6.867 Problem Set 2

October 26, 2016

1. Logistic Regression

Assignment description: *Use a gradient descent method to optimize the logistic regression objective, with L2 regularization on the weight vector. For L2 regularization, the objective function should be of the form*

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

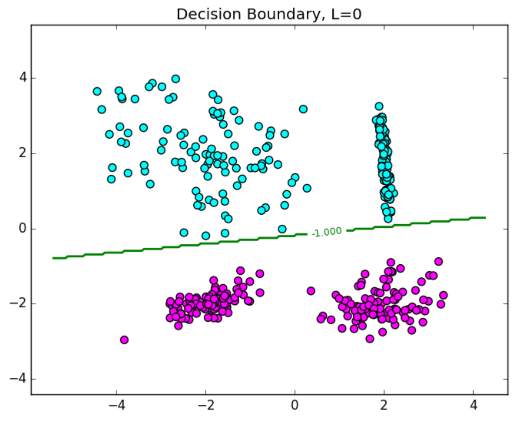
We had previously defined

$$NLL(w, w_0) = \sum_{i=1}^{n} \ln(1 + e^{-y^{(i)}(w^T x^{(i)}) + w_0})$$

1.1 Assignment description: *Run your code on data1 train.csv with $\lambda = 0$. What happens to the weight vector as a function of the number of iterations of gradient descent? What happens when $\lambda = 1$? Explain.*

In order to solve this optimzation problem, we use the gradient descent function developed for the previous pset in order to find optimal weights w, w_0. In order to deal with both L_1 and L_2 errors, as well as the NLL function, we use the numerical gradient in order to compute the gradient at a given vector (w, w_0). This numerical gradient was also developed for the previous pset.

When $\lambda = 0$, As we iterate through the gradient descent, the weight vector stabilizes in the first 50 iterations to $(w_0 = 6.07, w_1 = -4.3, w_2 = 38.6)$. When $\lambda = 1$, the solution vector fluctuates between two solutions: $(w_0 = 1.35, w_1 = 0.4, w_2 = 3.44)$ and $(w_0 = 1.35, w_1 = -0.7, w_2 = 3.28)$. Both of them are acceptable and we see that the norm of both is smaller than the one coming from the unconstrained regression. Decision boundaries for a solution of $\lambda = 0$ and $\lambda = 1$ are shown below.



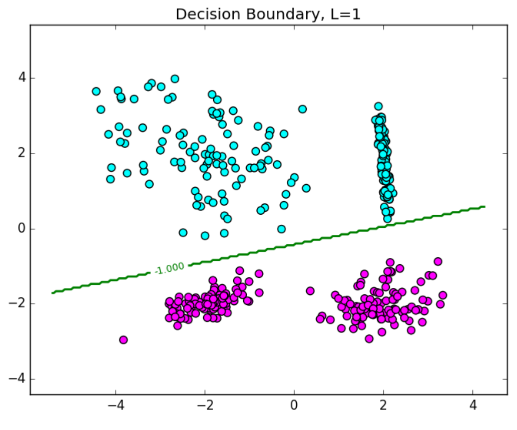


Figure 1: Decision boundaries for lambda =0 (left) and lambda =1 (right)

1.2 Assignment description: *Let's now compare the effects of L1 and L2 regularization on LR. Evaluate the effect of the choice of regularizer (L1 vs L2) and the value of λ on (a) the weights, (b) the decision boundary and (c) the classification error rate in each of the training data sets.*

The tables presented show summaries for each of the four datasets, in order. We observe the usual Ridge and Lasso effects, namely that as the penalty increases, some of the weights in the Lasso model become zero, and that the Ridge weights decrease but do not become zero. Also note that datasets 1, 2, 3 are well estimated by this model, while dataset 4 is not. In these tables we can also comment on the risk of overfitting, if we look at specific parameters with low reaining error but high validation error. However, there are few such cases.

We can observe that the decision boundary, graphically, does not change much with the value of the parameter, except for the Lasso effect where one of the weights becomes one, so the decision boundary passes through the origin.

