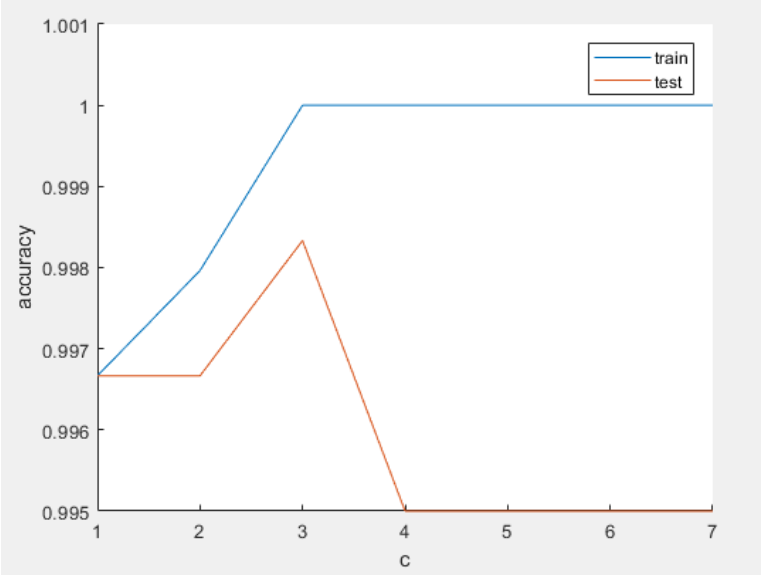
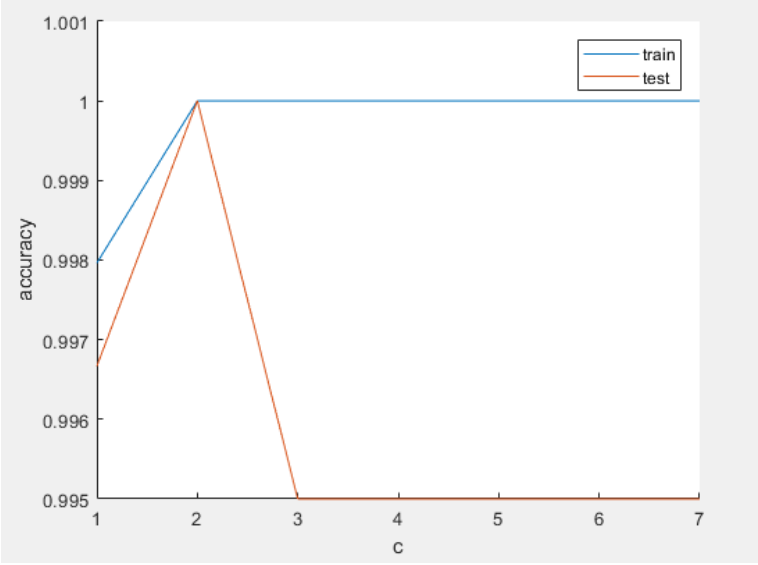
Binary Classification (LibSVM)

- Classes: (0, 1); First 10 features; Linear kernel:
  c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]



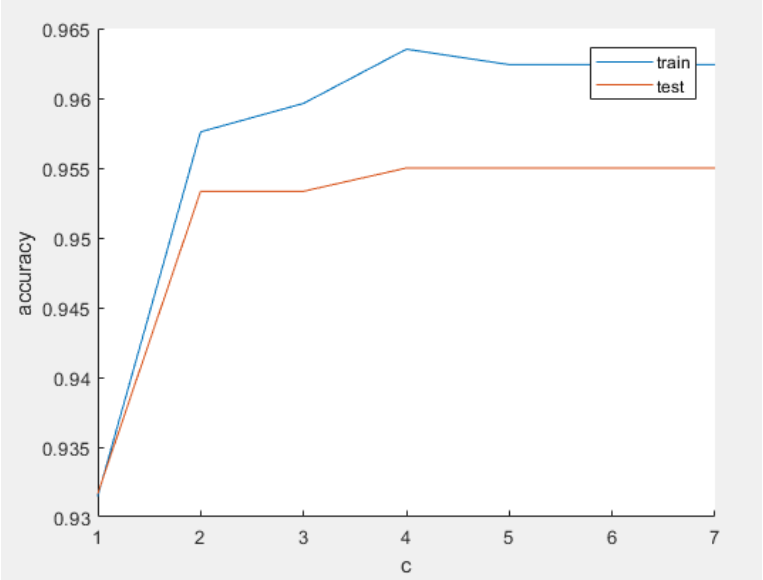
From the graph, c = 0.1 is the optimum c.

- Classes: (0, 1); All features; Linear kernel:
  c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]



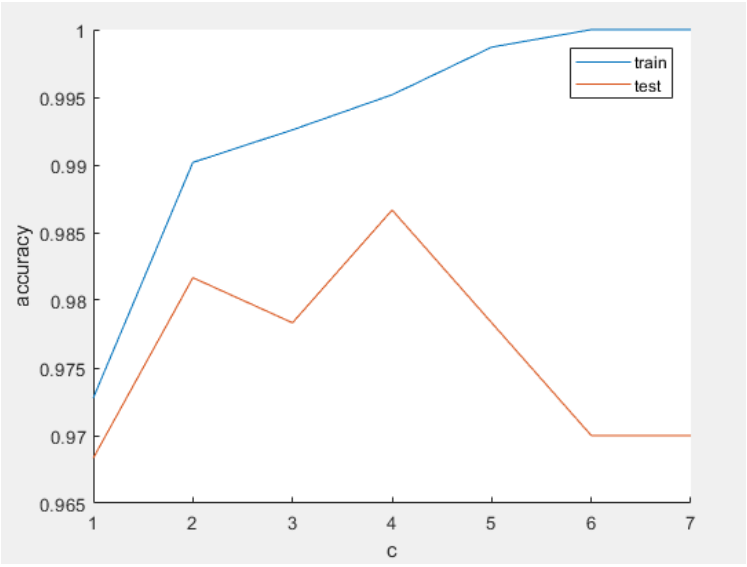
From the above graph, c = 0.01 is the optimum c.

