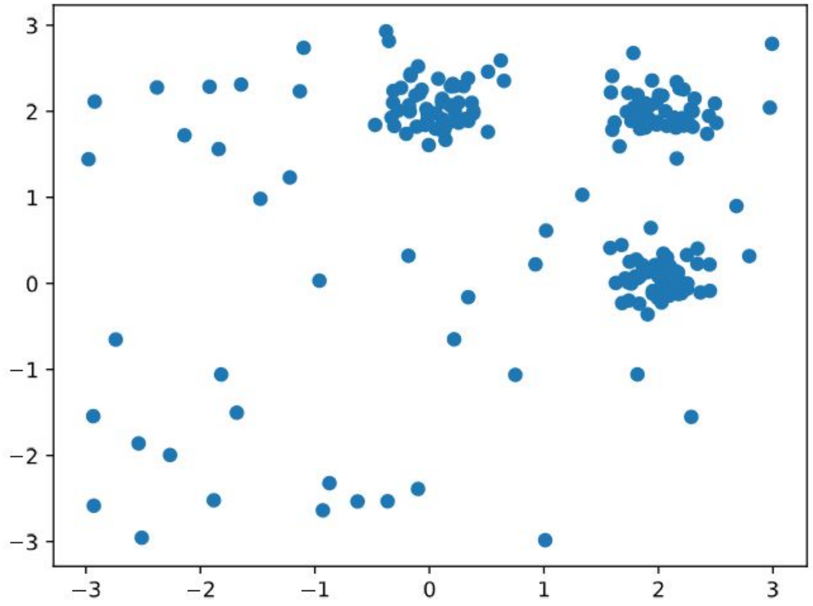
Pattern Recognition
Project 3

Clustering, dimensionality reduction and non-monotonous neurons

Task 3.1:
Fun with k-means clustering

Scatter Plot of Data:



Lloyd's algorithm :

Steps:

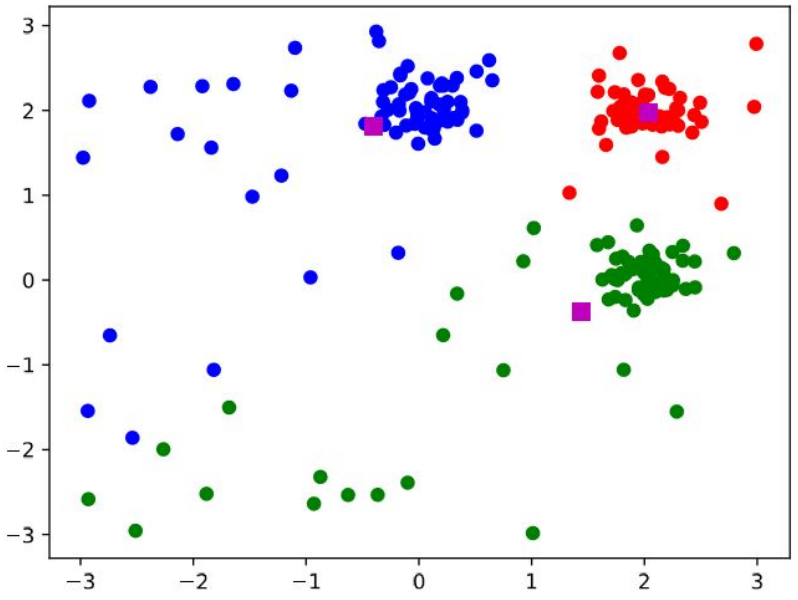
set t = 0 and initialize µt 1,µt 2,...,µt k repeat

until convergence

$$C_i^t = \left\{ x \in X \;\middle|\; \|x - \mu_i^t\|^2 \leqslant \|x - \mu_l^t\|^2 \right\}$$

up

$$\mu_i^{t+1} = \frac{1}{|C_i^t|} \sum_{x \in C_i^t} x$$ means



Average Run Time: 0.203 s

Hartigan’s algorithm :

Steps:

