CSDS 491: Assignment 4

Shaochen (Henry) ZHONG, sxz517@case.edu

Due and submitted on 04/21/2021
Spring 2021, Dr. Lewicki

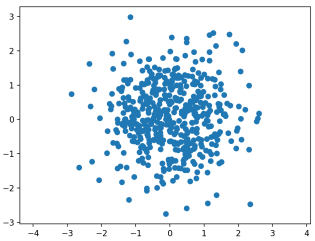
E1. Multivariate Gaussians

Please refer to `A4/code/e1.py` for the code implmentations of below plots.

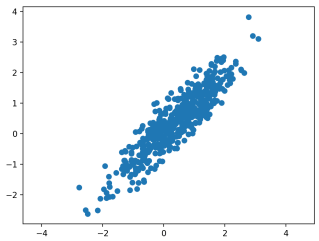
1.1.

For 2D multivariate normal we have $\Sigma = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix}$. Note:

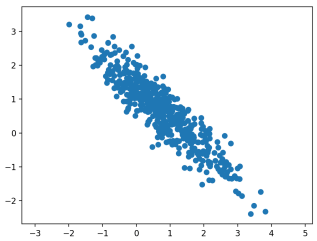
- For x and y to be uncorrelated, we have $\sigma_{xy} = \sigma_{yx} = 0$. For demo, we have $\mathcal{N}(\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix})$.
- For x and y to be uncorrelated, we have $\sigma_{xy} = \sigma_{yx} > 0$. For demo, we have $\mathcal{N}(\mu = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}, \Sigma = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix})$.
- For x and y to be anti-correlated, we have $\sigma_{xy} = \sigma_{yx} < 0$. For demo, we have $\mathcal{N}(\mu = \begin{bmatrix} 0.7 \\ 0.7 \end{bmatrix}, \Sigma = \begin{bmatrix} 1 & -0.9 \\ -0.9 & 1 \end{bmatrix})$.



Uncorrelated



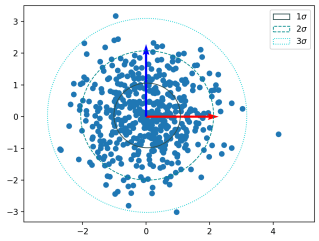
Correlated



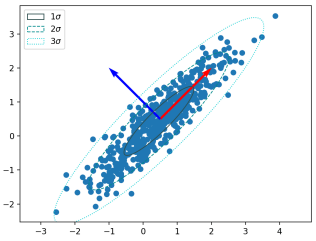
Anti-correlated

1.2.

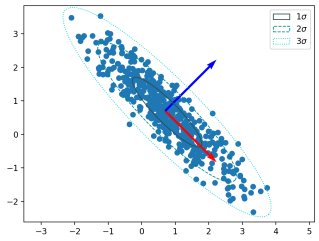
Assuming the same μ and Σ setup in **Exercise 1.1**, we have the $1-, 2-, 3-\sigma$ countour plots being (note eigenvectors are scaled `4x` for visual purposes):



Uncorrelated



Correlated



Anti-correlated

E2. Linear Gaussian Models

2.1.

Known the characteristic functions of x and z is:

$$\phi_x(u) = \exp(iu^T\mu_x - \frac{1}{2}u^T\Sigma_x u)$$
$$\phi_z(u) = \exp(iu^T\mu_z - \frac{1}{2}u^T\Sigma_z u)$$

Since x and z are independent, their sum will still conform to the normal distribution. We may confirm it by analysing the characteristic functions of $x+z$, which is $\phi_{x+z}(u) = E(e^{it(x+z)}) = E(e^{itx}) \cdot E(e^{itz}) = \phi_x(u)\phi_z(u)$, then we have:

$$\begin{aligned}\phi_{x+z}(u) = \phi_x(u)\phi_z(u) &= \exp(iu^T\mu_x - \frac{1}{2}u^T\Sigma_x u) \cdot \exp(iu^T\mu_z - \frac{1}{2}u^T\Sigma_z u) \\ &= \exp(iu^T(\mu_x + \mu_z) - \frac{1}{2}u^T(\Sigma_x + \Sigma_z)u) \\ \implies \phi_y(u) &= \exp(iu^T\mu_y - \frac{1}{2}u^T\Sigma_y u)\end{aligned}$$

Thus, we have $p(y) = \mathcal{N}(\mu_x + \mu_z, \Sigma_x + \Sigma_z)$.

2.2.

For clarity, we let $p(x, y) = \mathcal{N}(\mu, \Sigma)$, where:

$$\mu = \begin{bmatrix}\mu_x \\ \mu_y\end{bmatrix}$$
$$\Sigma = \begin{bmatrix}\Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy}\end{bmatrix}$$

Note each of Σab is a matrix of $a \times b$.
We further know that there must be $p(y \mid x) = \mathcal{N}(\overline{\mu}, \overline{\Sigma})$, where:

$$\begin{aligned}\overline{\mu} &= \mu_y + \Sigma_{yx}\Sigma_{xx}^{-1}(x - \mu_x) \\ \overline{\Sigma} &= \Sigma_{yy} - \Sigma_{yx}\Sigma_{xx}^{-1}\sigma yx\end{aligned}$$

