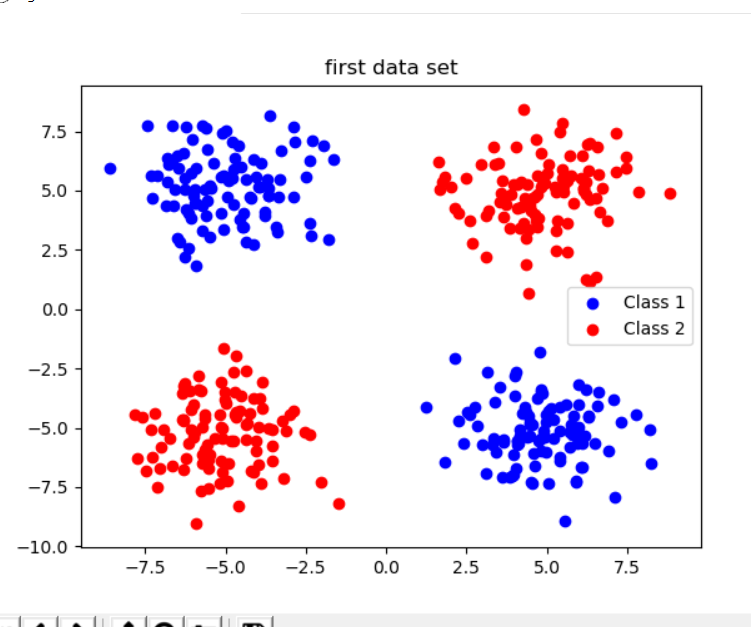
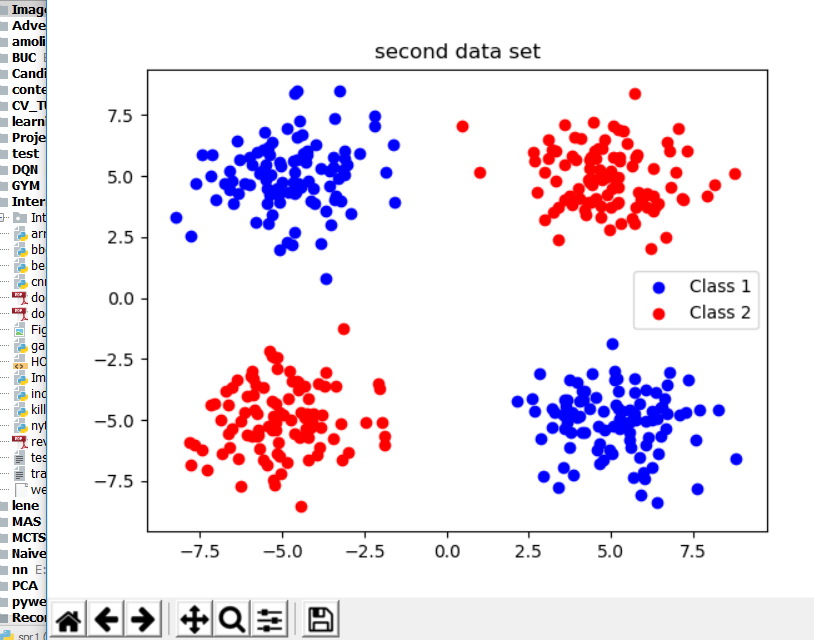
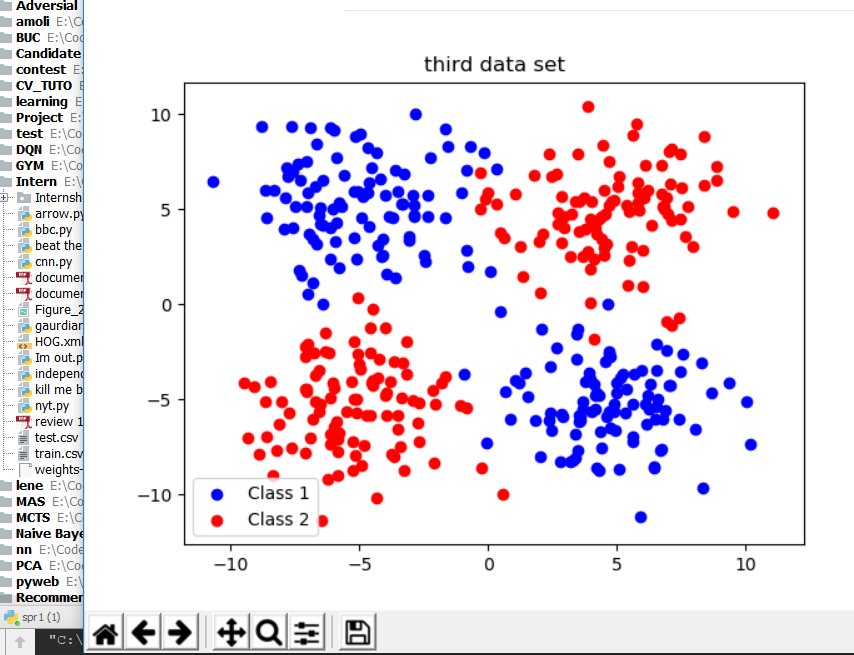
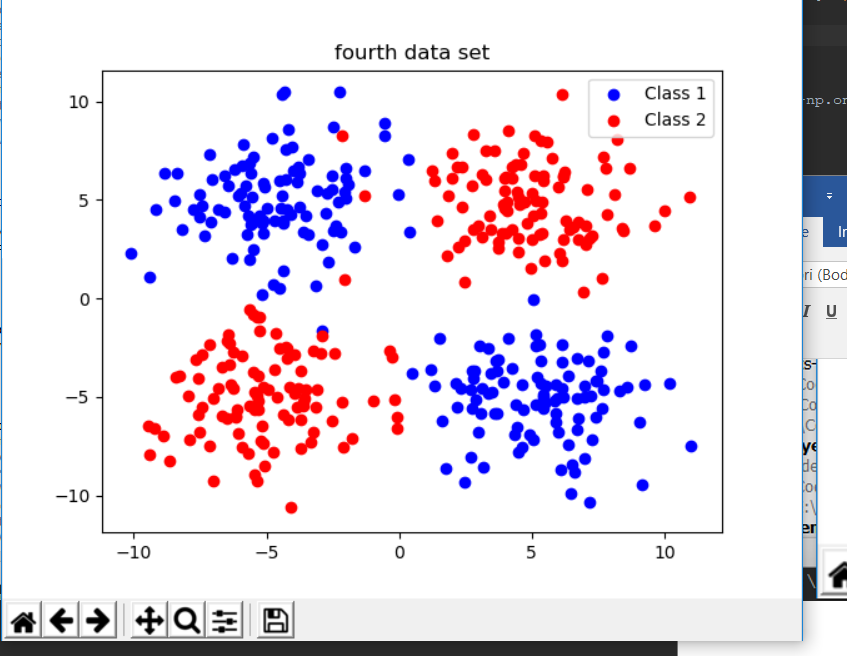
For first question, we write a function in python with inputs of m, s, n which indicates mean, sigma, and number of samples respectively. We use np.random.seed to give it seed mentioned in question which leads to:

