**Project 2 – Pattern Recognition**

**Name: Yildirim Kocoglu**

*The details of the generation of plots can be found in the given MATLAB script. The same variables: w, wo, λ, support vectors, etc. are used multiple times for generating multiple plots, they are replaced each time in the given script in loops.*

**Part-1)**

Figure-1 and Figure-2 is generated in Part-1 of the MATLAB script. Quadprog was used to solve for the λ values in the dual form and the standard equation given in the lecture was changed from a maximization problem to a minimization problem by multiplying the entire cost function with -1 since quadprog requires the format to be input as a minimization problem. Then: H, f, Aeq, Beq, A, B, lower bound, and upper bounds (0 < λ < C) were found from the given equation and the constraints to input into quadprog. The title of the graphs has all the information needed such as value of C value used, number of misclassified samples, length of the margin and total number of support vectors. Both figures show the “Support Vectors” but, it distinguishes between the support vectors on the margin hyperplanes and the support vectors that are inside the 2 margin hyperplanes and the support vectors that are outside the 2 margin hyperplanes and misclassified. The total number of support vectors are the summation of the following:



**Part-2)**

Figure-3 and Figure-4 is generated in Part-2 of the MATLAB script. Figure-3 is generated using C=0.1 and Figure-4 is generated using C=100. The number of samples are increased from 100-1000 in increments of 100 (100,200,…,1000) at each implementation and computation time has been saved each time for each method used. There are a total of 3 methods used for each C value (graph):

Method 1: SOFTSVM = our implementation of SVM using quadprog as the solver.

Method 2: fitcsvm(SMO) = MATLAB implementation of SVM using the SMO approach

Method 3: fitcsvm(L1-QP) = MATLAB implementation of SVM using quadprog

Method 3 is equivalent to Method 1 (our implementation) but, has more outputs (calculations) than our method which, explains the similar curve it has compared to our implementation but, takes slightly more time than our implementation.

In Figure-3, it’s obvious that the SMO approach (fitcsvm(SMO)) is computationally more efficient than the implementation done using the self-written MATLAB function (SOFTSVM) or the fitcsvm(L1-QP) method as the number of samples increase. This does not seem to be true in Figure-4 where C=100 since the SMO approach (fitcsvm(SMO)) takes a lot longer to converge to an answer than the self-written MATLAB function (SOFTSVM) or fitcsvm(L1-QP) as number of samples increase. The reason for this not clear yet, however, one feasible explanation was found in the MATLAB documentation of fitcsvm on the last page under “**Tips**” which states:

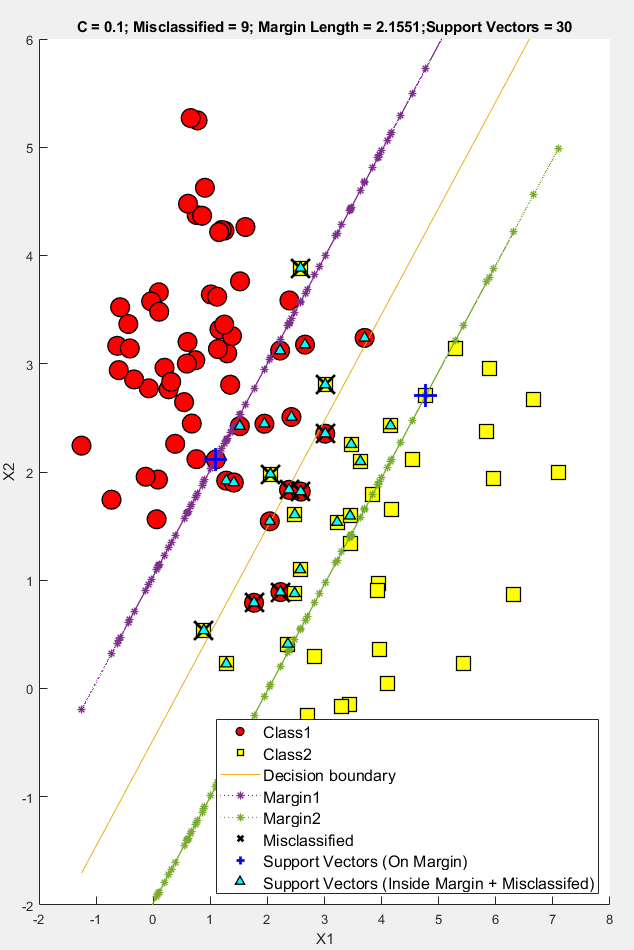
* Sparsity in support vectors is a desirable property of an SVM classifier. To decrease the number of support vectors, set **BoxConstraint** to a large value. **This action increases the training time.**

