AI6102 Group Project
Kaggle Competition: Identifying Age-Related Conditions

Chua Han Chong (G2305117G)
Ling Shahrul Al-Nizam (G2304209B)
Tan Jie Heng Alfred (G2304193L)

Abstract

In this project, we trained a Catboost classifier for a binary classification task. We experimented with many techniques such as Synthetic Minority Over-sampling Technique (SMOTE), variance and F-score thresholding, hierarchical clustering, and Principal Component Analysis (PCA) to address the different issues inherent in the dataset. Finally, with our best model, we attained a testing loss of 0.4257.

Introduction

In this project, we participated in a previous Kaggle competition [4]. The aim of this competition is for us to predict if a person has any of three medical conditions, given the measurements of some key characteristics (i.e., the features). Because medical information about a person is usually viewed as sensitive and private, these key characteristics are encoded, essentially anonymizing the features being measured, to protect the patient's privacy. Furthermore, some measurements are difficult to obtain, hence the number of data instances is generally small in the medical domain. As such, for the project, we will work with a relatively small training dataset, consisting of 617 data points, whose features are all anonymized. With this, we attempted to train a machine learning model for a binary classification task, where the outcomes are either a patient has or does not have either of the three medical conditions.

Review of Existing Works

When we first started on this problem, we wanted to gather some of the good practices that others have. In particular, we looked at some of the contestants' work, including the top three contestants for this competition. From this exploration, we found that some contestants implemented deep neural networks, while others have tried more traditional classifiers, such as Catboost and XGB. We also realized that performing additional feature engineering, specifically taking pairwise ratio between features, could be useful since many indicators in medical reports are calculated by ratios between other indicators [1]. This prompted some of our feature engineering steps. Furthermore, we came across a dendrogram [5] (Figure 1) that suggests that there are high correlations between the features. This also motivated our use of hierarchical clustering on correlation for feature selection (see below).

Exploratory data analysis (EDA)

At a first glance into the dataset, we see that the features have not been given with discernible labels. This means we lack the context to do meaningful domain-knowledge based feature analysis/engineering. What we can do is to look into the feature spread, variances and co-variances. We see that the feature spaces are largely different and normalization is required to avoid feature dominance.

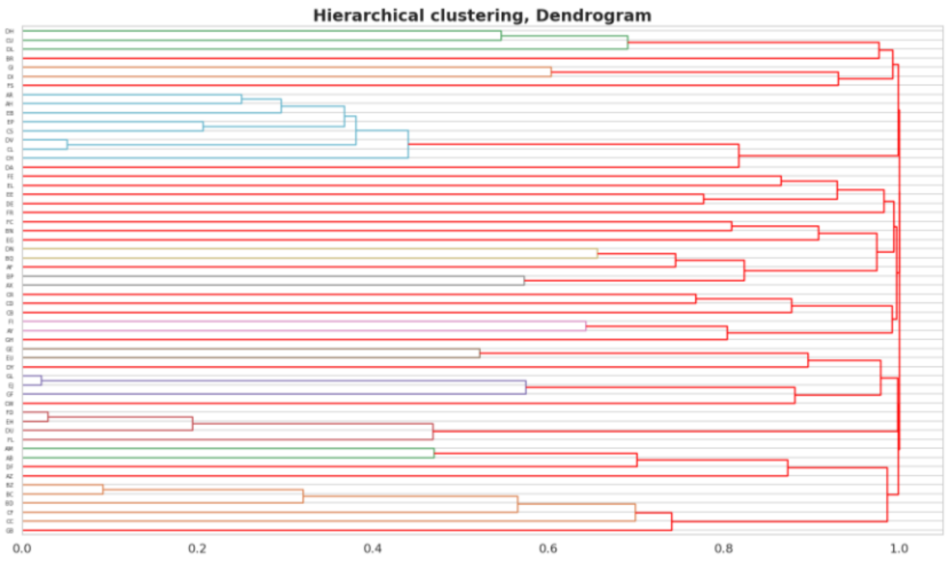


Figure 1: Dendrogram illustration of hierarchical clustering of features [5]

Some features are also of low variance, meaning a low info gain, and will likely be removed from consideration when building the model. We also see high covariance between certain features from Figure 6, meaning some can be eliminated without diminishing performance while also improving the dimensionality of the problem.

There are a few missing data points that we have to deal with properly to not suffer a skewing of results or reduction in accuracy. There is also a huge class imbalance of 108:509 for the 617 data points, which we most likely will have to address through sampling methodologies or loss balancing to avoid class dominance.

Motivations

From our review of existing works, we see that while there has been exploration of deep neural networks, majority of the participants found most success in shallow methods such as CatBoost and XGBoost. Further taking into account the runtime required to do a thorough feature search as we choose to do feature engineering, we face exponentially increasing training times. Amongst the gradient boosting algorithms CatBoost seems to boast the highest accuracy despite the longer training times. Therefore, as a happy medium, we went with the implementation of CatBoost as our base model to implement.

There are multiple methods to handling the missing data. Exploring other participants' submissions, there were usage of min/max/mean/median value or most frequent value imputation. However, there lacks a proper reasoning as to the chosen values. We chose to go with a k-nearest neighbour's (kNN) imputation method which will have a statistical backing for the chosen imputed values.

