6.867: HW2

I. INTRODUCTION

When presented with the task of classification, one is subsequently faced with choosing the method to accomplish this task. In this work we present a systematic study of binary classification. In Section 1, classification using logistic regression is presented. In section 2, classification using a quadratic programming solution, alongside the use of linear and gaussian (radial basis function or rbf) kernels, to the Support Vector Machine problem is presented. In section 3, classification using the Pegasos algorithm to minimize the regularized hinge loss is presented. For sections 1-3, the 4 provided data sets (henceforth labelled as 1, 2, 3, 4) are used. Finally in section 4, these approaches are applied to the MNIST handwritten dataset for digit classification.

II. SECTION 1

For logistic regression, we use the following loss function

$$NLL(w, w_0) = \sum_i \log(1 + \exp(-y_i(wx_i + w_0))) \quad (1)$$

where w, w_0 are the weights, x_i is the d-dimensional feature vector for sample i and $y_i \in \{-1, 1\}$ is the corresponding label. To find w, w_0, we use the following objective functions; referred to as L_2 and L_1 regularization respectively:

$$E_{LR}(w, w_0) = NLL(w, w_0) + \lambda \|w\|_2^2 \quad (2)$$

$$E_{LR}(w, w_0) = NLL(w, w_0) + \lambda \|w\|_1 \quad (3)$$

The predictor function is $\delta(wx_i + w_0)$, where δ is the sigmoid function. For L_2 regularization, minimizations are accomplished using the $fmin\_bfgs$ function from the *scipy.optimize* library. Figure 1 shows the norm of the weight vector as function of number of gradient descent iterations, where the weights generally decrease for $\lambda = 1$ as a function of iteration, but blow up for $\lambda = 0$.

Figure 2 compares the decision boundary obtained from L_1 and L_2 regularization, where L_1 has a smaller slope than L_2, consistent with the sparsity that L_1 enforces.

Generally, similar classification performance can be obtained for L_1 and L_2 regularization, given a validated selection of $C = 1/\lambda$ as shown in Figure 3. For L_1, a $C \approx 10$ is suitable, while for L_2, $C \approx 0.1$ is suitable across the datasets.

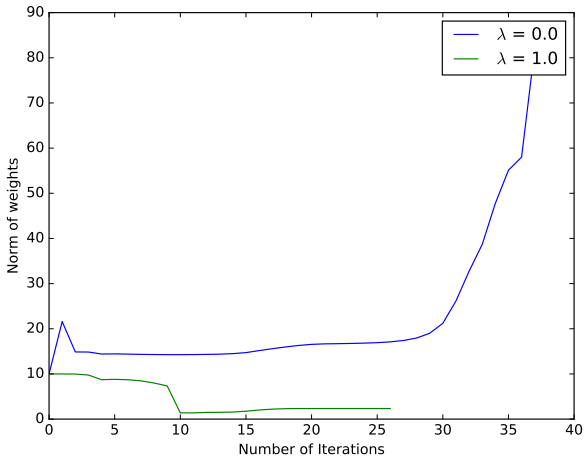
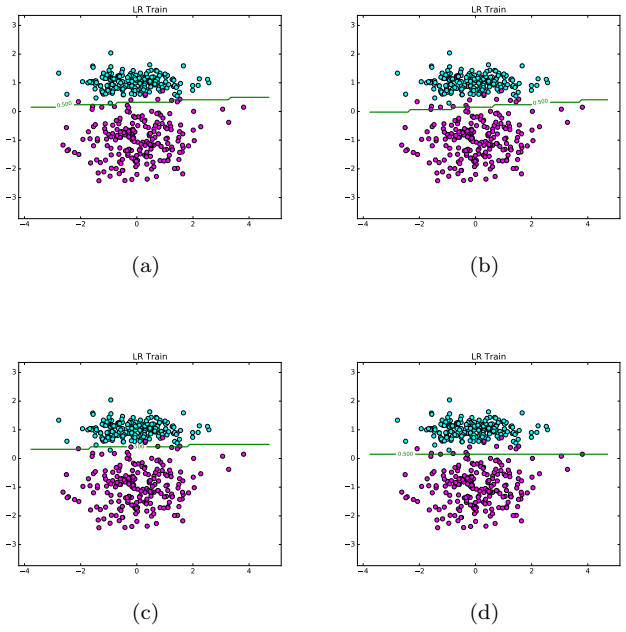


FIG. 1: Number of gradient descent iterations during logistic regression for $\lambda = 0, 1$.

