SVM Classification Implemented with Pegasos Algorithm and Quadratic Programming

Chang Liu, Siyuan Dong

October 23, 2016

1 Logistic Regression (LR)

1.1 Logistic regression with gradient descent

We will use logistic regression as our baseline algorithm for comparison with SVM classifiers. Common to these classification algorithms is the need for optimizating cost functions. For this task, we will use gradient descent algorithm:

Algorithm 1: Gradient Descent

Input: objective function E_{LR}, gradient function ∇E_{LR}, initial w, step size η, convergence criterion c

Output: hyper-parameter w

1: $V_{\text{old}} = \infty$,$V_{\text{new}} = E_{\text{LR}}(w)$
2: **while** $|V_{\text{old}} - V_{\text{new}}| > c$ **do**
3: Set $w = w - \eta * \nabla E_{\text{LR}}$
4: Set $V_{\text{old}} = V_{\text{new}}$
5: Update $V_{\text{new}} = E_{\text{LR}}(w)$
6: **end while**

This is a first-order optimization algorithm which guarantees to find a global minimum for convex functions. Our objective function in logistic regression is as follows:

$$E_L R(w) = \sum_i log(1 + e^{-y^{(i)}(x^{(i)}w + w_0)}) + \lambda w^T w \quad (1)$$

in which the first term stands for the negative log-likelihood, and the second term is the $L2$ penalty to the weights w. This objective function is proven to be convex, and therefore we could use our gradient descent approach to find out the MLE solution. We can derive its 1st order gradient as:

$$\nabla E_L R(w_0) = \sum_i \frac{-y^{(i)}x^{(i)}e^{-y^{(i)}(x^{(i)}w + w_0)}}{1 + e^{-y^{(i)}(x^{(i)}w + w_0)}} \quad (2)$$

$$\nabla E_L R(w) = \sum_i \frac{-y^{(i)}x^{(i)}e^{-y^{(i)}(x^{(i)}w + w_0)}}{1 + e^{-y^{(i)}(x^{(i)}w + w_0)}} + 2\lambda w \quad (3)$$

We test our algorithm with dataset 1, with the step size 0.1 and convergence criterion $c = 10^{-20}$. When λ is 0, there is no penalty to w, the hyper-parameter $w = [11.2, 1.12, 19.9]$. For the case of $\lambda = 1$, the $w = [0.816, -0.006, 2.48]$. Figure 1 shows the weight vector as a function of the number of iteration of gradient descent when λ equals to 0 (left) and 1(right). Because of the penalty of $L2$ norm when $\lambda = 1$, w_1 and w_2 get smaller and smaller as the iteration number increases. However, the absolute value of w_0,w_1 and w_2 increase when we get rid of the penalty term by setting make $\lambda = 0$.

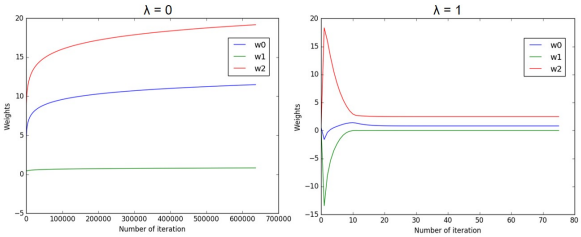

Figure 1: How the three weight vectors evolves through the iteration.

1.2 $L1$ norm vs $L2$ norm

In this section, we would like to compare the effects by using $L2$ norm and $L1$ norm. The $L1$ norm version of objective function in logistic regression is:

$$E_L R(w) = \sum_i log(1 + e^{-y^{(i)}(x^{(i)}w + w_0)}) + \lambda |w| \quad (4)$$

The gradient for updating w changes as well. We used the Sklearn's Logistic Regression package in Python for $L1$ norm to process the same dataset. Figure 2

shows the decision boundary calculated by $L1$ and $L2$ norm LR and the 2-D data points by using three different $\lambda : 10^{-5}, 1, 10^{5}$. Figure 3 shows the weight vectors with different λ. Figure 4 shows the error rate against λ. As λ gets larger, the penalty on w becomes stronger, thus making w_1 and w_2 smaller for both $L1$ and $L2$ norm. However, $L1$ norm tends to make w sparse, so when $\lambda > 10$, w_0 and w_1 both equals to 0. In addition, larger λ increases the error rate, which is clearly shown in Figure 4 and get verified from Figure 2 by observing the data around the decision boundary.

