Analyse et Traitement de l'Information

TP3: PCA , k-NN classification

The MNIST dataset of handwritten digits [1] consist of 28×28 grayscale images. To download dataset use this code:

```
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split

images, labels = fetch_openml('mnist_784', return_X_y=True)
train_images, test_images, train_labels, test_labels = \
            train_test_split(images, labels, random_state=42)
```

To filter images of specific class you can use following method:

```
import numpy as np

def select_with_label(images, labels, desired_labels):
    mask = np.isin(labels, desired_labels)
    return images[mask], labels[mask]

images_of_two, labels_of_two = \
        select_with_label(train_images, train_labels, desired_labels=['2'])
images_of_odd, labels_of_odd = \
        select_with_label(train_images, train_labels, \
        desired_labels=['1', '3', '5', '7', '9'])
```

You can copy-paste from this link.

1 Principal Component Analysis (PCA)

Randomly sample 5000 images of the digit 2 from the MNIST dataset. Denote this subset as $\mathcal{X} = \{x_i\}_{i=1}^n$, where each x_i is a 784-dimensional vector corresponding to a 28×28 image of the digit 2, and $n = 5000$. Perform PCA on $\mathcal{X}$. Denote the

[1] http://yann.lecun.com/exdb/mnist/

first m ($1 \leq m \leq 784$) principal components (with the largest variance) of $\mathcal{X}$ as $\{PC_1, \ldots, PC_m\}$. Note, each PC_j is also a 784 dimensional vector.

1. Any image $x_i \in \mathcal{X}$ can be reconstructed using $\{PC_1, \ldots, PC_m\}$ by

$$r_m(x_i) = \sum_{j=1}^{m} \langle x_i - \bar{x}, PC_j \rangle \cdot PC_j + \bar{x}$$

   where $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$ is the average image and $\langle \cdot, \cdot \rangle$ denotes the inner product. The corresponding reconstruction error of $\mathcal{X}$ can be defined by

$$err(\mathcal{X}, m) = \frac{1}{n} \sum_{i=1}^{n} \|x_i - r_m(x_i)\|^2$$

   - Plot $err(\mathcal{X}, m)$ as a function of m where $m = 1, \ldots, 784$.
   - Find m that corresponds to accuracy 50%, 95% and 100%.

2. Sample 5 random images of 2 from MNIST (outside from $\mathcal{X}$). For each image $\mathcal{I}$ in this 5 samples and for each $m = 1, 2, \ldots, 10$ do the following taks:

   - Compute the reconstruction of $\mathcal{I}$ using the first m principal components of $\mathcal{X}$
   - Calculate the reconstruction loss for each image and each m.

   In total, you should have 50 reconstructed images and 5 original images. Plot these images in a nice way (in a table with columns: *original image, reconstruction with* $m = 1, 2, \ldots, 10$).

2 *k*-Nearest-Neighbour (*k*-NN) Classification

Randomly sample 5000 images of "0", "1", "2", "3" and "4" as the training set. For the testing set, use all available testing images of the same classes.

The main idea of *1*-NN classifier is:

> To classify a sample x in the testing set, select the nearest sample from $\mathcal{X}$ in the *training set*, then predict the class label of x to be the same as this nearest sample.

To generalize this idea to k-NN classifier you need to select k nearest samples instead of only one.

1. Compute the baseline *1*-NN classification accuracy, which is defined as the percentage of samples correctly classified in the testing set;
2. (PCA + *1*-NN) Repeat the following task for each

$$m = 10, 20, 30, 40, 50, 100, 150, 200, \ldots, 750$$

   - Perform PCA on the training set and reconstruct it using the first m principal components $\{PC_1, \ldots, PC_m\}$. To classify a testing sample x, compute its reconstruction $r_m(x)$ using $\{PC_1, \ldots, PC_m\}$, then output the class label of the nearest reconstructed sample in the training set.

   Plot the classification accuracy with respect to m. Explain your observations on the effect of m on the classification performance.

3 *k*-Nearest-Neighbour (*k*-NN) Classification Implementation

In the previous task, you used a classifier from *sklearn* library. But you can write your own implementation! Please make a copy of this notebook and implement missed methods.

1. Compute the baseline *1*-NN classification accuracy (use k-NN from *sklearn* library), which is defined as the percentage of samples correctly classified in the testing set;
2. Complete the following method: `def predict_proba(self, X)`. You are allowed to use only *numpy* library and your code should work for any number of neighbours.
3. Compute the accuracy of your algorithm. *Tip: it should not be much lower!*
4. Compare the speed of your algorithm with *sklearn* library.
5. To find nearest neighbours faster you could use *KDTree* algorithm. Once again, complete the missing lines (you are allowed to use *KDTree* implementation from *sklearn*) and compare accuracy & speed.
6. *(optional)* If you have time and desire, you can try to implement *KDTree* algorithm. You may find following links useful:
   - How to Search Data with KDTree
   - Understanding K-Dimensional Trees

Submission

Please archive your report and codes in "Prénom Nom.zip" (replace "Prénom" and "Nom" with your real name), and upload to "`Upload Tps > Upload TP3`" on `https://moodle.unige.ch` before Friday, November 5 2021, 23:59 PM. Note, the assessment is mainly based on your report, which should include your answers to all questions and the experimental results. *Importance is given on the mathematical explanations of your works and your codes should be commented*