for all xj ∈x1,...,xn,

randomly assign xj to a cluster Ci for all Ci ∈C1,...,Ck

compute µi repeat until converged

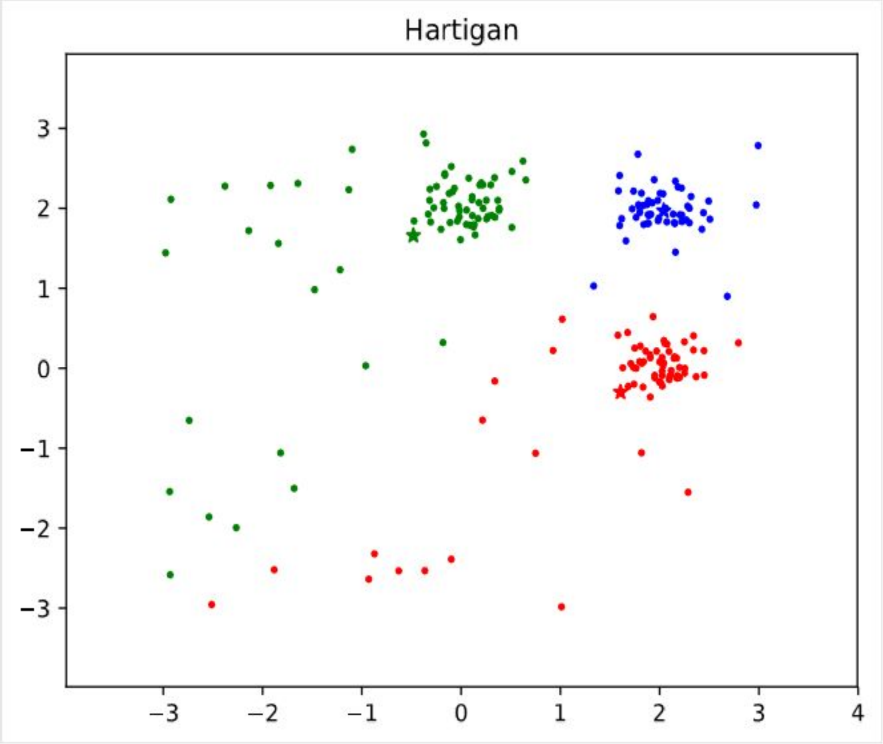
determine Ci = C(xj) remove xj from Ci and recompute µi

determine Cw = argminCl EC1,...,Cl ∪{xj},...,Ck

if Cw != Ci, then converged ←False assign xj to Cw

recompute µw

Average Time Taken:0.418s



McQueen's algorithm :

determine *winner centroid*

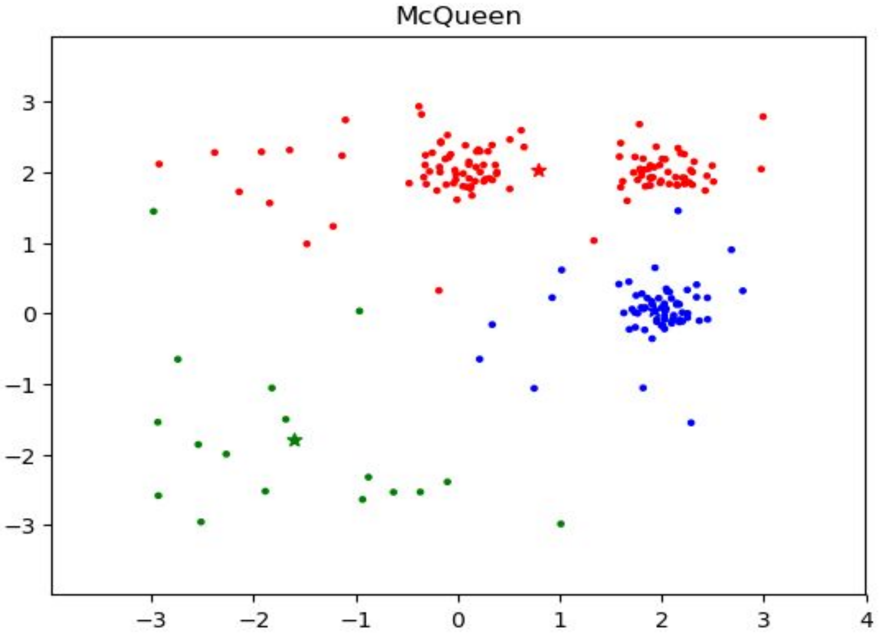
$$\boldsymbol{\mu}_w = \underset{i}{\operatorname{argmin}} \left\| \boldsymbol{x}_j - \boldsymbol{\mu}_i \right\|^2$$

