Diabetes DT and KNN - 21F21484 VIVEIK CATARAM SAICHANDRA

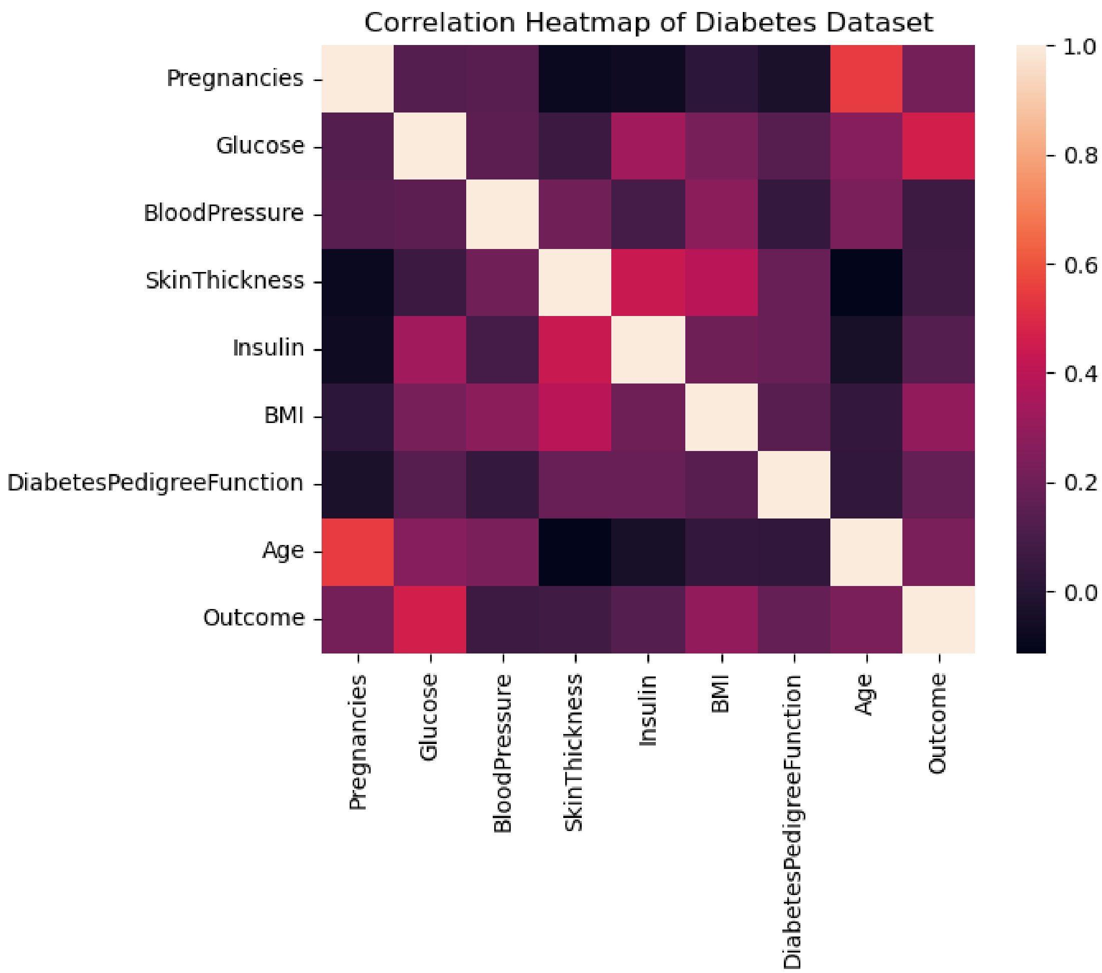
December 27, 2024

[10]:
```
#Importing Necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[11]:
```
#Importing and Reading the Dataset
dataset = pd.read_csv('diabetes.csv')
dataset.head()
```

[11]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[12]:
```
#Plotting Heatmap of the Dataset
plt.figure(1)
sns.heatmap(dataset.corr())
plt.title('Correlation Heatmap of Diabetes Dataset')
plt.show()
```



[13]:
```python
# Replace 0 values with the median of the feature's distribution to handle Zero␣
↪(0) values
columns_with_zeros = ['Pregnancies', 'Glucose', 'BloodPressure',␣
↪'SkinThickness', 'Insulin', 'BMI'] #List contatining parameters with 0 values
for column in columns_with_zeros:
    median = dataset[column].median()
    dataset[column] = dataset[column].replace(0, median)

dataset.head()
```

[13]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35       27  33.6
1            1       85             66             29       27  26.6
2            8      183             64             23       27  23.3
3            1       89             66             23       94  28.1
4            3      137             40             35      168  43.1
```