I don't know if we are expected to show the proof, but here's a trick that can do it rather quickly. Assume $z = y + Ax$ (not to be confused with the z in **Question 2.1**) with $A = -\Sigma_{yx}\Sigma_{xx}^{-1}$, we have:

$$\begin{aligned}cov(z, x) &= cov(y + Ax, x) = cov(y, x) + A(x, x) \\ &= \Sigma_{yx} + (-\Sigma_{yx}\Sigma_{xx}^{-}1)\Sigma_{xx} \\ &\quad \text{Now to deduce } E(y \mid x) \\ &\Rightarrow \begin{cases} E(z|x) & = E(z) = E(y + Ax) = \mu_y + A\mu_x = \mu_y - \Sigma_{yx}\Sigma_{xx}^{-1}\mu_x \\ E(z|x) & = E(y + Ax \mid x) = E(y \mid x) + AE(x \mid x) \end{cases} \\ \Longrightarrow E(y \mid x) &= E(z|x) - AE(x \mid x) = \mu_y - \Sigma_{yx}\Sigma_{xx}^{-1}\mu_x - (-\Sigma_{yx}\Sigma_{xx}^{-1})x \\ &= \mu_y + \Sigma_{yx}\Sigma_{xx}^{-1}(x - \mu_x) = \overline{\mu}\end{aligned}$$

Similarily, we may find out $\overline{\Sigma}$ as (note that $cov(x, y) = cov(y, x)$):

$$\begin{aligned}var(y \mid x) &= var(z) = var(y + Ax) \\ &= var(y) + Avar(x)A' + Acov(y, x) + cov(x, y)A' \\ &= \Sigma_{yy} + \Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xx}\Sigma_{xx}^{-1}\Sigma_{xy} - \Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{yx} - \Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy} \\ &= \Sigma_{yy} + \Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy} - 2\Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy} \\ \Longrightarrow \overline{\Sigma} &= \Sigma_{yy} - \Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy}\end{aligned}$$

As know from **Exercise 2.1** that $p(y) = \mathcal{N}(\mu_x + \mu_z, \Sigma_x + \Sigma_z)$ and $p(x) = \mathcal{N}(\mu_x, \Sigma_x)$, we have $p(y \mid x) = \mathcal{N}(\overline{\mu}, \overline{\Sigma})$, where:

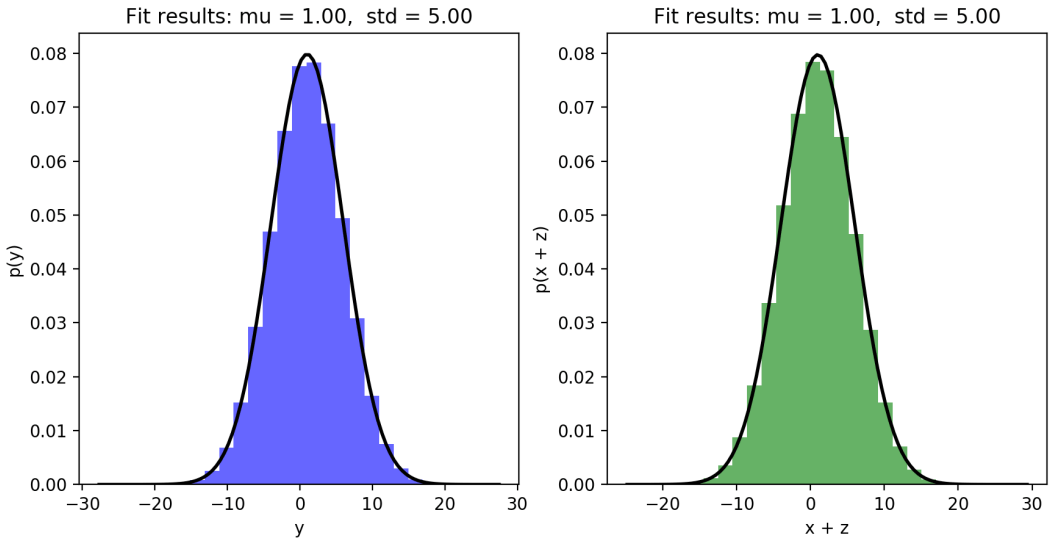
$$\begin{aligned}\overline{\mu} &= \mu_y + \Sigma_{yx}\Sigma_{xx}^{-1}(x - \mu_x) = \mu_x + \mu_z + \Sigma_{yx}\Sigma_{xx}^{-1}(x - \mu_x) \\ \overline{\Sigma} &= \Sigma_{yy} - \Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy} = \Sigma_x + \Sigma_z - \Sigma_{yx}\Sigma_{xx}^{-1}\Sigma_{xy}\end{aligned}$$

2.3.

According to **Exercise 2.1**, for $y = x + z$, we have $y \sim \mathcal{N}(\mu_x + \mu_z, \Sigma_x + \Sigma_z)$. For the ease of demo, let $\sigma_x = 3$ and $\sigma_z = 4$ so that we may have the nice $\sigma_x^2 + \sigma_z^2 = \sigma_y^2 \Longleftrightarrow 3^2 + 4^2 = 5^2$ relationship. In such case, we may have:

$$p(x) = \mathcal{N}(\mu_x = 0, \Sigma_x = 3^2)$$
$$p(y) = \mathcal{N}(\mu_y = 1, \Sigma_y = 4^2)$$
$$p(z) = \mathcal{N}(\mu_x + \mu_z, \Sigma_x + \Sigma_z) = \mathcal{N}(\mu_x = 1, \Sigma_x = 5^2)$$

The plots will look like:



We may tell these two plots are close resemblance of one and another by inspecting the shape of graph and the parameter of the fitted curve (showed as title of each plot).

Please refer to `code/p2.py` for the code implementation.

E3. dimensionality Reduction and PCA

3.1.

