Machine Learning Approaches for Early Detection of Diabetes: A Comparative Study and Novel Solution on the PIMA Indians Diabetes Dataset

Hai Nam LE

1 PIMA Indians Diabetes Dataset Classification Using Machine Learning

1.1 Dataset Summary

The **PIMA Indians Diabetes dataset** used in this study is a standard dataset sourced from the Kaggle repository, originally provided by the National Institute of Diabetes and Digestive and Kidney Diseases. This dataset contains medical data of **768 female patients**, with **268 positive cases** for diabetes and **500 negative cases**, making the dataset moderately imbalanced (about 35% positive class). The dataset includes **8 independent features**:

- Number of pregnancies
- Plasma glucose concentration
- Diastolic blood pressure
- Triceps skinfold thickness
- Serum insulin levels
- Body mass index (BMI)
- Diabetes pedigree function
- Age

The primary challenge of the dataset is the presence of missing values and outliers. Some features, such as insulin and skin thickness, have biologically implausible zero values that represent missing data. The dataset also includes outliers in multiple columns, which were addressed during preprocessing.

1.2 Experiment Protocol

The paper outlines the use of **machine learning algorithms** to classify diabetes patients based on the PIMA dataset. The following steps were taken:

- **Data Preprocessing**: Missing values were treated using median imputation, and outliers were handled using the interquartile range (IQR) method. The class imbalance was managed using **Synthetic Minority Over-sampling Technique (SMOTE)**.
- **Models Used**: The study implemented several machine learning models, including:
  - **Decision Tree (DT)**
  - **Random Forest (RF)**
  - **Support Vector Machine (SVM)**
  - **Stacking Ensemble**, which combines multiple models
- **Evaluation Protocol**: Two evaluation methods were used:
  - **Train-Test Split**: 70% of the data was used for training, and 30% was used for testing.
  - **Cross-Validation (CV)**: A **5-fold cross-validation** was applied to ensure generalizability.

The **stacking ensemble method** combined multiple classifiers at the base level and used **logistic regression** as the meta-learner. The models were evaluated based on accuracy, precision, recall, and F1-score.

1.3 The Research Article Results

The study presented results from both the train-test split and cross-validation methods. The key findings for the **PIMA Indians Diabetes dataset** are as follows:

Protocol	**Algorithm**	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Train-Test	DT	65.08	0.65	0.65	0.65
Train-Test	RF	79.33	0.80	0.79	0.79
Train-Test	SVM	69.03	0.69	0.69	0.69
Train-Test	Stacking Ensemble	75.03	0.75	0.75	0.75
Cross-Validation	DT	68.31	0.65	0.68	0.67
Cross-Validation	RF	76.81	0.77	0.77	0.78
Cross-Validation	SVM	68.61	0.68	0.70	0.69
Cross-Validation	Stacking Ensemble	77.10	0.68	0.70	0.69

Table 1: Performance of Machine Learning Algorithms on PIMA Dataset from the research article

1.4 Conclusion

This study demonstrated the effectiveness of machine learning models in predicting diabetes using the PIMA Indians Diabetes dataset. The Random Forest model achieved the highest performance on the train-test split, while the stacking ensemble provided slightly better results in cross-validation. The combination of various preprocessing techniques, including imputation and SMOTE, helped improve the model's ability to handle missing data and class imbalance.

2 Reproduction of Results

2.1 Experimental Procedure

- **Step 1: Fine Tuning**

  To reproduce the results close to the research article, fine-tuning is required. This involves listing out the top possible cases for the model by testing different hyperparameters. Afterward, a proper set of hyperparameters that yields evaluation metrics similar to those in the research article will be selected. Fine-tuning helps to align the model performance with the expected outcomes reported in the original research.

- **Step 2: Model Training for Train-Test**

  The dataset is split into a 70/30 ratio for training and testing. The model is then fit into a predefined pipeline that contains all the necessary preprocessing steps. These steps include:

  - **Outlier Handling**: Outliers are handled using the Interquartile Range (IQR) method.
  - **Missing Value Imputation**: Missing values are treated using median imputation.
  - **Feature Scaling**: The features are standardized using `StandardScaler` to ensure all features are on the same scale, which improves model convergence.
  - **Class Imbalance**: The class imbalance is managed using the Synthetic Minority Over-sampling Technique (SMOTE).

  The model is trained on the training set and evaluated using the test set to measure its accuracy and other performance metrics.

- **Step 3: Model Training for Cross-Validation**

  In this step, the model is fit into the same defined pipeline as in the previous step, which includes the preprocessing steps of missing value imputation, outlier handling, and managing class imbalance using SMOTE. However, instead of using a single train-test split, the model undergoes cross-validation using the `cross_validate` function. This ensures that the model is evaluated more robustly by splitting the dataset into multiple folds and training/testing on each of them.

- **Step 4: Comparison**

  The final step involves comparing the reproduced results with those reported in the research article. Two key metrics are used for this comparison:

  - **Mean Square Error (MSE)**: Measures the average squared difference between the predicted values and the true values.
  - **Mean Absolute Error (MAE)**: Measures the average absolute difference between the predicted values and the true values.

  These metrics are used to evaluate the distance between the reproduced results and those of the research article.

2.2 Comprehensive Analysis

2.2.1 Model Performance Results

The following table shows the performance of different machine learning models based on the train-test split and cross-validation using evaluation metrics such as accuracy, precision, recall, and F1-score.

Protocol	Algorithm	Accuracy	Precision	Recall	F1-Score
Train-Test	DT	66.00	0.67	0.62	0.65
Train-Test	RF	79.00	0.78	0.81	0.79
Train-Test	SVM	70.00	0.71	0.67	0.69
Train-Test	Stacking Ensemble	75.00	0.74	0.78	0.76
Cross-Validation	DT	71.00	0.58	0.69	0.63
Cross-Validation	RF	76.00	0.64	0.68	0.66
Cross-Validation	SVM	75.00	0.63	0.69	0.66
Cross-Validation	Stacking Ensemble	76.00	0.66	0.64	0.65

Table 2: Model Performance Results Using Train-Test and Cross-Validation Protocols

Observations:

- The **Random Forest (Train-Test)** model achieved the highest overall accuracy at 79%, along with high precision and recall scores. This suggests that Random Forest performs consistently well when trained on a single dataset split.
- The **Stacking Ensemble (Train-Test)** model also performed well, achieving 75% accuracy, showing that combining models can yield competitive results.
- When using **Cross-Validation**, the performance of most models slightly decreased, especially for Decision Tree, which saw an increase in recall but a decrease in precision and F1-score.
- The **Support Vector Machine (SVM)** model performed reasonably well across both protocols, achieving similar results in both Train-Test and Cross-Validation, showing its robustness across different validation strategies.
- Overall, the **Stacking Ensemble (Cross-validation)** model yielded a balanced performance, with 76% accuracy and relatively stable precision, recall, and F1-score metrics.