Considering the options to tackle the issue of class imbalance, we look at the most available method which is to use the balanced class weightage provided by CatBoost as well as upsampling the smaller class. The upsampling methodology chosen was Synthetic Minority Over-sampling Technique (SMOTE) [3], a simplistic approach to upsampling that does not simply duplicate existing data points. Downsampling was not explored due to the need for bagging which would greatly increase runtime while also increasing the unexpectibility of the chosen feature-set.

As mentioned, we chose to commit to a thorough feature engineering, search and selection. There were indicators from existing works that specific feature ratios greatly improved results. Coupled with the fact that we lack context to properly parse these features, this was the chosen route to pursue. We stuck with basic arithmetic, addition, subtraction, products and division, to retain a semblance of explainability while limiting the number of searches to ease training times.

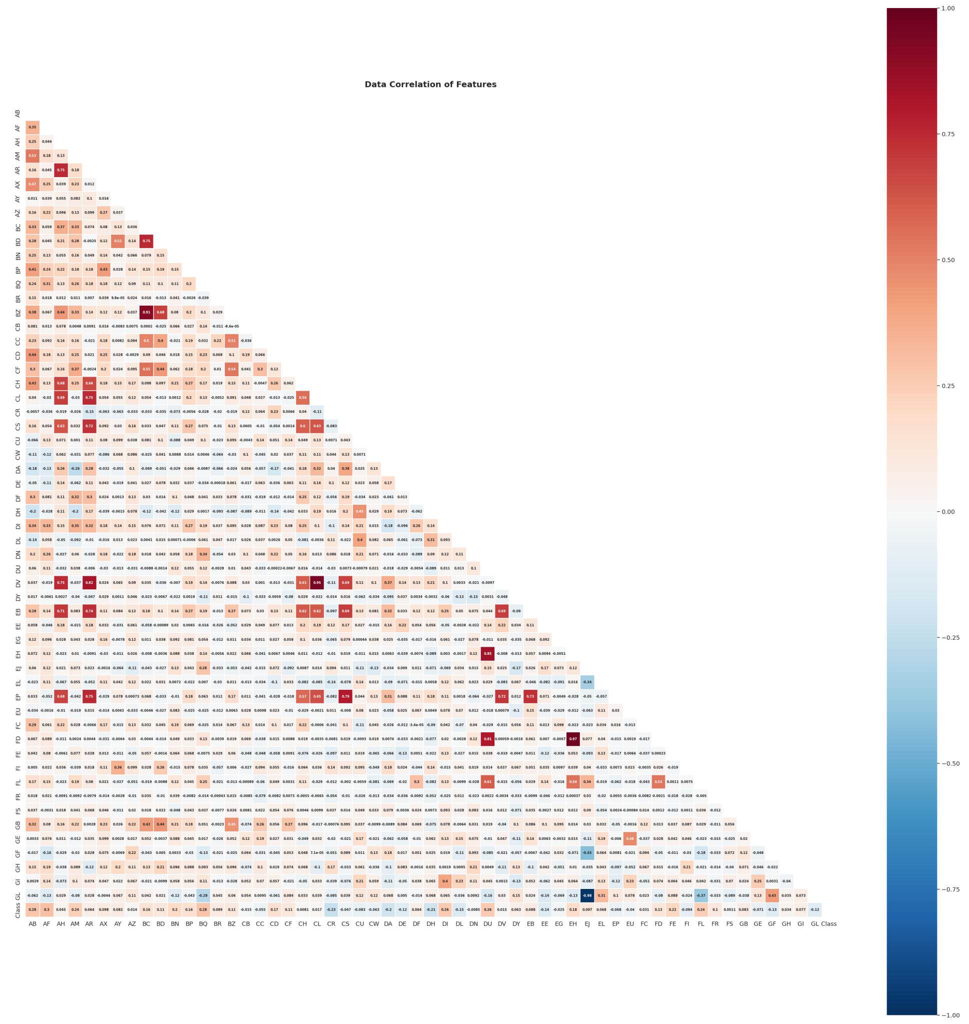


Figure 2: Covariance plot of features [5]

We then filter this large feature set with multiple steps. Looking to our EDA, we know that some features have low variance and some features have high covariance. This would also presumably be true for the engineered features. Therefore, we do a variance thresholding followed by a correlation clustering and feature-per-cluster selection. Furthermore, we know that through training, we may further uncover features that provide minimal to no improvement in the performance of the model. We then implement a recursive feature elimination using the cross-validated results.

Finally, for the CatBoost hyper-parameter tuning, we chose to not do a random or grid search as it would have been inefficient to potentially spend a large portion of our long runtime not optimizing for our task. Therefore, we opt for a Bayesian optimized search routine that will better search for these hyperparemeters due to the ability to make use of past iterations to have a probabilistic model to map the hyperparemeters to a score probability on the objective function.

Methodology

From the initial exploration of the dataset, we realized that there were missing values. To handle this, we performed a kNN imputation, setting $k = 5$. This approximates the missing value in a data instance by taking the average value of its nearest five neighbours. We implemented this using the `sklearn.impute.KNNImputer` function.

