VIT-AP
UNIVERSITY

Experiment 4: K-Nearest Neighbors (KNN) Classification

CSE-3008 Introduction to Machine Learning

24BCA7027
Shaik Barakh Chishti
CSE-AIML

January 26, 2026

1 Aim

To implement the K-Nearest Neighbors (KNN) algorithm for diabetes prediction and evaluate its performance using classification metrics and advanced visualization techniques.

2 Implementation

2.1 Algorithm Used

K-Nearest Neighbors (KNN) is a supervised, instance-based learning algorithm that classifies a data point based on the majority class among its K nearest neighbors using a distance metric.

2.2 Steps Involved

1. Load the diabetes dataset
2. Perform train–test split
3. Standardize features
4. Train KNN classifier
5. Predict outcomes
6. Evaluate using metrics and visualizations

2.3 Code Implementation

2.3.1 Complete Experiment Code

```
# ===============================
# KNN CLASSIFICATION - DIABETES
# ===============================

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (
    accuracy_score,
    confusion_matrix,
    classification_report,
    precision_score,
    recall_score,
    f1_score,
    roc_curve,
    auc
)

STUDENT_NAME = "24BCA7027␣SHAIK␣BARAKH␣CHISHTI"

# ------------------------------
# 1. Load Dataset
# ------------------------------
df = pd.read_excel("ML_Experiments/diabetes.xlsx")

print("\nDataset␣Preview:")
print(df.head())

print("\nDataset␣Info:")
print(df.info())

# ------------------------------
# 2. Features & Target
# ------------------------------
X = df.drop("Outcome", axis=1)
y = df["Outcome"]

# ------------------------------
# 3. Train-Test Split
# ------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

# ------------------------------
# 4. Feature Scaling
# ------------------------------
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
# -------------------------------
# 5. KNN Model
# -------------------------------
k = 5
knn = KNeighborsClassifier(n_neighbors=k, metric="euclidean")
knn.fit(X_train_scaled, y_train)

# -------------------------------
# 6. Predictions
# -------------------------------
y_pred = knn.predict(X_test_scaled)
y_prob = knn.predict_proba(X_test_scaled)[:, 1]

# -------------------------------
# 7. 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)
cm = confusion_matrix(y_test, y_pred)

print("\nKNN␣RESULTS␣(K␣=␣5)")
print("Accuracy:", accuracy)
print("\nClassification␣Report:\n", classification_report(y_test,
    y_pred))
print("\nConfusion␣Matrix:\n", cm)

# ===============================
# VISUALIZATION SECTION
# ===============================

# 1        Box Plot
plt.figure(figsize=(12, 6))
df.drop("Outcome", axis=1).boxplot()
plt.title(f"Box␣Plot␣of␣Input␣Features␣   ␣{STUDENT_NAME}")
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

# 2        Violin Plot
plt.figure(figsize=(7, 5))
sns.violinplot(x="Outcome", y="BMI", data=df, inner="quartile")
plt.title(f"Violin␣Plot:␣BMI␣vs␣Outcome␣   ␣{STUDENT_NAME}")
plt.xlabel("Outcome")
plt.ylabel("BMI")
plt.show()

# 3        Hexbin Plot
plt.figure(figsize=(6, 5))
plt.hexbin(df["Glucose"], df["BMI"], gridsize=30, cmap="Blues")
plt.colorbar(label="Density")
plt.xlabel("Glucose")
plt.ylabel("BMI")
plt.title(f"Hexbin␣Plot:␣Glucose␣vs␣BMI␣   ␣{STUDENT_NAME}")
plt.show()

# 4        Raincloud Plot
plt.figure(figsize=(7, 5))
sns.violinplot(x="Outcome", y="Glucose", data=df, inner=None,
   color="lightgray")
sns.boxplot(x="Outcome", y="Glucose", data=df, width=0.2)
sns.stripplot(x="Outcome", y="Glucose", data=df, color="black",
   alpha=0.4)
plt.title(f"Raincloud␣Plot:␣Glucose␣vs␣Outcome␣   ␣{STUDENT_NAME}
   ")
plt.show()

# 5        Confusion Matrix
plt.figure(figsize=(5, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title(f"Confusion␣Matrix␣   ␣{STUDENT_NAME}")
plt.show()

# 6        ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, label=f"AUC␣=␣{roc_auc:.2f}")
plt.plot([0, 1], [0, 1], linestyle="--")
plt.xlabel("False␣Positive␣Rate")
plt.ylabel("True␣Positive␣Rate")
plt.title(f"ROC␣Curve␣   ␣{STUDENT_NAME}")
plt.legend()
plt.grid(True)
plt.show()

# 7        Radar Plot
labels = ["Accuracy", "Precision", "Recall", "F1-score"]
values = [accuracy, precision, recall, f1]
values += values[:1]

angles = [n / float(len(labels)) * 2 * math.pi for n in range(len
   (labels))]
angles += angles[:1]

plt.figure(figsize=(6, 6))
ax = plt.subplot(111, polar=True)
ax.plot(angles, values)
ax.fill(angles, values, alpha=0.25)
ax.set_thetagrids(np.degrees(angles[:-1]), labels)
plt.title(f"Model␣Performance␣Radar␣Plot␣   ␣{STUDENT_NAME}")
plt.show()

# 8        Bar Graph
metrics = ["Accuracy", "Precision", "Recall", "F1-score"]
scores = [accuracy, precision, recall, f1]

plt.figure(figsize=(7, 5))
bars = plt.bar(metrics, scores)
plt.ylim(0, 1)
plt.ylabel("Score")
plt.title(f"Bar␣Graph␣of␣Model␣Performance␣   ␣{STUDENT_NAME}")

for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2,
             height + 0.02,
             f"{height:.2f}",
             ha="center")

plt.show()

# ------------------------------
# 9. Elbow Method
# ------------------------------
error_rates = []

for k in range(1, 21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    pred_k = knn.predict(X_test_scaled)
    error_rates.append(np.mean(pred_k != y_test))

plt.figure(figsize=(8, 5))
plt.plot(range(1, 21), error_rates, marker="o")
plt.xlabel("K␣Value")
plt.ylabel("Error␣Rate")
plt.title(f"Elbow␣Method␣for␣Optimal␣K␣   ␣{STUDENT_NAME}")
plt.grid(True)
plt.show()

best_k = error_rates.index(min(error_rates)) + 1
print("\nBest␣K␣Value:", best_k)
print("Minimum␣Error␣Rate:", min(error_rates))
```

