Predicting Binary Classes in Molecular Data Using Machine Learning

Mohammad Sharzehei

1. OVERVIEW

This project presents an end-to-end machine learning pipeline for binary classification of molecular data, specifically targeting the prediction of linear B-cell epitopes based on protein-derived features. It outlines the steps taken during exploratory data analysis (EDA), preprocessing, feature reduction, class balancing, and modelling, as well as the rationale behind each decision. After testing various models, a tuned Random Forest classifier was selected based on its performance using cross-validated AUC, F1-score, and MCC. The entire process is encapsulated in a reproducible and modular Python pipeline, suitable for real-world deployment or further biological validation.

2. BACKGROUND

Linear B-cell epitopes are small segments of proteins that are recognized by components of the immune system. Identifying these fragments is a crucial early step in designing vaccines, diagnostic tools, and targeted therapies for infectious diseases, allergic conditions, and certain types of cancer. Traditional methods for epitope discovery are time-consuming and resource-intensive, which is why computational approaches have become essential. Over the past decades, such methods have significantly improved the efficiency of identifying promising epitope candidates for laboratory validation.

3. DATASET

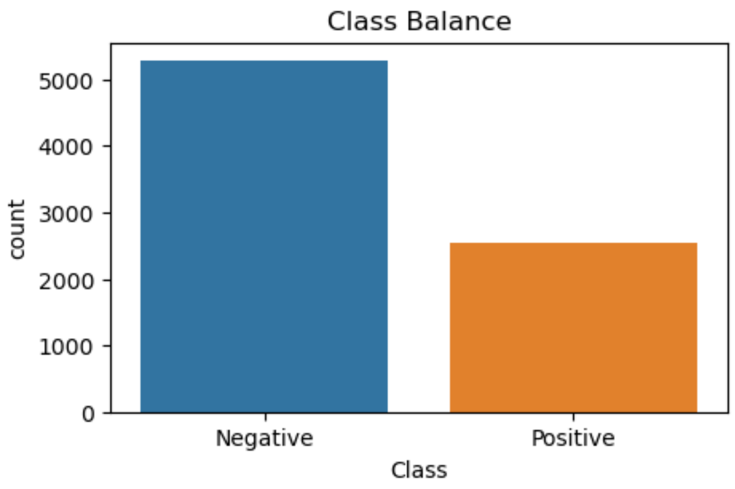
The dataset used in this project was derived from multiple online biological databases, primarily the Immune Epitope Database (IEDB) and NCBI Protein, and processed using feature extraction tools developed by researchers at Aston University.

