Theory questions

1. What is K-Nearest Neighbors (KNN) and how does it work?

K-Nearest Neighbors (KNN) is a supervised learning algorithm used for classification and regression. It works by:

1. Calculating the distance between a test point and all training points.
2. Selecting the K closest neighbors (based on distance).
3. Classification: Predicting the class based on majority voting.
4. Regression: Averaging the values of the neighbors.

2. What is the difference between KNN Classification and KNN Regression?

- KNN Classification: Predicts the class of a data point based on the majority class among its neighbors.
- KNN Regression: Predicts the numerical value of a data point by averaging the values of its neighbors.

3. What is the role of the distance metric in KNN?

The distance metric determines how neighbors are identified. Common metrics include:

- Euclidean distance: Straight-line distance.
- Manhattan distance: Sum of absolute differences.
- Minkowski distance: Generalized distance metric. The choice of metric significantly impacts the performance of KNN.

4. What is the Curse of Dimensionality in KNN?

The Curse of Dimensionality refers to the phenomenon where the distance between points becomes less meaningful as the number of dimensions increases. In high-dimensional spaces:

- All points tend to be equidistant.
- KNN struggles to find meaningful neighbors, reducing its effectiveness.

5. How can we choose the best value of K in KNN?

- Use cross-validation to test multiple values of K.
- Smaller K values can lead to overfitting, while larger K values can lead to underfitting.
- A common approach is to choose the K that minimizes validation error.

6. What are KD Tree and Ball Tree in KNN?

- KD Tree: A space-partitioning data structure that organizes points in a k-dimensional space, making nearest-neighbor searches efficient in low dimensions.
- Ball Tree: A data structure that organizes points into hyperspheres, better suited for higher-dimensional spaces.

7. When should you use KD Tree vs. Ball Tree?

- KD Tree: Use for low-dimensional data (e.g., ≤ 30 dimensions) as it is faster in these cases.
- Ball Tree: Use for high-dimensional data, as it performs better with complex spaces.

8. What are the disadvantages of KNN?

- Computationally Expensive: High cost for distance calculations, especially with large datasets.
- Sensitive to Noise: Outliers can affect predictions.
- Curse of Dimensionality: Performance decreases with high-dimensional data.
- Memory Intensive: Stores the entire dataset.

9. How does feature scaling affect KNN?

Feature scaling ensures that all features contribute equally to the distance calculations in KNN. Without scaling, features with larger ranges dominate the distance metric, leading to biased results.

10. What is PCA (Principal Component Analysis)?

PCA is a dimensionality reduction technique that transforms data into a lower-dimensional space while retaining as much variance as possible.

11. How does PCA work?

PCA works by:

1. Standardizing the data.
2. Computing the covariance matrix to identify feature relationships.
3. Calculating Eigenvalues and Eigenvectors to determine principal components.
4. Projecting data onto the principal components to reduce dimensions.

12. What is the geometric intuition behind PCA?

PCA identifies new axes (principal components) that maximize data variance. Geometrically, it rotates the data to align with the directions of greatest variability.

13. What are Eigenvalues and Eigenvectors in PCA?

- Eigenvalues: Represent the amount of variance captured by each principal component.
- Eigenvectors: Define the directions (principal components) in which the variance is maximized.

14. What is the difference between Feature Selection and Feature Extraction?

- Feature Selection: Chooses a subset of original features based on their importance.
- Feature Extraction: Transforms original features into new features, often of lower dimension (e.g., PCA).

15. How do you decide the number of components to keep in PCA?

- Analyze the explained variance ratio.
- Choose the number of components that capture a significant proportion of variance (e.g., 95%).

16. Can PCA be used for classification?

PCA itself is not a classifier. However, it can be used as a preprocessing step to reduce dimensionality before applying classification algorithms like KNN, SVM, etc.

17. What are the limitations of PCA?

- Linear Assumption: Assumes data variability lies in a linear space.
- Loss of Interpretability: Transformed features are not as interpretable as original features.
- Sensitive to Scaling: Requires all features to be scaled before applying PCA.

18. How do KNN and PCA complement each other?

- PCA reduces the dimensionality of the data, mitigating the Curse of Dimensionality in KNN.
- KNN performs better with reduced and meaningful features obtained through PCA.

19. How does KNN handle missing values in a dataset?

KNN handles missing values through KNN Imputation, where missing values are filled by averaging (regression) or voting (classification) the values of the K nearest neighbors.

