### Homework 3

### PageRank & node2vec

Name: Eduardo Wang Zheng

E-mail: eduardo@sjtu.edu.cn

1. Experiments
   1. Compute the PageRank vector, using the existing NetworkX graph object G
      1. Algorithm

1) Use **G.nodes()** method to get a list of nodes in the graph

2) Use **len()** method to get the number of the elements in the node list (the number of nodes in the graph) and store it into **graph\_size**

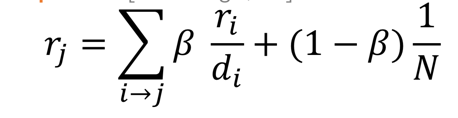
3) If **graph\_size** equals to 0, return

4) Let **init\_value** be 1/**graph\_size**

5) Use **dict.fromkeys** method to create a dictionary **pagerank.** The key is node ID, the value is **init\_value**

In this way, we generate the initial PageRank vector

6) Iterate the PageRank vector using PageRank equation:



In other words, we need to construct a double loop to perform the iteration.

In the outside loop, we traverse every node. And in the inner loop, we traverse every neighbor of the current node and update the **rank** of the current node (using PageRank equation).

I set up two iteration termination conditions:

i. The number of iterations is beyond **max\_iterations** (a constant)

ii. Difference between this iteration and last iteration is less than **min\_delta** (a constant)

Note I set the **damping\_factor ()** as 0.85, which means the teleport probability is 0.15 (that is, each step is about to jump to some random node with probability 0.15).

* + 1. Evaluation

The iteration converges after 27 rounds, and the convergence results

are stored in the file **result.txt**

Appendix A shows the whole process of the iteration (take 10 nodes

for convenience).

* 1. Write a function that takes as input a node id **n** and returns a list containing the **cosine similarity** between the node2vec vector of the input node **n** and all the nodes in the given **barbell\_1**
     1. Algorithm

Main function:

1) Construct Node2Vec model **n2v** (using four hyperparameters p,q,d,w).

2) Use **n2v.fit()** method to train the model

3) For each node in **barbell\_1,** use **n2v.predict()** method to get the node2vec embedding for the node.

4) Call function **f** and store the result into a list **test**

5) Output the elements in **test**

Function **f**

1) Get the specific embedding using **node\_id** and compute the square of the embedding vector’s module length, storing it into **a**

2) For every node2vec embedding for the node of the given **barbell\_1:**

**i.** compute the square of the embedding vector’s module length, storing it into **b**

**ii.** multiply the current embedding vector by each component of the specific embedding vector, storing it into **tmp**

**iii.** Let **result** equals to **tmp/math.sqrt(a)\*math.sqrt(b)**

**iv.** Store the result into a list **res**

3) Return **res**

* + 1. Evaluation

The output list containing the **cosine similarity** between the

node2vec vector of the input node n and all the nodes in the given

**barbell\_1** is stored in the file **task2.1.txt**

* 1. Learn the node2vec embeddings for the nodes of the given **barbell\_2** and tune the hyperparameters to get two distinct clusters in the scatter plot
     1. Algorithm

1) Construct Node2Vec model **n2v** (using four hyperparameters p,q,d,w).