update cluster size and centroid

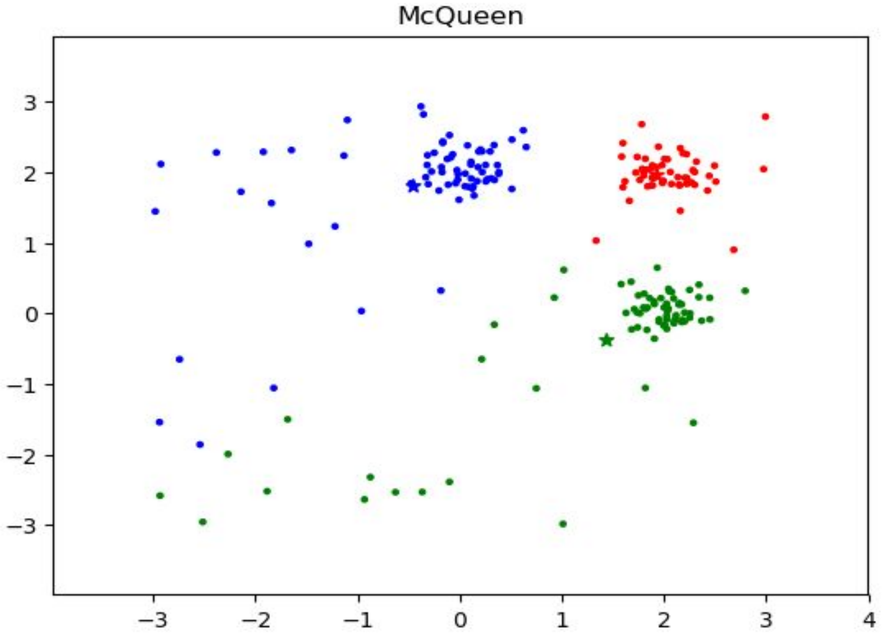
$$n_w \leftarrow n_w + 1$$

$$\boldsymbol{\mu}_w \leftarrow \boldsymbol{\mu}_w + \frac{1}{n_w} \left[\boldsymbol{x}_j - \boldsymbol{\mu}_w \right]$$

McQueen’s algorithm :



Most common case



Rare case

Average Time Taken: 0.63µs

Task 3.2:
Spectral Clustering

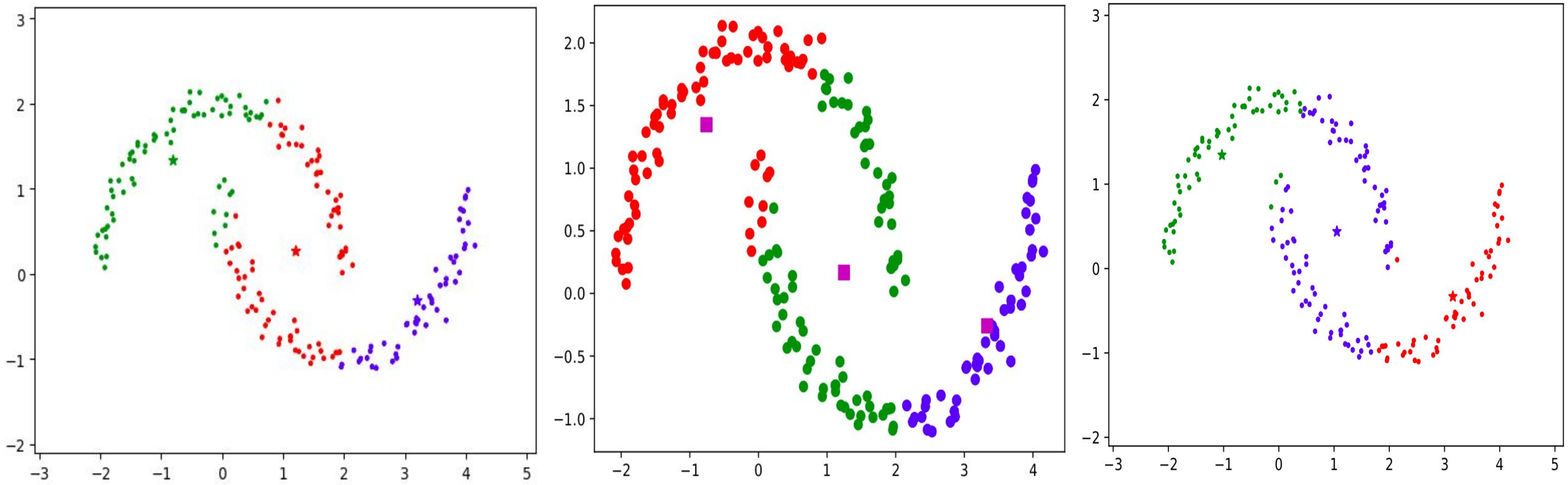
Spectral Clustering

Clustering method with roots in Graph Theory. The steps are as follows:

1. Convert set of points X to a matrix ***S***, **simulating an adjacency matrix** by using a similarity measure.
2. Calculate a (diagonal) degree matrix *D* by summing up similarities for each node.
3. Calculate Normalized Graph Laplacian Matrix ***L = D - S*** and calculate the eigen values and vectors of ***L***.
4. Calculate the Fiedler Vector which is the one with the second smallest eigen value and use it for clustering.