The BoxConstraint mentioned above is the same as the given C value (0.1 or 100). The link to the MATLAB documentation is given below to confirm this statement:

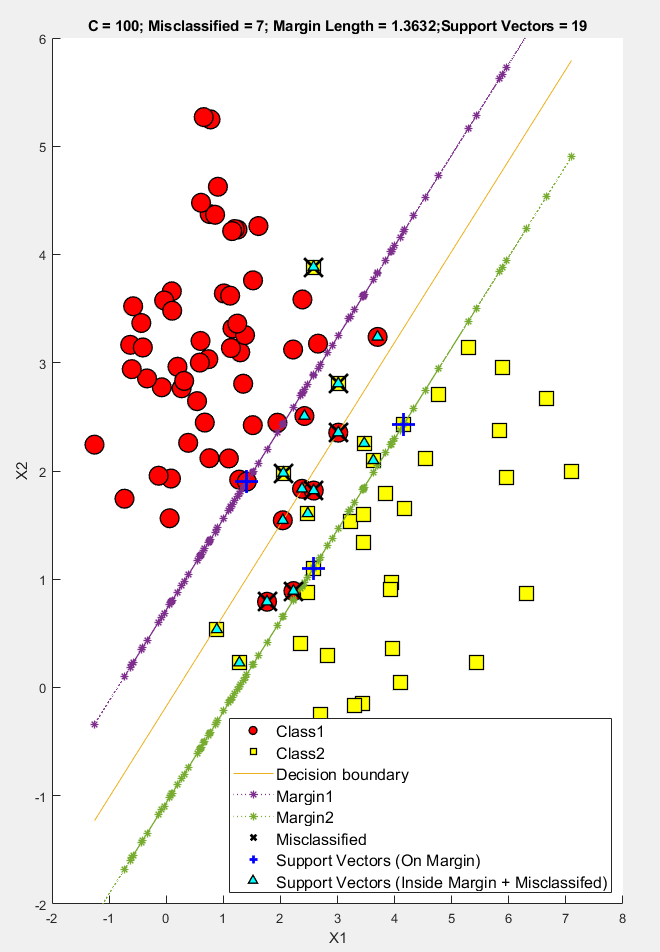
<https://www.mathworks.com/help/stats/fitcsvm.html>

After extensively researching for a better explanation for the results of Figure-4, no other feasible explanation could be found anywhere online. **The unexpected results in Figure-4 (C=100) can also come from the MATLAB implementation itself rather than the original implementation of SMO.** Also, the “up and down” pattern in the SMO approach in Figure-4 with increasing sample size can’t be explained at this point as well but, the general trend is still increasing computation time with increasing sample size even with “ups and downs” present. One suspected reason might be due to random generator generating samples (distribute samples in a certain manner) that may impact the training time of SMO method causing “ups and downs” but, this is just a speculation (thought) rather than a solid proof. **To confirm whether the results from Figure-4 are due to the MATLAB implementation of SMO or the original SMO approach, we will need to write the SMO implementation of both MATLAB and the original from scratch which I believe is beyond the scope of this project. It should also be noted that there are quite a large number of parameters that were left at default during this SMO application in MATLAB, that may cause this behavior but, to find this parameter(s), one still needs to be highly familiar with the SMO approach itself.**