2) Use **n2v.fit()** method to train the model

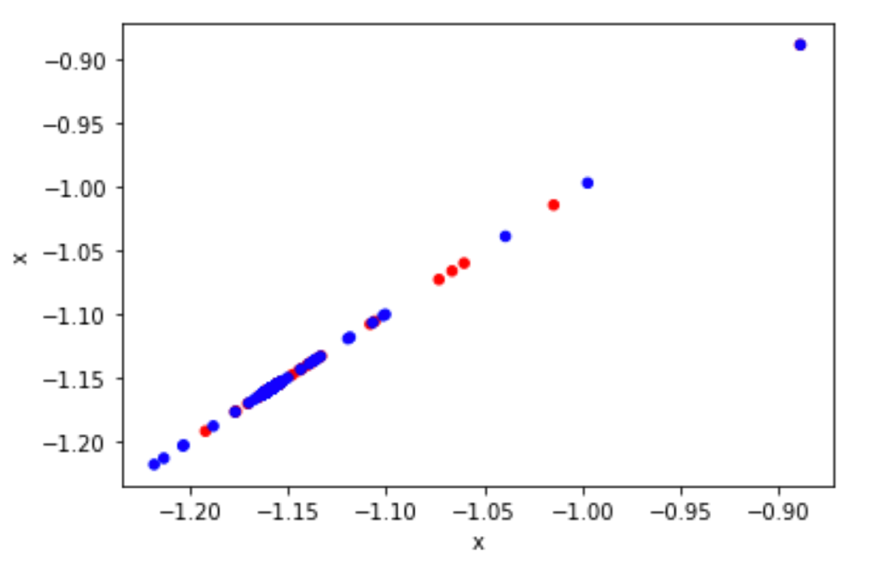
3) For each node in **barbell\_2,** use **n2v.predict()** method to get the node2vec embedding for the node.

4) Create pandas dataframe from the list of node embeddings and draw the scatter plot

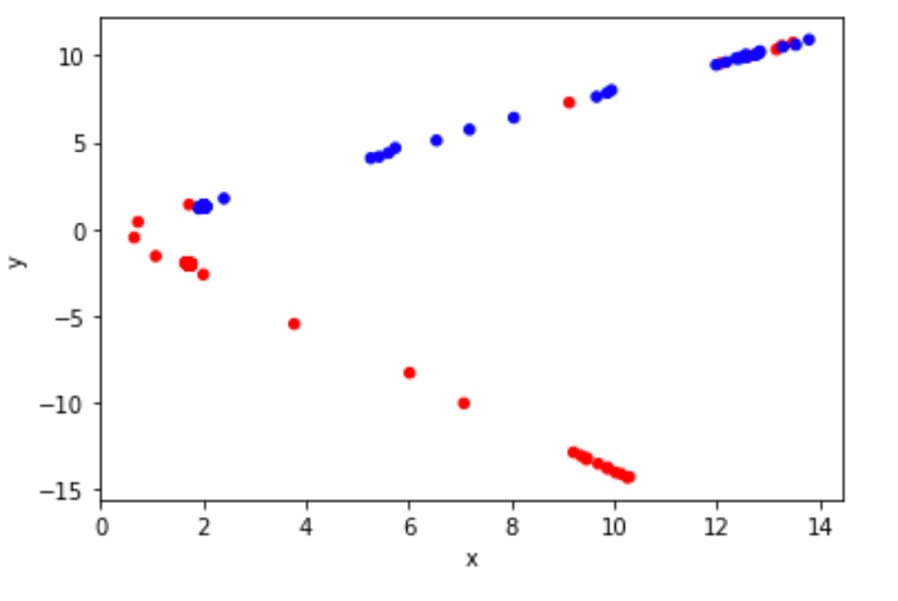
5) Tune the hyperparameters and go to (1)

* + 1. The effect of adjusting **d**

**d** determines thedimensionality of the embedding vectors. From the following figures, we can find that d=2 has better segmentation effect of the clusters than d=1 (other hyperparameters are same).