2.2.2 Comparison of Percentage Differences

This table compares the percentage differences in model performance metrics across different protocols and algorithms. The metrics include Accuracy, Precision, Recall, and F1-Score.

Protocol	Algorithm	Accuracy	Precision	Recall	F1-Score
Train-Test	DT	1.41%	3.08%	4.62%	0.00%
Train-Test	RF	0.42%	2.50%	2.53%	0.00%
Train-Test	SVM	1.41%	2.90%	2.90%	0.00%
Train-Test	Stacking Ensemble	0.04%	1.33%	4.00%	1.33%
Cross-Validation	DT	3.94%	10.77%	1.47%	5.97%
Cross-Validation	RF	1.05%	16.88%	11.69%	15.38%
Cross-Validation	SVM	9.31%	7.35%	1.43%	4.35%
Cross-Validation	Stacking Ensemble	1.43%	2.94%	8.57%	5.80%

Table 3: Comparison of Percentage Differences in Model Performance Metrics Across Protocols and Algorithms

Observations:

- **Accuracy**: The differences in accuracy metrics between the reproduced model and the research article are relatively minor, with a maximum difference of 9.31% (for Cross-validation - SVM). Overall, the reproduced model is consistent, and accuracy differences are mostly below 5% for other protocols and algorithms. This suggests that both models generalize similarly in most cases, with slight variations in classification effectiveness.
- **Precision**: The reproduced model demonstrates larger differences in precision, especially with Cross-validation - RF, showing a 16.88% difference. This indicates that while the reproduced model performs well, there may be certain inconsistencies in how precisely it classifies positive instances, especially for random forest models.
- **Recall**: The recall metric shows some fluctuations, with the highest difference being 11.69% (for Cross-validation - RF). This suggests that the reproduced model may struggle slightly in detecting positive instances, particularly for certain algorithms like Random Forest under cross-validation, where it diverges more significantly from the research article's performance.
- **F1-Score**: The F1-score, which balances precision and recall, generally shows modest differences, with the largest being 15.38% for Cross-validation - RF. This reflects the combined effect of deviations in both precision and recall. The overall trends indicate that the reproduced model closely follows the original research results, but certain algorithms (e.g., RF in cross-validation) exhibit greater variation in performance.

2.2.3 Comparison Based on Evaluation Metrics

Based on the comparison metrics of model performance using **Mean Squared Error (MSE)** and **Mean Absolute Error (MAE)**, several key observations can be made:

Metric	MSE	MAE
Accuracy	6.4789	1.6550
Precision	0.00325	0.0425
Recall	0.00181	0.03375
F1-Score	0.00232	0.0300

Table 4: Comparison of Results Based on Evaluation Metrics

Observations:

- **Accuracy**: The largest difference is seen in accuracy, with an MSE of 6.4789 and an MAE of 1.6550. This indicates a notable variation in how well the reproduced model and the research article's model generalize the correct classifications.
- **Precision**: Precision exhibits much lower differences with an MSE of 0.00325 and MAE of 0.0425, showing that the model reproduces similar results for precision.
- **Recall**: A slight difference is observed with an MSE of 0.00181 and an MAE of 0.03375. The model shows close reproduction in recall.
- **F1-Score**: The F1-score also shows minor differences, with MSE at 0.00232 and MAE at 0.0300.

2.2.4 Comparison Based on Different Methods

Based on the comparison metrics of model performance using **Mean Squared Error (MSE)** and **Mean Absolute Error (MAE)**, several key observations can be made:

Protocol	Algorithm	MSE	MAE
Train-Test	DT	0.2119	0.2425
Train-Test	RF	0.0274	0.0925
Train-Test	SVM	0.2354	0.2525
Train-Test	Stacking Ensemble	0.0005	0.0200
Cross-Validation	DT	1.8107	0.7025
Cross-Validation	RF	0.1739	0.2875
Cross-Validation	SVM	10.2089	1.6200
Cross-Validation	Stacking Ensemble	0.3039	0.3050

Table 5: Comparison of Results Based on Different Methods

Observations:

- **Train-Test Methods**:
  - **Stacking Ensemble (Train-Test)** shows the least error across both MSE (0.0005) and MAE (0.0200), demonstrating a strong alignment with the results from the research article.
  - **Random Forest (Train-Test)** also performs well with an MSE of 0.0274 and MAE of 0.0925, indicating minimal variation.

  - **SVM (Train-Test)** and **Decision Tree (Train-Test)** show larger errors, with SVM having an MSE of 0.2354 and Decision Tree showing 0.2119, indicating greater differences in model performance.

- **Cross-validation Methods**:
  - **Stacking Ensemble (Cross-validation)** again shows reasonable alignment with the original results, with an MSE of 0.3039 and MAE of 0.3050.
  - **Random Forest (Cross-validation)** shows an MSE of 0.1739, but a higher MAE of 0.2875 compared to the Train-Test method.
  - **SVM (Cross-validation)** has the highest MSE at 10.2089, suggesting significant divergence from the original results in this method.

2.2.5 Why Are They Different

Several key factors could explain the differences between my results and those reported in the research article, despite using similar preprocessing steps such as IQR, Median Imputation, Standard Scaler, and SMOTE.

- **Hyperparameters of Models:** Even though both I and the research article applied the same machine learning models, variations in hyperparameter tuning could lead to performance differences. Hyperparameters such as *max_depth*, *min_samples_split*, *kernel*, *C*, and others can greatly affect the model's performance.
  - In **Random Forest**, the number of estimators (*n_estimators*) or the maximum depth of the trees can significantly impact performance.
  - In **Support Vector Machine (SVM)**, the choice of the kernel (linear, RBF, etc.) and the value of the regularization parameter (C) play crucial roles in determining the decision boundary.

  Without an exact match in hyperparameter settings, even small changes can lead to different model performance.

