishna, Weiwei Pan, Purushottam Kar, Pavlos Protopapas, and Harish G. Ramaswamy. "Optimizing the multiclass F-measure via biconcave programming." In 2016 IEEE 16th international conference on data mining (ICDM), pp. 1101-1106. IEEE, 2016. https://www.cse.iitk.ac.in/users/purushot/papers/macrof1.pdf

[7] Panchal, Sandeep. 2019. "Performance Metrics of Supervised Learning". *Analytics Vidhya* (blog), July 6. https://medium.com/analytics-vidhya/performance-metrics-precision-recall-f1-score-efb51ac111bd

[8] InVitro Cell Research. 2023. "ICR - Identifying Age-Related Conditions: Evaluation". *Kaggle*, May 12. https://www.kaggle.com/competitions/icr-identify-age-related-conditions/overview/evaluation

[9] Singh, Himanshi. 2021. "Forward Feature Selection and its Implementation". *Analytics Vidhya*, April 9. https://www.analyticsvidhya.com/blog/2021/04/forward-feature-selection-and-its-implementation/