Regularizer	Lambda	Decision Boundary	Training Error Rate	Validation Error Rate
L1	0.01	$-0.96x_1 + 10.33x_2 + 5 = 0$	0	0
L1	0.1	$-0.60x_1 + 7.23x_2 + 3.35 = 0$	0	0
L1	1	$-0.18x_1 + 4.36x_2 + 0.06 = 0$	0	0.
L1	10	$2.43x_2 + 0.06 = 0$	0.01	0
L2	0.01	$-0.67x_1 + 6.96x_2 + 3.38 = 0$	0	0
L2	0.1	$-0.37x_1 + 4.67x_2 + 1.97 = 0$	0	0
L2	1	$-0.11x_1 + 2.89x_2 + 0.84 = 0$	0	0
L2	10	$-0.02x_1 - 1.67x_2 + 0.21 = 0$	0.025	0

Regularizer	Lambda	Decision Boundary	Training Error Rate	Validation Error Rate
L1	0.01	$1.82x_1 + 0.002x_2 + 0.18 = 0$	0.17	0.17
L1	0.1	$1.82x_1 + 0.001x_2 + 0.18 = 0$	0.17	0.17
L1	1	$1.78x_1 + 0.15 = 0$	0.16	0.17
L1	10	$1.52x_2 + 0.05 = 0$	0.18	0.18
L2	0.01	$1.81x_1 + 0.002x_2 + 0.18 = 0$	0.17	0.175
L2	0.1	$1.81x_1 + 0.002x_2 + 0.17 = 0$	0.17	0.17
L2	1	$1.71x_1 + 0.0002x_2 + 0.15 = 0$	0.16	0.17
L2	10	$1.24x_1 - 0.008x_2 + 0.06 = 0$	0.17	0.18

Regularizer	Lambda	Decision Boundary	Training Error Rate	Validation Error Rate
L1	0.01	$-0.45x_1 + 13.41x_2 - 6.5 = 0$	0.01	0.03
L1	0.1	$-0.4x_1 + 12x_2 - 5.86 = 0$	0.01	0.03
L1	1	$-0.18x_1 + 7.673.38x_2 - 3.14 = 0$	0.01	0.03
L1	10	$3.38x_2 - 0.44 = 0$	0.03	0.05
L2	0.01	$-0.4x_1 + 11.51x_2 - 5.40 = 0$	0.0125	0.03
L2	0.1	$-0.26x_1 + 7.42x_2 - 3 = 0$	0.0175	0.035
L2	1	$-0.19x_1 + 4.02x_2 - 1 = 0$	0.025	0.035
L2	10	$-0.11x_1 - 1.95x_2 - 0.22 = 0$	0.04	0.06

Regularizer	Lambda	Decision Boundary	Training Error Rate	Validation Error Rate
L1	0.01	$-0.02x_1 - 0.02x_2 = 0$	0.48	0.50
L1	0.1	$-0.02x_1 + 0.02x_2 = 0$	0.48	0.50
L1	1	$-0.02x_1 + 0.02x_2 = 0$	0.48	0.50
L1	10	$-0.03x_1 - 0.03x_2 = 0$	0.48	0.51
L1	0.01	$-0.02x_1 - 0.02x_2 = 0$	0.48	0.50
L1	0.1	$-0.02x_1 + 0.02x_2 = 0$	0.48	0.50
L1	1	$-0.02x_1 + 0.02x_2 = 0$	0.48	0.50
L1	10	$-0.02x_1 - 0.02x_2 = 0$	0.48	0.51

1.3 Assignment description: *Use the training and validation sets to pick the best regularizer and value of λ for each data set: data1, data2, data3, data4. Report the performance on the test sets.*

For datasets $1, 2, 3$, we find that the L1 regularizer with $\lambda = 0.01$ gives the lowest validation error rate. We also tried $\lambda = 0.02$ and got the same results. Note that for dataset 4, al our models do equally bad. Still, by a slight margin, the lowest validation error rate is still with $L1$ and $\lambda = 0.01$.

The test error rates for each of the four datasets are, in order, $0, 0.19, 0.05$ and 0.5. Note that these are slightly higher than the training and validation errors.