Then, we performed several feature engineering steps – ordered pairwise addition, subtract, product and division, between features on each data instance. This brings the total feature dimension to 5996. We note that feature subtraction and division are not commutative, and that the division could result in numerical instability. The former issue is trivial, whereas for the latter, we computed $\frac{X_i}{X_j+1}$ instead of $\frac{X_i}{X_j}$ for some features X_i, X_j. Since we do not have much context about the anonymized features, we wanted to create more features for us to select and use to fit the model. We outline the feature selection steps below.

Before training the model, we considered two methods to address the imbalance in our dataset. Specifically, we tried implementing SMOTE [3] and balanced loss for our Catboost. SMOTE tends to work well with tabular, and attempts to oversample the minority class through synthetically generated data. In particular, for a randomly selected minority example, we randomly choose one of its k-nearest neighbour (we set k = 5), and interpolate, randomly, a point between the neighbour and the minority example as our synthetic example. We implemented this using `imblearn.over_sampling.SMOTE`.

On the other hand, we also experimented with implementing balanced loss setting the arugment `auto_class_weights in catboost` to be `'Balanced'`. The weight CW_k of the k^{th} class is calculated to be:

$$CW_k = \frac{\max_{c=1}^{K}(\sum_{t_i=c} w_i)}{\sum_{t_i=k} w_i} \tag{1}$$

where w_i and t_i are the weight and class of the i^{th} instance, and $c = 1, 2, ..., K$ are the class index. By default, $w_i = 1$ for all i, hence CW_k is equivalently the inverse of the ratio of the number of instances in class k to the total number of data instances. This means that a minority class would have a higher class weight, and hence instances from the minority class will be given more importance by the model. In the spirit of experimentation, we decided to apply these two techniques independently and analyze their impact on the model's performance.

As the Catboost classifier has many hyperparameters, we opted for Optuna [2] to perform automated and parallelizable search for the best hyperparameter. We perform a 5-fold cross-validation on our training set for each round of hyerparameter tuning. The objective is to minimize the balanced log loss, given by,

$$\text{Balanced Log Loss} = \frac{-\frac{1}{N_0}\sum_{i=1}^{N_0}(y_{0i}\log p_{01}) - \frac{1}{N_1}\sum_{i=1}^{N_1}(y_{1i}\log p_{1i})}{2}, \tag{2}$$

where N_c refers to the number of instances in class c and y_{ci} and p_{ci} are the c^{th} entry of the one-hot label and prediction vector associated to the i^{th} data instance, respectively.

Finally, we can attempt to train the model. Instead of fitting all 5996 features, which may result in higher computational cost and a potential problem of overfitting to the training set, we decided to perform feature selection. For this, we experimented with different selection criteria:

1. **Variance thresholding**: Features with variance $\leq t$ will be eliminated, for some specified threshold t. In our implementation, we set $t = 0.16$
2. **ANOVA F-score thresholding**: Perform an ANOVA F-test and select the top p of features with the highest F-score, for some specified percentile p. In our implementation, we set $p = 0.3$.
3. **Principal Component Analysis (PCA)**: Perform PCA and retain the principal components that contribute to v of the variance of the initial dataset, for some specified ratio v. In our implementation, $v = 0.95$

The first method of variance thresholding, implemented using `sklearn.feature_selection.VarianceThreshold` aims to naively remove features that do not show much variation across the dataset. Because of how little the feature value changes with the data instances, these eliminated features likely provide little information for our prediction task. Along the same vein, the ANOVA F-test calculates an F-score for each feature, which could help us select features according to their distribution. The F-score for the t^{th} can be computed as,

$$F = \frac{\text{between-group variability}}{\text{within-group variability}}, \text{ where}$$
$$\text{between-group variability} = \sum_{i=1}^{K} \frac{n_i(\bar{x_i^{(t)}} - \bar{x^{(t)}})^2}{K-1}, \quad \text{and} \tag{3}$$
$$\text{within-group variability} = \sum_{i=1}^{C} \sum_{j=1}^{n_i} \frac{(x_{ij}^{(t)} - \bar{x_i^{(t)}})^2}{N-K},$$

where $\bar{x^{(t)}}$ represents the mean value of the t^{th} feature across all N number of data instances, $\bar{x_i^{(t)}}$ refers to the mean value of the t^{th} feature across all the data instances in class i, $x_{ij}^{(t)}$ represents the j^{th} data instance in class i, and n_i refers to the number of data instances in class i, for $i \in \{1, 2, ..., C\}$.

Although typically used as a test statistic, the F-score can provide us with information about the distribution of each feature. In particular, a higher F-score suggests that the between-group variability of this feature is much higher than its within-group variability, suggesting that the feature can discriminate two classes (or more generally, C classes) well. As such, we would retain these features to be used for fitting our predictive model.

For both variance and F-score thresholding, we also implemented an agglomeration hierarchical clustering to further reduce the feature dimension. The distance metric we used is defined to be $d(X_i, X_j) = 1 - |SC(X_i, X_j)|$, where $SC(X_i, X_j)$ is the Spearman rank-order correlations between features X_i and X_j, implemented using `scipy.stats.spearmanr`.This way, features that are highly correlated are considered to be more similar, and vice versa. Then, we use Ward's linkage for the merging of clusters. That is, at every merging step, until termination, we merge the two clusters that lead to the minimum increase in total within-cluster variance, based upon our distance metric, after merging. Using `scipy.cluster.hierarchy.fcluster`, we perform the hierarchical clustering and obtain a flat clustering by setting our threshold $t = 1$. Then, we select only one feature from each of the clusters as our selected features for model training. The idea is that, after the clustering, features within the same clusters are highly correlated and we could remove all but one of them. Also, if an initial feature and an engineered feature happens to be in the same cluster, we choose the initial feature as our cluster representative.

