LAB 11 KNN

The K-Nearest Neighbors (KNN) algorithm is a simple yet effective classification algorithm. It works based on the principle that data points with similar features tend to belong to the same class. KNN is a non-parametric algorithm, meaning it doesn't make any assumptions about the underlying data distribution.

- **Explanation of how the KNN algorithm works with an example:**

Step 1: Set the Value of K

Start by selecting a value for K, which represents the number of neighbors to consider when making predictions. K is typically an odd number to avoid ties in class assignments.

Step 2: Load and Preprocess the Data

Load the dataset and preprocess it as needed. This may involve scaling features, handling missing values, or encoding categorical variables.

Step 3: Calculate Distances

Calculate the distance between the input point and all other points in the dataset. The most commonly used distance metric is Euclidean distance, but other metrics like Manhattan distance or cosine similarity can also be used.

Step 4: Find K Nearest Neighbors

Select the K data points with the shortest distances to the input point. These data points are considered the "nearest neighbors."

Step 5: Assign a Class Label

Determine the class label of the input point based on the majority vote of the K nearest neighbors. For example, if the majority of the K nearest neighbors belong to class A, the input point is assigned to class A.

Step 6: Make Predictions

Repeat steps 3 to 5 for each new input point to make predictions on unseen data.

Step 7: Evaluate Accuracy

Compare the predicted class labels with the true class labels of a validation or test set to evaluate the accuracy of the KNN model.

- **Example to illustrate the KNN algorithm:**

Suppose we have a dataset of flower samples with features like sepal length, sepal width, petal length, and petal width. The samples are labeled with their corresponding flower species: setosa, versicolor, or virginica.

- We load the dataset and split it into a training set and a test set.
- For a new flower sample in the test set, we calculate the distances to all the samples in the training set.
- Assuming K=5, we find the 5 nearest neighbors to the new sample based on the calculated distances.
- Among these 5 neighbors, if 3 of them belong to the setosa species and 2 of them belong to the versicolor species, we assign the new sample to the setosa species.
- We repeat this process for all the samples in the test set, making predictions for each of them.
- Finally, we evaluate the accuracy of the KNN model by comparing the predicted species with the true species labels.

It's worth noting that the choice of K can have an impact on the KNN model's performance. A small K value may lead to overfitting, while a large K value may lead to underfitting. Therefore, it's important to experiment with different values of K to find the optimal balance for a given dataset.

- **Step-by-step guide on how to use the KNN algorithm for classification with an example:**

Step 1: Import Libraries

First, import the necessary libraries in Python, including scikit-learn, which provides the KNN algorithm implementation.

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

Step 2: Prepare the Data

Load your dataset and split it into input features (X) and corresponding class labels (y).

```
# Assuming your data is stored in a pandas DataFrame called 'data'
X = data.drop('target', axis=1)  # Features
y = data['target']  # Class labels
```

Step 3: Split the Data

Split the dataset into training and testing sets. This allows us to train the KNN algorithm on a portion of the data and evaluate its performance on the remaining unseen data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Step 4: Create and Train the KNN Model

Create an instance of the KNeighborsClassifier class and specify the desired value for K (the number of neighbors to consider). Then, train the model using the training data.

```
k = 5  # Number of neighbors
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
```

Step 5: Make Predictions

Use the trained KNN model to make predictions on the test set.

```
y_pred = knn.predict(X_test)
```

Step 6: Evaluate Accuracy

Compare the predicted class labels (y_pred) with the true class labels (y_test) to evaluate the accuracy of the KNN model.

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Note: The example assumes that you have already preprocessed the dataset and have numerical features. If your data contains categorical or text features, you would need to apply appropriate preprocessing techniques before using the KNN algorithm.

Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

TASK: The task of this program is to classify the IRIS data set examples by using the k-Nearest Neighbour algorithm. The new instance has to be classified based on its k nearest neighbors.

Dataset: iris.csv

ALGORITHM

Let m be the number of training data samples. Let p be an unknown point.

1. Store the training samples in an array of data points arr[]. This means each element of this array represents a tuple (x, y).
2. for i=0 to m:
   Calculate Euclidean distance d(arr[i], p).
3. Make set S of K smallest distances obtained. Each of these distances correspond to an already classified data point.
4. Return the majority label among S.

Implement the KNN algorithm for given dataset. For k = 7 and 11