- **Application of SMOTE and Data Splitting:** The most critical issue that could explain the discrepancy is the order of applying SMOTE and splitting the dataset. According to my analysis, the researchers in the article applied **SMOTE before splitting the dataset** into training and testing sets. This process can lead to **data leakage**, where the test set is influenced by information from the training set.
  - When SMOTE is applied before the data is split, synthetic samples are generated based on the entire dataset, including what would become the test set. As a result, the synthetic samples are not solely generated from the training set, which violates the principle that the test set should represent unseen data.
  - This results in an artificial boost in model performance because the model essentially "sees" part of the test data during training, leading to overly optimistic evaluation metrics.

  The proper process is to first split the data into training and testing sets, and then apply SMOTE only to the training set. This ensures that the model has no prior knowledge of the test set, preventing data leakage.

- **Train-Test Split Differences:** Another potential reason for the difference in results could be how the train-test split was conducted:
  - Different **random states** or **shuffling mechanisms** during the split can lead to different subsets of data for training and testing. A well-performing model on one random split may not perform equally well on another.
- **Handling of Outliers and Missing Data:** While both I and the research article used **IQR** to handle outliers and **Median Imputation** for missing values, the specific implementation details could vary:
  - The IQR method could be applied with different thresholds for identifying outliers, such as using a 1.5x IQR rule versus a more conservative or aggressive threshold.
  - Median imputation could also differ in how the median values are calculated, especially if stratified by certain groups (e.g., class label) or calculated across the entire dataset.
- **Feature Scaling (Standard Scaler):** Another possible cause of the discrepancy could be in the application of the **Standard Scaler**:
  - If the Standard Scaler is fitted to the entire dataset before splitting, it may lead to information from the test set being used to scale the training data (**data leakage** again).
  - Proper scaling should be applied only on the training set, and then the same scaling should be applied to the test set using the parameters (mean and standard deviation) derived from the training data.
- **Cross-Validation Implementation:** The article's use of **cross-validation** could also differ from my implementation:
  - Ensuring that cross-validation is correctly nested within the hyperparameter tuning process is crucial. If cross-validation is improperly applied, it can also lead to data leakage or biased performance estimates.

3 My Machine Learning Solution

3.1 Model Description

The machine learning pipeline implemented involves multiple components aimed at robust preprocessing, feature engineering, and model optimization. Below is a detailed breakdown:

3.1.1 Preprocessing Steps

- **Handling Zero Values**: In the PIMA Indians diabetes dataset, certain medical measurements such as *Glucose*, *BloodPressure*, *SkinThickness*, *Insulin*, and *BMI* contain zero values, which are not realistic in a medical context (e.g., a glucose level of zero is impossible for a living person). Therefore, these zero values likely represent missing data.

- **Handling Missing Values**: Missing values in critical columns such as *Glucose*, *BloodPressure*, *SkinThickness*, *Insulin*, and *BMI* were replaced with *NaN* values, and a *SimpleImputer* with a median strategy was applied. This approach ensures that missing data does not negatively affect the models while preserving central tendencies of the features.
- **Outlier Detection and Capping**: An Interquartile Range (IQR) method was used to cap outliers. Capping helps prevent extreme values from distorting the model's performance. Each feature's values beyond 1.5 times the IQR were capped at the lower or upper bounds, making the data more robust to the influence of outliers.
- **Log Transformation**: To handle skewness, particularly for highly skewed features like *Insulin*, *SkinThickness*, and *BMI*, log transformation was applied (*log1p*). This transformation helps normalize the distribution of the features, making them more suitable for machine learning algorithms.
- **Feature Scaling**: After processing, *StandardScaler* was applied to normalize all features, which is essential for distance-based algorithms and ensemble methods like *RandomForest* and *XGBClassifier*, where feature scaling helps ensure uniform influence of each feature.

3.1.2 Feature Engineering

- **Polynomial Features**: Polynomial features of degree 2 were generated for selected variables, capturing interactions between features that may improve the model's predictive power. This transformation creates new features based on combinations of existing ones, potentially uncovering relationships that the original features alone could not capture.

3.1.3 Feature Selection

- **SelectKBest**: ANOVA F-values were used as a criterion for feature selection, with 40 of the most important features being retained based on their relevance to the target variable. This step reduces noise and enhances computational efficiency while focusing the model on the most predictive features.

3.1.4 Modeling (Stacking Ensemble Classifier)

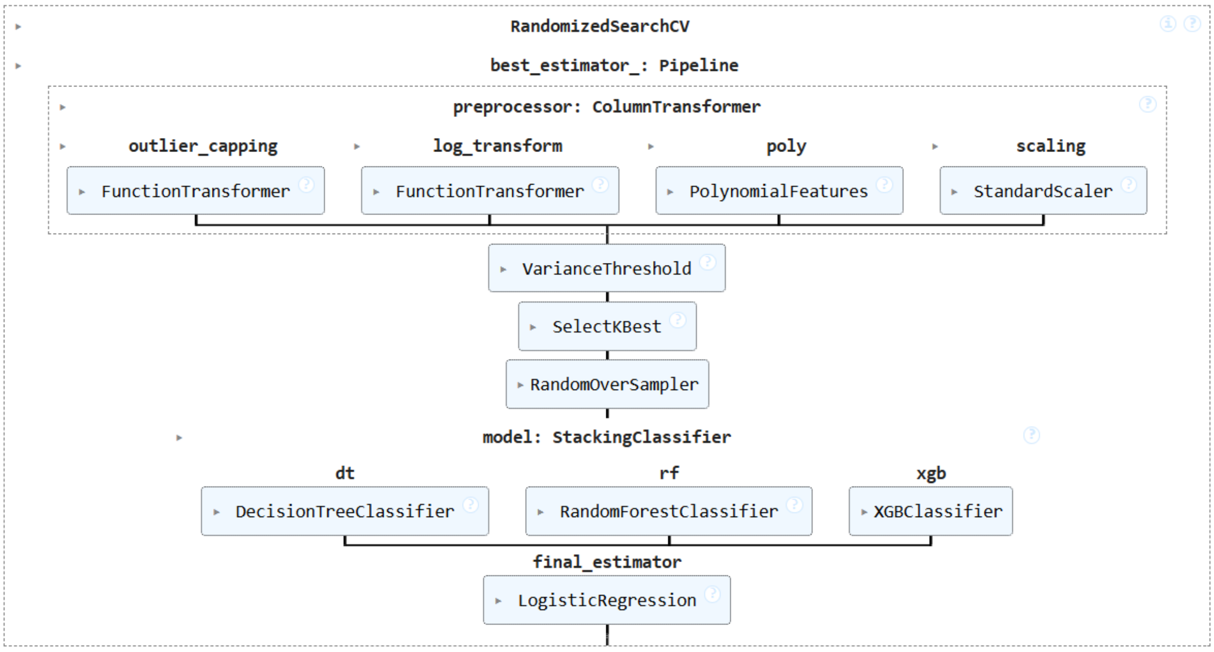

Figure 1: Machine Learning Pipeline Overview

A stacking ensemble classifier was implemented, combining three base classifiers:

- *DecisionTreeClassifier*: A decision tree model, tuned for depth and minimum samples per split and leaf.
- *RandomForestClassifier*: An ensemble of decision trees, with controlled depth and number of trees to avoid overfitting.
- *XGBClassifier*: A gradient boosting model known for handling imbalanced data and producing high accuracy by sequentially improving weak learners.