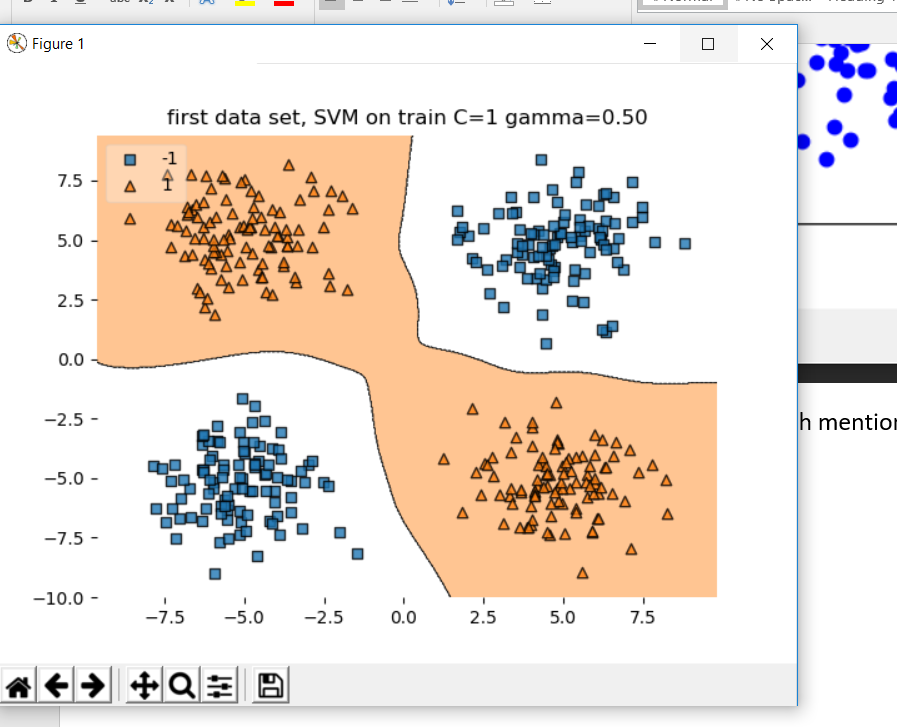




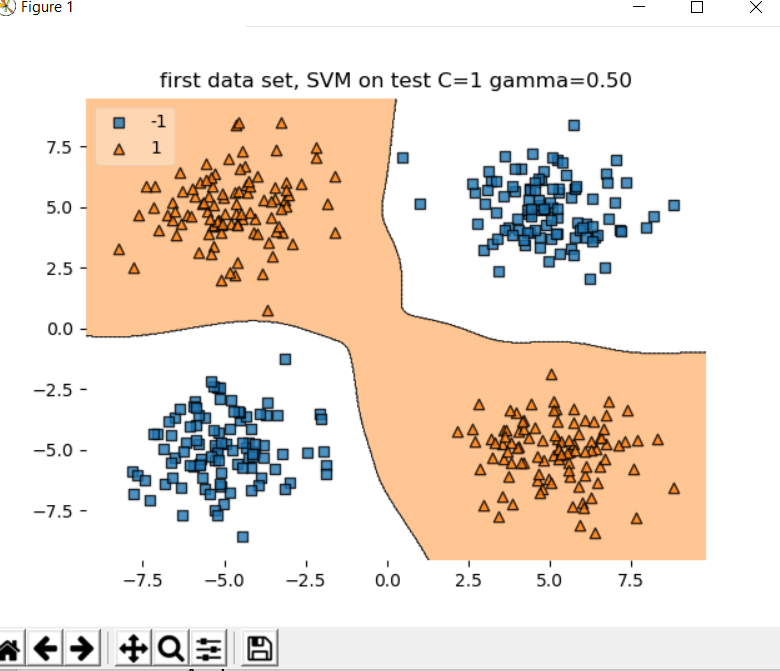




Then I used scikit svm to get results of svm with rbf with mentioned hyper parameters which lead to:



And



For test part as we can see in the picture it has no misclassification which also confusion matrix shows:

[[200 0]

[ 0 200]]

