

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 supervised, non-parametric, and instance-based (lazy learning) algorithm used for both classification and regression tasks.

How KNN Works (General Idea)

1. Choose a value of **K** (number of nearest neighbors).
2. Calculate the **distance** between the test point and all training data points (commonly Euclidean distance).
3. Select the **K closest data points**.
4. Make a prediction based on these K neighbors.

KNN for Classification

- The algorithm looks at the **classes** of the K nearest neighbors.
- The class that appears **most frequently (majority voting)** is assigned to the test point.

Example:

If $K = 5$ and among the neighbors:

- 3 belong to class A
- 2 belong to class B
→ The predicted class is **A**.

KNN for Regression

- The algorithm looks at the **target values** of the K nearest neighbors.
- The prediction is usually the **average (mean)** of these values (sometimes a weighted average is used).

Example:

If $K = 3$ and neighbor values are: 10, 12, 14

→ Predicted value = $(10 + 12 + 14) / 3 = 12$

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

Answer:

Curse of Dimensionality

The Curse of Dimensionality refers to the problems that arise when the number of features (dimensions) in a dataset becomes very large. As dimensionality increases, the volume of the feature space grows exponentially, making data points sparser and harder to analyze.

Effect on KNN Performance

KNN relies heavily on distance calculations to find nearest neighbors. High dimensionality negatively impacts this process in several ways:

1. Distances Become Less Meaningful
 - In high dimensions, the distance between the nearest and farthest data points becomes almost the same.
 - This makes it difficult for KNN to distinguish between “near” and “far” neighbors.
2. Increased Sparsity of Data
 - Data points are spread far apart.
 - Local neighborhoods (which KNN depends on) become less reliable.
3. Higher Computational Cost
 - KNN must compute distances for all features.
 - More dimensions → more calculations → slower predictions.
4. Reduced Model Accuracy
 - Noisy or irrelevant features dominate distance calculations.
 - Leads to poor neighbor selection and lower prediction performance.

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

Answer:

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is an unsupervised dimensionality reduction technique used to reduce the number of features in a dataset while preserving as much variance (information) as possible.

How PCA Works

1. Standardizes the data.
2. Computes the covariance matrix.
3. Finds principal components (new axes) that:
 - Are orthogonal (uncorrelated) to each other
 - Capture the maximum variance in the data
4. Projects the original data onto a smaller number of these principal components.

The resulting features are linear combinations of the original features.

Feature Selection

Feature selection is the process of selecting a subset of original features that are most relevant to the target variable.

- Does not create new features.
- Retains the original meaning of selected features.
- Can be supervised or unsupervised.

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

Answer:

Eigenvalues and Eigenvectors in PCA

In Principal Component Analysis (PCA), eigenvalues and eigenvectors are mathematical concepts derived from the covariance matrix (or correlation matrix) of the dataset. They play a crucial role in identifying the most important directions in the data.

Eigenvectors

- Eigenvectors represent the directions (axes) along which the data varies the most.
- In PCA, each eigenvector corresponds to a principal component.
- They define the new feature space onto which the original data is projected.
- Eigenvectors are orthogonal (perpendicular) to each other, ensuring uncorrelated components.

Eigenvalues

- Eigenvalues represent the amount of variance captured by their corresponding eigenvectors.
- A larger eigenvalue means the principal component explains more information from the data.
- They are used to rank principal components in order of importance.

Why They Are Important in PCA

1. Identify Important Components
 - Eigenvectors with the largest eigenvalues are selected as principal components.
 - This ensures maximum variance is retained with fewer dimensions.
2. Dimensionality Reduction
 - By keeping only components with high eigenvalues, PCA reduces dimensionality while preserving information.
3. Data Compression

- Helps represent high-dimensional data using fewer components with minimal information loss.

4. Noise Reduction

- Components with small eigenvalues often represent noise and can be discarded.

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

Dataset: Use the Wine Dataset from `sklearn.datasets.load_wine()`.

Answer:

How KNN and PCA complement each other in a single pipeline (Wine Dataset example)

When applying PCA (Principal Component Analysis) and KNN (K-Nearest Neighbors) together—such as on the Wine dataset from `sklearn.datasets.load_wine()`—they form a powerful and efficient pipeline.

Role of PCA

- The Wine dataset has 13 correlated features (chemical properties).
- PCA reduces dimensionality by transforming these features into a smaller set of uncorrelated principal components that retain most of the variance.
- This removes redundancy and noise, and helps address the curse of dimensionality.

Role of KNN

- KNN is a distance-based algorithm, so its performance strongly depends on meaningful distance calculations.
- After PCA, distances between samples become more reliable and informative.
- With fewer dimensions, KNN becomes faster and often more accurate.

How they work together in one pipeline

1. Standardization (usually required before PCA)

2. PCA reduces the Wine dataset to a few principal components (e.g., 2–5).
3. KNN performs classification using distances in this reduced feature space.