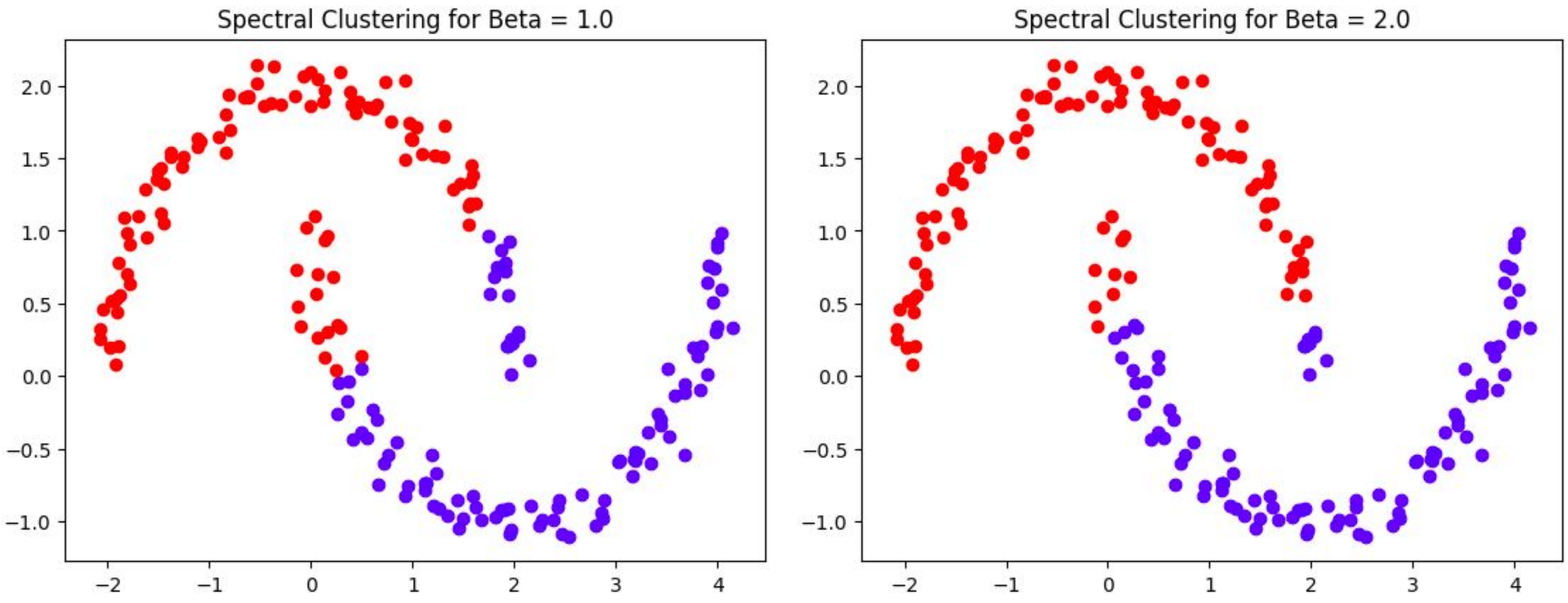
FV gives the approximation of the minimum graph cut needed to separate the “graph” into 2 CCs.