Final Estimator: LogisticRegression was used as the final estimator to combine predictions from the three base models. The logistic regression model applies L2 regularization to prevent overfitting.

3.1.5 Hyperparameter Tuning

The entire pipeline was optimized using *RandomizedSearchCV*, where hyperparameters of the base classifiers, polynomial features, and *SelectKBest* were tuned. The best combination of hyperparameters was selected based on cross-validated accuracy.

```
{
    'selecting__k': 40,
    'preprocessor__poly__degree': 2,
    'model__xgb__n_estimators': 200,
    'model__xgb__max_depth': 5,
    'model__xgb__learning_rate': 0.3,
    'model__rf__n_estimators': 100,
    'model__rf__min_samples_split': 5,
    'model__rf__max_depth': 20,
    'model__final_estimator__solver': 'lbfgs',
    'model__final_estimator__penalty': 'l2',
    'model__final_estimator__max_iter': 100,
    'model__final_estimator__C': 0.1,
    'model__dt__min_samples_split': 10,
    'model__dt__min_samples_leaf': 2,
    'model__dt__max_depth': 20
}
```

3.1.6 Used Parameters

This section provides the exact parameters used in the code for data preprocessing, model building, and hyperparameter tuning to ensure that others can follow the same procedure.

- **Data Preprocessing**
  - **SimpleImputer**:
    - **strategy**: `'median'`
  - **FunctionTransformer (cap_outliers)**:
    - Outlier capping method: IQR method where outliers below the lower bound or above the upper bound are capped.
    - **Lower Bound**: $Q1 - 1.5 \times IQR$
    - **Upper Bound**: $Q3 + 1.5 \times IQR$
  - **FunctionTransformer (log_transform)**:
    - Log transformation: $\log(1+x)$ applied to the `['Insulin', 'SkinThickness', 'BMI']` columns to handle skewness in the data.
  - **PolynomialFeatures**:
    - **Degree**: 3
  - **StandardScaler**: Standardizes the features to have a mean of 0 and a variance of 1 for the following columns:
    - Columns: `['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']`
- **Feature Selection**
  - **SelectKBest (f_classif)**:
    - **k**: 20 (for selecting the top 20 features)
  - **VarianceThreshold**:
    - **threshold**: 0 (for removing features with zero variance)
- **Models Used in Stacking Classifier**
  - **DecisionTreeClassifier**:
    - **random_state**: 42

  - **RandomForestClassifier**:
    - **random_state**: 42
  - **XGBClassifier**:
    - **random_state**: 42
  - **LogisticRegression (as final estimator in stacking)**: Default parameters unless specified in hyperparameter tuning.

- **Data Resampling**
  - **RandomOverSampler**:
    - **random_state**: 42
- **Hyperparameter Tuning**
  - **DecisionTreeClassifier**:
    - **model__dt__max_depth**: [10, 20, 30]
    - **model__dt__min_samples_split**: [5, 10]
    - **model__dt__min_samples_leaf**: [2, 5]
  - **RandomForestClassifier**:
    - **model__rf__n_estimators**: [100, 200]
    - **model__rf__max_depth**: [10, 20]
    - **model__rf__min_samples_split**: [5, 10]
  - **XGBClassifier**:
    - **model__xgb__n_estimators**: [100, 200]
    - **model__xgb__learning_rate**: [0.1, 0.2, 0.3, 0.5]
    - **model__xgb__max_depth**: [5, 7]
  - **LogisticRegression (final estimator)**:
    - **model__final_estimator__C**: [0.1, 1, 10]
    - **model__final_estimator__solver**: ['lbfgs']
    - **model__final_estimator__penalty**: ['l2']
    - **model__final_estimator__max_iter**: [100, 150, 200]
  - **SelectKBest**:
    - **selecting__k**: [10, 20, 30, 40, 50]
  - **PolynomialFeatures**:
    - **preprocessor__poly__degree**: [2, 3, 4]

3.2 Visualization

Visualizations provide insights into the distribution of features, relationships between variables, and correlations within the dataset. Below is a description of the key visualizations used to better understand the dataset and support the feature engineering process.