4. EDA AND DATA PREPROCESSING

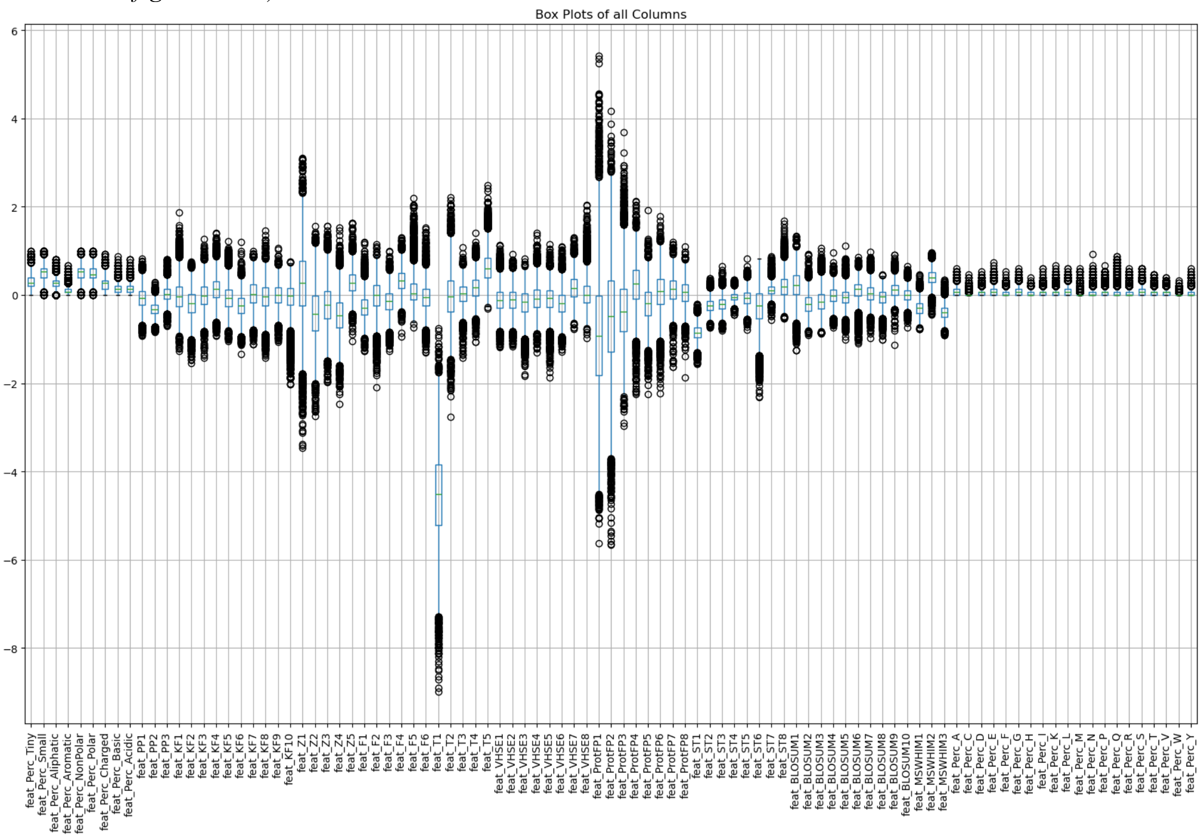
4.1 Exploratory Data Analysis

Starting with EDA, we examined the dataset to understand its structure, identify patterns, and detect issues such as missing values and outliers. The following steps were taken for EDA:

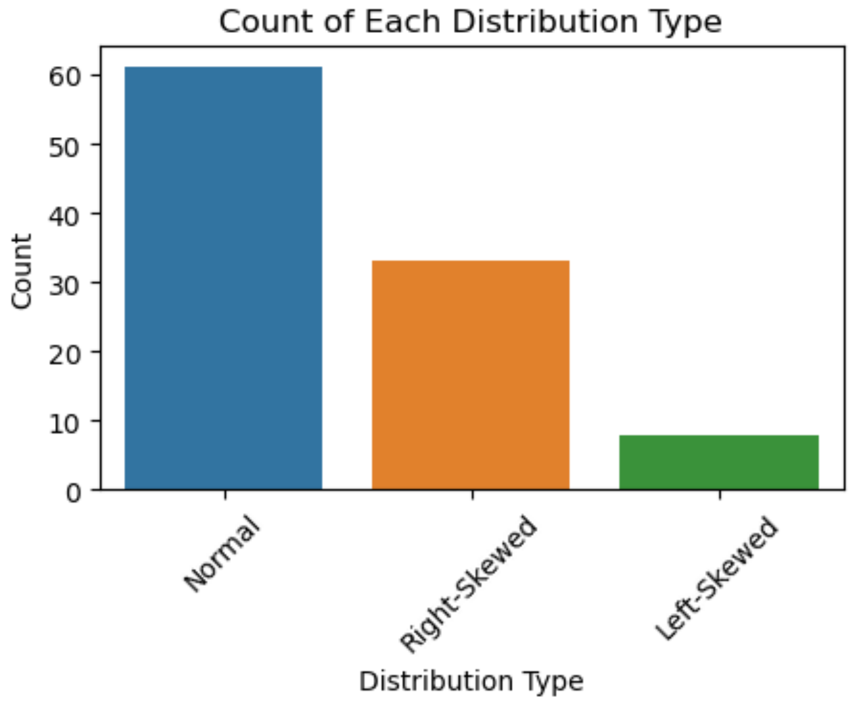
- ***Size of DataFrame**: The provided dataset,' df.csv', contains 7832 entries and 103 columns.*
- ***Data types**: The columns include 97 of type 'float64', 5 of type 'int64', and 1 of type 'object'.*
- ***Class Distribution:** The target variable 'Class' is imbalanced, with approximately 67.5% negative and 32.5% positive samples.*



- ***Missing Values:*** *No missing values were found in the dataset.*
- ***Identify outliers****: Many columns in the DataFrame have a large number of outliers (some of which are shown in the figure below).*