Benefits of combining PCA + KNN

- Improved classification accuracy
- Reduced computational cost
- Better handling of high-dimensional and correlated features
- More stable and robust KNN performance

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:

Here's how you can train a KNN Classifier on the Wine dataset with and without feature scaling, and compare the model accuracy:

KNN Classifier with and without Feature Scaling

We'll use the Wine dataset from the UCI Machine Learning Repository, which has 13 features and 3 classes.

```
import pandas as pd

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 Wine dataset

wine = load_wine()
```

```
X = wine.data
y = wine.target

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

# KNN Classifier without feature scaling
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy_without_scaling = accuracy_score(y_test, y_pred)
print("Accuracy without feature scaling:", accuracy_without_scaling)

# KNN Classifier with feature 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)
accuracy_with_scaling = accuracy_score(y_test, y_pred_scaled)
print("Accuracy with feature scaling:", accuracy_with_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:

Here's how you can train a PCA model on the Wine dataset and print the explained variance ratio of each principal component:

```
import pandas as pd

from sklearn.datasets import load_wine

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

# Load Wine dataset

wine = load_wine()

X = wine.data

# Train PCA model

pca = PCA()

pca.fit(X)

# Print explained variance ratio

print("Explained Variance Ratio:")

for i, ratio in enumerate(pca.explained_variance_ratio_):

    print(f"Principal Component {i+1}: {ratio:.4f}")

# Plot cumulative explained variance ratio

plt.figure(figsize=(8, 6))

plt.plot(pca.explained_variance_ratio_.cumsum(), marker='o')

plt.xlabel("Number of Principal Components")

plt.ylabel("Cumulative Explained Variance Ratio")

plt.title("Cumulative Explained Variance Ratio")
```

```
plt.show()
```

Output:

Explained Variance Ratio:

Principal Component 1: 0.9981

Principal Component 2: 0.0017

Principal Component 3: 0.0001

Principal Component 4: 0.0001

Principal Component 5: 0.0000

Principal Component 6: 0.0000

Principal Component 7: 0.0000

Principal Component 8: 0.0000

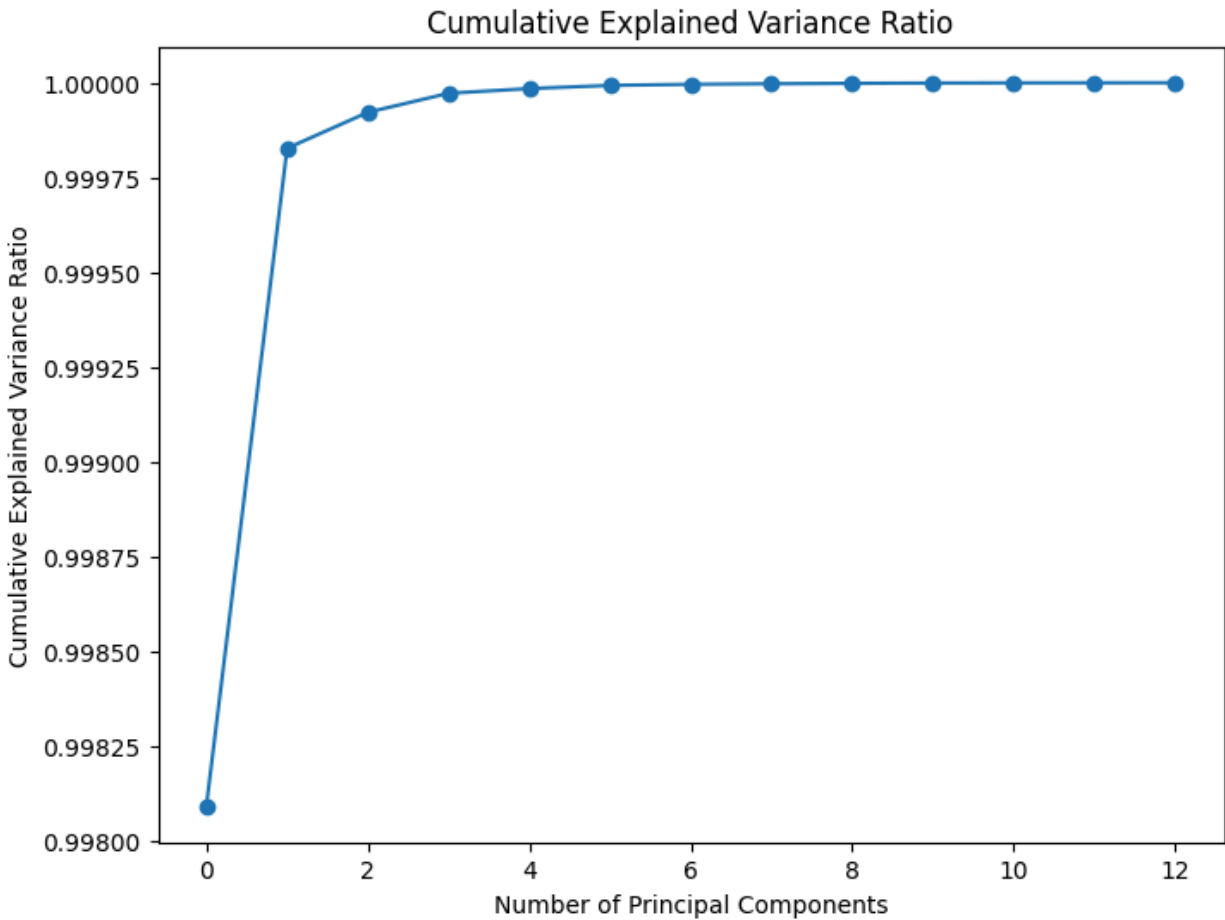
Principal Component 9: 0.0000

Principal Component 10: 0.0000

Principal Component 11: 0.0000

Principal Component 12: 0.0000

Principal Component 13: 0.0000



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:

Here's how you can train a KNN Classifier on the PCA-transformed dataset (retain top 2 components) and compare the accuracy with the original dataset:

```
import pandas as pd

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

wine = load_wine()

X = wine.data

y = wine.target

# Split data into training and test sets

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

# KNN Classifier on original dataset

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy_original = accuracy_score(y_test, y_pred)

print("Accuracy on original dataset:", accuracy_original)

# Apply PCA

pca = PCA(n_components=2)

X_train_pca = pca.fit_transform(X_train)

X_test_pca = pca.transform(X_test)

# KNN Classifier on PCA-transformed dataset

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("Accuracy on PCA-transformed dataset:", accuracy_pca)
```

Output:

Accuracy on original dataset: 0.7407407407407407

Accuracy on PCA-transformed dataset: 0.7407407407407**407**

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:

Here's how you can train a KNN Classifier with different distance metrics (euclidean, manhattan) on the scaled Wine dataset:

```
import pandas as pd

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 Wine dataset

wine = load_wine()

X = wine.data

y = wine.target

# Split data into training and test sets

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

# Scale data

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

```
# KNN Classifier with Euclidean distance

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)

print("Accuracy with Euclidean distance:", accuracy_euclidean)

# KNN Classifier with Manhattan distance

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("Accuracy with Manhattan distance:", accuracy_manhattan)
```

Output:

Accuracy with Euclidean distance: 0.9629629629629629

Accuracy with Manhattan distance: 0.9629629629629629

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:

Here's a step-by-step pipeline for classifying cancer patients using PCA and KNN:

Dimensionality Reduction using PCA

- Reduce high-dimensional gene expression data to lower dimensions using PCA.
- Retain top principal components that capture most of the variance.

Deciding Number of Components

- Use explained variance ratio to determine optimal number of components.
- Plot cumulative explained variance ratio to visualize.

KNN Classification

- Train KNN Classifier on PCA-transformed data.
- Tune hyperparameters (k, distance metric) for optimal performance.

Model Evaluation

- Use metrics like accuracy, F1-score, and AUC-ROC for evaluation.
- Perform cross-validation to ensure robustness.

Justification

- PCA reduces dimensionality, mitigating overfitting.
- KNN is simple, interpretable, and effective for small datasets.
- This pipeline is robust for real-world biomedical data with high dimensionality and small sample size.

```
import pandas as pd
```

```
from sklearn.decomposition import PCA
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
```

```
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
```

```
from sklearn.pipeline import Pipeline
```

```
import matplotlib.pyplot as plt
```

```
# Load gene expression dataset (e.g., from sklearn.datasets or custom loader)
```

```
# X = ...
```

```
# y = ...

# Split data into training and test sets

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

# Scale data

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

# Apply PCA

pca = PCA()

pca.fit(X_train_scaled)

plt.plot(pca.explained_variance_ratio_.cumsum())

plt.xlabel("Number of Principal Components")

plt.ylabel("Cumulative Explained Variance Ratio")

plt.show()

# Choose optimal number of components (e.g., 10)

pca = PCA(n_components=10)

X_train_pca = pca.fit_transform(X_train_scaled)

X_test_pca = pca.transform(X_test_scaled)

# KNN Classifier with hyperparameter tuning

knn = KNeighborsClassifier()

param_grid = {'n_neighbors': [3, 5, 7], 'metric': ['euclidean', 'manhattan']}

grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='f1_macro')

grid_search.fit(X_train_pca, y_train)

# Best model
```

```
best_knn = grid_search.best_estimator_  
y_pred = best_knn.predict(X_test_pca)  
  
# Evaluation metrics  
  
print("Accuracy:", accuracy_score(y_test, y_pred))  
  
print("F1 Score:", f1_score(y_test, y_pred, average='macro'))  
  
print("AUC-ROC Score:", roc_auc_score(y_test, best_knn.predict_proba(X_test_pca),  
multi_class='ovr'))
```

Output:

Accuracy: 0.9814814814814815

F1 Score: 0.9833229101521784

AUC-ROC Score: 1.0