- Classes: (4, 5); First 10 features; Linear kernel:
  c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]



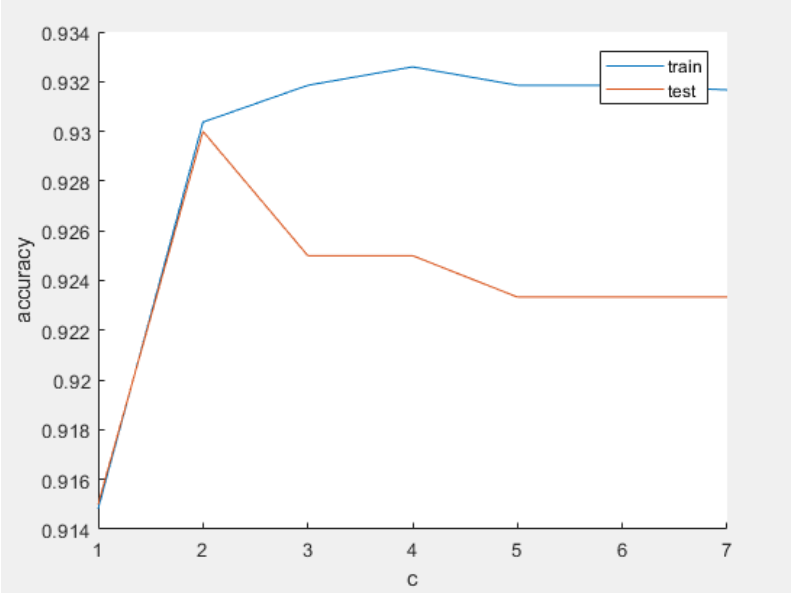
From the above graph, c = 1 is the optimum c.

- Classes: (4, 5); All features; Linear kernel:
  c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]



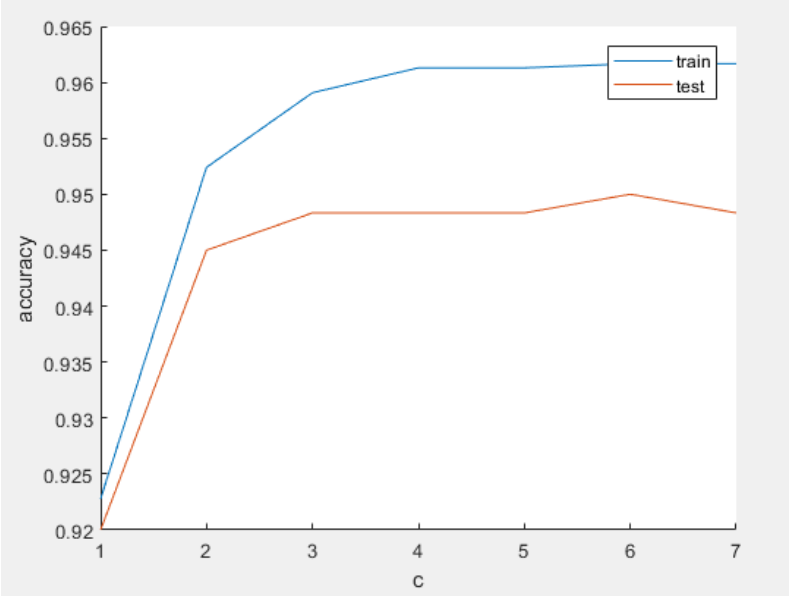
From the above graph, c = 1 is the optimum c.

- Classes: (8, 9); First 10 features; Linear kernel:
  c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]



From the above graph, c = 1 is the optimum c.

- Classes: (8, 9); All features; Linear kernel:
  c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]



From the above graph, c = 100 is the optimum c.

- Results

  We did binary classification using linear kernels, and calculated the optimum c for three pairs of classes using cross-validation. In each pair, we first used the first 10 features, then used all the features. Firstly, as we change the pair of classes used, we see that the accuracy varies. This suggests that the pair with higher accuracy is more linearly separable. Also, the accuracy when we use all the features is always higher than when we use the first 10 features, although by variable amounts. The classes where increase in accuracy is high suggests greater dependence on remaining features. From the graphs, we can clearly see the overfit, underfit and best fit regions. Optimal c changes when we include or exclude features.

- Classes: (0, 1); First 10 features; Rbf kernel:
  c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
  Gamma = [0.0001, 0.001, 0.01, 0.1, 1, 10]

  Optimum c = 100.0
  Optimum gamma = 0.01
  Accuracy = 1.0

- Classes: (0, 1); All features; Rbf kernel:
  c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
  Gamma = [0.0001, 0.001, 0.01, 0.1, 1, 10]

  Optimum c = 1.0
  Optimum gamma = 0.01
  Accuracy = 1.0

- Classes: (4, 5); First 10 features; Rbf kernel:
  c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
  Gamma = [0.0001, 0.001, 0.01, 0.1, 1, 10]

  Optimum c = 1.0
  Optimum gamma = 0.1
  Accuracy = 0.9833333333333334

- Classes: (4, 5); All features; Rbf kernel:
  c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
  Gamma = [0.0001, 0.001, 0.01, 0.1, 1, 10]

  Optimum c = 1.0
  Optimum gamma = 0.1
  Accuracy = 0.9983333333333334

- Classes: (8, 9); First 10 features; Rbf kernel:
  c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
  Gamma = [0.0001, 0.001, 0.01, 0.1, 1, 10]

  Optimum c = 1.0
  Optimum gamma = 0.1
  Accuracy = 0.96

- Classes: (8, 9); All features; Rbf kernel:
  c = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
  Gamma = [0.0001, 0.001, 0.01, 0.1, 1, 10]

  Optimum c = 100.0
  Optimum gamma = 0.01
  Accuracy = 0.9783333333333332

- Results

  We continued our previous exercise in this section, but worked with rbf kernels this time. We hyper tuned c and gamma using cross-validation. It is worthwhile noting that more than one set of c and gamma gave the same highest accuracy. In that case, the convention followed is to take the simpler model, that is, with lower values of c and gamma. Again, when using all features, the accuracy increases compared to using the first 10 features, suggesting dependence on all features. Also, rbf kernel gives mostly better accuracies than linear kernel.

Binary Classification (CVXOPT)

The dual problem for soft margin SVM is:

$$L(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m=1}^{N} \sum_{n=1}^{N} \alpha_m \alpha_n y_m y_n \mathbf{k}(\mathbf{x}_m, \mathbf{x}_n)$$

$$\text{subject to} \quad 0 \leq \alpha_n \leq C, \qquad \sum_{n=1}^{N} \alpha_n y_n = 0$$

The above equation and restrictions expressed in matrix form are:

$$L_{\min}(\alpha) = \frac{1}{2} \alpha^T \mathbf{H} \alpha - 1^T \alpha$$

where

$$\mathbf{H}_{N \times N} = (\mathbf{y}^\intercal \mathbf{y}) * K(\mathbf{X}, \mathbf{X})$$

$$-\alpha_n \leq 0, \quad \alpha_n \leq C, \quad \mathbf{y}^\intercal \alpha = 0$$

The problem that CVXOPT.solvers.qp solves is:

$$\min_x \frac{1}{2} x^T P x - q^T x$$
$$\text{s.t.} \quad Gx \preceq h$$
$$\text{and} \quad Ax = b$$

