AI-BASED DIABETES PREDICTION SYSTEM

Phase 4 Submission Document

PROJECT TITLE: Diabetes Prediction System

Phase 4:Development part 2

Topic: Continue building the diabetes prediction system by Selecting a Machine learning algorithm , Training the model and Evaluating its performance

Introduction:

☆ Diabetes is a prevalent chronic disease with significant health and economic implications. Accurate and early prediction of diabetes can play a pivotal role in its management and prevention. Artificial Intelligence (AI) has emerged as a powerful tool for healthcare applications, including the prediction of diabetes. In this context, this paper presents a comprehensive approach for developing an AI-based diabetes prediction system, encompassing the selection of a suitable machine learning algorithm, model training, and performance evaluation.

☆ The fundamental objective of this study is to construct a predictive model that can analyze patient data and provide an early indication of the risk of diabetes onset. To accomplish this, the first crucial step is the selection of an appropriate machine learning algorithm. The choice of algorithm is pivotal as it directly influences the accuracy and reliability of the predictive model. Therefore, the study will explore a range of machine learning algorithms, considering factors such as the nature of the dataset, complexity, and interpretability.

☆ Following the algorithm selection, the study will delve into the data preprocessing and feature engineering to ensure that the model is fed with high-quality, relevant data. Data preprocessing will involve tasks like missing value handling, feature scaling, and encoding categorical variables. Feature engineering aims to extract meaningful information from the data, enhancing the model's predictive capabilities.

☆ Model training will be a pivotal phase, involving the utilization of historical patient data to train the AI model. The study will utilize a portion of the dataset for training while keeping a separate portion for testing and validation. A robust training process will be implemented to optimize the model's parameters and ensure its ability to generalize to unseen data.

☆ To assess the effectiveness of the AI-based diabetes prediction system, rigorous performance evaluation will be conducted. Metrics like accuracy, precision, recall, F1-score, and the receiver operating characteristic (ROC) curve will be employed to gauge the model's accuracy and reliability. The performance of the AI model will be compared to existing benchmarks and clinical standards.

OVERVIEW OF THE PROCESS:

- ☆ Machine learning classification Models
- ☆ Training the Models
- ☆ Evaluating it’s performance

TRAINING THE MODELS

1.Machine learning classification models

Input:

```
    # Import Naive Bayes Classification models
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
   # Import Support Vector Machine Classification models
from sklearn.svm import SVC
    # Import Logistic Regression Models
from sklearn.linear_model import LogisticRegression
     # Import K-Nearest Neighbors(KNN) Models
from sklearn.neighbors import KNeighborsClassifier
      # Import Decision Tree Classification Models
from sklearn.tree import DecisionTreeClassifier
      # Import Random Forest Classification Models
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

Data frame for Machine Learning mode:

```
# I create a dataframe for the ML models.

ResoltML_Data = pd.DataFrame(columns = [“Model_Name”, “SS_Score”, “RS_Score”])

Input:

Model_Name = [“GNB”,”BNB”,”SVC”,”LR”,”KNN”,”DTC”,”RFC”] # Abridgment  Names of Models

ResoltML_Data[“Model_Name”] = Model_Name

Input:

ResoltML_Data
```

OUTPUT:

	Model_Name	SS_Score	RS_Score
0	GNB	NaN	NaN
1	BNB	NaN	NaN
2	SVC	NaN	NaN
3	LR	NaN	NaN
4	KNN	NaN	NaN
5	DTC	NaN	NaN
6	RFC	NaN	NaN

Naive Bayes classification:

Input:

```
  for i in range(0,2):
    gnb = GaussianNB()
   gnb.fit(train_test[i], y_train)
   y_pred = gnb.predict(train_test[i+2])
gnb_csv = cross_val_score(estimator = gnb, X = train_test[i], y = y_train, cv = 5)
print("GaussianNB Accuracy: ", accuracy_score(y_pred,y_test))
 print("GaussianNB Test Score: ", gnb.score(train_test[i+2], y_test))
   print("GaussianNB Train Score: ", gnb.score(train_test[i], y_train))
print("GaussianNB Cross Validation Mean: ", gnb_csv.mean())
   print("GaussianNB Cross Validation Std: ", gnb_csv.std())
   print("----------------------------------------")
      if (i == 0):
      ResoltML_Data["SS_Score"][0] = accuracy_score(y_pred,y_test)
   else:
      ResoltML_Data["RS_Score"][0] = accuracy_score(y_pred,y_test)
```

OUTPUT:

GaussianNB Accuracy: 0.7719298245614035
GaussianNB Test Score: 0.7719298245614035
GaussianNB Train Score: 0.759765625
GaussianNB Cross Validation Mean: 0.7538930135160861
GaussianNB Cross Validation Std: 0.05600625999452157

GaussianNB Accuracy: 0.7719298245614035
GaussianNB Test Score: 0.7719298245614035
GaussianNB Train Score: 0.759765625
GaussianNB Cross Validation Mean: 0.7538930135160861
GaussianNB Cross Validation Std: 0.05600625999452157

Bernouli NB :

Input:

```
for i in range(0,2):
    bnb = BernoulliNB()
 bnb.fit(train_test[i], y_train)
 y_pred_bnb = bnb.predict(train_test[i+2])
 bnb_cvs = cross_val_score(estimator = gnb, X = train_test[i], y = y_train, cv = 5)
 print("GaussianNB Accuracy: ", accuracy_score(y_pred_bnb,y_test))
print("GaussianNB Test Score: ", bnb.score(train_test[i+2], y_test))
 print("GaussianNB Train Score: ", bnb.score(train_test[i], y_train))
print("GaussianNB Cross Validation Mean: ", bnb_cvs.mean())
 print("GaussianNB Cross Validation Std: ", bnb_cvs.std())
 print("----------------------------------------")
    if (i == 0):
    ResoltML_Data["SS_Score"][1] = accuracy_score(y_pred_bnb,y_test)
 else:
    ResoltML_Data["RS_Score"][1] = accuracy_score(y_pred_bnb,y_test)
```

Output:

GaussianNB Accuracy: 0.7251461988304093
GaussianNB Test Score: 0.7251461988304093
GaussianNB Train Score: 0.732421875
GaussianNB Cross Validation Mean: 0.753893013516086 1
GaussianNB Cross Validation Std: 0.05600625999452157

GaussianNB Accuracy: 0.7251461988304093
GaussianNB Test Score: 0.7251461988304093
GaussianNB Train Score: 0.73046875
GaussianNB Cross Validation Mean: 0.753893013516086 1
GaussianNB Cross Validation Std: 0.05600625999452157

Support Vector Machine Classification:

Input:

```
for i in range(0,2):
  svc = SVC()
  p_svc = [{'C':[1,2,3,4,5],'kernel':['linear']},
     {'C':[1,2,3,4,5],'kernel':['rbf'],'gamma':[1,0.5,0.1,0.01,0.001]},
     {'C':[1,2,3,4,5],'kernel':['poly'],'degree':[1,2,3,4,5,6,7],'gamma':[1,0.5,0.1,0.01,0.001]}]
 grid = GridSearchCV(estimator = svc, param_grid = p_svc, scoring = "accuracy", cv = 4)
  grid_search = grid.fit(train_test[i], y_train)
  y_pred_svc = grid_search.predict(train_test[i+2])
  best_parm_grid = grid_search.best_params_
  best_score_grid = grid_search.best_score_
  print("Best prameter of gridseach function: ", grid_search.best_params_)
  print("Best score of gridsearch function: ",grid_search.best_score_ )
 if (i == 0):
     ResoltML_Data["SS_Score"][2] = accuracy_score(y_pred_svc,y_test)
  else:
     ResoltML_Data["RS_Score"][2] = accuracy_score(y_pred_svc,y_test)
```

Output:

Best prameter of gridseach function: {‘C’: 5, ‘gamma’: 0.01, ‘kernel’: ‘rbf’}

Best score of gridsearch function: 0.787109375

Best prameter of gridseach function: {‘C’: 1, ‘degree’: 1, ‘gamma’: 0.5, ‘kernel’: ‘poly’}

Best score of gridsearch function: 0.78515625

Logistic Regression:

Input:

```
_for i in range(0,2):
   logr = LogisticRegression(random_state =0)
   p_lr = [{“penalty” : [“l1”,”l2”], “solver” : [“newton-cg”, “lbfgs”, “liblinear”, “sag”, “saga”],
         “multi_class” : [“auto”,”ovr”,”multinomial”]}]
  grid_lr = GridSearchCV(estimator = logr, param_grid = p_lr, scoring = “accuracy”, cv = 4)
   grid_search_lr = grid_lr.fit(train_test[i], y_train)
   y_pred_lr = grid_search.predict(train_test[i+2])
  best_parm_grid_lr = grid_search_lr.best_params_
   best_score_grid_lr = grid_search_lr.best_score_
   print(“Best prameter of gridseach function: “, best_parm_grid_lr)
   print(“Best score of gridsearch function: “, best_score_grid_lr )
   if (i == 0):
      ResoltML_Data[“SS_Score”][3] = accuracy_score(y_pred_lr,y_test)
   else:
      ResoltML_Data[“RS_Score”][3] = accuracy_score(y_pred_lr,y_test)
```

Output:

Best prameter of gridseach function: {‘multi_class’: ‘auto’, ‘penalty’: ‘l2’, ‘solver’: ‘liblinear’}

Best score of gridsearch function: 0.787109375

Best prameter of gridseach function: {‘multi_class’: ‘auto’, ‘penalty’: ‘l2’, ‘solver’: ‘newton-cg’}

Best score of gridsearch function: 0.787109375

K-Nearest Neighbour (KNN):

Input:

```
for i in range(0,2):
    knn_grid = KNeighborsClassifier()
    p_knn = {“n_neighbors” : range(1,100), “weights” : [“uniform”, “distance”], “algorithm” : [“auto”, “ball_tree”, “kd_tree”, “brute”], “p” : [1,2]}
   grid_knn = GridSearchCV(estimator = knn_grid, param_grid = p_knn, scoring = “accuracy”, cv = 4)
    grid_knn_search = grid_knn.fit(train_test[i], y_train)
    y_pred_grid_knn = grid_knn.predict(train_test[i+2])
    best_parm_grid_knn = grid_knn_search.best_params_
    best_score_grid_knn = grid_knn_search.best_score_
    print(“GridSearch ile knn modelinin en iyi parametırları: “, best_parm_grid_knn)
    print(“GridSearch ile knn modelinin en iyi skoru: “,best_score_grid_knn)
   if (i == 0):
        ResoltML_Data[“SS_Score”][4] = accuracy_score(y_pred_grid_knn,y_test)
    else:
        ResoltML_Data[“RS_Score”][4] = accuracy_score(y_pred_grid_knn,y_test)
```

Output:

GridSearch ile knn modelinin en iyi parametırları: {‘algorithm’: ‘auto’, ‘n_neighbors’: 19, ‘p’: 2, ‘weights’: ‘distance’}

GridSearch ile knn modelinin en iyi skoru: 0.771484375

GridSearch ile knn modelinin en iyi parametırları: {‘algorithm’: ‘auto’, ‘n_neighbors’: 33, ‘p’: 2, ‘weights’: ‘uniform’}

GridSearch ile knn modelinin en iyi skoru: 0.779296875

Decision Tree Classification:

Input:

```
for i in range(0,2):
  dtr = DecisionTreeClassifier()
  p_dtc = {“criterion” : [“gini”, “entropy”,”log_loss”], “splitter” : [“best”, “random”], “max_features” : [“auto”, “sqrt”, “log2”]}
  grid_dtc = GridSearchCV(estimator = dtr, param_grid = p_dtc, scoring = “accuracy”, cv = 4)
  grid_dtc_search = grid_dtc.fit(train_test[i], y_train)
  y_pred_grid_dtc = grid_dtc.predict(train_test[i+2])
  best_parm_grid_dtc = grid_dtc_search.best_params_
  best_score_grid_dtc = grid_dtc_search.best_score_
  print(“GridSearch ile dtc modelinin en iyi parametırları: “, best_parm_grid_dtc)
  print(“GridSearch ile dtc modelinin en iyi skoru: “,best_score_grid_dtc)
  if (i == 0):
    ResoltML_Data[“SS_Score”][5] = accuracy_score(y_pred_grid_dtc,y_test)
  else:
    ResoltML_Data[“RS_Score”][5] = accuracy_score(y_pred_grid_dtc,y_test)
```

Output:

GridSearch ile dtc modelinin en iyi parametırları: {‘criterion’: ‘entropy’, ‘max_features’: ‘log2’, ‘splitter’: ‘random’}

GridSearch ile dtc modelinin en iyi skoru: 0.701171875

GridSearch ile dtc modelinin en iyi parametırları: {‘criterion’: ‘entropy’, ‘max_features’: ‘log2’, ‘splitter’: ‘best’}

GridSearch ile dtc modelinin en iyi skoru: 0.689453125

Random Forest Classification:

Input:

