PREDICTING DIABETES SYSTEM USING MACHINE LEARNING

TEAM LEADER: 922321106033

X. SHOBIA ROSALIN MARRIE

Phase - 4 submission document

Project title: Diabetes Prediction System

Phase-4: Development part 2

Topic: Continue predicting the diabetes prediction model by feature engineering, model training and evaluation.

INTRODUCTION:

A diabetes prediction system involves using machine learning techniques to predict whether an individual is likely to develop diabetes based on certain features or risk factors. Here's an overview of the key components involved:

1)Feature Engineering:

Feature Selection: Identify the most important features through techniques like correlation analysis or feature importance scores.

Feature Scaling: Normalize or standardize the features to ensure that they have similar scales.

2)Model Selection and Training: Choose an appropriate machine learning algorithm for classification tasks. Common choices include logistic regression, decision trees, random forests, support vector machines, or deep learning models like neural networks.

3)Model Evaluation: Assess the model's performance using various evaluation metrics, such as accuracy, precision, recall, F1-score, and the area under the ROC curve.

- Utilize techniques like cross-validation to ensure the model's robustness and avoid overfitting.

- Visualize and interpret the results to gain insights into the model's behavior.
- Hyper parameter Tuning
- Continuous Monitoring
- Ethical Considerations

A successful diabetes prediction system should not only be accurate but also interpretable and considerate of ethical implications, especially when applied in real-world healthcare scenarios.

GIVEN DATA SET:

	A	B	C	D	E	F	G	H	I
1	Pregnanci	Glucose	BloodPres	SkinThickr	Insulin	BMI	DiabetesF	Age	Outcome
2	6	148	72	35	0	33.6	0.627	50	1
3	1	85	66	29	0	26.6	0.351	31	0
4	8	183	64	0	0	23.3	0.672	32	1
5	1	89	66	23	94	28.1	0.167	21	0
6	0	137	40	35	168	43.1	2.288	33	1
7	5	116	74	0	0	25.6	0.201	30	0
8	3	78	50	32	88	31	0.248	26	1
9	10	115	0	0	0	35.3	0.134	29	0
10	2	197	70	45	543	30.5	0.158	53	1
11	8	125	96	0	0	0	0.232	54	1
12	4	110	92	0	0	37.6	0.191	30	0
13	10	168	74	0	0	38	0.537	34	1
14	10	139	80	0	0	27.1	1.441	57	0
15	1	189	60	23	846	30.1	0.398	59	1
16	5	166	72	19	175	25.8	0.587	51	1
17	7	100	0	0	0	30	0.484	32	1
18	0	118	84	47	230	45.8	0.551	31	1
19	7	107	74	0	0	29.6	0.254	31	1
20	1	103	30	38	83	43.3	0.183	33	0
21	1	115	70	30	96	34.6	0.529	32	1
22	3	126	88	41	235	39.3	0.704	27	0
23	8	99	84	0	0	35.4	0.388	50	0
24	7	196	90	0	0	39.8	0.451	41	1

769 Rows*9 Columns

DATASET LINK:

https://1drv.ms/x/s!AhQBfjMJZbYSi2Z8ifTdYFtnhilO

FEATURE ENGINEERING:

Feature engineering in a diabetes prediction system involves selecting and transforming relevant attributes (features) from the dataset to improve the accuracy of the predictive model. This process is critical in developing an effective diabetes prediction system. Some key feature engineering techniques include:

- **Feature Selection:** Identifying the most informative features and excluding irrelevant ones can reduce noise in the data and improve model performance. Techniques like correlation analysis and feature importance ranking can help in this process.
- **Feature Scaling:** Normalizing or standardizing features to a common scale (e.g., mean of 0 and standard deviation of 1) ensures that no single feature dominates the model due to its scale.
- **Feature Transformation:** This includes techniques such as logarithmic transformations or polynomial features to handle non-linearity in the data and make it more suitable for certain algorithms.
- **Handling Missing Data:** Addressing missing values through imputation methods, such as

mean, median, or advanced techniques like K-nearest neighbors, is crucial for a robust model.

