Elements of AIML

Assignment - 1

Name   -   Yashika Garg

SAP ID   -   500119489

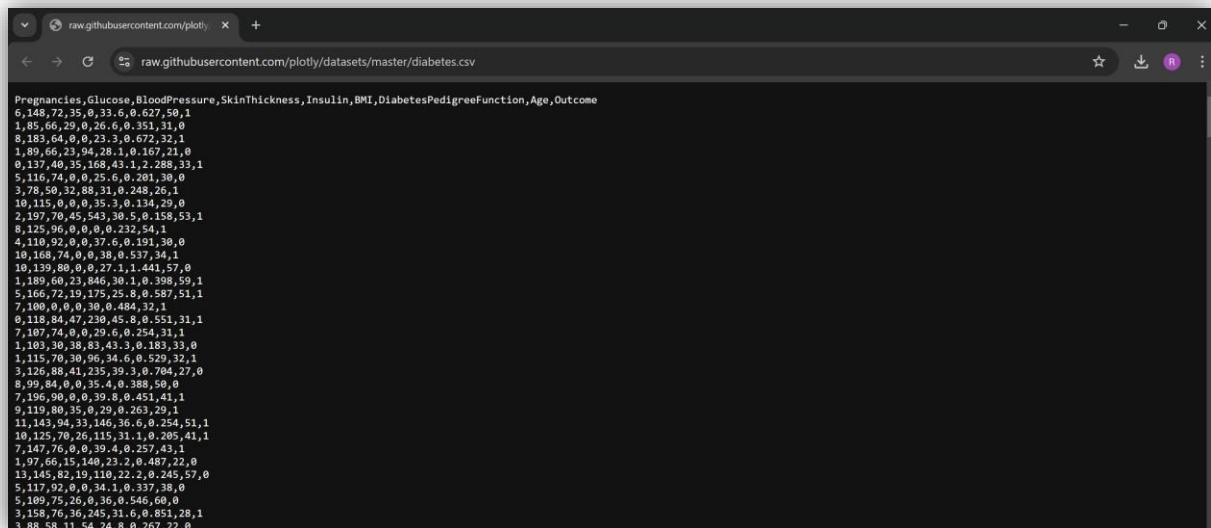Batch   –   12

Roll No.   –   R2142230357

Topic: Prediction of Diabetes in Females(P_D_F)

STEPS OF THE CODE:

Step 1: Data Acquisition -- Look for a problem and a dataset that you can use

Problem :- Predict diabetes risk based on health metrics

Dataset :- Diabetes.csv (Imported From GitHub)


```
# Load Dataset
url = 'https://raw.githubusercontent.com/plotly/datasets/master/diabetes.csv'
try:
    data = pd.read_csv(url)
except Exception as e:
    print("Error loading data:", e)
```

Step 2 -- Define the methodology and the objectives of your work

Methodology

1. **Data Preprocessing**: The diabetes dataset is loaded, cleaned, and preprocessed. Missing or zero values in key medical features are replaced with median values. SMOTE is applied to address class imbalance, and all features are standardized using StandardScaler

2. **Model Training & Evaluation**: Four models—Logistic Regression, Random Forest, SVM, and KNN—are trained using 5-fold cross-validation to evaluate accuracy and precision. The model with the best performance is chosen for real-time predictions.

3. **User Interaction & Deployment**: The best model is trained on the full dataset, allowing for user input of health metrics to predict diabetes risk. The model and scaler are saved as .pkl files for easy deployment.

Objectives

1. **Develop an Accurate Diabetes Prediction Model** using balanced data and optimal scaling.

2. **Select the Best Algorithm** for robust predictions by comparing multiple models.

3. **Enable Real-Time Predictions** through user inputs.

4. **Deploy a Scalable Tool** that supports healthcare applications for early diabetes detection.

Step 3: Data Preprocessing

```python
# Data Preprocessing
# Handle missing values and separate features/target
for column in ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']:
    data.loc[data[column] == 0, column] = data[column].median()

X = data.drop('Outcome', axis=1)
y = data['Outcome']

# Balance classes using SMOTE
smote = SMOTE(random_state=42)
X, y = smote.fit_resample(X, y)

# Scale features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

Step 4: Use multiple ML methods and validate them using K-Fold Cross Validation.

