**AI BASED DIABETES PREDICTION SYSTEM**

**1. Data Collection**:

- Obtain a dataset related to diabetes. A popular one is the Pima Indians Diabetes dataset.

- Ensure your dataset includes features such as age, glucose levels, insulin levels, BMI, etc., and a target variable indicating if the person has diabetes.

2. **Data Preprocessing:**

- Handle missing values.

- Normalize or standardize the data (for instance, using MinMaxScaler or StandardScaler from sklearn).

- Split the dataset into training and testing sets.

3. **Selecting a Machine Learning Algorithm**:

- For a start, you can use algorithms like Logistic Regression, Random Forest, Gradient Boosting, or Support Vector Machine (SVM).

- Consider using libraries like `scikit-learn` for implementation.

4. **Training the Model:**

- Train the selected model using the training data.

```python

From sklearn.ensemble import RandomForestClassifier

Classifier = RandomForestClassifier()

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

```

5. **Evaluation**:

- Evaluate the model’s performance on the test data.

- Use metrics like accuracy, precision, recall, F1-score, and ROC-AUC to gauge performance.

From sklearn.metrics import accuracy\_score

Y\_pred = classifier.predict(X\_test)

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

Print(“Accuracy:”, accuracy)

6. **Hyperparameter Tuning:**

- Depending on the algorithm you choose, you might want to adjust its parameters to optimize performance.

- Consider using tools like GridSearchCV or RandomizedSearchCV in `scikit-learn`.

7. **Deployment**:

- Once satisfied with the model’s performance, deploy it using platforms like Flask, Django, or FastAPI for web-based interfaces.

- Alternatively, integrate the model within mobile apps or other software applications.

8. **Feedback Loop:**

- Continuously collect data and feedback to retrain and improve the model.

9. **Ethical Considerations:**

- Ensure your system is transparent about its predictions, and users understand that while your system might be a helpful tool, it shouldn’t replace advice from healthcare professionals.

**CODE :**

# 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

# Load dataset (assuming it’s in a file called “diabetes.csv”)

Data = pd.read\_csv(“diabetes.csv”)

# Split dataset into predictors (X) and target (y)

X = data.drop(“Outcome”, axis=1)

Y = data[“Outcome”]

# Split data into training and test sets

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

# Standardize the data

Scaler = StandardScaler()

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

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

# Train a logistic regression model

Clf = LogisticRegression()

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

# Predict on the test set

Y\_pred = clf.predict(X\_test)

# Evaluate the model

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

Report = classification\_report(y\_test, y\_pred)

Print(f”Accuracy: {accuracy}”)

Print(“Classification Report:”)

Print(report)

**MACHINE LEARNING APPROACH:**

1. **Data Collection:**

Gather data on patients, both diabetic and non-diabetic. Features might include age, gender, BMI, glucose level, insulin level, family history, etc.

2. **Data Cleaning:**

- Handle missing values (impute or remove)

- Remove duplicates

- Handle outliers (based on domain knowledge)

3. **Feature Engineering:**

- Normalize/standardize data

- Create new features (e.g., BMI categories)

- Reduce dimensionality using PCA or feature selection methods if necessary

4. **Data Splitting**:

Divide the dataset into training and testing sets. A common split ratio is 80% for training and 20% for testing. Optionally, use a validation set or cross-validation.

5. **Model Selection**:

- Start with simple models like Logistic Regression.

- Move on to more complex models such as Random Forest, Gradient Boosting Machines, Neural Networks, etc.

- Use tools like GridSearchCV for hyperparameter tuning.

6. **Training**:

Use the training data to train your chosen model.

7. **Evaluation**: Use the test set to evaluate model performance. Common metrics for classification problems are accuracy, precision, recall, F1-score, ROC curve, and AUC.

8. **Model Interpretability**: For medical applications, understanding why a model makes a certain prediction can be crucial. Tools like SHAP (Shapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) can help.

9. **Deployment**:

Once satisfied with the model’s performance, deploy it as an API or integrate it into a healthcare system for real-time predictions.

10. **Monitoring and Maintenance:**

Continually monitor the model’s performance in real-world scenarios and retrain it periodically with fresh data.