- **One-Hot Encoding:** Converting categorical features into numerical representations (binary vectors) enables the model to process them effectively.
- **Feature Engineering for Time Series Data:** For diabetes prediction systems that involve time-series data, creating lag features, rolling statistics, and temporal aggregations can capture relevant trends and patterns.
- **Domain-Specific Features:** Incorporating domain knowledge can be essential. For example, in diabetes prediction, creating features related to dietary habits, physical activity, or medical history can be valuable.

The effectiveness of feature engineering in a diabetes prediction system can significantly impact the model's performance and its ability to make

accurate predictions about an individual's risk of developing diabetes.

Example 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 'diabetes_data.csv' with your dataset)

Data = pd.read_csv('diabetes_data.csv')

# Feature engineering (you may need more extensive feature engineering)

# For this example, we'll use a few common features

Features = data[['Glucose', 'BMI', 'Age', 'Insulin']]

# Target variable

Target = data['Diabetes']

# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# Create a machine learning model (Random Forest in this case)

Model = RandomForestClassifier()

# Train the model

Model.fit(X_train, y_train)

# Make predictions on the test data
```

```
Y_pred = model.predict(X_test)

# Calculate accuracy

Accuracy = accuracy_score(y_test, y_pred)

Print(f'Accuracy: {accuracy * 100:.2f}%')

# Example prediction for a new data point

New_data_point = [[150, 30, 45, 50]] # Replace with your own values

Prediction = model.predict(new_data_point)

Print(f'Prediction for new data point: {"Diabetic" if prediction[0] == 1 else "Not Diabetic"}')
```

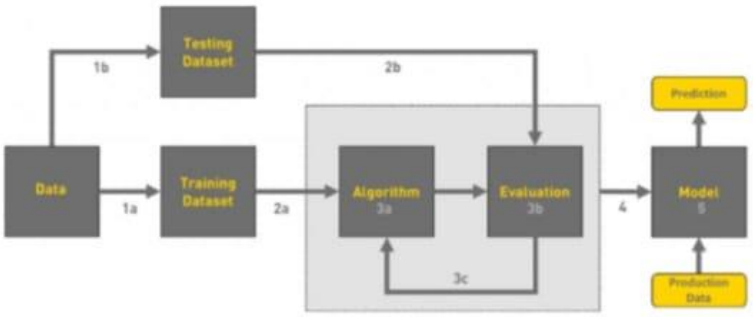
MODEL EVALUATION:

Model evaluation in a diabetes prediction system using machine learning is crucial to assess the performance and reliability of the model. Common evaluation techniques include:

- **Accuracy:** This measures the percentage of correctly predicted instances. However, it may not be ideal if the dataset is imbalanced.
- **Precision and Recall:** Precision indicates the proportion of true positive predictions among all positive predictions, while recall measures the proportion of true positives among actual

positives. These metrics are useful when false positives or false negatives carry different costs.

- **F1 Score:** The F1 score is the harmonic mean of precision and recall, providing a balance between the two.
- **ROC and AUC:** Receiver Operating Characteristic (ROC) curves and Area Under the Curve (AUC) can help evaluate the trade-off between true positive rate and false positive rate at different thresholds.
- **Confusion Matrix:** This matrix provides a detailed breakdown of true positives, true negatives, false positives, and false negatives.
- **Cross-Validation:** Techniques like k-fold cross-



validation help assess a model's generalization performance, reducing the risk of overfitting.

- **Mean Squared Error (MSE) and Mean Absolute Error (MAE):** These are used when the prediction is a continuous variable, like blood glucose levels.

- **Receiver Operating Characteristic (ROC) Analysis:** It's beneficial in binary classification problems to choose an appropriate threshold for classifying instances.
- **Feature Importance Analysis:** Understanding which features contribute most to the model's predictions can provide insights into the underlying data.
- **Domain Expert Evaluation:** In healthcare applications like diabetes prediction, involving domain experts to assess the clinical relevance and interpretability of the model is crucial.

These evaluation techniques help ensure that the diabetes prediction model is accurate, reliable, and suitable for its intended purpose while considering factors like class imbalance, clinical significance, and the type of data being predicted.

Example Program:

```
# Calculate evaluation metrics
Accuracy = accuracy_score(y_test, y_pred)
Precision = precision_score(y_test, y_pred)
Recall = recall_score(y_test, y_pred)
F1 = f1_score(y_test, y_pred)
Confusion = confusion_matrix(y_test, y_pred)
Print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Print(f'Precision: {precision * 100:.2f}%')