2. Support Vector Machines

2.1 Assignment description: *Implement the dual form of linear SVMs with slack variables. Please do not use the built-in SVM implementation in Matlab or sklearn. Instead, write a program that takes data as input, converts it to the appropriate objective function and constraints, and then calls a quadratic programming package to solve it. See the file optimizers.txt for installation and usage for matlab/python.*

We write a program that takes the following as inputs

1. n one-dimensional values $Y = (y^{(1)}, y^{(2)}, \dots, y^{(n)})$, where each $y^{(i)} \in \{-1, +1\}$
2. n D - dimensional vectors $X = (x^{(1)}, x^{(2)}, \dots, x^{(n)})$, where each $x^{(i)} \in \mathbb{R}^D$
3. A parameter C
4. A kernel function $K : \mathbb{R}^D \times \mathbb{R}^D \mapsto \mathbb{R}$

and outputs weights $w \in \mathbb{R}^D, w_0 \in \mathbb{R}$, with the following properties (standard for SVMs) :

1. There exists a function $\phi : \mathbb{R}^D \mapsto \mathbb{R}^\infty$ and an inner product $\langle , \rangle$ such that $K(x, x') = \langle \phi(x), \phi(x') \rangle$. We write $\mathbb{R}^\infty$ to represent a finite, or infinite-dimensional real vector space.
2. w can be written as $w = \sum_{i=1}^{n} \alpha_i y^{(i)} \phi(x^{(i)})$, for $\alpha_1, \alpha_2, \dots \alpha_n \in \mathbb{R}$, $0 \le \alpha_i \le C$ for all $i \in \{1, 2, \dots n\}$
3. The parameters α_i solve the optimization problem $\min_{\alpha_1, \alpha_2, \dots \alpha_n} \|\frac{1}{2} \sum_{i=1}^{n} \alpha_i y^{(i)} \phi(x^{(i)})\|^2 - \sum_{i=1}^{n} \alpha_i$ subject to $0 \le \alpha_i \le C$ for all $i \in \{1, 2 \dots n\}$ and $\sum_{i=1}^{n} \alpha_i y^{(i)} = 0$

In order to solve this optimization problem, we use the cvxopt package in python, which allows us to solve the following problem:

$$max_\alpha \frac{1}{2} \alpha^T P \alpha + q^T \alpha \text{ such that } G\alpha \le h, Ax = b.$$

In our problem, solving the C-SVM objective is equivalent to solving the above optimization, with the following input values:

1. $P = K$, where $K_{(ij)} = y^{(i)} y^{(j)} \langle \phi(x^{(i)}, x^{(j)} \rangle$.
2. q is a $n \times 1$ vector of -1s.
3. G is a $2n \times n$ matrix, where the first n rows (and the n columns) represent the matrix $-I_n$, and the next n rows (and the n columns) represent the matrix I_n. Note that I_n is the identity matrix in n dimensions.
4. h is a $2n \times 1$ vector, where the first n rows are 0, and the last n rows are C.
5. A is a $1 \times n$ vector, $A = (y^{(1)}, y^{(2)}, \dots y^{(n)})$.
6. b is a real number, $b = 0$.

Assignment description: *Show in your report the constraints and objective that you generate for the 2D problem with positive examples (2, 2), (2, 3) and negative examples (0, -1), (-3, -2). Which examples are support vectors?*

In this case we use the regular kernel $K(x,x') = x^T x'$ (simply the dot product between two D-dimensional vectors), our inputs are $x^{(1)} = (2,2), x^{(2)} = (2,3), x^{(3)} = (0,-1), x^{(4)} = (-3,-2)$ and $(y^{(1)}, y^{(2)}, y^{(3)}, y^{(4)}) = (+1,+1,-1,-1)$, and the optimization problem is

$$\min_{\alpha_1,\alpha_2,\alpha_3,\alpha_4} \frac{1}{2} \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \end{bmatrix} \begin{bmatrix} 8 & 10 & 2 & 10 \\ 10 & 13 & 3 & 12 \\ 2 & 2 & 1 & 2 \\ 10 & 12 & 2 & 13 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} + \begin{bmatrix} -1 & -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix}$$

such that

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ C \\ C \\ C \\ C \end{bmatrix},$$

$$\begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = 0$$

The optimal values which we find are $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) = (0.153, 0, 0.153, 0)$. The support vectors are α_1 and α_3.

2.2. Assignment description: *Test your implementation on the 2D datasets. Set C=1 and report/explain your decision boundary and classification error rate on the training and validation sets.*

The results are below. The decision boundary represents the equation that a 2D point $x = (x_1, x_2)$ satisfies in order that the classifier is indiferent between assigning + or − to that prediction.

Dataset ID	Decision Boundary	Training Error Rate	Validation Error Rate
1	$-0.18x_1 + 1.76x_2 - 0.67 = 0$	0.02	0.01
2	$1.31x_1 - 0.04x_2 - 0.37 = 0$	0.2	0.17
3	$-0.04x_1 + 3.43x_2 - 0.39 = 0$	0.03	0.05
4	$-0.21x_1 - 0.21x_2 - 6.52 = 0$	0.5	0.5

Notice that out model does poorly on dataset 4 (*data_4_test* and *data_4_validate*).

