Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 2.1

Exercise 2.1.1 (Jupyter Notebook) Let

$$X = \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 7 \end{bmatrix}$$

Perform the following numpy operations:

(a) Define each of X, Y and Z as a 2d numpy arrays
(b) Concatenate X and Y in the row direction (X on top of Y)
(c) Reshape Z into a 2 row and 3 column 2d numpy array
(d) Compute X +3*Y+reshapedZ
(e) Compute X*Y (componentwise multiplication with broadcasting)

Hint: look at UnsupervisedML/Examples/Section02/NumpyBasic.ipynb

Solution: see UnsupervisedML/Exercises/Section02/Exercise_02.1.1.ipynb

Exercise 2.1.2 (Jupyter Notebook) Let

$$X = \begin{bmatrix} 1 & 2 & 3 & -1 & -3 & -4 \\ 4 & 5 & 7 & -6 & -5 & -7 \\ 7 & 8 & 9 & 2 & 4 & 6 \end{bmatrix} \text{ and } a = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Perform the following numpy operations:

(a) Use np.where to find the indices where a = 0
(b) Create a 2d array from X by indexing into its columns where a=0 - call it Y
(c) Compute the mean of Y in the column direction (use np.mean with the appropriate axis option and keepdims=True)
(d) Use np.where to find the indices where a = 1
(e) Create a 2d array from X by indexing into its columns where a=1 – call it Z
(f) Compute the mean of Z in the column direction

Note: we will be using this approach for the K Means clustering algorithm

Hint: see UnsupervisedML/Examples/Section02/NumpyBasic.ipynb

Solution: see UnsupervisedML/Exercises/Section02/Exercise_02.1.2.ipynb

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 2.2

Exercise 2.2.1 (Jupyter Notebook) Let

$$W = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad X = \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 7 \end{bmatrix}$$

Compute the following (all multiplications here are matrix multiplications)

(a) $X + Y$
(b) WX
(c) $Z + Y^T$
(d) XZW^{-1}
(e) $X + Z^T$
(f) ZX
(g) $X^T Z^T$

Hint: see UnsupervisedML/Exercises/Section02/NumpyMatrixOperations.ipynb

Solution: see UnsupervisedML/Exercises/Section02/Exercise_02.2.1.ipynb

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 2.3

Exercise 2.3.1 (Jupyter Notebook) Using numpy and matplotlib functionality plot the following functions for x between -5 and 5.

(a) $f(x) = \frac{1}{1+e^{-x}}$ - Hint: use numpy exp function

(b) $f(x) = \log{(1 + e^{x})}$ Hint: use numpy log and exp functions

(c) Create an object oriented subplot with 1 row and 2 columns and add the plots in (a) and (b). Note: you need not specify the column index in the ax object since there is only 1 row. Use ax[0] and ax[1] to refer to the individual subplots in the figure.

Hint: see UnsupervisedML/Examples/Section02/NumpyBasic.ipynb and UnsupervisedML/Examples/Section02/MatplotlibBasic.ipynb

Solution: see UnsupervisedML/Exercises/Section02/Exercise_02.3.1.ipynb

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 2.4

Exercise 2.4.1 (Jupyter Notebook)

(a) Consider the following matrix:

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Plot using pcolormesh. What digit is it? (You will need to use flipud)

(b) Create the matrices for digits 3 and 6 and plot using pcolormesh.

Hint: see UnsupervisedML/Examples/Section02/MatplotlibAdvanced.ipynb

Solution: see UnsupervisedML/Exercises/Section02/Exercise_02.4.1.ipynb

Exercise 2.4.2 (Jupyter Notebook)

The goal of this problem is to create an animation that combines the 2 animations in the demo in UnsupervisedML/Examples/Section02/MatplotlibAdvanced.ipynb.

(1) Change colors for a fixed set of 50 randomly chosen data points in 2 dimensions:

(2) Change the locations of 5 randomly chosen data points in 2 dimensions:

Hint: create a single figure and create separate scatter objects to hold (1) and (2). Update both scatter objects in the update function. This approach will be used for animation for the K Means algorithm.

Solution: see UnsupervisedML/Exercises/Section02/Exercise_02.4.2.ipynb

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 2.5

Exercise 2.5.1 (Jupyter Notebook) Repeat the first part of Unsupervised/ML/Examples/Section02/PandasDemo.ipynb.
Read data from mydf1.csv and

(a) Output a Y numpy array that has the square of the entries in the “label” column of the file
(b) Output a X numpy array whose first column has the square root of the entries in the “feature 1” column and whose second column has the exponential of the entries in the “feature 2” column.

