Implementation Report

DiabPredict - Predictive Modeling for Diabetes Onset

Team Members: Nagendra Madi Reddy & Vishwajeeth Balaji

The given code conducts a detailed analysis of a diabetes dataset using Python and various libraries, including NumPy, Pandas, Matplotlib, Seaborn, and scikit-learn. Let's categorize the code into distinct sections:

Hello

1) Loading and Exploring the Dataset:

* The initial section of the code involves importing a diabetes dataset from a CSV file using pd.read_csv and displaying the initial rows using head().

* The column names are inspected using diabetes_df.columns, and an overview of the dataset's structure is obtained through diabetes_df.info().

2) Handling Missing Data:

* **Dataset Loading**: The dataset, presumably containing diabetes-related information, is loaded using pd.read_csv().

* **Columns Replacement and Copy Creation**: A list of columns to be replaced, namely 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', is defined. A deep copy of the original dataframe (diabetes_df) is created with the name diabetes_df_copy.

* **Mean Imputation:** Imputation is performed on diabetes_df_copy by replacing NaN values in specified columns with their respective mean values obtained from diabetes_df_copy.mean().

* **Display of NaNs:** The resulting count of NaN values in each column is displayed using print(diabetes_df_copy.isnull().sum()).

Hello

3) Data Distribution Visualization:

Histograms Before and After Imputation: Histograms depicting the data distribution are plotted using hist() to compare the dataset's characteristics before and after mean imputation.

4) Feature Correlation Visualization:

Correlation Heatmap: A heatmap is created based on the correlation matrix to visually represent the inter-relationships between different features.

5) Data Scaling:

Original Dataset Display: The initial dataset is showcased.

Feature Scaling: Selected columns undergo feature scaling using StandardScaler to ensure uniformity and comparability among different features.

6) Data Splitting – Feature Selection:

Random Forest

Feature and Target Variable Specification:

Excluding 'Outcome' Column: The 'Outcome' column, denoting the target variable, is omitted using the drop() function, defining the features (X) and target variable (y).

Train-Test Split: Dividing the Dataset: Utilizing train_test_split() from sklearn.model_selection, the dataset undergoes separation into training and testing sets. A random state value of 7 is specified, with a test size of 0.33.

Building and Training a Random Forest Classifier: Random Forest Construction: The RandomForestClassifier() constructor from sklearn.ensemble establishes a Random Forest Classifier, configured with 200 estimators. Training is executed using the fit() method with the training set.

Significance of Features Extraction:

Feature Importance Retrieval: Extracting feature importances is performed through the feature_importances_ property of the trained Random Forest model.

Significance of Features Visualization:

Bar Plot Display: A bar plot, created with matplotlib.pyplot and seaborn, illustrates the feature importances. Emphasis is placed on the most significant features by arranging them in descending order.

Top Features Selection: Threshold-based Feature Selection: Features surpassing a specified threshold (threshold = 0.02) in relevance are selected. These chosen features are extracted from the feature importance DataFrame.

Selection of Chosen Features for Training and Testing Data:

Filtering Datasets: The training and testing datasets are filtered to include only the features deemed relevant. This is achieved by indexing the original datasets (X_train and X_test) with the chosen feature names. The resultant datasets are named X_train_selected and X_test_selected, respectively.

Decision Tree:

Classifier Creation and Training:

DecisionTreeClassifier Initialization:

Employing the DecisionTreeClassifier() constructor, a DecisionTreeClassifier is instantiated with default parameters.

Training the Classifier: The fit() method facilitates the training of the classifier using the provided training data (X_train, y_train).

Significance of Features Extraction:

Feature Importance Retrieval: Utilizing the feature_importances_ property, feature importances are extracted from the trained Decision Tree model.

Significance of Features Visualization:

Visualizing Feature Importances: A bar plot, constructed with matplotlib.pyplot and seaborn, visually represents feature importances. The significance ratings are sorted in descending order, emphasizing the most crucial aspects.

Top Features Selection:

Threshold-based Feature Selection: Features surpassing a specified threshold (threshold_dtree = 0.02) in importance are selected. These chosen features are extracted from the feature importance DataFrame.

Selection of Training and Testing Datasets with Specified Features:

Filtering Datasets: Based on the relevance values of the specified features, the training and testing datasets are filtered. This is achieved by indexing the original datasets (X_train and X_test) with the chosen feature names. The resulting datasets are named X_train_selected_dtree and X_test_selected_dtree, respectively.