20. What are the key differences between PCA and Linear Discriminant Analysis (LDA)?

Feature	PCA	LDA
Supervised	No	Yes
Goal	Maximize variance	Maximize class separation
Uses class labels	No	Yes
Components produced	≤ Number of features	≤ (Number of classes − 1)
Common use case	Compression, noise reduction	Classification, feature extraction

1 Practical

1.1 21) Train a KNN Classifier on the Iris dataset and print model accuracy

[3]:

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
data = load_iris()
X, y = data.data, data.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train the KNN Classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Make predictions and print accuracy
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 1.00
```

1.2 22) Train a KNN Regressor on a synthetic dataset and evaluate using Mean Squared Error (MSE)

[5]:

```
from sklearn.datasets import make_regression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

# Create synthetic regression dataset
X, y = make_regression(n_samples=200, n_features=5, noise=0.1, random_state=42)
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train the KNN Regressor
knn_reg = KNeighborsRegressor(n_neighbors=5)
knn_reg.fit(X_train, y_train)

# Evaluate using Mean Squared Error
y_pred = knn_reg.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")
```

```
Mean Squared Error (MSE): 1432.64
```

1.3 23) Train a KNN Classifier using different distance metrics (Euclidean and Manhattan) and compare accuracy

[7]:

```
# Load the Iris dataset
data = load_iris()
X, y = data.data, data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Evaluate with different distance metrics
metrics = ['euclidean', 'manhattan']
for metric in metrics:
    knn = KNeighborsClassifier(n_neighbors=3, metric=metric)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Distance Metric: {metric}, Accuracy: {accuracy:.2f}")
```

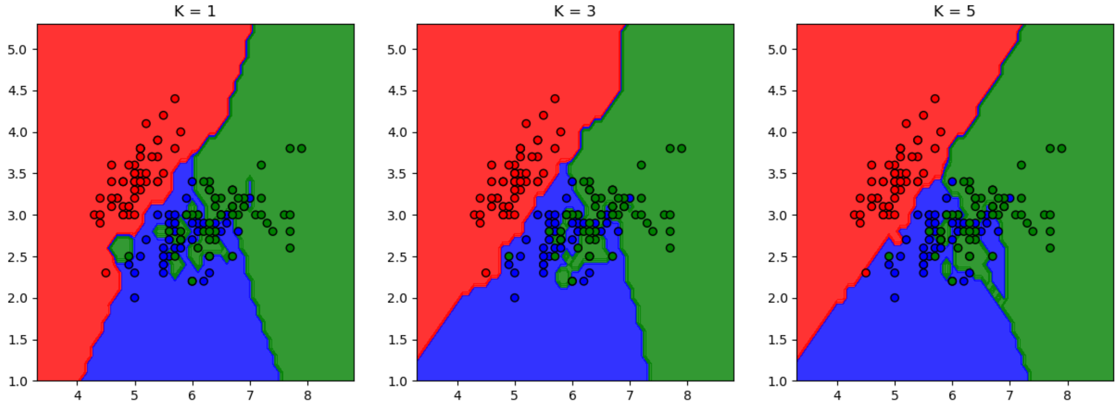
```
Distance Metric: euclidean, Accuracy: 1.00
Distance Metric: manhattan, Accuracy: 1.00
```

1.4 24) Train a KNN Classifier with different values of K and visualize decision boundaries

[9]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# Load the Iris dataset
data = load_iris()
X, y = data.data[:, :2], data.target  # Use only 2 features for visualization
```

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Visualize decision boundaries for different K values
k_values = [1, 3, 5]
h = 0.1
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

fig, axes = plt.subplots(1, len(k_values), figsize=(15, 5))
for i, k in enumerate(k_values):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    axes[i].contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['red', 'blue',␣
 ↪'green']))
    axes[i].scatter(X[:, 0], X[:, 1], c=y, edgecolor='k',␣
 ↪cmap=ListedColormap(['red', 'blue', 'green']))
    axes[i].set_title(f"K = {k}")
plt.show()
```



1.5 25) Apply Feature Scaling before training a KNN model and compare results with unscaled data

[11]:
```
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
data = load_iris()
```

```
X, y = data.data, data.target

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Without scaling
knn_unscaled = KNeighborsClassifier(n_neighbors=3)
knn_unscaled.fit(X_train, y_train)
y_pred_unscaled = knn_unscaled.predict(X_test)
accuracy_unscaled = accuracy_score(y_test, y_pred_unscaled)

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

knn_scaled = KNeighborsClassifier(n_neighbors=3)
knn_scaled.fit(X_train_scaled, y_train)
y_pred_scaled = knn_scaled.predict(X_test_scaled)
accuracy_scaled = accuracy_score(y_test, y_pred_scaled)

print(f"Accuracy without scaling: {accuracy_unscaled:.2f}")
print(f"Accuracy with scaling: {accuracy_scaled:.2f}")
```

