UNIVERSITÉ DE GENÈVE

ANALYSE ET TRAITEMENT DE L'INFORMATION
14X026

TP 2.2: PCA, k-NN classification

Author: Joao Costa

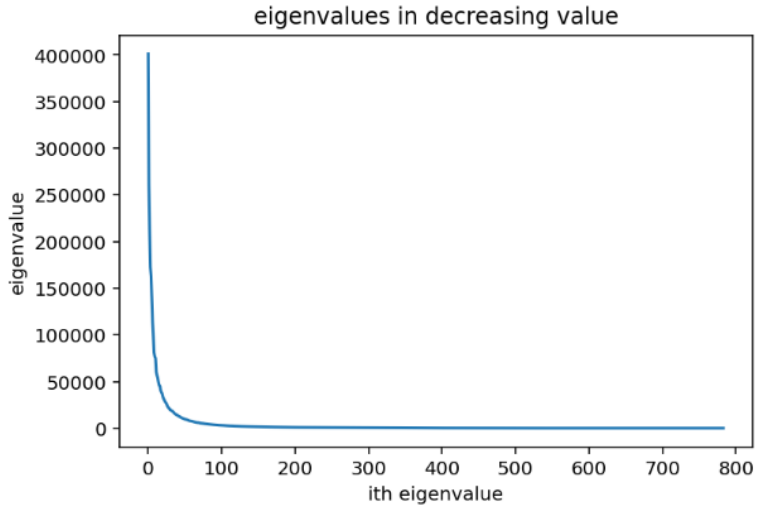
E-mail: Joao.Costa@etu.unige.ch

October 2022

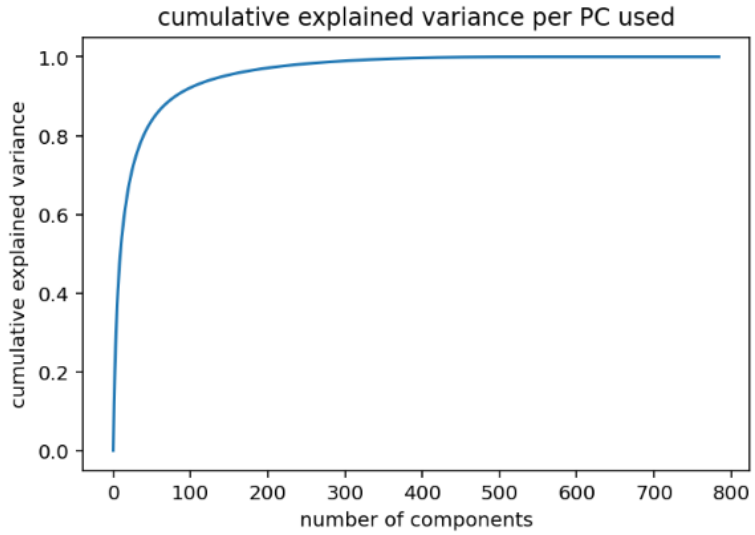
UNIVERSITÉ
DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