Solution: see UnsupervisedML/Exercises/Section02/Exercise_02.5.1.ipynb

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 3.1

Exercise 3.1.1 (Jupyter Notebook) Consider the messages:
"A bird in the hand is worth two in the bush", "The early bird gets the worm", "Time is money"
"Honesty is the best policy"

(a) Repeat the countvectorizer example in UnsupervisedML/Examples/Section03/SklearnText.ipynb for this new dataset including creation of the word cloud.

(b) Repeat (a) using stop words ("the", "is" "in"). Stop words are typically common words that should not have an impact to sentiment or topic. Hint: create an instance of the CountVectorizer with the format: CountVectorizer(stop_words=["the", "is", "in"]). Notice that the stop words are not in the word cloud.

Solution: See UnsupervisedML/Exercises/Section03/Exercise_03.1.1.ipynb

Exercise 3.1.1 (Jupyter Notebook) Let's go back to the example from the demo with documents:
"Call me soon", "CALL to win", "Pick me up soon"
The goal of this exercise is to generate the Tfidf matrix from the Countvectorizer matrix.

(a) The document frequency df(i) is the number of documents in which word i appears. For example, "call" appears in 2 documents (once we convert to lowercase) and "soon" appears in 2 documents. Starting with the Countvectorizer matrix, create a column vector (# of word rows and 1 column) representing the document frequency.

(b) Create the inverse document frequency vector idf(i) defined by following formula, where n is number of documents (column vector # of word rows and 1 column)

$$idf(i) = \log\left(\frac{1+n}{1+df(i)}\right) + 1$$

(c) Compute the unscaled tfidf matrix defined by:

$$U_{ij} = \text{Count(i, j)} * \text{idf(i)}$$

Here U_{ij} is entry of unscaled tfidf matrix for word i and document j and Count(i,j) is the result of the countvectorizer for word i and document j.

(d) Compute the scaled tfidf matrix T, where entries are obtained by scaling the unscaled matrix U using the following formula (W is number of words) and compare with tfidf matrix obtained directly from sklearn Tfidf vectorizer (scale entries of column j of U by the length of column j to get T)

$$T_{ij} = \frac{U_{ij}}{\left[\sum_{i=0}^{W-1} U_{ij}^2\right]^{1/2}}$$

Hint: see demo UnsupervisedML/Examples/Section03/SklearnText.ipynb

Solution: see UnsupervisedML/Exercises/Section03/Exercise_03.1.2.ipynb

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 3.2

Exercise 3.2.1 (Theory and Jupyter Notebook) Consider matrix/matrix multiplication A*A where A is an MxM matrix.

(a) What is the computational complexity (number of operations) for this calculation as $M \to \infty$ - work this out on paper.

(b) Confirm the computational complexity in (a) by computing time to perform A*A for matrices of various dimensions. Hint: see UnsupervisedML/Examples/Section03/Complexity.ipynb

Solution: see next page and also UnsupervisedML/Exercises/Section03/Exercise_03.2.1.ipynb

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Solution: Exercise 3.2.1(a)

Consider matrix multiplication $B = A * A$, where A is MxM. B is of dimension MxM, so it has M^2 entries. It can be shown that B_{ij} is dot product of row i of A and column j of A. Each dot product requires $O(M)$ operations as $M \to \infty$. Hence, the total number of operations for computing all entries of B is $O(M^3)$ as $M \to \infty$.

Solution: Exercise 3.2.1(b) see Unsupervised/Exercises/Section03/Exercise_03.2.1.ipynb

Section 3.3

Exercise 3.3.1 (Jupyter Notebook) Consider the following datasets/clusters:

$$X = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix} \qquad Y = \begin{bmatrix} 11 & 15 & -9 \\ 2 & -6 & 1 \\ 13 & 7 & -11 \\ 4 & 18 & 12 \end{bmatrix}$$

(a) Compute the mean of each dataset. (Hint: express X and Y as 2d arrays in numpy and use np.mean() function to compute the mean. Make sure that you take the mean in the appropriate direction with the axis setting and make sure you use the appropriate keepdims setting.) The output for the mean should be a 2d array column vector of dimension 3x1.

(b) Compute the distances between the 2 clusters defined as distance between cluster means.