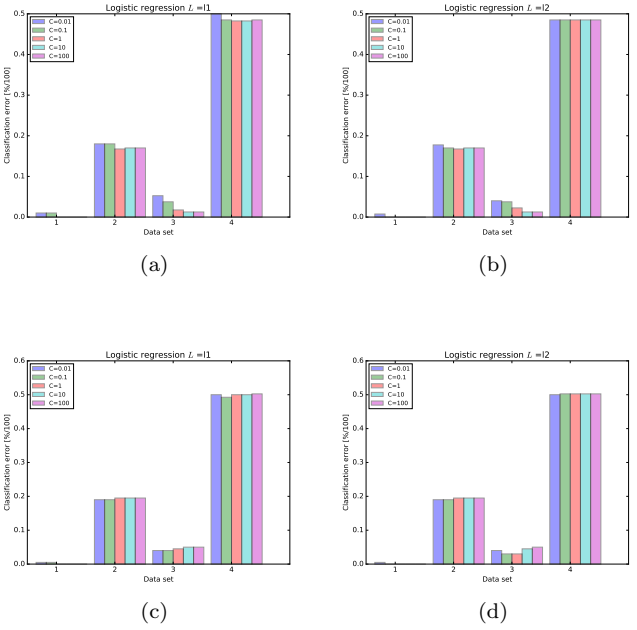


(a) (b) (c) (d)

FIG. 2: Example decision boundaries (a) $L_2, \lambda = 1$, (b) $L_2, \lambda = 0.1$, (a) $L_1, \lambda = 1$, (b) $L_1, \lambda = 0.1$

III. SECTION 2

We now move on to using quadratic programming to maximize the margin of an SVM classification problem. In the dual framework, the optimization problem is equivalent to the following:



(a) (b) (c) (d)

FIG. 3: Classification performance for the 4 test sets as a function of $C = 1/\lambda$.

$$\begin{aligned} \max_{\alpha} \quad & \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \boldsymbol{x}_n^T \boldsymbol{x}_m \\ \text{s.t.} \quad & 0 \leq \alpha_n \leq C \; \forall n \in N \\ & \sum_{n=1}^{N} \alpha_n y_n = 0 \end{aligned} \tag{4}$$

Here C is the hyperparameter (effectively analogous to the λ in Section 1). Mapping this problem back to the original SVM description, we can assign significance to the values of α as follows:

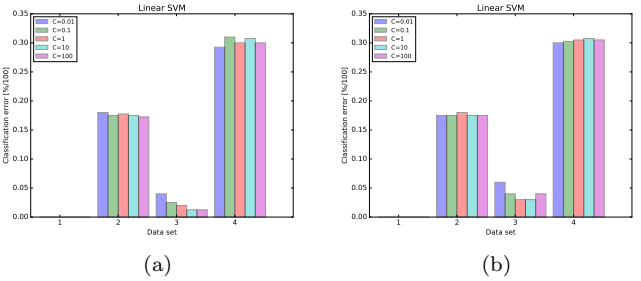
$$\begin{aligned} \alpha_i &= 0 && \text{Correctly classified} \\ \alpha_i &= C && \text{Margin error} \\ 0 < \alpha_i &< C && \text{Support vector} \end{aligned} \tag{5}$$

Example boundaries for $C = 1$ using a linear kernel, defined as $K_{linear}(x, x') = \boldsymbol{x}^T \boldsymbol{x'}$ are shown in at the end of this document in Figure. Classification performance is shown in Figure 4.

Before moving forward, the following kernel definition is needed. The Gaussian (or rbf) kernel is defined as follows:

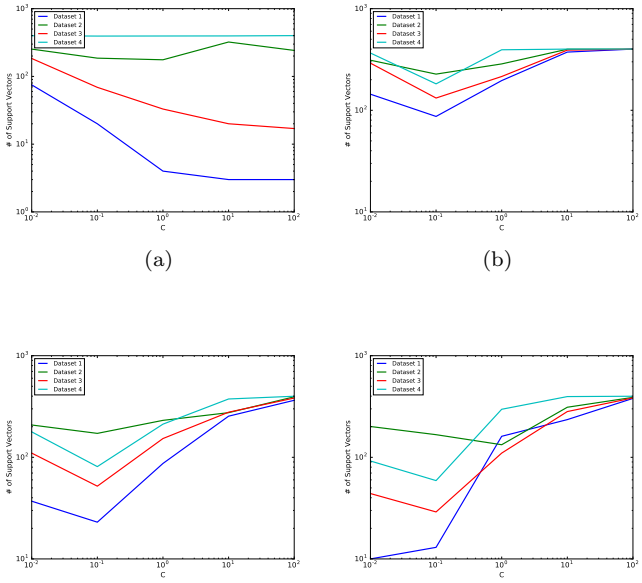
$$K_{rbf}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma \|\boldsymbol{x} - \boldsymbol{x}'\|^2\right) \tag{6}$$

Figure 5 shows the number of support vectors as a function of C, for both linear and rbf kernels. We see that



(a) (b)

FIG. 4: Classification performance for the 4 (a) test and (b) validation sets as a function of C.



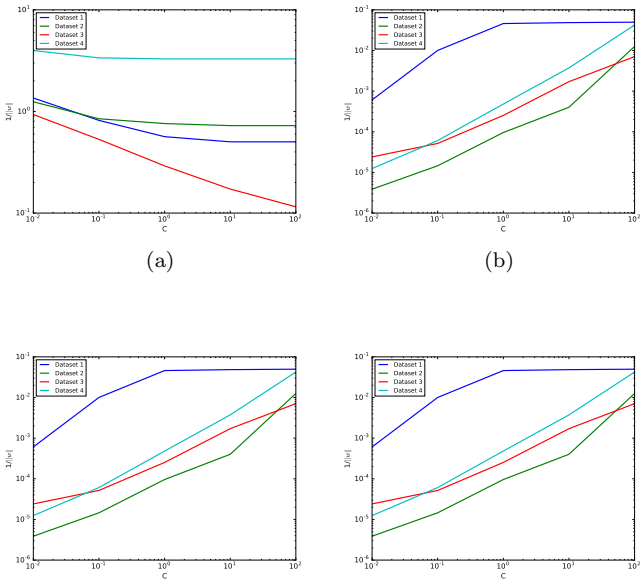
(a) (b) (c) (d)

FIG. 5: Number of support vectors for the 4 test sets as a function of C for QP SVM.

while for the linear kernel, the number of support vectors decreases as a function of C for the linearly separable case, the number of support vectors will increase with C for all examined datasets for the rbf kernel, regardless of the value of γ.

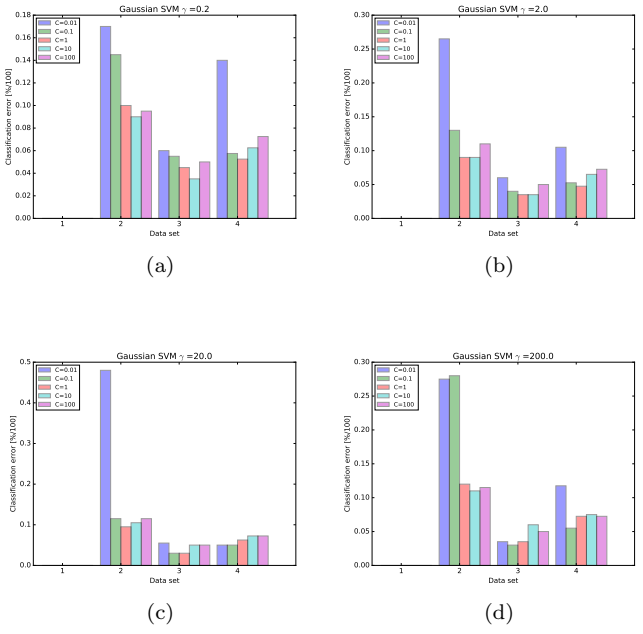
The size of the margin $(1/\|w\|)$ as a function of C is reported in Figure 6. We observe that a large value of C effectively penalizes the weight vector for the Gaussian kernel case. The margin cannot simply be used as a good measure to obtain C because we run the risk of overfitting.

This overfitting can be observed in Figure 7, which presents the classification performance as a function of C on the test sets. The use of tests sets elucidates this overfitting as large values of C show an uptick in the classification error.



(a) (b) (c) (d)

FIG. 6: $1/\|w\|$ for the 4 test sets as a function of C.