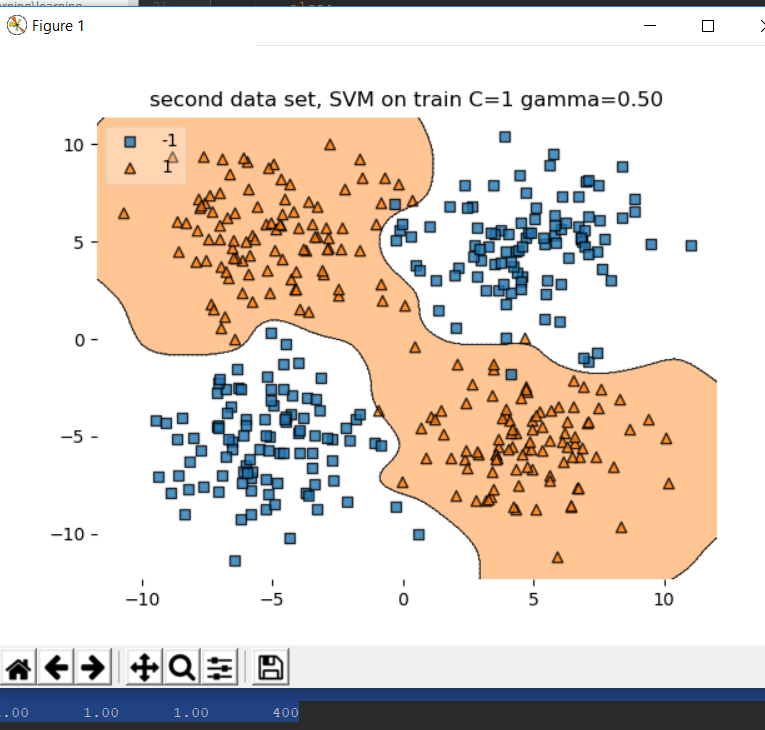
precision recall f1-score support

-1.0 1.00 1.00 1.00 200

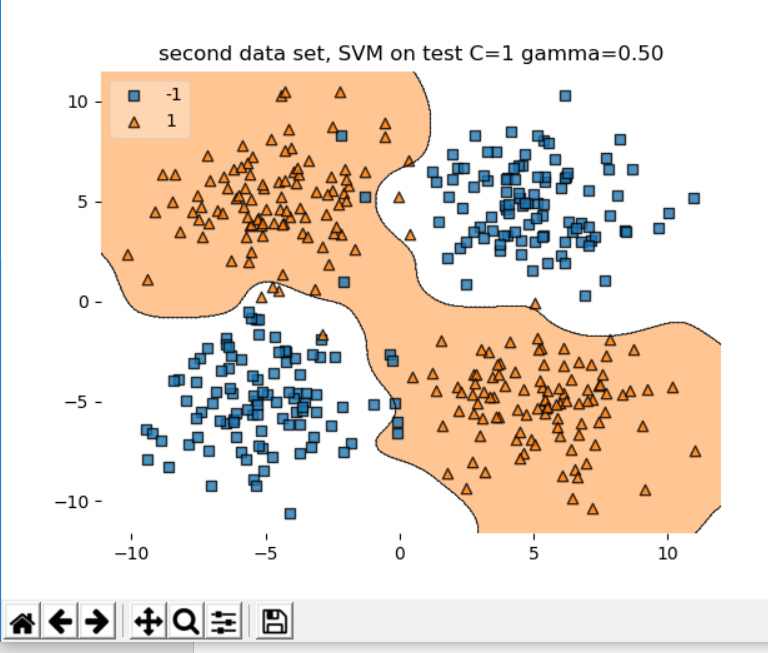
1.0 1.00 1.00 1.00 200

avg / total 1.00 1.00 1.00 400

after I used it to classify the third and fourth which led to



And for tests:



As we can see there are some missclassifications which confustion matrix shows:

[[193 7]

[ 7 193]]

