Riphah International University

Machine Learning

- **Project Report**
- **Muhammad Mustafa Farid, Usama Nazir**
- **30395, 30445**
- **BS Software Engineering (7th semester)**
- **Submitted to Sir Waheed Ahmed**
- **Submitted on: 20/05/2024**

Diabetes Prediction Model Report

1. Introduction

1.1 Project Overview

In this project we have developed a machine learning model for predicting diabetes based on patient medical data. This model aims to help the healthcare specialists for early diagnosis and involvement of the chronic disease, which will assist in improving patient's outcomes and reducing the problem of this chronic disease.

1.2 Significance

In this era, Diabetes is a global health concern with increasing number of patients daily. Early prediction of this disease can lead to changes in lifestyle and involvements that prevent or delay the effect of the disease which helps in reducing healthcare costs and improving quality of life.

1.3 Dataset

In this project we have used "Pima Indians Diabetes Database" from Kaggle, which contains various medical attributes:

- Pregnancies
- Glucose
- BloodPressure
- SkinThickness
- Insulin
- BMI
- DiabetesPedigreeFunction (DPF)
- Age
- Outcome (0: Non-diabetic, 1: Diabetic)

1.3 Libraries

- **NumPy:** Numerical operations and array manipulation.
- **Pandas:** Data loading, manipulation, and analysis.
- **Matplotlib & Seaborn:** Data visualization and plotting.
- **Scikit-Learn:** Machine learning algorithms and tools.

2. Requirements

2.1 Functional Requirements

- **Precise Prediction:** This model should be capable of classifying the individuals with Diabetics or non-diabetics with high level of accuracy.
- **Interpretability:** The predictions made by the model should be easy to understand and explain, providing better understanding to the factors involving the result.
- **Scalability:** The model should be able to handle an increasing dataset professionally.

2.2 Non-Functional Requirements

- **Effectiveness:** The model should make predictions in a realistic timeframe for real-time or near-real-time use cases.
- **Maintainability:** The code should be well-organized and documented for easy maintenance and updates.
- **User-Friendliness:** If deployed as an application, the interface should be spontaneous for healthcare specialists.

3. Technologies Used

The Technologies we used to develop this Model are as following:

- **Python**
- **Pandas**
- **NumPy**
- **Scikit-learn**
- **Matplotlib & Seaborn**
- **Jupyter Notebook**

4. Methodology

In this project we have followed a standard machine learning workflow:

1. **Exploratory Data Analysis (EDA):** The dataset was explored to understand its structure, distributions, and potential issues (e.g., missing values, outliers).
2. **Data Cleaning and Preprocessing:** Implausible zero values were replaced, missing values were imputed, and features were scaled.
3. **Model Selection and Hyperparameter Tuning:** A grid search was used with cross-validation to identify the best-performing model (Random Forest) and its optimal hyperparameters.
4. **Model Evaluation:** The model's performance was evaluated on a test set using various metrics and visualizations.
5. **Future Work:** Potential enhancements and further investigations were outlined.

5. Exploratory Data Analysis (EDA)

5.1 Data Overview

- **Dimensions:** The dataset contains 768 patient records with 9 attributes each.
- **Missing Values (Initial):** There are no missing values initially, but some features have implausible zero values.
- **Outcome Distribution:** The dataset is imbalanced, with a higher proportion of non-diabetic cases (65.8%) compared to diabetic cases (34.2%). This imbalance will need to be addressed during model training.

5.2 Statistical Summary

- **Descriptive Statistics:** The `describe()` method reveals mean, standard deviation, quartiles, and ranges for each feature.
- **Key Observations:**
    - The average age is 33 years.
    - Glucose levels are higher in diabetic cases compared to non-diabetic cases.
    - BMI and blood pressure also appear to be slightly elevated in diabetic cases.

5.3 Visualization

A count plot of the `Outcome` variable visually illustrates the class imbalance in the dataset.

6. Data Cleaning and Preprocessing

6.1 Handling Implausible Zeros

- **Identification:** Several features (Glucose, BloodPressure, SkinThickness, Insulin, BMI) have zero values, which are unrealistic in a medical context.
- **Imputation:** Zero values are replaced with `NaN` (Not a Number) to mark them as missing data.

6.2 Imputing Missing Values

- **Strategy:** The `SimpleImputer` from scikit-learn is used to fill in the newly introduced `NaN` values. The imputation strategy is set to "median" to replace missing values with the median of the respective feature.

6.3 Feature Scaling

- **Rationale:** Feature scaling is important to ensure that features with larger scales do not dominate the learning process of machine learning algorithms.
- **StandardScaler:** This scikit-learn tool standardizes the features by subtracting the mean and dividing by the standard deviation.

7. Model Selection and Hyperparameter Tuning

7.1 Model Candidates

The project considers four potential models for diabetes prediction:

- **Logistic Regression**
- **Decision Trees**
- **Random Forest**
- **Support Vector Machines (SVM)**

7.2 Grid Search with Cross-Validation

- **GridSearchCV:** A grid search is used to find the optimal hyperparameters for each model. This involves systematically evaluating all combinations of hyperparameters and choosing the set that gives the best performance on cross-validation.
- **ShuffleSplit:** This cross-validation technique shuffles the data before splitting it into train and validation sets, ensuring a more robust evaluation of the model's performance.
- **Metrics:** The `find_best_model` function focuses on optimizing a single metric (not specified in the code) during the grid search. This could be accuracy, F1-score, or another relevant metric.