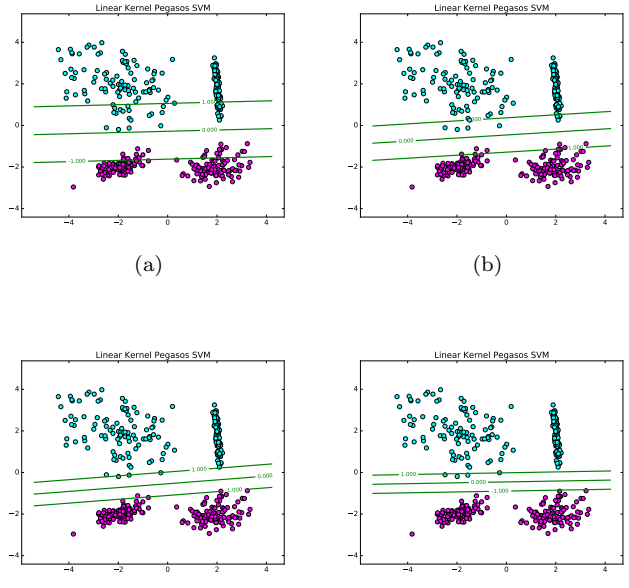


(a) (b) (c) (d)

FIG. 7: Classification performance for the 4 test sets as a function of C.

IV. SECTION 3

There is an alternative to the quadratic programming approach known as the Pegasos algorithm. In this approach, a stochastic like gradient descent scheme is used to find the minimum of an objective function. In our



(a) (b) (c) (d)

FIG. 8: Classification performance for the 4 test sets as a function of λ = (a) 0.2, (b) 0.02, (c) 0.002, (d) 0.0002.

case, we use the hinge loss, defined as:

$$HingeLoss(w, w_0) = \sum_i \max(0, 1 - y_i(wx_i + w_0)) \quad (7)$$

Thus our objective function is:

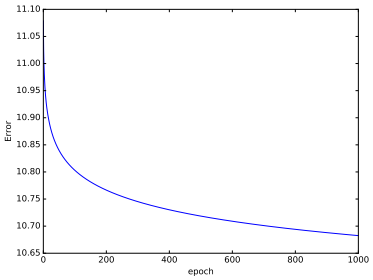
$$E_{pegasos}(w, w_0) = \frac{1}{n}\sum_i \max(0, 1 - y_i(w^T x_i)) + \frac{\lambda}{2}\|w\|_2^2 \quad (8)$$

The clever insight in the Pegasos algorithm is to observe that the SDG can be efficiently calculated. The algorithm is sketched below:

Require: (x_i, y_i), λ, $maxepoch$
  $t \leftarrow 0, w_{t=0} \leftarrow 0$
  while $epoch < maxepoch$ **do**
    for i = 1, ..., n **do**
      $t \leftarrow t + 1$
      $\eta_t \leftarrow 1/(t\lambda)$
      if $y_i(w_t^T x_i < 1$ **then**
        $w_{t=1} \leftarrow (1 - \eta_t\lambda)w_t + \eta y_i x_i$
      else
        $w_{t=1} \leftarrow (1 - \eta_t\lambda)w_t$
      end if
    end for
  end while

Figure 8 shows some example decision boundaries obtained from the Pegasos algorithm at different λ, where increasing λ increases the margin, via decreasing norm of the weight vector.

The pegasos algorithm is easily extended to include arbitrary kernels as follows:



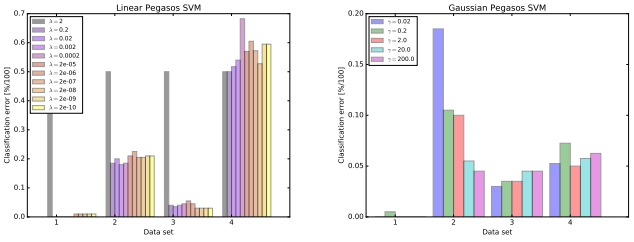
(a)

FIG. 9: Example of decision boundaries for Gaussian kernels with the Pegasos algorithm.

