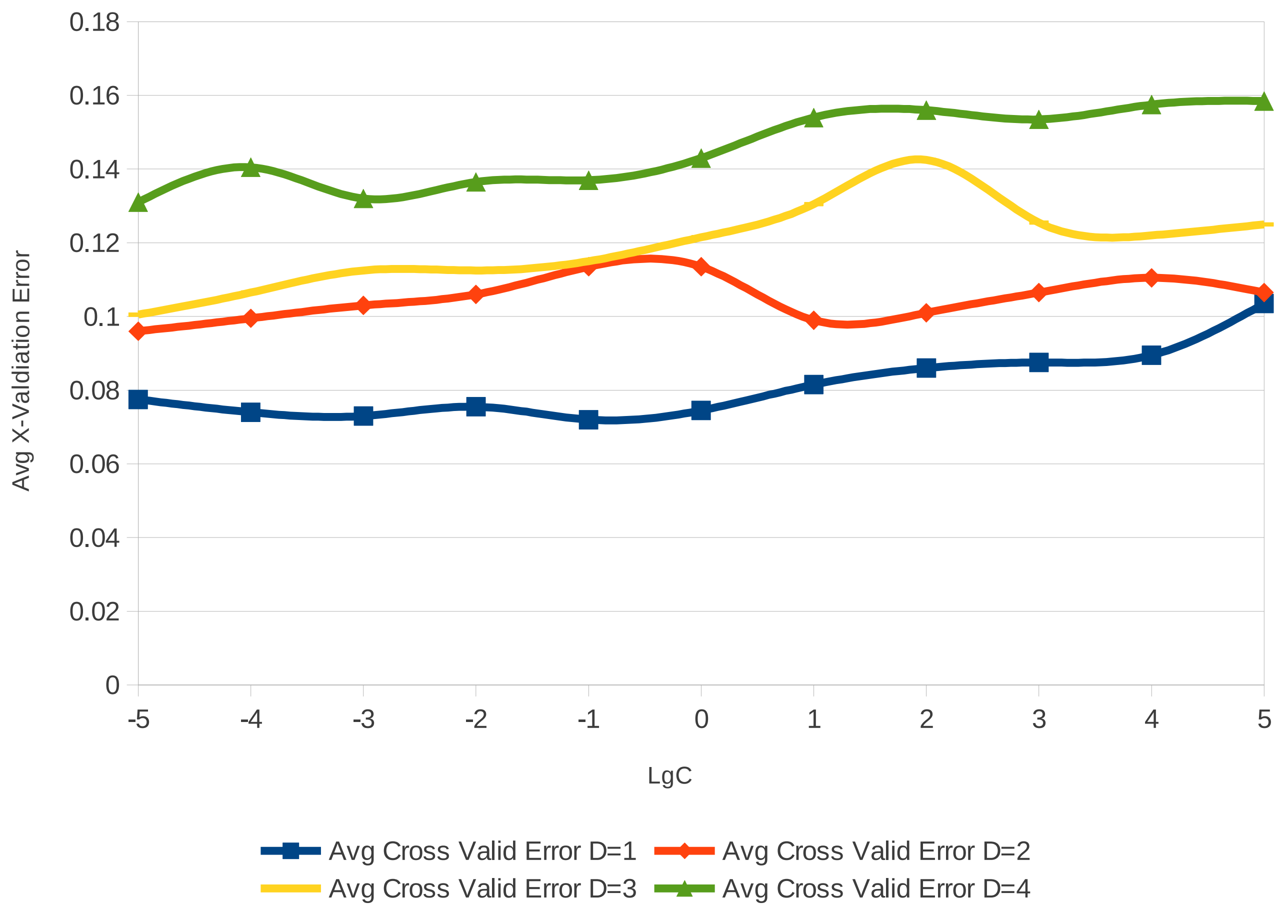
1. Implementation

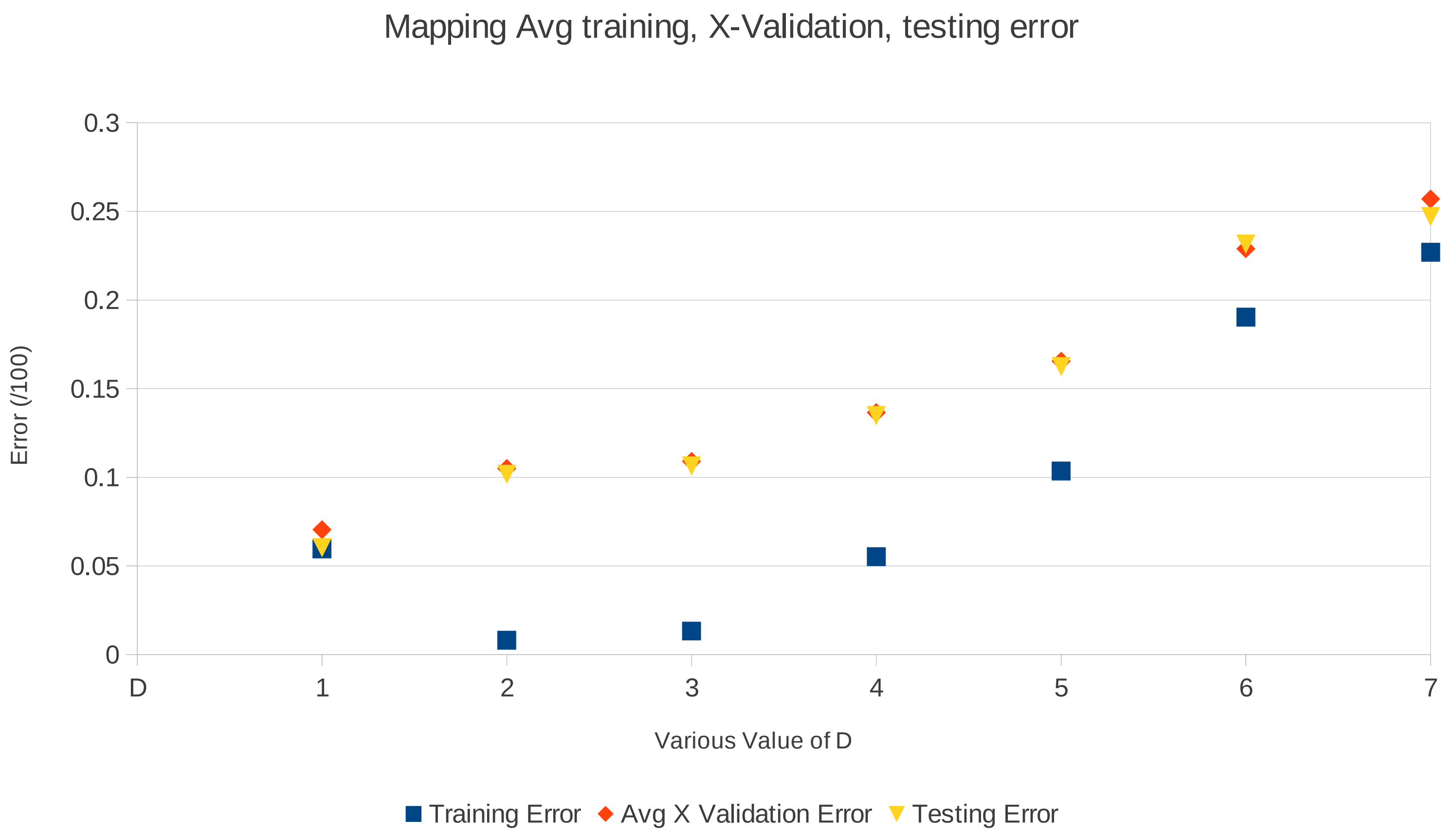
All done
For Question 2, 8 different processes were run separately, for different D (1:4) and C (2^(-5):2(5)). The Cross-validation error and training error were logged into separate files (Eg. Log_1_-5_0.txt), from which they were copied to an Excel file and the graph made. Once this was done and C selected, main.lua was changed to make the loop for C inactive and testing included in the new run. These data were also saved in 7 different files (Eg. test_d_1.txt). These again were copied into excel and graphs plotted. All these files are also being attached for reference.

2. Cross-validation with kernel SVM on spambase

2.1) We observe the best cross-validation error is around 8%, for D=1, and c=2^(-2).
For other values of D also, the average cross-validation error is low for Lg(c)~2



2.2) For a fixed C=0.25, and varying D from 1 to 7, we plot the Average Training Error, Average Cross-validation Error, Average Testing Error. We see the average testing error matches (with almost full consistency) the average error rates of cross-validation. This makes sense too, as the average cross-validation error is nothing really but something like a testing error, performed on a set of data, chosen out of the training data.



2.3) The value of C has already been determined to be 2^(-2) {its even better around 2^(-2.5)}.

The training error is low for D=2 and 3, but the cross-validation and testing error are high, and highly inconsistent with training error, so not the best choice. For D=1, the training error is higher than that for D=2 or 3, but the cross-validation and testing error here are much better, and all the 3 errors are also very close to each other, giving us a much more appropriate model.