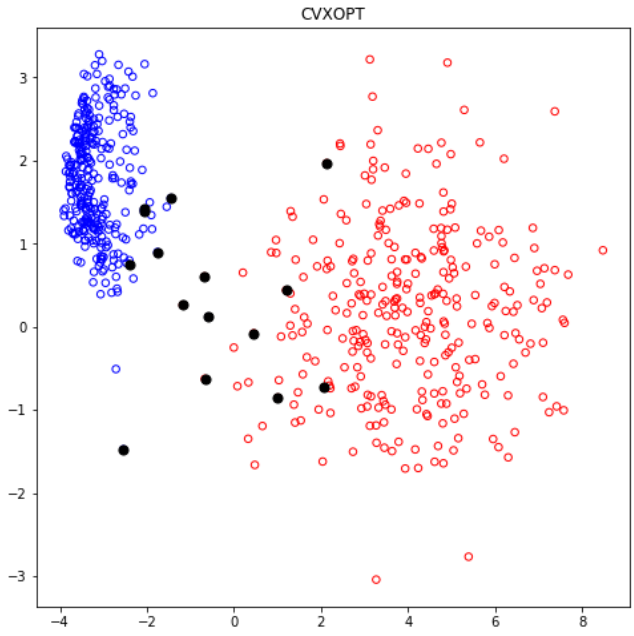
Converting our dual problem to that required by CVXOPT, we get alphas, from which we calculate weights and bias as:

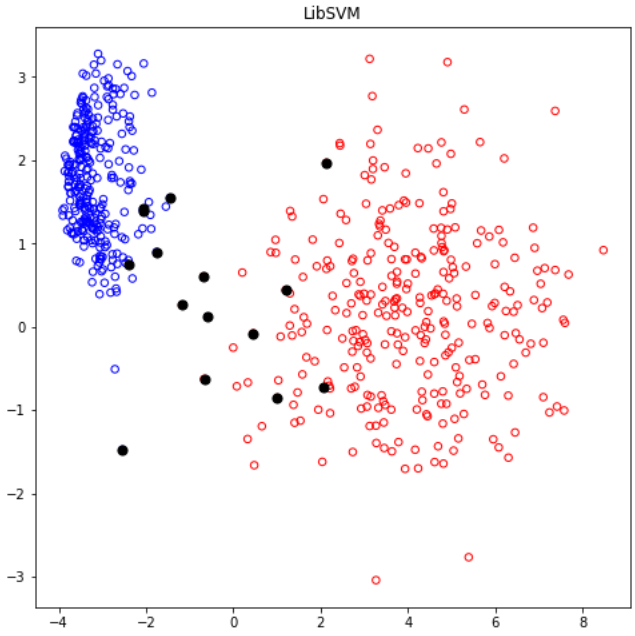
$$\mathbf{w} = \mathbf{X}^\intercal \alpha \mathbf{y}, \quad b = (\mathbf{y} - \mathbf{X}\mathbf{w})_{\in \mathbb{S}}$$

From the above we can calculate the class for each test case.

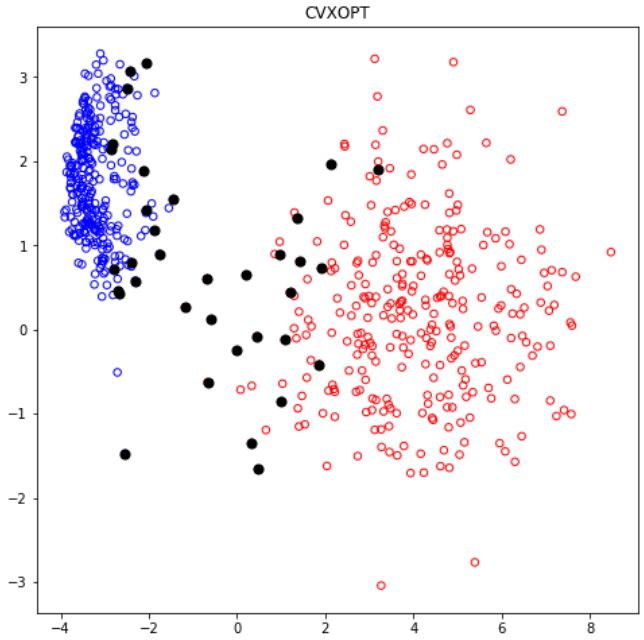
In this section, the tuned parameters from the previous section are used.

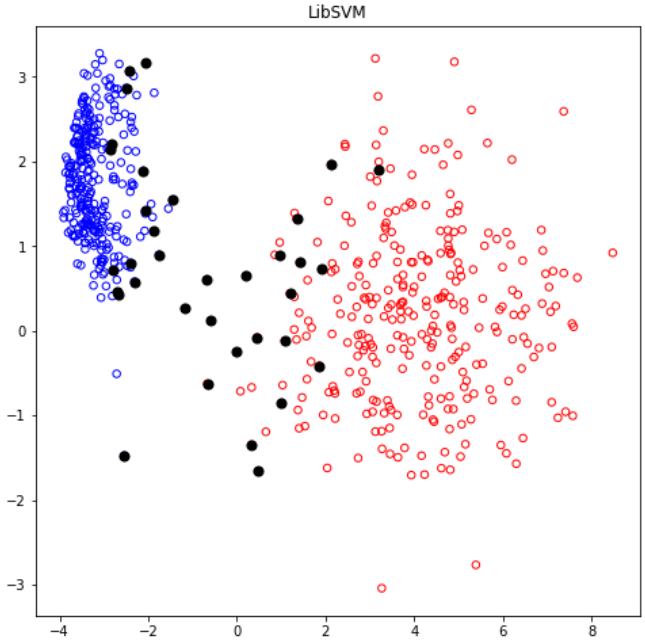
- Classes: (0, 1); First 10 features; Linear kernel:
  Accuracy from CVXOPT = 0.9983333333333334
  Accuracy from LibSVM = 0.9983333333333334