Failure of K-Means

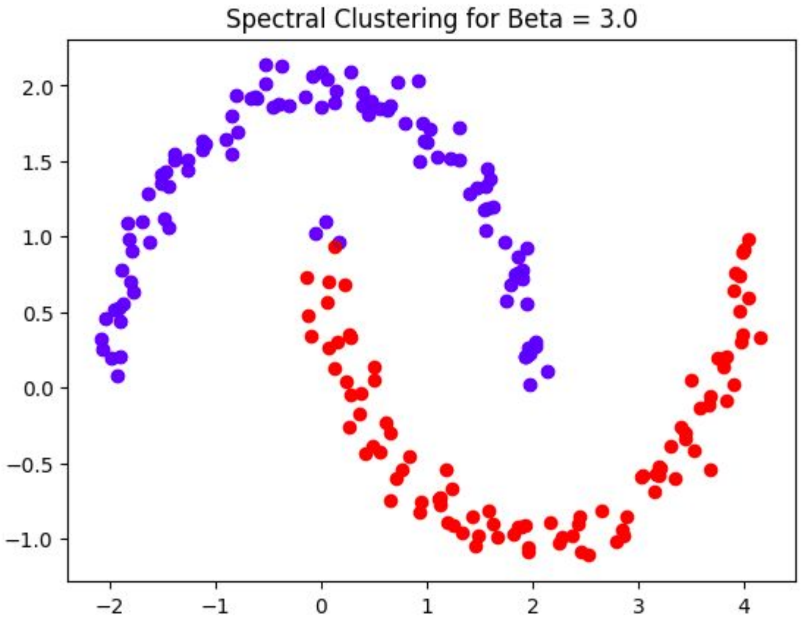


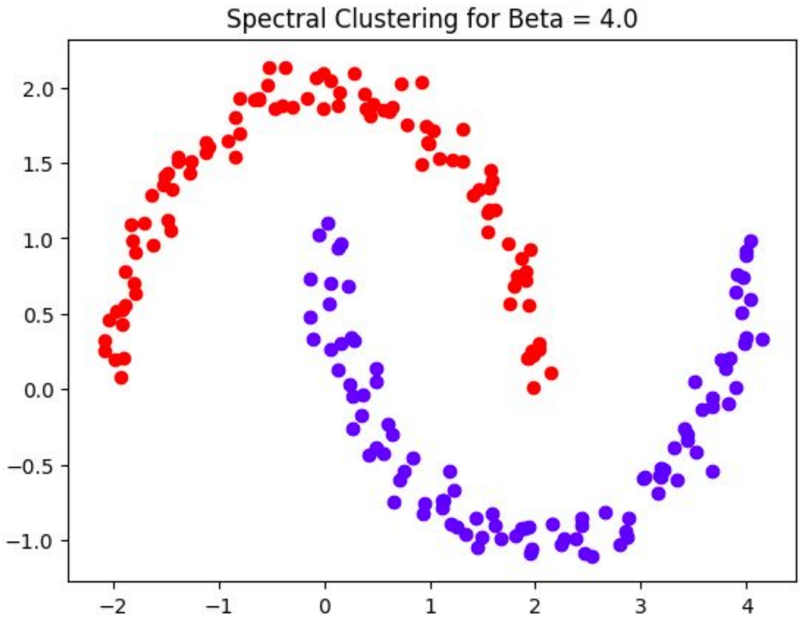
McQueen’s, Lloyd and Hartigan’s K-Means Clustering Algorithms