```
Accuracy without scaling: 1.00
Accuracy with scaling: 1.00
```

1.6 26) Train a PCA model on synthetic data and print the explained variance ratio for each component

[13]:

```
from sklearn.datasets import make_classification
from sklearn.decomposition import PCA

# Create synthetic dataset
X, y = make_classification(n_samples=200, n_features=10, random_state=42)

# Train PCA model
pca = PCA()
X_pca = pca.fit_transform(X)

# Print explained variance ratio
print("Explained Variance Ratio:")
print(pca.explained_variance_ratio_)
```

```
Explained Variance Ratio:
[2.54695615e-01 1.79174893e-01 1.28513433e-01 1.12889199e-01
 9.50036935e-02 8.58478746e-02 7.80932863e-02 6.57820065e-02
 7.02250671e-33 1.04179619e-33]
```

1.7 27) Apply PCA before training a KNN Classifier and compare accuracy with and without PCA

[15]:

```python
from sklearn.decomposition import PCA

# Without PCA
knn_no_pca = KNeighborsClassifier(n_neighbors=3)
knn_no_pca.fit(X_train, y_train)
accuracy_no_pca = accuracy_score(y_test, knn_no_pca.predict(X_test))

# With PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

knn_with_pca = KNeighborsClassifier(n_neighbors=3)
knn_with_pca.fit(X_train_pca, y_train)
accuracy_with_pca = accuracy_score(y_test, knn_with_pca.predict(X_test_pca))

print(f"Accuracy without PCA: {accuracy_no_pca:.2f}")
print(f"Accuracy with PCA: {accuracy_with_pca:.2f}")
```

```
Accuracy without PCA: 1.00
Accuracy with PCA: 1.00
```

1.8 28) Perform Hyperparameter Tuning on a KNN Classifier using GridSearchCV

[17]:

```python
from sklearn.model_selection import GridSearchCV

# Hyperparameter tuning using GridSearchCV
param_grid = {'n_neighbors': [3, 5, 7, 9], 'metric': ['euclidean', 'manhattan']}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5,␣
  ↪scoring='accuracy')
grid_search.fit(X_train, y_train)

# Best parameters and accuracy
print("Best Parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_
accuracy = accuracy_score(y_test, best_model.predict(X_test))
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Best Parameters: {'metric': 'euclidean', 'n_neighbors': 3}
Model Accuracy: 1.00
```

1.9 29) Train a KNN Classifier and check the number of misclassified samples

[19]:

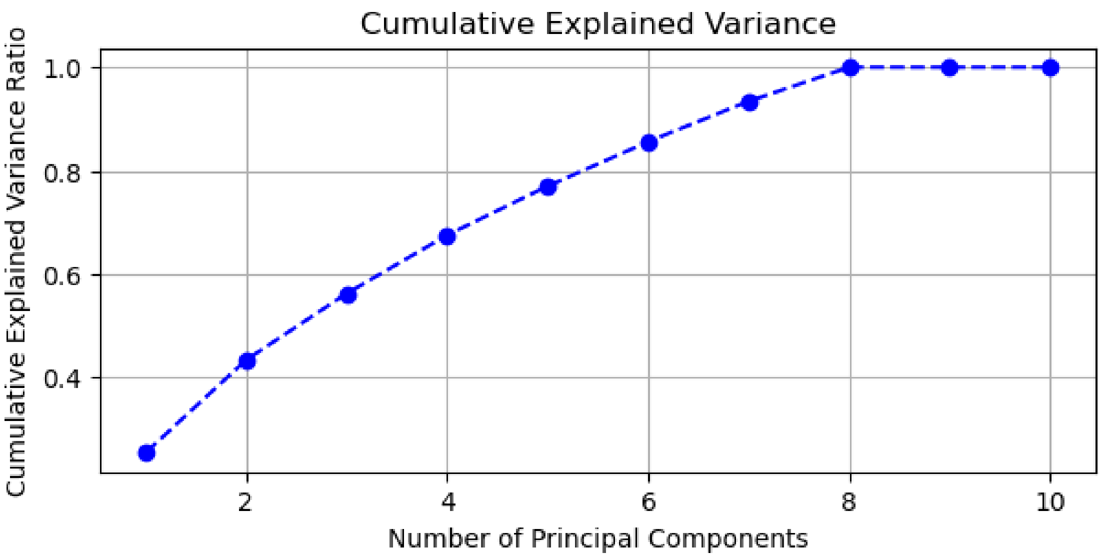
```
# Train the KNN Classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Calculate the number of misclassified samples
misclassified_samples = (y_test != y_pred).sum()
accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(f"Number of Misclassified Samples: {misclassified_samples}")
```