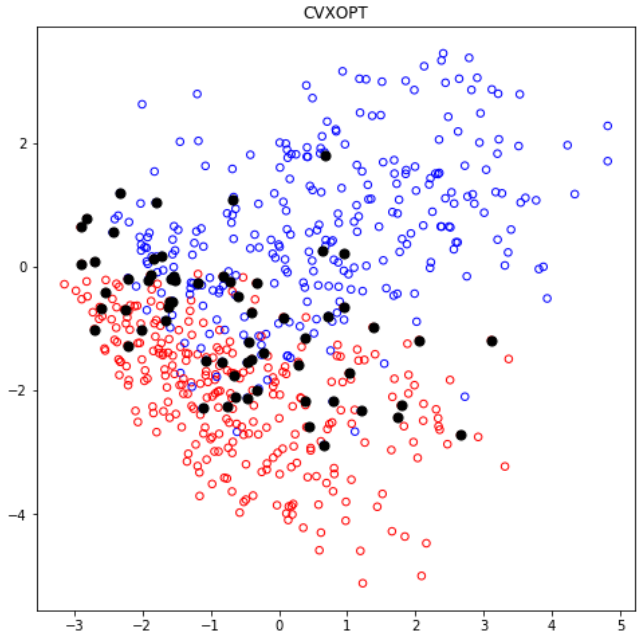


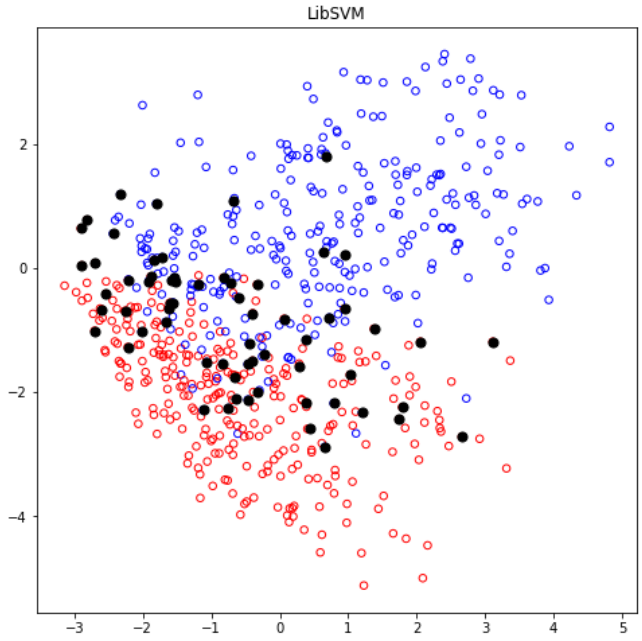
- Classes: (0, 1); All features; Linear kernel:
- Accuracy from CVXOPT = 1.0
- Accuracy from LibSVM = 1.0



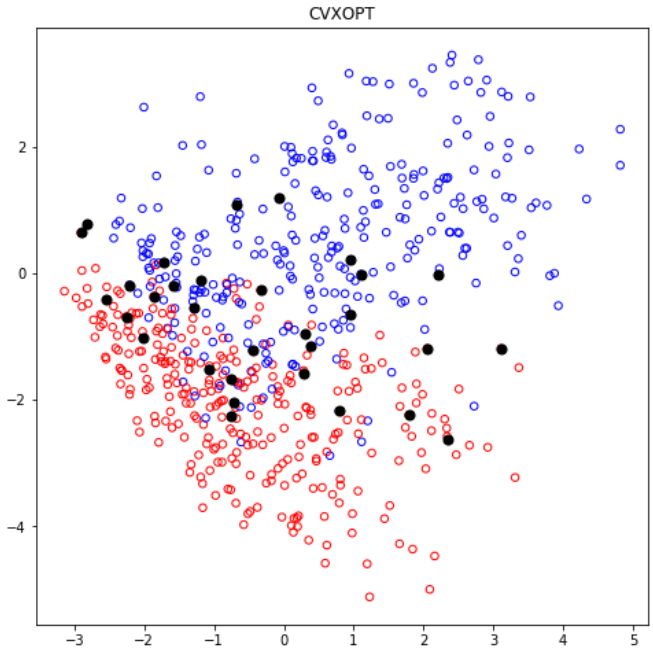


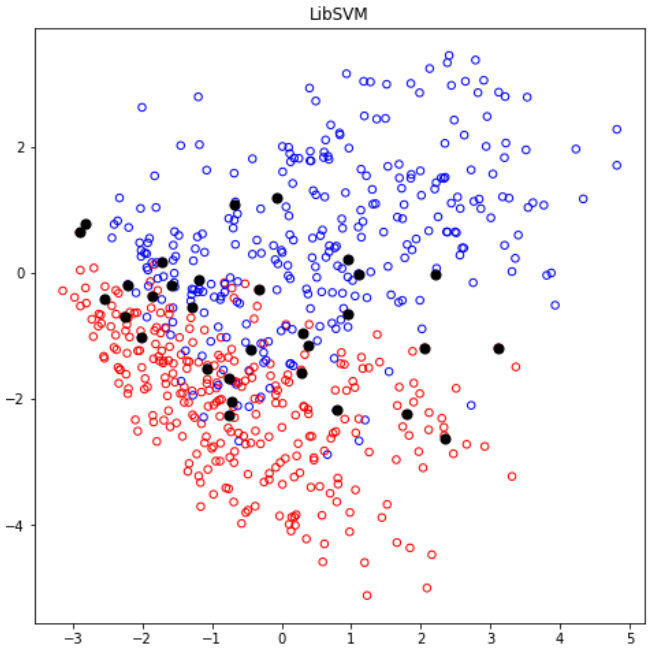
- Classes: (4, 5); First 10 features; Linear kernel:
  Accuracy from CVXOPT = 0.9533333333333334
  Accuracy from LibSVM = 0.9550000000000001



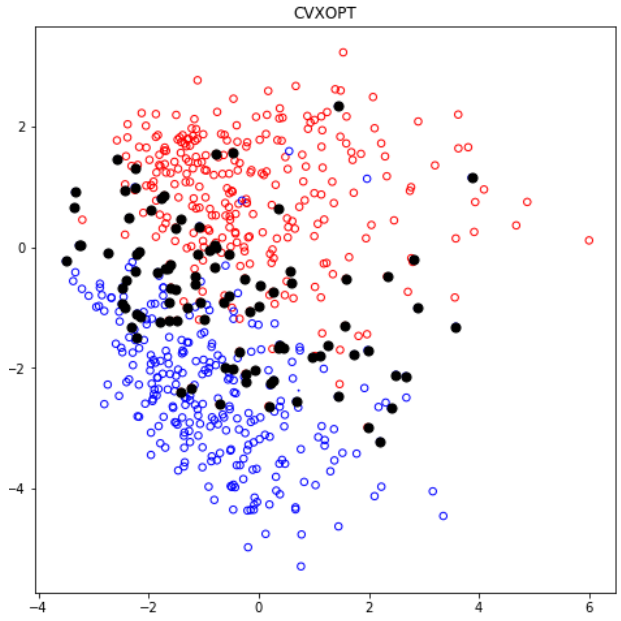


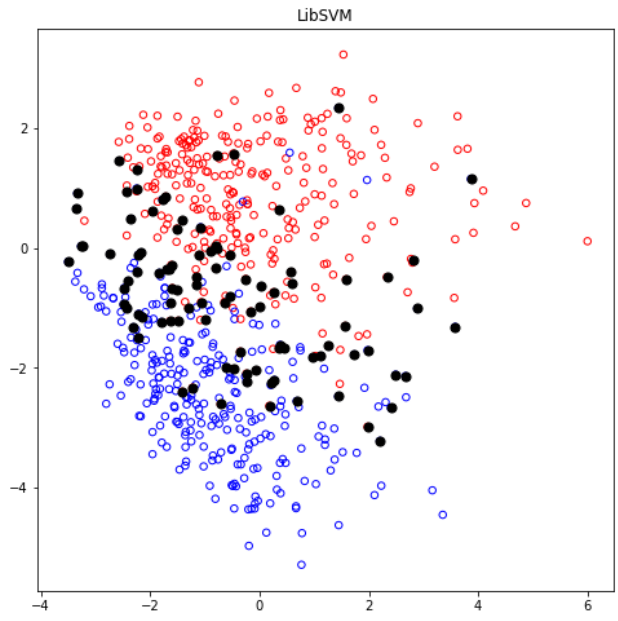
- Classes: (4, 5); All features; Linear kernel:
  Accuracy from CVXOPT = 0.9866666666666666
  Accuracy from LibSVM = 0.9866666666666666