We also implemented PCA as a standalone feature dimensionality reduction technique. Instead of selecting the existing features, we wanted to experiment with a linear projection to extract some features that contain most of the variance of our dataset. To implement this, we performed a standardization step, obtaining features that have zero mean and unit variance. Then we performed PCA using `sklearn.decomposition.PCA` and retain the eigenvectors, with the highest eigenvalues, such that the projection of the initial dataset onto the subspace spanned by these eigenvectors retains at least 95% of the initial variance of the dataset.

After performing feature selection or extraction, we used these features to perform hyperparameter tuning using Optuna. Then, with the best set of hyperparameters, we performed a Recursive Feature Elimination with (5-fold) Cross-Validation(RFECV), implemented using `sklearn.feature_selection.RFECV`. This is a post-processing step, where the least important features are iteratively removed, and the set of remaining features will be used to train the model with tuned hyperparameters, performing a 5-fold cross-validation and noting the average balanced log loss during cross validation. Then, the set of features with the lowest average cross-validation balanced log loss will be selected as our final set of features. In other words, given a new data instance, we select its values for these features to perform inference on this model.

Experiment

In this study, four experiments were conducted to evaluate the effectiveness of different training workflows on CatBoost's performance. By comparing these experiments, we aim to identify the combination of techniques that optimize model performance and generalization to unseen data. The overview of different training workflows, illustrated in Figure 3, encompasses data imputation, feature engineering, feature normalization, feature selection, feature extraction, imbalanced classification, and hyperparameter tuning.