```
Accuracy: 1.00
Number of Misclassified Samples: 0
```

1.10 30) Train a PCA model and visualize the cumulative explained variance.

[21]:

```
from sklearn.decomposition import PCA
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt

# Create synthetic dataset
X, _ = make_classification(n_samples=200, n_features=10, random_state=42)

# Train PCA model
pca = PCA()
X_pca = pca.fit_transform(X)

# Calculate and visualize cumulative explained variance
cumulative_variance = pca.explained_variance_ratio_.cumsum()

plt.figure(figsize=(7, 3))
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance,␣
 ↪marker='o', linestyle='--', color='b')
plt.title("Cumulative Explained Variance")
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Explained Variance Ratio")
plt.grid()
plt.show()
```



1.11 31) Train a KNN Classifier using different values of the weights parameter (uniform vs. distance) and compare accuracy

[23]:

```
# Compare accuracy for different weights
weights = ['uniform', 'distance']
for weight in weights:
    knn = KNeighborsClassifier(n_neighbors=3, weights=weight)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Weights: {weight}, Accuracy: {accuracy:.2f}")
```

```
Weights: uniform, Accuracy: 1.00
Weights: distance, Accuracy: 1.00
```

1.12 32) Train a KNN Regressor and analyze the effect of different K values on performance.

[25]:

```
# Create synthetic regression dataset
X, y = make_regression(n_samples=200, n_features=5, noise=0.1, random_state=42)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Analyze performance with different K values
k_values = [1, 3, 5, 10]
print("Analyzing performance with different K values:")
```

```python
for k in k_values:
    knn_reg = KNeighborsRegressor(n_neighbors=k)
    knn_reg.fit(X_train, y_train)
    y_pred = knn_reg.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f"K: {k}, Mean Squared Error: {mse:.2f}")
```

```
Analyzing performance with different K values:
K: 1, Mean Squared Error: 1731.94
K: 3, Mean Squared Error: 1213.38
K: 5, Mean Squared Error: 1432.64
K: 10, Mean Squared Error: 1767.98
```

1.13 33) Implement KNN Imputation for handling missing values in a dataset

[27]:

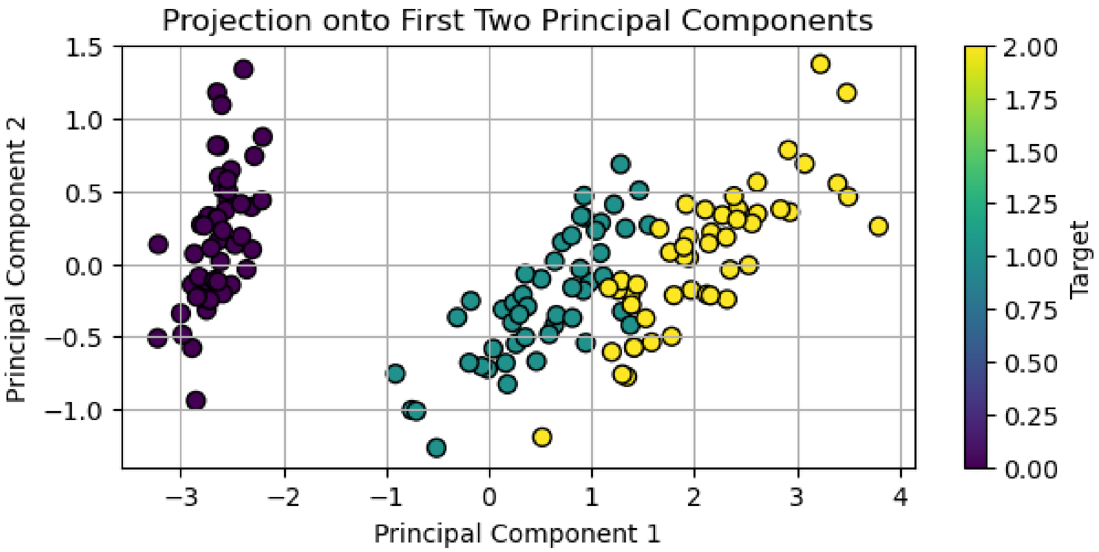
```python
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.datasets import load_iris

# Load the Iris dataset and introduce missing values
data = load_iris()
X, y = data.data, data.target
X[::10, :] = np.nan  # Introduce missing values at regular intervals