Exercise 1. PCA

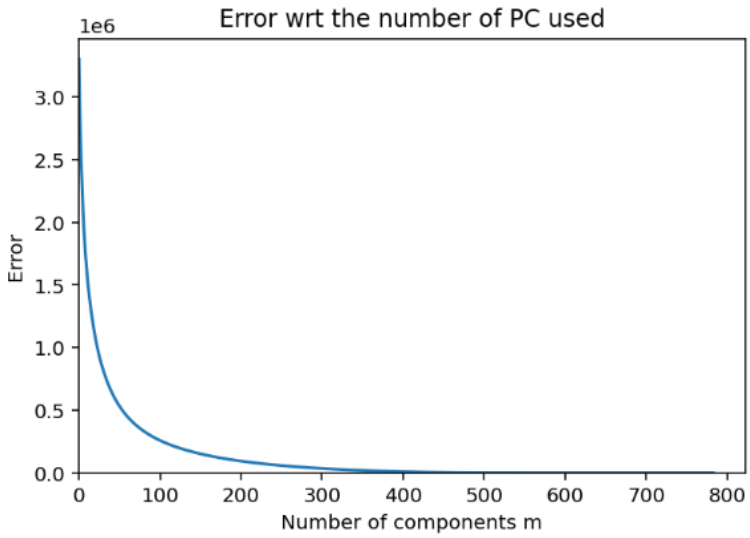
1. Plot sorted eigenvalues



2. Plot cumulative explained variance ratio as a function of the number of components



3. Ploterr(X,m) as a function of m where m = 1, . . . , 784.



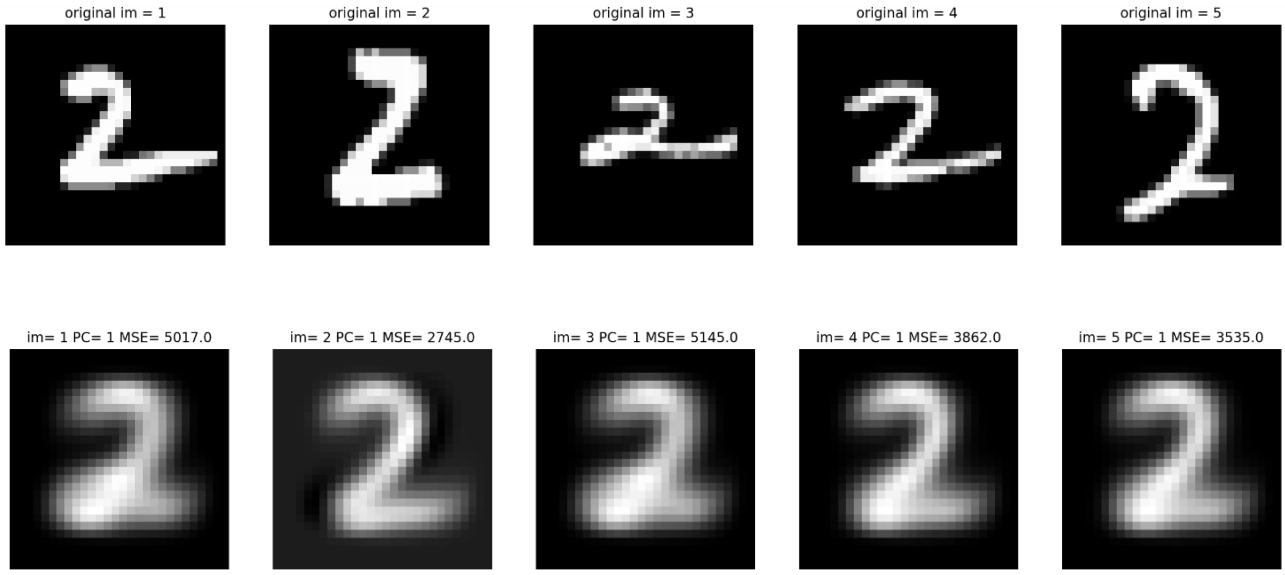
4. Find m that corresponds to accuracy 50%, 95% and 99% (same as error 0.5, 0.05, 0.01).

```
9 principal components are required to have 50.0 % accuracy
141 principal components are required to have 95.0 % accuracy
302 principal components are required to have 99.0 % accuracy
```

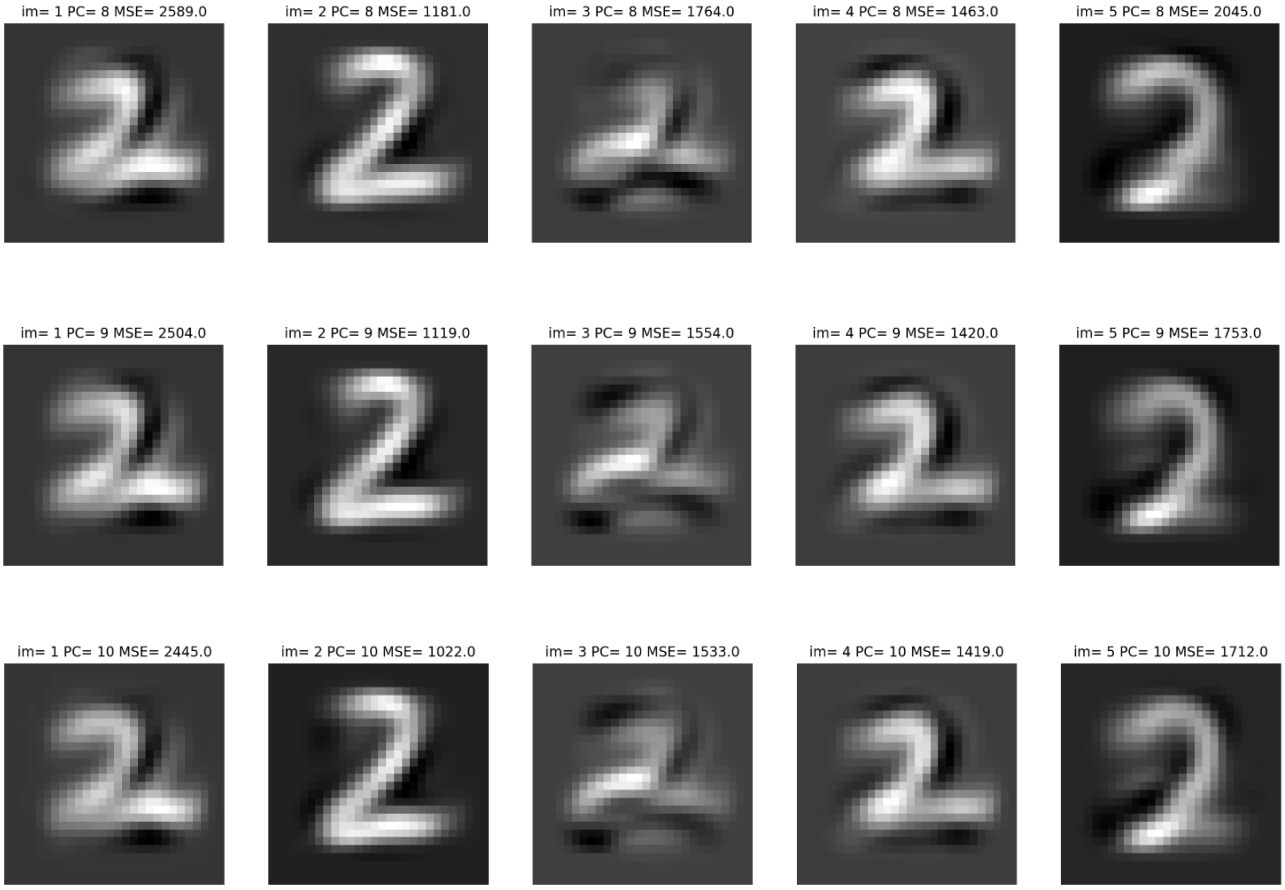
5. Compare all three previous plots. Do you notice anything in common? Could you explain anything?