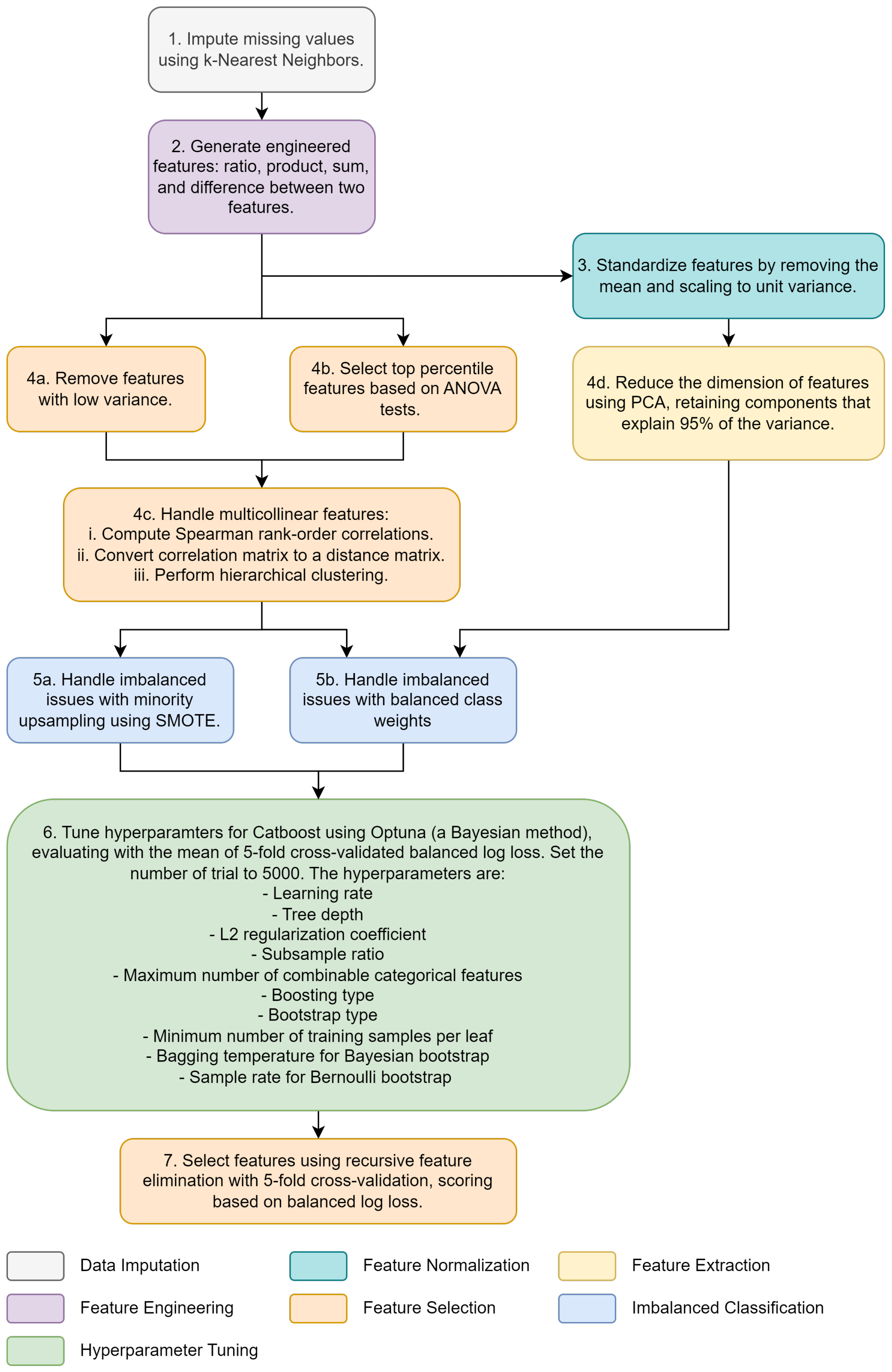

1. Impute missing values using k-Nearest Neighbors.
2. Generate engineered features: ratio, product, sum, and difference between two features.
3. Standardize features by removing the mean and scaling to unit variance.
4a. Remove features with low variance.
4b. Select top percentile features based on ANOVA tests.
4d. Reduce the dimension of features using PCA, retaining components that explain 95% of the variance.
4c. Handle multicollinear features: i. Compute Spearman rank-order correlations. ii. Convert correlation matrix to a distance matrix. iii. Perform hierarchical clustering.
5a. Handle imbalanced issues with minority upsampling using SMOTE.
5b. Handle imbalanced issues with balanced class weights
6. Tune hyperparamters for Catboost using Optuna (a Bayesian method), evaluating with the mean of 5-fold cross-validated balanced log loss. Set the number of trial to 5000. The hyperparameters are:
- Learning rate
- Tree depth
- L2 regularization coefficient
- Subsample ratio
- Maximum number of combinable categorical features
- Boosting type
- Bootstrap type
- Minimum number of training samples per leaf
- Bagging temperature for Bayesian bootstrap
- Sample rate for Bernoulli bootstrap
7. Select features using recursive feature elimination with 5-fold cross-validation, scoring based on balanced log loss.
Data Imputation
Feature Normalization
Feature Extraction
Feature Engineering
Feature Selection
Imbalanced Classification
Hyperparameter Tuning

Figure 3: Sequential Training Workflow for CatBoost Model Optimization.

CatBoost's Hyperparameters were tuned within defined ranges, seeking to balance model complexity and learning efficiency. The hyperparameter search ranges were set as follows:

- Learning rate: 0.001 to 0.1, on a logarithmic scale
- Tree depth: 3 to 10

- L2 regularization coefficient: 1 to 10
- Subsample ratio: 0.01 to 1.0, on a logarithmic scale
- Maximum number of combinable categorical features: 0 to 8
- Boosting type: "Ordered" or "Plain"
- Bootstrap type: "Bayesian", "Bernoulli", or "MVS"
- Minimum number of training samples per leaf: 1 to 100
- Bagging temperature for Bayesian bootstrap: 0 to 10
- Sample rate for Bernoulli bootstrap: 0.1 to 1, on a logarithmic scale

A hierarchical clustering method is employed to tackle multicollinearity among features, a crucial step in the training workflow; this process is outlined in the pseudocode below, titled *Multicollinearity Removal Using Hierarchical Clustering*, where highly correlated features are grouped based on a distance matrix, and only one representative from each group is retained.

Algorithm 1 Multicollinearity Removal Using Hierarchical Clustering

```
1: Compute Spearman rank-order correlations among features, $\mathbf{X}$
2: Make correlation matrix symmetric
3: Convert correlations to distances: distance $= 1 - |\text{correlation}|$
4: Perform hierarchical clustering using Ward's method
5: Form clusters using a distance threshold of 1
6: Select one feature from each cluster
7: Update $\mathbf{X}$ to contain only selected features
```

Table 1 shows the balanced log loss for training and testing across four experiments. The training loss is the average log loss from 5-fold cross-validation, including a 95% confidence interval for the training data, while the testing loss is calculated automatically on the testing data in Kaggle. A greater log loss value indicates improved model performance.

Experiment	**1**	**2**	**3**	**4**
Training loss	0.2444 (± 0.0314)	0.2496 (± 0.0212)	0.2471 (± 0.0337)	0.3802 (± 0.0530)
Testing loss	0.4257	0.4839	0.4265	0.5816

Table 1: Summary of training and testing losses, rounded to four decimal places.

Experiment	**1**	**2**	**3**	**4**
Learning rate	0.0323	0.0209	0.0404	0.0231
Tree depth	3	4	3	3
L2 regularization coefficient	9.8022	8.0678	5.1202	7.7124
Subsample ratio	0.8436	0.2806	0.0804	0.3442
Maximum number of combinable categorical features	3	8	1	0
Boosting type	Plain	Plain	Plain	Plain
Bootstrap type	Bernoulli	Bernoulli	Bayesian	Bayesian
Minimum number of training samples per leaf	84	64	10	75
Bagging temperature for Bayesian bootstrap	-	-	8.3845	8.0588
Sample rate for Bernoulli bootstrap	0.3338	0.5741	-	-

Table 2: Optimal CatBoost hyperparameters for each experiment, rounded to four decimal places.