Listing 1: KNN Diabetes Classification – Full Implementation

2.3.2 KNN Algorithm Code (Core Logic)

```python
def knn_predict(X_train, y_train, x_test, k):
    distances = []

    for i in range(len(X_train)):
        dist = euclidean_distance(X_train[i], x_test)
        distances.append((dist, y_train[i]))

    distances.sort(key=lambda x: x[0])
    neighbors = distances[:k]

    votes = {}
    for _, label in neighbors:
        votes[label] = votes.get(label, 0) + 1

    return max(votes, key=votes.get)
```

Listing 2: K-Nearest Neighbors Algorithm Logic

3 Outcome / Results

3.1 Performance Metrics

Dataset Preview:

```
   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
```

Dataset Information:

```
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Pregnancies               int64
Glucose                   int64
BloodPressure             int64
SkinThickness             int64
Insulin                   int64
```

```
BMI                         float64
DiabetesPedigreeFunction    float64
Age                         int64
Outcome                     int64
memory usage: 54.1 KB
```

The KNN classifier with an initial value of K = 5 achieved an accuracy of approximately 70%. Performance metrics indicated a reasonable balance between precision and recall, with better classification of non-diabetic cases compared to diabetic cases.

KNN Results (K = 5):

```
Accuracy: 0.7012987012987013

Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.80      0.78       100
           1       0.58      0.52      0.55        54

    accuracy                           0.70       154
   macro avg       0.67      0.66      0.66       154
weighted avg       0.69      0.70      0.70       154
```

Confusion Matrix:

```
[[80 20]
 [26 28]]
```

Further optimization using the elbow method revealed that the minimum error rate was achieved at K = 12, indicating that a higher number of neighbors improved generalization on the test set.