The dataset I used in this exercise is the UCI Machine Learning *Glass Classification* dataset I obtained from Kaggle.

```
       RI     Na   Mg   Al     Si     K    Ca  Ba  Fe  Type
0  1.52101  13.64  4.49  1.10  71.78  0.06  8.75  0.0  0.0     1
1  1.51761  13.89  3.60  1.36  72.73  0.48  7.83  0.0  0.0     1
2  1.51618  13.53  3.55  1.54  72.99  0.39  7.78  0.0  0.0     1
3  1.51766  13.21  3.69  1.29  72.61  0.57  8.22  0.0  0.0     1
4  1.51742  13.27  3.62  1.24  73.08  0.55  8.07  0.0  0.0     1
...
(214, 10)
```

As showned above, it is a dataset of 214 sample, each with 9 attributes, and each sample has a label that belong to 7 of the potential glass types. The attributes is rather straightforward, with `RI` being refractive index, and all the others are chemistry elements measured by weight percent in

corresponding oxide (e.g. `Na` is Sodium, `Mg` is Magnesium, etc). The output labels are something among the lines of `container`, `tableware`, and `headlamps`.

I opted to use this dataset as this is one of the few lightweight datasets with continues variables that I may find, and it has more than 6 dimensions. It probably doesn't make much sense to illustrate the dataset further, but the Kaggle page I linked above has some distribution plots for each element – if you are interested. One thing to note about this dataset is, due to its lightweight, the dataset is really unbalanced as it has `70` samples on `Type 1`, `76` samples on `Type 1`, but only `17, 13, 9, 29` samples on `Type 3, 5, 6, 7` respectively (yes, `Type 4: vehicle` is entirely missing).

3.2.

As we can't nicely plot 9-dimensional data. The first three eigenvalues and their corresponding eigenvectors are:

```
3.00200916e+00 1.65917340e+00 6.79576475e-01

[-9.28126899e-04 1.52290883e-03 -1.37689385e-03 3.10643441e-04
-7.12950233e-04 -1.82174928e-03 -3.32594524e-04 -4.12235755e-03 9.99986948e-01]
[-1.72248332e-02 -3.98797552e-01 -6.54934730e-01 -3.46599960e-01
3.98381798e-01 1.55680962e-02 -3.76900981e-02 -3.62242832e-01 -1.39622177e-03]
[ 7.23534913e-01 5.43050989e-01 -1.31198879e-01 -9.86931157e-02
-7.68490459e-02 4.77602532e-02 -7.49534298e-02 -3.75274748e-01 -1.84522571e-03]
```

First we may confirm that the eigenvalues are indeed sorted in decreasing order, suggesting an eigenvalue with smaller index implies greater squared sum of variances and therefore accounts for higher percentage of variation.

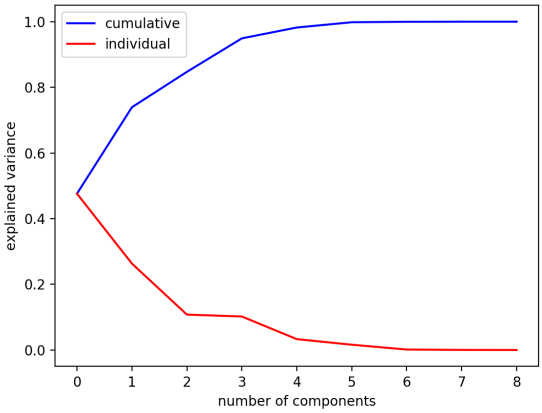
We may also confirm that these eigenvectors are indeed 9-dimensional as expected, and they are orthognal to eachother as the inner products between them yield 0:

```
<v_1, v_2> = 4.336808689942018e-19
<v_2, v_3> = -5.551115123125783e-17
<v_1, v_3> = -4.336808689942018e-19
```

(There are not exactly zero due to datatype accuracy problem, but with at least 10^{-17} they are close enough to zero.)

Note the magnitutde of the entries of each eigenvalues are similar, with the exception that the first eigenvector shows greater value on element `Fe`, the first eigenvector shows lesser value on element `Mg`, and the third eigenvector shows greater value on `RI` (the reflective index) – suggesting that the classification of glass samples might be postively correlated to `Fe` and `RI`, and negatively correlated to `Mg`. In terms of co-vary between eigenvectors, it is hard to tell if there is much consistency.

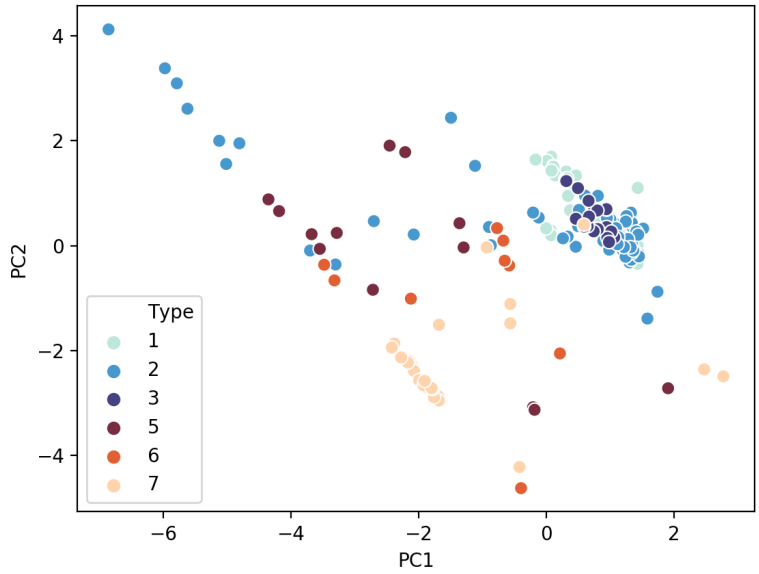
3.3.