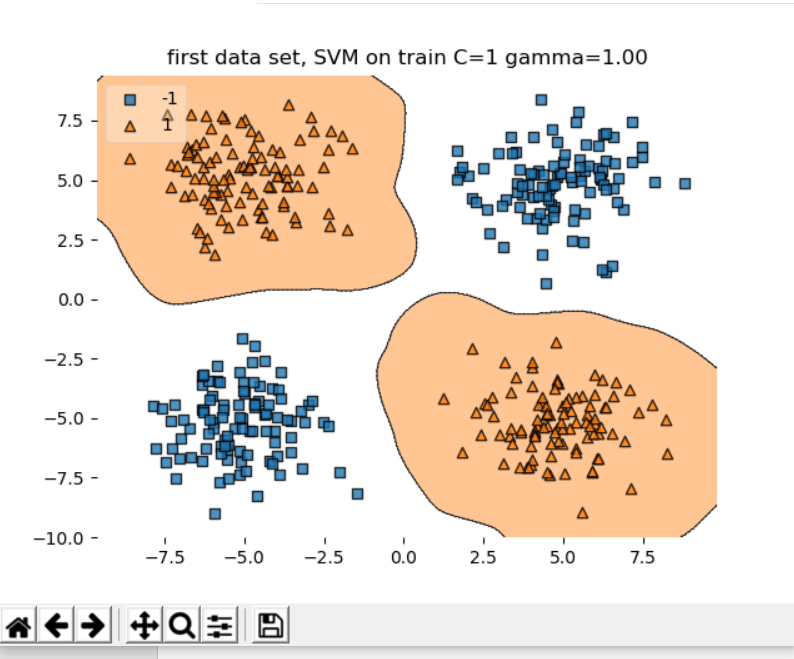
precision recall f1-score support

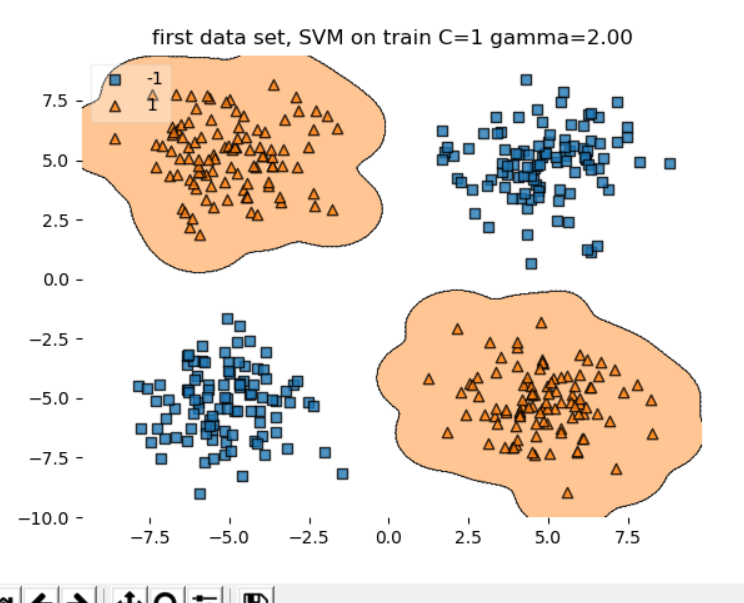
-1.0 0.96 0.96 0.96 200

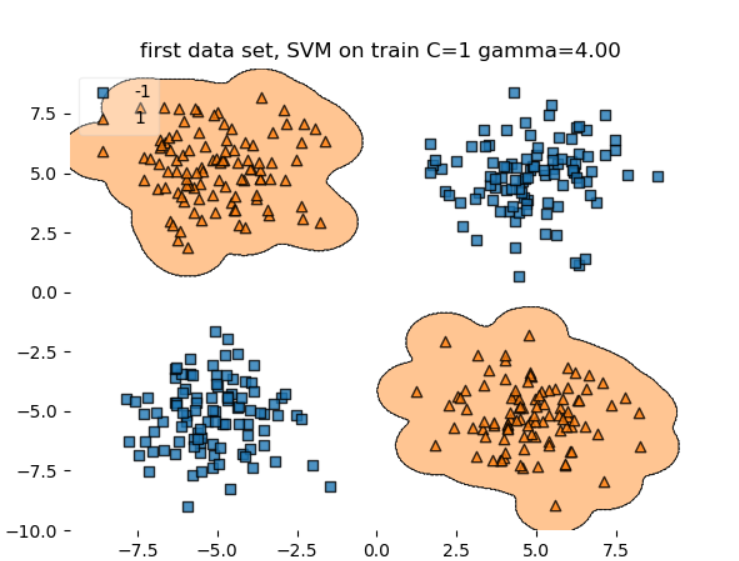
1.0 0.96 0.96 0.96 200

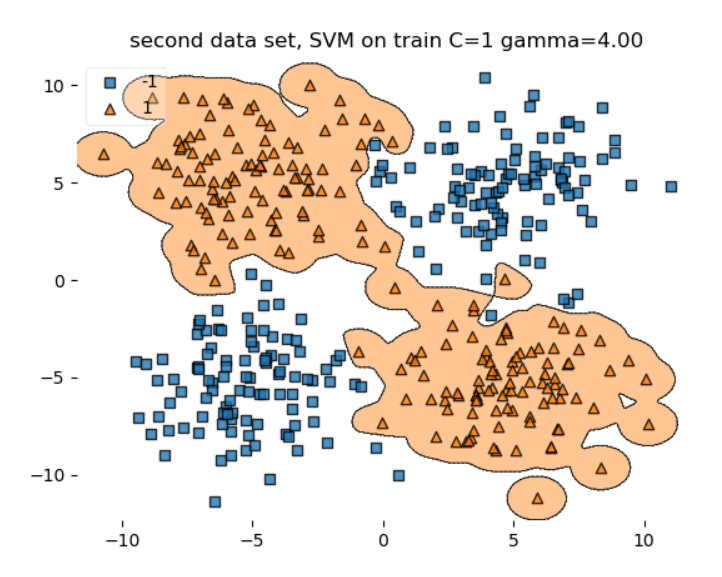
avg / total 0.96 0.96 0.96 400

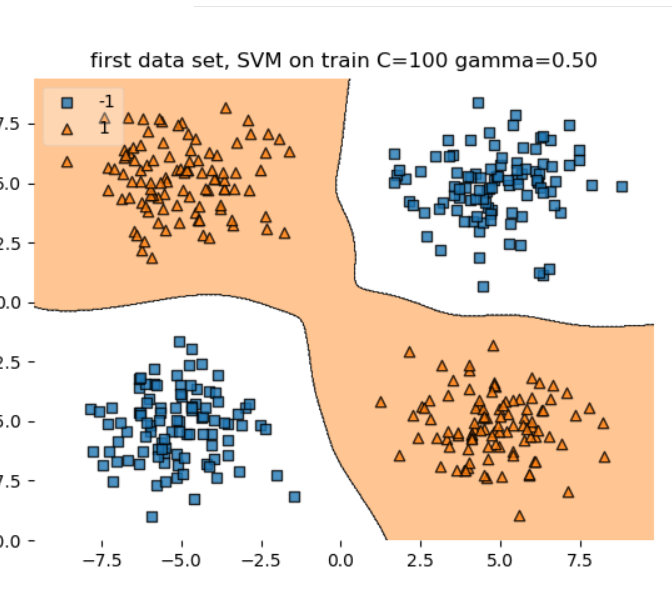