```python
# Install required libraries for imbalanced learning and XGBoost
!pip install -q imbalanced-learn xgboost
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
import warnings
import joblib
from google.colab import files
# Suppress the specific UserWarning from sklearn
warnings.filterwarnings("ignore", category=UserWarning, module='sklearn.base')
```

```python
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Check the shapes of the training and testing sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape

# Define Models for Comparison
models = {
    'Logistic Regression': LogisticRegression(max_iter=200),
    'Random Forest': RandomForestClassifier(),
    'Support Vector Machine': SVC(probability=True),
    'K-Nearest Neighbors': KNeighborsClassifier(),
}

# Model Training and Evaluation with K-Fold Cross Validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
results = {}

for model_name, model in models.items():
    with np.errstate(divide='ignore', invalid='ignore'):
        accuracy = cross_val_score(model, X_train, y_train, cv=kf, scoring='accuracy').mean() * 100
        precision = cross_val_score(model, X_train, y_train, cv=kf, scoring='precision').mean() * 100
        results[model_name] = {'Accuracy': accuracy, 'Precision': precision}
```

Step 5: Comparing the results using suitable performance metrics such as accuracy, precision.

```python
# Display results for each model
for model_name, metrics in results.items():
    print(f"\n{model_name} Performance:")
    for metric, score in metrics.items():
        print(f"{metric}: {score:.2f}")

best_model_name = max(results, key=lambda x: (results[x]['Accuracy'], results[x]['Precision']))
best_model_metrics = results[best_model_name]
print(f"\nBest Model: {best_model_name}")
print(f"Accuracy: {best_model_metrics['Accuracy']:.2f}%", f"Precision: {best_model_metrics['Precision']:.2f}%")
# Test the Best Model on Sample Input Data
# Here, we select the best model based on accuracy and precision
best_model = models[best_model_name]
best_model.fit(X_train, y_train)  # Train on the full training dataset after K-Fold validation
```

```
Logistic Regression Performance:
Accuracy: 73.75
Precision: 76.11

Random Forest Performance:
Accuracy: 80.50
Precision: 79.76

Support Vector Machine Performance:
Accuracy: 79.38
Precision: 77.99

K-Nearest Neighbors Performance:
Accuracy: 76.63
Precision: 73.43

Best Model: Random Forest
Accuracy: 80.50% Precision: 79.76%
```

Testing the model on sample data (user given)

This code segment takes user input for various health metrics related to diabetes risk and uses the trained model to predict the likelihood of diabetes. Here's a brief breakdown of the steps:

1. Collect User Input: Prompts the user to enter values for key features such as pregnancies, glucose level, blood pressure, skin thickness, insulin level, BMI, diabetes pedigree function, and age.

2. Scale the Input: Converts the user input into a NumPy array, then scales it using the previously fitted StandardScaler to match the model's training data scale.

3. Make Prediction: The model predicts whether the user has diabetes (1) or not (0) and provides the probability of diabetes.

4. Display Results: Prints the prediction (0 or 1) and the probability (as a percentage) of diabetes for the input data.

This enables real-time diabetes risk assessment based on user-provided health metrics.

```
try:
    # Collect user inputs for each feature
    user_input = []
    user_input.append(float(input("Pregnancies (0-17): ")))
    user_input.append(float(input("Glucose (0-199): ")))
    user_input.append(float(input("Blood Pressure (0-122): ")))
    user_input.append(float(input("Skin Thickness (0-99): ")))
    user_input.append(float(input("Insulin (0-846): ")))
    user_input.append(float(input("BMI (0-67): ")))
    user_input.append(float(input("Diabetes Pedigree Function (0.078-2.42): ")))
    user_input.append(float(input("Age (21-81): ")))

    # Convert to numpy array and scale the input
    user_input_scaled = scaler.transform(np.array([user_input]))

    # Make prediction
    with np.errstate(divide='ignore', invalid='ignore'):
        prediction = best_model.predict(user_input_scaled)
        prediction_proba = best_model.predict_proba(user_input_scaled)

    # Output prediction and probability
    print(f"\nSample Prediction (0 = No Diabetes, 1 = Diabetes): {prediction[0]}")
    print(f"Probability of Diabetes: {prediction_proba[0][1]:.2f}")
```

```
Pregnancies (0-17): 14
Glucose (0-199): 180
Blood Pressure (0-122): 115
Skin Thickness (0-99): 55
Insulin (0-846): 341
BMI (0-67): 34
Diabetes Pedigree Function (0.078-2.42): 1.24
Age (21-81): 32

Sample Prediction (0 = No Diabetes, 1 = Diabetes): 1
Probability of Diabetes: 0.68
```