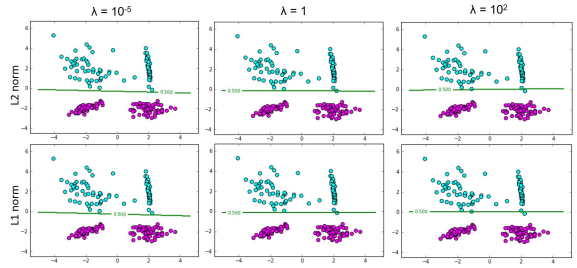

Figure 2: Decision boundary of L1 and L2 norm against λ

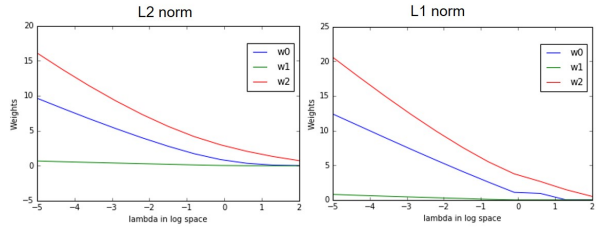

Figure 3: Weight vectors of L1 and L2 norm against λ

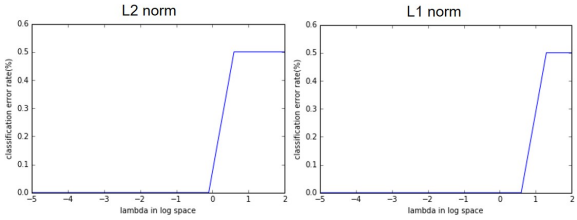

Figure 4: Error rate of L1 and L2 norm against λ

1.3 Model selection

After analyzing different effects of $L2$ norm and $L1$ norm, we would like to train models with different λ and penalizing methods, choose the best model by using validation data. We have 4 datasets and we display the model selection process from dataset 3 as an illustration. 1) Sweep λ, norm and calculate corresponding w. Then calculate the loss using validation data for the w. 2) The minimum loss corresponds to the best model(w). Figure 5 shows the loss verse λ of $L2$ norm and $L1$ norm, and performance on the test sets by using the best model. For dataset 3, the best model is: $w = [-2.741, -0.136, 7.053]$, L1 norm and $\lambda = 1.53$. For dataset 1, the best model is: $w = [9.259, -1.868, 18.228]$, L2 norm and $\lambda = 10^{-5}$. For dataset 2, the best model is: $w = [0.164, 1.749, 0.0008]$, L2 norm and $\lambda = 1.36$. For dataset 4,The best model is: $w = [0., -0.008, -0.007]$, L1 norm and $\lambda = 8.05$. Figure 6 shows the three other datasets and the corresponding decision boundary.

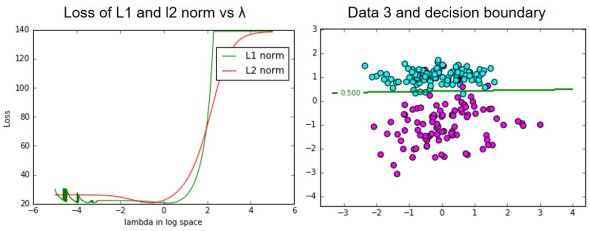

Figure 5: Parameter sweep of λ (LEFT) and decision boundary from the optimal model on dataset 3 (RIGHT)

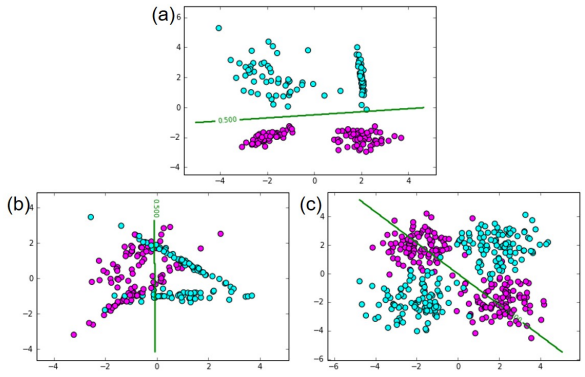

Figure 6: Decision boundary of optimal model on datasets 1, 2, and 4

2 Support Vector Machine (SVM)

2.1 Implement of SVM algorithm

SVM is another very popular algorithm in machine learning for classification and regression. Given any

n data pair of $x^{(i)}$ and $y^{(i)}$, the dual form of soft-SVM is:

$$\begin{aligned} \max_{\alpha} -\frac{1}{2}||\sum_{i=1}\alpha_i y^{(i)} x^{(i)}||^2 \sum_{i=1}\alpha_i \\ s.t. \sum_{i=1}\alpha_i y^{(i)} = 0 \\ 0 \leq \alpha_i \leq C, 1 \leq i \leq n \end{aligned} \tag{5}$$

We used quadratic programming (QP) package called *cvxopt* to solve the optimization problem. To simply test the code, we use four 2D data: positive example $(2,2),(2,3)$ and negative example $(0,-1),(-3,-2)$. Figure 7 shows the decision boundary when $C = 1$.

