a)

Sigmoid function

|  |  |
| --- | --- |
| # of hidden layer | Class error |
| 32 | 70 |
| 64 | 32 |
| 128 | 14 |

Hyperbolic function

|  |  |
| --- | --- |
| # of hidden layer | Class error |
| 32 | 65 |
| 64 | 32 |
| 128 | 10 |

Relu function

|  |  |
| --- | --- |
| # of hidden layer | Class error |
| 32 | 64 |
| 64 | 29 |
| 128 | 8 |

1. The more neuron, the more precise of the result. My computer is not able to bear more neurons, so I haven't reach the trade off point between high-accuracy and overfitting.

2. In this specific case, relu function seems to be the most efficient activation function, then hyperbolic, and then sigmoid. And relu is computationally cheaper than the others.

b)

SVM reaches 95% accuracy.

c)

Logistic regression also reaches 95% accuracy.

d)

Sigmoid function

|  |  |
| --- | --- |
| # of hidden layer | Class error |
| 32 | 15 |
| 64 | 19 |
| 128 | 27 |

Hyperbolic function

|  |  |
| --- | --- |
| # of hidden layer | Class error |
| 32 | 109 |
| 64 | 64 |
| 128 | 40 |

Relu function

|  |  |
| --- | --- |
| # of hidden layer | Class error |
| 32 | 64 |
| 64 | 29 |
| 128 | 8 |

SVM reaches 92% accuracy.

Logistic regression reaches 97.14% accuracy.

Summary:

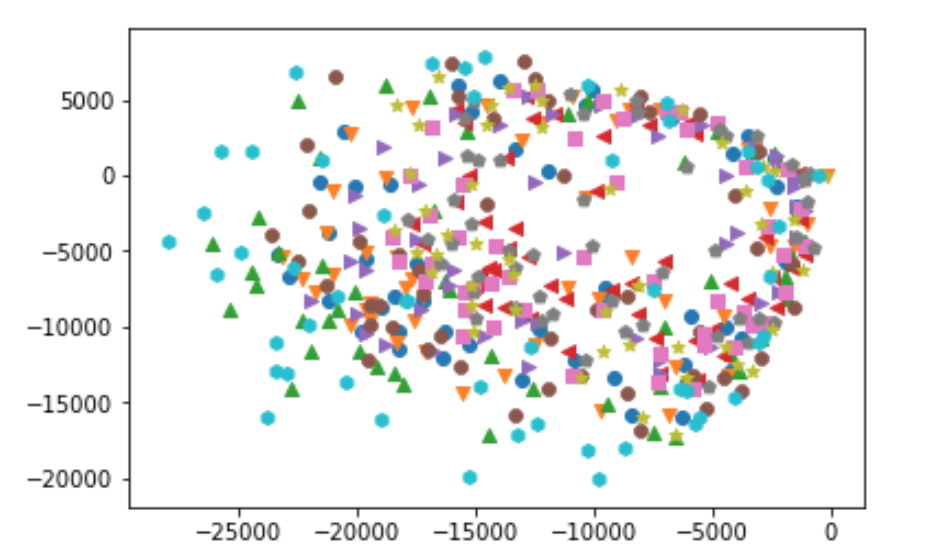
Generally speaking, the result doesn’t change much, which means pca is doing a good job. The program runs much faster. More neurons might result in worse results; this is probably due to over-fitting.

Activation function doesn’t affect the result much.

e)

Can’t do it. Took forever to run.

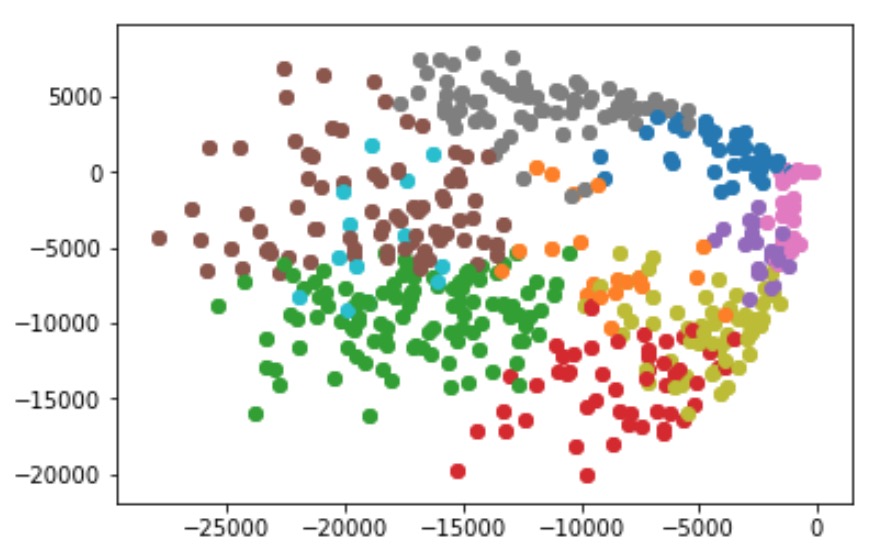
f)



According to the graph, they are not spreadable.

g)

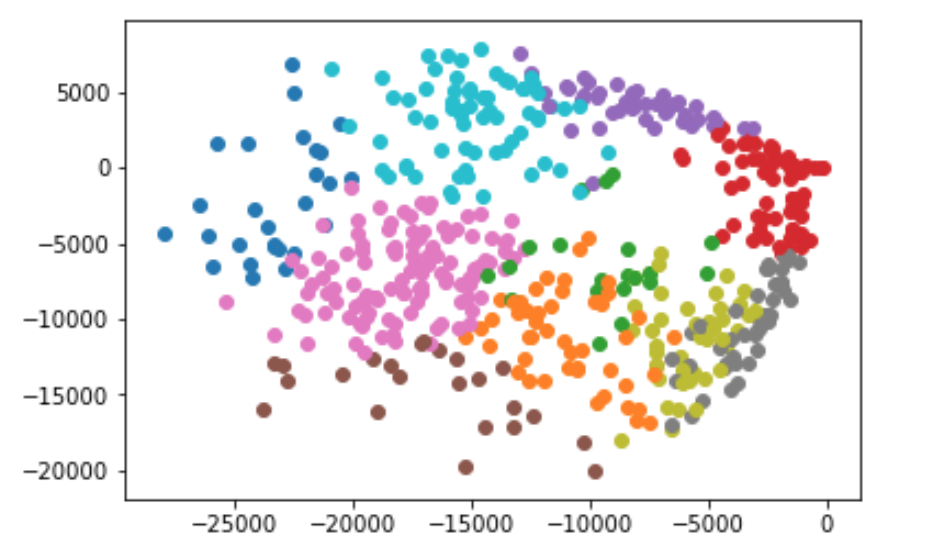
Apply kmeans directly on the dataset



Apparently kmeans cannot correctly separates the data

Miss-classification error = 17.8%. (sum{1(y\_i, y\_j)/number of pairs of (y\_i, y\_j)}

Apply kmeans to d=100 dataset



Apparently kmeans cannot correctly separates the data

Miss-classification error = 16.6%. (sum{1(y\_i, y\_j)/number of pairs of (y\_i, y\_j)}

h)

I got best result with sigma=10 and K=10

Miss-classification error = 12.6%. (sum{1(y\_i, y\_j)/number of pairs of (y\_i, y\_j)}

It’s not successfully recovering the result, though it does a better job than kmeans.