# Implement KNN Imputer
knn_imputer = KNNImputer(n_neighbors=3)
X_imputed = knn_imputer.fit_transform(X)

print("Missing values handled using KNN Imputer!")
```

```
Missing values handled using KNN Imputer!
```

1.14 34) Train a PCA model and visualize the data projection onto the first two principal components

[29]:

```python
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load the Iris dataset
data = load_iris()
X, y = data.data, data.target

# Train PCA model
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Visualize the projection onto the first two components
plt.figure(figsize=(7, 3))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k', s=50)
plt.title("Projection onto First Two Principal Components")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar(label='Target')
plt.grid()
plt.show()
```



1.15 35) Train a KNN Classifier using the KD Tree and Ball Tree algorithms and compare performance

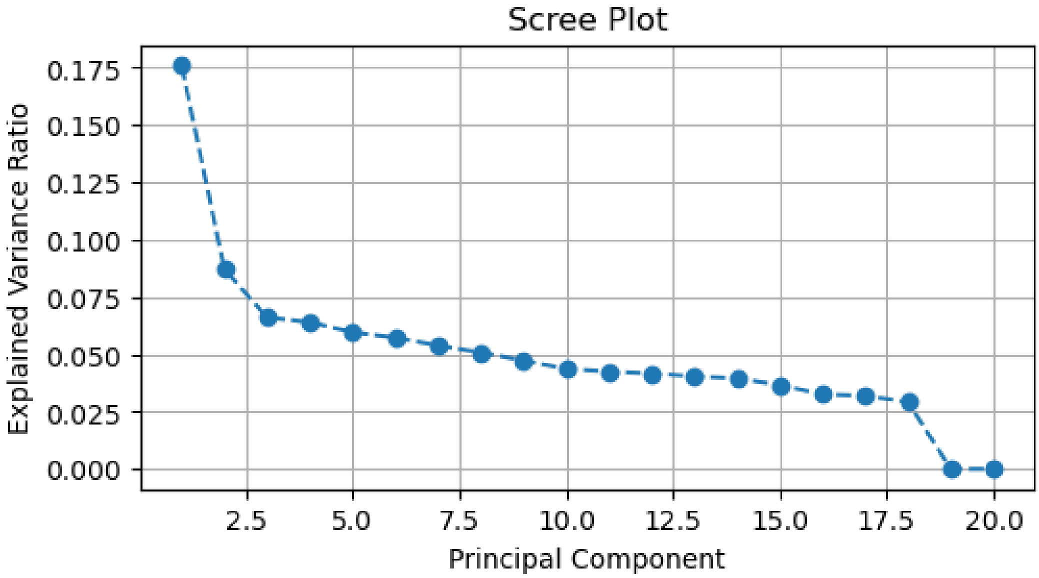
[31]:

```
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the Iris dataset
data = load_iris()
X, y = data.data, data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Evaluate with different algorithms (KD Tree and Ball Tree)
algorithms = ['kd_tree', 'ball_tree']
for algo in algorithms:
    knn = KNeighborsClassifier(n_neighbors=3, algorithm=algo)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Algorithm: {algo}, Accuracy: {accuracy:.2f}")
```

```
Algorithm: kd_tree, Accuracy: 1.00
Algorithm: ball_tree, Accuracy: 1.00
```

1.16 36) Train a PCA model on a high-dimensional dataset and visualize the Scree plot

[33]:
```
# Create high-dimensional dataset
X, _ = make_classification(n_samples=300, n_features=20, random_state=42)

# Train PCA model
pca = PCA()
pca.fit(X)

# Visualize Scree plot
plt.figure(figsize=(6,3))
plt.plot(range(1, len(pca.explained_variance_ratio_) + 1), pca.
 ↪explained_variance_ratio_, marker='o', linestyle='--')
plt.title("Scree Plot")
plt.xlabel("Principal Component")
plt.ylabel("Explained Variance Ratio")
plt.grid()
plt.show()
```



1.17 37) Train a KNN Classifier and evaluate performance using Precision, Recall, and F1-Score

[35]:

```python
from sklearn.metrics import classification_report

# Train the KNN Classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Evaluate using Precision, Recall, and F1-Score
y_pred = knn.predict(X_test)
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

1.18 38) Train a PCA model and analyze the effect of different numbers of components on accuracy