Spectral Clustering



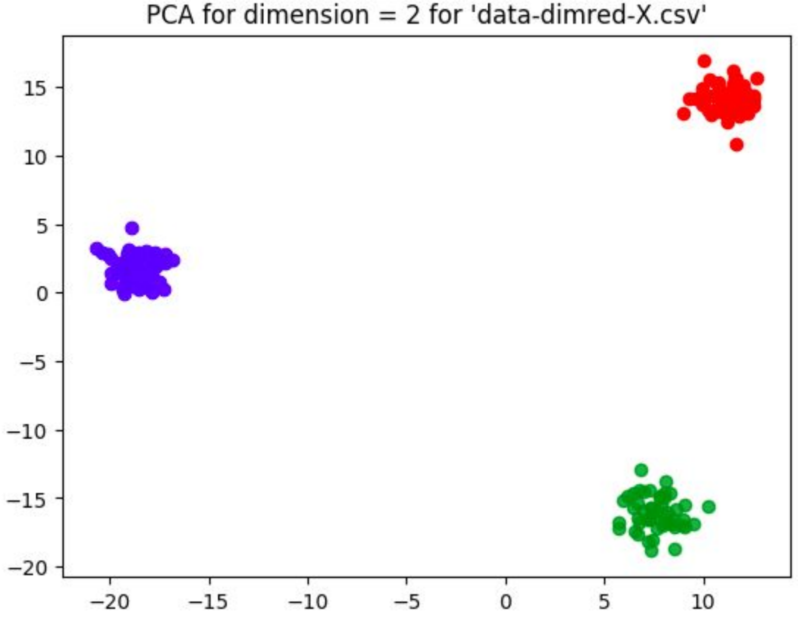
Spectral Clustering

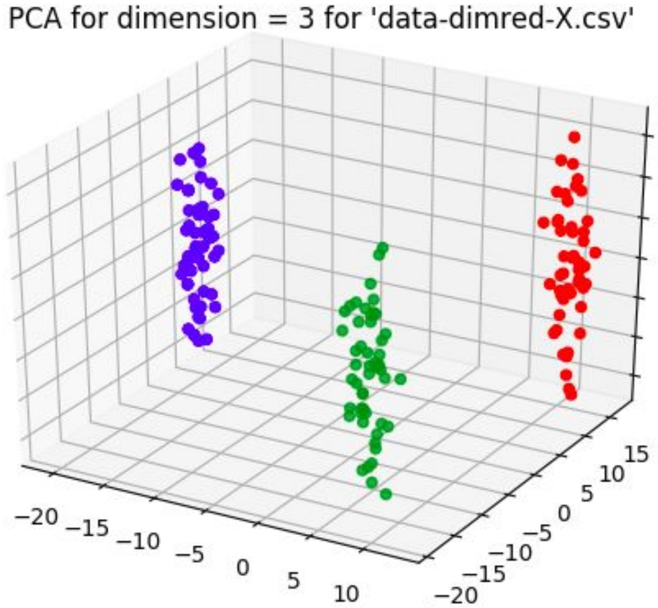




Task 3.3:
Dimension Reduction

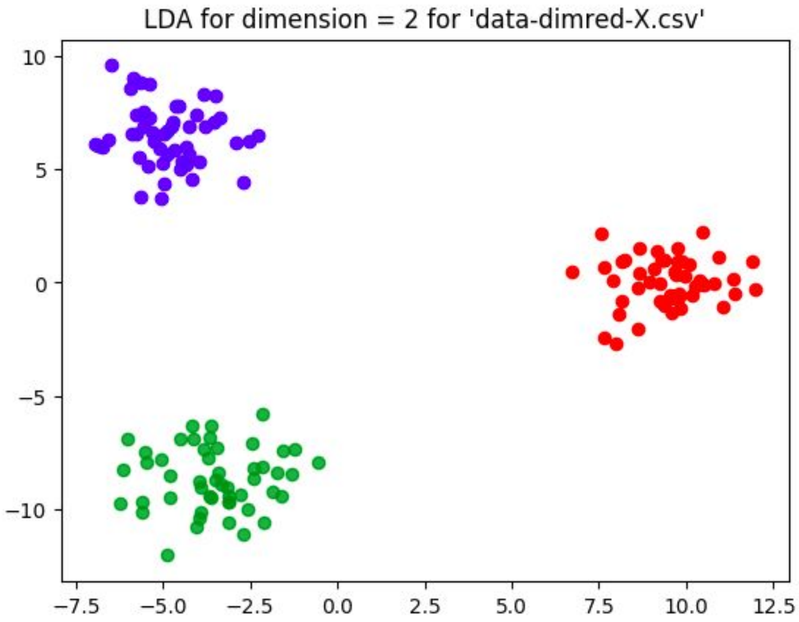
Principal Components Analysis (PCA)

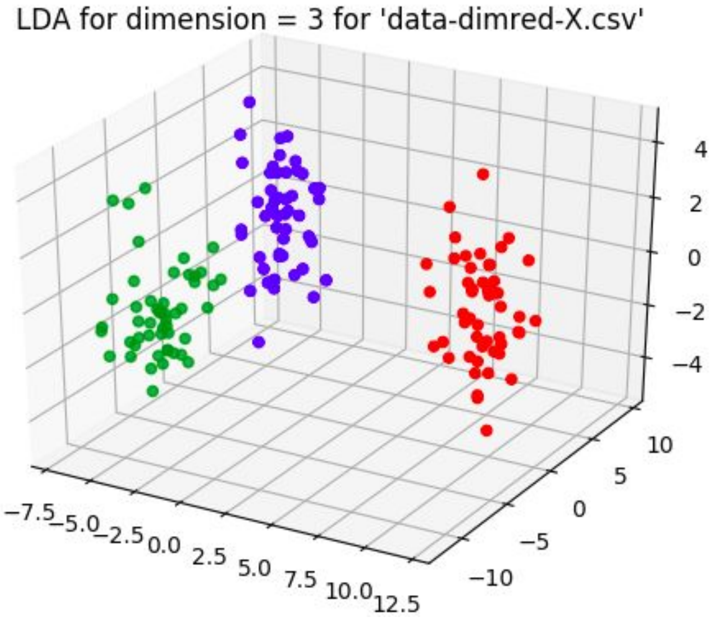




PCA: Projection using the top eigenvectors of C

Linear Discriminant Analysis (LDA)





LDA: Projection using the top eigenvectors of $S_W^{-1}S_B$

Task 3.4:
Non-monotonous neurons

Problem Statement:

- Train a non-monotonous neuron on a XOR problem, with an activation function *f* and loss function *E* using Gradient Descend:

$$y = f(z)$$

$$f(z) = 2.exp(-\frac{1}{2}z^2) - 1$$

$$z = w^T x - \theta$$

$$E = \frac{1}{2}\sum_{i=1}^{n}(y(x_i) - y_i)^2$$

Equations for Computing Gradients

$$\frac{\partial E}{\partial z} = (\widehat{Y} - Y) \odot (\widehat{Y} + 1) \odot (-Z)$$

$$\frac{\partial E}{\partial W} = \frac{1}{m} X \frac{\partial E}{\partial z}^T$$

$$\frac{\partial E}{\partial \theta} = -\frac{1}{m} \sum_{i=1}^{m} \frac{\partial E}{\partial z}$$