Require: $(x_i,\, y_i)$, λ, K, $maxepoch$
  $t \leftarrow 0, \alpha_{t=0} \leftarrow 0$
  while $epoch < maxepoch$ **do**
    for i = 1, ..., n **do**
      $t \leftarrow t + 1$
      $\eta_t \leftarrow 1/(t\lambda)$
      if $y_i(\sum_j \alpha_j K(x_j, x_i) < 1$ **then**
        $\alpha_{t=1} \leftarrow (1 - \eta_t\lambda)\alpha_t + \eta y_i x_i$
      else
        $\alpha_{t=1} \leftarrow (1 - \eta_t\lambda)\alpha_t$
      end if
    end for
  end while

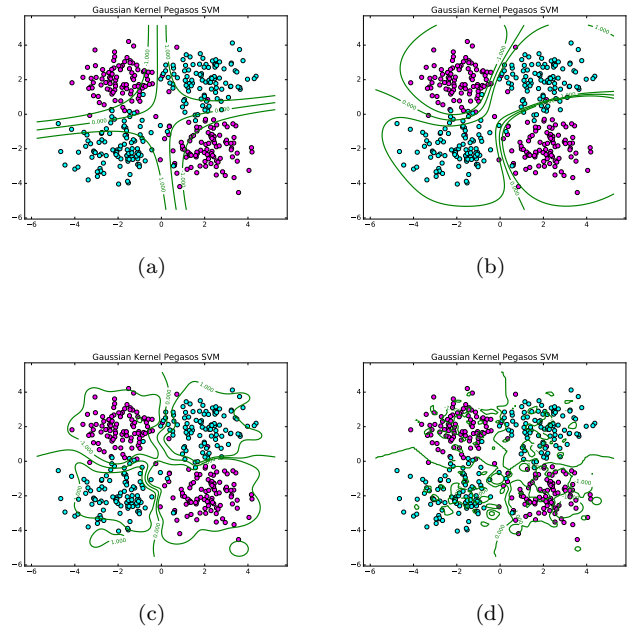
An example of the performance of the pegasos algorithm is shown in Figure 9. The prediction function in for the Gaussian kernel is $y(x) = \sum_i \alpha_i K(x_i, x)$, which is different that the prediction used in section, $y(x) = \sum_i \alpha_i y_i K(x_i, x)$

The number of support vectors obtained the linear and Gaussian pegasos algorithm are shown in Figure 11 (keep in mind $C = 1/\lambda$ so the trend is consistent with QR). In Figure 12, we see that we can obtain comparable classification performance to the QR-SVM, where we run into similar overfitting challenges.



(a) (b)

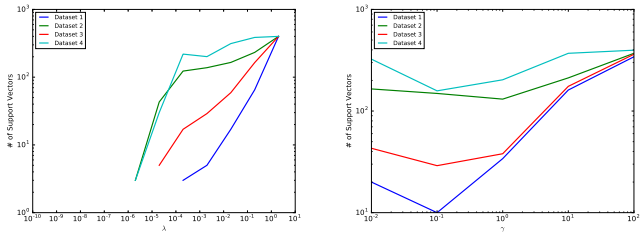
FIG. 12: Classification performance for the 4 test sets for Linear and Gaussian kernels with the Pegasos algorithm.



(a) (b)

(c) (d)

FIG. 10: Example of decision boundaries for Gaussian kernels with the Pegasos algorithm.

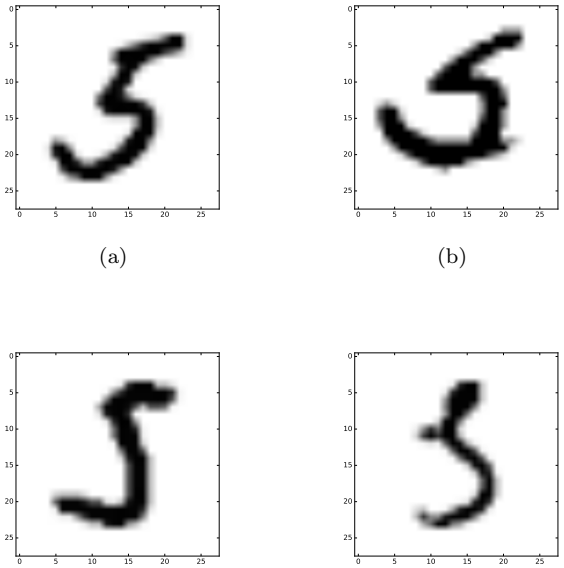


(a) (b)

FIG. 11: Number of support vectors for the 4 test sets as a function of hyperparameter.