Print(f'Recall: {recall * 100:.2f}%')

Print(f'F1 Score: {f1 * 100:.2f}%')

Print(f'Confusion Matrix:\n{confusion}')

# Example prediction for a new data point

New_data_point = [[150, 30, 45, 50]] # Replace with your own values

Prediction = model.predict(new_data_point)

Print(f'Prediction for new data point: {"Diabetic" if prediction[0] == 1 else "Not Diabetic"}')
```

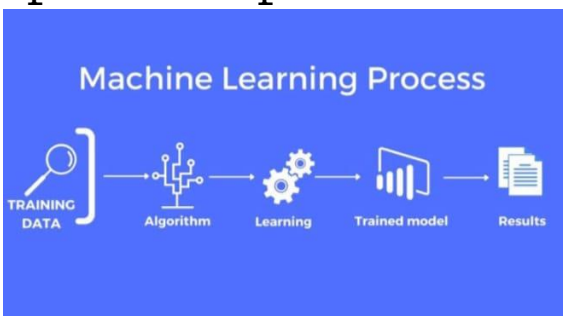
MODEL TRAINING:

In a diabetes prediction system using machine learning (ML), model training is a critical step. Here's a brief overview:

- **Data Collection:** The first step is to gather a dataset containing information about individuals, including features like age, body mass index (BMI), blood pressure, and historical diabetes status.
- **Data Preprocessing:** Raw data may be noisy or incomplete, so preprocessing is necessary. This includes handling missing values, scaling features, and encoding categorical variables.
- **Splitting Data:** The dataset is typically divided into two parts: a training set and a testing set. The

training set is used to train the ML model, while the testing set is used to evaluate its performance.

- **Model Selection:** Various ML algorithms, such as logistic regression, decision trees, or support vector machines, can be considered for diabetes prediction. The choice depends on the dataset and problem specifics.



- **Model Training:** The selected algorithm is trained on the training data, which involves learning the underlying patterns and relationships between the input features and the target variable (diabetes prediction).
- **Hyperparameter Tuning:** Fine-tuning the model's hyperparameters, like learning rates or tree depths, is essential to optimize its performance.
- **Evaluation:** The model's performance is assessed using the testing set. Common evaluation metrics include accuracy, precision, recall, and F1 score.
- **Deployment:** Once the model performs well, it can be deployed in a real-world healthcare setting to predict diabetes risk in new individuals based on their data.

- **Continuous Monitoring:** The model should be periodically updated and retrained with new data to maintain its accuracy and relevance.

Overall, model training is a pivotal stage in building a diabetes prediction system using ML, as the quality of the model directly impacts its ability to make accurate predictions and assist in healthcare decision-making.

Example Program:

```
# Create a machine learning model (Random Forest in this case)

Model = RandomForestClassifier()

# Train the model

Model.fit(X_train, y_train)

# Save the trained model to a file for later use

Joblib.dump(model, 'diabetes_prediction_model.pkl')

Print("Model trained and saved as 'diabetes_prediction_model.pkl'")
```

PYTHON PROGRAM:

Importing Dependencies

In [1]:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import
train_test_split,GridSearchCV
from sklearn.metrics import
accuracy_score,confusion_matrix
import warnings
warnings.filterwarnings('ignore')
sns.set()
%matplotlib inline
```

Data Collection and Analysis

In [2]:

```
data = pd.read_csv('/kaggle/input/pima-indians-diabetes-database/diabetes.csv')  #import dataset
```

Out[2]:

	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin
0	6	148	72	35	0
1	1	85	66	29	0
2	8	183	64	0	0

3	1	89	66	23	94
4	0	137	40	35	168

data.head() *#top 5 rows*

Training the Models

1. Logistic Regression

In [3]:

```
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression(C=1,penalty='l2')
reg.fit(X_train,Y_train)
log_acc=accuracy_score(Y_test,reg.predict(X_test))
print("Train Set Accuracy:"+str(accuracy_score(Y_train,reg.predict(X_train))*100))
print("Test Set Accuracy:"+str(accuracy_score(Y_test,reg.predict(X_test))*100)
```

Out[3]:

Train Set Accuracy:78.77094972067039

Test Set Accuracy:74.45887445887446

2. KNearestNeighbors

In [4]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=9)
#knn classifier
knn.fit(X_train,Y_train)
knn_acc = accuracy_score(Y_test,knn.predict(X_test))
print("Train Set
Accuracy:"+str(accuracy_score(Y_train,knn.predict(X
_train))*100))
print("Test Set
Accuracy:"+str(accuracy_score(Y_test,knn.predict(X_t
est)) *100))
Out[4]:
Train Set Accuracy:82.12290502793296
Test Set Accuracy:71.42857142857143
```

3. SVC

```
In [5]:
from sklearn.svm import SVC
svm = SVC()
svm.fit(X_train,Y_train)
svm_acc= accuracy_score(Y_test,svm.predict(X_test))
print("Train Set
Accuracy:"+str(accuracy_score(Y_train,svm.predict(X
_train))*100))
```

```
print("Test Set
Accuracy:"+str(accuracy_score(Y_test,svm.predict(X_
test))*100))
```

Out[5]:

```
Train Set Accuracy:85.1024208566108

Test Set Accuracy:74.02597402597402
```

4. DecisionTreeClassifier

In [6]:

```
from sklearn.tree import DecisionTreeClassifier

dtc =
DecisionTreeClassifier(criterion='entropy',max_depth
=5)

dtc.fit(X_train, Y_train)

dtc_acc= accuracy_score(Y_test,dtc.predict(X_test))

print("Train Set
Accuracy:"+str(accuracy_score(Y_train,dtc.predict(X_
train))*100))

print("Test Set
Accuracy:"+str(accuracy_score(Y_test,dtc.predict(X_t
est))*100))
```

Out[6]:

```
Train Set Accuracy:83.42644320297951

Test Set Accuracy:71.86147186147186
```

5. GradientBoostingClassifier

In [7]:

```
from sklearn.ensemble import
GradientBoostingClassifier

gbc = GradientBoostingClassifier()

gbc.fit(X_train, Y_train)

gbc_acc=accuracy_score(Y_test,gbc.predict(X_test))

print("Train Set
Accuracy:"+str(accuracy_score(Y_train,gbc.predict(X_
train))*100))

print("Test Set
Accuracy:"+str(accuracy_score(Y_test,gbc.predict(X_t
est))*100))
```

Out[7]:

Train Set Accuracy:94.22718808193669

Test Set Accuracy:73.59307359307358

6. XGBClassifier

In [8]:

```
from xgboost import XGBClassifier

xgb = XGBClassifier(booster = 'gbtree', learning_rate
= 0.1, max_depth=6,n_estimators = 10)

xgb.fit(X_train,Y_train)

xgb_acc= accuracy_score(Y_test,xgb.predict(X_test))

print("Train Set
Accuracy:"+str(accuracy_score(Y_train,xgb.predict(X_
train))*100))
```

```
print("Test Set
Accuracy:"+str(accuracy_score(Y_test,xgb.predict(X_t
est))*100))
```

Out[8]:

Train Set Accuracy:90.5027932960894

Test Set Accuracy:73.16017316017316

7.Stacking

In [9]:

```
from sklearn.model_selection import train_test_split
#splitting the dataset


train,val_train,test,val_test =
train_test_split(X,y,test_size=.50,random_state=3)
x_train,x_test,y_train,y_test =
train_test_split(train,test,test_size=.20,random_state
=3)
```

In [10]:

```
#first model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train, y_train)
```

Out[10]:

KNeighborsClassifier()

In [11]:

```
# second model
svm = SVC()
```

```
svm.fit(x_train, y_train)
Out[11]:
SVC()
In [12]:
pred_1=knn.predict(val_train)
pred_2=svm.predict(val_train)
# addition of 2 predictions
result = np.column_stack((pred_1,pred_2))
In [13]:
pred_test1=knn.predict(x_test)
pred_test2=svm.predict(x_test)
predict_test=np.column_stack((pred_test1,pred_test2
))
In [14]:
# stacking classifier
#RandomForestClasifier:- In this prediction of other 2
classification is taken as x value
from sklearn.ensemble import
RandomForestClassifier
rand_clf = RandomForestClassifier()
rand_clf.fit(result,val_test)
Out[14]:
RandomForestClassifier()
In [15]:
```

rand_clf.score(result,val_test)

Out[15]:

0.7291666666666666

In [16]:

```
rand_acc=accuracy_score(y_test,rand_clf.predict(pre
dict_test))
```

rand_acc

Out[16]:

0.7922077922077922

In [17]:

```
models = pd.DataFrame({'Model': ['Logistic','KNN',
'SVC',  'Decision Tree Classifier', 'Gradient Boosting
Classifier',  'XgBoost','Stacking'], 'Score': [
log_acc,knn_acc, svm_acc, dtc_acc, gbc_acc,
xgb_acc,rand_acc,]})
```

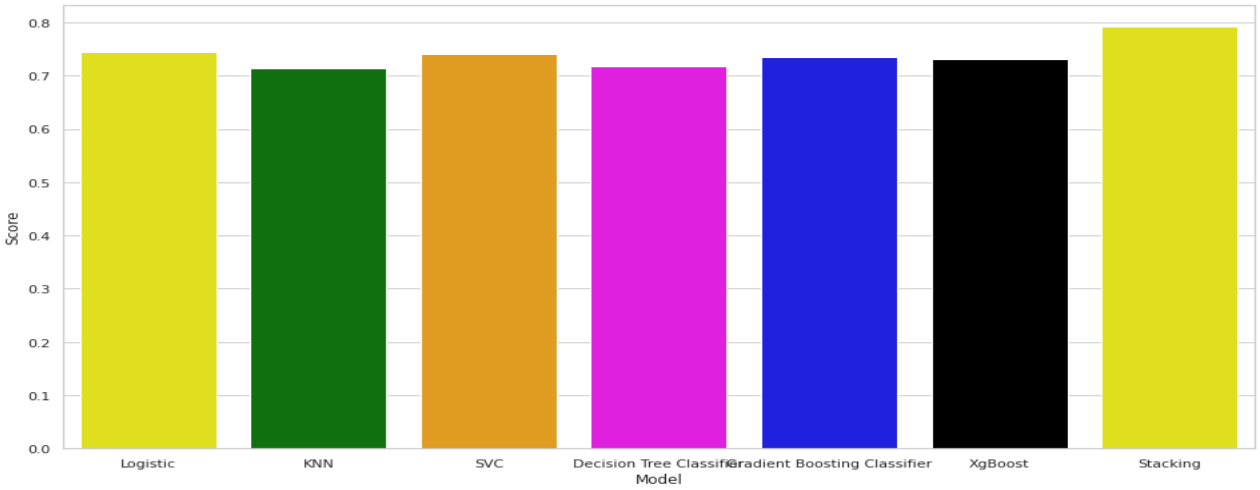
models.sort_values(by = 'Score', ascending = False)

Out[17]:

	Model	Score
6	Stacking	0.792208
0	Logistic	0.744589
2	SVC	0.740260
4	Gradient Boosting Classifier	0.735931
5	XgBoost	0.731602
3	Decision Tree Classifier	0.718615
1	KNN	0.714286

In [18]:

```
colors = ["yellow", "green", "orange",
"magenta","blue","black"]
sns.set_style("whitegrid")
```




```
plt.figure(figsize=(16,8))
plt.ylabel("Accuracy %")
plt.xlabel("Algorithms")
sns.barplot(x=models['Model'],y=models['Score'],
palette=colors )
plt.show()
```

Feature Engineering:

In [19]:

```
# According to BMI, some ranges were determined
and categorical variables were assigned.
NewBMI = pd.Series(["Underweight", "Normal",
"Overweight", "Obesity 1", "Obesity 2", "Obesity 3"],
dtype = "category")
df["NewBMI"] = NewBMI
```

```
df.loc[df["BMI"] < 18.5, "NewBMI"] = NewBMI[0]
df.loc[(df["BMI"] > 18.5) & (df["BMI"] <= 24.9),
"NewBMI"] = NewBMI[1]
df.loc[(df["BMI"] > 24.9) & (df["BMI"] <= 29.9),
"NewBMI"] = NewBMI[2]
df.loc[(df["BMI"] > 29.9) & (df["BMI"] <= 34.9),
"NewBMI"] = NewBMI[3]
df.loc[(df["BMI"] > 34.9) & (df["BMI"] <= 39.9),
"NewBMI"] = NewBMI[4]
df.loc[df["BMI"] > 39.9 ,"NewBMI"] = NewBMI[5]
```

In [20]:

```
df.head()
```

Out[20]:

	Pregnancies	Glucose	BloodPressure	SkinThicknes	Insuli n	BMI	Diabetes PedigreeFunctio n	Age	Outcome	NewBMI
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50	1	Obesity 1
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31	0	Overweight
2	8	183.0	64.0	32.0	16	23	0.672	32	1	Normal

					9.5	.3				
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0	Overweight
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1	Obesity 3

One Hot Encoding:

In [21]:

 # Here, by making One Hot Encoding transformation, categorical variables were converted into numerical values. It is also protected from the Dummy variable trap.

df = pd.get_dummies(df, columns =["NewBMI","NewInsulinScore", "NewGlucose"], drop_first = True)

In [22]:

df.head()

df.head()

Out[22]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50	1
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31	0
2	8	183.0	64.0	32.0	169.5	23.3	0.672	32	1
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21	0
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33	1

In [23]:

```
categorical_df = df[['NewBMI_Obesity 1','NewBMI_Obesity 2', 'NewBMI_Obesity 3', 'NewBMI_Overweight','NewBMI_Underweight',
```

'NewInsulinScore_Normal','NewGlucose_Low','NewGlucose_Normal', 'NewGlucose_Overweight',

In [24]:

```
y = df["Outcome"]
X = df.drop(["Outcome",'NewBMI_Obesity 1','NewBMI_Obesity 2', 'NewBMI_Obesity 3', 'NewBMI_Overweight','NewBMI_Underweight',
,'NewGlucose_Normal', 'NewGlucose_Overweight', 'NewGlucose_Secret'], axis = 1)
cols = X.columns
index = X.index
```

In [25]:

```
X.head()
```

Out[25]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.0	169.5	33.6	0.627	50
1	1	85.0	66.0	29.0	102.5	26.6	0.351	31

2	8	183.0	64.0	32.0	169.5	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33

In [26]:

```
from sklearn.preprocessing import RobustScaler
X = transformer.transform(X)
X = pd.DataFrame(X, columns = cols, index = index)
```

In [27]:

```
X.head()
```

Out[27]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.6	0.775	0.000	1.000000	1.000000	0.177778	0.669707	1.235294
1	-0.4	-0.800	-0.375	0.142857	0.000000	-0.600	-0.049511	0.117647

						000		
2	1.0	1.650	-0.500	0.571429	1.000000	-0.966667	0.786971	0.176471
3	-0.4	-0.700	-0.375	-0.714286	-0.126866	-0.433333	-0.528990	-0.470588
4	-0.6	0.500	-2.000	1.000000	0.977612	1.233333	4.998046	0.235294

In [28]:

```
X = pd.concat([X,categorical_df], axis = 1)
```

In[29]:

```
y.head()
```

Out[29]:

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

Base Models:

In [30]:

```
# Validation scores of all base models
models = []
models.append(('LR',
LogisticRegression(random_state = 12345)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART',
DecisionTreeClassifier(random_state = 12345)))
models.append(('RF',
RandomForestClassifier(random_state = 12345)))
models.append(('SVM', SVC(gamma='auto',
random_state = 12345)))
models.append(('XGB',
GradientBoostingClassifier(random_state = 12345)))
models.append(("LightGBM",
LGBMClassifier(random_state = 12345)))