since there is a limitation in upload I won’t upload all cases because the code is runnable:

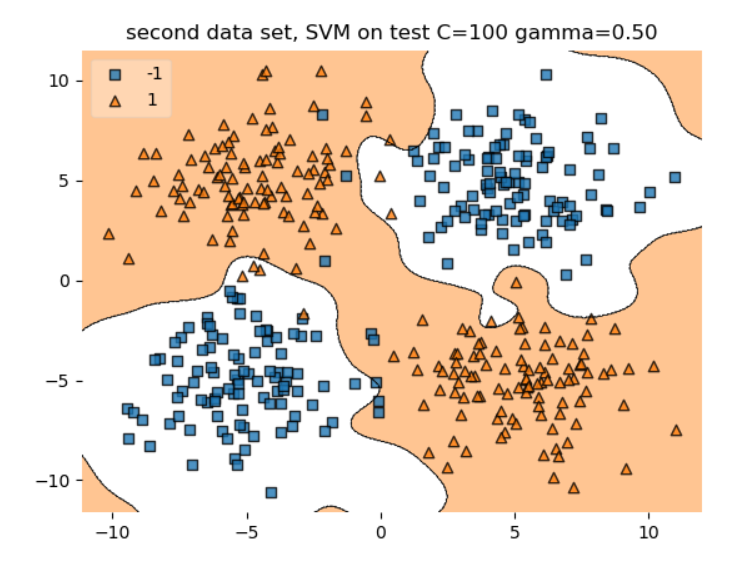












After Watching all figures I draw some conclusion which I share now:

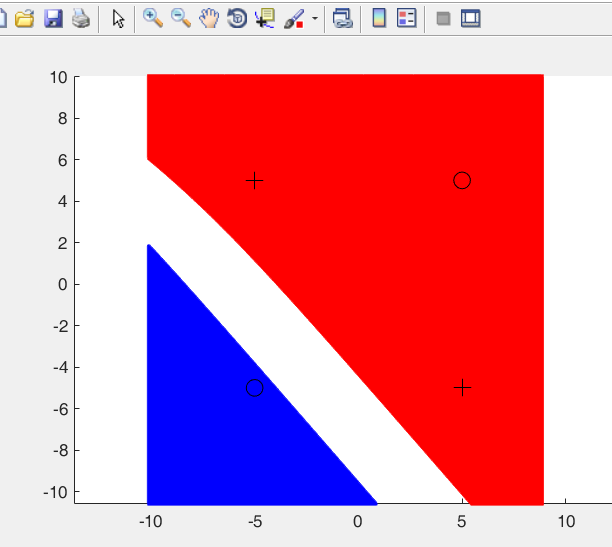
C which is misclassification penalty when it moves toward higher value we use smaller margin to classify our classes in another