3.2.1 Boxplot of Features

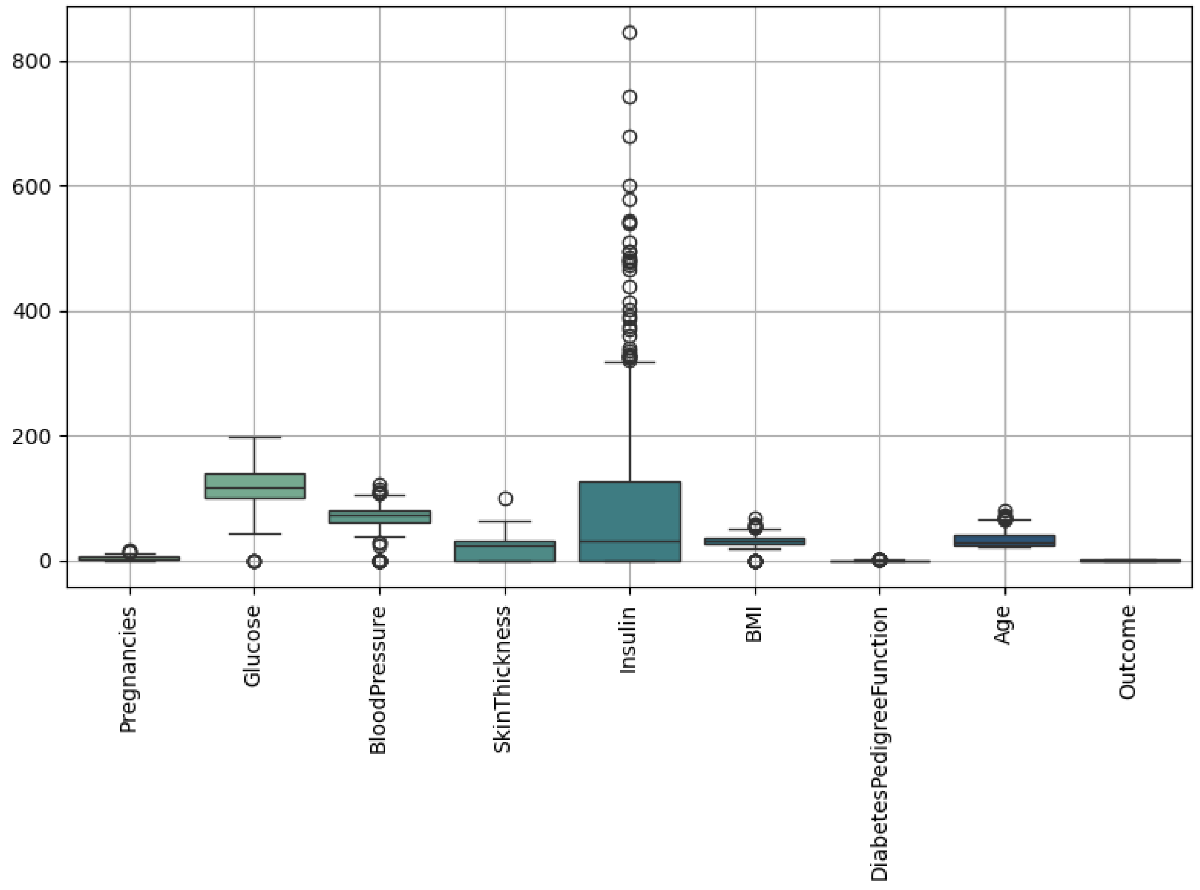

Figure 2: Boxplot displaying the distribution of features in the dataset

The boxplot displays the distribution of each feature, highlighting the presence of outliers. Features like *Insulin* and *SkinThickness* show significant outliers, particularly in the upper ranges. These outliers could potentially distort the model's performance, which is why *IQR-based capping* was applied in preprocessing to mitigate their influence. Features such as *Pregnancies*, *BloodPressure*, and *BMI* have less pronounced outliers, but still require careful treatment to ensure robust modeling.

3.2.2 Correlation Heatmap

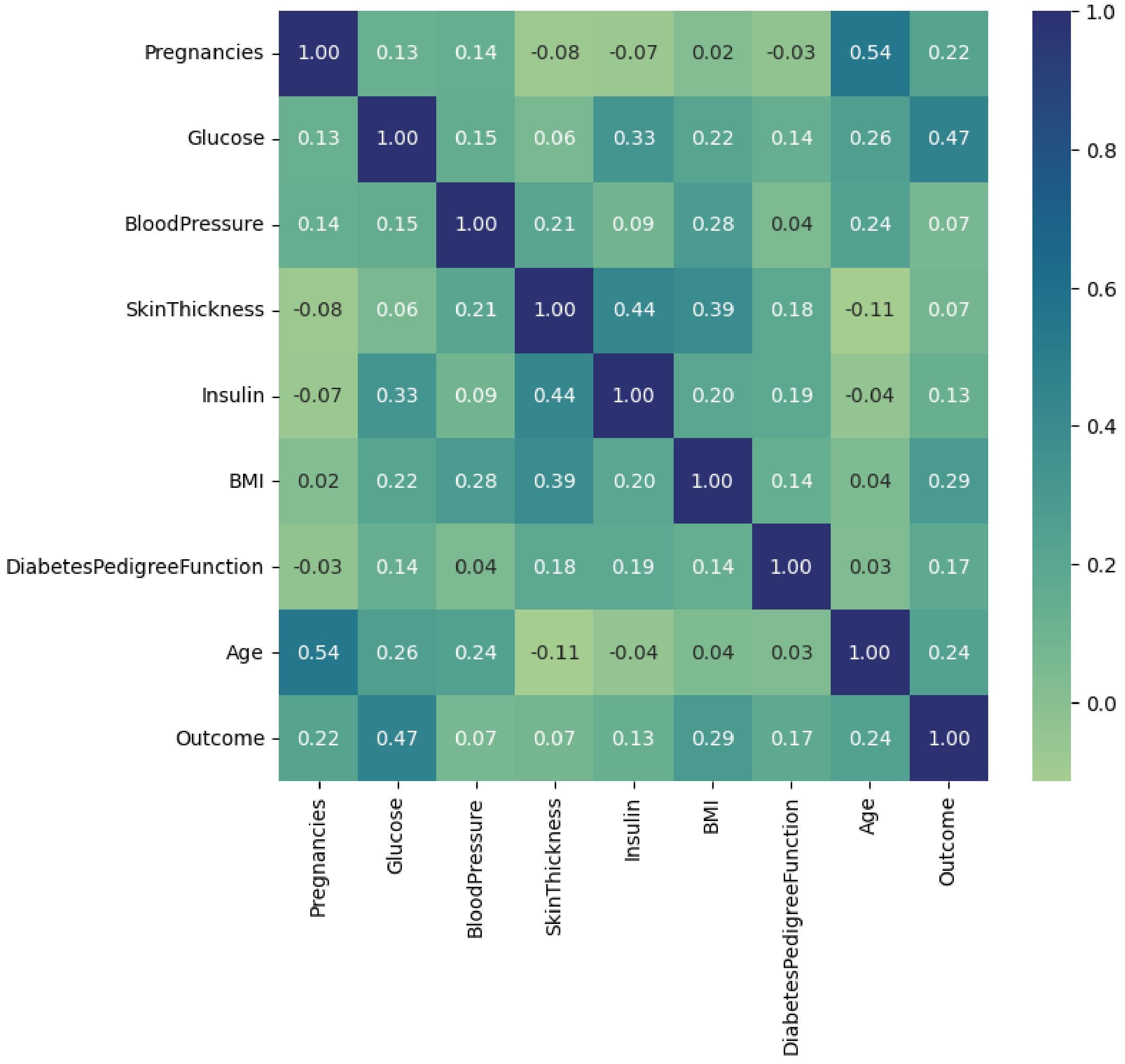

Figure 3: Correlation heatmap between features and the target variable

The heatmap visualizes the correlation between different features and the target variable (*Outcome*). *Glucose* shows the strongest correlation with *Outcome* (0.47), suggesting that higher glucose levels are associated with a higher likelihood of diabetes. Other features like *Age* and *BMI* also show moderate correlations. This heatmap informs feature selection, where features with higher correlations to *Outcome* are likely to contribute more to the prediction process.

