Task 6: K-Nearest Neighbors (KNN) Classification

Objective

Understand and implement KNN for classification problems.

Steps & Implementation

1. Dataset

- Dataset Used: **Iris Dataset** (scikit-learn / Kaggle)
- Target: Species (Setosa, Versicolor, Virginica)

2. Data Preprocessing

- Features normalized using `StandardScaler`
- Train-test split (80:20)

3. KNN Classifier

- Implemented using `KNeighborsClassifier`
- Tried different values of K (3, 5, 7, 9)

4. Model Evaluation

- Metrics: Accuracy, Confusion Matrix
- Example results (K=5):
- Accuracy: \~97%
- Confusion Matrix:

```
[[10  0  0]
 [ 0  9  1]
 [ 0  0 10]]
```

5. Decision Boundaries

- Visualized using matplotlib (2D feature pairs)
- Showed how classification regions change with different K values

---

Interview Questions & Answers

1. **How does the KNN algorithm work?**

2. Classifies a data point based on majority vote of its K nearest neighbors using a distance metric.

3. **How do you choose the right K?**

4. Small K → high variance (overfitting), Large K → high bias (underfitting). Usually odd K to avoid ties.
5. **Why is normalization important in KNN?**
6. Distance metrics are scale-sensitive; normalization ensures fair contribution of all features.
7. **What is the time complexity of KNN?**
8. Training: O(1), Prediction: O(n × d) per query (n = samples, d = dimensions).
9. **What are pros and cons of KNN?**
10. ✅Pros: Simple, no training, works well on small datasets.
11. ❌Cons: Slow on large datasets, sensitive to noise & irrelevant features.
12. **Is KNN sensitive to noise?**
13. Yes, mislabeled or noisy data can mislead neighbor voting.
14. **How does KNN handle multi-class problems?**
15. Majority voting among neighbors naturally extends to multiple classes.
16. **What's the role of distance metrics in KNN?**
17. Defines neighbor closeness (e.g., Euclidean, Manhattan, Minkowski).

Code (Colab / Jupyter Notebook)

```
# KNN Classification - Iris Dataset
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
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
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Load dataset
data = load_iris()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)
```

```python
# 2. 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)

# 3. Normalize features
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)

# 4. Train KNN with different K values
for k in [3, 5, 7, 9]:
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(X_train_s, y_train)
    y_pred = clf.predict(X_test_s)
    acc = accuracy_score(y_test, y_pred)
    print(f'K={k} -> Accuracy: {acc:.3f}')
    print(confusion_matrix(y_test, y_pred))

# 5. Final model (K=5)
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_s, y_train)
y_pred = knn.predict(X_test_s)

print("\nClassification Report:\n", classification_report(y_test, y_pred))

# 6. Confusion Matrix Heatmap
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('KNN Confusion Matrix (K=5)')
plt.show()

# 7. Decision Boundaries (2 features for visualization)
from matplotlib.colors import ListedColormap

X_vis = X.iloc[:, :2].values
y_vis = y.values
X_train_v, X_test_v, y_train_v, y_test_v = train_test_split(X_vis, y_vis,
test_size=0.2, random_state=42, stratify=y_vis)
scaler_v = StandardScaler()
X_train_v = scaler_v.fit_transform(X_train_v)
X_test_v = scaler_v.transform(X_test_v)

clf_vis = KNeighborsClassifier(n_neighbors=5)
clf_vis.fit(X_train_v, y_train_v)

# Mesh grid
gx, gy = np.meshgrid(np.arange(X_train_v[:,0].min()-1, X_train_v[:,0].max()
+1, 0.01),
                       np.arange(X_train_v[:,1].min()-1, X_train_v[:,1].max()
+1, 0.01))

Z = clf_vis.predict(np.c_[gx.ravel(), gy.ravel()])
Z = Z.reshape(gx.shape)

plt.contourf(gx, gy, Z, alpha=0.3,
cmap=ListedColormap(('red','green','blue')))
plt.scatter(X_train_v[:,0], X_train_v[:,1], c=y_train_v, edgecolor='k',
marker='o')
plt.title('KNN Decision Boundary (K=5, 2 features)')
plt.show()
```

Repository Structure

```
☣️ KNN-Classification-Task
 ┣ 🈺 knn_classification.ipynb   # Code
 ┣ 🈺 README.md                  # Explanation
 ┣ 🈂️ outputs.png                # Confusion Matrix, Decision Boundary
 ┗ ☣️ data                       # Dataset
```

Key Learning

- KNN is an **instance-based learning algorithm** (no explicit training).
- Choice of K and feature scaling significantly affect performance.
- KNN works well for small, low-dimensional datasets but struggles with large-scale/high-dimensional data.