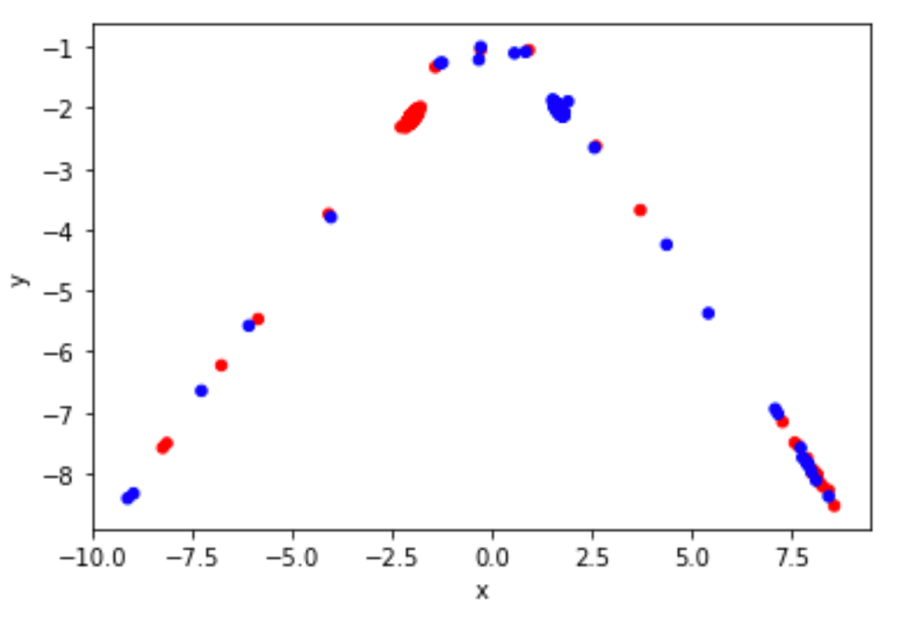
d=1



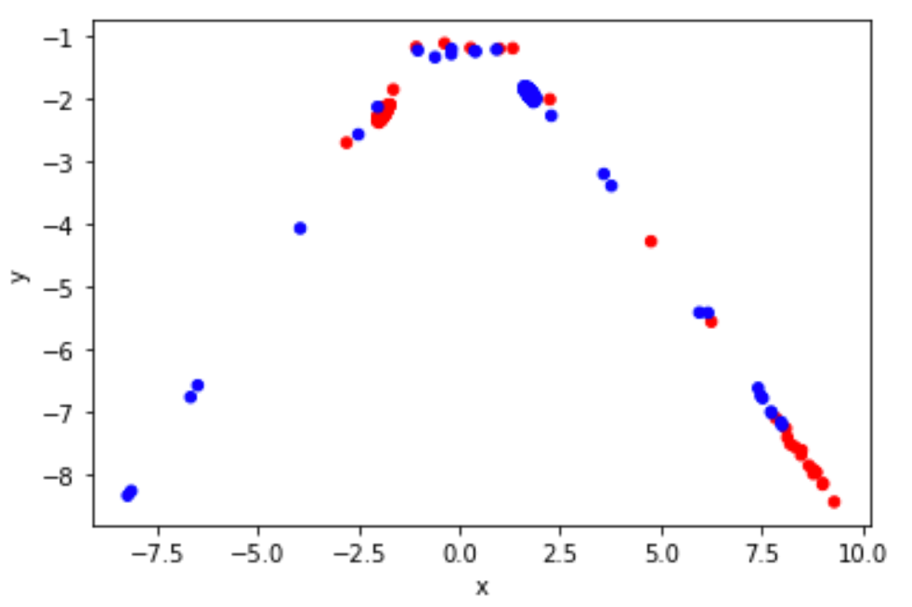
d=2

* + 1. The effect of adjusting **p**

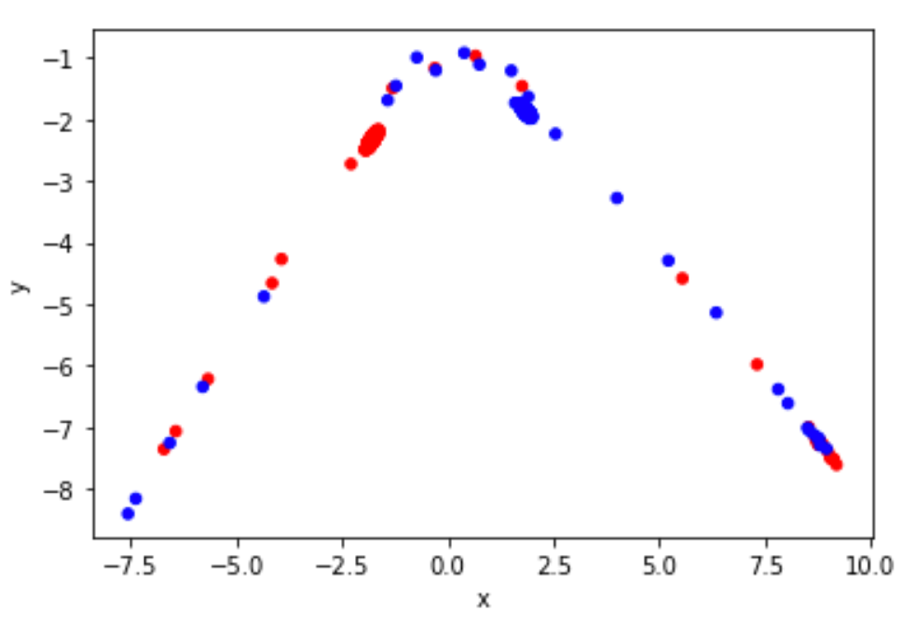
**p** determines theratio between BFS and return to the previous node. From the following figures, we can find that p=0.8 or 1.2 slightly improve segmentation effect of the clusters compare with p=1 (other hyperparameters are same).