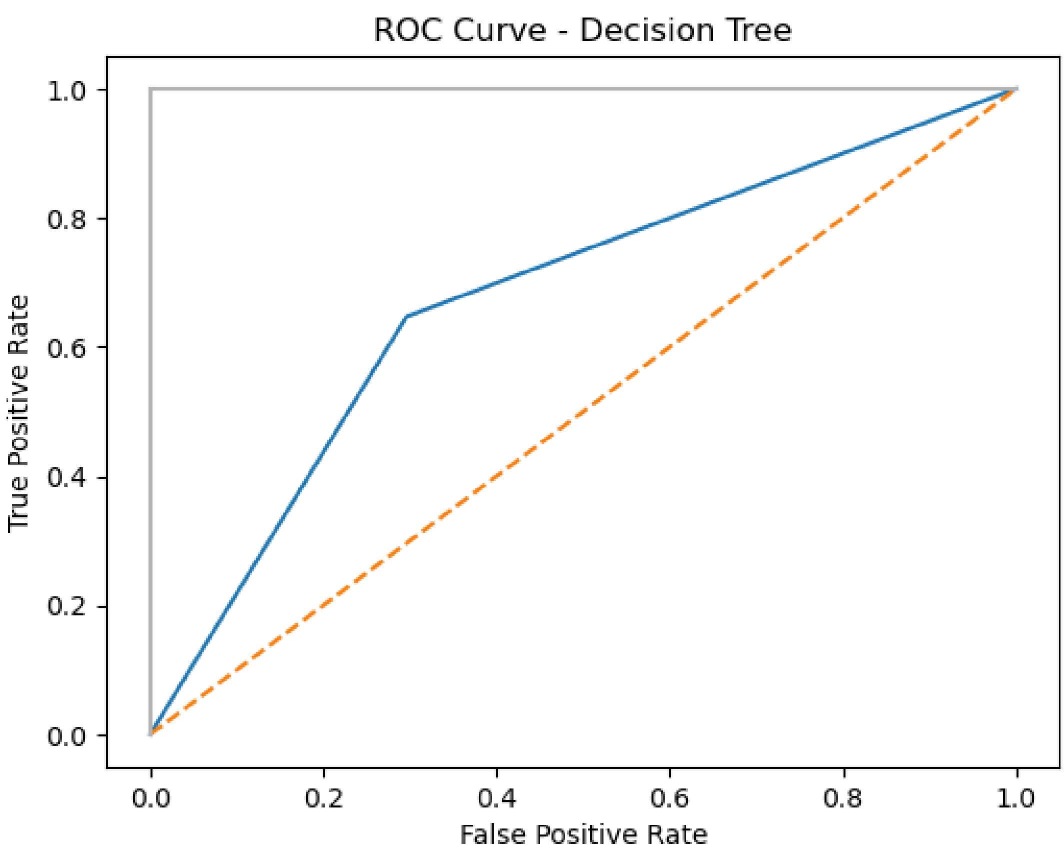
```
   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[14]:
```python
#Splitting dataset into train-test data
from sklearn.model_selection import train_test_split

x = dataset.drop(columns = ['Outcome']) #Features
y = dataset['Outcome'] #Target
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25)␣
 ↪#Train-Test Split
```

[15]:
```python
#DECISION TREE (DT)
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix, roc_curve

dt_classifier = DecisionTreeClassifier() #Decision Tree Classifier model
dt_classifier.fit(x_train, y_train) #Training the Classifier with the training␣
 ↪data
dt_pred = dt_classifier.predict(x_test) #Using the trained Classifier model to␣
 ↪predict the testing data

