595Group Project

Hongxiang Fei

December 2024

Introduction

This document summarizes the K-Nearest Neighbors (KNN) implementation on the MNIST dataset. The provided code loads the dataset, preprocesses the data, reduces dimensionality using PCA, and applies KNN to classify handwritten digits. The following sections break down the code and explain their workings.

What is KNN

K-Nearest Neighbors (KNN) is a simple and intuitive algorithm for classification and regression tasks. The core idea is to calculate the distance between the test sample and all training samples, identify the k nearest neighbors, and make predictions based on their labels. A commonly used distance metric is the Euclidean distance, defined as:

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$$

where $\mathbf{x}_1$ and $\mathbf{x}_2$ are two samples, and n is the number of features. In classification tasks, KNN assigns the majority label among the k neighbors, while in regression tasks, it computes the average of the neighbors' values.

Code and Explanation

Data Loading and Preprocessing

The following code loads the MNIST dataset and preprocesses it by flattening the images into vectors, normalizing the data, and applying PCA for dimensionality reduction.

Listing 1: Loading and Preprocessing MNIST Dataset

```
# Load MNIST dataset
```

```
transform = transforms.Compose([
    transforms.ToTensor()
])
train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, d
test_dataset = datasets.MNIST(root='./data', train=False, transform=transform, d

# Extract features and labels
X_train = train_dataset.data.numpy().reshape(len(train_dataset), -1)
y_train = train_dataset.targets.numpy()
X_test = test_dataset.data.numpy().reshape(len(test_dataset), -1)
y_test = test_dataset.targets.numpy()

# Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Perform PCA for dimensionality reduction
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

Explanation: In this part of the code, I first downloaded the MNIST dataset using the datasets.MNIST method and converted the image data into tensors using the ToTensor transformation. I then extracted the features and labels of the dataset and flattened each image into a one-dimensional vector for subsequent processing. For data preprocessing, I used StandardScaler to normalize the features so that each feature has zero mean and unit variance, thus reducing the impact of scale differences between features on the algorithm. Finally, the data was downscaled by PCA (Principal Component Analysis), which preserves 95% of the total variance and provides a more streamlined representation of the features for subsequent classification tasks.

KNN Implementation and Training

The next block initializes the KNN classifier, trains it on the preprocessed dataset, and evaluates it on the test set.

Listing 2: Training and Evaluating KNN

```
# Initialize KNN classifier
k = 4
knn = KNeighborsClassifier(n_neighbors=k, n_jobs=-1)
knn.fit(X_train_pca, y_train)
```

```
# Predict on test set
y_pred = knn.predict(X_test_pca)

# Calculate overall accuracy
accuracy = accuracy_score(y_test, y_pred)

# Calculate the number of correct and incorrect predictions
correct = sum(y_pred == y_test)
wrong = len(y_test) - correct

# Print results
print(f"KNN Accuracy on MNIST Test Set (k={k}): {accuracy:.4f}")
print(f"Total Correct: {correct}")
print(f"Total Wrong: {wrong}")
```

Explanation: In this part of the code scikit-learn provides KNeighborsClassifier which is an efficient implementation of the KNN algorithm, I have tried to list Euclid's formulas directly in the code to try to do the calculations and then arrive at the result, but the code often needs to run for a long time. KNeighborsClassifier can automatically select a KNeighborsClassifier can automatically choose a tree-based algorithm (such as KD-Tree or Ball-Tree) to speed up the nearest neighbor search.

Results and Analysis

The results of the KNN classifier on the MNIST test set are summarized as follows:

- **Accuracy**: The overall accuracy achieved on the test set is 94.88%.
- **Correct Predictions**: A total of 9488 test samples were correctly classified.
- **Incorrect Predictions**: 512 test samples were misclassified.

Conclusion

KNN (K-Nearest Neighbors) is a very simple and intuitive classification algorithm whose core idea is to classify instances based on their similarity. Compared to other machine learning methods, such as Support Vector Machines (SVM) or Neural Networks, KNN does not require a complex training process or optimization of model parameters. It directly utilizes the training data as the basis for classification, and determines the category to which it belongs by calculating the distance between the test samples and the training samples. Therefore, KNN is especially suitable for small-scale datasets or scenarios that do not require online training.