Solution: UnsupervisedML/Exercises/Section03/Exercise_03.3.1.ipynb

Exercise 3.3.2 (Jupyter Notebook) In the demo for this section we looked at 2 different functions (looping version and vectorized version) for computing distance between a vector Y and a dataset X. Confirm that the computational complexity of this calculation is $O(M)$ operations as $M \to \infty$ for both approaches, where M is the number of data points in the dataset.

Hint: follow the approach for determining the computational complexity power shown in UnsupervisedML/Examples/Section03/Complexity.ipynb

Solution: UnsupervisedML/Exercises/Section03/Exercise_03.3.2.ipynb

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 3.4

Exercise 3.4.1 (Jupyter Notebook) Consider the following matrix:

$$X = \begin{bmatrix} 11 & 15 & -9 \\ 2 & -6 & 1 \\ 13 & 7 & -11 \\ 4 & 18 & 12 \end{bmatrix}$$

(a) Compute the compact SVD and confirm that $X = U\Sigma V^T$

(b) Confirm that each column of U has L2 length = 1 and that the columns are pairwise orthogonal. (For example confirm that $U_0^T U_1$=0 and similar for other pairs of columns.)

(c) Repeat (b) for the columns of V

Hint: see UnsupervisedML/Examples/Section03/SVD.ipynb

Solution: see UnsupervisedML/Exercises/Section03/Exercise_03.4.1.ipynb

Exercise 3.4.2 (Jupyter Notebook) Consider an MxM matrix A. Confirm that the computational complexity for the SVD calculation is $O(M^3)$ as $M \to \infty$.

Hint: follow the approach in UnsupervisedML/Examples/Section03/Complexity.ipynb

Solution: see UnsupervisedML/Exercises/Section03/Exercise_03.4.2.ipynb

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 3.5

Exercise 3.5.1 (Jupyter Notebook) Consider the following dataset:

$$X = \begin{bmatrix} 1 & -2 & 3 & -4 & 5 \\ 6 & 7 & -10 & 11 & 3 \\ 8 & 9 & 12 & -13 & 2 \end{bmatrix}$$

(a) Compute the mean of the dataset

(b) Compute the covariance matrix for the dataset

Hint: see UnsupervisedML/Examples/Section03/MeanCovariance.ipynb

Solution: see UnsupervisedML/Exercises/Section03/Exercise_03.5.1.ipynb

Section 4.1

Exercise 4.1.1 (Theory) Suppose that a dataset has M data points. Provide an explanation for the complexity estimate of $O(M^3)$ as $M \rightarrow \infty$ to determine the clusters at all levels for hierarchical clustering algorithm.

Solution: see next page

Solution 4.1.1: Suppose that there are M points in the dataset. Let us assume that we measure the distance between clusters as the distance between their cluster centres.

(1) Initially, each data point is a cluster so each cluster centre is the data point itself.
(2) At each level when we combine clusters, we only need to recompute the cluster centre for the one newly created cluster. The work to compute the cluster centre is the work to compute the mean and is at most $O(M)$ operations as $M \to \infty$
(3) At the level that there are m clusters, we need to compute m(m-1)/2 pairwise distances between cluster centres. Each of these distance calculations requires $O(d)$ operations as $d \to \infty$, so is independent of M.

The amount of work in (2) is significantly less than amount of work in (3). If we sum the amount of work in (3) over m = 2,3,4,…,M we get

$$\sum_{m=2}^{M} m(m-1)/2 = O(M^3) \quad as\ M \to \infty$$

Specifically, you may have seen the result

$$\sum_{m=1}^{M} m^2 = M(M+1)(2M+1)/6$$

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 4.3

Exercise 4.3.1 (Testing) Experiment with the driver_hierarchical.py code and try the other cases: blobs, varied_blobs2, aniso, noisy_moons, and noisy_circles. For the latter 2 cases, set the nlevel=2 when plotting results.

Exercise 4.3.2 (Programming) Suppose that a dataset has M data points. For the Hierarchical Clustering Algorithm, confirm that the number of operations is $O(M^3)$ as $M \to \infty$ by measuring time for completing the algorithm for various values of M.

Hint: use one of the sklearn datasets for this Exercise and run the hierarchical clustering algorithm code for various values of M. I suggest going up to M = 200 or else the clustering code may take too long to run. See UnsupervisedML/Exercises/Section03/Complexity.ipynb notebook for an example of how to determine the power in the complexity estimate.

