**Introduction**:

Diabetes is a widespread chronic health condition that affects millions of people worldwide. Early detection and prediction of diabetes risk are crucial for effective disease management and prevention. Artificial Intelligence (AI) has emerged as a powerful tool in healthcare, offering the potential to improve the accuracy and efficiency of diabetes prediction. In this project, we will outline the key steps involved in building an AI-based diabetes prediction system, including feature engineering, model training, and evaluation.

**Feature** **Engineering**:

Feature engineering is the process of selecting and transforming relevant data attributes (features) to improve the performance of a machine learning model. In the context of diabetes prediction, it involves identifying the most significant features that can influence the risk of diabetes. These features can include:

- Patient Demographics: Age, gender, ethnicity.

- Medical History: Family history of diabetes, previous diagnoses, and comorbidities.

- Anthropometric Data: Body Mass Index (BMI), waist circumference.

- Lifestyle Factors: Diet, physical activity, and smoking status.

- biometric Measurements: Blood pressure, cholesterol levels, and glucose levels.

Additionally, feature engineering may involve dealing with missing data, normalizing or scaling features, and creating new features derived from existing ones.

**Model** **Training**:

Once the feature set is defined, the next step is to choose an appropriate machine learning model to predict diabetes risk. Some common models for this purpose include:

- Logistic Regression: Suitable for binary classification problems.

- Random Forest: Effective for handling complex, non-linear relationships.

- Support Vector Machine (SVM):Good for separating data into distinct classes.

- Neural Networks: Deep learning models can capture intricate patterns in data.

The dataset should be split into training and testing sets to train the model and evaluate its performance. Hyper parameter tuning and cross-validation may be necessary to optimize the model’s accuracy and generalizability.

**Evaluation**:

Evaluating the model’s performance is crucial to ensure its effectiveness. Common evaluation metrics for a diabetes prediction model include:

-Accuracy: The proportion of correct predictions.

-Precision and Recall: Precision measures the proportion of true positive predictions among all positive predictions, while recall assesses the proportion of true positives among actual positive cases.

- F1-Score: The harmonic mean of precision and recall.

- Receiver Operating Characteristic (ROC) Curve and Area Under the Curve (AUC): Assess the trade-off between true positive rate and false positive rate.

Additionally, model performance can be validated using techniques like k-fold cross-validation to ensure its stability and robustness.

In this project, we will walk through the process of creating an AI-based diabetes prediction system, demonstrating how to perform feature engineering, train a machine learning model, and evaluate its performance to help healthcare professionals in early diabetes risk assessment and patient care.

**Dataset link**

<https://www.kaggle.com/datasets/mathchi/diabetes-data-set>

Feature engineering is a crucial step in building an AI-based diabetes prediction system. Here’s a brief overview of the theory and coding involved:

**Feature** **Selection**:

**Domain** **Knowledge**: Start by understanding the domain. For diabetes prediction, common features include age, BMI, family history, blood pressure, and glucose levels.

**Feature** **Importance**: Use techniques like correlation analysis or tree-based methods to identify the most important features.

**Feature** **Transformation**:

Scaling: Standardize or normalize numerical features to have a common scale, e.g., using Scikit-Learn’s `StandardScaler` or `MinMaxScaler`.

Categorical Encoding: Convert categorical features (e.g., gender) into numerical representations, like one-hot encoding or label encoding using libraries like Scikit-Learn or pandas.

**Feature** **Creation**:

Polynomial Features: Generate polynomial features to capture non-linear relationships using Scikit-Learn’s `Polynomial Features`.

Interaction Terms: Create interaction features to capture relationships between two or more features.

**Feature** **Selection** **Techniques**:

**Filter** **Methods**: Use statistical measures (e.g., chi-squared, mutual information) to select relevant features.

Wrapper Methods: Use methods like recursive feature elimination (RFE) with cross-validation to select the best subset of features.

Coding in Python (using Scikit-Learn):

```python

Import pandas as pd

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

From sklearn.preprocessing import StandardScaler

From sklearn.feature\_selection import SelectKBest

From sklearn.ensemble import RandomForestClassifier

# Load your dataset

Data = pd.read\_csv(“diabetes\_data.csv”)

# Split data into features (X) and target (y)

X = data. Drop(“diabetes”, axis=1)

Y = data[“diabetes”]

# Split data 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)

# Feature scaling

Scaler = StandardScaler()

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

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

# Feature selection

Feature\_selector = SelectKBest(k=5)

X\_train\_selected = feature\_selector.fit\_transform(X\_train, y\_train)

X\_test\_selected = feature\_selector.transform(X\_test)

# Train a machine learning model (e.g., RandomForest) on the selected features

Model = RandomForestClassifier()

Model.fit(X\_train\_selected, y\_train)

# Evaluate the model on the test set

Accuracy = model.score(X\_test\_selected, y\_test)

```

This is a simplified example. You should customize it based on your dataset and the specific algorithms you plan to use for diabetes prediction. Feature engineering can significantly impact your model’s performance, so it’s essential to experiment with different techniques to find the best set of features.