dt_report = classification_report(y_test, dt_pred) #Classification Report
dt_matrix = confusion_matrix(y_test, dt_pred) #Confusion Matrix
dt_accuracy = accuracy_score(y_test, dt_pred) #Accuracy Score
dt_fp_rate, dt_tp_rate, dt_threshold = roc_curve(y_test, dt_pred) #ROC Curve
dt_sensitivity = dt_matrix[0,0] / (dt_matrix[0,0] + dt_matrix[0,1])␣
 ↪#Sensitivity = TP / (TP + FN)
dt_specificity = dt_matrix[1,1] / (dt_matrix[1,1] + dt_matrix[1,0])␣
 ↪#Specificity = TN / (TN + FP)
dt_precision = dt_matrix[0,0] / (dt_matrix[0,0] + dt_matrix[1,0]) #Precision =␣
 ↪TP / (TP + FP)


print(dt_report)
print(dt_matrix)
print('Decision Tree Accuracy = ',dt_accuracy)
print('Decision Tree Sensitivity = ',dt_sensitivity)
print('Decision Tree Specificity = ',dt_specificity)
print('Decision Tree Precision = ',dt_precision)
plt.title('ROC Curve - Decision Tree')
plt.plot(dt_fp_rate, dt_tp_rate)
```

```
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.79      0.70      0.74       125
           1       0.54      0.65      0.59        68

    accuracy                           0.68       193
   macro avg       0.66      0.68      0.67       193
weighted avg       0.70      0.68      0.69       193

[[88 37]
 [24 44]]
Decision Tree Accuracy =  0.6839378238341969
Decision Tree Sensitivity =  0.704
Decision Tree Specificity =  0.6470588235294118
Decision Tree Precision =  0.7857142857142857
```



[16]:
```python
#K-NEAREST NEIGHBOUR (KNN)
from sklearn.preprocessing import MinMaxScaler

#Normalization Preproccesing Technique
scaler = MinMaxScaler()

#Columns that require normalization
scale_columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',␣
↪'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']
normalized_data = scaler.fit_transform(dataset[scale_columns])
normalized_dataset = pd.DataFrame(normalized_data, columns = scale_columns)␣
↪#Converting normalized data into dataset
normalized_dataset['Outcome'] = dataset['Outcome'] #Adding the unaltered Target␣
↪variable to the Normalized Dataset
normalized_dataset.head()
```

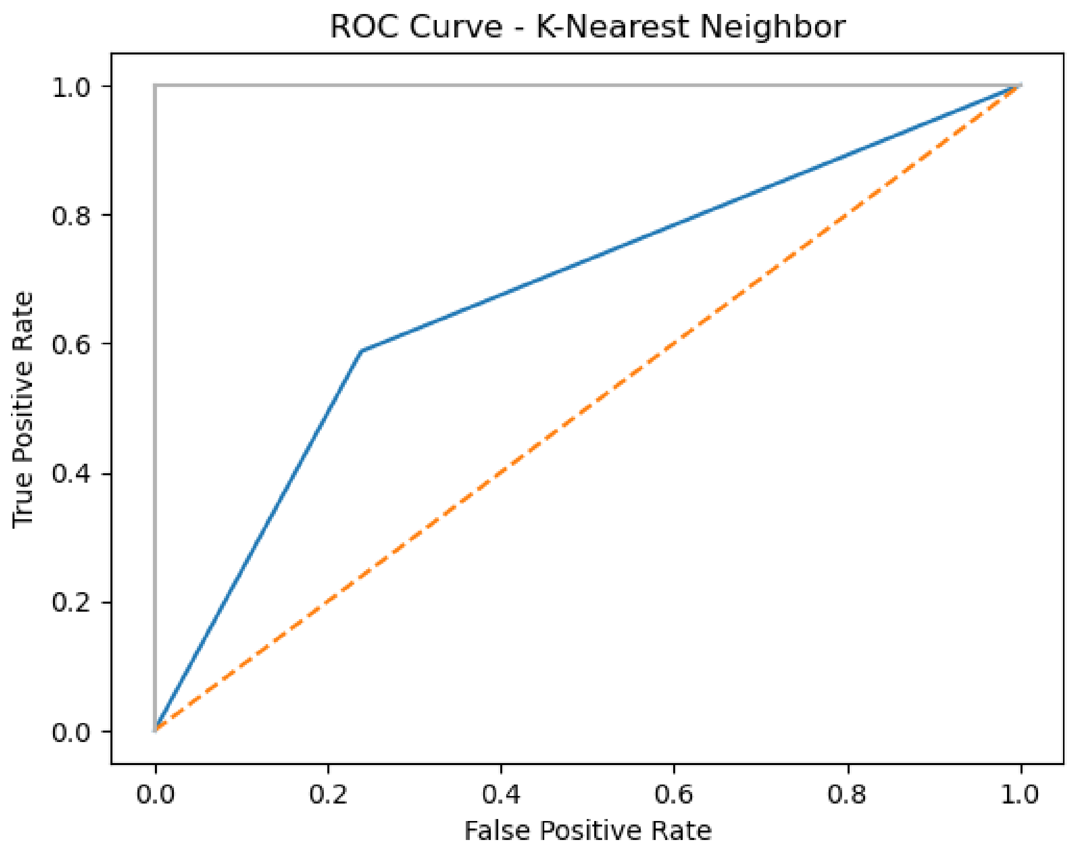
[16]:
```
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0       0.3125  0.670968       0.489796       0.304348  0.015625  0.314928
1       0.0000  0.264516       0.428571       0.239130  0.015625  0.171779
2       0.4375  0.896774       0.408163       0.173913  0.015625  0.104294
3       0.0000  0.290323       0.428571       0.173913  0.096154  0.202454
4       0.1250  0.600000       0.163265       0.304348  0.185096  0.509202