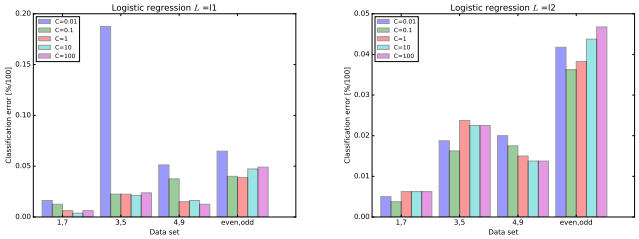
V. SECTION 4

The above approaches can be applied to other classification problems where the input vector has a larger dimension. Here we test these methods on digit classification. First, logistic regression is used. Some examples of misclassified digits are shown in Figure 13 and to the author's eye, we can forgive logistic regression for failing with these samples. A human would have a difficult time correctly classifying these images.



(a) (b) (c) (d)

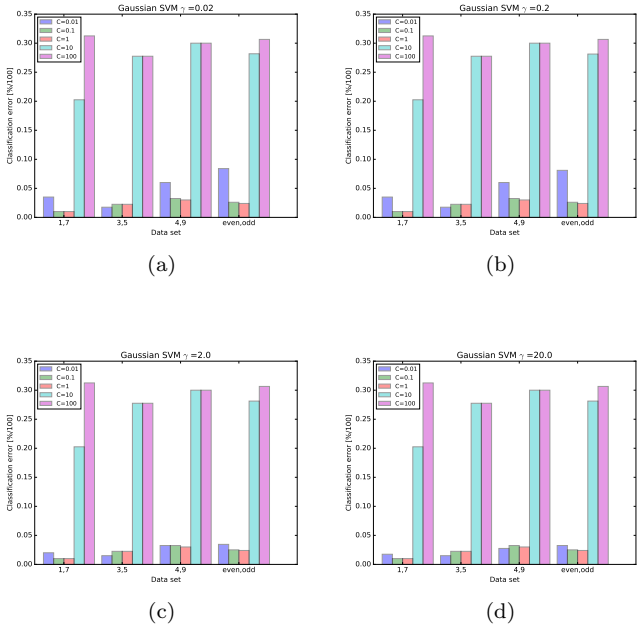
FIG. 13: Some misclassified images from logistic regression.



(a) (b)

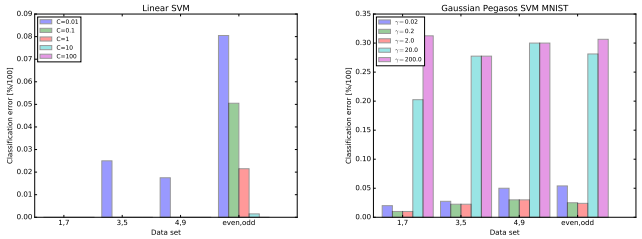
FIG. 14: Classification performance for the 4 (a) test and (b) validation sets as a function of C.

The linear kernel from the quadratic SVM is found to have remarkable performance. The rbf kernel when used in either the pegasus algorithm or the quadratic SVM gives comparable performance. Excluding the logistic regression or the linear QP SVM approaches for a moment, if one were to choose between the Gaussian QP SVM and the Gaussian pegasos approaches, computational cost will be the deciding factor. The pegasus algorithm scales as $O(maxepochs * N)$ while QP is (in the most general sense, NP-hard) is $O(N^k)$ where is k 3. Thus the pegasos algorithm is selected for large training sets.



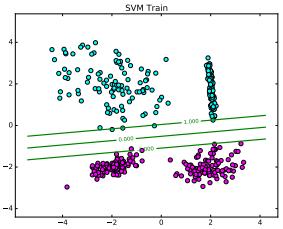
(a) (b) (c) (d)

FIG. 15: Classification performance on MNIST dataset using QP SVM with rbf kernel.

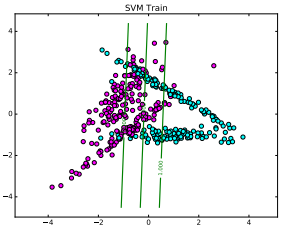


(a) (b)

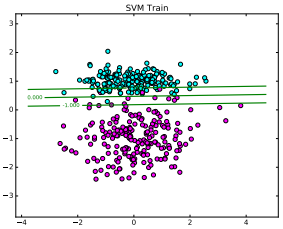
FIG. 16: Classification performance for the 4 (a) test and (b) validation sets as a function of C.