Solution: see UnsupervisedML/Exercises/Section04/Exercise_04.3.2.py. Run this program in folder UnsupervisedML/Code/Programs

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 5.1

Exercise 5.1.1 (Theory) Suppose that a dataset has M data points. Can you provide an explanation as to why the worst case complexity for number of operations for the DBSCAN algorithm is $O(M^2)$ as $M \to \infty$.

Solution: see next page

Exercise 5.1.2 (Programming) Create a program to generate the nearest neighbour plots as seen in the lecture. Use the varied_blob1 dataset with M = 200 and 1500 data points and set minpts = 4.

Solution: see file UnsupervisedML/Exercises/Section05/Exercise_05.1.2.py. Run this program in the UnsupervisedML/Code/Programs folder

Exercise 5.1.3 (Theory) Suppose that DBSCAN is performed with minpts = 4 and $\varepsilon = 0.3$. It found that a new cluster is created with exactly 2 points. How is this possible?

Solution 5.1.1: The amount of work to determine the number of points in the ε neighbourhood for a single data point is $O(M)$ as $M \to \infty$, since one must compute the distances to all other data points before determining the number within a distance ε. In the case of all noise points, we must do this distance calculation for all data points. Since there are M data points, then total work is $O(M^2)$ as $M \to \infty$. There may be more efficient ways to determine points in the ε neighbourhood. These are mentioned in the Wikipedia page on DBSCAN (see section on Complexity):

https://en.wikipedia.org/wiki/DBSCAN

Solution 5.1.3: For DBSCAN with minpts = 4 and $\varepsilon = 0.3$ a cluster is created if there is a core point (with at least 4 points within $\varepsilon = 0.3$, including the core point itself.) If 2 of the neighbours are border points that belong to other clusters and they have been assigned to that other cluster first, then one is left with a cluster with 2 points.

Section 5.3

Exercise 5.3.1 (Testing) Experiment with the driver_kmeans.py code and try the other cases: blobs, varied_blobs2, aniso, noisy_moons, and noisy_circles. Experiment with the minpts and epsilon and the number of samples as well.

Exercise 5.3.2 (Programming) Suppose that a dataset has M data points. Confirm that the number operations is $O(M^2)$ as $M \to \infty$ for the DBSCAN algorithm by measuring time for completing the algorithm for various values of M.

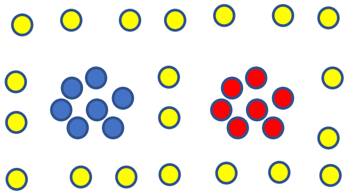
Hint: use one of the sklearn datasets for this Exercise and run the DBSCAN algorithm code for various values of M.

Solution: see UnsupervisedML/Exercises/Section05/Exercise_05.3.2.py. Run this program in the UnsupervisedML/Code/Programs folder.

Section 6.1

Exercise 6.1.1 (Theory) Suppose that a dataset has M data points. Can you provide an explanation as to why the complexity for number of operations for the K Means algorithm is $O(M)$ as $M \to \infty$.

Exercise 6.1.2 (Theory) Can the final cluster assignment in the figure below be generated by the K Means algorithm? Provide an explanation for your answer. (Blue dots belong to cluster 0, Red dots belong to cluster 1, Yellow dots belong to cluster 2.)



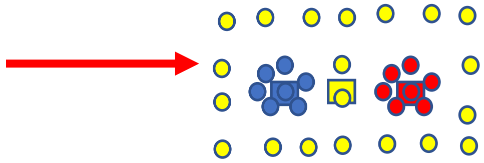
Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Solution 6.1.1: At each iteration of the algorithm one must perform the following steps:

(1) Compute distance between each data point and each of cluster means
(2) Compute the closest cluster mean for each data point
(3) Compute the updated cluster mean values

Each of these steps requires $O(M)$ operations as $M \to \infty$. If we assume a fixed number of iterations for convergence as M increases, the total work is $O(M)$ operations as $M \to \infty$.

Solution 6.1.2: The figure below shows the final cluster assignments with the cluster means in the square symbols. Remember that a data point belongs to the cluster corresponding to the nearest cluster mean, so the yellow point on the left should be in the blue cluster and not the yellow cluster. Same logic applies to other yellow data points on left and right.



This point is closer to blue mean than yellow mean, so it should belong to blue cluster

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 6.3

Exercise 6.3.1 (Testing) Experiment with the driver_kmeans.py code and try the other cases: blobs, varied_blobs2, aniso, noisy_moons, and noisy_circles. Experiment with the number of clusters and initialization and number of samples.