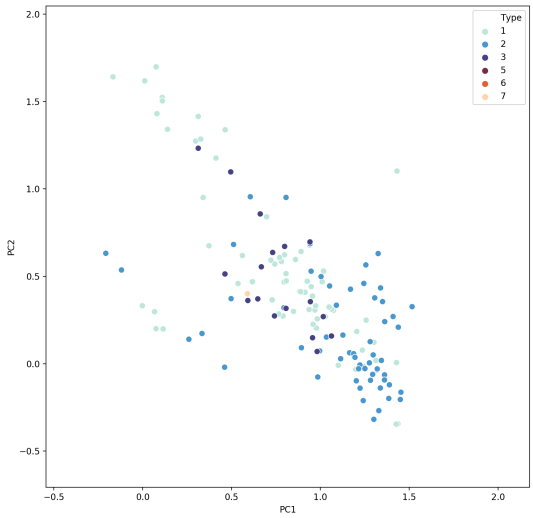
The red line represents the percentage of variation that particular principle component accounts for. In this case, as we are ploting with respect to the decreasing order of eigenvalues, each principle component accountss for less variation than the previous one. Reversely, the blue, whic represents the cumulative variance, is increasing as we add in more principle components – and when we have all principle components (which is the dimension of the attributes or number of samples, in this case it is the former) considered, the cummulative explained variance reaches 1.

3.4.

By reducing the dataset to 2 dimensions (`PC1, PC2`) – meaning that these two eigenvectors has the two largest eigenvalues, suggesting the range of data projection are more distinguishiable on these two eigenvectors and therefore they contain most information – we may have the following plot. Note the data were standardized with respect to their mean in this plot and throughout the PCA process, so that we won't have one catagory of data dominating all the others:



I guess we may say the first two principle components perform pretty well on `Type 7` glass, but not much the others, especially between `Type 1, 2, 3`, a zoom in plot will look like this:



This is something rather expected as we have 7 output labels (in fact, 6 output labels as we have no `Type 4`), but only 2 dimensions to project the clustering. The performance is further limited due to the unbalanced nature of having significantly more `Type 1` and `Type 2` samples, granting `Type 1&2` glasses to display more features that are potentially similar to other types of glasses (just like doing multiple testing).

E4. Gaussian Mixture Models

Please refer to `A4/code/e4.py` for code implementation. Note the plotting is refered to this, and the GMM is refered to this tutorials.

4.1

1 4.1

The EM algorithm iteratively performs the `expectation` step and `maximization` step – thus "EM" – until the log-likelihood (the indicator of convergence) falls into an acceptable `thlr`.

Expectation Step

The expectation step intends to solve the inference problem: which Gaussian distribution generated each of the data point? By applying Baye's rule, the conditional probability of giving a data point $z_i = m$ given $z_i \in N_k$ can be represented as:

$$\begin{aligned}\gamma_k = p(z_i = m|N_k) &= \frac{p(z_i = k)p(N_k|z_i = m)}{p(N_k)} \\ &= \frac{p(z_i = k)p(N_k|z_i = m)}{\sum_{j=1}^{K} p(z_i = j)p(N_k|z_i = j)} \\ &= \frac{\pi_k \mathcal{N}(N_k|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \mathcal{N}(N_k|\mu_j, \Sigma_j)}\end{aligned}$$

Based on the parameters γ_k, π_k, μ_k, and Σ_k, we are able to compute the log-likelihood for the purpose of optimization:

$$\sum_k \sum_i \gamma_k^{(i)} log(\pi_k) + \sum_k \sum_i \gamma_k^{(i)} log(\mathcal{N}(N_k^{(i)|\mu_k, \Sigma_k}))$$

The stopping criteria for the iteration is when the change of likelihood falls into a tolerance range that we predefined. We also need to define the maximum number of iterations in order to prevent it runs into infinite loop.

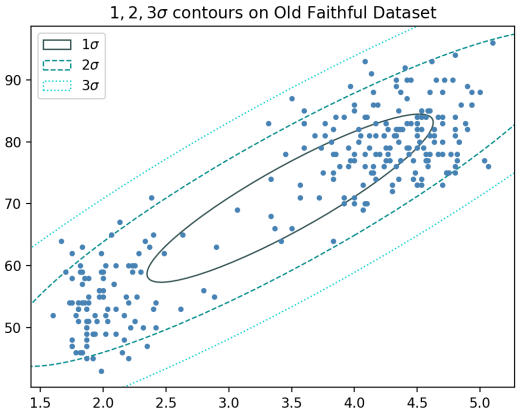
Maximization Step

Since the change of the membership of data points will also affect the distributions, we can derive a closed form updates for all the parameters (the means μ, the responsibility π, and the covariance matrix Σ).

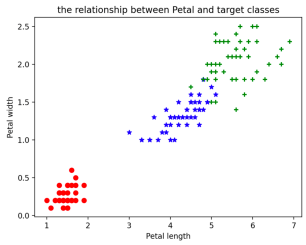
$$\begin{aligned}\mu_k &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} x^{(n)} \\ \Sigma_k &= \frac{1}{N_k} \sum_{n=1}^{N} (x^n - \mu_k)(x^n - \mu_k)^T \\ \pi_k &= \frac{N_k}{N}\end{aligned}$$

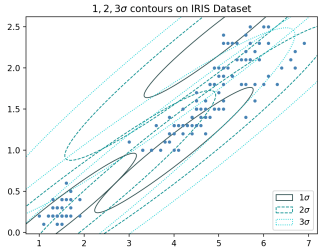
, where $N_k = \Sigma_{n=1}^{N} \gamma_k^{(n)}$. We will perform E-step and M-step interchangeably to optimize the classification of data points.