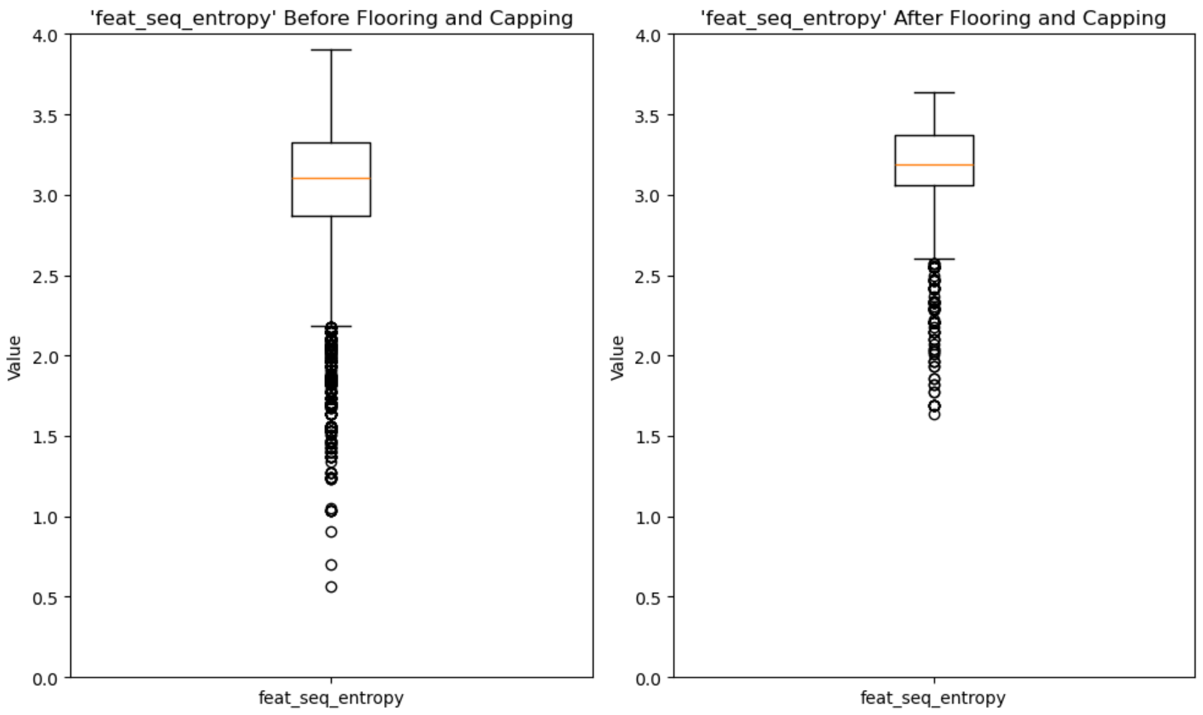
- ***Scale Check****: Some features have values ranging from 0 to over 2500, while others have values between -10 and 10.*
- ***Feature Correlation:*** *A correlation matrix was plotted to identify relationships between features. Some features showed strong positive or negative correlations.*
- ***Distribution Analysis:*** *Most features followed a normal distribution, while some have right-skewed or left-skewed distributions.*