Training an AI-based diabetes prediction system involves selecting a machine learning or deep learning model, preparing your data, and optimizing the model for accuracy. Here’s a general overview of the theory and coding involved:

**Model** **Selection**:

Algorithm Choice: Choose an appropriate algorithm for classification. Common choices include logistic regression, decision trees, random forests, support vector machines, or deep learning models like neural networks.

Model Evaluation: Consider metrics like accuracy, precision, recall, F1-score, and AUC-ROC to evaluate model performance.

**Data** **Preparation**:

Data Cleaning: Handle missing values, outliers, and ensure data quality.

Data Split: Split your dataset into training and testing sets for model validation. You can also use techniques like cross-validation.

Feature Scaling: Normalize or standardize numerical features to ensure they have the same scale.

Model Training and Coding (using Python):

Here’s an example using a popular classification algorithm, Random Forest:

```python

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 pre-processed dataset

Data = pd.read\_csv(“diabetes\_data.csv”)

# Split data into features (X) and target (y)

X = data. Drop(“diabetes”, axis=1)

Y = data[“diabetes”]

# Split data 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)

# Initialize and train the model

Model = RandomForestClassifier()

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

# Make predictions on the test set

Y\_pred = model. Predict(X\_test)

# Evaluate the model’s accuracy

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

Print(“Accuracy:”, accuracy)

```

**Data** **Modelling**

# Logistic Regression Algorithm

From sklearn.linear\_model import LogisticRegression

Logres = LogisticRegression(random\_state = 42)

Logreg.fit(X\_train, Y\_train)

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

Intercept\_scaling=1, max\_iter=100, multi\_class=’warn’,

N\_jobs=None, penalty=’l2’, random\_state=42, solver=’warn’,

Tol=0.0001, verbose=0, warm\_start=False)

# Plotting a graph for n\_neighbors

From sklearn import metrics

From sklearn.neighbors import KNeighborsClassifier

X\_axis = list(range(1, 31))

Acc = pd.Series()

X = range(1,31)

For I in list(range(1, 31)):

Knn\_model = KNeighborsClassifier(n\_neighbors = i)

Knn\_model. Fit(X\_train, Y\_train)

Prediction = knn\_model.predict(X\_test)

Acc = acc.append(pd.Series(metrics. Accuracy\_score(prediction, Y\_test)))

Plt.plot(X\_axis, acc)

Plt.xticks(x)

Plt.title(“Finding best value for n\_estimators”)

Plt.xlabel(“n\_estimators”)

Plt.ylabel(“Accuracy”)

Plt.grid()

Plt.show()

Print(‘Highest value: ‘,acc.values.max())

Highest value: 0.7857142857142857

# K nearest neighbors Algorithm

From sklearn.neighbors import KNeighborsClassifier

Knn = KNeighborsClassifier(n\_neighbors = 24, metric = ‘minkowski’, p = 2)

Knn. Fit(X\_train, Y\_train)

KNeighborsClassifier(algorithm=’auto’, leaf\_size=30, metric=’minkowski’,

Metric\_params=None, n\_jobs=None, n\_neighbors=24, p=2,

Weights=’uniform’)

# Support Vector Classifier Algorithm

From sklearn.svm import SVC

Svc = SVC(kernel = ‘linear’, random\_state = 42)

Svc.fit(X\_train, Y\_train)

SVC(C=1.0, cache\_size=200, class\_weight=None, coef0=0.0,

Decision\_function\_shape=’ovr’, degree=3, gamma=’auto\_deprecated’,

Kernel=’linear’, max\_iter=-1, probability=False, random\_state=42,

Shrinking=True, tol=0.001, verbose=False)

# Naïve Bayes Algorithm

From sklearn.naive\_bayes import GaussianNB

Nb = GaussianNB()

Nb.fit(X\_train, Y\_train)

GaussianNB(priors=None, var\_smoothing=1e-09)

# Decision tree Algorithm

From sklearn.tree import DecisionTreeClassifier

Dectree = DecisionTreeClassifier(criterion = ‘entropy’, random\_state = 42)

Dectree. Fit(X\_train, Y\_train)

DecisionTreeClassifier(class\_weight=None, criterion=’entropy’, max\_depth=None,

Max\_features=None, max\_leaf\_nodes=None,

Min\_impurity\_decrease=0.0, min\_impurity\_split=None,

Min\_samples\_leaf=1, min\_samples\_split=2,

Min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=42,

Splitter=’best’)

# Random forest Algorithm

From sklearn.ensemble import RandomForestClassifier

Ranfor = RandomForestClassifier(n\_estimators = 11, criterion = ‘entropy’, random\_state = 42)

Ranfor. Fit(X\_train, Y\_train)