   DiabetesPedigreeFunction       Age  Outcome
0                  0.234415  0.483333        1
1                  0.116567  0.166667        0
2                  0.253629  0.183333        1
3                  0.038002  0.000000        0
4                  0.943638  0.200000        1
```

[17]:
```python
#Splitting dataset into train-test data
x_normal = normalized_dataset.drop(columns = 'Outcome') #Features
y_normal = normalized_dataset['Outcome'] #Target
x_train_normal, x_test_normal, y_train_normal, y_test_normal =␣
↪train_test_split(x_normal, y_normal, test_size = 0.25) #Train-Test Split
```

[18]:
```python
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors = 5) #K-Nearest Neighbor␣
↪Classifier model
knn_classifier.fit(x_train_normal, y_train_normal) #Training the Classifier␣
↪with the training data
knn_pred = knn_classifier.predict(x_test_normal) #Using the trained Classifier␣
↪model to predict the testing data
```

```python
knn_report = classification_report(y_test_normal, knn_pred) #Classification Report
knn_matrix = confusion_matrix(y_test_normal, knn_pred) #Confusion Matrix
knn_accuracy = accuracy_score(y_test_normal, knn_pred) #Accuracy Score
knn_fp_rate, knn_tp_rate, knn_threshold = roc_curve(y_test_normal, knn_pred) #ROC Curve
knn_sensitivity = knn_matrix[0,0] / (knn_matrix[0,0] + knn_matrix[0,1]) #Sensitivity = TP / (TP + FN)
knn_specificity = knn_matrix[1,1] / (knn_matrix[1,1] + knn_matrix[1,0]) #Specificity = TN / (TN + FP)
knn_precision = knn_matrix[0,0] / (knn_matrix[0,0] + knn_matrix[1,0]) #Precision = TP / (TP + FP)


print(knn_report)
print(knn_matrix)
print('K-Nearest Neighbor Accuracy = ',knn_accuracy)
print('K-Nearest Neighbor Sensitivity = ',knn_sensitivity)
print('K-Nearest Neighbor Specificity = ',knn_specificity)
print('K-Nearest Neighbor Precision = ',knn_precision)
plt.title('ROC Curve - K-Nearest Neighbor')
plt.plot(knn_fp_rate, knn_tp_rate)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.79      0.76      0.78       130
           1       0.54      0.59      0.56        63

    accuracy                           0.70       193
   macro avg       0.67      0.67      0.67       193
weighted avg       0.71      0.70      0.71       193

[[99 31]
 [26 37]]
K-Nearest Neighbor Accuracy =  0.7046632124352331
K-Nearest Neighbor Sensitivity =  0.7615384615384615
K-Nearest Neighbor Specificity =  0.5873015873015873
K-Nearest Neighbor Precision =  0.792
```


ROC Curve - K-Nearest Neighbor
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0