3.2.3 Histograms of Features

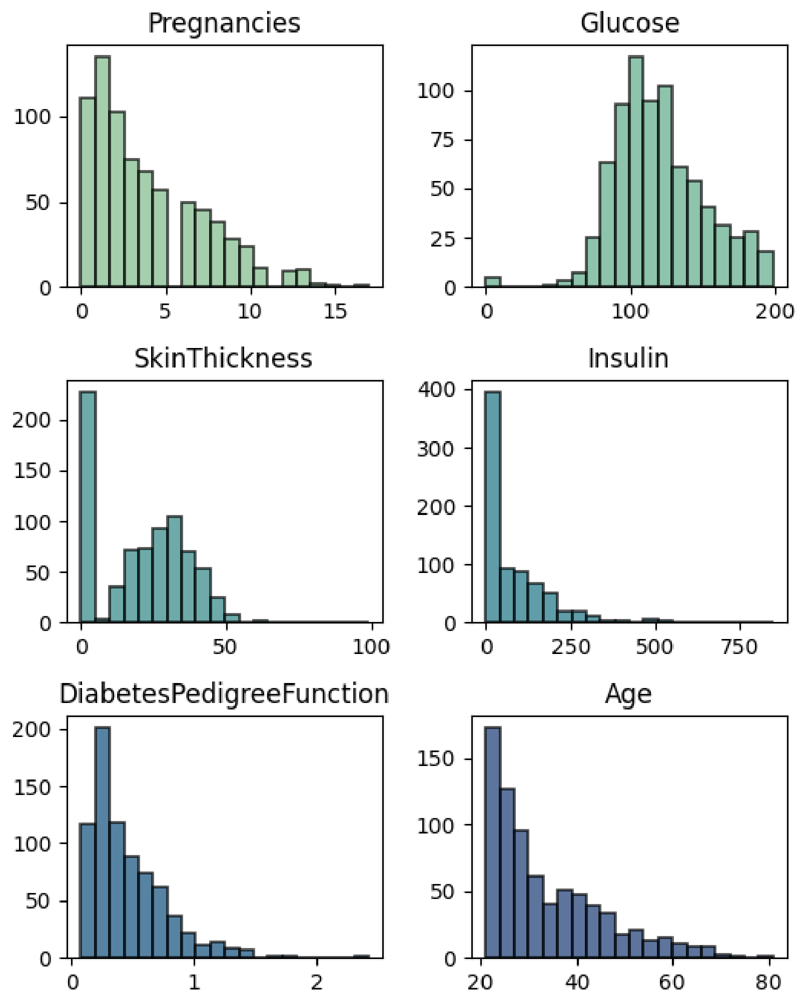

Figure 4: Histograms showing the distribution of each feature in the dataset

The histograms illustrate the distribution of each feature in the dataset. Features like *Insulin*, *SkinThickness*, and *DiabetesPedigreeFunction* show highly skewed distributions, necessitating the use of *log transformations* in the preprocessing pipeline to normalize them. For instance, *Insulin* has a large number of zeros, potentially indicating missing data that needs imputation. Features such as *Pregnancies* and *Glucose* show more balanced distributions, but still display some degree of skewness.