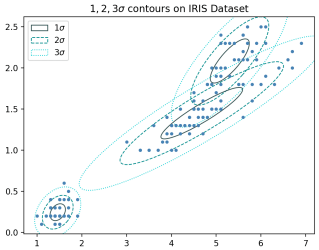
4.2



4.3



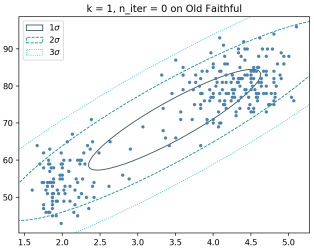


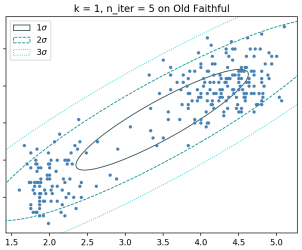


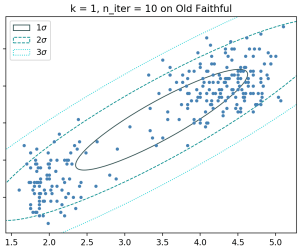
We can tell that all species from the Iris dataset (on two attributes) are nicely clustered within their sigma contours.

4.4

K = 1



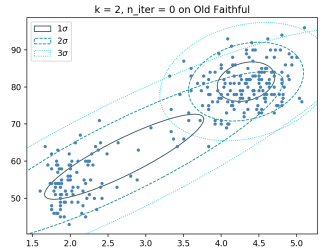


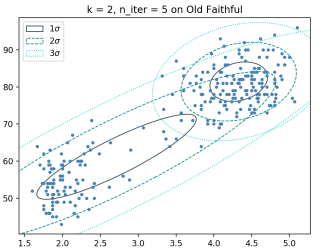


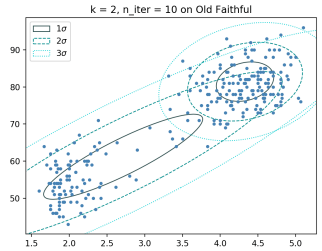
Where the μ, Σ, π for $K = 1$ are respectively:

```
mean: [[ 3.48778309 70.89705882]]
cov: [[[ 1.29793889 13.92641885]
  [ 13.92641885 184.14381488]]]
pi: [1.]

mean: [[ 3.48778309 70.89705882]]
cov: [[[ 1.29793889 13.92641885]
  [ 13.92641885 184.14381488]]]
pi: [1.]

mean: [[ 3.48778309 70.89705882]]
cov: [[[ 1.29793889 13.92641885]
  [ 13.92641885 184.14381488]]]
pi: [1.]
```

$\mathbf{K = 2}$





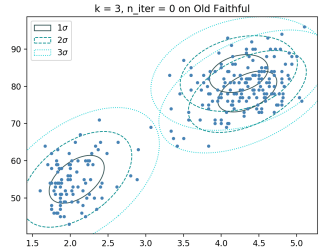


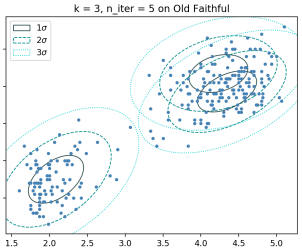
Where the μ, Σ, π for $K = 2$ are respectively:

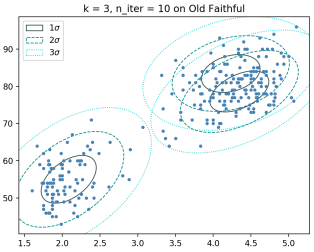
```
mean: [[ 4.32816932 81.43143106]
[ 2.71366246 61.19333632]]
cov: [[[ 0.14492959  0.45361294]
  [  0.45361294 28.18429193]]

[[  1.11020876 10.67014897]
  [ 10.67014897 131.42085961]]]
pi: [0.47947807 0.52052193]

mean: [[ 4.32816932 81.43143106]
[ 2.71366246 61.19333632]]
cov: [[[ 0.14492959  0.45361294]
  [  0.45361294 28.18429193]]
[[  1.11020876 10.67014897]
  [ 10.67014897 131.42085961]]]
pi: [0.47947807 0.52052193]

mean: [[ 4.32816932 81.43143106]
[ 2.71366246 61.19333632]]
cov: [[[ 0.14492959  0.45361294]
  [  0.45361294 28.18429193]]
```

```
[[  1.11020876 10.67014897]
  [ 10.67014897 131.42085961]]]
pi: [0.47947807 0.52052193]
```