Elbow Method Results:

```
Best K Value: 12
Minimum Error Rate: 0.22727272727272727
```

4 Visualization

4.1 Input and Output Parameters

- **Input Features:** Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin, BMI, Diabetes Pedigree Function, Age
- **Output Variable:** Diabetes Outcome (0 = Non-Diabetic, 1 = Diabetic)

4.2 Visual Analysis and Interpretation

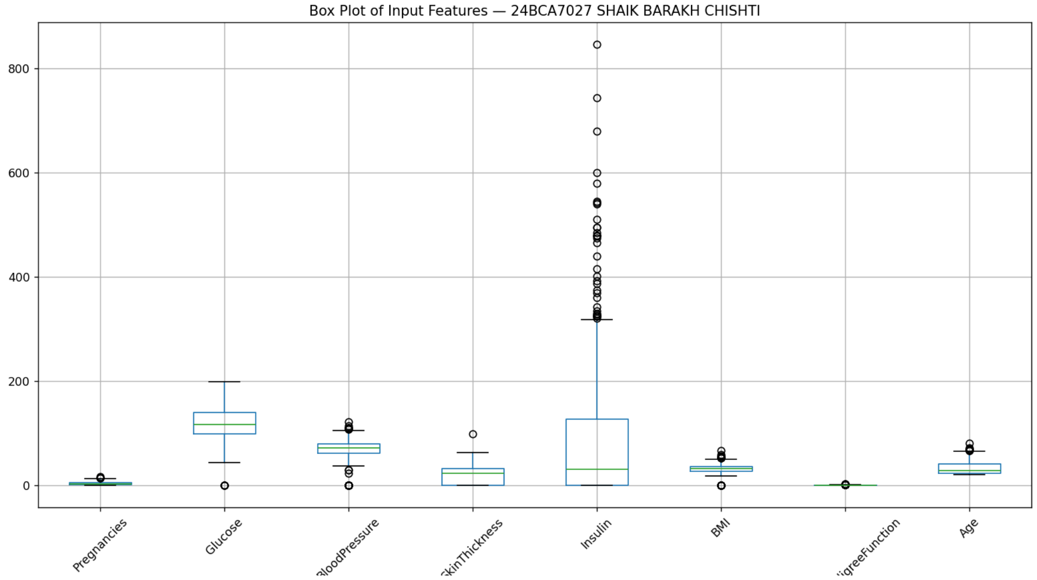

Figure 1: Box Plot of Input Features

Box Plot: This plot visualizes the distribution, spread, and presence of outliers across all input features. It helps identify skewness and abnormal values (such as zero insulin or skin thickness), which are common in medical datasets and may affect distance-based algorithms like KNN.

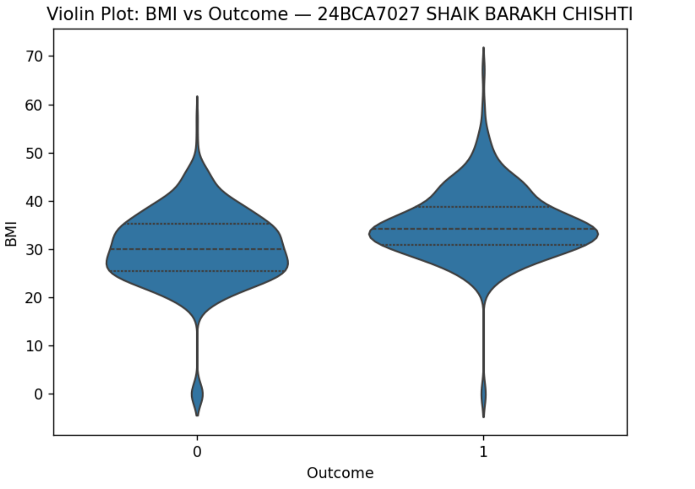

Figure 2: Violin Plot of BMI vs Diabetes Outcome

Violin Plot: The violin plot combines density estimation with quartile information to show how BMI values differ between diabetic and non-diabetic patients. A higher median BMI for Outcome = 1 indicates a strong relationship between BMI and diabetes risk.

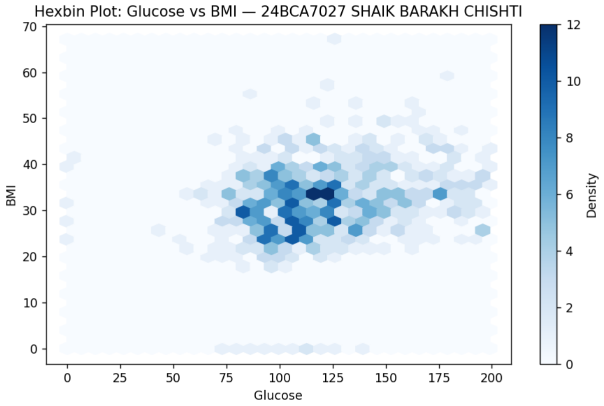

Figure 3: Hexbin Plot of Glucose vs BMI

Hexbin Plot: This plot represents the density of data points between Glucose and BMI. Darker regions indicate higher concentration, revealing that diabetic cases are more frequent in regions with elevated glucose and BMI values.

