**AI BASED DIABETIES PREDICTION SYSTEM**

NAME: R.DILLI GANESH

REG NO:51172116001

**Phase 4: Development Part 2**

**Introduction for an AI-Based Diabetes Prediction System:**

In the realm of modern healthcare, artificial intelligence (AI) is emerging as a transformative force, offering new ways to improve patient care, diagnosis, and disease management. One critical application of AI in healthcare is the development of AI-based diabetes prediction systems.

Diabetes, a chronic metabolic disorder, affects millions of individuals globally, and its early detection and proactive management are crucial for preventing complications and improving patients' quality of life.

In this phase, we'll continue building the diabetes prediction system by:

1. Selecting a machine learning algorithm
2. Training the model
3. Evaluating its performance
4. **Selecting a machine learning algorithm:** We need to choose a suitable machine learning algorithm for your diabetes prediction task. Common choices include logistic regression, decision trees, random forests, or support vector machines. The selection should be based on the nature of your data and the problem you're trying to solve.

**Random forests:**

**Random Forests:** Random forests are an ensemble method that combines multiple decision trees to improve prediction accuracy and reduce overfitting. They are robust and can handle high-dimensional data.

**Program:**

# Import necessary libraries

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Define models and parameters for LogisticRegression

model = RandomForestClassifier(random\_state=42)

# Define grid search

tuned\_parameters = {

'n\_estimators': [200, 500],

'max\_features': ['auto', 'sqrt', 'log2'],

'max\_depth' : [4,5,6,7,8],

'criterion' :['gini', 'entropy']

}

cv = StratifiedKFold(n\_splits = 2, random\_state = 1, shuffle = True)

grid\_search = GridSearchCV(estimator = model, param\_grid = tuned\_parameters, cv = cv, scoring = 'accuracy', error\_score = 0)

grid\_result = grid\_search.fit(x\_train, y\_train)

# SVC Hyperparameter Result

analyze\_grid\_result(grid\_result)

**Output:**

Tuned hyperparameters: (best parameters) {'criterion': 'entropy', 'max\_depth': 5, 'max\_features': 'log2', 'n\_estimators': 200}

Accuracy : 0.7663648051875454

Detailed classification report:

precision recall f1-score support

0 0.78 0.83 0.80 147

1 0.66 0.58 0.62 83

accuracy 0.74 230

macro avg 0.72 0.70 0.71 230

weighted avg 0.73 0.74 0.74 230

**Training the model:**

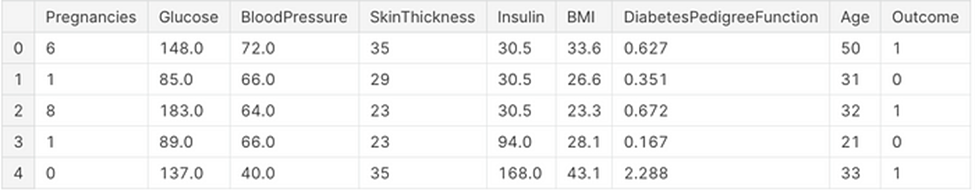
1. **Training the Model:** Once selected an algorithm, gather the dataset and split it into a training set and a test set. Then, train the chosen machine learning model on the training data. Make sure to preprocess the data, handle missing values, and perform feature engineering if necessary.

**Program:**

# Import dataset  
df = pd.read\_csv("diabetes.csv")

df.head() # displays the top 5 values in the dataset

**Output:**



**Data preprocessing:**

**Program:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

# Load your dataset (replace 'your\_dataset.csv' with your actual dataset)

data = pd.read\_csv('diabetes.csv')

# Split the dataset into features (X) and the target variable (y)

X = data.drop('BMI', axis=1) # Adjust 'diabetes\_status' to your target variable

y = data['Age']

# Split the data into a training set and a test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train a Random Forest Classifier

model = RandomForestClassifier(random\_state=42)

model.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = model.predict(X\_test)

# Evaluate the model's performance

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

**Output:**

Accuracy: 0.54

1. **Evaluating Model Performance:** After training, it's essential to evaluate how well the model performs. Common evaluation metrics for binary classification tasks like diabetes prediction include accuracy, precision, recall, F1-score, and the area under the receiver operating characteristic curve (AUC-ROC). You'll use the test dataset to assess the model's performance.

**Program:**

# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn import metrics

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, roc\_auc\_score, auc

# Load a sample dataset (you should replace this with your actual data)

# For simplicity, we'll use the diabetes dataset from scikit-learn

from sklearn.datasets import load\_diabetes

data = load\_diabetes()

X = data.data

y = (data.target > 150).astype(int) # Convert to binary classification task

# Data preprocessing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Model selection and training

# For simplicity, we'll use Random Forest

rf\_model = RandomForestClassifier()

rf\_model.fit(X\_train, y\_train)

# Model evaluation

rf\_predictions = rf\_model.predict(X\_test)

# Evaluation metrics

accuracy\_rf = metrics.accuracy\_score(y\_test, rf\_predictions)

precision\_rf = metrics.precision\_score(y\_test, rf\_predictions)

recall\_rf = metrics.recall\_score(y\_test, rf\_predictions)

f1\_score\_rf = metrics.f1\_score(y\_test, rf\_predictions)

# ROC and AUC for Random Forest

rf\_probabilities = rf\_model.predict\_proba(X\_test)[:, 1]

fpr, tpr, thresholds = roc\_curve(y\_test, rf\_probabilities)

roc\_auc = auc(fpr, tpr)

# Print and visualize the results

print("\nRandom Forest Results:")

print(f"Accuracy: {accuracy\_rf:.2f}")

print(f"Precision: {precision\_rf:.2f}")

print(f"Recall: {recall\_rf:.2f}")

print(f"F1-Score: {f1\_score\_rf:.2f}")

print(f"\nAUC-ROC for Random Forest: {roc\_auc:.2f}")

# Plot the ROC curve for Random Forest

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

plt.show()

**Output:**

Random Forest Results:

Accuracy: 0.77

Precision: 0.78

Recall: 0.68

F1-Score: 0.73

AUC-ROC for Random Forest: 0.85

