Implementation Report

DiabPredict - Predictive Modeling for Diabetes Onset

Team Members: Nagendra Madi Reddy & Vishwajeeth Balaji

In the provided code, the analysis focuses on a diabetes dataset using Python with libraries such as NumPy, Pandas, Matplotlib, Seaborn, and scikit-learn. Let's break down the code into sections:

1) Dataset Upload and Exploration:

* The code begins by loading a diabetes dataset from a CSV file using pd.read_csv and displaying the first few rows using head().

* Column names are checked using diabetes_df.columns, and dataset information is obtained with diabetes_df.info().

2) Handling Missing Values:

* Dataset Loading: The pd.read_csv() function loads the dataset from the supplied file path, presumably containing information about diabetes.

* Columns to Replace and Copy Creation: 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI') is a list of columns that need to be replaced. Next, diabetes_df_copy is used to store a deep copy of the original dataframe (diabetes_df).

* Mean Imputation: Using the mean values derived from diabetes_df_copy.mean(), mean imputation is carried out on the designated columns of diabetes_df_copy via the fillna() function.

* Display of NaNs: Lastly, print(diabetes_df_copy) is used to show the number of NaN values in each column of the updated dataframe.sum().isnull()).

3) Data Distribution Plots Before and After Mean Imputation:

* Histograms of the data distribution before and after handling missing values are plotted using hist().

4) Feature Correlation:

* A heatmap of the correlation matrix is generated to visualize the relationships between features.

5) Data Scaling:

The original dataset is displayed, and then feature scaling is performed using StandardScaler on selected columns.

6) Data Splitting – Feature Selection:

Random Forest

* Feature and Target Variable Specification: The 'Outcome' column is excluded as the target variable by using the drop() function to describe the features (X) and target variable (y).

* Train-Test Split: The train_test_split() method from the sklearn.model_selection module divides the dataset into training and testing sets. A random state value of 7 is supplied, and the split is carried out using a test size of 0.33.

* Building and Training a Random Forest Classifier: The RandomForestClassifier() constructor from the sklearn.ensemble module is used to build a Random Forest Classifier with 200 estimators. The fit() method is then used to train the classifier using the training set.

* The Significance of Features Extraction: The feature_importances_ property is used to retrieve the feature importances from the Random Forest model that has been trained.

* The Significance of Features Visualization: A bar plot made with matplotlib.pyplot and seaborn is used to display the feature importances. The most significant aspects are emphasized by sorting the significance ratings in descending order.

* Top Features Selection: Features with relevance values over the threshold are chosen using a threshold value (threshold = 0.02). Next, the chosen features are taken out of the feature significance DataFrame.

* Selection of Selected Features for Training and Testing Data: Finally, only the features that have been chosen based on their relevance values are included in the training and testing datasets. In order to accomplish this, the original datasets (X_train and X_test) are indexed using the chosen feature names that were acquired in the previous phase. The datasets that have been filtered are called X_train_selected and X_test_selected, in that order.

Decision Tree

* Classifier Creation and Training: The DecisionTreeClassifier() constructor is used to generate a DecisionTreeClassifier with the default parameters. Using the fit() technique, the classifier is trained on the training data (X_train, y_train).

* The Significance of Features Extraction: The feature_importances_ property is used to retrieve the feature importances from the trained Decision Tree model.

* The Significance of Features Visualization: A bar plot made with matplotlib.pyplot and seaborn is used to display the feature importances. The most significant aspects are emphasized by sorting the significance ratings in descending order.

* Top Features Selection: Features with importance ratings over the threshold are chosen using a threshold value (threshold_dtree = 0.02). Next, the chosen features are taken out of the feature significance DataFrame.

* Selection of Training and Testing Datasets with specified Features: Based on the relevance values of the specified features, only those datasets from the training and testing phases are included. In order to accomplish this, the original datasets (X_train and X_test) are indexed using the chosen feature names that were acquired in the previous phase. X_train_selected_dtree and X_test_selected_dtree are the names of the filtered datasets, respectively.