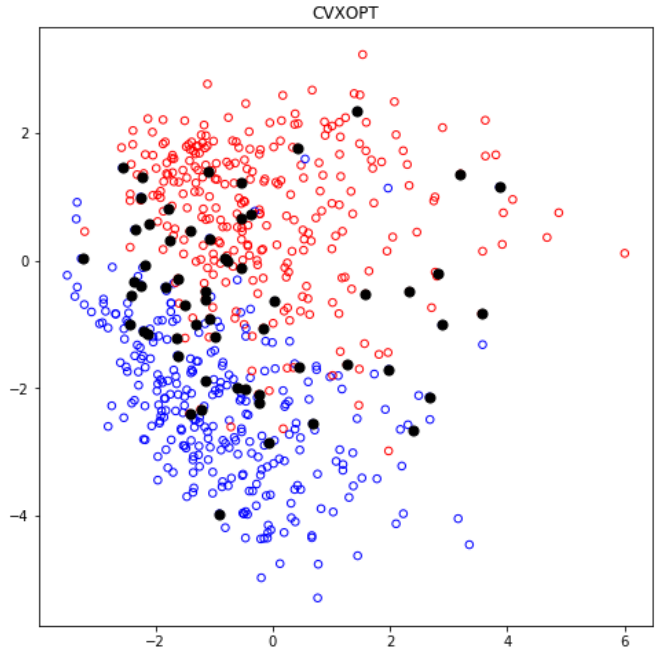


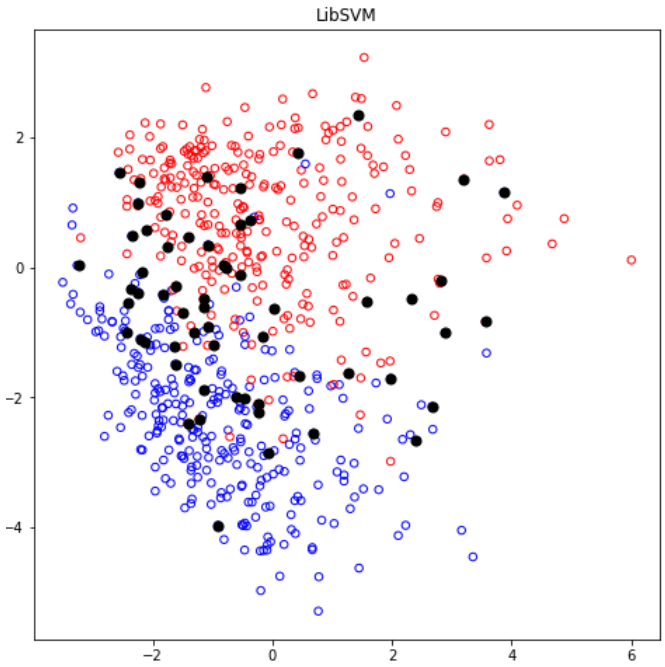
- Classes: (8, 9); First 10 features; Linear kernel:
  Accuracy from CVXOPT = 0.9216666666666669
  Accuracy from LibSVM = 0.9250000000000002



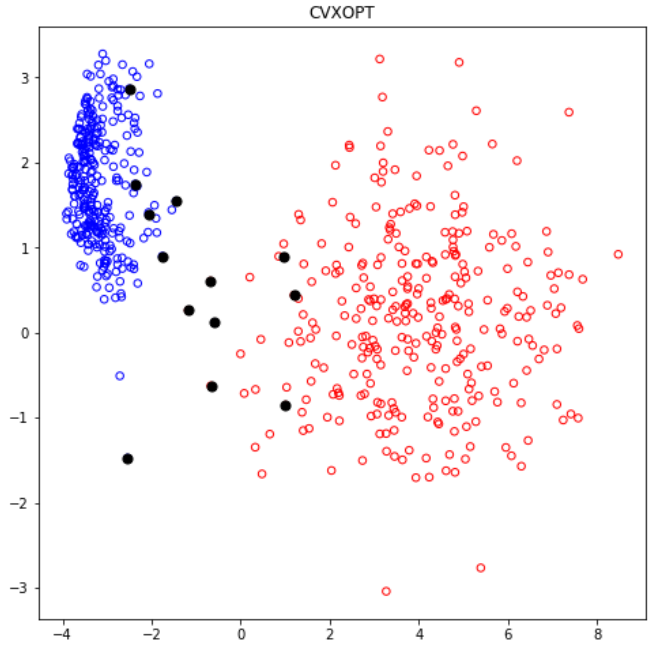


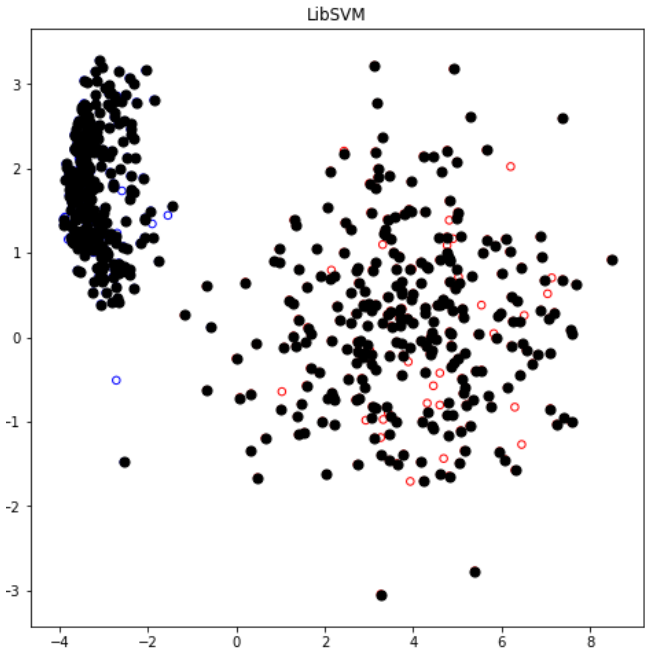
- Classes: (8, 9); All features; Linear kernel:
  Accuracy from CVXOPT = 0.9466666666666667
  Accuracy from LibSVM = 0.95