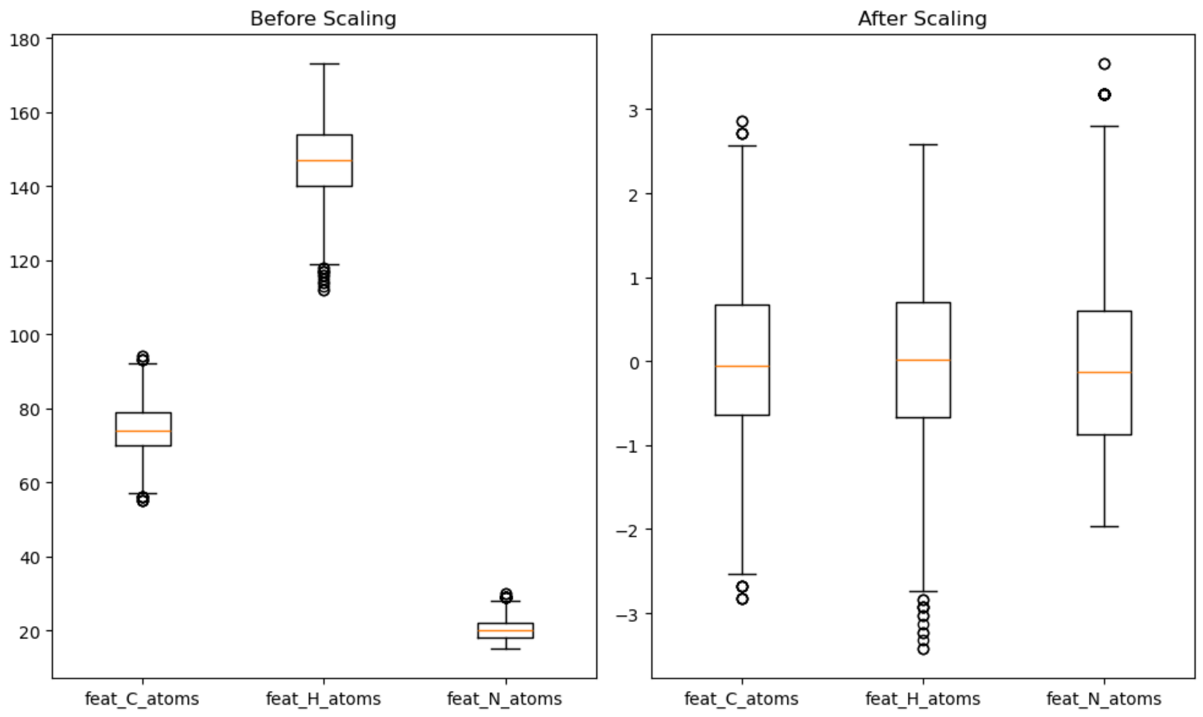
4.2 Data Preprocessing

Data preprocessing involved cleaning and transforming the data to make it suitable for analysis. Key steps included:

- ***Split the Data**: The data was divided into 70% for training and 30% for testing.*
- ***Outlier Detection and Treatment**: Many features had significant outliers, which were addressed through quantile-based flooring and capping. This method effectively removed outliers and minimized their impact on the dataset. It was chosen because it is less affected by extreme values compared to methods that rely on mean and standard deviation, providing more stable results. (The figure below shows boxplots for first column before and after applying flooring and capping.)*



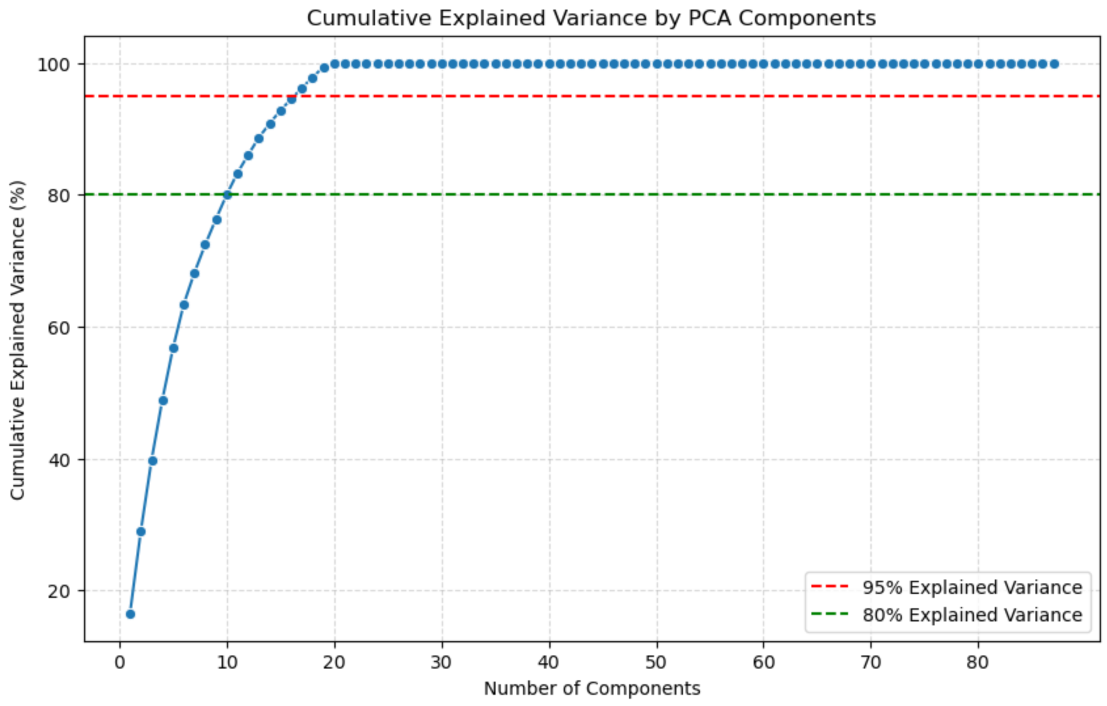
- ***Handling Missing Values**: Although there is no missing data, this step is included to ensure that preprocessing is complete if we use a new DataFrame later.*
- ***Data Normalisation**: Features were scaled using 'StandardScaler' to ensure they were on a similar scale.*