Experiment 1

Experiment 1 was designed using steps 1, 2, 4a, 4c, 5b, 6, and 7 from the training workflow. Experiment 1 employed the default KNN imputation for missing values. A total of 5,996 features, including 5,940 engineered features, were created. Subsequently, the feature set was reduced to 5,403

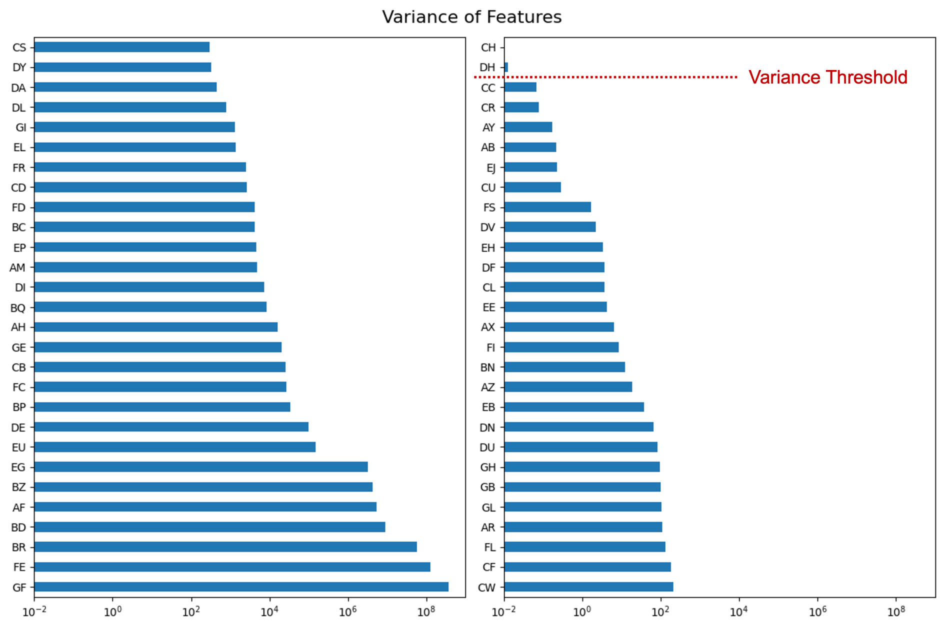


Figure 4: Bar chart of feature variance. The features are sorted in descending order from the bottom. A red dotted line indicates the threshold set for variance thresholding.

after removing 593 features with low variance, as determined by the variance threshold depicted in Figure 4. Following this, the number of features was further reduced to 106 after the removal of 5,297 highly collinear features, identified through Spearman rank-order correlation and hierarchical clustering. Class imbalance issues were addressed by using balanced class weights. Then, the best hyperparameters were found using the Optuna module and the 106 features, as outlined in Table 2. Finally, the model's performance was further improved by using only 65 features identified by the recursive feature elimination method, after applying the optimal hyperparameters.

As indicated in Table 1, Experiment 1 resulted in a training loss of 0.2444 with a 95% confidence interval of 0.0314, while the testing loss was observed to be 0.4598, signifying a gap of approximately 0.2154 between training loss and testing loss.

Experiment 2

Experiment 2 followed steps 1, 2, 4a, 4c, 5a, 6, and 7 from the training workflow, which is similar to Experiment 1 but differed in the class imbalance approach by using SMOTE (step 5a). Then, the best hyperparameters were found using the Optuna module and the 106 features, as outlined in Table 2. Finally, the model's performance was further improved by using only 67 features identified by the recursive feature elimination method, after applying the optimal hyperparameters.

As indicated in Table 1, Experiment 2 showed a training loss of 0.2496 with a 95% confidence interval of 0.0212, and a testing loss of 0.4839, also showing a gap of approximately 0.2343 between training and testing loss. In summary, Experiment 2 performed worse than Experiment 1 in both training and testing loss and showed a larger gap between training and testing loss.

Experiment 3

Experiment 3 followed steps 1, 2, 4b, 4c, 5b, 6, and 7 from the training workflow, which is similar to Experiment 1 but differed in the selection of features through ANOVA tests (step 4b). A total of 5,996 features, including 5,940 engineered features, were created. Then, the top 30th percentile of features(1,799 features) was selected based on ANOVA tests, as shown in Figure 5. Next, only 50 features remained after 1,749 highly collinear features were removed using Spearman rank-order correlation and hierarchical clustering. Class imbalance issues were addressed using balanced class weights, the same as in Experiment 1. Then, the best hyperparameters were found using the Optuna