Support Vector Machine:

Specify Features and Target Variable:

Feature Matrix and Target Variable Definition: The 'Outcome' column is removed from the dataset to form the feature matrix (X), and the 'Outcome' column itself is extracted as the target variable (y).

Train-Test Split:

Data Splitting: Utilizing the train_test_split function from sklearn.model_selection, the data is divided into training and testing sets. A random state of 7 and a test size of 33% are specified for the split.

Model Development and Training:

Linear Support Vector Classifier (LinearSVC): The LinearSVC() constructor from sklearn.svm is employed to instantiate a Linear Support Vector Classifier (LinearSVC) model.

Training the Model: The fit() function is used to train the LinearSVC model on the provided training data (X_train, y_train).

Recursive Feature Elimination (RFE):

Feature Selection with RFE: Using the RFE function from sklearn.feature_selection, Recursive Feature Elimination (RFE) is applied to select the top 5 features. The n_features_to_select parameter is set to 5 to indicate the desired number of features to be chosen.

Extract Selected Features:

Feature Extraction: The support_ attribute of the RFE object produces a boolean mask indicating the selected features. Subsequently, the original feature columns are indexed based on these chosen features.

Training and Testing Data Filtering:

Dataset Filtering: Only the selected features are retained in both the training and testing datasets. This involves indexing the feature names obtained in the previous phase into the original datasets (X_train and X_test). The resulting datasets are denoted as X_train_selected_svc and X_test_selected_svc.

7) Model Building - Random Forest:

* A Random Forest Classifier is constructed and assessed on both the training and test sets.

* The evaluation metrics, including accuracy scores, confusion matrix, and classification report, are presented.

8) Model Building - Decision Tree:

* Similar to the Random Forest approach, a Decision Tree Classifier is created and evaluated.

* The evaluation includes accuracy scores, confusion matrix, and classification report.

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

* A Support Vector Machine Classifier is developed and evaluated on the dataset.

* The evaluation metrics, such as accuracy score, confusion matrix, and classification report, are demonstrated.

10) Models Comparison:

* Random Forest, Decision Tree, and SVM models are individually trained.

* A comparative analysis is conducted by examining their accuracy scores, f1 scores, recall, and precision.

* The findings are presented visually using a heatmap.

Each section of the code is dedicated to a specific aspect of the analysis, covering data exploration, preprocessing, model construction, and comprehensive model comparison.

Conclusion :

In the DiabPredict predictive modeling analysis, three distinct machine learning algorithms—Decision Tree, Random Forest, and Support Vector Machines (SVM)—were employed to predict the onset of diabetes. Evaluating the primary metric, accuracy, yielded the following insights:

Random Forest: Achieving an accuracy of 0.77, the Random Forest model showcased robust predictive capabilities. Notably, it demonstrated high precision values of 0.81 and 0.70 for non-diabetic and diabetic classes, respectively. With a recall of 0.85 for class 0 and 0.64 for class 1, the model effectively captured true negatives and true positives. Balanced F1-scores of 0.83 and 0.67 further underscored the model's proficiency.

Decision Tree: The Decision Tree model exhibited an accuracy of 0.72, with precision values of 0.77 and 0.61 for class 0 and class 1. While class 0 recall was 0.78, class 1 recall lagged at 0.60. F1-scores of 0.78 for class 0 and 0.60 for class 1 suggested the model's ability to categorize examples accurately, albeit slightly trailing the Random Forest approach.

Support Vector Machines (SVM): With an accuracy of 75%, the SVM model demonstrated competitive performance. Precisions of 76% (class 0) and 73% (class 1) indicated balanced classification. Notably, class 0 recall at 90% showcased the model's proficiency in identifying true negatives, while class 1 recall at 49% highlighted room for improvement. The F1-scores of 0.82 (class 0) and 0.58 (class 1) indicated a promising overall performance, leveraging SVM's capability to handle intricate decision boundaries.

In conclusion, the DiabPredict analysis provides a strong foundation for diabetes onset prediction. Among the evaluated algorithms, the Random Forest model emerges as the most accurate and reliable choice.

GITHUB LINK: https://github.com/Vishwajeeth2000/DiabPredict---Predictive-Modeling-for-Diabetes-Onset