P=0.8



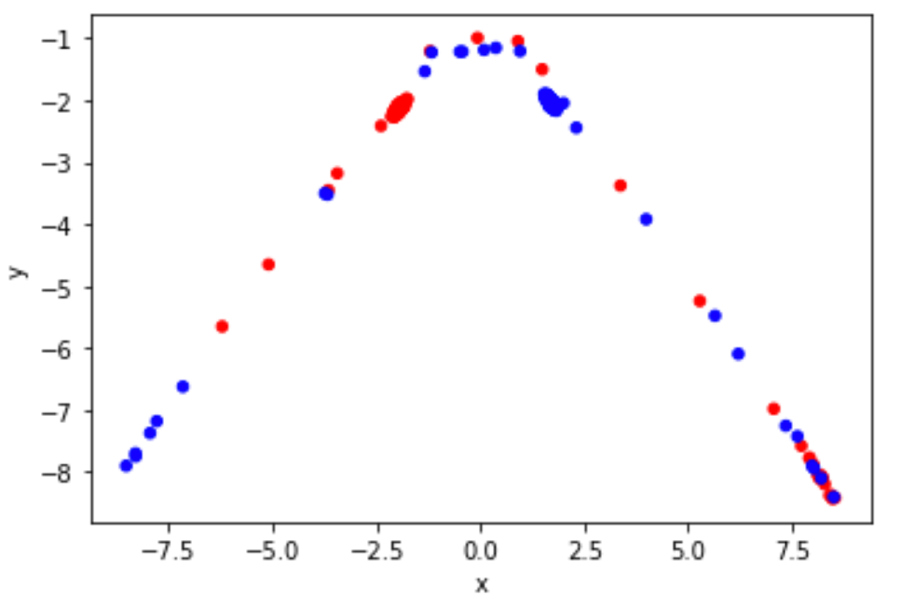
P=1



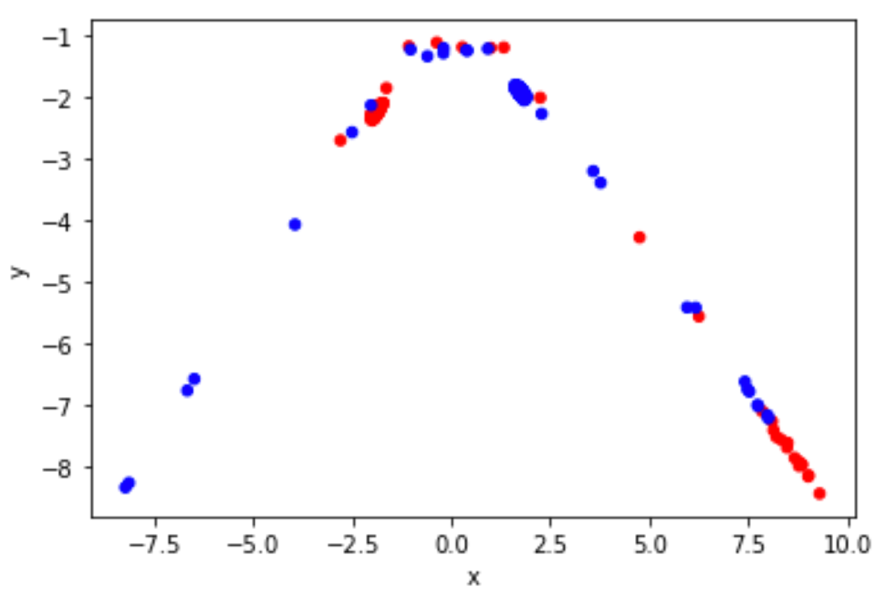
P=1.2

* + 1. The effect of adjusting **q**

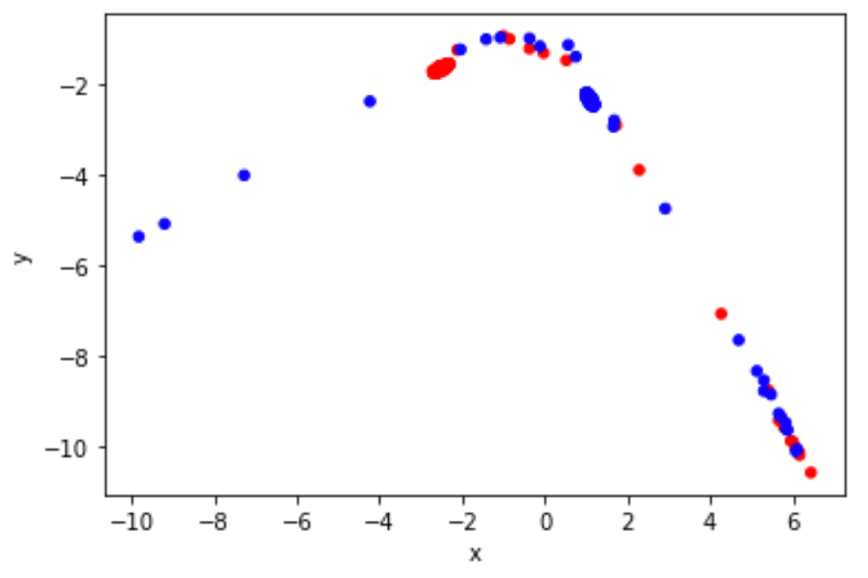
**q** determines theratio between BFS and DFS. From the following figures, we can find that q=0.8 or 1.2 slightly improve segmentation effect of the clusters compare with q=1 (other hyperparameters are same).