2.3 Assignment description: *Extend your dual form SVM code to operate with kernels. Explore the effects of choosing values of* $C \in \{0.01, 0.1, 1, 10, 100\}$ *on linear kernels and Gaussian RBF kernels as the bandwidth is also varied.Report your results and answer the following questions:*

(a) Assignment description: *What happens to the geometric margin* $\frac{1}{\|w\|}$ *as C increases? Will this always happen as we increase C?*

(b) Assignment description: *What happens to the number of support vectors as C increases?*

For each of the four training datasets, we train SVM models with errors $C \in \{0.01, 0.1, 1, 10, 100\}$, for the following four kernels: (linear kernel, Gaussian RBF with $\gamma = 0.01$, Gaussian RBF with $\gamma = 0.1$, Gaussian RBF with $\gamma = 1$). We plot the margin and the number of support vectors for each of these models.

Notice that the number of support vector drops as a function of C, and then does not change between $C = 10$ and $C = 100$. Also notice that the margin decreases as a function of C. This makes sense- we penalize slack variables more as we increase C. The number of support vectors is small for the linear kernel function, and large for the Gaussian RBF. Finally, the linear kernel predictor has the lowest margin, but in the validation test the Gaussian RBF functions have a lower error rate.

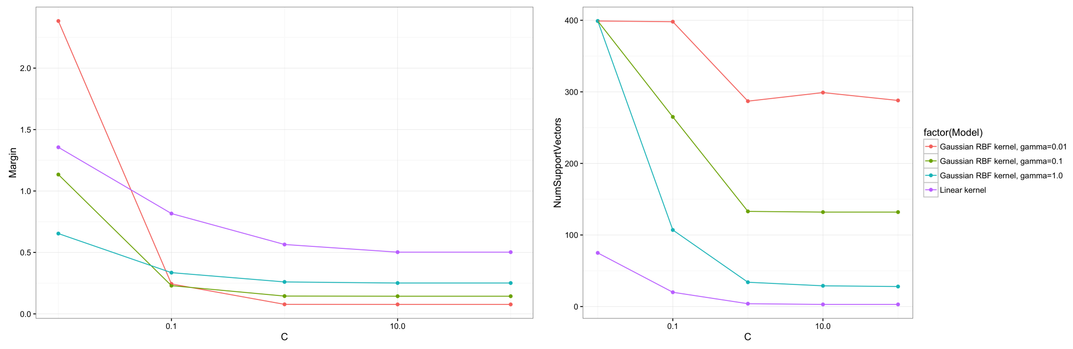


Figure 2: Margin and number of support vectors for Dataset 1

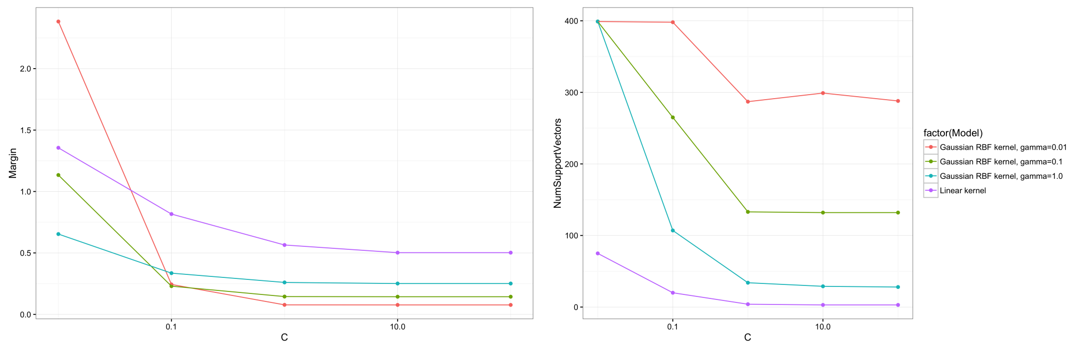


Figure 3: Margin and number of support vectors for Dataset 2

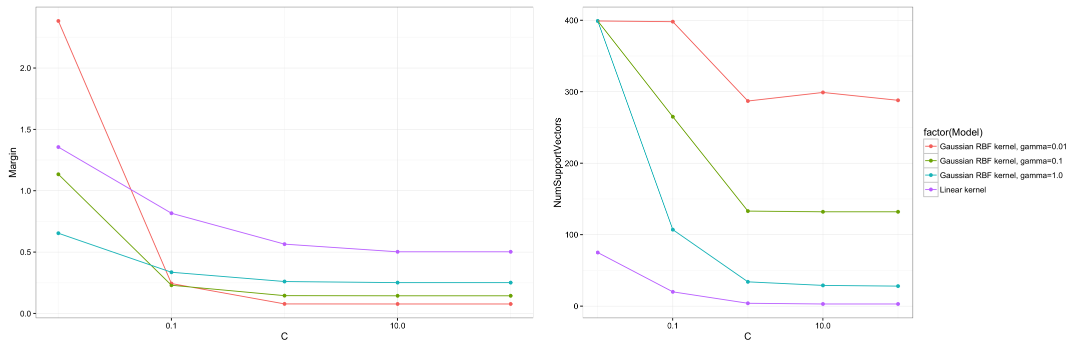


Figure 4: Margin and number of support vectors for Dataset 3

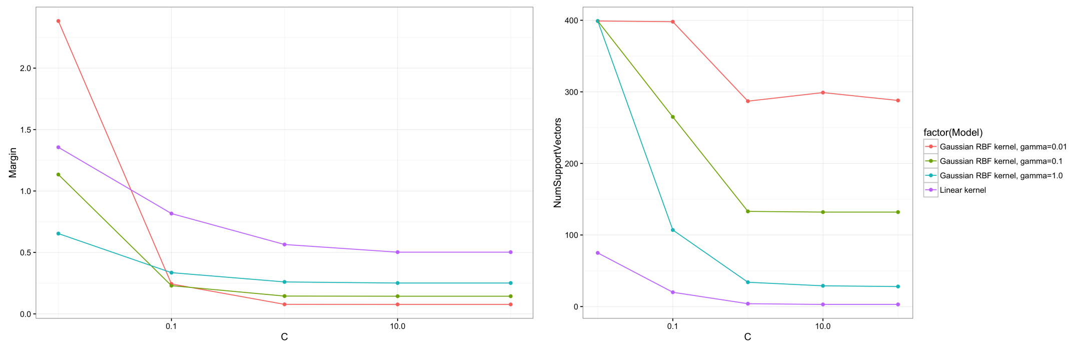


Figure 5: Margin and number of support vectors for Dataset 4