Support Vector Machine

* Specify Features and Target Variable: Deleting the 'Outcome' column from the dataset to generate the feature matrix X and extracting the 'Outcome' column as the target variable y will specify the features (X) and target variable (y).

* Train-Test Split: The train_test_split function from sklearn.model_selection divides the data into training and testing sets. A random state value of 7 and a test size of 33% are used for the split.

* Model Development and Instruction: The LinearSVC() constructor from sklearn.svm is used to create a Linear Support Vector Classifier (LinearSVC) model, which is then trained using the fit() function on the training data (X_train, y_train).

* Recursive Feature Elimination (RFE): Using the RFE function from sklearn.feature_selection, RFE is used to choose the top 5 features. The desired number of features to select is indicated by the n_features_to_select option, which is set to 5.

* Extract Selected Features: To extract the selected features, use the RFE object's support_ attribute, which yields a boolean mask that indicates the selected features. After then, the original feature columns are index-based for these chosen features.

* Training and Testing Data Filtering: In the training and testing datasets, only the features that have been chosen are kept. To do this, select feature names from the previous phase are indexed into the original datasets (X_train and X_test).

7) Model Building - Random Forest:

* A Random Forest Classifier is trained and evaluated on both the training and test sets.

* The accuracy scores, confusion matrix, and classification report are displayed.

8) Model Building - Decision Tree:

* A Decision Tree Classifier is trained and evaluated similarly, with accuracy scores, confusion matrix, and classification report.

9) Model Building - Support Vector Machines (SVM):

* A Support Vector Machine Classifier is trained and evaluated, showing accuracy score, confusion matrix, and classification report.

10) Models Comparison:

* Random Forest, Decision Tree, and SVM models are trained, and their accuracy scores, f1score, recall, precision are compared using a heatmap.

Each section of the code focuses on a specific aspect of the analysis, including data exploration, preprocessing, model building, and comparison.

Conclusion:

The DiabPredict predictive modeling analysis has been conducted using three different machine learning algorithms: Decision Tree, Random Forest, and Support Vector Machines (SVM). The evaluation metrics, accuracy, precision, recall & f1score are utilized to assess the performance of each model on predicting diabetes onset. The obtained scores are as follows:

* With an accuracy of 0.77, the Random Forest model demonstrates its resilience in forecasting the onset of diabetes. The model shows a significant capacity to identify non-diabetic and diabetic persons correctly, with precision values of 0.81 and 0.70 for class 0 and class 1, respectively. Additionally, class 0's recall of 0.85 shows how well the model captures true negatives, and class 1's recall of 0.64 emphasizes the model's capacity to recognize true positives. The model's balanced performance in terms of precision and recall for both classes is further shown by the equivalent F1-scores for class 0 and class 1, which are 0.83 and 0.67, respectively.

* With a precision of 0.77 for Class 0 and 0.61 for Class 1, the Decision Tree model yields an accuracy of 0.72. Class 0 recall is 0.78, but Class 1 recall is 0.60. Class 0 has an F1-score of 0.78, whereas Class 1 has a score of 0.60. All all, these metrics show that the model can accurately categorize examples, while it lags somewhat behind the Random Forest ensemble method. All things considered; the Decision Tree model shows promise in identifying patterns in the dataset.

* The accuracy of the Support Vector Machines (SVM) model is 75%; for class 0 and class 1, the precision is 76% and 73%, respectively. Class 0 recall is 90%, which shows that the model can accurately identify true negatives; class 1 recall is 49%, which shows that the model performs poorly in identifying true positives. Class 0's F1-score of 0.82 indicates a balance between recall and precision, whereas class 1's F1-score of 0.58 shows potential for progress in catching positive cases. SVM performs competitively overall, making use of its capacity to manage complicated decision boundaries, even though its

accuracy is marginally less than that of the Random Forest model.

In conclusion, the DiabPredict predictive modeling analysis provides a promising foundation for predicting diabetes onset, with the Random Forest model standing out as the most accurate and reliable choice among the evaluated algorithms.

GITHUB LINK: https://github.com/nmadired/DiabPredict---Predictive-Modeling-for-Diabetes-Onset