K = 3







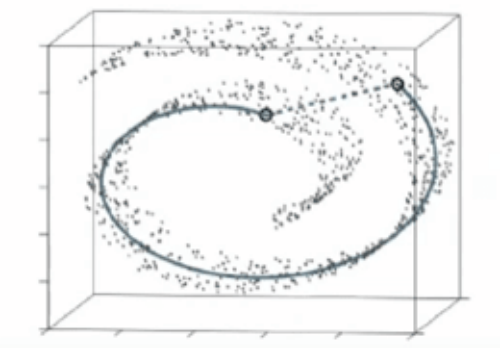
Where the μ, Σ, π for $K = 3$ are respectively:

```
mean: [[ 4.23203926 83.42947492]
[ 2.08519156 55.01970556]
[ 4.34563196 78.61168712]]
cov: [[[ 0.1506054 0.52994426]
  [ 0.52994426 25.80940221]]
[[ 0.13317075 1.1482405 ]
  [ 1.1482405 41.28312871]]
[[ 0.1526638  1.03335337]
  [ 1.03335337 30.27117787]]]
pi: [0.20632691 0.36913676 0.42453633]

mean: [[ 4.23203926 83.42947492]
[ 2.08519156 55.01970556]
[ 4.34563196 78.61168712]]
cov: [[[ 0.1506054 0.52994426]
  [ 0.52994426 25.80940221]]
[[ 0.13317075 1.1482405 ]
  [ 1.1482405 41.28312871]]
[[ 0.1526638  1.03335337]
  [ 1.03335337 30.27117787]]]
pi: [0.20632691 0.36913676 0.42453633]

mean: [[ 4.23203926 83.42947492]
[ 2.08519156 55.01970556]
[ 4.34563196 78.61168712]]
cov: [[[ 0.1506054 0.52994426]
  [ 0.52994426 25.80940221]]
[[ 0.13317075 1.1482405 ]
  [ 1.1482405 41.28312871]]
[[ 0.1526638  1.03335337]
  [ 1.03335337 30.27117787]]]
pi: [0.20632691 0.36913676 0.42453633]
```

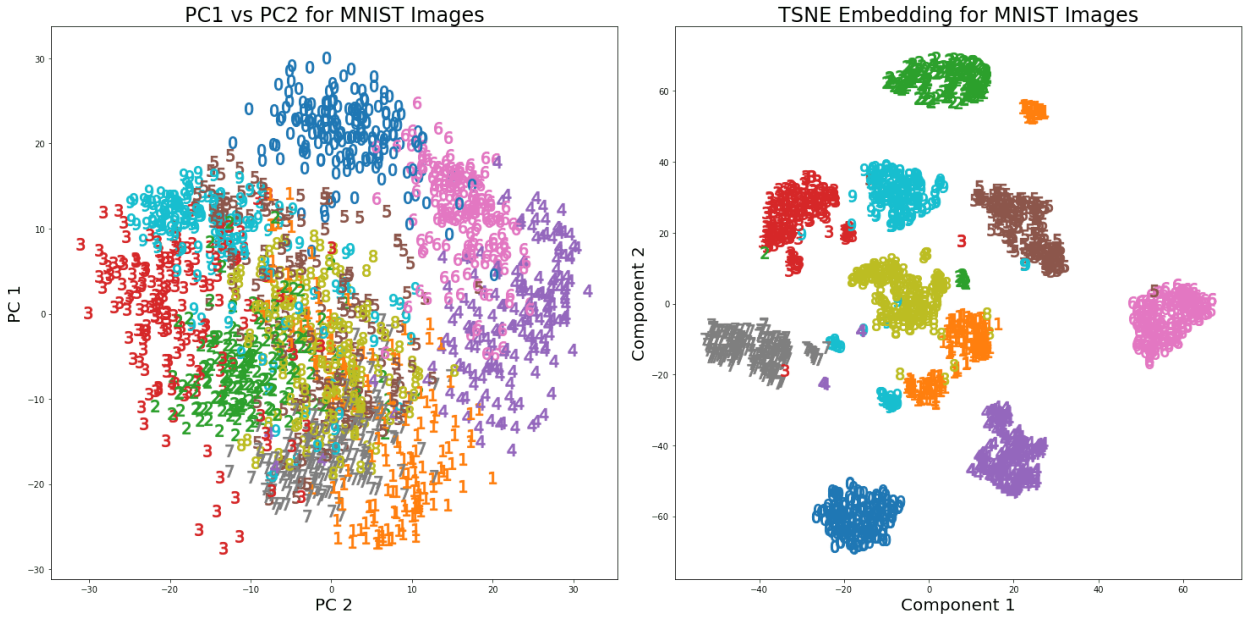
Exploration: t-SNE

Motivation

There are many dimensionality reduction algorithms, as we have learned in this class, PCA is a dimensionality reduction algorithm that works fairly well on linear datasets and it is known to be used in combination with distance-based clustering algorithms. Intuitively, this makes PCA unapplicable of capturing dataset with non-linear structure – such as a manifold "Swiss roll" we showed in class.



Well there are certainly PCA-based algorithms that are designed to handle this problem, e.g., the various avalivable and kernel PCAs, the SNEs and t-SNE in particular attracted my attention as it is a dimensionality reduction algorithm that doesn't wotk with distance-based clustering algorithms, yet yields promising results (ref.):

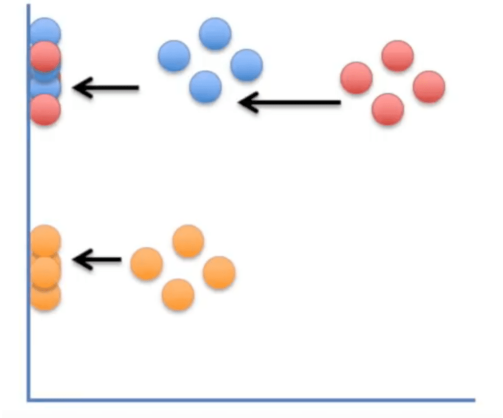


Clearly the t-SNE algorithms shows better clusters in comparison to PCA. So in this exploration I will give a brief walk-through on how t-SNE works, run it on the same dataset I used in **Exercise 3** and interpret the result, then discuss about the advantages between t-SNE and PCA in terms of determinability, locality, usability in combination with clustering algorithms, computational costs,

