AI Based Diabetes Prediction System

Phase-4 submission document

Phase -4:Development part 2

An AI diabetes prediction system is a cutting-edge technological solution designed to help healthcare professionals, individuals, and patients manage diabetes more effectively by predicting the likelihood of developing diabetes or identifying the risk factors associated with the disease. Diabetes is a chronic condition characterized by high blood sugar levels and can lead to serious health complications if not properly managed. AI-driven prediction systems leverage advanced algorithms and data analysis to offer several benefits, including early detection, personalized recommendations, and improved patient outcomes.

Given data set:

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes Pedigree Function	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
…	….	…	…	…	…	…	…	…

6	190	92	0	0	35.5	0.278	66	1
2	88	58	26	16	28.4	0.766	22	0
9	170	74	31	0	44	0.403	43	1
9	89		0	0	22.5	0.142	33	0
10	101	76	48	180	32.9	0.171	63	0
2	122	70	27	0	36.8	0.34	27	0
5	121	72	23	112	26.2	0.245	30	0
1	126	60	0	0	30.1	0.349	47	1
1	93	70	31	0	30.4	0.315	23	0

1**. Data Collection and Integration:** These systems gather diverse data from sources like electronic health records, medical history, lifestyle habits, genetic information, and wearable devices. The data is aggregated, organized, and prepared for analysis.

2. Machine Learning Algorithms: AI models, including machine learning and deep learning algorithms, process the collected data to recognize patterns, relationships, and risk factors associated with diabetes. These models continuously learn and improve their predictive accuracy over time.

3. Risk Assessment: The system assesses an individual's risk of developing diabetes based on their unique profile. It considers factors like family history, age, weight, dietary habits, physical activity, blood glucose levels, and other health parameters.

4. Early Detection: AI models enable early detection of diabetes risk, allowing for timely intervention and preventive measures. Early diagnosis and management can help individuals reduce the progression of the disease and its complications.

5. Personalized Recommendations: The system provides personalized recommendations for individuals based on their risk assessment. These recommendations may include dietary changes, exercise plans, and other lifestyle modifications tailored to an individual's specific needs.

6. Monitoring and Feedback: Some AI diabetes prediction systems offer continuous monitoring and feedback. Patients can track their progress and receive real-time guidance for managing their health effectively.

7.Healthcare Provider Support: Healthcare professionals can utilize the system to receive alerts and insights on their patients, helping them deliver more individualized care and treatment options. The system may suggest appropriate tests, medication adjustments, or lifestyle interventions.

Machine learning algorithm:

1.Logistic regression:

```
# Import necessary libraries

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


# Load your dataset (replace 'diabetes_data.csv' with your data file)
```

```
data = pd.read_csv('diabetes_data.csv')

# Split the data into features and target
X = data.drop('diabetes', axis=1)  # Features
y = data['diabetes']  # Target variable

# Split the data into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features (optional but recommended)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(classification_rep)
```

2.Ridge regression:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score

# Load your dataset (replace 'diabetes_data.csv' with your data file)
data = pd.read_csv('diabetes_data.csv')

# Split the data into features and target
X = data.drop('target', axis=1)  # Features
y = data['target']  # Target variable

# Split the data into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features (optional but recommended)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the Ridge regression model
alpha = 1.0  # Ridge regularization parameter (you can adjust this)
model = Ridge(alpha=alpha)
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)
```

```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the results
print(f"Mean Squared Error: {mse}")
print(f"R-squared (R2) Score: {r2}")
```

3.Lasso regression:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score

# Load your dataset (replace 'diabetes_data.csv' with your data file)
data = pd.read_csv('diabetes_data.csv')

# Split the data into features and target
X = data.drop('target', axis=1)  # Features
y = data['target']  # Target variable

# Split the data into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features (optional but recommended)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the Lasso regression model
```

```
alpha = 1.0  # Lasso regularization parameter (you can adjust this)
model = Lasso(alpha=alpha)
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the results
print(f"Mean Squared Error: {mse}")
print(f"R-squared (R2) Score: {r2}")
```

5.Elastic net:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error, r2_score

# Load your dataset (replace 'diabetes_data.csv' with your data file)
data = pd.read_csv('diabetes_data.csv')

# Split the data into features and target
X = data.drop('target', axis=1)  # Features
y = data['target']  # Target variable

# Split the data into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Standardize features (optional but recommended)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the Elastic Net model
alpha = 1.0  # Regularization parameter (you can adjust this)
l1_ratio = 0.5  # Mixing parameter between L1 and L2 regularization (you can adjust this)
model = ElasticNet(alpha=alpha, l1_ratio=l1_ratio)
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the results
print(f"Mean Squared Error: {mse}")
print(f"R-squared (R2) Score: {r2}")
```

6.Support vector machines:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load your dataset (replace 'diabetes_data.csv' with your data file)
```

```
data = pd.read_csv('diabetes_data.csv')

# Split the data into features and target
X = data.drop('diabetes', axis=1)  # Features
y = data['diabetes']  # Target variable

# Split the data into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features (recommended for SVM)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Create and train the SVM model
model = SVC(kernel='linear', C=1.0)  # You can choose different kernels and C values
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(classification_rep)
```