Exercise 6.3.2 (Programming) Write a program to create the elbow plot from Section 6.1 for the varied_blobs1 dataset with 200 points. Use the K Means with “random” initialization.

Solution: see UnsupervisedML/Exercises/Section06/Exercise_06.3.2.py. Run this program in folder UnsupervisedML/Code/Programs.

Exercise 6.3.3 (Programming) Suppose that a dataset has M data points. Confirm that the number operations is $O(M)$ as $M \to \infty$ for the K Means algorithm by measuring time for completing the algorithm for various values of M.

Hint: use one of the sklearn datasets for this Exercise and run the K Means algorithm code for various values of M.

Solution: see UnsupervisedML/Exercises/Section06/Exercise_06.3.3.py. Run this program in folder UnsupervisedML/Code/Programs

Section 7.1

Exercise 7.1.1 (Jupyter Notebook)

Let ϕ=1 and define:

$$\mu = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 2 & 0.5 \\ 0.5 & 3 \end{bmatrix}$$

Plot the contours for the 2d normal distribution pdf

$$N(X, \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(X-\mu)^T \Sigma^{-1}(X-\mu)} = c$$

for c = 0.04, 0.06 and d=2 using the Matplotlib Ellipse approach shown in the demo. What happens when c is large? Why?

Hint: see UnsupervisedML/Examples/Section07/Normal.ipynb

Solution: see next page and UnsupervisedML/Exercises/Section07/Exercise_07.1.1.ipynb

Exercise 7.1.2 (Jupyter Notebook)

The goal of this problem is to help you create a vectorized version of the function to compute the multi-dimensional normal probability density function.

(a) We start off with a vectorized version of the dot product. Define the following d-dimensional column vectors: $X_0, \dots, X_{M-1}$ and $Y_0, \dots, Y_{M-1}$. Define X and Y to be the dxM dimensional matrices:

$$X = [X_0 \ \ X_1 \dots X_{M-1}] \quad Y = [Y_0 \ \ Y_1 \dots Y_{M-1}]$$

Create a function using numpy functionality that takes in X and Y and computes the dot products $X_0^T Y_0, X_1^T Y_1, \dots, X_{M-1}^T Y_{M-1}$ **without explicit looping**. The output should be a 2d numpy array with dimension 1xM. Hint: use component-wise multiplication and numpy sum.

(b) Using the approach from (a) create a function that takes in $X = [X_0 \ \ X_1 \dots X_{M-1}]$ (dxM), mean μ (dx1), and covariance Σ (dxd) and computes the normal pdf $N(X, \mu, \Sigma)$ for each of the vectors in X. The output should be a 2d numpy array (1xM). There should not be any explicit looping. Hint: one can compute $\Sigma^{-1}(X-\mu)$ using appropriate numpy functions and then compute $(X-\mu)^T \Sigma^{-1}(X-\mu)$ using the approach in (a).

(c) Consider the normal pdf calculation for dataset X for d=2 dimensions and M =100000 samples for the mean and covariance from 7.1.1. Compare time for computing the normal pdf for columns of X for the non-vectorized version (from demo) and the vectorized version (from (b)).

Solution: see UnsupervisedML/Exercises/Section07/Exercise_07.1.2.ipynb

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Exercise 7.1.1: Solution: Consider the equation:

$$N(X,\mu,\Sigma)=\frac{1}{\sqrt{(2\pi)^d|\Sigma|}}e^{-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)}=c$$

If c is sufficiently large, then there are no real value solutions X for this equation. From the document UnsupervisedML_GMM.pdf, the width (long dimension) of the elliptical contour is:

$$width=2\sqrt{-2\log\left[c2\pi\sqrt{|\Sigma|}\right]}\sqrt{\sigma_0}$$

If c is too large, then the term in the first square root will be negative, so you will get an error when trying to compute width. Same issue occurs for the height.

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 7.2

Exercise 7.2.1 (Theory) Suppose that a dataset has M data points. Can you provide an explanation as to why the complexity for number of operations for the Gaussian Mixture Model algorithm is $O(M)$ as $M \rightarrow \infty$.

Solution: see next page

Solution 7.2.1: At each iteration of the algorithm one performs:

(1) Expectation Step:
Compute the conditional probabilities γ_{ki} for clusters k=0,…,K-1 and data points i=0,…,M-1.

(2) Maximization Step:
Compute the means $\{\mu_k\}$, covariances $\{\Sigma_k\}$, weights $\{\phi_k\}$ for clusters k = 0,…,K-1.