q=0.8



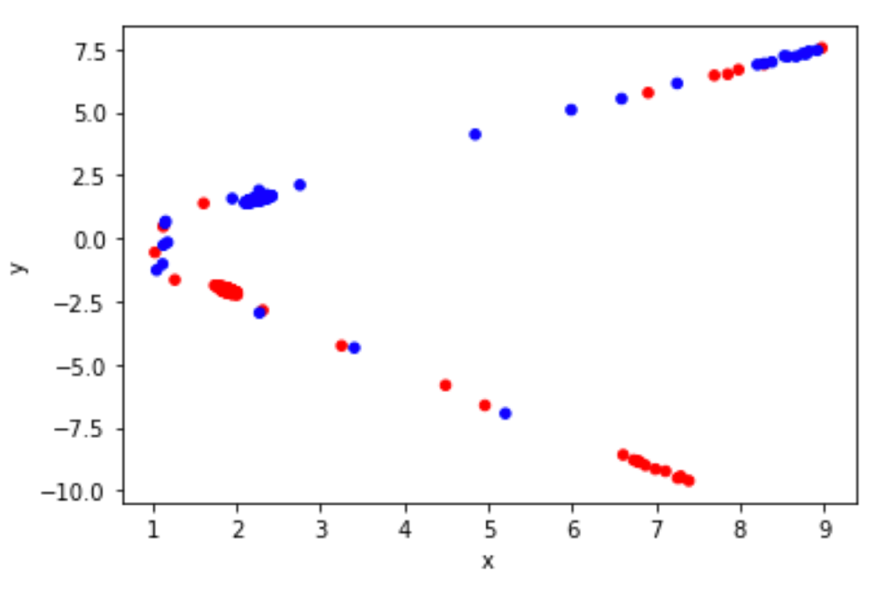
q=1



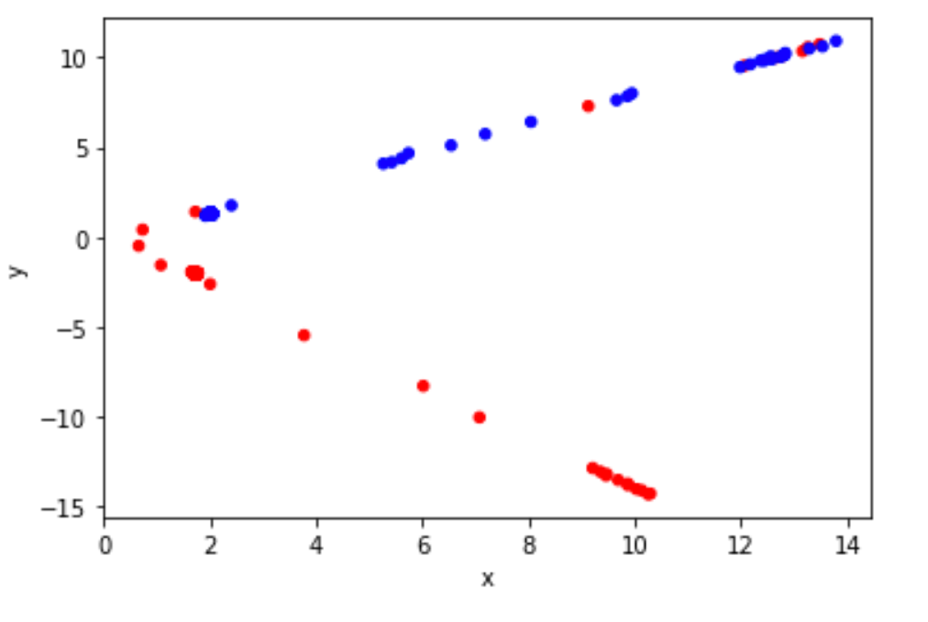
q=1.2

* + 1. The effect of adjusting **w**

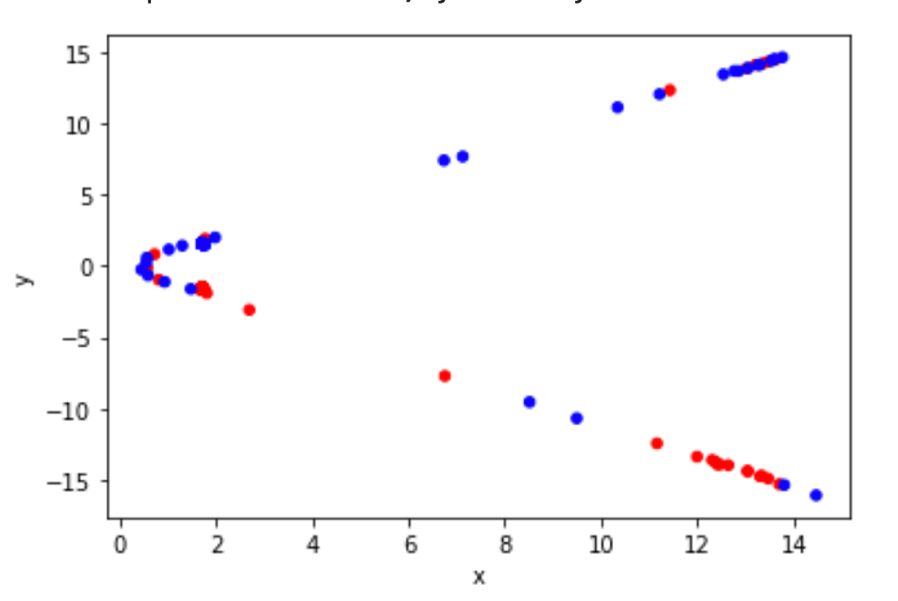
**w** determines length of each truncated random walk. From the following figures, we can find when w is small, segmentation effects of the clusters are almost the same, however, when we hugely increase w, segmentation effects of the clusters is improved significantly (other hyperparameters are same).