6. Plot the mean vector of X and first 10 principal components.

7. Sample 5 random images of digit 2 from MNIST (outside from X). For each image I in this 5 samples and for each m = 1, 2, . . . , 10, reconstruct image from m PC, compute loss




im= 1 PC= 2 MSE= 3433.0
im= 2 PC= 2 MSE= 2351.0
im= 3 PC= 2 MSE= 2797.0
im= 4 PC= 2 MSE= 2053.0
im= 5 PC= 2 MSE= 3430.0
im= 1 PC= 3 MSE= 3131.0
im= 2 PC= 3 MSE= 2233.0
im= 3 PC= 3 MSE= 2345.0
im= 4 PC= 3 MSE= 2053.0
im= 5 PC= 3 MSE= 3196.0
im= 1 PC= 4 MSE= 3117.0
im= 2 PC= 4 MSE= 1754.0
im= 3 PC= 4 MSE= 2257.0
im= 4 PC= 4 MSE= 1907.0
im= 5 PC= 4 MSE= 3015.0
im= 1 PC= 5 MSE= 3041.0
im= 2 PC= 5 MSE= 1468.0
im= 3 PC= 5 MSE= 2200.0
im= 4 PC= 5 MSE= 1827.0
im= 5 PC= 5 MSE= 2074.0
im= 1 PC= 6 MSE= 2736.0
im= 2 PC= 6 MSE= 1355.0
im= 3 PC= 6 MSE= 2169.0
im= 4 PC= 6 MSE= 1775.0
im= 5 PC= 6 MSE= 2071.0
im= 1 PC= 7 MSE= 2729.0
im= 2 PC= 7 MSE= 1339.0
im= 3 PC= 7 MSE= 1765.0
im= 4 PC= 7 MSE= 1716.0
im= 5 PC= 7 MSE= 2059.0



We can see that by increasing PC we get less and less MSE. (image 2 actually got a worse MSE with PC=10 rather than with PC=9).

Exercise 2. k-Nearest-Neighbour (k-NN) Classification

From all the data points, first I split it into train and test datasets, 5000 random test data points, and the rest is the training data.

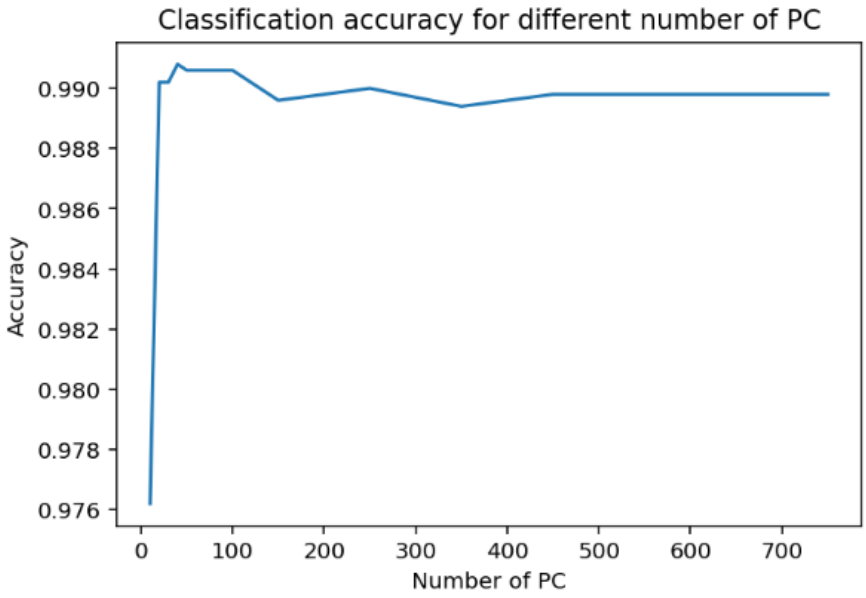
1. Compute the baseline 1 -NN classification accuracy, which is defined as the percentage of samples correctly classified in the testing set;

```
accuracy of 1-NN classifier is :  0.9898
```

   It is a very high baseline score!!