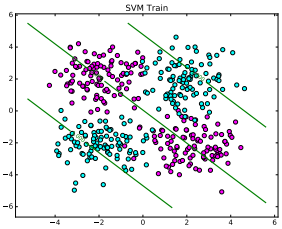
(a) Linear QP SVM Training set 1, C=1



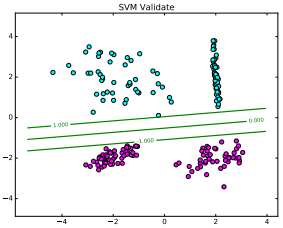
(b) Linear QP SVM Training set 2, C=1



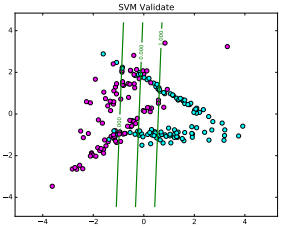
(c) Linear QP SVM Training set 3, C=1



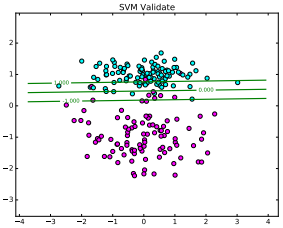
(d) Linear QP SVM Training set 4, C=1



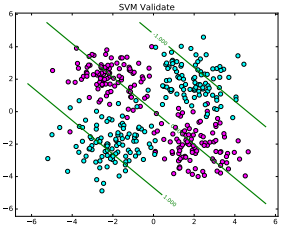
(e) Linear QP SVM Validation set 1, C=1



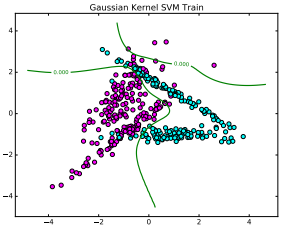
(f) Linear QP SVM Validation set 2, C=1



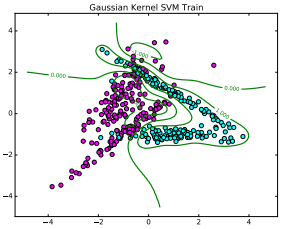
(g) Linear QP SVM Validation set 3, C=1



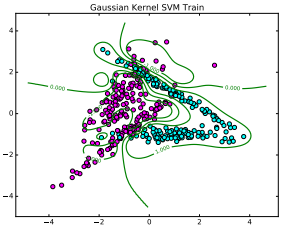
(h) Linear QP SVM Validation set 4, C=1



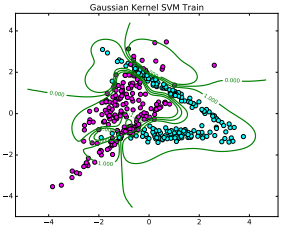
(i) Gaussian QP SVM Training set 1, C=0.01



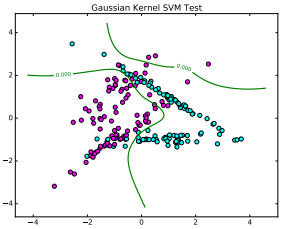
(j) Gaussian QP SVM Training set 1, C=1



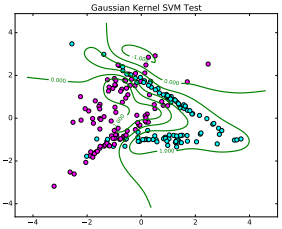
(k) Gaussian QP SVM Training set 1, C=10



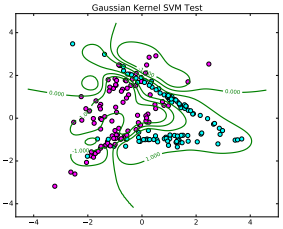
(l) Gaussian QP SVM Training set 1, C=100



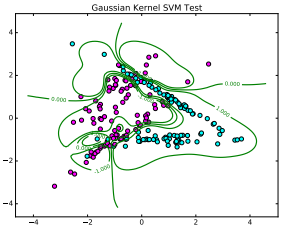
(m) Gaussian QP SVM Validation set 2, C=0.01



(n) Gaussian QP SVM Validation set 2, C=1



(o) GaussianQP SVM Validation set 2, C=10



(p) Gaussian QP SVM Validation set 2, C=100