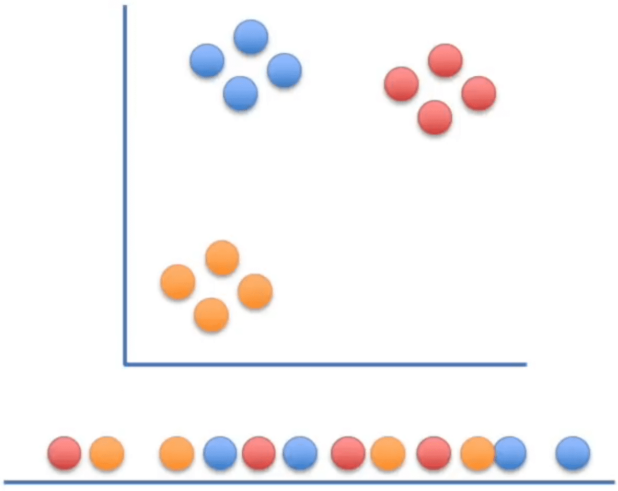
etc.

Mechanism of t-SNE

As we already familiar from learning PCA, any direct projection of high-dimensional datapoints to lower dimensions will likely result in a mishmash:



Intuitively, we would like to move around these projected datapoints, so that there can be a distinguishiable distance between different labels. However, the question is how? Which point should we move? And in which direction? Assume we have the following raw projection of two dimensional data onto one line:



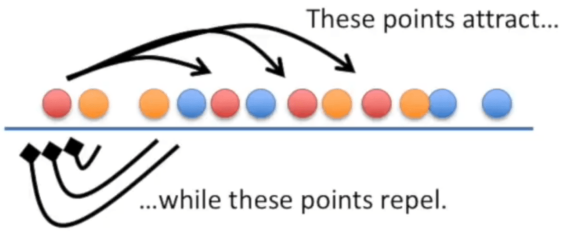
t-SNE runs in an iterative manner, as it will move one point at a time untill it reaches the stopping criteria (which is usually numbers of iterations). W.L.O.G., let's check out how will t-SNE move the leftmost red point (note the following plots are coming from StatQuest with Josh Starmer with a little bit post-process for better representation in written media):

1. First t-SNE will datapoints which is similar to this leftmost red, which are other red datapoints.

2. So naturally, we should move this leftmost red datapoint closer to other red datapoints – which is to its right.

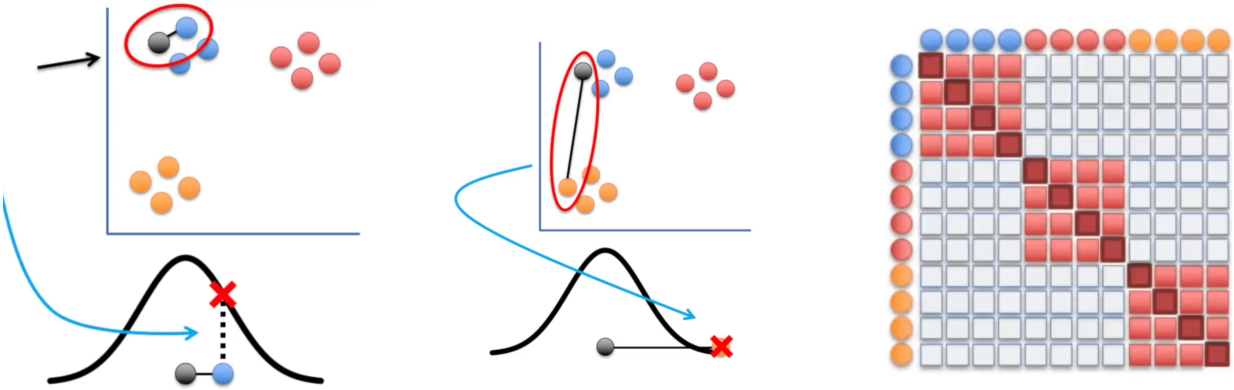
3. Similarily, since this leftmost red is far away from the orange and blue datapoints in its original space, it will like to move away from them – which is to its left.

So we have the following "tractions":



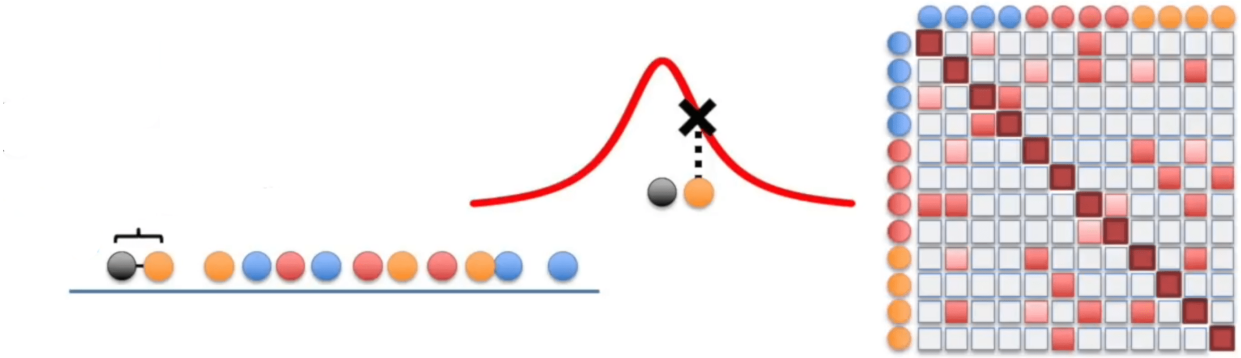
Let's assume this time the traction from the red points are stronger, so the leftmost red moves to its right for a bit. Then we do this evaluation to each and every points, over and over, untill the stopping criteria is reached, and we may have some nice clusters on this one dimensionl line.