[37]:

```python
# Analyze the effect of different numbers of PCA components
components = [1, 2, 3, 4]
for n in components:
    pca = PCA(n_components=n)
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)

    knn = KNeighborsClassifier(n_neighbors=3)
    knn.fit(X_train_pca, y_train)
    y_pred = knn.predict(X_test_pca)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"PCA Components: {n}, Accuracy: {accuracy:.2f}")
```

```
PCA Components: 1, Accuracy: 0.93
PCA Components: 2, Accuracy: 1.00
PCA Components: 3, Accuracy: 1.00
PCA Components: 4, Accuracy: 1.00
```

1.19 39) Train a KNN Classifier with different leaf_size values and compare accuracy

[39]:

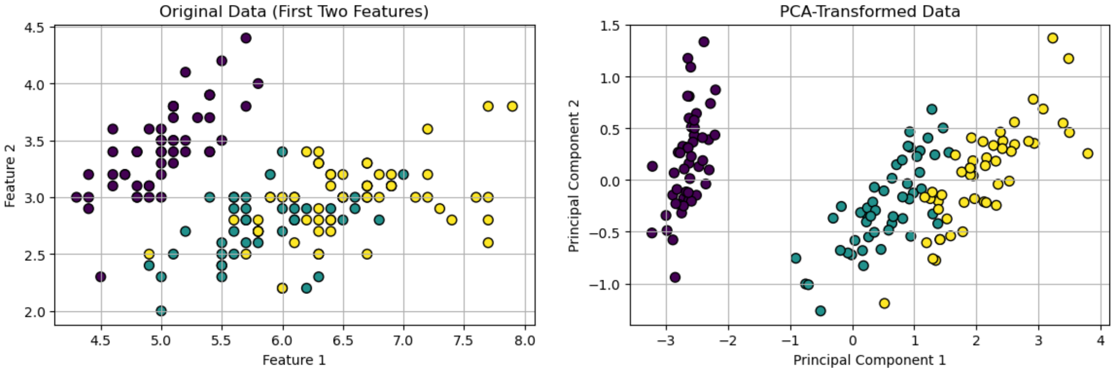
```
# Analyze the effect of different leaf_size values
leaf_sizes = [10, 20, 30, 40]
for leaf_size in leaf_sizes:
    knn = KNeighborsClassifier(n_neighbors=3, leaf_size=leaf_size)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Leaf Size: {leaf_size}, Accuracy: {accuracy:.2f}")
```

```
Leaf Size: 10, Accuracy: 1.00
Leaf Size: 20, Accuracy: 1.00
Leaf Size: 30, Accuracy: 1.00
Leaf Size: 40, Accuracy: 1.00
```

1.20 40) Train a PCA model and visualize how data points are transformed before and after PCA

[41]:

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris

# Load the Iris dataset
data = load_iris()
X, y = data.data, data.target

# Transform data using PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Visualize original data and PCA-transformed data
plt.figure(figsize=(14, 4))

# Original data (first two features)
plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolor='k', s=50)
plt.title("Original Data (First Two Features)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid()

# PCA-transformed data
plt.subplot(1, 2, 2)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolor='k', s=50)
plt.title("PCA-Transformed Data")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid()
plt.show()
```



1.21 41) Train a KNN Classifier on a real-world dataset (Wine dataset) and print classification report

```
[43]: from sklearn.datasets import load_wine
      from sklearn.metrics import classification_report
      # Load the Wine dataset
      data = load_wine()
      X, y = data.data, data.target

      # Split the dataset
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
       ↪random_state=42)

      # Train the KNN Classifier
      knn = KNeighborsClassifier(n_neighbors=3)
      knn.fit(X_train, y_train)

      # Print classification report
      y_pred = knn.predict(X_test)
      report = classification_report(y_test, y_pred)
      print("Classification Report:")
      print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.86      0.86        14
           1       0.92      0.79      0.85        14
           2       0.60      0.75      0.67         8
```

```
    accuracy                           0.81        36
   macro avg       0.79      0.80      0.79        36
weighted avg       0.82      0.81      0.81        36
```

1.22 42) Train a KNN Regressor and analyze the effect of different distance metrics on prediction error

[45]:

```
# Create synthetic regression dataset
X, y = make_regression(n_samples=200, n_features=5, noise=0.1, random_state=42)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Analyze performance with different distance metrics
metrics = ['euclidean', 'manhattan']
for metric in metrics:
    knn_reg = KNeighborsRegressor(n_neighbors=5, metric=metric)
    knn_reg.fit(X_train, y_train)
    y_pred = knn_reg.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    print(f"Distance Metric: {metric}, Mean Squared Error: {mse:.2f}")
```