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion=’entropy’,

Max\_depth=None, max\_features=’auto’, max\_leaf\_nodes=None,

Min\_impurity\_decrease=0.0, min\_impurity\_split=None,

Min\_samples\_leaf=1, min\_samples\_split=2,

Min\_weight\_fraction\_leaf=0.0, n\_estimators=11, n\_jobs=None,

Oob\_score=False, random\_state=42, verbose=0, warm\_start=False)

# Making predictions on test dataset

Y\_pred\_logreg = logreg.predict(X\_test)

Y\_pred\_knn = knn.predict(X\_test)

Y\_pred\_svc = svc.predict(X\_test)

Y\_pred\_nb = nb.predict(X\_test)

Y\_pred\_dectree = dectree.predict(X\_test)

Y\_pred\_ranfor = ranfor.predict(X\_test)

Remember, you can replace `RandomForestClassifier` with other classifiers or deep learning frameworks like TensorFlow or PyTorch for more complex models. Hyperparameter tuning, cross-validation, and handling class imbalances are additional considerations to improve model performance. Ensure you preprocess and format your data according to the specific requirements of the chosen algorithm or framework.

Evaluating an AI-based diabetes prediction system is essential to ensure its accuracy and effectiveness. Here’s a general overview of the theory and coding involved:

**Model** **Evaluation** **Metrics**:

Accuracy: It measures the overall correctness of predictions.

Precision: It quantifies how many of the positive predictions were correct.

Recall (Sensitivity): It measures how many of the actual positive cases were predicted correctly.

F1-Score:The harmonic mean of precision and recall, balancing both metrics.

Area Under the ROC Curve (AUC-ROC): It measures the model’s ability to distinguish between positive and negative cases.

**Coding in** **Python**:

Here’s an example of how to evaluate a diabetes prediction model using these metrics:

```python

From sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score

# Assuming you have predictions (y\_pred) and true labels (y\_true)

Y\_true = [0, 1, 1, 0, …] # True labels

Y\_pred = [0, 1, 0, 1, …] # Predictions

# Calculate metrics

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

Precision = precision\_score(y\_true, y\_pred)

Recall = recall\_score(y\_true, y\_pred)

F1 = f1\_score(y\_true, y\_pred)

Roc\_auc = roc\_auc\_score(y\_true, y\_pred)

# Print the results

Print(“Accuracy:”, accuracy)

Print(“Precision:”, precision)

Print(“Recall:”, recall)

Print(“F1 Score:”, f1)

Print(“AUC-ROC Score:”, roc\_auc)

```

Evaluating using accuracy\_score metric

From sklearn.metrics import accuracy\_score

Accuracy\_logreg = accuracy\_score(Y\_test, Y\_pred\_logreg)

Accuracy\_knn = accuracy\_score(Y\_test, Y\_pred\_knn)

Accuracy\_svc = accuracy\_score(Y\_test, Y\_pred\_svc)

Accuracy\_nb = accuracy\_score(Y\_test, Y\_pred\_nb)

Accuracy\_dectree = accuracy\_score(Y\_test, Y\_pred\_dectree)

Accuracy\_ranfor = accuracy\_score(Y\_test, Y\_pred\_ranfor)

# Accuracy on test set

Print(“Logistic Regression: “ + str(accuracy\_logreg \* 100))

Print(“K Nearest neighbors: “ + str(accuracy\_knn \* 100))

Print(“Support Vector Classifier: “ + str(accuracy\_svc \* 100))

Print(“Naïve Bayes: “ + str(accuracy\_nb \* 100))

Print(“Decision tree: “ + str(accuracy\_dectree \* 100))

Print(“Random Forest: “ + str(accuracy\_ranfor \* 100))

Logistic Regression: 71.42857142857143

K Nearest neighbors: 78.57142857142857

Support Vector Classifier: 73.37662337662337

Naïve Bayes: 71.42857142857143

Decision tree: 68.18181818181817

Random Forest: 75.97402597402598

#From the above comparison, we can observe that K Nearest neighbors gets the highest accuracy of 78.57 %

# Confusion matrix

From sklearn.metrics import confusion\_matrix

Cm = confusion\_matrix(Y\_test, Y\_pred\_knn)

Cm

Array([[87, 13],

[20, 34]], dtype=int64)

# Heatmap of Confusion matrix

Sns.heatmap(pd.DataFrame(cm), annot=True)

<matplotlib.axes.\_subplots.AxesSubplot at 0x1976620cb70>

# Classification report

From sklearn.metrics import classification\_report

Print(classification\_report(Y\_test, Y\_pred\_knn))

Precision recall f1-score support

0.0 0.81 0.87 0.84 100

1.0 0.72 0.63 0.67 54

Micro avg 0.79 0.79 0.79 154

Macro avg 0.77 0.75 0.76 154

Weighted avg 0.78 0.79 0.78 154

Remember that you should replace the `y\_true` and `y\_pred` lists with your actual ground truth labels and model predictions. These metrics provide a comprehensive evaluation of your model's performance, allowing you to make informed decisions on its effectiveness.