5. FEATURE REDUCTION

Feature reduction was an important part of preprocessing. It helped improve model performance and made computations faster and less complex. The methods used were:

- ***Correlation Analysis****: Examined the correlation matrix to identify highly correlated features and removed redundant ones to prevent multicollinearity. However, this step was not necessary because we used PCA in later steps.*
    - *The highly correlated features (15 features) were removed in this step.*
- ***Feature Selection:*** *Features with low variance were removed using 'VarianceThreshold'.*
    - *There were no constant features in the training data.*
- ***Removing Duplicates****: Removing duplicate data helps in maintain a high-quality dataset, leading to better, more reliable machine learning models.*
    - *Identified and removed 228 duplicate rows in the training data.*
- ***Principal Component Analysis (PCA)****: Applied PCA to reduce the dimensionality of the data while retaining 80% of the variance. This step helped reduce the feature space and improve model efficiency. PCA was chosen because it effectively reduces the number of features by transforming them into uncorrelated variables that capture maximum variance, handles multicollinearity, filters out noise, retains important information, is versatile for any algorithm, and improves computational efficiency for large datasets. (The figure below shows that 10 and 17 features were needed to retain 80% and 95% of the variance, respectively.)*



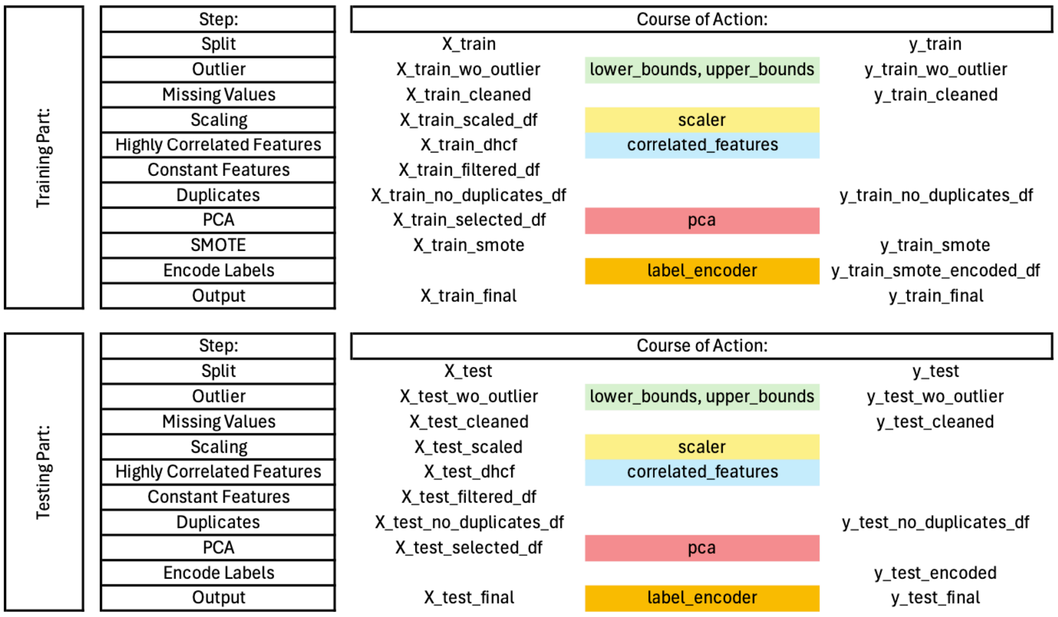
- ***Addressing Class Imbalance:*** *Synthetic Minority Over-sampling Technique (SMOTE) was applied to balance the class distribution. SMOTE was chosen because it reduces overfitting by generating synthetic samples, maintains all existing samples while balancing classes, and is straightforward to apply without requiring cost parameter tuning.*