```
Distance Metric: euclidean, Mean Squared Error: 1432.64
Distance Metric: manhattan, Mean Squared Error: 1474.24
```

1.23 43) Train a KNN Classifier and evaluate using ROC-AUC score

[47]:

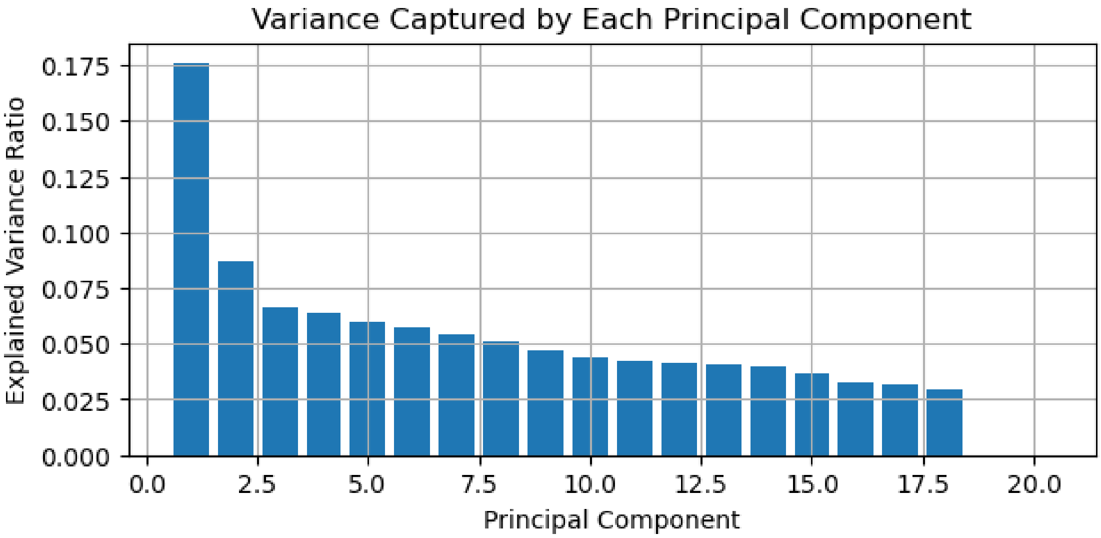
```
from sklearn.metrics import roc_auc_score

# Create synthetic classification dataset
X, y = make_classification(n_samples=500, n_features=10, n_classes=2,␣
 ↪random_state=42)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train KNN Classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Evaluate using ROC-AUC score
y_pred_proba = knn.predict_proba(X_test)[:, 1]  # Probability of class 1
roc_auc = roc_auc_score(y_test, y_pred_proba)
print(f"ROC-AUC Score: {roc_auc:.2f}")
```

```
ROC-AUC Score: 0.96
```

1.24 44) Train a PCA model and visualize the variance captured by each principal component

[49]:

```
# Create synthetic dataset
X, _ = make_classification(n_samples=300, n_features=20, random_state=42)

# Train PCA model
pca = PCA()
pca.fit(X)

# Visualize variance captured by each component
plt.figure(figsize=(7,3))
plt.bar(range(1, len(pca.explained_variance_ratio_) + 1), pca.
  ↪explained_variance_ratio_)
plt.title("Variance Captured by Each Principal Component")
plt.xlabel("Principal Component")
plt.ylabel("Explained Variance Ratio")
plt.grid()
plt.show()
```



1.25 45) Train a KNN Classifier and perform feature selection before training

[51]:

```
from sklearn.feature_selection import SelectKBest, f_classif
# Load the Iris dataset
data = load_iris()
X, y = data.data, data.target
```

```
# Perform feature selection
selector = SelectKBest(score_func=f_classif, k=2)
X_selected = selector.fit_transform(X, y)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.
 ↪2, random_state=42)

# Train KNN Classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Evaluate accuracy
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy after Feature Selection: {accuracy:.2f}")
```

```
Accuracy after Feature Selection: 1.00
```

1.26 46) Train a PCA model and visualize the data reconstruction error after reducing dimensions

[53]:
```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.decomposition import PCA