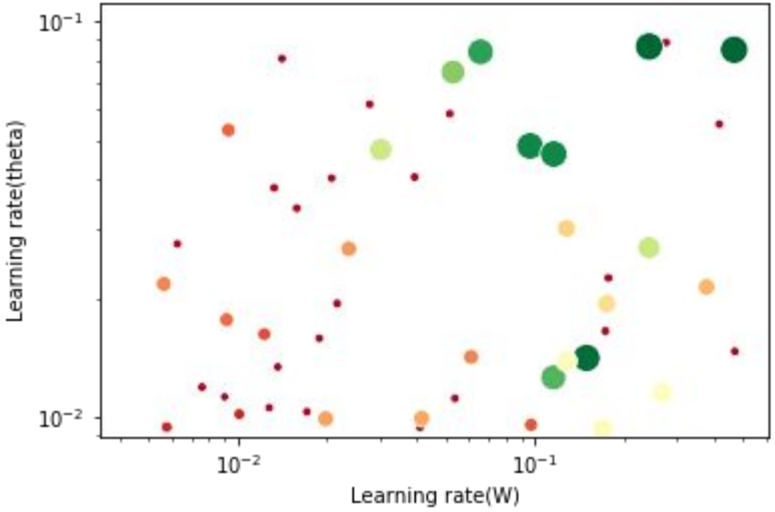
Numpy Implementation

```
for epoch in range(num_epochs):

    # forward propagation
    Z = np.dot(W.T,X) - theta
    Y_hat = activation(Z)
    loss = loss_function(Y_hat,Y)

    #backward propagation
    dE_dZ = (Y_hat-Y)*(Y_hat+1)*-Z
    dE_dW = (1/m)*np.dot(X,dE_dZ.T)
    dE_dtheta = -(1/m)*np.sum(dE_dZ)

    # gradient descent
    W = W - lr_W*dE_dW
    theta = theta - lr_theta*dE_dtheta
```

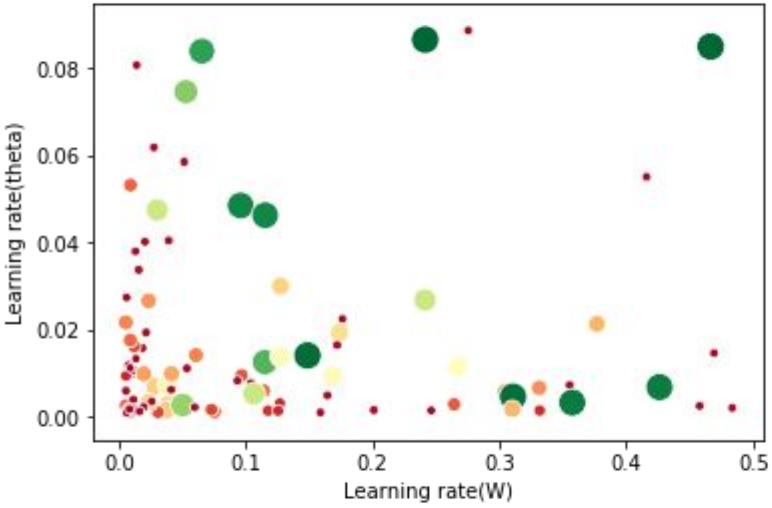
Hyper parameter Tuning

- Learning rate for the parameters were searched in a random grid on a logarithmic scale.
- η_w was tuned in the range (0.005,0.5)
- For η_θ, exploration was done in the range (0.001,0.1)