Class Distribution	*Before SMOTE*	*After SMOTE*
Negative	*2344*	*2344*
Positive	*1102*	*2344*

- ***Encoding categorical label:*** *The categorical label 'Class' was encoded into a numerical label using scikit-learn's 'LabelEncoder'.*

Class	*Encoded Value*
Negative	*0*
Positive	*1*

To help clarify the process, I've shown the preprocessing steps for both the training and test data at each stage. Rather than processing the training data first and then the test data separately, I illustrate how each preprocessing step affects both datasets together. For example, when scaling the data, I first determine the scaling parameters from the training data and then apply those same parameters to both the training and test data. This ensures that both sets are treated the same way. The diagram below shows the steps taken for preprocessing both the training and test data.



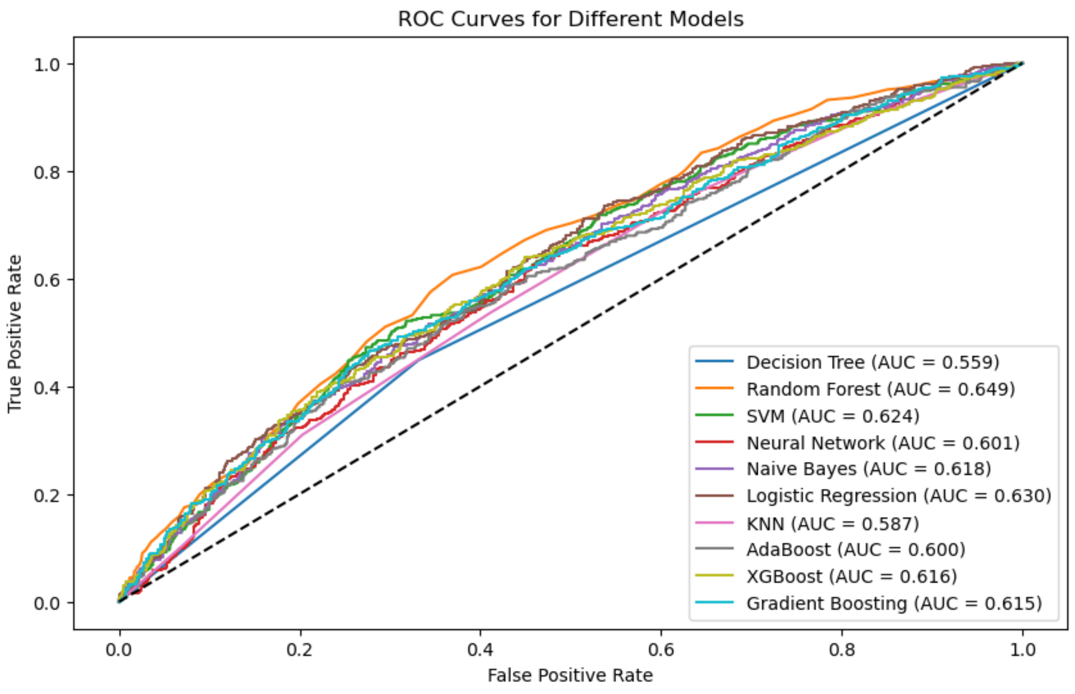
Training Part:	Step:	Course of Action:		
	Split	X_train		y_train
	Outlier	X_train_wo_outlier	lower_bounds, upper_bounds	y_train_wo_outlier
	Missing Values	X_train_cleaned		y_train_cleaned
	Scaling	X_train_scaled_df	scaler	
	Highly Correlated Features	X_train_dhcf	correlated_features	
	Constant Features	X_train_filtered_df		
	Duplicates	X_train_no_duplicates_df		y_train_no_duplicates_df
	PCA	X_train_selected_df	pca	
	SMOTE	X_train_smote		y_train_smote
	Encode Labels		label_encoder	y_train_smote_encoded_df
	Output	X_train_final		y_train_final