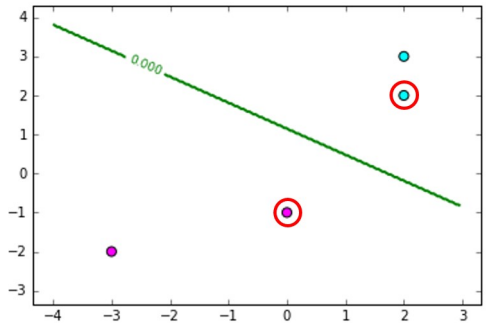

Figure 7: Decision boundary by SVM using QP on sample dataset. Support vector are labeled by red circle.

To further analyze the characteristic of decision boundary of the SVM algorithm, we process four datasets (training and validation from the previous section) when $C = 1$. For illustration, Figure 8 shows the decision boundary given by QP on dataset 2 and dataset 3. The main difference of Soft-SVM from hard-SVM is the tradeoff between margin distance and classification error. We can still get a reasonable linear separator in non-linearly separable data, like dataset 2. For dataset 2, the weight vector is $[1.268, 1.315, -0.042]$. The error rate for training data is 28.25% and 26.50% for validation data, because training data is more complex than validation data. For dataset 3, the weight vector is $[-3.028, -0.048, 3.432]$. The error rate for training data is 14.50% and 17.50% for validation data.

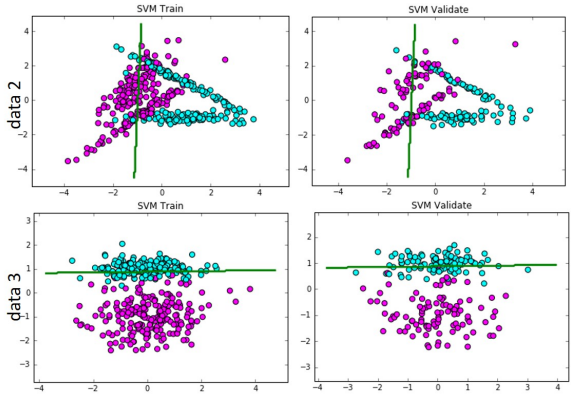

Figure 8: Decision boundary by SVM using QP on datasets 2 and 3

2.2 Kernelized SVM

The dual form SVM is useful for several reasons, including an ability to handle kernel functions that are hard to express as feature functions in the primal form. The kernelized dual SVM can be written in the following form:

$$\arg \max_{0 \leq \alpha_i \leq C} \sum_{i=1}^{n} -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} k(x^{(i)}, x^{(j)}) \tag{6}$$

where $k(x^{(i)}, x^{(j)})$ is a kernel function. In this section, we use linear kernel and Gaussian RBP kernel (with different variance) to process the data. For better representation, dataset 3 is chosen to show the effects of the two kernels and different C.

Firstly, we will use linear kernel. Figure 9 shows the decision boundary when $C = 0.01$ and 100. C is the weight factor that controls how much we want to consider the classification error. Large C will reduce the classification error in training data, and that is why the error rate decreases (right) at first but then it increases as C gets larger and the model overfits in Figure 10. In the left plot in Figure 10, margin $\frac{1}{||w||}$ increases with larger C, which makes sense since we focus on minimizing error instead of maximizing margin distance. However, after C cross some line, $\frac{1}{||w||}$ will saturate as the plot indicates. The number of support vector decreases as C increases. In addition, the decision boundary in Figure 9 shows that when $C = 0.01$, the margin distance is large, but there are more misclassified points. But when $C = 100$, the margin get very smaller but the error rate also get smaller. So we cannot use margin on the training data as criterion for selecting C; rather a combination

of margin and error rate is a good choice of criterion.