(c) Assignment description: *The value of C will typically change the resulting classifier and therefore also* *affects the accuracy on test examples. Why would maximizing the geometric margin $\frac{1}{\|w\|}$ on the training set not be an appropriate criterion for selecting C? Is there an alternative criterion that we could use for this purpose?*

Maximizing the geometric margin on the training set could lead to overfitting, and this objective is not speficially designed in order to optimize classification error on unseen data. Moreover, we saw in part *b*) that the margin does not change when C is set high enough. If we care about the accuracy of our classifier on unseen data, then we should choose the parameter C which delivers the lowest error metric (for example, classification error) on a validation set.

3. Pegasos SVM

3.1 Assignment description: *Implement the Pegasos algorithm for the linear kernel, adding a formula for the bias term w_0 as well) , but take care not to penalize the magnitude of w_0. Your function should output classifier weights for a linear decision boundary.*

We implement the algorithm Pegasos 1, described below.

Algorithm 1 Pegasos 1

```
1: procedure PEGASOS 1
2:     Specify inputs $(x^{(i)}, y^{(i)})$, parameter $\lambda$, $ME$ (max epochs)
3:     Infer $n$ = number of observations
4:     Set $t = 0$, $w^0 = 0$, $b_0 = 0$, $n_k = \frac{1}{k\lambda}$ for $1 \leq k \leq ME$.
5:     while $t < ME$ do
6:         for i = 1, 2, ... n do
7:             if $y^{(i)}(w_t^T \cdot x^{(i)} + b_t) < 1$ then
8:                 $w_{t+1} := (1 - n_t\lambda)w_t + n_t y^{(i)} x^{(i)}$
9:                 $b_{t+1} := b_t + n_t y^{(i)}$
10:            else
11:                $w_{t+1} = (1 - n_t\lambda)w_t$
12:                $b_{t+1} = b_t$
13:            end if
14:            $t := t + 1$
15:        end for
16:    end while Output $w_t, b_t$
17: end procedure
```

Note that w_t is our vector of weights w, and b_t is the bias w_0.