In (1), there K*M probabilities to compute, where K is the number of clusters and M is the number of data points. Each of these probabilities involves a computation of the normal distribution pdf given by the formula:

$$N(X,\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^d|\Sigma|}} e^{-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)}$$

The calculation of the log likelihood function and the cluster assignments also is proportional to M.

In (2), one must compute K mean, covariances, and K weight updates. Each of these calculations involves summing from 0 to M-1. For example, for the means we have:

$$\mu_k = \frac{1}{M_k}\sum_{i=0}^{M-1}\gamma_{ki}X_i$$

Hence, the amount of work for each of the mean, covariance, and weight calculations is $O(M)$ operations as $M \to \infty$ as the sum is over the M data points.

Looking at each of these calculations carefully, it can be shown that the total amount of work $O(M)$ operations as $M \to \infty$. Note that the amount of will depend on the number of clusters, the dimension, and the number of iterations. The O(M) estimate assumes that the number of iterations is bounded (has a maximum independent of the number of data points M).

Section 7.4

Exercise 7.4.1 (Testing) Experiment with the driver_gaussianmm.py code and try the other cases: blobs, varied_blobs2, aniso, noisy_moons, and noisy_circles. Experiment with the number of clusters and number of samples.

Exercise 7.4.2 (Programming) For the varied_blobs1 dataset, write a program to determine the number of clusters for the Gaussian Mixture Model using the elbow method. (Plot final log likelihood value as a function of number of clusters for various number of clusters for the varied_blobs1 dataset with 200 points.)

Solution: see UnsupervisedML/Exercises/Section07/Exercise_7.3.2.py. Run this program in folder UnsupervisedML/Code/Programs.

Exercise 7.4.3 (Programming) Suppose that a dataset has M data points. Confirm that the number operations is $O(M)$ as $M \to \infty$ for the GaussianMM algorithm by measuring time for completing the algorithm for various values of M.

Hint: use one of the sklearn datasets for this Exercise and run the GaussianMM algorithm code for various values of M.

Solution: see UnsupervisedML/Exercises/Section07/Exercise_7.3.3.py. Run this program in folder UnsupervisedML/Code/Programs

Exercise 7.4.4 (Programming) Write a code to perform the Expectation Maximization algorithm in the case of the "spherical" approach. In this approach each covariance matrix is diagonal with the same variance value in each entry. The algorithm for this case is described in the UnsupervisedML/Resources/UnsupervisedML_GMM.pdf file.

Hints: You can implement the algorithm using any approach that you like. Here are suggestions:

(1) Create a function to compute the multi-dimensional normal pdf in case of constant variance (instead of a full covariance matrix).
(2) Create a derived class gaussianmm_spherical with parent gaussianmm
(3) Use a variable to track the variance for each component of the mixture at each iteration
(4) Keep self.Covsave variable and update it after each maximization step with the constant diagonal covariance matrix for each component at each iteration. This will allow you to use the plot_cluster and plot_cluster_animation methods without any changes.

Use the spherical approach to perform clustering for the varied_blobs1 dataset.

Solution: see UnsupervisedML/Exercises/Section07/gaussianmm_spherical.py and Exercise_7.4.4.py and run these files in folder UnsupervisedML/Code/Programs. I have created a separate class. It is probably better coding practice to implement both the original case and spherical case in the same class.

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

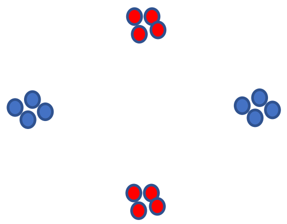
Section 8.1

Exercise 8.1.1 (Programming) Confirm that the number of operations for the Davies-Bouldin calculation is $O(M)$ as $M \to \infty$. Confirm that the number of operations for the Silhouette is $O(M^2)$ as $M \to \infty$.

Hint: perform clustering with K Means for one of the sklearn datasets and measure time to compute Davies-Bouldin and Silhouette as a function of M.

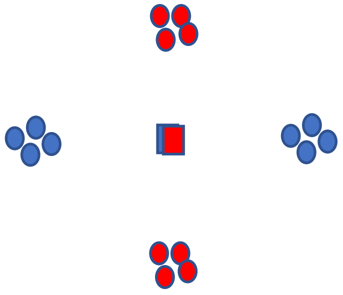
Solution: see next page and UnsupervisedML/Exercises/Section08/Exercise_08.1.1.py

Exercise 8.1.2 (Theory) What can you say about the Davies-Bouldin index value and the Silhouette Index value for the following clustering result?