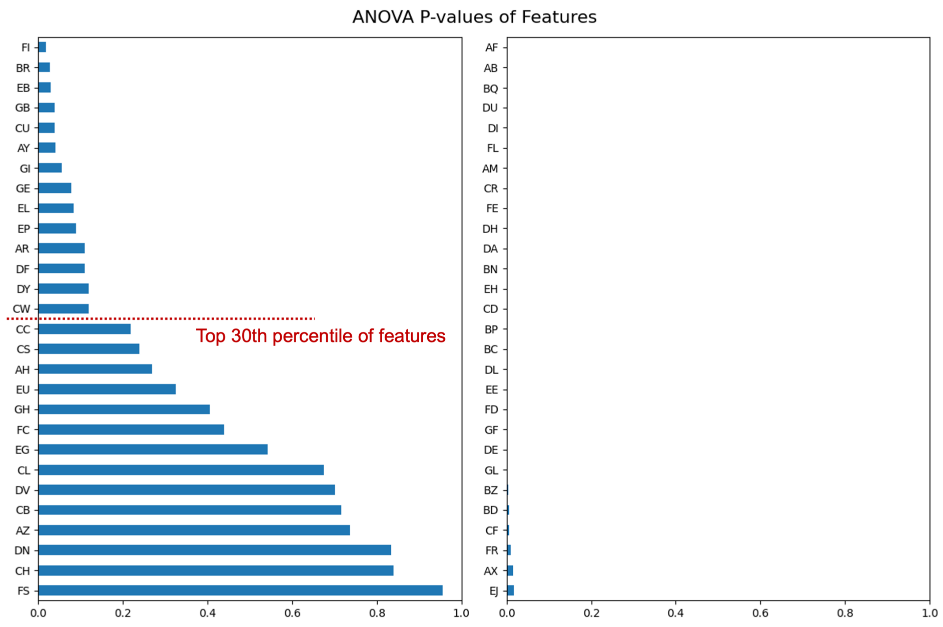


Figure 5: Bar chart of ANOVA P-values for features. The features are sorted in descending order from the bottom. A red dotted line indicates the cutoff for the top 30th percentile of features.

module for the 50 features, as outlined in Table 2. However, the model's performance was not improved by the recursive feature elimination method, and the 50 features were still retained.

As indicated in Table 1, Experiment 3 showed a training loss of 0.2471 with a 95% confidence interval of 0.0337, and a testing loss of 0.4265, also showing a gap of approximately 0.1794 between training and testing loss. In summary, Experiment 3 performed worse than Experiment 1 in both training and testing loss; however, Experiment 3 showed a smallest gap between training and testing loss among all the experiments.

Experiment 4

In Experiment 4, steps 1, 2, 3, 4d, 5b, 6, and 7 were implemented, including the standardization of features (step 3) and dimensionality reduction through PCA (step 4d). This experiment mainly aimed to investigate the effect of PCA on model improvement. Surprisingly, PCA was able to reduce the dimensions of the original dataset from 5,996 features to 60 principal components that retained 95% of the variance from the original dataset. Class imbalance issues were addressed using balanced class weights, the same as in Experiment 1. Then, the best hyperparameters were found using the Optuna module and the 60 principal components, as outlined in Table 2. Finally, the model's performance was further improved by using only 17 selected principal components out of the 60 principal components identified by the recursive feature elimination method, after applying the optimal hyperparameters.

As indicated in Table 1, Experiment 4 showed the worst training loss and testing loss among all the experiments. Its training loss is 0.3802 with a 95% confidence interval of 0.0530, and its testing loss is 0.5816, also showing a gap of approximately 0.2014 between training and testing loss.

Discussion

Handling Class Imbalance

Comparing Experiment 1 and 2, it is apparent that balanced class weights yield better results than the SMOTE up-sampling method. This may be attributed to the intrinsic handling of class imbalance by CatBoost, which internally adjusts the loss function to give more weight to the minority class. Balanced class weights preserve the original data distribution without introducing synthetic data, as SMOTE does. Synthetic data generation by SMOTE can sometimes create noisy samples if the minority class data points are outliers or noise, which might confuse the model rather than helping it learn the underlying patterns.

Feature Selection Techniques

Experiment 1 and Experiment 3 offer insights into feature selection methods: features selected with variance thresholding versus those selected from the top 30th percentile based on ANOVA tests. The superiority of variance thresholding in Experiment 1 can be ascribed to its preservation of more relevant features. Variance thresholding is a non-parametric method; it removes features with low variance on the premise that they contain less information. In contrast, selecting the top 30th percentile of features based on ANOVA tests in Experiment 3 leads to the removal of a substantial number of features, potentially including those with predictive power. This selection process can cause significant information loss and degrade the model's ability to capture the complexity of the data. Hence, while ANOVA tests are useful in identifying features with statistically significant differences, variance thresholding is more effective in retaining valuable information for the model.

Impact of PCA on Feature Extraction

The performance deterioration using PCA in Experiment 4 can be understood by recognizing PCA as a linear dimensionality reduction technique. PCA reduces the feature space by transforming the original variables into a smaller set of uncorrelated components, which are linear combinations of the original features. However, the assumption that principal components with the highest variance always carry the most informative signals for all types of analysis is not universally valid. The removal of low-variance components might discard nuanced, yet crucial, information that could be predictive for the outcome. PCA also assumes linearity, and it may not capture complex relationships that might be better modeled with the original features or through non-linear dimensionality reduction methods. Therefore, PCA does not guarantee an improvement in model performance and may, in fact, degrade it if critical information is lost in the lower variance components that are discarded.

Conclusion

The insights garnered from the series of experiments conducted in this study illuminate the path toward effective model optimization. Among the methodologies tested, the techniques applied in Experiment 1 have proven to be superior. This experiment adeptly selected features via variance thresholding, which judiciously retained features with potential predictive value. It also effectively addressed the issue of multicollinearity through the use of Spearman correlation and hierarchical clustering, a crucial step to ensure that the CatBoost model was not hindered by redundant information. Moreover, the hyperparameter tuning process was meticulous, leading to a well-optimized CatBoost configuration.