Given a new input x, we make the prediction $sgn(w^T \cdot x + b)$ to determine the predicted group (positive or negative) that the datapoint belongs to.

3.2 Assignment description: *Test various values of the regularization constant, $\lambda \in \{2, 1, 2^{-1}, \ldots 2^{-10}\}$ Observe the the margin as a function of λ. Does this match your understanding of the objective function?*

We train our model on the first training dataset. The figure below shows margin as a function of λ, on a log scale. We can see that this function is roughly linear. In particular, it is increasing in λ, which makes sense - as we increase the penalty λ, we force $\frac{1}{\|w\|^2}$ to be small, which forces the margin to increase. Also, increasing λ is equivalent to decreasing the C parameter in SVM-C, which is equivalent to penalizing slack variables less, which gives a larger margin as well.

3.3 Assignment description: *Implement a kernelized version of the Pegasos algorithm. It should take in a Gram matrix, where entry (i, j) is $K(x^{(i)}, x^{(j)}) = \langle\phi(x^{(i)}, \phi(x^{(j)})\rangle$ and should should output the support vector values, , or a function that makes a prediction for a new input. In this version, you do not need to add a bias term. The kernelized version of the Pegasos algorithm is*

$$min_w \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y^{(i)} w^T \phi(x^{(i)})\}$$

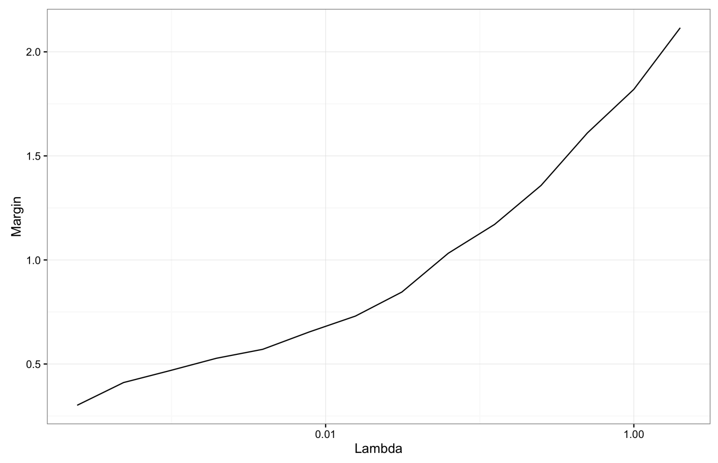


Figure 6: Margin as a function of the lambda parameter

Given this formulation, how should you make a prediction for a new input x? Does it have the same sparsity properties as the dual SVM solution?

We implement the algorithm Pegasos 2, described below:

Algorithm 2 Pegasos 2

```
1: procedure PEGASOS 2
2:     Specify inputs $(x^{(i)}, y^{(i)})$, parameter $\lambda$, $ME$ (max epochs), K (Kernel function)
3:     Infer $n$ = number of observations
4:     Set $t = 0$, $\alpha_i = 0$ for $1 \le i \le n$, $n_k = \frac{1}{k\lambda}$ for $1 \le k \le ME$.
5:     while $t < ME$ do
6:         for i = 1, 2, ... n do
7:             if $y^{(i)} \left( \sum_{j=1}^{n} \alpha_j K(x^{(j)}, x^{(i)}) \right) < 1$ then
8:                 $\alpha_i := (1 - n_t\lambda)\alpha_i + n_t y^{(i)}$
9:             else
10:                $\alpha_i := (1 - n_t\lambda)\alpha_i$
11:            end if
12:            $t := t + 1$
13:        end for
14:    end while Output $(\alpha_1, \alpha_2, \ldots, \alpha_n)$
15: end procedure
```

Given this formulation, the prediction function for a new input x is $f(x) = sgn\left(\sum_{i=1}^{n} \alpha_i K(x, x^{(i)})\right)$.

Does this have the same sparsity properties as the dual SVM solution? Well, this depends on whether the α values will be predominantly zero. But note that nowhere in our code do we specifically restrict that α_i is greater than zero, and we do indeed see many nonzero (and negative) alphas in our solution.

3.4 Assignment description: *Classify the same data using a Gaussian kernel and test various values of the* $\gamma \in \{4, 2, 1, 0.5, 0.25\}$. *Use a fixed* $\lambda = .02$. How does the decision boundary and the number of support vectors change depending on γ? How do your results compare to those obtained with the SVM in the previous section?

