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:

1. K-Nearest Neighbors (KNN) is a supervised machine learning algorithm.
2. It works based on the similarity (distance) between data points.
3. The value K represents the number of nearest neighbors considered.
4. When a new data point is given, KNN calculates its distance from all training data points.
5. It then selects the K closest data points.

For classification:
6. In classification problems, KNN predicts the class by majority voting of the neighbors.
7. The class with the highest number of neighbors is chosen as the final prediction.

For regression:
8. In regression problems, KNN predicts a value by taking the average of the neighbors'

values.
9. This average becomes the predicted output.

10. KNN does not have a training phase, so it is called a lazy learning algorithm.

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

Answer:

- The Curse of Dimensionality refers to the problems that arise when the number of features (dimensions) in a dataset becomes very large.
- As the number of dimensions increases, the data points become more sparse in the feature space.
- Because of this sparsity, the distance between data points becomes less meaningful.
- KNN relies heavily on distance calculations to find nearest neighbors.
- In high-dimensional space, the distances between nearest and farthest neighbors become almost the same.
- This makes it difficult for KNN to correctly identify truly similar neighbors.
- As a result, the accuracy of KNN decreases with high-dimensional data.
- KNN also becomes computationally expensive because it must calculate distances in many dimensions.
- Therefore, the Curse of Dimensionality negatively affects the performance and efficiency of KNN.

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

Answer:

1. Principal Component Analysis (PCA) is an unsupervised dimensionality reduction technique.
2. It transforms the original features into a new set of features called principal components.
3. These principal components are uncorrelated and arranged in order of maximum variance.
4. PCA reduces the number of features while preserving most of the important information.

Difference from Feature Selection:

5. Feature selection selects a subset of the original features without changing them.
6. PCA, on the other hand, creates new features by combining the original ones.
7. Feature selection keeps the original meaning of features.
8. PCA's components usually do not have a direct physical meaning.
9. Feature selection is often simpler and more interpretable.
10. PCA is more effective when features are highly correlated.

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

Answer:

- In PCA, eigenvectors represent the directions (principal components) along which the data varies the most.
- Eigenvalues represent the amount of variance captured by each eigenvector.
- Eigenvectors are calculated from the covariance matrix of the dataset.
- The eigenvector with the largest eigenvalue shows the direction of maximum variance.
- Eigenvectors with smaller eigenvalues capture less important variations in the data.
- PCA ranks eigenvectors based on their eigenvalues.
- Only the top eigenvectors are selected to reduce dimensionality.
- Eigenvalues help decide how many principal components to keep.
- Eigenvectors define the new feature axes for the transformed data.
- Therefore, eigenvalues and eigenvectors are important because they preserve maximum information while reducing dimensions.

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

Answer:

- The Wine dataset from sklearn.datasets.load_wine() contains multiple chemical features of wine samples.
- Due to the high number of features, KNN performance can degrade because of the Curse of Dimensionality.
- Principal Component Analysis (PCA) is first applied to the Wine dataset to reduce dimensionality.
- PCA transforms the original wine features into a smaller set of principal components.
- These components retain most of the important variance present in the dataset.
- After applying PCA, the dataset becomes lower-dimensional and less noisy.
- KNN is then applied on this transformed dataset for classification.
- With fewer dimensions, distance calculations in KNN become more meaningful and accurate.
- PCA also reduces the computational **cost** of KNN.
- Therefore, combining PCA and KNN in a single pipeline improves both the efficiency and accuracy of wine classification.

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:

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

data = load_wine()
X = data.data
y = data.target


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

knn_no_scaling = KNeighborsClassifier(n_neighbors=5)
knn_no_scaling.fit(X_train, y_train)
y_pred_no_scaling = knn_no_scaling.predict(X_test)
accuracy_no_scaling = accuracy_score(y_test, y_pred_no_scaling)

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

knn_scaling = KNeighborsClassifier(n_neighbors=5)
knn_scaling.fit(X_train_scaled, y_train)
y_pred_scaling = knn_scaling.predict(X_test_scaled)
accuracy_scaling = accuracy_score(y_test, y_pred_scaling)

print("Accuracy without feature scaling:", accuracy_no_scaling)
print("Accuracy with feature scaling:", accuracy_scaling)