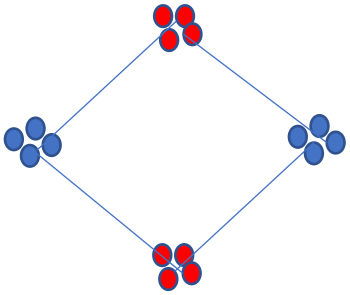


Solution 8.1.1: For Davies-Bouldin, I am finding a complexity power of 0.87 instead of 1. For Silhouette, I am finding a complexity power of around 1.5 instead of 2. The Silhouette power seems to be on the low side. For Silhouette I am using M up to around 25000. This suggests one should increase the value of M to get a result that is closer to the power of 2.

Solution 8.1.2: For Davies-Bouldin the centres of the red and blue cluster are near each other as shown in the figure below. The average distance between points in the cluster to the cluster centre is >0. I expect that the DB index value will be much greater than 1.



For Silhouette, look at the figure below. Let’s say that the line segments are roughly equal to 1. Hence, for each data point, average distance to points in another cluster b is roughly 1. For distances to other points within cluster, let’s say that distance to 3 of the points is roughly 0 and to the other 4 points is roughly $\sqrt{2} \approx 1.4$. Hence, average distance to other points within the cluster is roughly a=4*1.4/7 = 0.8. Hence, for each data point S = (b-a)/max(a,b) = (1-0.8)/1 = 0.2. Averaging over all data points, my rough estimate for S for entire dataset is 0.2



Section 9.1

Exercise 9.1.1 (Jupyter Notebook) Consider the following dataset:

$$X = \begin{bmatrix} 1 & -2 & 7 & -8 \\ -3 & 4 & -9 & 10 \\ 5 & 6 & -11 & 12 \end{bmatrix}$$

(a) Compute the compact SVD of X using numpy.linalg.svd function and confirm that $X = U\Sigma V^T$.
(b) Determine the new dataset R using only the first 2 principal components.
(c) Determine the reconstructed X using the first 2 principal components.
(d) What proportion of the variance is captured by the first 2 principal components?

Solution: See UnsupervisedML/Exercises/Section09/Exercise_09.1.1.ipynb

Exercise 9.1.2 (Jupyter Notebook) Consider the following dataset:

$$X = \begin{bmatrix} 1 & 3 \\ 5 & 7 \\ 2 & 4 \\ 6 & 8 \end{bmatrix} \quad X_0 = \begin{bmatrix} 1 \\ 5 \\ 2 \\ 6 \end{bmatrix} \quad X_1 = \begin{bmatrix} 3 \\ 7 \\ 4 \\ 8 \end{bmatrix}$$

This is an example of a dataset with 2 data points and 4 dimensions. We can represent X in 2 dimensions and still retain all the “information”.

(a) Compute the compact SVD of X using the numpy.linalg.svd function.
(b) Compute the representation R using 2 dimensions.
(c) Determine the length (L2 Euclidean) of the individual data points X_0 and X_1 and the L2 distance between these points.
(d) Determine the length (L2 Euclidean) of the individual data points R_0 and R_1 and the L2 distance between these points. The lengths and distance should be the same as in (c), so no “information” is lost.

Solution: see UnsupervisedML/Exercises/Section09/Exercise_09.1.2.ipynb

Section 9.3

Exercise 9.3.1 (Testing) Consider the matrix:

$$X = \begin{bmatrix} 1 & 2 & 11 & -12 & 2 \\ 3 & 4 & 13 & 1 & -7 \\ -1 & -2 & -1 & -2 & -3 \\ -7 & -1 & -4 & -5 & -6 \end{bmatrix}$$

Use the code in pca.py to answer the following questions:

(a) Compute the compact version of the singular value decomposition and plot the cumulative variance proportion.

(b) Compute the reduced dimension version of X-Xmean capturing 0.99 proportion of the variance.

(c) Compute the reconstructed version of X capturing 0.99 proportion of the variance.

Solution: see UnsupervisedML/Exercises/Section09/Exercise_09.3.1.py and run this file in UnsupervisedML/Code/Programs

Exercise 9.3.2 (Testing) Consider a 50x1000 random dataset (50 dimensions and 1000 samples). Use the pca code to estimate number of principal components needed to capture 0.85 proportion of the variance.