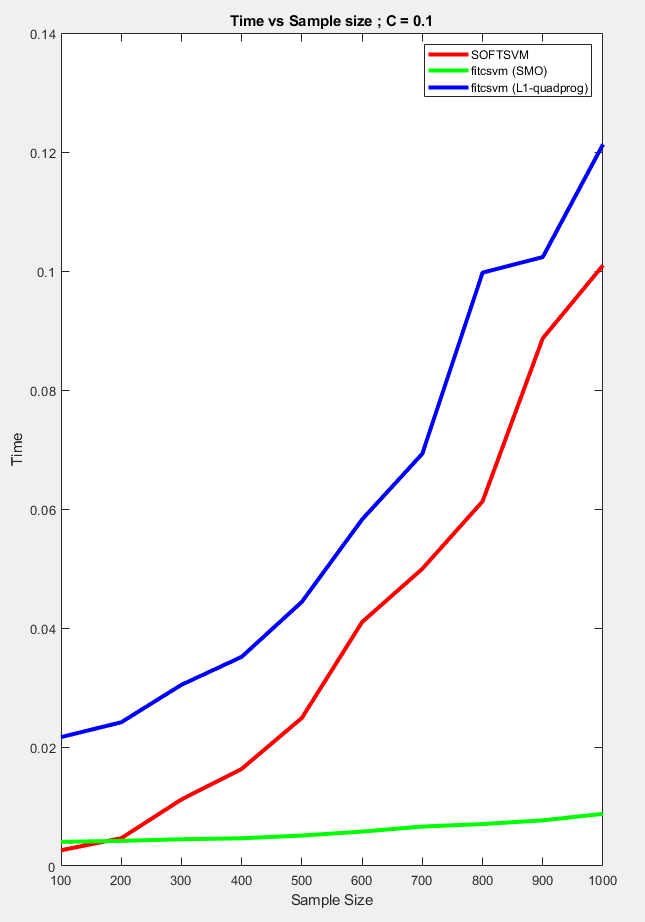
It should also be mentioned that the Figure-3 and Figure-4 is the result of the average of 100 time trials for all 3 methods which took about 388.484534 seconds (nearly 6 minutes) to finish running. After running a method 100 times, the average of all 100 elapsed times are taken to give a single value for the specific sample size (ex: sample size = 100). This is done to remove the bias that comes from the fact that the run times might be different each time even for the same script (with no changes to the script) due to the computer working on multiple tasks at the same time in the background. There is a high doubt that increasing this number from 100 to a 1000 will make much difference in the results after a visual inspection of the values for each time trial for each sample size. **The MATLAB script given uses a time trial value of 5 (~20 seconds to finish) to avoid long waiting times for the script to finish running but, this number can be changed as desired (lower value for shorter wait times and vice versa).** Also, needless to say, the run times will differ for each computer since the computational capacity of each computer is different.



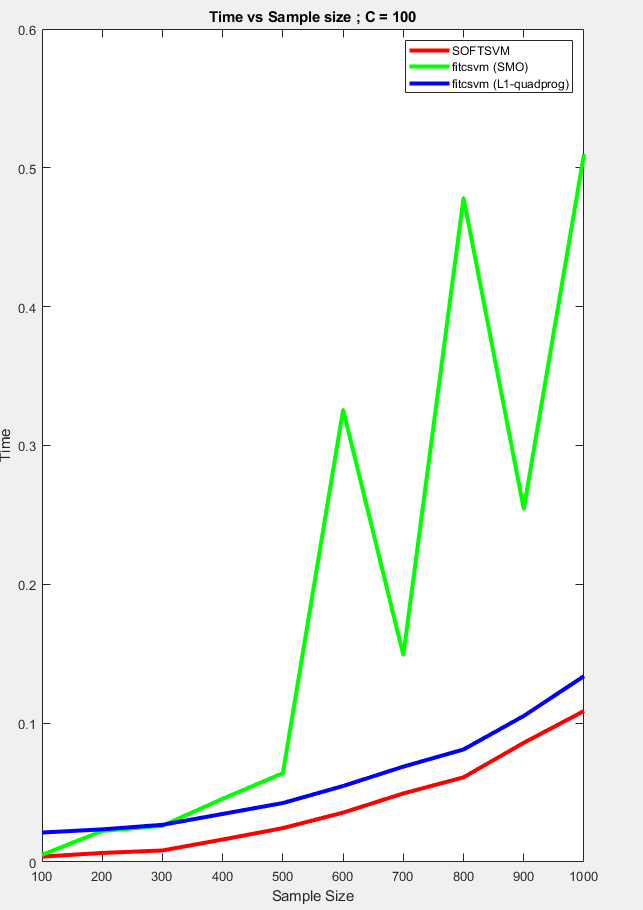
**Figure-1:** Soft-Margin SVM application with **C = 0.1**



**Figure-2:** Soft-Margin SVM application with **C = 100**



**Figure-3:** Time vs Sample Size: Soft-Margin SVM application (SOFTSVM vs fitcsvm(SMO)) with **C = 0.1**



**Figure-4:** Time vs Sample Size: Soft-Margin SVM application (SOFTSVM vs fitcsvm(SMO)) with **C = 100**