***Student: Victor Pugliese***

***R#: 11492336***

***Course: Independent Study on Python Machine Learning for Petroleum Engineering Application (PETR 5000)***

***Self-Homework #8***

1. ***What are the main motivations for reducing a dataset’s dimensionality? What are the main drawbacks?***

The main reason is that we could turn an intractable problem into a tractable one, i.e. to speed up the training algorithm. The main drawback is that we lose information.

1. ***What is the curse of dimensionality?***

The curse of dimensionality is the fact that as the features are increased (more dimensions), we increase the risk of overfitting the training set, making the predictions less reliable.

1. ***Once a dataset’s dimensionality has been reduced, is it possible to reverse the operation? If so, how? If not, why?***

It is possible to reverse the dimensionality reduction. We can use the principal components transpose matrix.

1. ***Can PCA be used to reduce the dimensionality of a highly nonlinear dataset?***

Yes, it can if we use the Kernel trick.

1. ***Suppose you perform PCA on a 1,000-dimensional dataset, setting the explained variance ratio to 95%. How many dimensions will the resulting dataset have?***
2. ***In what cases would you use regular PCA, Incremental PCA, Randomized PCA, or Kernel PCA?***

When the data fits in the memory, we can use the regular PCA. If we have to use batches because the data is too large or for online task, we would prefer to use Incremental PCA. In case we want to reduce considerably the dimensions, we should use Randomized PCA. In case we have nonlinear datasets, we need to use Kernel PCA.

1. ***How can you evaluate the performance of a dimensionality reduction algorithm on your dataset?***

We can compare the original data and the obtained after apply the inverse algorithm. The reconstruction error gives us information about how much information is lost.

1. ***Does it make any sense to chain two different dimensionality reduction algorithms?***

*Author’s solution:* It can absolutely make sense to chain two different dimensionality reduction algorithms. A common example is using PCA to quickly get rid of a large number of useless dimensions, then applying another much slower dimensionality reduction algorithm, such as LLE. This two-step approach will likely yield the same performance as using LLE only, but in a fraction of the time.

1. ***Load the MNIST dataset (introduced in Chapter 3) and split it into a training set and a test set (take the first 60,000 instances for training, and the remaining 10,000 for testing). Train a Random Forest classifier on the dataset and time how long it takes, then evaluate the resulting model on the test set. Next, use PCA to reduce the dataset’s dimensionality, with an explained variance ratio of 95%. Train a new Random Forest classifier on the reduced dataset and see how long it takes. Was training much faster? Next evaluate the classifier on the test set: how does it compare to the previous classifier?***

See file: HML\_Chap08\_Exercise\_09.py

1. ***Use t-SNE to reduce the MNIST dataset down to two dimensions and plot the result using Matplotlib. You can use a scatterplot using 10 different colors to represent each image’s target class. Alternatively, you can write colored digits at the location of each instance, or even plot scaled-down versions of the digit images themselves (if you plot all digits, the visualization will be too cluttered, so you should either draw a random sample or plot an instance only if no other instance has already been plotted at a close distance). You should get a nice visualization with well-separated clusters of digits. Try using other dimensionality reduction algorithms such as PCA, LLE, or MDS and compare the resulting visualizations.***