Solution: see next page and UnsupervisedML/Exercises/Section09/Exercise_09.3.2.py and run this file in UnsupervisedML/Code/Programs

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Exercise 9.3.2 Solution: in my example 40 principal components are required to capture 0.85 proportion of the variance.

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 9.4

Exercise 9.4.1 (Testing) Run the driver_pca_mnist.py (with 60000 samples and the train dataset) to answer the following questions

(a) What is minimum number of dimensions to capture 50%, 90%, and 99% of the variance?
(b) What is the minimum number of dimensions for you to be able distinguish the digits?

If 60000 samples is too many for your computer, change to 10000 samples (or fewer) with the valid dataset.

Solution: see next page

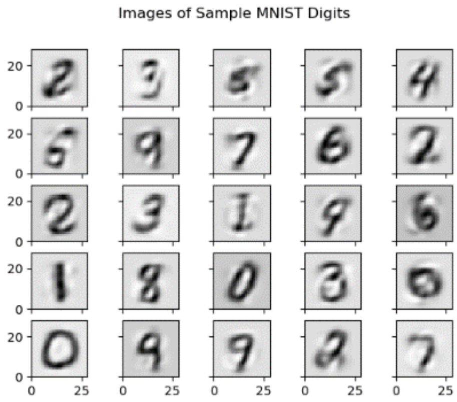
Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

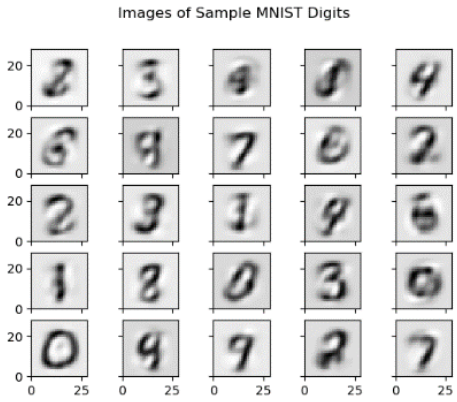
Exercise 9.4.1 Solution: run the driver_pca_mnist.py in UnsupervisedML/Code/Programs and vary the variance_capture setting.

For 0.50 variance capture: 11 dimensions
For 0.95 variance capture: 154 dimensions
For 0.99 variance capture: 331 dimensions

Exercise 9.4.2 Solution:

The plot on the left is based on 20 dimensions (capturing 0.644 proportion of the variance). The plot on the right is based on 15 dimensions (capturing 0.529 proportion of the variance). With the figure on the right, the digits are a bit fuzzy and starting to be difficult to identify.





Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 10.2

Exercise 10.2.1 (Programming) Perform clustering for the Iris Flower dataset using the DBSCAN, K Means, and Gaussian Mixture Model methods after using PCA to reduce the number of dimensions to 2.

Solution: See files
UnsupervisedML/Exercises/Exercise_10.2.1_DBSCAN.py
UnsupervisedML/Exercises/Exercise_10.2.1_KMeans.py
UnsupervisedML/Exercises/Exercise_10.2.1_GMM.py
Run these files in the UnsupervisedML/Code/Programs folder.

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 10.3
Exercise 10.3.1 (Testing) The idea with this Exercise is to determine the lowest proportion of variance capture that still yields reasonable results for MNIST Digits Dataset clustering. I will let you define "reasonable" for yourself. Experiment with the casestudy_mnist.py code and examine results as a function of the PCA variance capture amount. (Use the K Means algorithm for clustering)

Exercise 10.3.2 (Testing) In Section 07, we talked about the "spherical" approach for the Gaussian Mixture Model. This approach appears to be more stable and runs faster than the full covariance matrix approach that is implemented in the file gaussianmm.py. Do the same thing as Exercise 10.3.1 with this spherical approach.

Solution: use the code in UnsupervisedML/Exercises/Section07/gaussianmm_spherical.py with driver in UnsupervisedML/Exercises/Section10/Exercise_10.3.2.py. Run these files in the folder UnsupervisedML/Code/Programs.

Course: Unsupervised Machine Learning with Python
Solution files located at Github site: https://github.com/satishchandrareddy/UnsupervisedML

Section 10.4

Exercise 10.4.1 (Testing) The idea with this Exercise is to determine the lowest proportion of variance capture that still yields reasonable results for BBC Text clustering. I will let you define “reasonable” for yourself. Experiment with the casestudy_bbctext.py code and examine results as a function of the PCA variance capture amount.