CSci 4270 and 6270
Computational Vision,
Spring Semester, 2024
Homework 4 — Part 1
Due: Tuesday, March 19 at 11:59:59 pm EST

Overview

This is the first of two parts to HW 4. The second part will be distributed on the Monday after spring break. This part is worth 60 points toward your homework grade.

Your goal is to explore the use of k-nearest neighbors as a classifier on the Fashion MNIST dataset. The Fashion MNIST dataset, often used as an alternative to the famous MNIST digits dataset, consists of 60,000 28x28 dimensional grayscale images, each showing one ten different clothing types. (A quick search on the internet will show you many examples We are going to develop and test a k-nearest neighbor classifier that takes each test image and assigns it to one of the clothing type labels. Of course, k-NN is rarely used directly on images as a classifier, but it is used after images have been processed (transformed) into a low dimensional "descriptor" or "latent" space. Our goal here is two-fold: to study the basic k-NN algorithm and to begin to practice use of classifiers.

You will submit a single Jupyter notebook containing your complete solution and analysis. A starting notebook is provided for you, and it includes code to download the data.

Getting Started

1. Download the Fashion MNIST images. These are split into "train" and "test" sets, each of which has images and corresponding labels.

2. Load the train and test sets.

3. Using pyplot, display a random selection of images in your notebook. Include text indicating the label for each image.

4. Output statistics indicating how many images belong to each class from the training set. Do the same for the test set.

5. Unravel the images to form vectors and then normalize the images so that each has a mean intensity value of 0 and a standard deviation of 1.0.

6. Build a `sklearn.neighbors.KNeighborsClassifier`

   `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifi`
   `html`

   and apply this using a small sampling (for example 25) of the test images. Adjust the parameters of the initializer of the classifier so that you get at least some correct matches. For each sample, output the nearest-neighbor class and, if it is wrong, output the correct class. Note that this is just a brief test to make sure the basic techniques are working.

Performance Metrics

Implement a set of metrics to measure performance:

1. The overall accuracy is the percentage of classification decisions the classifier gets correct.
2. The per class accuracy is the percentage of correct classification decisions for each class.
3. The confusion matrix for n classes is an $n \times n$ matrix where entry (i, j) is the number of times where i is the correct class and j is the class predicted by the classifier. In the ideal scenario, the confusion matrix would be diagonal.

Implement each as a separate function in your notebook that takes a list of pairs (i, j) and returns the result. A cell is provided in the notebook that tests the functions.

Tune the k-NN Classifier

"Train" your classifier:

1. Randomly select 1,000 training examples as your "validation" set. Keep the other 59,000 as your training set.
2. Select a series of parameters to tune. This must include k (the number of nearest neighbors), and may include the weights parameter (see the scikit description). Explore others as your time and interest allow.
3. For each combination of tuning parameters, build a k-NN classier from the 59,000 training images and apply it to the 1,000 validation images. Output the overall accuracy for each combination.
4. Retain the set of parameters that give the best overall performance on the validation data. Output a message indicating which is the best parameter set.

Note that this is not really "training" the classifier, but rather "tuning" the "hyper-parameters" that form the classifier.

Final Test

Apply your tuned classifier with all 60,000 images to the 10,000 images in the test set. Output the overall accuracy, the per class accuracy, and the confusion matrix (nicely formatted).

Discuss Your Results

Add a brief discussion of your training and test results. What classes are easiest to recognize? What are hardest? What are the easiest to confused? What surprised you about the results? What parameter settings are most important?

Additional 6270 Component

Students in the graduate version of the course and anyone in 4270 who wants to earn up to 10 points extra credit please repeat the tuning and final test while running principle component analysis (using `sklearn.decomposition.PCA`).

`https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`

PCA is a dimension reduction technique where the training images are each unraveled, formed into an outer product matrix, and then the eigenvectors corresponding to m largest eigenvalues are found. These are the m principle components and they describe the primary variations in the data. Prior to the k-NN computation, the images — both training and test — are projected onto these principle components to form the data vectors used in k-NN. The key parameter here is the number of components, m, which in our case can't be more than 28x28 = 784, but will likely be closer to 100.

Repeat the tuning and final test by including m as one of the parameters to vary. Be careful to keep the sampling of parameters manageable.