AI & ML Homework - prof. Caputo, year 2018

Unknown

November 7, 2018

1 Task definition and steps

In this homework you will dirty your hands on **PCA** applied on images. You have to show what happens if different **principal components** (**PC**) are chosen as basis for images representation and classification. Then, you have to choose and apply a classifier in order to..classify the images ;-) under different PC re-projection and you should comment the obtained results. In particular, here is the brief list of this homework sub-tasks:

1. Download and load the provided subset of PACS dataset; setup your programming environment accordingly your needings.
2. Choose one image and shows what happens to the image when you re-project it with only first 60 PC, first 6 PC, first 2 PC, last 6 PC. Comment the results.
3. Using scatter-plot, visualize the dataset projected on first 2 PC. Repeat the exercise with only 3&4 PC, and with 10&11. What do you notice? Justify your answer from theoretical perspective behind PCA.
4. Classify the dataset (divided into training and test set) using a Naive Bayes Classifier in those cases: unmodified images, images projected into first 2PC, and on 3&4 PC. Show accuracy and compare results: what are your conclusions?
5. (Optional) Visualize decision boundaries of the classier in the first 2 PC case. Any consideration about those boundaries?

1.1 Steps more into details

General information. Problem solutions should be submitted in PDF format in report style, with the main code file attached into the email. All reports must be submitted at this email address:

- *paolo.russo@iit.it*

with subject [AI&ML2018] PCA report. It is advised to use Python as a programming language, but you can use any language of your choice (at your own risk). In case you use Python, you could use free Anaconda distribution that comes with all needed packages, or pip to install the needed packages. In particular, you might find useful scikit-learn general machine learning library and matplotlib plotting facilities. When in doubt, read the manual and take a look at the large set of examples:
http://scikit-learn.org/stable/documentation.html

http://scikit-learn.org/stable/auto_examples/
http://matplotlib.org/examples/

Data preparation. In this homework you will work with a subset of PACS dataset made of only 4 visual object categories. The dataset contains 1087 samples with 3 x 227 x 227 sample size.

1. Download and unpack dataset file "homework1.zip" from:

   https://drive.google.com/open?id=1isX_H74AGk3iy_YjBSfvKO1bU_rr6BqK (available on the course google drive folder)

2. Read raw pixels of all images for four classes of your choice. In python you can do so by:

   from PIL **import** Image *# or import Image*

   import numpy as np

   img data = np.asarray(Image.open($< pathtoimage >$))

   This will give you a 3-D array, where the last dimension species color of a pixel in RGB format.

3. Convert every image into a 154587-dimensional vector and prepare $Nx154587$ matrix X, where n is the number of images you have read. We refer to the rows of X as examples and to columns as features. Next, prepare an n-dimensional vector y holding ordinal labels of your image. Note that in python you can get vectorial representation of your 3-D array image array by:

   x = img_data.ravel()

1.2 Principal Component Visualization

1. Standardize **X** (make each feature zero-mean and unit-variance).

2. Use Principal Component Analysis (PCA) to extract rst two principal components from sample covariance matrix of X. Project X onto those two components. You can do it in python by running:

   from sklearn.decomposition **import** PCA

   X_t = PCA(2).fit_transform(X)

3. Choose one single image from the dataset and plot it after re-projection using only the first 60PC / 6 PC / 2 PC and the last 6 PC. Compare the results with respect to the original image. **UPDATE**: you have to calculate the principal components of the WHOLE dataset, and THEN you can transform the single image to the new chosen base.

4. Visualize X_t using scatter-plot with different colors standing for different classes:

   import matplotlib.pyplot as plt plt.scatter(X_t[:,0],X_t[:,1],c=y)

   Repeat this exercise when considering third and fourth principal component, and then tenth and eleventh. What do you notice? Justify your answer from theoretical perspective behind PCA.

5. How would you decide on the number of principal components needed to preserve data without much distortion?

1.3 Classification

1. Write down formulation of Naive Bayes classier:
$$\widehat{y} = \underset{y \in \{1,\ldots,k\}}{\arg\max}\, p(y \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_d) \ ,$$
where $\hat{y}$ is a predicted label, k is the number of classes, x_i, i=1..d are examples, $p(x|y)$ is a Gaussian, and distribution of labels is uniform.

2. Split examples in X and y into training and testing set. You can use train_test_split from sklearn.cross_validation package.

3. Train and test Nave Bayes classier with Gaussian class-conditional distribution. You can use *GaussianNB* from package from *sklearn.naive_bayes* for this purpose.

4. Repeat the splitting, training, and testing for the data projected onto rst two principal components, then third and fourth principal components. Compare results: what are your conclusions?

5. (Optional) Visualize decision boundaries of the classier. To learn how, look at this example: http://scikit-learn.org/stable/auto_examples/ensemble/plot_voting_decision_regions.html

2 Sparse suggestions

1. If your computer has not enough ram to load the whole dataset into memory you can stop reading files after 100 samples per class (400 total). Try to allocate the whole dataset with X = np.zeros((1087,3,227,227), dtype=float) before filling with actual data.

2. A simple, sometime tricky reminder: in ML some functions look for channels dimension after sample dimension (eg [1087,3,227,227]) others expect it after resolution (eg [1087,227,227,3]). You can always move axes with numpy.moveaxis function. In our case you for PCA and training you just need "flattened" images where you do not mind about channels and resolution, so you can just use np.zeros((1087,196608), but whenever you need the images for visualization you must "unflatten" your images back to the correct shape (use np.reshape !)

3. Before crying on how hard/impossible is to do a thing/fix a bug, *search on Google*. You can find 99.999999..... % of things you need on Google, you just need to learn how to implement their solution/code in your specific case.

4. Implementing solutions made by others often means just stealing their code. It is perfectly okay to steal their code as soon as you understand **what the code is doing**.

5. I strongly suggest to learn Python for everything ML related, and probably for everything else too. If you don't know any programming language (!!!) Python is the fastest way to solve that homework. If you know any programming language, learning Python is easy as eating a piece of cake. Programming in Python is easy, is fast, is fun and is rewarding. **But** if you are **skilled enough** in another language, feel free to use Java/C/assembly etc. whatever you like most ;-)

6. **Be aware of scikit-learn vs. other algorithms rows-columns convention!!!** In scikit-learn and in general in everything numpy-data related, the data is usually stored as: each row represent one sample, each column represent one feature. But for example in eigenfaces algorithm

each sample is stored into a column! So, if you wanna apply eigenfaces complexity trick you have to make a clever use of transposition: for example you could work with $154587xN$ data (eigenfaces convention) remembering to apply matrix transposition whenever you have to use scikit-learn functions (to get scikit-learn matrix convention $Nx154587$).

7. The transformation with the last 6PC cannot be automatically done by sklearn, as several of you discovered. In this case you have to explicitly extract ALL PC, discard all PC but the last 6 ones, and then manually apply the transformation to the new subspace following the learned formulas :) and if your computer has not enough memory, **use the eigenfaces trick!**

8. Before applying the PCA algorithms, remember to **normalize** the dataset matrix by substracting the mean and by dividing by variance! And after the transformation, in order to correctly plot the transformed images, remember to "**unnormalize**" the data!As to image matrix values must always be between 0 and 255 for a correct visualization. You can keep the dataset normalized for classification purpose, as having the data rnormalized usually improve the accuracy. Last thing about that: remember to normalize eigenvectors too ;-)

9.