As γ increases in the set $\{4, 2, 1, 0.5, 0.25\}$, the number of support vectors increases to $\{53, 54, 62, 86, 121\}$. This makes sense, since a smaller γ implies more model sensitivity. The decision boundary also becomes more curved towards our sample points, as we increase γ.

Below is the decision boundary for dataset 4, with $\gamma = 0.25$:

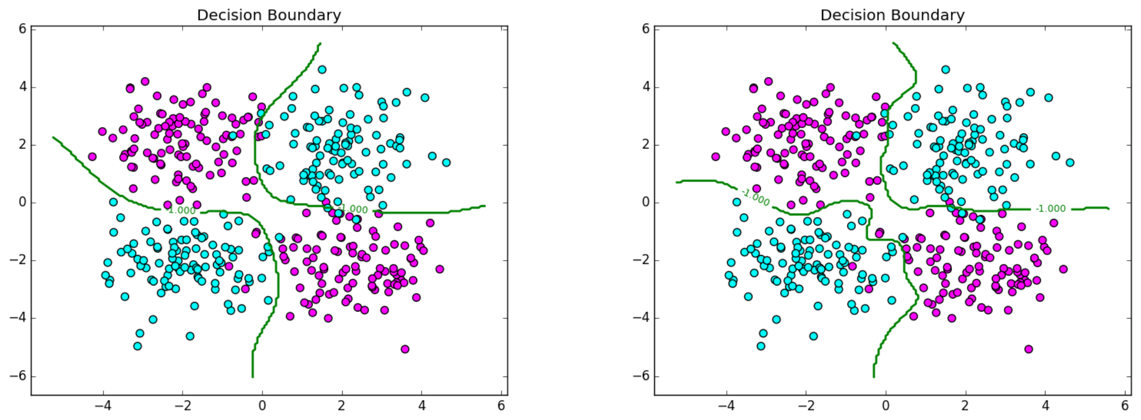


Figure 7: Decision boundary for gamma =4 (left) and gamma =0.25(right)

4.MNIST Classification

4.1 We take a look at the classification of 1 vs 7 and compare the Lasso penalty model from problem 1 to the linear SVM model from problem 2. We try out values of $\lambda \in \{0.01, 0.04, 0.16\}$ and values of $C \in \{0.4, 1, 2.5\}$. Results are in the table below.

Model Type	Training Error Rate	Validation Error Rate	Test Error Rate
Logistic regression, lasso penalty 0.01	0	0.013	0.013
Logistic regression, lasso penalty 0.04	0	0.01	0.013
Logistic regression, lasso penalty 0.16	0	0.01	0.0016
C-SVM, C=0.4	0	0.02	0.05
C-SVM, C=1	0	0.02	0.05
C-SVM, C=2.5	0	0.05	0.05

We observe that the Lasso model does better than our C -SVM, which is surprising. We also note that re-scaling our X inputs leads to very similar results.

4.2 We run the gaussian RBF kernel classification with $\gamma \in \{0.01, 0.02, 0.04\}$ and we observe that while fitting the test data perfectly, this model does very poorly on the validation and test sets - this is an indication of a potential issue with our optimization function, despite showing very sensible results for problems 2 and 3. See for example the decision boundary above. We were expecting the data to be fit more tightly by the RBF, since that is what we saw in the 2 dimensional case.

Normalization of data should matter, based on a normalization test we did on a 2-dimensional set, although this is unintuitive at first. It could be an indicator of the variance in this estimator (very unlikely), or of a potential issue with the optimization code, which makes it prone to changes in the inputs.

4.3 The accuracies between the quadratic programming aproach and the pegasos aproach are similar, although pegasus has one major drawback- in its current form, described in class, it does not have a bias term. This will mean in practice that the decision boundaries between the quadratic solution and the pegasos solution will be shifted from each other (by exactly the bias term) - and indeed this is what we saw in the two-dimensional case.

The pegasos algorithm presents an inprovement over the quadratic solution, in runtime. As we run both the quadratic algorithm and the pegasos algorithm on datasets of size (200, 400, 600, 800, 1000), we observe that the pegasus algorithm finds the weights in time approximately (approximately $size^{1.1}$), whereas pegasos solves it in polynomial time (approximately $size^{2.3}$)