**3.1.A** The loss function was implemented and a value of J=1 and we obtained a gradient vector [-1.19917069, -1.56709412] for an all zeros theta vector.



Figure1. Generated Data Visualization

**3.1.B** We tried varying C and found that C=100 wasn’t giving a very clean boundary due to the larger size of the dots. We therefore attempted C = [100,110,120] in addition to C=1.

As can be clearly seen for C >= 100, the boundary classifies the sample almost perfectly but looks unnatural and indicates a risk of high error rate for larger datasets.

 

Figure 2. C=1 Figure 3. C=100



Figure 3. C=110(cleanest) Figure 4. C=120

**3.1C** Implemented the Gaussian Kernel as instructed and the example calculation output is below:

Guassian kernel value (should be around 0.324652) = 0.324652467358

Using the Gaussian kernel svm we obtained a nice separating boundary for the non linearly classifiable dataset:



Figure 5. Gaussian SVM boundary for σ = 0.02, α = 10-4 and C = 1

**3.2** We varied the hyper-parameters as follows:

Cvals = [0.01,0.03,0.1,0.3,1,3,10,30]

sigma\_vals = [0.01,0.03,0.1,0.3,1,3,10,30]

We received the following output from our cross validation code:

best\_C=30 best\_sigma=30 best\_accuracy=0.96

We then trained our svm with the above hyper-parameters and obtained a decision boundary as follows:



Figure 6. Best gaussian SVM boundary for σ = 30.0, α = 10-4 and C = 30

**3.3** Spam Classification:

Range of tuning parameters –

sigma\_vals = [0.01,0.03,0.1,0.3,1,3,10,30]  
Cvals = [0.01, 0.1, 0.5, 1, 10, 50]  
lr\_vals = [1e-3, 1e-1]  
num\_iters\_val = [1000, 500]

We split the input into 3900 training and 100 validation examples.

Outline of our approach(pseudocode):

*Split training data randomly into train and validation datasets*

*Initialize best hyperparameters*

*for all sigma\_values:*

*Kernelize and scale train\_dataset*

*Kernelize and scale validation\_dataset*

*for all C\_values:*

*for all num\_iterations:*

*for all learning\_rates:*

*train the svm*

*get an accuracy score* ***ac***

*if* ***ac*** *> best\_accuracy\_score:*

*update best\_hyperparameter\_values*

*Kernelize and scale training setagain using best\_sigma obtained*

*Train svm using best hyperparameters*

*Kernalize and scale test input*

*Predict on this kernelized data and get accuracy score*

We initially attempted the non kernelized version of the svm classification algorithm (we didn’t want to have 4 levels of a for-loop ) to get an initial idea of the accuracy. We tried the straightforward method twice – one without the intercept term and once with the extra intercept feature.

In both runs we obtained really high accuracies on the test data:

* For no intercept term (Input shape 4000x1899) our best output was:

Best Hyperparameters:

C = 0.1 num\_iters = 1000 leraning\_rate = 0.001

Accuracy of model on training data is: 0.99975

Accuracy of model on test data is: 0.988

* Best output with intercept term (Input shape 4000x1900):

Best Hyperparameters:

C = 0.5 num\_iters = 1000 leraning\_rate = 0.1

Accuracy of model on training data is: 0.9995

Accuracy of model on test data is: 0.989

Clearly, even without using a Gaussian kernel we achieved a high validation accuracy of 99.95% and a test accuracy of 98.9%. For a relatively simple problem of spam classification, using a Gaussian Kernel and burning the CPU seemed to be an overkill. We did it anyways just for curiosity’s sake. Results as follows:

Best Hyperparameters:

sigma = 1.0 , C = 10, num\_iters = 1000, leraning\_rate = 0.1, accurancy = 0.93

Accuracy of model on training data is: 0.9655

Accuracy of model on test data is: 0.95

Since we got very high (close to 100%) accuracies both for training as well as test data without resorting to scaling and kernelization, we believe the input data is linearly separable.

Surprisingly, using RBF kernel resulted in lower accuracies (as low as 65% for some hyperparameter values) with the best being 94.61%.

Most of our runs produced values in the 85%-90% range.

We suspect this to be the result of the randomness in selection of the test and the training sets.

Our split was (3900:100)