word: decision boundary becomes less biased and has more variance (more dependent on the individual data points).

Gamma which we can see in this: K(x,xi) = exp(-gamma \* sum((x – xi)^2)

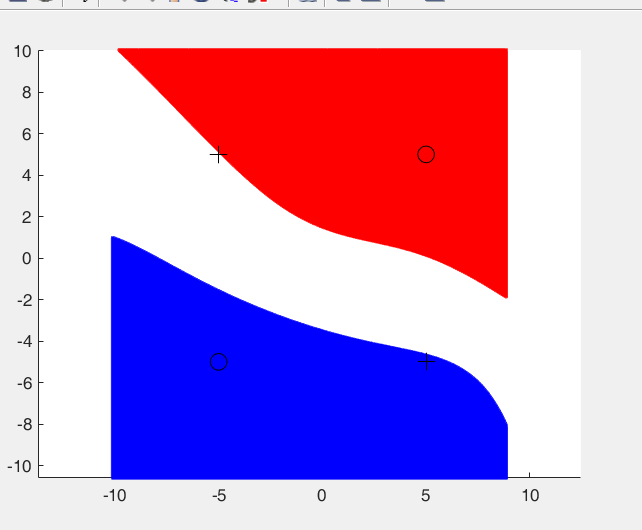
It makes lots of curves and islands which means higher variance which means now boundary is dependent on individual data.