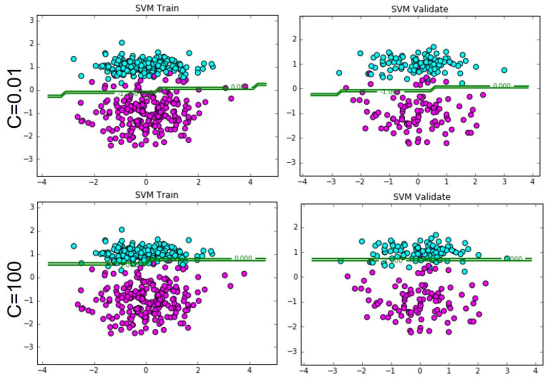

Figure 9: Decision boundary of soft SVM for different C on training and validation set

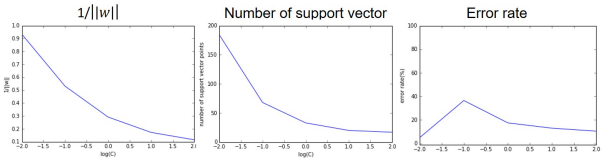

Figure 10: Margin, sparsity of support vectors, and error rate of soft SVM

For the data that cannot be linearly separated, Gaussian RBF is a very popular nonlinear kernel: $k(x,z) = exp(-\gamma||x-z||^2)$. We still use dataset 3 to show the result. Figure 11 shows the decision boundary of Gaussian kernel with $\gamma = 0.25$ and 4 and $C = 0.01$ and 100. From Figure 11, we can draw similar conclusions about parameter C. Larger C will minimize the error, but decrease the margin. Also, larger C makes the decision boundary over-fit the data. The error rate and number of support vectors also decrease when C get larger. Further more, small γ in Gaussian kernel will make the decision boundary smooth. Too large γ will tend to over-fit the data.

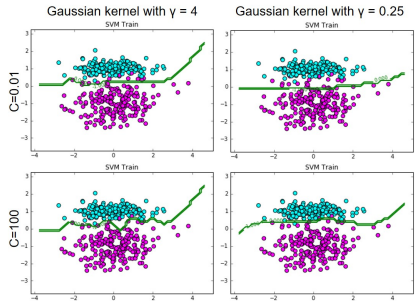

Figure 11: Decision boundary of Gaussian SVM with different γ

3 Pegasos Algorithm

3.1 Soft-SVM with Pegasos

We implement the Pegasos algorithm to train a linear classifier, by maintaining a weight vector w and test it with regularization constants $\lambda = [2^1, ...2^{-10}]$ (recall $C = \frac{1}{n\lambda}$). We selected a few plots of the boundary and margin in Figure 12. Regularization increases with margin, which is consistent with theory that more regularization means more tolerance for error, hence increasing the distance between the decision boundary and the margin boundary.

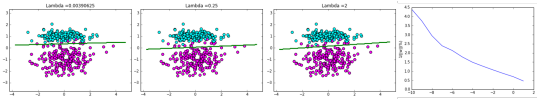

Figure 12: Decision boundary and margin of Pegasos implementation of soft SVM across $C = \frac{1}{n\lambda}$

3.2 Kernelized Soft-SVM with Pegasos

We then implement a kernelized version of the Pegasos algorithm to solve the soft SVM problem. Given α of size N, we can predict the label of a new input X_{new}:

$$Y_{pred} = \sum_j \alpha_j * K(X_{new}, X_j)$$

where $K(X,Z) = \phi(X) \cdot \phi(Z)$ and ϕ. We use both linear and Gaussian kernel. We will also compare Pegasos with the QP solution to the dual SVM problem using the same degree of regularization $C = \frac{1}{n\lambda}$. First, we look at linear classifier. Table 1 shows that 1) Pegasos has comparable performance except for slightly more validation errors and 2) Pegasos has the same sparsity property, in fact, with much fewer support vectors than QP.

	Training Error	Validation Error	Test Error	Number of Support Vectors
Pegasos	4.50%	8.50%	4%	26
QP	4.25%	6.00%	4%	63

Table 1: Performance and sparsity of support vectors in Pegasos and QP implementations of linear classifier. $\lambda = \frac{1}{nC} = 0.02$ with 100 epochs for Pegasos.

3.3 Effect of γ on Classification

We now turn to non-linear Gaussian classifier. To investigate the effect of γ in the Gaussian kernel,

we look various values from 4 to $\frac{1}{4}$. Theoretically, smaller γ has a larger variance which means support vectors can exert influence on more distant points. Figure 13 displays the decision boundary and Figure 14 compares the number of support vectors and performance, as a function of γ . It is clear that larger γ is associated with more over-fitting behavior. Another piece of evidence is that the number of support vectors increases with γ and therefore model complexity. Comparing QP and Pegasos, we see that the QP solution is consistently biased towards the blue points, making more errors than Pegasos. For Pegasos' solution, we see fewer and fewer training errors while validation errors bottoms out and test errors even increases drastically, indicating overfitting, whereas QP seems to have more room for improvement with larger γ. On the other extreme, very small γ under-fits the training data and exhibits higher bias and lower variance.

