Assignment Code: DA-AG-016

KNN & PCA | **Assignment**

Instructions: Carefully read each question. Use Google Docs, Microsoft Word, or a similar tool to create a document where you type out each question along with its answer. Save the document as a PDF, and then upload it to the LMS. Please do not zip or archive the files before uploading them. Each question carries 20 marks.

Total Marks: 200

Question 1: What is K-Nearest Neighbors (KNN) and how does it work in both classification and regression problems?

Answer:

K-Nearest Neighbors (KNN) is a simple, non-parametric, and instance-based learning algorithm used for both classification and regression.
It works on the principle that similar data points are located close to each other in the feature space.

- **In Classification:**
  The algorithm identifies the *k* closest data points to a new input instance and assigns the majority class among these neighbors to the new instance.
  Example: If k = 5 and among the nearest neighbors, 3 belong to class A and 2 belong to class B, the instance is classified as class A.
- **In Regression:**
  KNN predicts the output by taking the average (or weighted average) of the target values of the *k* nearest neighbors.

Question 2: What is the Curse of Dimensionality and how does it affect KNN performance?

Answer:

The Curse of Dimensionality refers to the phenomenon where the feature space becomes sparse as the number of dimensions (features) increases, making distance measures less meaningful.

- As dimensions grow, all data points appear equally distant, reducing the effectiveness of the distance-based KNN algorithm.
- It increases computation cost and leads to overfitting.
- Hence, dimensionality reduction techniques like PCA are often used before applying KNN to improve performance.

Question 3: What is Principal Component Analysis (PCA)? How is it different from feature selection?

Answer:

PCA is an unsupervised linear dimensionality reduction technique that transforms correlated features into a smaller set of uncorrelated variables called principal components.

- These components are ordered by the amount of variance they explain in the data.
- PCA does not remove features but creates new ones based on combinations of original features.

Difference from Feature Selection:

- Feature selection removes less important features.
- *PCA* creates new composite features from the original ones, preserving maximum variance.

Question 4: What are eigenvalues and eigenvectors in PCA, and why are they important?

Answer:

- **Eigenvectors** represent the directions (axes) along which data varies the most.
- **Eigenvalues** indicate the amount of variance captured by each eigenvector.

In PCA:

- Eigenvectors define the principal components.
- Eigenvalues determine their significance (variance explained).
  Larger eigenvalues correspond to more important components that explain more data variance.

Question 5: How do KNN and PCA complement each other when applied in a single pipeline?

Answer:

- PCA reduces the dimensionality of the data, removing noise and redundancy.
- This helps KNN by improving distance calculations and reducing computational load.
- Combining PCA and KNN enhances accuracy and efficiency, especially with high-dimensional datasets.

Dataset:

Use the **Wine Dataset** from sklearn.datasets.load_wine().

Question 6: Train a KNN Classifier on the Wine dataset with and without feature scaling. Compare model accuracy in both cases.

(*Include your Python code and output in the code box below.*)
Answer:

Code:

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load dataset
data = load_wine()
X, y = data.data, data.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Without scaling
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
acc_no_scaling = accuracy_score(y_test, y_pred)

# With scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn_scaled = KNeighborsClassifier(n_neighbors=5)
knn_scaled.fit(X_train_scaled, y_train)
y_pred_scaled = knn_scaled.predict(X_test_scaled)
acc_scaled = accuracy_score(y_test, y_pred_scaled)

print("Accuracy without scaling:", acc_no_scaling)
print("Accuracy with scaling:", acc_scaled)
```

Output:

```
Accuracy without scaling: 0.7407407407407407
Accuracy with scaling: 0.9629629629629629
```

Question 7: Train a PCA model on the Wine dataset and print the explained variance ratio of each principal component.

(*Include your Python code and output in the code box below.*)

Answer:

Code:

```
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(X)

print("Variance ratio of each component:")
print(pca.explained_variance_ratio_)
```

Output:

```
Variance ratio of each component:
[9.98091230e-01 1.73591562e-03 9.49589576e-05 5.02173562e-05
 1.23636847e-05 8.46213034e-06 2.80681456e-06 1.52308053e-06
 1.12783044e-06 7.21415811e-07 3.78060267e-07 2.12013755e-07
 8.25392788e-08]
```

Question 8: Train a KNN Classifier on the PCA-transformed dataset (retain top 2 components). Compare the accuracy with the original dataset.

(*Include your Python code and output in the code box below.*)

Answer:

Code:

```
# Import libraries
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load Wine dataset
data = load_wine()
X, y = data.data, data.target

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Scale features
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply PCA (retain top 2 components)
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Train KNN on PCA-transformed data
knn_pca = KNeighborsClassifier(n_neighbors=5)
knn_pca.fit(X_train_pca, y_train)
y_pred_pca = knn_pca.predict(X_test_pca)

# Accuracy using PCA
acc_pca = accuracy_score(y_test, y_pred_pca)

# Compare with accuracy using full scaled dataset
knn_full = KNeighborsClassifier(n_neighbors=5)
knn_full.fit(X_train_scaled, y_train)
y_pred_full = knn_full.predict(X_test_scaled)
acc_full = accuracy_score(y_test, y_pred_full)

# Print results
print("Accuracy using PCA (2 components):", acc_pca)
print("Accuracy using full scaled dataset:", acc_full)
```

Output:

```
Accuracy using PCA (2 components): 0.9814814814814815
Accuracy using full scaled dataset: 0.9629629629629629
```

Question 9: Train a KNN Classifier with different distance metrics (euclidean, manhattan) on the scaled Wine dataset and compare the results.

(*Include your Python code and output in the code box below.*)

Answer:

Code:

```
metrics = ['euclidean', 'manhattan']
for m in metrics:
    knn = KNeighborsClassifier(n_neighbors=5, metric=m)
    knn.fit(X_train_scaled, y_train)
    y_pred = knn.predict(X_test_scaled)
    print(f"Accuracy using {m} distance: {accuracy_score(y_test, y_pred):.2f}")
```

Output:

```
Accuracy using euclidean distance: 0.96
Accuracy using manhattan distance: 0.96
```

Question 10: You are working with a high-dimensional gene expression dataset to classify patients with different types of cancer.
Due to the large number of features and a small number of samples, traditional models overfit.

Explain how you would:

- Use PCA to reduce dimensionality
- Decide how many components to keep
- Use KNN for classification post-dimensionality reduction
- Evaluate the model
- Justify this pipeline to your stakeholders as a robust solution for real-world biomedical data

(*Include your Python code and output in the code box below.*)

Answer:

Step 1: Use PCA to reduce dimensionality

- Apply PCA to capture maximum variance with fewer components.
- This helps remove noise and prevents overfitting due to many irrelevant features.

Step 2: Decide number of components

- Use cumulative explained variance (>90%) to determine optimal components.

Step 3: Use KNN for classification

- Train KNN on PCA-transformed data for robust classification.

Step 4: Evaluate model

- Use train-test split and evaluate using accuracy, precision, recall, and confusion matrix.

Step 5: Justify to stakeholders

- PCA reduces complexity and overfitting.
- KNN ensures interpretability and simplicity.
- Combined pipeline is efficient, scalable, and suitable for biomedical applications.