Testing Part:	Step:	Course of Action:		
	Split	X_test		y_test
	Outlier	X_test_wo_outlier	lower_bounds, upper_bounds	y_test_wo_outlier
	Missing Values	X_test_cleaned		y_test_cleaned
	Scaling	X_test_scaled	scaler	
	Highly Correlated Features	X_test_dhcf	correlated_features	
	Constant Features	X_test_filtered_df		
	Duplicates	X_test_no_duplicates_df		y_test_no_duplicates_df
	PCA	X_test_selected_df	pca	
	Encode Labels			y_test_encoded
	Output	X_test_final	label_encoder	y_test_final

6. MODELLING AND ASSESSMENT

6.1 Initial Model Evaluation

Multiple machine learning models were trained on the training data and evaluated on the test data using scikit-learn with default parameters. The best-performing model was identified based on the Area Under the Receiver Operating Characteristic Curve (AUC). The models evaluated included Decision Tree, Random Forest, SVM, Neural Network, Naive Bayes, Logistic Regression, K-Nearest Neighbors (KNN), AdaBoost, XGBoost, and Gradient Boosting. The figure below shows evaluation results:



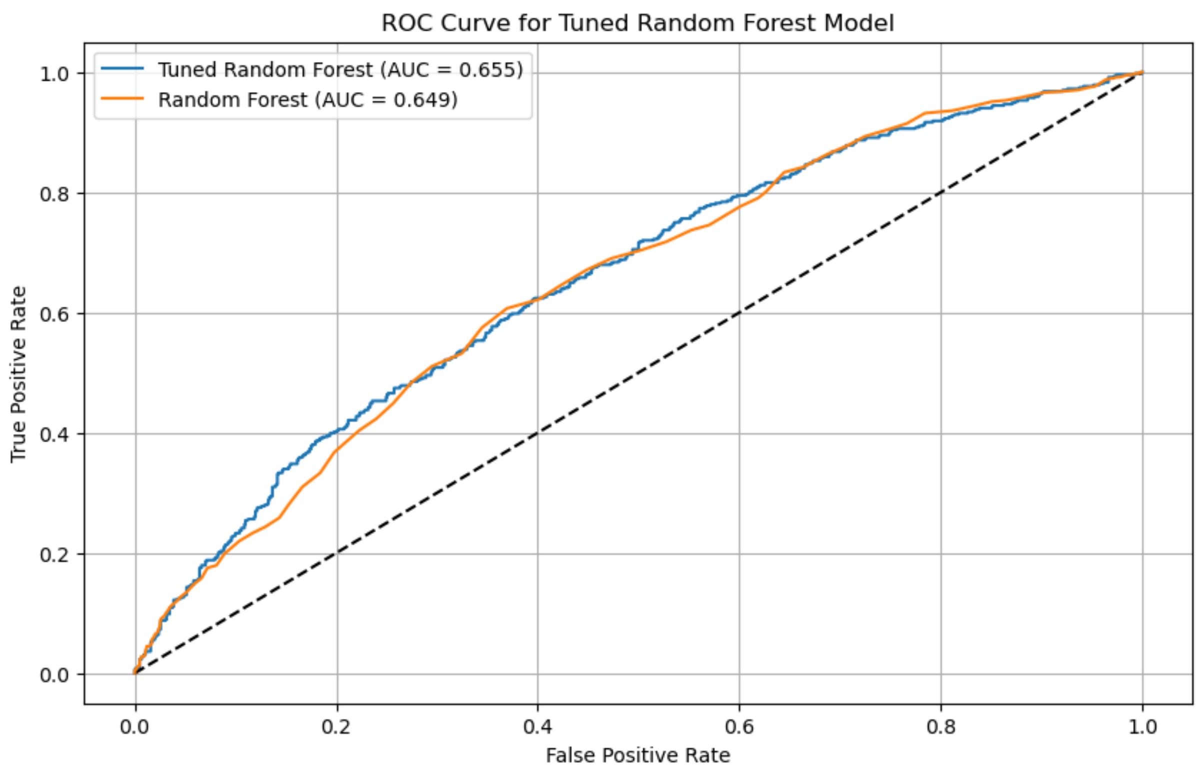
Based on these results, the Random Forest model demonstrated the best performance with an AUC of 0.649. Therefore, we proceeded with hyperparameter tuning for the Random Forest model to further enhance its performance.