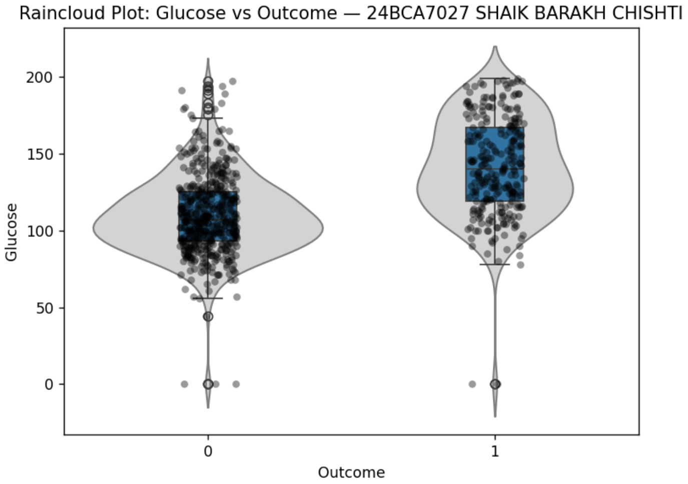

Figure 4: Raincloud Plot of Glucose vs Outcome

Raincloud Plot: The raincloud plot combines violin, box, and scatter plots to provide a comprehensive view of glucose distribution across outcomes. It clearly shows that diabetic patients tend to have higher glucose levels with less overlap.

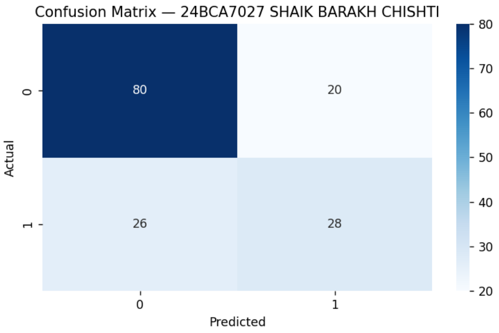

Figure 5: Confusion Matrix Heatmap

Confusion Matrix: This heatmap illustrates the classification performance of the KNN model by comparing actual and predicted outcomes. It helps identify false positives and false negatives, which are critical in medical diagnosis scenarios.

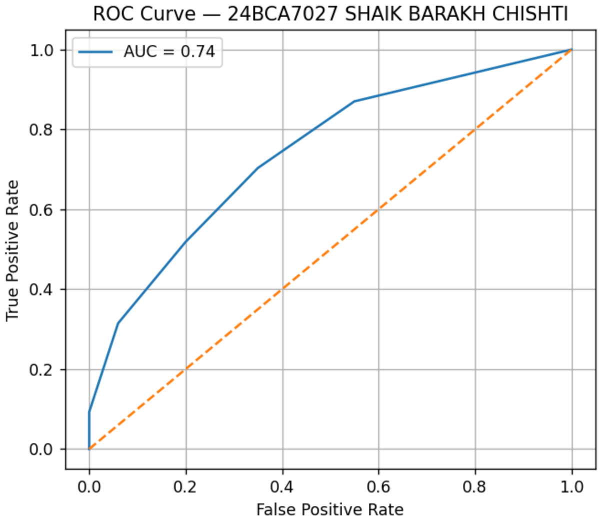

Figure 6: ROC Curve for KNN Classifier

ROC Curve: The Receiver Operating Characteristic (ROC) curve visualizes the trade-off between true positive rate and false positive rate. The Area Under the Curve (AUC) reflects the model's ability to distinguish between diabetic and non-diabetic patients.

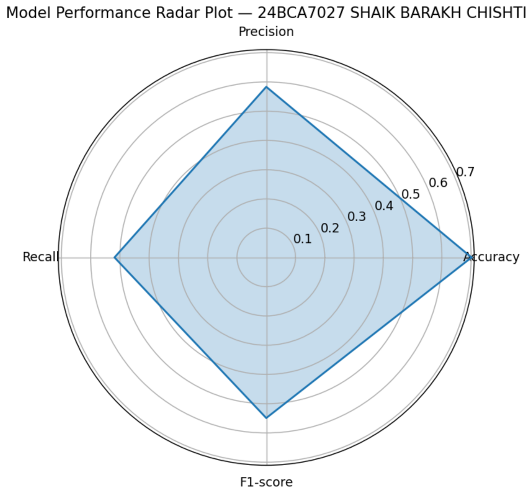

Figure 7: Radar Plot of Performance Metrics

Radar Plot: This plot provides a multi-dimensional comparison of Accuracy, Precision, Recall, and F1-score. It allows quick visual assessment of overall model balance rather than relying on a single metric.