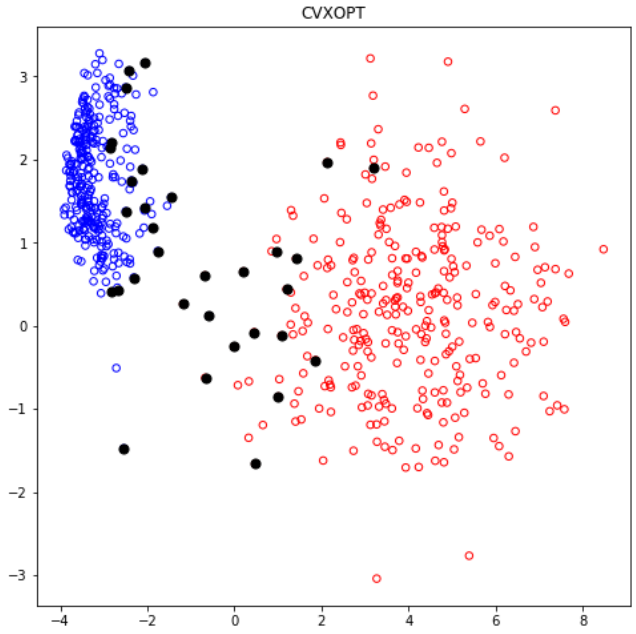


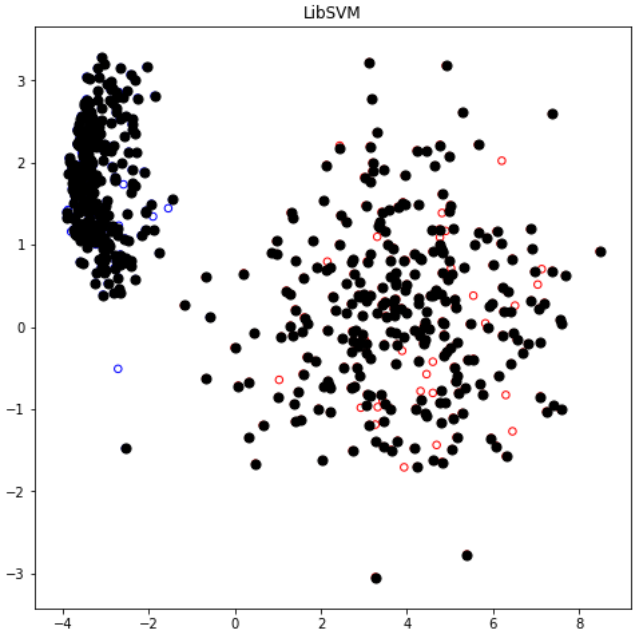
- Classes: (0, 1); First 10 features; Rbf kernel:
  Accuracy from CVXOPT = 0.9966666666666667
  Accuracy from LibSVM = 1.0



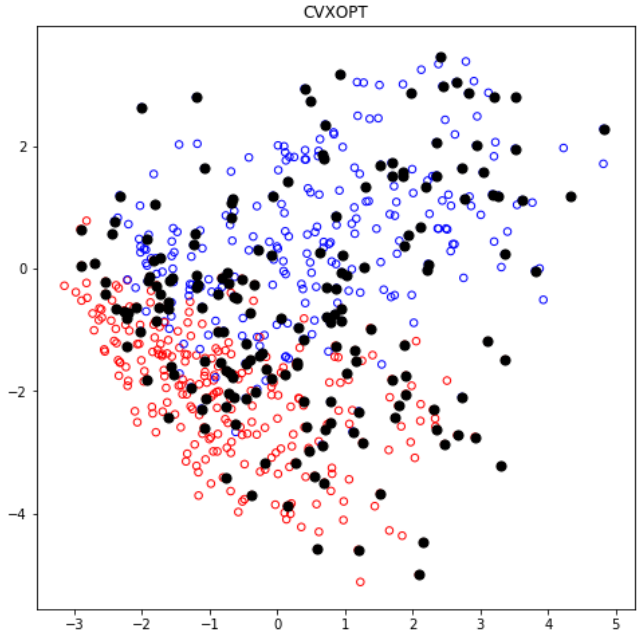


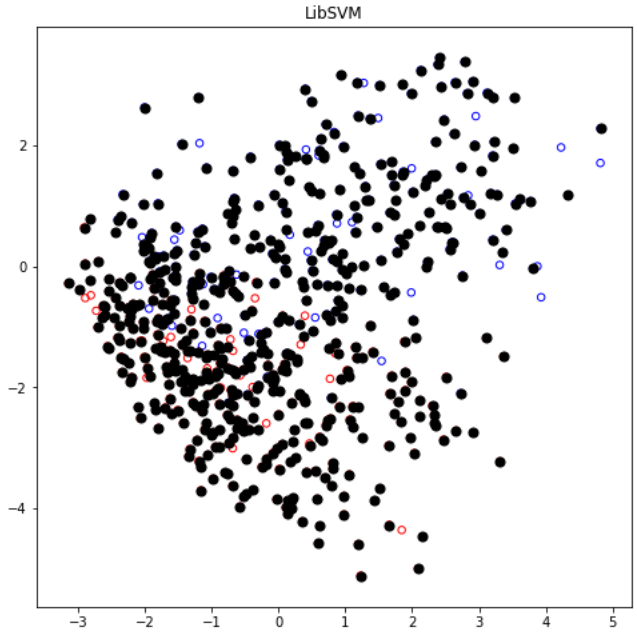
- Classes: (0, 1); All features; Rbf kernel:
  Accuracy from CVXOPT = 0.9983333333333334
  Accuracy from LibSVM = 1.0



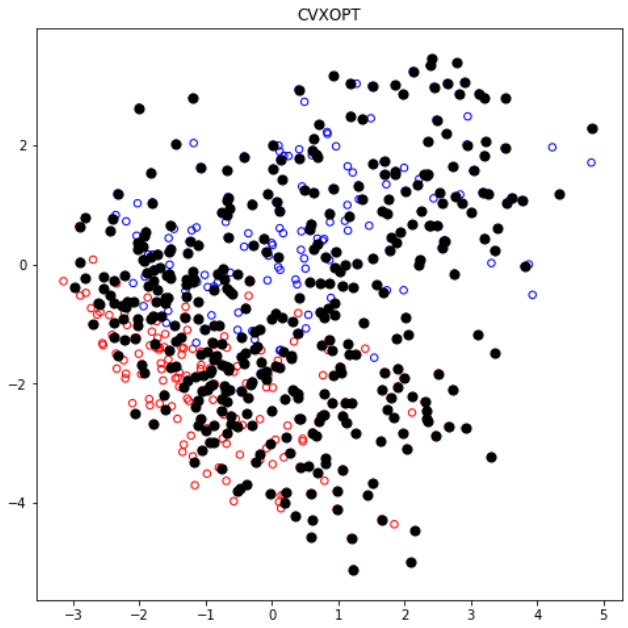


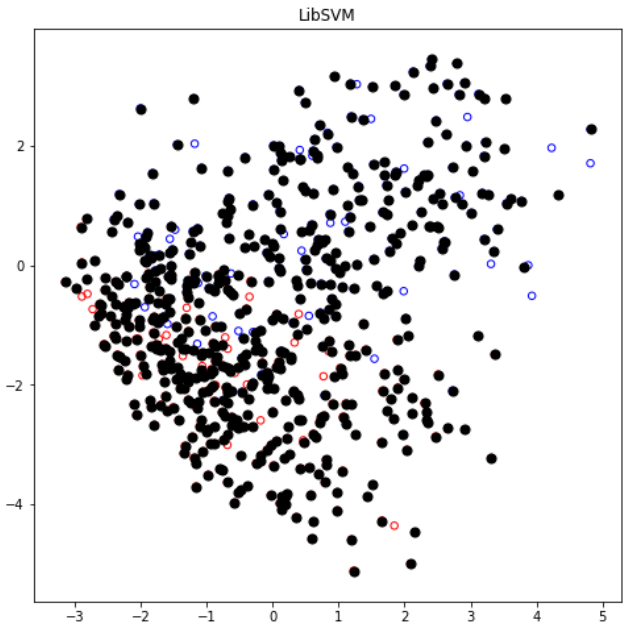
- Classes: (4, 5); First 10 features; Rbf kernel:
  Accuracy from CVXOPT = 0.9150000000000003
  Accuracy from LibSVM = 0.9833333333333334