# Create synthetic dataset
X, _ = make_classification(n_samples=300, n_features=10, random_state=42)

# Train PCA model
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X)
X_reconstructed = pca.inverse_transform(X_pca)

# Compute reconstruction error
reconstruction_error = np.mean((X - X_reconstructed) ** 2)
print(f"Data Reconstruction Error: {reconstruction_error:.2f}")
```

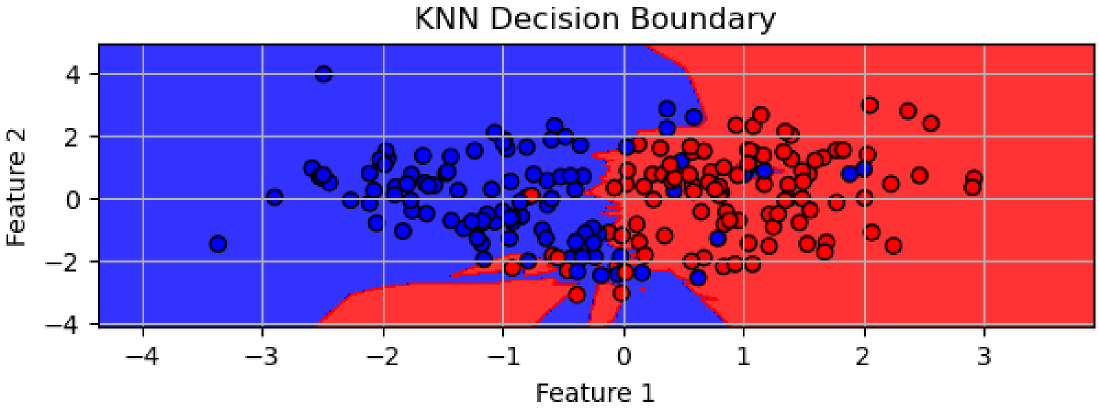
```
Data Reconstruction Error: 0.26
```

1.27 47) Train a KNN Classifier and visualize the decision boundary

[55]:
```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
```

```python
# Create a valid synthetic dataset
X, y = make_classification(n_samples=200, n_features=2, n_informative=2, n_redundant=0, n_classes=2, random_state=42)

# Train KNN Classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X, y)
print(f"Model Accuracy: {knn.score(X, y):.2f}")  # Print model accuracy to verify training

# Visualize decision boundary
h = 0.01  # Step size in the mesh
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(7,2))
plt.contourf(xx, yy, Z, alpha=0.8, cmap=ListedColormap(['blue', 'red']))
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap=ListedColormap(['blue', 'red']))
plt.title("KNN Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid()

plt.show()
```

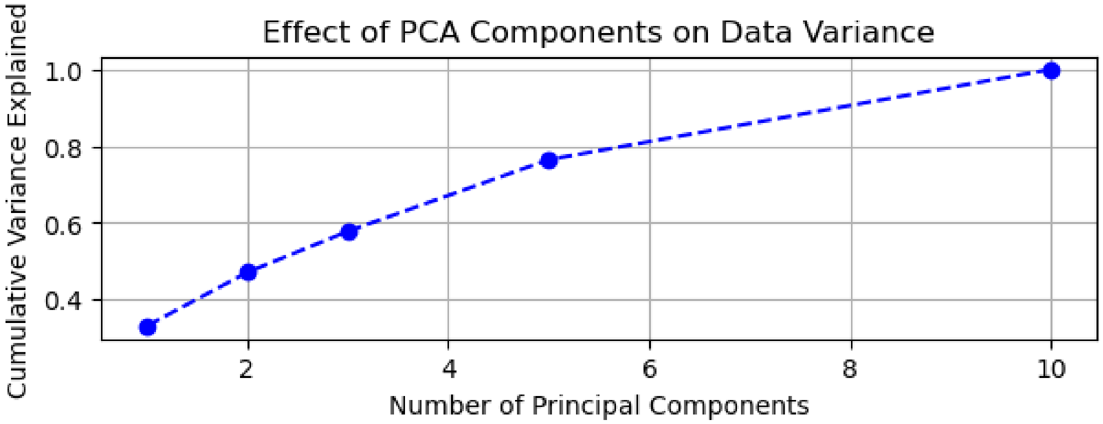
```
Model Accuracy: 0.91
```



1.28 48) Train a PCA model and analyze the effect of different numbers of components on data variance.

[89]:

```
from sklearn.datasets import make_classification
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Create synthetic dataset
X, _ = make_classification(n_samples=300, n_features=10, random_state=42)

# Analyze effect of different numbers of components
components = [1, 2, 3, 5, 10]
explained_variances = []
for n in components:
    pca = PCA(n_components=n)
    pca.fit(X)
    explained_variances.append(sum(pca.explained_variance_ratio_))

# Plot cumulative variance explained
plt.figure(figsize=(7, 2))
plt.plot(components, explained_variances, marker='o', linestyle='--', color='b')
plt.title("Effect of PCA Components on Data Variance")
plt.xlabel("Number of Principal Components")
plt.ylabel("Cumulative Variance Explained")
plt.grid()
plt.show()
```


Effect of PCA Components on Data Variance
Cumulative Variance Explained
Number of Principal Components