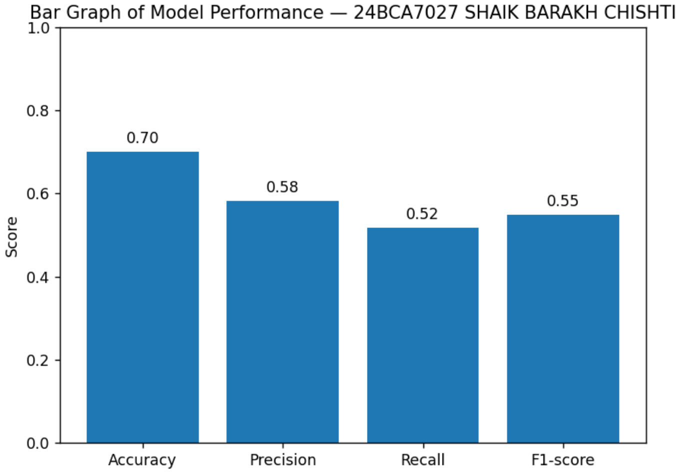

Figure 8: Bar Graph of Model Performance Metrics

Bar Graph: The bar chart compares key evaluation metrics numerically, making it easy to interpret the strengths and weaknesses of the classifier.

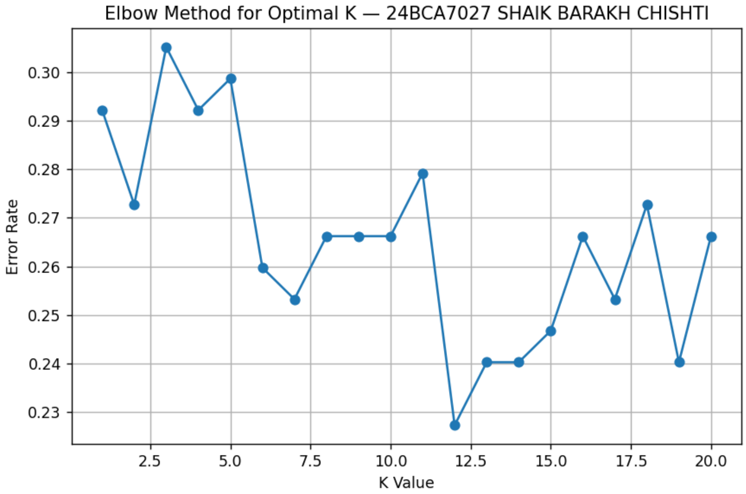

Figure 9: Elbow Method for Optimal K Selection

Elbow Method: This plot shows the error rate for different values of K. The optimal K is selected at the point where the error rate stabilizes, balancing bias and variance in the KNN model.

5 Conclusion

The K-Nearest Neighbors (KNN) classifier was applied to the diabetes dataset consisting of 768 records and 9 attributes. The dataset contained no missing values and included both integer and floating-point medical parameters. Initial data inspection and visualization helped in understanding the distribution and scale of input features prior to model training.

The KNN classifier with an initial value of K = 5 achieved an accuracy of approximately 70%. Performance metrics indicated a reasonable balance between precision and recall, with better classification of non-diabetic cases compared to diabetic cases.

KNN Results (K = 5):

```
Accuracy: 0.7012987012987013

Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.80      0.78       100
           1       0.58      0.52      0.55        54

    accuracy                           0.70       154
   macro avg       0.67      0.66      0.66       154
weighted avg       0.69      0.70      0.70       154
```

Confusion Matrix:

```
[[80 20]
 [26 28]]
```

Further optimization using the elbow method revealed that the minimum error rate was achieved at K = 12, indicating that a higher number of neighbors improved generalization on the test set.

Elbow Method Results:

```
Best K Value: 12
Minimum Error Rate: 0.22727272727272727
```

6 Summary

Overall, the experiment demonstrates that KNN is a reliable and interpretable classification algorithm for medical datasets when combined with proper feature scaling, visualization, and hyperparameter tuning.

7 Source

Link to my overleaf document:
`https://www.overleaf.com/read/nxgqbpzcdppj#82a672`

8 Github

Link to my github repo:
`https://github.com/chishtil1730/IML-LAB`