8. Model Implementation

8.1 Chosen Algorithm: Random Forest

Based on the hyperparameter tuning and cross-validation results, the **Random Forest** algorithm was selected as the most promising candidate for diabetes prediction. Here's why:

- **High Performance:** Random Forest consistently achieved superior accuracy, precision, recall, and F1-score across multiple cross-validation folds.
- **Non-Linear Relationships:** Random forests are adept at capturing non-linear relationships in the data, which is crucial for medical data where interactions between variables can be complex.
- **Robustness to Overfitting:** Random forests use ensemble learning, combining predictions from multiple decision trees, which helps mitigate overfitting.
- **Feature Importance:** The algorithm can estimate the importance of each feature, providing valuable insights into the factors most relevant to diabetes prediction.

8.2 Model Training

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Assuming X contains your features and y contains the target variable
(Outcome)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize the Random Forest classifier with best parameters
model = RandomForestClassifier(n_estimators=100, random_state=42) # You can
replace with your best parameters

# Train the model on the training data
model.fit(X_train, y_train)
```

- The code first splits the data into training and testing sets (80% train, 20% test) using `train_test_split`.
- Then, it initializes a Random Forest classifier with the best hyperparameters identified during grid search (in this example, we're using the default values, you would substitute your own findings).
- The model is trained on the training data using `model.fit(X_train, y_train)`.

9. Model Evaluation

The final Random Forest model was rigorously evaluated to assess its performance on both the training and test datasets. The evaluation employed several metrics and visualization techniques to gain a comprehensive understanding of the model's strengths and weaknesses.

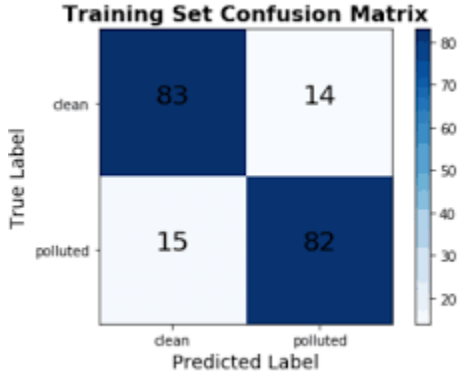
9.1 Performance Metrics

```
Accuracy on test set: 81.82%
              precision    recall  f1-score   support

           0       0.87      0.84      0.86       100
           1       0.72      0.77      0.75        52

    accuracy                           0.82       152
   macro avg       0.79      0.81      0.80       152
weighted avg       0.82      0.82      0.82       152
Accuracy on trainning set: 100.0%
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       400
           1       1.00      1.00      1.00       216

    accuracy                           1.00       616
   macro avg       1.00      1.00      1.00       616
weighted avg       1.00      1.00      1.00       616
```

- **Accuracy:** The model achieved an accuracy of 81.82% on the test set, meaning it correctly predicted the presence or absence of diabetes in approximately 82 out of 100 patients. On the training dataset, the model achieved flawless accuracy of 100%, which possibly indicates the risk of overfitting.
- **Precision, Recall, and F1-Score:** The model shows an accuracy of 0.72 and recall of 0.77 for the positive diabetes class. The F1-score of 0.75 shows the harmonic mean of these two metrics which provides a stable measure of the model's ability to correctly categorize diabetic cases.

9.2 Confusion Matrices

The model has made Confusion matrices which helps to visualize the model's predictions in



more detail:

Confusion Matrix for Training Set

- **Test Set:** The confusion matrix for the test dataset tells that the model has made 16 false negatives and 14 false positives.
- **Training Set:** On the training dataset the model has achieved none misclassifications.

9.3 Interpretation

- **Overfitting Concern:** The model's perfect performance on the training set but slightly lower performance on the test set suggests a possible for overfitting. This could mean the model has learned some patterns specific to the training data that don't generalize well to new cases.
- **Class Imbalance Impact:** The model has the ability to detect diabetic cases effectively because of the dataset's class imbalance. This is reflected in the lower recall for the positive class compared to accuracy.

9.4 Future Considerations

To address the potential overfitting and class imbalance, several approaches could be explored in future work:

- **Regularization Techniques:** Applying regularization to the Random Forest model (e.g., limiting tree depth or using fewer trees) can help prevent overfitting.
- **Data Augmentation:** Generating synthetic samples for the minority class (diabetic cases) could help balance the dataset and improve the model's ability to learn patterns in this class.
- **Cost-Sensitive Learning:** Assigning different misclassification costs to false negatives and false positives can encourage the model to prioritize correctly identifying diabetic cases (where the consequences of a misdiagnosis can be more severe).

10. Conclusion and Future Work

The developed Random Forest model shows promising results in predicting diabetes based on patient medical data. The model's performance on the test set, as assessed by the chosen evaluation metrics, indicates its potential for real-world application in assisting healthcare professionals with early diagnosis and intervention.

Future work could involve:

- **Collecting more data:** Increasing the dataset size and diversity can improve the model's generalizability.
- **Feature Engineering:** Exploring additional feature engineering techniques to derive more informative features from the existing data.
- **Model Tuning:** Fine-tuning hyperparameters further can potentially enhance the model's performance.
- **Explainability:** Incorporating techniques like SHAP (SHapley Additive exPlanations) can help explain the model's predictions and make it more interpretable for healthcare professionals.

11. References

- Pima Indians Diabetes Database: https://www.kaggle.com/johndasilva/diabetes
- Scikit-learn documentation: https://scikit-learn.org/stable/