In Second part which we concentrate on backpropagation algorithms:

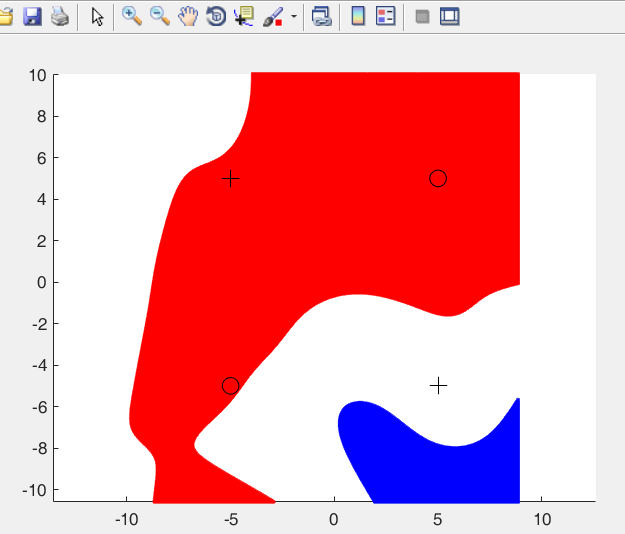
After writing data\_generator and watching data I wrote a confusion which counts error and give us ratio of error and wrote NN trainer which led to this results:



Circles shows mean of data for class 1 and plusses show mean for class 2 and we can see what regions had been learned for 2 hidden, lr 0.01 in 1000 epochs then I increased hidden layer:



For 4 hidden and lastly for 15 hidden:



2 hidden Error train : 34.50%

2 hidden Error test : 36.50%

4 hidden Error train : 65.00%

4 hidden Error test : 65.25%

15 hidden Error train : 79.25%

15 hidden Error test : 79.00%

In next exercise we changed learning rate and epochs which led to below results again I restrain myself from adding more figures because of upload limitation and just write results because we can run it in Matlab:

lr = .01 epochs = 300 Error train : 60.25%

lr = .01 epochs = 300 Error test : 62.00%

lr = .001 epochs = 300 Error train : 70.75%

lr = .001 epochs = 300 Error test : 72.50%

lr = .01 epochs = 1000 Error train : 55.00%

lr = .01 epochs = 1000 Error test : 54.75%

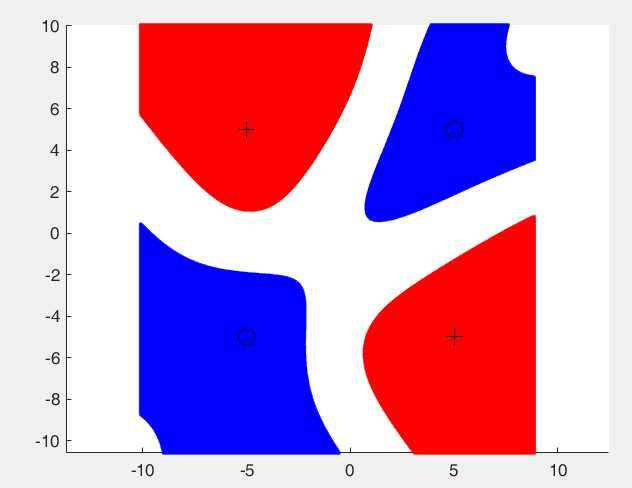
lr = .001 epochs = 1000 Error train : 73.50%

lr = .001 epochs = 1000 Error test : 73.75%

Lastly, using adaptive learning:

Adaptive Learning Rate Error train: 8.75%

Adaptive Learning Rate Error test: 8.75%



Which It shows adaptive learning learns faster and the best and the more hidden node you add it can be useless from somewhere on(which we can find with exhaustive search and other approaches).