#OUTPUT
Accuracy without feature scaling: 0.7407407407407407
Accuracy with feature 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:

```
 from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load Wine dataset
data = load_wine()
X = data.data

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Print explained variance ratio
explained_variance = pca.explained_variance_ratio_
for i, ratio in enumerate(explained_variance):
    print(f"Principal Component {i+1}: {ratio:.4f}")

#OUTPUT

Principal Component 1: 0.3620
Principal Component 2: 0.1921
Principal Component 3: 0.1112
Principal Component 4: 0.0707
Principal Component 5: 0.0656
Principal Component 6: 0.0494
Principal Component 7: 0.0424
Principal Component 8: 0.0268
Principal Component 9: 0.0222
Principal Component 10: 0.0193
Principal Component 11: 0.0174
Principal Component 12: 0.0130
Principal Component 13: 0.0080
```

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:

```
 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

data = load_wine()
X = data.data
y = data.target

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

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

knn_original = KNeighborsClassifier(n_neighbors=5)
knn_original.fit(X_train_scaled, y_train)
y_pred_original = knn_original.predict(X_test_scaled)
accuracy_original = accuracy_score(y_test, y_pred_original)

pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

knn_pca = KNeighborsClassifier(n_neighbors=5)
knn_pca.fit(X_train_pca, y_train)
y_pred_pca = knn_pca.predict(X_test_pca)
accuracy_pca = accuracy_score(y_test, y_pred_pca)

print(f"Accuracy with original scaled data: {accuracy_original:.4f}")
print(f"Accuracy with PCA-transformed data: {accuracy_pca:.4f}")

#OUTPUT

Accuracy with original scaled data: 0.9630
Accuracy with PCA-transformed data: 0.9815
```

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:

```
 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

data = load_wine()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

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


knn_euclidean = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn_euclidean.fit(X_train_scaled, y_train)
y_pred_euclidean = knn_euclidean.predict(X_test_scaled)
accuracy_euclidean = accuracy_score(y_test, y_pred_euclidean)


knn_manhattan = KNeighborsClassifier(n_neighbors=5, metric='manhattan')
knn_manhattan.fit(X_train_scaled, y_train)
y_pred_manhattan = knn_manhattan.predict(X_test_scaled)
accuracy_manhattan = accuracy_score(y_test, y_pred_manhattan)


print("KNN Accuracy with Euclidean distance:", round(accuracy_euclidean,
4))
print("KNN Accuracy with Manhattan distance:", round(accuracy_manhattan,
4))
```

```
#OUTPUT

KNN Accuracy with Euclidean distance: 0.9444
KNN Accuracy with Manhattan distance: 0.9815
```

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:

1. **Use PCA to reduce dimensionality:**

   Standardize the gene expression data.

   Apply PCA to convert thousands of gene features into principal components that capture most variance.

2. **Decide how many components to keep:**

   Check explained variance ratio.

   Keep enough components to retain ~90–95% of the variance.

3. **Use KNN for classification:**

   Train KNN on the PCA-transformed dataset.

   Fewer dimensions make distance calculations more meaningful and reduce overfitting.

4. **Evaluate the model:**

   Use train-test split or cross-validation.

   Metrics: accuracy, precision, recall, F1-score.

5. **Justify to stakeholders:**

   PCA reduces noise and prevents overfitting.

   KNN is simple and interpretable.

   The pipeline is efficient, robust, and retains most information.

```
from sklearn.datasets import make_classification
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

X, y = make_classification(n_samples=100, n_features=1000, n_informative=50,
                           n_classes=3, random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, stratify=y,
random_state=42)

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

pca = PCA(n_components=0.95, random_state=42)
X_train_pca = pca.fit_transform(X_train_scaled)
```

```
X_test_pca = pca.transform(X_test_scaled)

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_pca, y_train)
y_pred = knn.predict(X_test_pca)

print("Number of PCA components:", X_train_pca.shape[1])
print("KNN Accuracy:", round(accuracy_score(y_test, y_pred), 4))

#OUTPUT

Number of PCA components: 35
KNN Accuracy: 0.8667
```