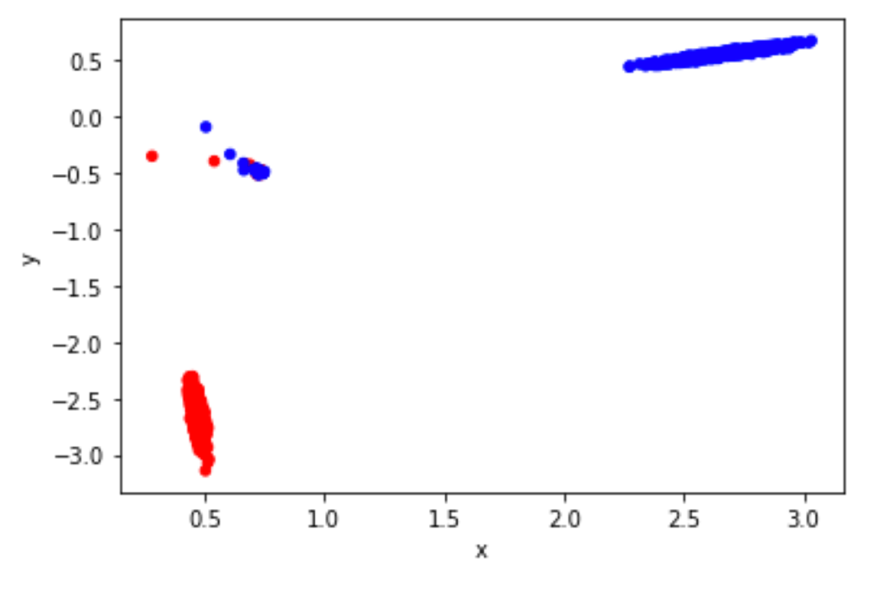
W=5



W=10



W=15

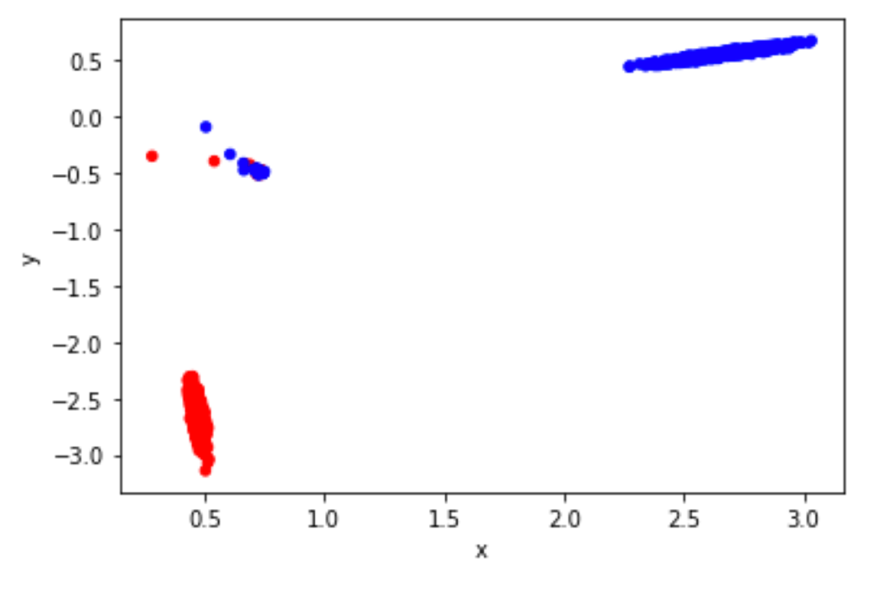


W=50

* + 1. Conclusion

I think w has the largest influence on the segmentation effects of the clusters among four hyperparameters p,q,d,w.

When setting p=1, q=1, d=2, w=50, the best result I currently get is as follows:



Appendix A