7.Random forest regression:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Load your dataset (replace 'diabetes_data.csv' with your data file)
data = pd.read_csv('diabetes_data.csv')

# Split the data into features and target
X = data.drop('target', axis=1)  # Features
y = data['target']  # Target variable

# Split the data into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Random Forest regression model
n_estimators = 100  # Number of trees in the forest (you can adjust this)
model = RandomForestRegressor(n_estimators=n_estimators, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the results
print(f"Mean Squared Error: {mse}")
```

print(f"R-squared (R2) Score: {r2}")

Model training:

Model training is a critical step in building an AI-based diabetes prediction system. In this step, you'll use your dataset to teach a machine learning model how to make predictions. Below, I'll provide a general outline of the model training process for a diabetes prediction system:

1. Data Preprocessing:

- Ensure your dataset is clean and well-structured. Handle missing values, outliers, and encode categorical features if necessary. Standardize or normalize numerical features to ensure consistency.

2. Data Splitting:

- Divide your dataset into two subsets: a training set and a testing set. The training set is used to teach the model, while the testing set is used to evaluate its performance. Common split ratios are 70/30, 80/20, or 90/10.

3. Feature Selection/Engineering (if not already done):

- Choose relevant features or perform feature engineering to create new features that might improve the model's predictive power.

4. Model Selection:

- Select an appropriate machine learning algorithm for your diabetes prediction task. Common choices include logistic regression, decision trees, random forests, support vector machines, neural networks, and more.

5. Model Training:

- Train the selected model on the training data. The model learns the underlying patterns and relationships between the features and the target variable (diabetes).

Dividing data setv in to features and target variable:

```
import pandas as pd


# Load your dataset
data = pd.read_csv('diabetes_data.csv')


# Define features (independent variables) and target variable (dependent variable)
```

X = data.drop('diabetes', axis=1) # Features (all columns except 'diabetes')

y = data['diabetes'] # Target variable ('diabetes' column)

You can also print the first few rows of X and y to verify

print(X.head()) # Features

print(y.head()) # Target variable

Model evaluation:

Model evaluation is a crucial step in building an AI-based diabetes prediction system. It helps you assess the performance and reliability of your model. Here are some common evaluation metrics and techniques for evaluating the performance of your diabetes prediction model:

1. Accuracy: For binary classification problems (e.g., diabetes/no diabetes), accuracy is a commonly used metric. It calculates the ratio of correctly predicted instances to the total number of instances in the testing dataset.

```python
from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred)
```

2. Precision and Recall: Precision and recall are important when dealing with imbalanced datasets. Precision measures the proportion of true positive predictions among all positive predictions, while recall measures the proportion of true positive predictions among all actual positives.

```python
from sklearn.metrics import precision_score, recall_score

precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
```

3. F1-Score: The F1-score is the harmonic mean of precision and recall and is especially useful when you want to balance precision and recall. It provides a single metric that considers both false positives and false negatives.

```python
from sklearn.metrics import f1_score

f1 = f1_score(y_test, y_pred)
```

4. Confusion Matrix: A confusion matrix provides a detailed breakdown of the model's performance. It shows the number of true positives, true negatives, false positives, and false negatives.

```python
from sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(y_test, y_pred)
```

5. Receiver Operating Characteristic (ROC) and Area Under the ROC Curve (AUC-ROC): ROC curves are useful for evaluating the model's ability to discriminate between classes. AUC-ROC summarizes the overall performance of a binary classification model.

```python
from sklearn.metrics import roc_curve, roc_auc_score

fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)
```

6. Mean Squared Error (MSE): For regression problems, where you're predicting continuous variables (e.g., blood glucose levels), you can use MSE to evaluate the accuracy of the model's predictions. Lower MSE values indicate better performance.

```python
from sklearn.metrics import mean_squared_error

mse = mean_squared_error(y_test, y_pred)
```

7. R-squared (R2) Score: R2 measures the proportion of the variance in the target variable that is predictable from the independent variables. It is commonly used for regression model evaluation.

```python
from sklearn.metrics import r2_score

r2 = r2_score(y_test, y_pred)
```

8. Cross-Validation: To ensure the model's generalization performance, use techniques like k-fold cross-validation, where the dataset is divided into k subsets. This helps in estimating the model's performance on unseen data.

9. Hyperparameter Tuning: Fine-tuning hyperparameters can significantly impact the model's performance. Use techniques like grid search or randomized search to find the best hyperparameter values.

Evaluate your diabetes prediction model using these metrics and techniques to ensure it performs well on real-world data. Depending on your specific use case, you may prioritize different metrics to balance precision, recall, and overall accuracy.

Conclusion:

Developing an AI-based diabetes prediction system is a complex and ongoing process that requires multidisciplinary collaboration and a strong commitment to ethical and responsible AI development. When done effectively, it has the potential to improve early diagnosis, patient outcomes, and the efficiency of healthcare systems.