Despite these robust methodologies, it must be noted that the test results from Experiment 1 did not secure a place within the top 10% of the competition standings. This outcome may suggest that the dataset and problem at hand could benefit from a more sophisticated modeling approach, possibly through the implementation of neural networks. Neural networks are known for their ability to capture complex, non-linear relationships within data, which traditional methods like CatBoost might overlook. Transitioning to such advanced models could potentially unveil deeper insights and lead to improved predictive performance, potentially elevating the results to compete within the top tier of the competition.

References

[1] *3rd Place Solution for the "ICR - Identifying Age-Related Conditions" Competition*. Kaggle. Accessed: 2024-04-21. URL: `https://www.kaggle.com/competitions/icr-identify-age-related-conditions/discussion/430978`.

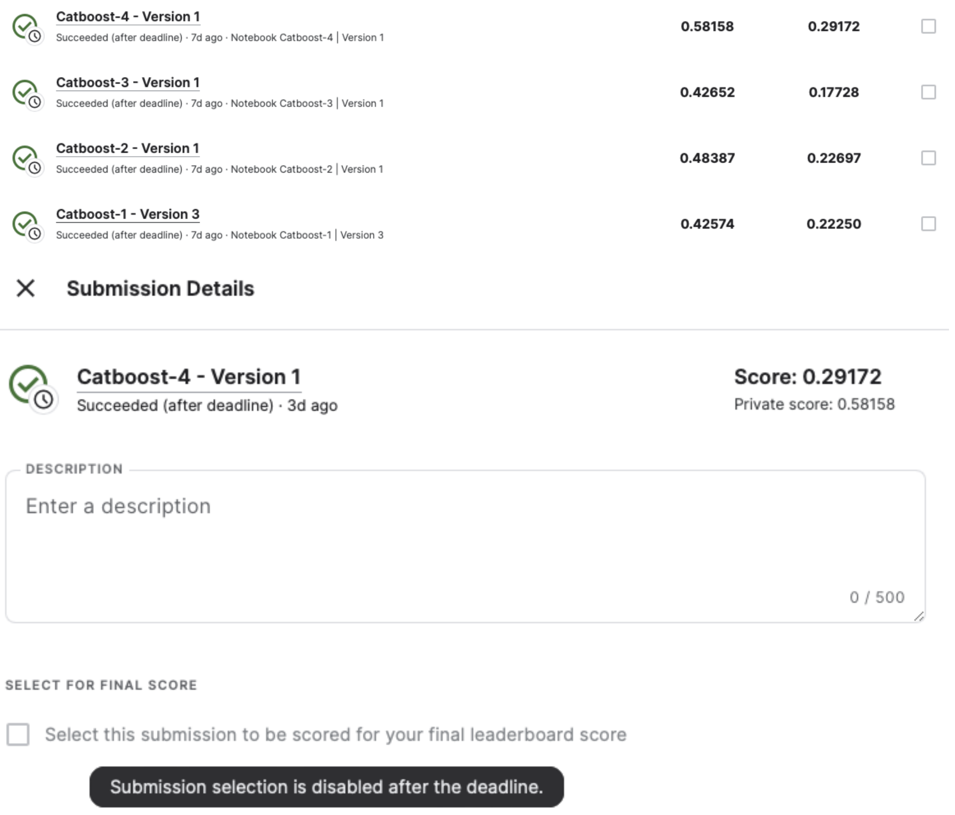
[2] Takuya Akiba et al. *Optuna: A Next-generation Hyperparameter Optimization Framework*. 2019. arXiv: `1907.10902 [cs.LG]`.

[3] N. V. Chawla et al. “SMOTE: Synthetic Minority Over-sampling Technique”. In: *Journal of Artificial Intelligence Research* 16 (June 2002), pp. 321–357. ISSN: 1076-9757. DOI: `10.1613/jair.953`. URL: `http://dx.doi.org/10.1613/jair.953`.

[4] *ICR - Identifying Age-Related Conditions*. Kaggle. Accessed: 2024-04-25. URL: `https://www.kaggle.com/competitions/icr-identify-age-related-conditions/overview`.

[5] *ICR IARC EDA| Ensemble and Stacking baseline*. Kaggle. Accessed: 2024-04-21. URL: `https://www.kaggle.com/code/tetsutani/icr-iarc-eda-ensemble-and-stacking-baseline`.

Appendix



Catboost-4 - Version 1 Succeeded (after deadline) · 7d ago · Notebook Catboost-4 | Version 1	0.58158	0.29172	☐
Catboost-3 - Version 1 Succeeded (after deadline) · 7d ago · Notebook Catboost-3 | Version 1	0.42652	0.17728	☐
Catboost-2 - Version 1 Succeeded (after deadline) · 7d ago · Notebook Catboost-2 | Version 1	0.48387	0.22697	☐
Catboost-1 - Version 3 Succeeded (after deadline) · 7d ago · Notebook Catboost-1 | Version 3	0.42574	0.22250	☐

Submission Details

Catboost-4 - Version 1
Succeeded (after deadline) · 3d ago

Score: 0.29172
Private score: 0.58158

DESCRIPTION
Enter a description
0 / 500

SELECT FOR FINAL SCORE
☐ Select this submission to be scored for your final leaderboard score

Submission selection is disabled after the deadline.

Figure 6: Submission results to Kaggle. Catboost-1 refers to experiment 1, Catboost-2 refers to experiment 2, and so on.