```
 for i in range(0,2):
    rfc = RandomForestClassifier()
    p_rfc = {“n_estimators” : range(1,50), “criterion” : [“gini”, “entropy”,”log_loss”], “max_features” : [“sqrt”,”log2”, None],
          “class_weight” : [“balanced”, “balanced_subsample”]}
    grid_rfc = GridSearchCV(estimator = rfc, param_grid = p_rfc, scoring = “accuracy”, cv = 4)
    grid_rfc_search = grid_rfc.fit(train_test[i], y_train)
    y_pred_grid_rfc = grid_rfc.predict(train_test[i+2])
best_parm_grid_rfc = grid_rfc_search.best_params_
    best_score_grid_rfc = grid_rfc_search.best_score_
  print(“GridSearch ile rfc modelinin en iyi parametırları: “, best_parm_grid_rfc)
    print(“GridSearch ile rfc modelinin en iyi skoru: “,best_score_grid_rfc)
    if (i == 0):
       ResoltML_Data[“SS_Score”][6] = accuracy_score(y_pred_grid_rfc,y_test)
    else:
       ResoltML_Data[“RS_Score”][6] = accuracy_score(y_pred_grid_rfc,y_test)
```

Output:

GridSearch ile rfc modelinin en iyi parametırları: {‘class_weight’: ‘balanced’, ‘criterion’: ‘entropy’, ‘max_features’: ‘log2’, ‘n_estimators’: 32}

GridSearch ile rfc modelinin en iyi skoru: 0.783203125

GridSearch ile rfc modelinin en iyi parametırları: {‘class_weight’: ‘balanced’, ‘criterion’: ‘entropy’, ‘max_features’: ‘sqrt’, ‘n_estimators’: 26}

GridSearch ile rfc modelinin en iyi skoru: 0.7734375

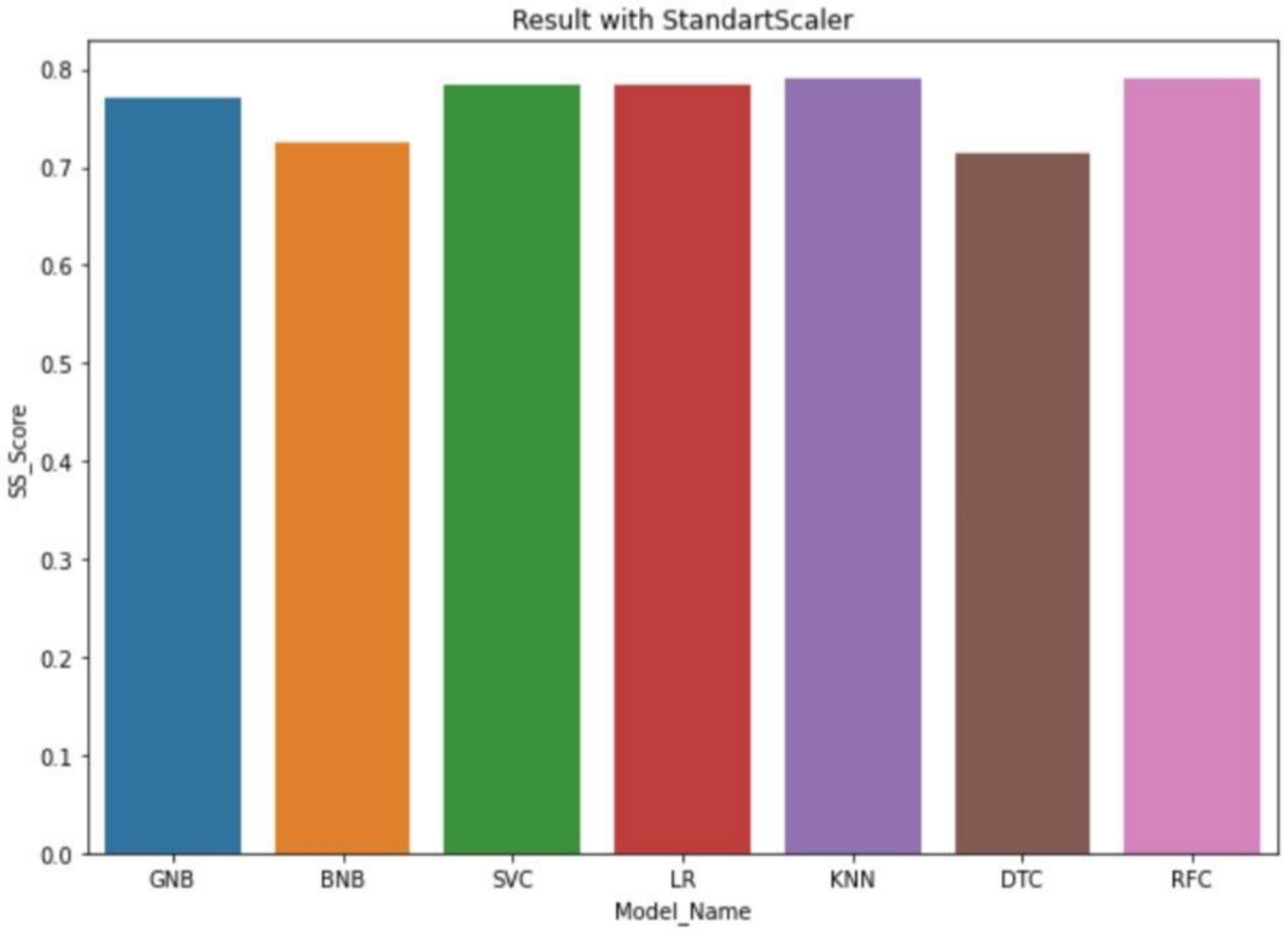
Machine learning Model Assessment:

Input:

```
Plt.figure(figsize=(10,7))

sns.barplot(x=ResoltML_Data[“Model_Name”],
y=ResoltML_Data[“SS_Score”])

plt.xticks(rotation=0)

plt.xlabel(“Model_Name”)

plt.ylabel(“SS_Score”)

plt.title(“Result with StandartScaler”)
```

Output:

Text(0.5, 1.0, ‘Result with StandartScaler’)



Evaluation:

Evaluating an AI-based diabetes prediction system is essential to assess its performance and reliability. Here are some common evaluation metrics and methods:

1. Accuracy: This measures the overall correctness of predictions. It is the ratio of correctly predicted instances to the total instances in the dataset. However, accuracy can be misleading if the dataset is imbalanced.

2. Precision: Precision measures the ratio of true positive predictions to the total positive predictions. It indicates how many of the positive predictions are correct. Precision is particularly important in healthcare applications where false positives can have serious consequences.

3. Recall (Sensitivity): Recall measures the ratio of true positive predictions to the total actual positives in the dataset. It indicates how many of the actual positive cases were correctly predicted. High recall is crucial to ensure that actual cases are not missed.

4. F1-Score: The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall, which is useful when you need to consider both false positives and false negatives.

5. Area Under the Receiver Operating Characteristic Curve (ROC-AUC): ROC-AUC measures the ability of the model to distinguish between positive and negative cases. It's especially relevant when there is class imbalance in the dataset.

6. Confusion Matrix: A confusion matrix provides a detailed breakdown of true positives, true negatives, false positives, and false negatives, offering insights into where the model is making errors.

7. Specificity: Specificity measures the ratio of true negative predictions to the total actual negatives in the dataset. It indicates how well the model correctly predicts negative cases.

8. Mean Absolute Error (MAE) and Root Mean Square Error (RMSE): For regression-based models, these metrics evaluate the accuracy of predicted numerical values like glucose levels.

9. Receiver Operating Characteristic (ROC) Curve: This graphical representation shows the trade-off between sensitivity and specificity at various thresholds. It helps in choosing the appropriate threshold for a given application.

Input:

```
Evaluation = pd.DataFrame({'Model': names,
                           'Score': roc_list})
Evaluation = evaluation.sort_values(by='Score', ascending=False)
evaluation
```

Output:

	Model	Score
5	Catboost	0.811744
6	Lightgbm	0.811317
7	Xgboost	0.810692
3	RandomForest	0.801506
0	Logistic_Regression	0.793212
4	Decision_Tree	0.789043
1	Guassian	0.765399
2	ExtraTree	0.741116

Conclusion:

☆ In conclusion, building an AI-based diabetes prediction system involves several key stages, including the selection of machine learning models, training these models, and evaluating their performance. Here's a summary of each stage:

Machine Learning Models Selection:

Choosing the right machine learning models is crucial. Common choices include decision trees, random forests, support vector machines, logistic regression, and neural networks.The selection of models depends on the complexity of the problem, the type of data available, and the specific requirements of the healthcare application.

Training the Models:

Model training involves using a portion of the dataset to train the chosen model, optimizing hyperparameters, and ensuring the model generalizes well to unseen data.

Evaluating Model Performance:

Model performance is assessed using various metrics and methods, including accuracy, precision, recall, F1-score, ROC-AUC, and the confusion matrix.Cross-validation techniques are used to validate the model's performance on multiple subsets of the data.

☆In the development of an AI-based diabetes prediction system, it's essential to collaborate with healthcare professionals, data privacy experts, and domain specialists to ensure the system's accuracy, safety, and compliance with healthcare regulations. Regular monitoring and updates are necessary to keep the system relevant and reliable, given the dynamic nature of healthcare data and patient populations.