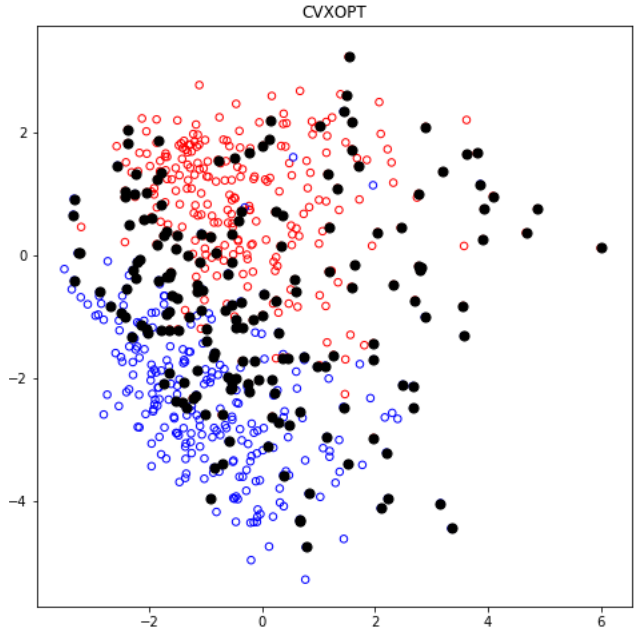


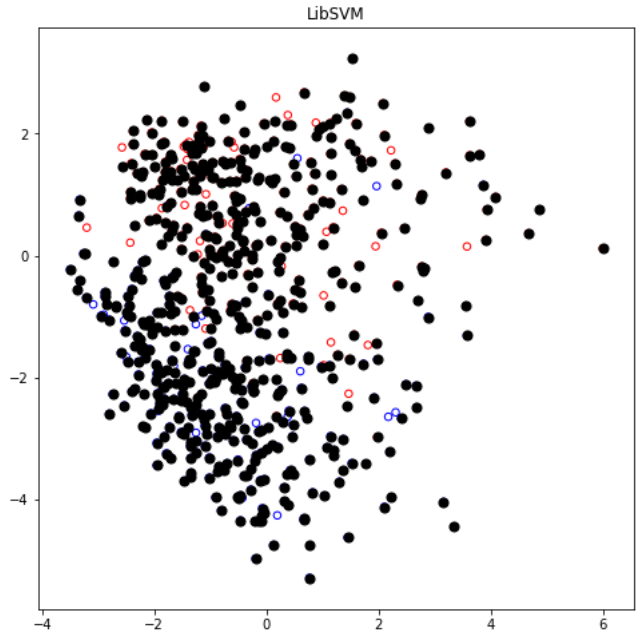
- Classes: (4, 5); All features; Rbf kernel:
  Accuracy from CVXOPT = 0.9316666666666669
  Accuracy from LibSVM = 0.9983333333333334



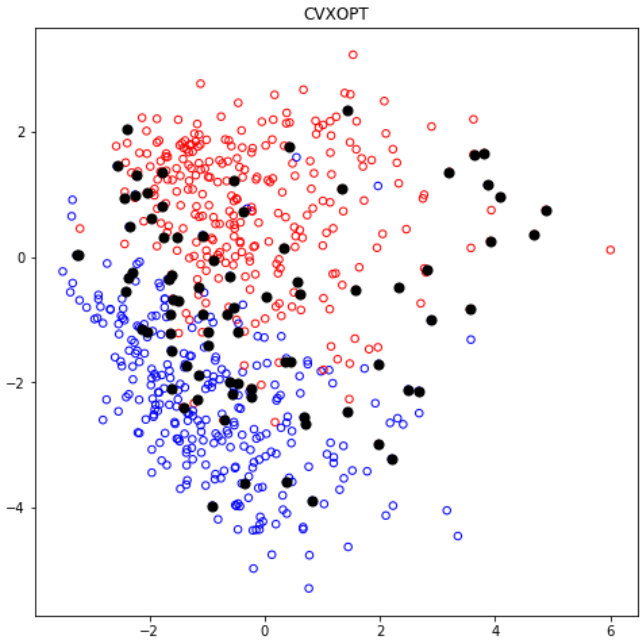


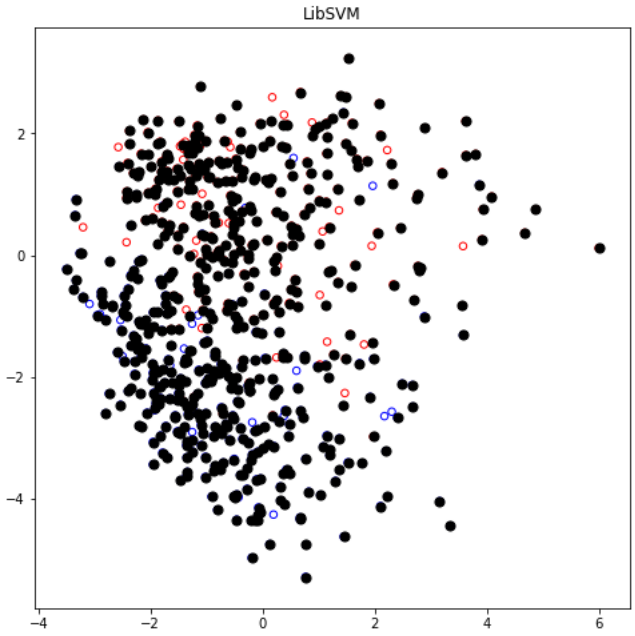
- Classes: (8, 9); First 10 features; Rbf kernel:
  Accuracy from CVXOPT = 0.8900000000000002
  Accuracy from LibSVM = 0.96





- Classes: (8, 9); All features; Rbf kernel:
  Accuracy from CVXOPT = 0.9316666666666669
  Accuracy from LibSVM = 0.9783333333333332