C=2^(-2)=0.25 D=1

3. Multi-classification using binary classifiers on MNIST

3.1) The L2 regularization parameter \lambda and the slack parameter C are inversely related.

C=1/(\lambda)

C control the relative importance of correctly classifying the training examples and obtaining max separation margin. The soft margins reduce to hard margin is C goes to infinity. Similarly, \lambda controls trade-off between minimizing training error and simplicity of function. Only with lambda, a small lambda means the ideal scenario of a normal loss function.

3.2) In One-Vs-All, there are N scenario for total N classes. In one-vs-all, we find if the given input belongs to particular class or not. If not, we do not check which class it belongs to. The logic is only to determine if its in a particular class or not. If it isn't, there will automatically be an iteration where it will be judged as part of the class in question. We check for every class, the value of f(x) function and use the one with maximum value.

Training Error-	29%
Testing Error-	25%

3.3) In one-vs-one, we test each input and check the decision. If the decision is 1, and we increase the weight of that class. Class is assigned based on maximum weight.

3.4) For One-Vs-One,

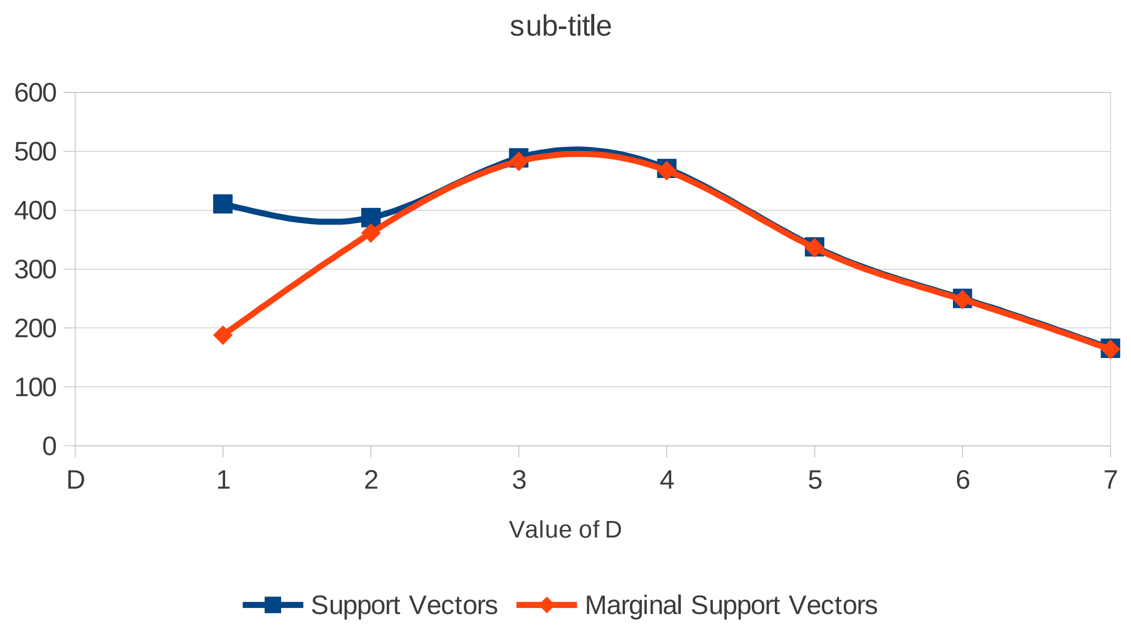
Training Error-	15%
Testing error-	22.3%

The training/training error is better for one-vs-one as compared to one-vs-all

4. Bonus, Challenging and Optional Questions

4.1) While for any non-marginal support vector, value of *alpha* will be equal to C. For marginal support vectors, the inequality if actually 0<*alpha*<C. A simple check of this inequality gives us which of the support vectors are marginal support vectors.

4.2)



We see that the percentage of Marginal Support Vector as a percentage of Total Support Vector increases with increase in D. The reason for the same is also quite clear. In a higher dimensional space, the probability to for the dataset to be separable is much more than a low dimensional space.

4.3) The training error increases as there is increase in value of P. As we increase P in max(0,P-z), we are providing a, say, bigger value acceptable for the function.
As, say, P=3 will give us max(0,3-z) while P=1 would have given us max(0,1-z). P=3 will allow a bigger value to the loss function.

4.4) With P=0, the process comes close to acting like a perceptron, where the loss would be -2*y*wx