Hyper parameter Tuning

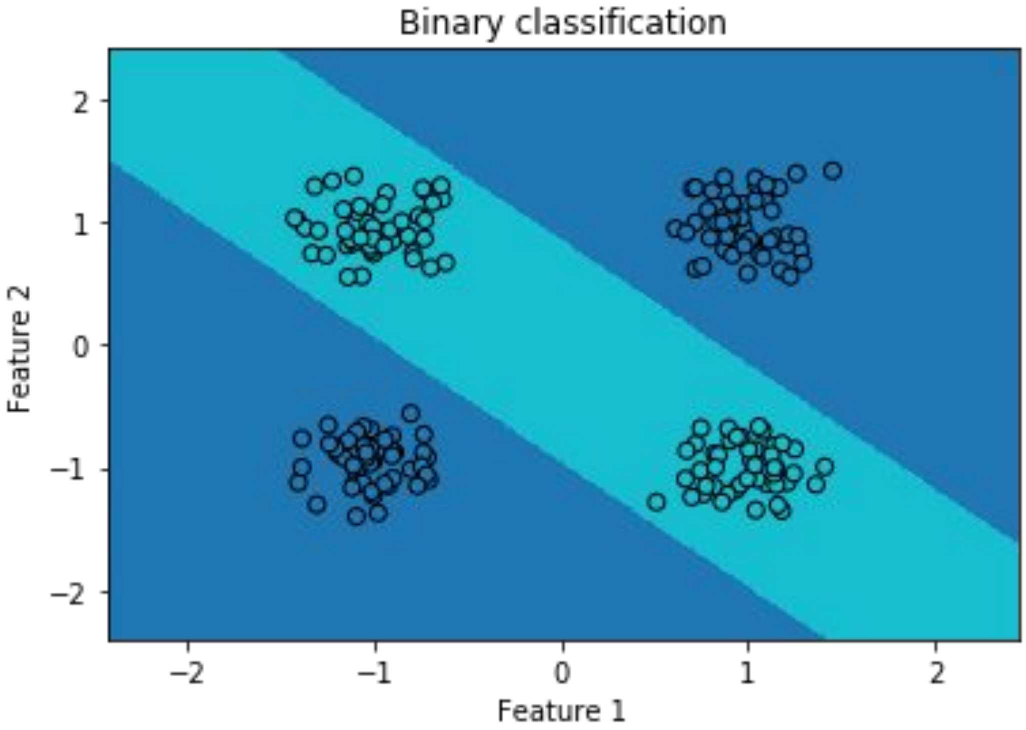


On Logarithmic Scale



On Linear Scale

Results

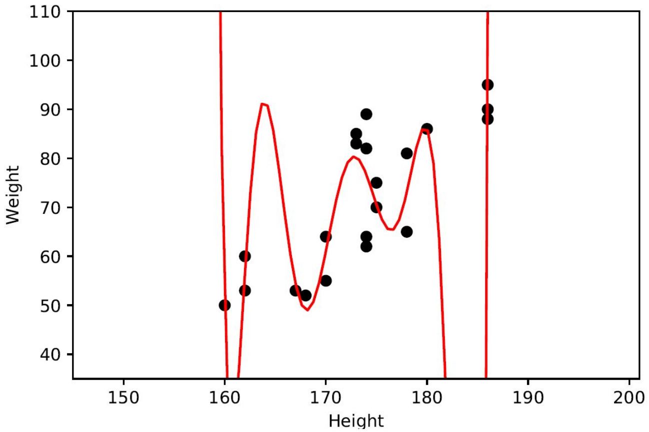


Task 3.5:
Exploring Numerical Instabilities

Problem Statement:

- To revisit the Least Squares regressions for prediction at higher dimensions.

Method: Polyfit



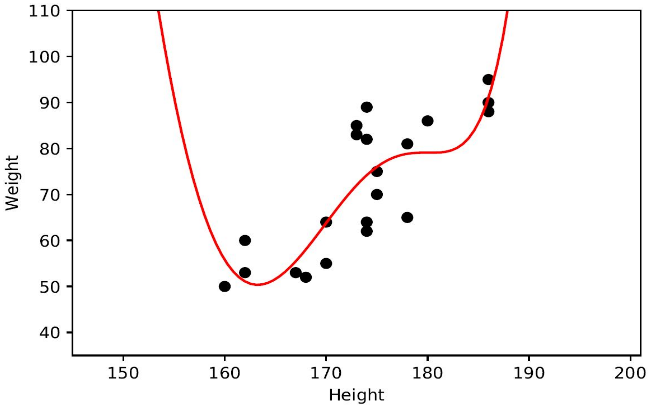
Warning:RankWarning: The fit may be poorly conditioned

c = poly.polyfit(hgt, wgt, 10)

Fit using inbuilt Polyfit.

This implies that the best fit might not be well-defined due to numerical errors. The results may be improved by lowering the polynomial degree or by transforming the data.

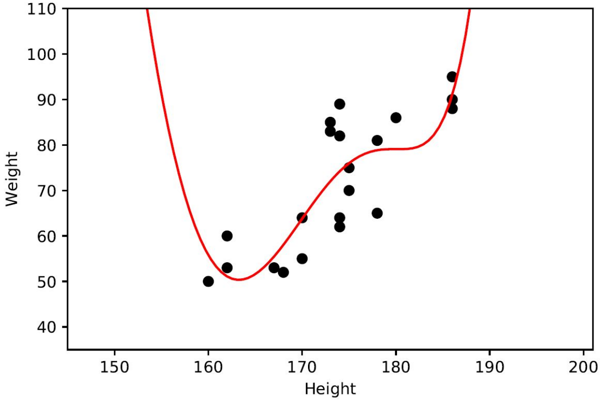
Method: Vandermonde matrix and pinv



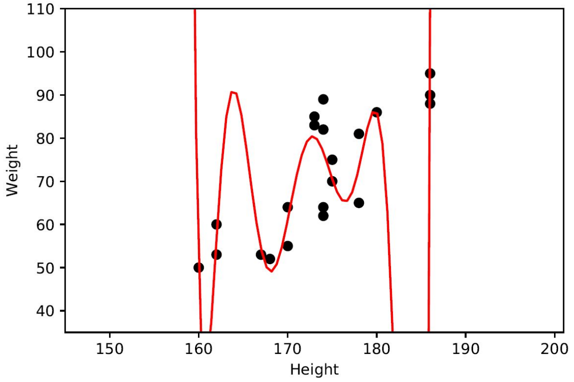
Standard fitting using inverse of the vandermonde matrix

Improper fit.

Method: Vandermonde matrix and lstsq



Method: transformed Vandermonde - pinv



Standard fitting on Vandermonde Matrix using least squares

Improper fit.

Fitting on Transformed Vandermonde Matrix using pinv

pinv(X/100) gives the correct pinv(..) values while avoiding numerical errors

Thank You.