What is K Nearest Neighbor?

K Nearest Neighbor (KNN) is a simple, non-parametric, and lazy learning algorithm used for classification and regression tasks in machine learning. It operates on the principle that similar instances exist in close proximity in feature space.

What does the K stand for in K Nearest Neighbors?

The 'K' in K Nearest Neighbors stands for the number of nearest neighbors to consider when making a prediction. For example, if K=3, the algorithm will look at the three closest data points to make its prediction.

What is K Nearest Neighbor used for?

K Nearest Neighbor is primarily used for:

Classification: Assigning a class label to a given data point based on the majority class among its K nearest neighbors.

Regression: Predicting a continuous value based on the average of the values of its K nearest neighbors.

How does K Nearest Neighbor work?

KNN works by:

Storing the training data: It retains all the training examples.

Calculating distances: For a new input instance, it calculates the distance between this instance and all the training data points using a distance metric (e.g., Euclidean distance).

Finding nearest neighbors: It identifies the K data points in the training set that are closest to the input instance.

Making a prediction:

For classification: It assigns the class that is most common among the K nearest neighbors.

For regression: It computes the average of the values of the K nearest neighbors.

What is K Nearest Neighbor Classification?

K Nearest Neighbor Classification is a type of KNN used for classifying data points into predefined classes based on the majority class of their K nearest neighbors.

What is Euclidean Distance and Manhattan Distance?

Euclidean Distance: This is the straight-line distance between two points in Euclidean space. For two points $(p1,p2,\ldots,pn)(p_1,p_2,\ldots,p_n)$ and $(q1,q2,\ldots,qn)(q_1,q_2,\ldots,q_n)$, the Euclidean distance is given by:

$d(p,q)=\sum i=1n(pi-qi)2d(p,q)=\sum_{i=1n}(p_i-q_i)2$

Manhattan Distance: Also known as the L1 distance or city block distance, it is the sum of the absolute differences of their coordinates. For two points $(p1,p2,\ldots,pn)(p_1,p_2,\ldots,p_n)$ and $(q1,q2,\ldots,qn)(q_1,q_2,\ldots,q_n)$, the Manhattan distance is given by:

$d(p,q)=\sum i=1n|pi-qi|d(p,q)=\sum_{i=1n}|p_i-q_i|$

What is Nearest Neighbor Rule?

The Nearest Neighbor Rule states that an instance is classified by the majority label among its nearest neighbors. In its simplest form (K=1), this rule states that a data point should be assigned the class of its single nearest neighbor.

What is K Nearest Neighbor Classification Diagram?

A K Nearest Neighbor Classification diagram visually represents the process of classifying a new data point based on its K nearest neighbors in the feature space. Here's a simple step-by-step visualization:

Plot the training data: Points are plotted on a graph based on their feature values.

Identify the new data point: A new data point is introduced to the graph.

Calculate distances: Compute distances from the new point to all training points.

Select K nearest neighbors: Highlight the K closest training points.

Assign class label: Determine the majority class among these K points and assign it to the new data point.

How to Implement K Nearest Neighbor Classification in Python using Scikit-learn?

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score


# Sample data (features and labels)

X = np.array([[1, 2], [2, 3], [3, 4], [5, 6], [6, 7], [7, 8], [2, 6], [6, 2]])

y = np.array([0, 0, 0, 1, 1, 1, 0, 1])
```

```
# Splitting the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Creating and training the KNN classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Making predictions
y_pred = knn.predict(X_test)

# Evaluating the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100}%")

# Visualization
# Plot the training data
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis', marker='o', label='Train Data')

# Plot the test data
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='viridis', marker='s', edgecolor='k', label='Test Data')

# Create mesh grid for decision boundary
h = .02  # 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))

# Plot decision boundary
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.2, cmap='viridis')

# Plot test points with predicted class
for i, txt in enumerate(y_pred):
    plt.annotate(txt, (X_test[i, 0], X_test[i, 1]), textcoords="offset points", xytext=(0, 10), ha='center')

plt.legend()
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('K Nearest Neighbors Classification')
plt.show()
```

Explanation of the Code and Results:

Data Preparation:

The dataset `X` and labels `y` are created with two features each.

The data is split into training and test sets using `train_test_split`.

Model Training:

A `KNeighborsClassifier` with `n_neighbors=3` is created and trained on the training data.

Making Predictions:

Predictions are made on the test set.

The accuracy of the model is computed and printed.

Visualization:

Training and Test Data: The training data is plotted as circles (`o`), and the test data is plotted as squares (`s`).

Decision Boundary: The decision boundary is visualized by creating a mesh grid and plotting the predicted class for each point in the grid.

Annotations: Test data points are annotated with their predicted class labels.

The plot shows the training and test data points, the decision boundary, and the test data points annotated with their predicted class labels. The decision boundary helps visualize how

the classifier is partitioning the feature space into different class regions. The accuracy score indicates the performance of the classifier on the test data.

Example 2:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data[:, :2]  # We'll use only the first two features for visualization purposes
y = iris.target

# Split the dataset 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)

# Create and train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)
```

```
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Visualization
# Plot the training data
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis', marker='o', label='Train Data')

# Plot the test data
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='viridis', marker='s', edgecolor='k', label='Test Data')

# Create mesh grid for decision boundary
h = .02  # 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))

# Plot decision boundary
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.2, cmap='viridis')

# Plot test points with predicted class
for i, txt in enumerate(y_pred):
```

```
    plt.annotate(txt, (X_test[i, 0], X_test[i, 1]), textcoords="offset points", xytext=(0, 10), ha='center')

plt.legend()
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('K Nearest Neighbors Classification (Iris Dataset)')
plt.show()
```

Explanation:

Load the Iris Dataset:

The Iris dataset is loaded using `load_iris()` from Scikit-learn.