2. (PCA + 1 -NN) Repeat the following task for each -> m = 10, 20, 30, 40, 50, 100, 150, 200, 250, . . . , 750

   The goal of this exercice is to use PCA to filter out the least valuable information, to improve our accuracy.



   The result was very good, however we must note that the best accuracy isn't from the most PC, there is an optimal value for PC, and I found it to be PC = 40, with accurcy of 0.9908.
   Which means that by using PCA we improved the accuracy.

Exercise 3. k-NN Classification Implementation

1. Compute the baseline 5 -NN classification accuracy (use k-NN from sklearn library), which is defined as the percentage of samples correctly classified in the testing set;

```
startSklearn = time.time()

kNNClassifier = KNeighborsClassifier(5).fit(X_train,y_train)
kNNClassifier.predict(X_test)
baseline = kNNClassifier.score(X_test, y_test)

endSklearn = time.time()
print('The baseline 5-NN classification accuracy with sklearn is :',baseline)
```

```
The baseline 5-NN classification accuracy with sklearn is : 0.8444444444444444
```

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

   The code is explained but the steps are quite simple:
   -> Compute distance to all training samples
   -> find k closest data points, and create a list of their class
   -> count how many occurrences each class has (in the list comuted previously)
   -> compute probability for each class by dividing by number of neighbours

   do this for every test point!

```
def predict_proba(self, X):
    _, n_classes = np.unique(self.y, return_counts=True)
    res = np.zeros((X.shape[0], len(n_classes)))
    i = 0
    for x in X:
        dist = np.linalg.norm(x - self.X, axis=1) # dist to all training samples
        #now we need to find the n closer neighboors
        neigh = self.y[np.argpartition(dist, self.n_neighbors)[0:self.n_neighbors]]
        # neigh is list of class of k closesnt nieghbours -> example : [2,3,3,3,3] -> closest is class 2, rest is class 3

        res[i] = np.bincount(neigh, minlength=np.bincount(y).size)/self.n_neighbors # count each classes occurrence
        # res[i] is a list of each neighbours class occurrence -> example : [0,0,1,4,0] -> 1 of class 2, 4 of class 3
        # to compute the probability we must divide by #neighbours
        # example : [0,0,1/5,4/5,0] -> 1/5 chance of being of class 2, 4/5 of being of class 3
        i = i + 1
    return res
```

3. Compute the accuracy of your algorithm. Tip: it should not be much lower!
   The accuracy of both versions was the same

```
SKLEARN LIB :

The baseline 5-NN classification accuracy with sklearn is : 0.8444444444444444

MY KNN :

My 5-NN classification accuracy is : 0.8444444444444444
```

4. Compare the speed of your algorithm with sklearn library.
   My algorithm was faster!

```
SKLEARN LIB :

The baseline 5-NN classification Computing time 0.007979154586791992

MY KNN :

My 5-NN classification Computing time 0.00500178337097168
```

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.

```
class KNearestWithKDTree(BaseEstimator):
    def __init__(self, n_neighbors=5, leaf_size=30):
        self.leaf_size = leaf_size
        self.n_neighbors = n_neighbors

    def fit(self, X, y):
        self.tree = KDTree(X, leaf_size=self.leaf_size)
        self.y = y

    def predict_proba(self, X):
        _, n_classes = np.unique(self.y, return_counts=True)
        res = np.zeros((X.shape[0], len(n_classes)))

        closest = self.tree.query(X, k=self.n_neighbors, return_distance=False) # returns closesnt n neighbors to our test data
        i = 0
        # for every result we compute the probability by dividing by number of neighbors jst like before
        for close in closest:
            pred = np.bincount(self.y[close], minlength= np.bincount(y).size)/self.n_neighbors
            res[i] = np.array(pred)
            i = i + 1
        return res


    def predict(self, X):
        return np.argmax(self.predict_proba(X), axis=1)
```

```
My 5-NN classification accuracy with kdtree is 0.8444444444444444
My 5-NN classification Computing time with kdtree is 0.001995086669921875
```

The accuracy is the same, wich is expected, as this method only has a different way of computing the closest neighbours, but the results will be the same, however, we can see that it does this computation much faster!