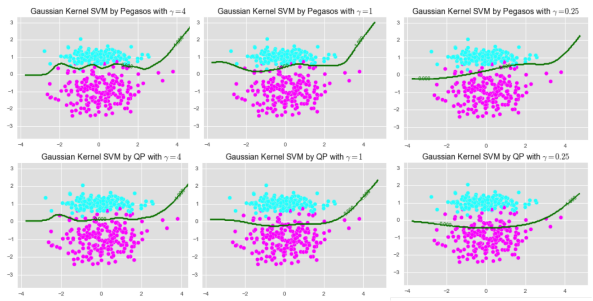

Figure 13: Decision boundary of training set in Pegasos and QP implementations of Gaussian classifier as γ varies. $\lambda = \frac{1}{nC} = 0.02$ with 100 epochs for Pegasos.

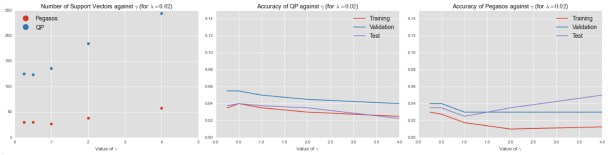

Figure 14: Number of support vectors (out of 400 training points) and performance of Gaussian classifier in Pegasos and QP implementations as γ varies. $\lambda = 0.02$ with 100 epochs for Pegasos.

4 Classifying MNIST data

4.1 Logistic Regression

We implement logistic regression and linear SVM classifier on the MNIST data (with $N = 200$ training samples unless specified otherwise). We pick C from $[10^{-1}, ..., 10^{2}]$ on the validation set. The results are summarized in Table 2. Normalization matters to linear SVM because without normalization, the algorithm outputs no support vector. Yet normalization has no obvious effect on logistic regression. In general, linear SVM performs worse than logistic regression, in particular for the more complicated task (0,2,4,6,8) vs (1,3,5,7,9). The simplest task is 1 vs 7. Some sampled errors were shown in Figure 15. It is indeed quite hard to tell the first and last digit, while middle two digits are obvious, yet the classifiers don't see it.

Tasks	1 vs 7	3 vs 5	4 vs 7	0,2,4,6,8 vs 1,3,5,7,9
Logistic Regression (Norm)	1.3%	5.0%	5.7%	11.1%
Linear SVM (Norm)	1.3%	4.6%	6.3%	16.3%
Logistic Regression	1.3%	4.7%	6.3%	11.2%
Linear SVM	NA	NA	NA	NA

Table 2: Test Errors of Logistic Regression and Linear SVM Classifier for Various Classification Tasks. Norm: model trained with normalized features. NA entries: Linear SVM without normalization gives 0 support vectors.

Figure 15: Decision boundary of Soft SVM

4.2 Tuning Linear and Gaussian SVM Classifier

Table 3 reports the test accuracy of Gaussian RBF SVM classifer using the QP implementation. We pick C from $[10^{-1}, ..., 10^{2}]$ and γ from $[2^{2}, .., 2^{-2}]$ on the validation set. We find Gaussian classifier is very sensitive to normalization of features and performs poorly. However, Gaussian classifier can find support vectors, unlike the linear classifier with unnormalized features which could not. Comparing these methods, we find the Gaussian classifier outperforms all other methods, in particular for the most difficult task, given sufficient tuning of parameters. Moreover, we think linear classifier can hardly be improved in this case because the data are inherently linearly unseparable. There is no amount of wriggle room that is enough to divide this particular data set by a straight line.

Tasks	1 vs 7	3 vs 5	4 vs 7	0,2,4,6,8 vs 1,3,5,7,9
Gaussian SVM (Norm)	1.3%	4.3%	5.0%	8.2%
Gaussian SVM	42.3%	37.6%	39.3%	40.3%

Table 3: Test Errors of Gaussian SVM Classifier for Various Classification Tasks. Norm: model trained with normalized features.