3.2.4 Pair Plot of Feature Relationships

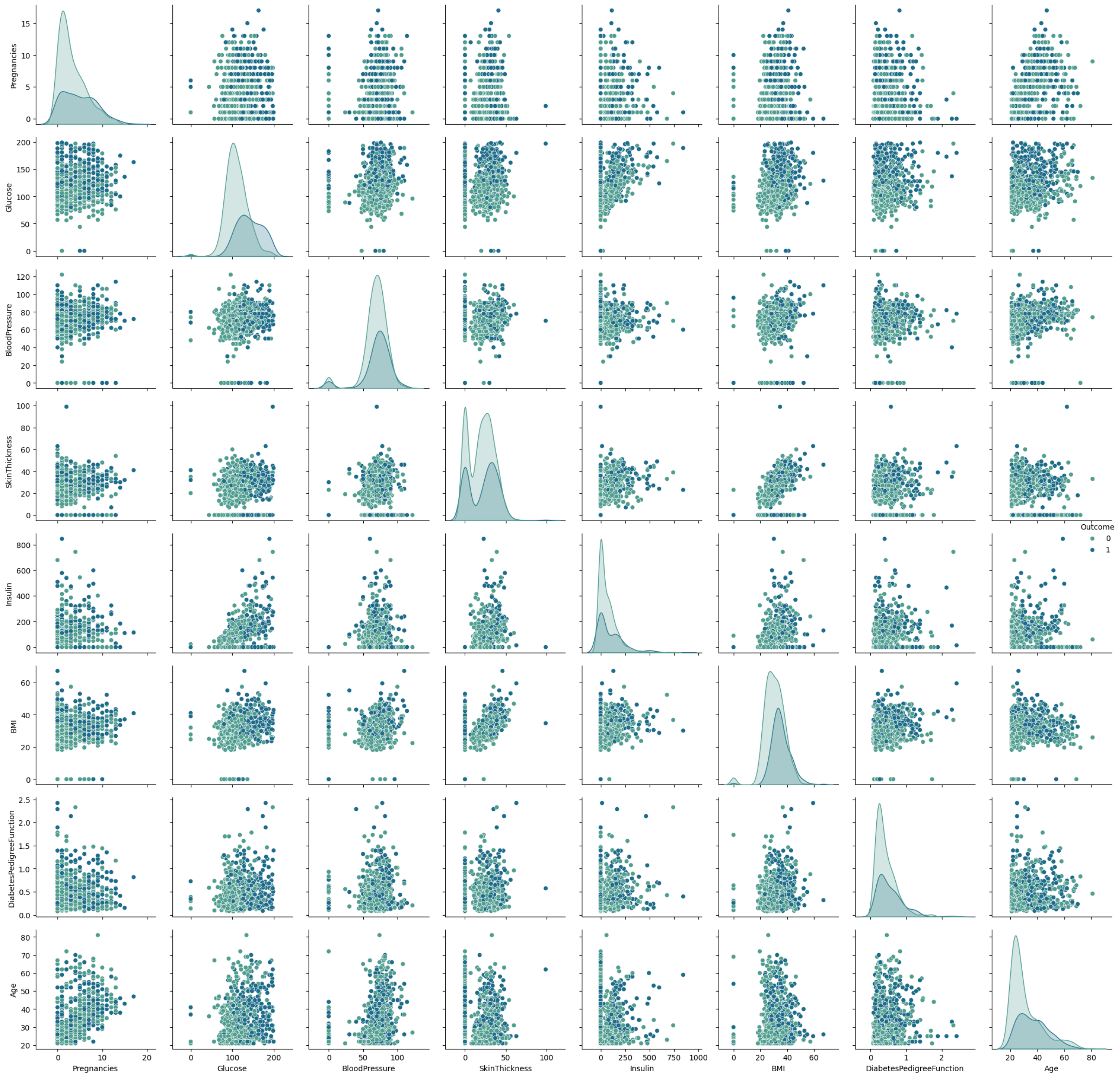

Figure 5: Pair plot showing relationships between features and their distributions

The pair plot shows the relationships between pairs of features, as well as the distribution of each feature for the two classes (0 for non-diabetic and 1 for diabetic). By visually inspecting the scatter plots, it becomes evident that certain features, such as *Glucose* and *Insulin*, exhibit distinct separation between the two outcome classes, making them valuable predictors. The diagonal plots represent the distribution of individual features, with features like *BMI* and *Glucose* showing some potential for class distinction based on their separation of values between the two classes.

3.2.5 Pie Chart of Class Distribution

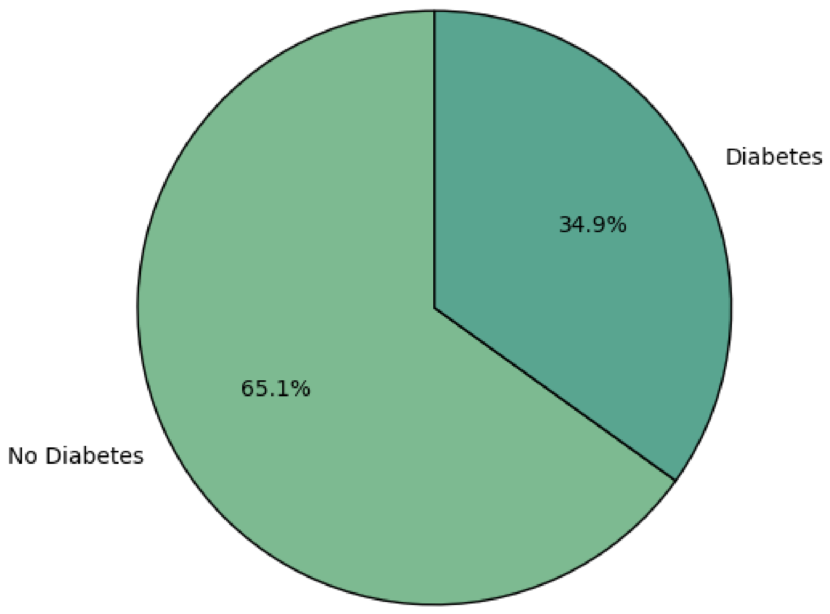

Figure 6: Pie chart displaying the class distribution of diabetic and non-diabetic individuals

The pie chart illustrates the distribution of the two classes (*No Diabetes* and *Diabetes*) in the dataset. Approximately 65.1% of the samples represent individuals without diabetes, while 34.9% of the samples represent individuals with diabetes. This imbalance highlights the need for techniques like *RandomOverSampler* to balance the training data and avoid bias toward the majority class.

3.2.6 Model Performance Comparison Using LazyPredict

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken (s)
RandomForestClassifier	0.7532	0.7349	0.7349	0.7549	0.4079
LGBMClassifier	0.7403	0.7308	0.7308	0.7439	0.4474
BaggingClassifier	0.7576	0.7264	0.7264	0.7561	0.1091
XGBClassifier	0.7316	0.7212	0.7212	0.7354	0.8347
GaussianNB	0.7403	0.7190	0.7190	0.7417	0.0314
QuadraticDiscriminantAnalysis	0.7489	0.7110	0.7110	0.7455	0.0413
CalibratedClassifierCV	0.7446	0.7077	0.7077	0.7416	0.0838
RidgeClassifier	0.7446	0.7077	0.7077	0.7416	0.0364
AdaBoostClassifier	0.7273	0.7062	0.7062	0.7291	0.2764
SVC	0.7446	0.7047	0.7047	0.7407	0.0528
LinearSVC	0.7403	0.7043	0.7043	0.7377	0.0266
LinearDiscriminantAnalysis	0.7403	0.7043	0.7043	0.7377	0.0323
LogisticRegression	0.7403	0.7043	0.7043	0.7377	0.0311
RidgeClassifierCV	0.7403	0.7014	0.7014	0.7368	0.0307
ExtraTreesClassifier	0.7316	0.7007	0.7007	0.7308	0.3774
NuSVC	0.7359	0.6922	0.6922	0.7308	0.0458
KNeighborsClassifier	0.7056	0.6896	0.6896	0.7092	0.0971
SGDClassifier	0.7013	0.6834	0.6834	0.7047	0.0316
NearestCentroid	0.6840	0.6760	0.6760	0.6897	0.0281
ExtraTreeClassifier	0.7056	0.6749	0.6749	0.7056	0.0296
DecisionTreeClassifier	0.6970	0.6742	0.6742	0.6994	0.0331
BernoulliNB	0.6753	0.6635	0.6635	0.6808	0.0342
Perceptron	0.6970	0.6624	0.6624	0.6960	0.0269
LabelPropagation	0.6537	0.6264	0.6264	0.6564	0.0581
LabelSpreading	0.6537	0.6264	0.6264	0.6564	0.0555
PassiveAggressiveClassifier	0.6234	0.6002	0.6002	0.6284	0.0281
DummyClassifier	0.6537	0.5000	0.5000	0.5168	0.0263

Table 6: Model performance comparison using LazyPredict on the dataset.

The table presents key metrics for various classifiers using *LazyPredict*, including *Accuracy*, *Balanced Accuracy*, *ROC AUC*, *F1 Score*, and *Time Taken*. The top-performing models, such as the *RandomForestClassifier* and *BaggingClassifier*, show the highest accuracy, while models like *SVC* and *LogisticRegression* offer competitive performance with lower computational time. The *DummyClassifier* serves as a baseline, illustrating the improvements made by other models.