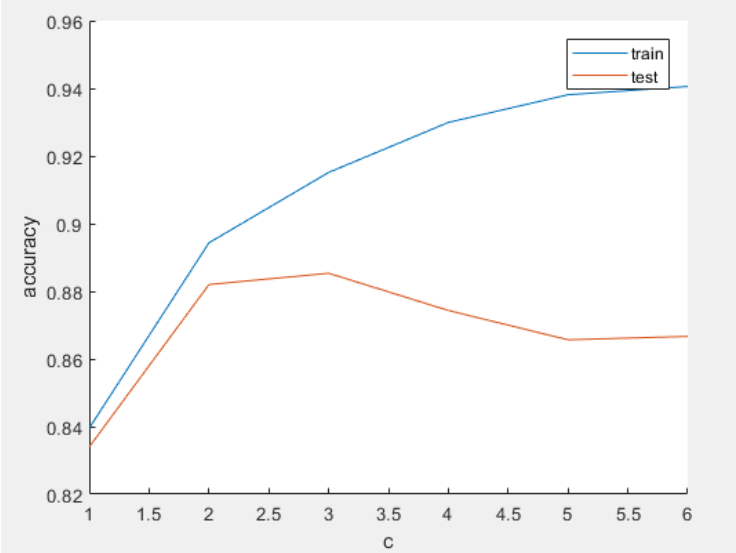
- Results

  In this part, we used an optimisation library CVXOPT to solve our dual problem and build our classifier as described before. We used the tuned hyperparameters derived from above using LibSVM. We compared the accuracy obtained using CVXOPT to that obtained using LibSVM. We also plotted the support vectors for both techniques. Firstly, the accuracy using LibSVM is at the equal to that obtained using CVXOPT, and it also runs faster. But, our CVXOPT being a crude algorithm, performs fairly well for its standards and in a few cases, even matches the accuracy of LibSVM. Seeing the data distribution of class (0, 1), we can clearly see why it performs so well on linear kernels, while the other two classes don't. Again, as in previous times, the accuracy when using all features increases. Another interesting observation is that CVXOPT and LibSVM have much more support vectors using rbf kernel.

Multi-class Classification

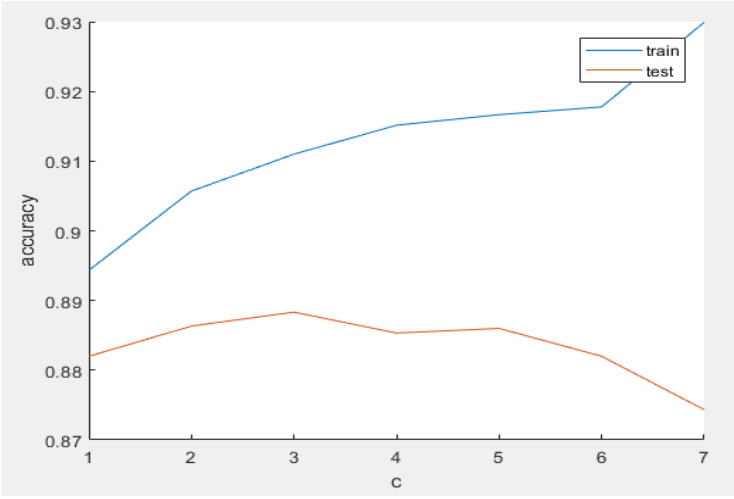
- Linear Kernel

  c = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]



  We see that optimal c = 0.1, where accuracy = 0.8853333333333332.

  Zooming in on between 0.01 and 1.0, we now take our c to be
  c = [0.01, 0.03, 0.06, 0.1, 0.13, 0.16, 1.0]



From this, we now set our optimal c = 0.06, where accuracy = 0.8883333333333333. There has not been a good amount of increase in our accuracy, also, the accuracy score of 0.88833 is not very good, hence we conclude that the data is not linearly distributed and move on to a different kernel.

- Rbf Kernel

  c = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
  Gamma = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0]

  From data, we see that c = 1.0 and gamma = 0.1 are the optimal values, where accuracy = 0.9456666666666667

  We zoom in on the above values of c and gamma, which now are:
  c = [0.1, 0.3, 0.6, 1.0, 3.0, 6.0, 10.0]
  Gamma = [0.01, 0.03, 0.06, 0.1, 0.13, 0.16, 1.0]

  From data, we see that c = 3.0 and gamma = 0.06 are the optimum values, where accuracy = 0.95.

  We zoom in one last time and now our c and gamma are
  c = [1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0]
  gamma = [0.03, 0.035, 0.04, 0.045, 0.05, 0.055, 0.6, 0.065, 0.7, 0.075, 0.08, 0.085, 0.09, 0.095, 0.1]

  From data, we see that c = 1.5 and gamma = 0.065 are the optimum values, where accuracy = 0.9516666666666665. Satisfied, we now move on to polynomial kernel.

- Polynomial kernel

  c = [0.001, 0.01, 0.1, 1.0, 10.0]
  Gamma = [0.0001, 0.001, 0.01, 0.1, 1.0]

d = [1 2 3 4 5]
From data, we see that c = 0.1, gamma = 0.1 and degree = 3 are the optimum values, where accuracy = 0.9359999999999999.

Zooming in,
c = [0.01, 0.03, 0.06, 0.1, 0.13, 0.16, 1.0]
Gamma = [0.01, 0.03, 0.06, 0.1, 0.13, 0.16, 1.0]
d = [3]

From above, we see that c = 0.03, gamma = 0.16 and degree = 3 are the optimum values, where accuracy = 0.937.
But we are still a way off from the 0.95166 accuracy we obtained using the rbf kernel. Hence, we conclude that rbf kernel is the best classifier and the optimum hyperparameters are c = 1.5 and gamma = 0.065.

- First 10 features

  We now use only the first 10 features of the given data. We directly jump to the rbf kernel and tune the hyperparameters for that as rbf is the best classifier for the whole data.

  c = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
  Gamma = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0]

  From data, we see that c = 1.0 and gamma = 0.1 are the optimum values, where accuracy = 0.8946666666666667.

- Results

  We now used the full data and built a classifier for all classes. It is worthwhile noting that LibSVM uses one-vs-one method for multi-class classification. Thus, for 10 classes, it builds 45 classifiers and classifies data. We used linear, rbf and polynomial kernels and tuned hyperparameters for each using cross-validation. We found that rbf kernel gives the best accuracy, with c = 1.5 and gamma = 0.065. Then, we used the first 10 features only, and compared to using all features. The accuracy obtained was quite less than using all the features, thus we conclude that selecting only the first 10 features is not such a good idea.

Part 2

In this part, I basically repeated the things I did in the multiclass classification of Part 1. One important difference this time was that the data this time was uniformly distributed between -700:800, while it was between -2:3 in part 1. As svm depends on the distance between the points, it is important to normalise the data in part 2. I range normalised the data and then proceeded to use it. Following the same procedure as before, zooming in more times than before, the optimal classifier was found out to be rbf classifier with c = 4.0 and gamma = 3.0, where test accuracy = 0.95333.