# evaluate each model in turn
results = []
names = []
```

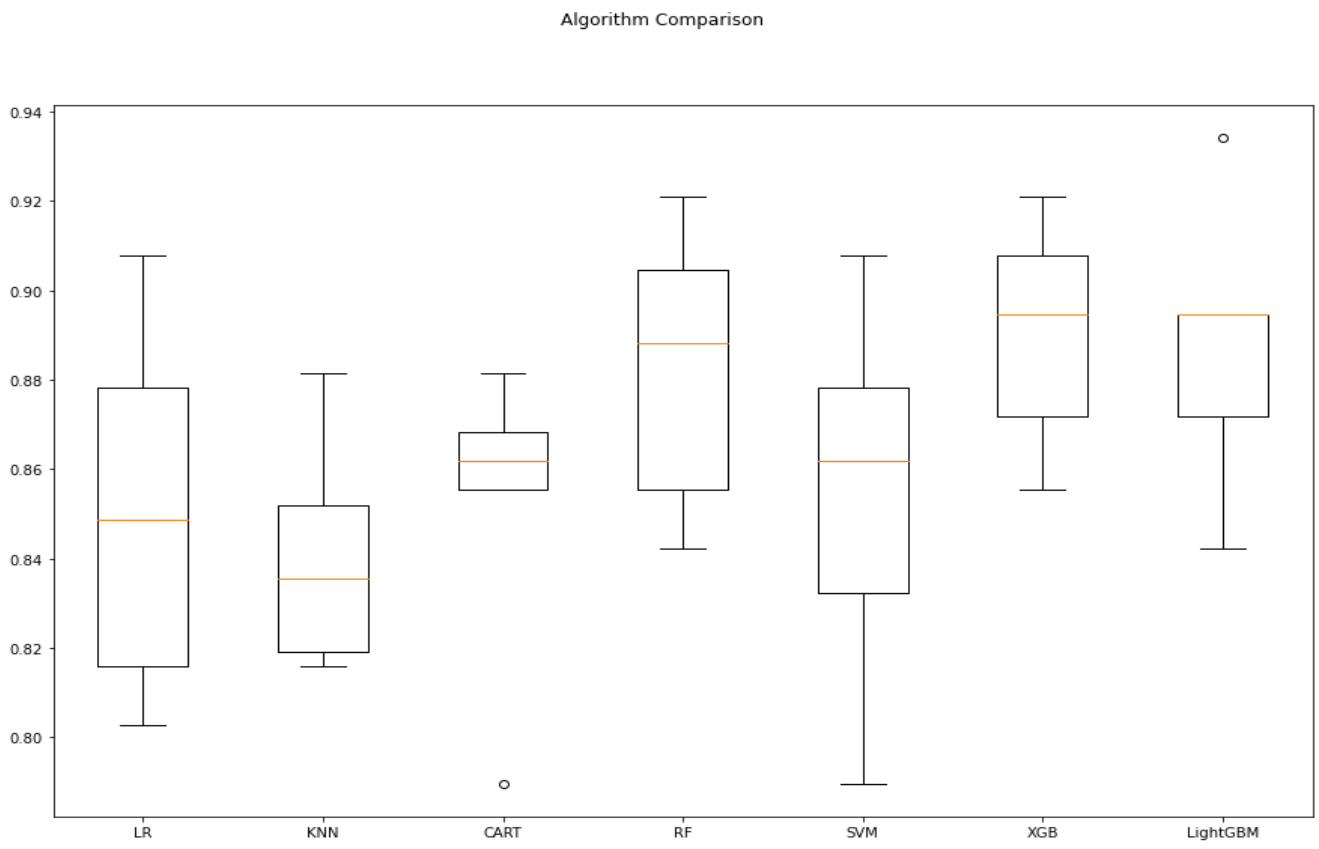
In [31]:

```
for name, model in models:
kfold = KFold(n_splits = 10, random_state = 12345)
```

```
cv_results = cross_val_score(model, X, y, cv = 10,
scoring= "accuracy")
results.append(cv_results)
names.append(name)
msg = "%s: %f (%f)" % (name, cv_results.mean(),
cv_results.std())
print(msg)
# boxplot algorithm comparison
fig = plt.figure(figsize=(15,10))
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
LR: 0.848684 (0.036866)
KNN: 0.840789 (0.023866)
CART: 0.857895 (0.024826)
RF: 0.881579 (0.026316)
SVM: 0.853947 (0.036488)
XGB: 0.890789 (0.020427)
LightGBM: 0.885526 (0.024298)