6.2 Hyperparameter Tuning of Random Forest

Hyperparameter tuning was performed using RandomizedSearchCV with a defined search space for the Random Forest model. The search space included the following parameters:

Hyperparameter	*Value*
'n_estimators'	*585*
'max_features'	*'sqrt'*
'max_depth'	*None*
'min_samples_split'	*3*
'min_samples_leaf'	*1*

Using these parameters, the tuned Random Forest model achieved an improved AUC of 0.655 on the test set. The ROC curves for the Random Forest model, as well as for the tuned Random Forest model, are shown below:



The tuned Random Forest model outperforms the base Random Forest model, as indicated by the higher AUC value and the ROC curve.

***Note**: The F1 score and Matthews correlation coefficient (MCC) were also checked in the notebook to provide additional evaluation metrics for all models.*

- ***F1 Score**: Harmonic mean of precision and recall to balance the two metrics.*
- ***MCC**: Offers a balanced measure considering all parts of the confusion matrix.*
- ***ROC and AUC**: Provide a comprehensive view of the trade-offs between true positive rate and false positive rate across different thresholds.*

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

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

$$\text{MCC} = \frac{(\text{TP} \times \text{TN}) - (\text{FP} \times \text{FN})}{\sqrt{(\text{TP}+\text{FP})(\text{TP}+\text{FN})(\text{TN}+\text{FP})(\text{TN}+\text{FN})}}$$

True Positive Rate (TPR):

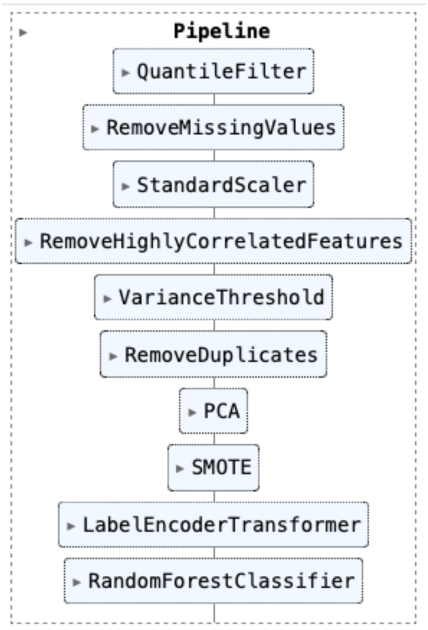
$$\text{TPR} = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

False Positive Rate (FPR):

$$\text{FPR} = \frac{\text{FP}}{\text{FP}+\text{TN}}$$

6.3 Creating Pipeline

In this stage, preprocessing and modelling choices were incorporated into a pipeline. This approach ensured that all steps were applied consistently to both training and test dataset. To address specific preprocessing needs, custom transformers were created using 'BaseEstimator' and 'TransformerMixin' from 'sklearn.base'. To utilize this approach, certain actions on the features and the target were performed simultaneously at some steps before proceeding to the next step. The pipeline was implemented with this in mind. For example, when a row identified as duplicate data was removed, the corresponding label was also deleted. A tuned Random Forest model was used for the final evaluation based on the parameters we found in the Hyperparameter Tuning phase. The results of our pipeline were the same as in the previous part, which is expected since we did not change any steps, we just organized them into a pipeline.



7. CONCLUSIONS AND DISCUSSION

This project demonstrated the development of a complete machine learning pipeline for binary classification of molecular data, with a focus on predicting linear B-cell epitopes. The process involved data cleaning, outlier handling, class balancing using SMOTE, dimensionality reduction via PCA, and model evaluation using several classification algorithms. Among the models tested, a tuned Random Forest classifier achieved the best performance with an AUC of 0.655, demonstrating balanced predictive ability as confirmed by F1-score and Matthews Correlation Coefficient (MCC). All preprocessing and modeling steps were encapsulated into a modular and reproducible pipeline using scikit-learn. Overall, this project provided hands-on experience in building robust ML pipelines for biological data and highlighted the importance of proper preprocessing and model validation in real-world applications.

REFERENCES

All relevant references and supporting materials are cited within the notebook alongside the corresponding analysis steps.