4.3 Pegasos vs QP Implementations

Now we can compare our Pegasos and QP implementations of our Gaussian RBF SVM classifier. Here, we fix λ at 2 (for QP, $C = \frac{1}{n\lambda} = \frac{1}{2n}$) and γ at 4 for direct comparison. Since the comparison also depends on the number of epochs, we terminate the algorithm either when $||\alpha_{i+1} - \alpha_i|| < \epsilon$ where $\epsilon = 10^{-4}$ or when the number of epochs reaches 500. Table 4 compares the running time and the rate of validation errors achieved by these algorithms with respect to the number of training points. Note running time includes training and making predictions with the bottleneck being the costly computation of kernel matrix. As the number of training samples increase, both algorithms reduce validation error by except for the task ('1' vs '7') which has already the lowest errors on all tasks. These correspond to 3 persistent errors, no matter how we tune the algorithm, shown in Figure 16. Moreover, both algorithms achieve almost the same error rate on validation set except for 2 instances. However, the running time for QP is consistently much lower than Pegasos, possibly due to the stringent ϵ and the threshold of 500 epochs.

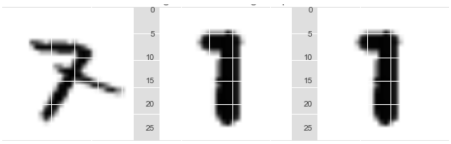

Figure 16: Samples Persistently Misclassified by Gaussian SVM classifer

Given the clear benefit of larger training samples, we aim to select the best combination of parameters for each algorithm and make a final comparison on test data and use various additional tasks. Here, we use $N = 500$ training samples; for λ, we perform a parameter sweep from $[2^1, ...2^{-10}]$ using validation set; the threshold of maximum epochs is set to 1000 for Pegasos. The result is displayed in Table 5. We can see some marginal difference between the two implementations in terms of test errors, without a clear winner, so the results are comparable. We also find that 1) despite a range of regularization constant λ, the validation errors (not shown for brevity) are mostly the same as in the previous experiment for $N = 500$. This perhaps suggests the algorithms converge to the same classifier given large enough training samples. 2) The number of non-zero α_i is 99% to 100% for all classification tasks, which means that there is little structure in the MNIST data that the classifier can focus on, and literally every training sample contributes to the weight vector w and thus how complex the classifier should be. Note, these non-zero α_i, or "support vectors", include training errors and therefore may not lie on the margin. 3) The number of epochs Pegasos took to converge increases as λ decreases until 1000 epochs are met. In other words, the algorithm converges faster with more regularization.
Finally, we compare all of classifiers we analyzed. Clearly, the Gaussian SVM classifier performs on par or slightly better with the the first three easier tasks, but significantly better on the last task, which is the most difficult of all. It also performs well on some additional sophisticated tasks as shown in Table 5.

Classifying 1 vs 7				
N	Pegasos	QP	Pegasos	QP
200	1.00%	1.00%	4.4s	2.0s
300	1.00%	1.00%	7.1s	2.9s
400	1.00%	1.00%	11.9s	4.1s
500	1.00%	1.00%	15.7s	5.3s
Classifying 3 vs 5				
200	6.00%	6.00%	8.2s	1.7s
300	5.67%	5.67%	8.8s	2.8s
400	4.33%	4.33%	13.6s	4.0s
500	4.33%	4.33%	18.4s	5.2s
Classifying 4 vs 7				
200	6.00%	6.00%	6.0s	1.7s
300	6.00%	5.67%	9.3s	2.9s
400	4.33%	4.33%	11.7s	4.0s
500	3.67%	3.67%	18.1s	5.0s
Classifying 0, 2, 4, 6, 8 vs 1, 3, 5, 7, 9				
200	8.67%	8.67%	90.0s	45.3s
300	6.33%	6.33%	154.1s	82.6s
400	5.00%	5.07%	242.4s	118.6s
500	4.60%	4.67%	370.9s	158.3s

Table 4: Validation Errors and Training Time of Pegasos and QP Implementations of Gaussian RBF SVM Classifier. N is the number of samples for each digit. All test sets have 150 samples for each digit. λ is fixed at 2 and γ at 4.

Tasks	1 vs 7	3 vs 5	4 vs 7	0,2,4,6,8 vs 1,3,5,7,9	3,5,7 vs 6,8,9	0,2,7,8 vs 1,3,5,6	0,2,8 vs 3,5,6
Pegasos	1%	4.33%	3.33%	4.44%	4.89%	4.66%	4.44%
QP	1%	4.33%	3.67%	4.56%	4.89%	4.58%	4.44%

Table 5: Test Errors of Pegasos and QP Implementation for Various Classification Tasks. γ is fixed at 4