# boxplot algorithm comparison
```

```
fig = plt.figure(figsize=(15,10))
fig.suptitle('Algorithm Comparison')
```



Various Features to perform model evaluation:

Various features can be used to perform model training in a diabetes prediction system using machine learning. Some of the most common features include:

Demographic data: Age, gender, race/ethnicity, family history of diabetes, and socioeconomic status.

Physical examination findings: Height, weight, body mass index (BMI), waist circumference, blood pressure, and heart rate.

Laboratory results: Blood sugar levels, blood glucose tolerance test results, insulin levels, and other metabolic markers.

Medical history: Presence of other medical conditions, such as high blood pressure, high cholesterol, heart disease, and stroke.

In addition to these common features, other features that may be useful for diabetes prediction include:

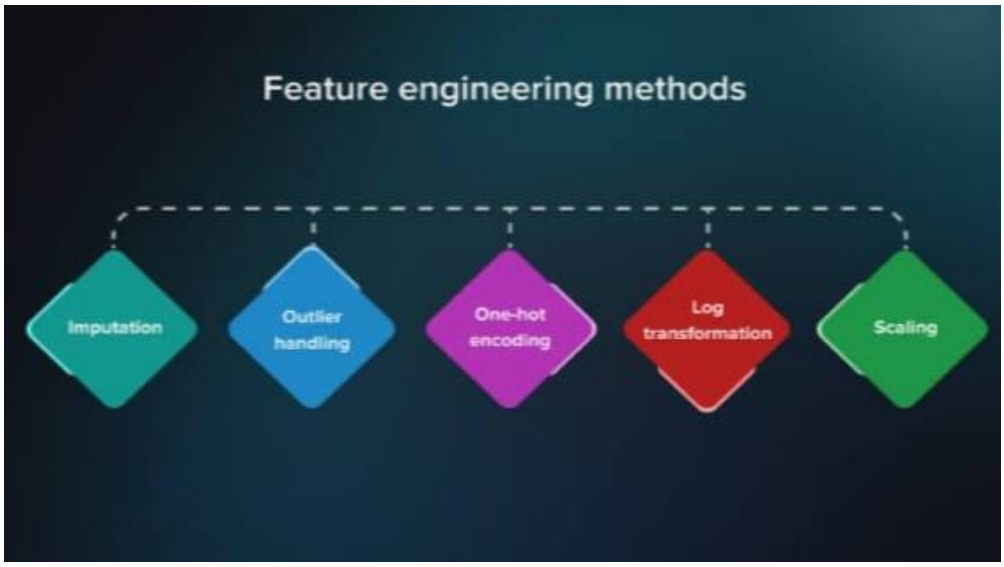
- Genetic data
- Biomarkers
- Wearable device data

It is important to note that not all features will be equally important for diabetes prediction. The best features to use will depend on the specific dataset and the machine learning algorithm being used.

Here are some examples of how features can be used to predict diabetes:

A machine learning model could use the patient's age, BMI, and blood sugar levels to predict their risk of developing diabetes in the next 5 years.

Machine learning models can be used to combine



data from multiple features to generate more accurate predictions. For example, a model could use the patient's age, BMI, blood sugar levels, genetic data, and wearable device data to predict their risk of developing diabetes.

By using machine learning to predict diabetes, healthcare providers can identify patients who are at high risk of developing the condition and intervene early to prevent or delay its onset.

CONCLUSION:

- A diabetes prediction system using feature engineering, model training, and evaluation can be a valuable tool for identifying people at high risk of developing the disease. By using a variety of features, including demographic data, physical examination findings, laboratory results, lifestyle factors, and medical history, machine learning models can generate accurate predictions of diabetes risk.
- Feature engineering is the process of transforming raw data into features that are more informative and useful for machine learning. This can be done by creating new features, combining existing features, or transforming features in other ways. For example, a feature engineer might create a new feature representing the patient's waist-to-hip ratio, or combine the patient's blood sugar levels and insulin levels to create a new feature representing their insulin sensitivity.
- Model training is the process of teaching a machine learning model to predict diabetes risk based on the features. This is done by feeding the model a dataset of patients with known diabetes status and allowing it to learn the relationships between the features and the outcome. Once the

model is trained, it can be used to predict the diabetes risk of new patients.

- Model evaluation is the process of assessing the performance of a machine learning model on a held-out test set. This is important because it helps to ensure that the model is generalizable to new data. There are a number of different metrics that can be used to evaluate model performance, such as accuracy, precision, recall, and F1 score.
- A diabetes prediction system using feature engineering, model training, and evaluation can be used to identify people at high risk of developing the disease and intervene early to prevent or delay its onset. This can lead to improved health outcomes and reduced healthcare costs.