3.2.7 Insights Gained from Visualizations

The visualizations and model performance comparison provided valuable insights that guided the preprocessing and modeling steps:

- **Outlier Detection and Treatment**: Based on the boxplots, outlier capping was necessary for several features to ensure robust model performance.
- **Feature Selection**: The correlation heatmap helped identify highly correlated features, such as *Glucose* and *BMI*, which were prioritized during feature selection.
- **Class Imbalance**: The pie chart indicated a significant imbalance between the two classes, reinforcing the use of oversampling methods like *RandomOverSampler*.
- **Feature Distributions**: Log transformations were applied to features like *Insulin* and *SkinThickness* based on their skewed distributions observed in the histograms.

- **Model Selection**: The LazyPredict table highlighted that ensemble models, such as *RandomForestClassifier* and *BaggingClassifier*, performed the best, while models like *SVC* and *LogisticRegression* offered a balance between performance and computational efficiency.

3.3 Experimental Protocol

The experimental setup was carefully designed to ensure rigorous model evaluation and comparison:

- **Data Splitting**: The dataset was split into a training set (70%) and a test set (30%). This allows the model to be trained on a majority of the data while preserving a separate test set for final performance evaluation.
- **Handling Imbalanced Data**: Since the PIMA dataset has imbalanced classes, *RandomOverSampler* was used to balance the class distribution in the training set. Oversampling the minority class helps improve the classifier's ability to correctly identify positive cases (i.e., patients with diabetes).
- **Cross-Validation**: During hyperparameter tuning, 5-fold cross-validation was used. This ensures that the model is not overfitting to a single fold, and the performance is validated on multiple subsets of the training data.
- **Performance Evaluation**: The model's performance was evaluated on the held-out test set, and various metrics such as accuracy, precision, recall, and F1-score were computed.

3.4 Evaluation Metrics

3.4.1 Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positives. It answers the question: *What proportion of positive identifications was actually correct?*

- Precision for class 0 (non-diabetic): 0.85
- Precision for class 1 (diabetic): 0.65

3.4.2 Recall (Sensitivity)

Recall is the ratio of correctly predicted positive observations to all observations in the actual class. It answers the question: *How many of the actual positives were correctly predicted?*

- Recall for class 0: 0.79
- Recall for class 1: 0.72

3.4.3 F1-Score

F1-score is the weighted average of precision and recall. This score is useful when the class distribution is uneven.

- F1-score for class 0: 0.82
- F1-score for class 1: 0.69

3.4.4 Accuracy

Accuracy is the overall correctness of the model, defined as the proportion of correctly predicted instances out of all instances.

- Accuracy: 0.77 (77%)

3.4.5 Classification Report

The detailed classification report on the test set is as follows:

```
             precision    recall  f1-score   support

        0       0.85      0.79      0.82       151
        1       0.65      0.72      0.69        80

accuracy                          0.77       231
macro avg       0.75      0.76      0.75       231
weighted avg    0.78      0.77      0.77       231
```

3.5 Comparison with Existing Literatures

3.5.1 Comparison and Discussion of Results

The accuracy achieved in my machine learning model is 77%, which is quite competitive when compared to the accuracy reported in the research article. The highest accuracy they reported is 79%; however, this comes with an important caveat — their entire dataset was preprocessed before splitting, which can lead to data leakage. Data leakage occurs when information from outside the training set is used to create the model, making the results overly optimistic and not representative of real-world performance. If they had properly split the dataset before preprocessing, their accuracy would likely drop, with a maximum expected accuracy around 73%. In contrast, my model strictly followed best practices by splitting the dataset before performing any preprocessing. This ensures that the test set remains unseen and prevents any leakage, leading to a more realistic performance estimate.

3.5.2 Focus on Recall for Imbalanced Data

It's important to note that accuracy is not the only metric to consider, especially for imbalanced datasets like the one used here. For early detection of diabetes, the focus should be on recall because it is critical to identify patients who have the disease (class 1) so that they can receive early treatment. In my model, the recall for class 1 is 72%, which is a relatively strong performance, particularly considering the slight imbalance in the dataset. In comparison to the research article, their methodology may have overemphasized accuracy, while neglecting other important metrics like recall and precision, which provide a more holistic view of model performance for imbalanced datasets. Focusing solely on accuracy can be misleading in this case because a model can achieve high accuracy simply by predicting the majority class more frequently, even if it fails to capture minority class instances effectively.

3.5.3 Final Thoughts

The performance of my model is robust, with a high recall and balanced metrics, which are more suitable for the task of early detection of diabetes. While the research article achieved slightly higher accuracy, the methodology used (preprocessing the entire dataset before splitting) likely inflated their results due to data leakage. By avoiding this pitfall, my model's results are more trustworthy and generalizable.

References

- Reza, M.S., Amin, R., Yasmin, R., Kulsum, W. and Ruhi, S. (2024) 'Improving diabetes disease patients classification using stacking ensemble method with PIMA and local healthcare data', *Heliyon*, 10(2), p. e24536. Available at: `https://www.sciencedirect.com/science/article/pii/S240584402400567X`
- pandas development team (2024). *pandas: Python Data Analysis Library.* Available at: `https://pandas.pydata.org/`
- NumPy Developers (2024). *NumPy.* Available at: `https://numpy.org/`
- scikit-learn Developers (2024). *scikit-learn: Machine Learning in Python.* Available at: `https://scikit-learn.org/`
- Matplotlib Developers (2024). *Matplotlib: Visualization with Python.* Available at: `https://matplotlib.org/`
- Waskom, M. L. (2024). *seaborn: Statistical Data Visualization.* Available at: `https://seaborn.pydata.org/`
- imbalanced-learn Developers (2024). *imbalanced-learn: A Python Package to Tackle the Curse of Imbalanced Datasets in Machine Learning.* Available at: `https://imbalanced-learn.org/`
- LazyPredict Developers (2024). *LazyPredict: A Tool for Quick Model Training and Evaluation.* Available at: `https://github.com/shankarpandala/lazypredict`
- Microsoft (2024). *LightGBM: A Fast, Distributed, High-Performance Gradient Boosting Framework.* Available at: `https://lightgbm.readthedocs.io/`

Appendix: Code Implementation

For a detailed explanation of the code implementation, you can view the code by clicking on the image below:

Click the image to watch the code.