Only the first two features are used to make visualization easier.

Data Splitting:

The dataset is split into training and test sets using `train_test_split`.

Training the Model:

A KNN classifier is created with `n_neighbors=5` and trained on the training data.

Making Predictions:

Predictions are made on the test set.

The accuracy of the model is calculated and printed.

Visualization:

Training data is plotted with circles (`o`) and test data with squares (`s`).

A mesh grid is created to plot the decision boundary.

The decision boundary is plotted to show how the classifier partitions the feature space.

Test data points are annotated with their predicted class labels.

Results:

The plot shows the training and test data points, the decision boundary, and the test data points annotated with their predicted class labels.

The decision boundary helps visualize how the classifier is partitioning the feature space into different class regions.

The accuracy score indicates the performance of the classifier on the test data.

Explanation of Each Line of Code

`numpy (np)`: A fundamental package for array computing with Python.

`matplotlib.pyplot (plt)`: A plotting library used for creating static, animated, and interactive visualizations in Python.

`sklearn.datasets.load_iris`: A function to load the Iris dataset, which is a classic dataset used in machine learning.

`sklearn.model_selection.train_test_split`: A function to split the dataset into training and testing sets.

`sklearn.neighbors.KNeighborsClassifier`: A class for the K-Nearest Neighbors classification algorithm.

`sklearn.metrics.accuracy_score`: A function to compute the accuracy of the classifier.

`iris = load_iris()`: Loads the Iris dataset into a dictionary-like object with keys such as 'data', 'target', and 'feature_names'.

`X = iris.data[:, :2]`: Extracts the first two features (columns) of the dataset for easier 2D visualization.

`y = iris.target`: Extracts the target labels (species of iris).

`train_test_split(X, y, test_size=0.3, random_state=42)`: Splits the dataset into training and test sets.

`X_train, y_train`: The training data and corresponding labels.

`X_test, y_test`: The test data and corresponding labels.

`test_size=0.3`: 30% of the data is used for testing.

`random_state=42`: Ensures reproducibility by seeding the random number generator.

`knn = KNeighborsClassifier(n_neighbors=5)`: Creates a KNN classifier with K=5 (i.e., it considers 5 nearest neighbors).

`knn.fit(X_train, y_train)`: Trains the KNN classifier using the training data.

`y_pred = knn.predict(X_test)`: Uses the trained KNN classifier to predict the labels for the test set.

`accuracy = accuracy_score(y_test, y_pred)`: Calculates the accuracy of the predictions by comparing the predicted labels with the true labels.

`print(f"Accuracy: {accuracy * 100:.2f}%")`: Prints the accuracy as a percentage, formatted to two decimal places.

`plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis', marker='o', label='Train Data')`:

Plots the training data points.

`X_train[:, 0], X_train[:, 1]`: The first and second features of the training data.

`c=y_train`: Colors the points according to their labels.

`cmap='viridis'`: Uses the 'viridis' colormap.

`marker='o'`: Uses circles for the training data points.

`label='Train Data'`: Adds a label for the legend.

`plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap='viridis', marker='s', edgecolor='k', label='Test Data')`:

Plots the test data points.

`marker='s'`: Uses squares for the test data points.

`edgecolor='k'`: Adds a black edge color to the test data points.

`label='Test Data'`: Adds a label for the legend.

`h = .02`: Defines the step size of the mesh grid.

`x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1`: Finds the minimum and maximum values of the first feature, adding a margin of 1.

`y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1`: Finds the minimum and maximum values of the second feature, adding a margin of 1.

`xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))`: Creates a mesh grid that covers the feature space.

`Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])`:

Flattens the mesh grid arrays and combines them into a single array of coordinate pairs.

Predicts the class for each coordinate pair using the trained KNN classifier.

`Z = Z.reshape(xx.shape)`: Reshapes the predictions to match the shape of the mesh grid.

plt.contourf(xx, yy, Z, alpha=0.2, cmap='viridis'): Plots the decision boundary as a filled contour plot.

for i, txt in enumerate(y_pred): Loops through the predicted labels of the test set.

plt.annotate(txt, (X_test[i, 0], X_test[i, 1]), textcoords="offset points", xytext=(0, 10), ha='center'):

Annotates each test data point with its predicted label.

textcoords="offset points": Specifies that the text position is offset from the point.

xytext=(0, 10): Offsets the text by 10 points vertically.

ha='center': Centers the text horizontally.

plt.legend(): Adds a legend to the plot.

plt.xlabel(iris.feature_names[0]): Labels the x-axis with the name of the first feature.

plt.ylabel(iris.feature_names[1]): Labels the y-axis with the name of the second feature.

plt.title('K Nearest Neighbors Classification (Iris Dataset)'): Sets the title of the plot.

plt.show(): Displays the plot.

By following these steps, we train a KNN classifier, evaluate its performance, and visualize the results, including the decision boundaries and annotations of predicted classes for the test data