So how does t-SNE determine the similarity between datapoints? It does it, again, in an iterative manner. t-SNE will locate a point first (in this case, it will be the **black**black point from the **blue**blue cluster), and model its relative distances to all other datapoints under a normal distribution curve. So datapoints with same or similar lable, will have a higher probablity score, as it is closer to the center of the normal curve; yet datapoints that is vast unlike to eachother will have a lower probablity score, as it will be at the tails of normal curve.



t-SNE will then "scale" these probablity scores to unsure the total probably will be added to 1. In other words, assume datapoint A has relative probablity scores of $p(A_x), p(A_y), \ldots p(A_n)$ to datapoint $X, Y, \ldots, N$, we will have $p(A'_i) = \frac{p(A_i)}{\sum_j^n p(A_j)}$. Following this procedure, t-SNE will obtain the relative normalized probablity score between every two datapoints, and thus fill this similarity matrix (note that it is common that $p(A'_b) \neq p(B'_a)$ as datapoints A, B are under two different normal distributions. So t-SNE simple take the average out of these two) – and the end result will be the rightmost matrix above; where red entries represent higher similarity.

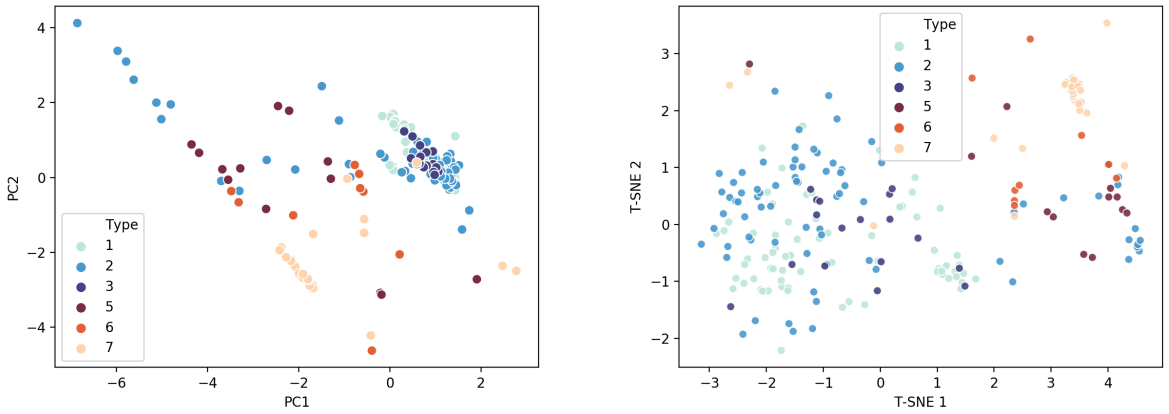
Now we repeat such matrix filling operation with respect to the one-dimensional line, as we will fill in another matrix where the noralized probablity score will be determined by the relative distance between one point to another on this one-dimensional line. However, instead of modeling its with normal distribution, it will use t-distribution (and therefore, the "t" of "t-SNE") – this is because t-distribution curve has a lower center and higher tails in comparison to normal distribution, thus encourges more seperation of datapoints.



With the two similarity matrices, t-SNE may then try to "convert" the second matrix more like the first one – and the steps of this conversion will determine the moving direction of each datapoints.

t-SNE on UCI ML Glass Classification Dataset

I have implemented t-SNE on the UCI ML Glass Classification Dataset I used in **Exercise 3** and got the following result:



PCA t-SNE

Although we don't have a clean clustered result like t-SNE on `MINIST` did. It is observable that we have better clustering result as the `Type 5, Type 6, Type 7` glasses are better gathered. Although we still can't effectively distinguish `Type 1` from `Type 2` glasses, they are certainly more scattered.

(please refer to `A4/code/exploration` for code implementation.)

t-SNE vs PCA

Although t-SNE offers better clustering visualization on many popular tasks, it also has its pros and cons. The main pros, are that t-SNE can certainly able to capture manifold structure, and it is better on visualization due to the procedure of t-SNE encourages different datapoints to "move away from eachother," where PCA only cares about the maximum of sum of squared variance.

t-SNE also has many cons, and one of the most significant one is probably its incompability with distanced-based clustering algorithms. As we are directly moving the datapoints around in a lower-dimensional space, these datapoints lose their original distance relationship and thus become incapable of being a direct feed of distance-based clustering algorithms. Also on this note, t-SNE is not capable of learning a high-dimensional space to a lower-dimensions, but rather just lower the dimensions of known data – so if an unseen data joins, it can't directly project it onto a lower-dimensionl space.

Further t-SNE is a non-determistic algorithm as it has a turnable paramater *perplexity*, which is a loose balancer between the local and global aspect of given data, so you might have to run multiple trials to achive the optimal result. This is especially a problem as t-SNE is a lot more computational intense than PCA due to 1) it has to calculate the similarity scores that is proporsional to the combinations of datapoints, but not simply project all datapoints on to a line; 2) it has to do it iteraively over and over.

The effect of this parameter is complicate, but according to many articles it works best if it is a rough estimation of cluster size. This implies t-SNE is probably not so suitable to unbalanced dataset (like the UCI ML Glass Classification I used). PCA, on the other hand, is parameter-free, but I don't consider this to be such a major advantage as the countless kernel PCA algorithms has much more turnable parameters.

A more through discussion between t-SNE and PCA can be found here and here.