6.867 Machine Learning Homework 2

1 LOGISTIC REGRESSION

Logistic Regression is a discriminative model used for classification. Given an input x, it finds the posterior of x belonging to one of the classes and then uses that probability to classify x. In the simplest case, it takes a linear combination of the input x and uses that as an input to the sigmoid function, which outputs a number between 0 and 1. One of the problems with logistic regressions is that they are very prone to overfitting to the training data. One way to prevent the overfitting is to add a regularization term, λ, which penalizes the size of the weight vector. The size of the weight vector can be penalized using the L_1 norm and the L_2 norm. Here, we explore how different values of λ and the different norms affect several aspects of the logistic regression.

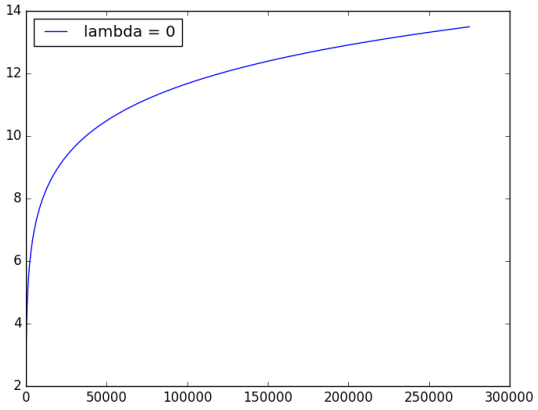
$$\sum_i \log(1 + exp(-y^{(i)}(wx^{(i)} + w_0))) + \lambda \|w\|_2^2 \tag{1}$$

$$\sum_{i=1}^{n} (\sigma(W^T x^{(i)}) - y^{(i)}) x^{(i)} + 2\lambda \tag{2}$$

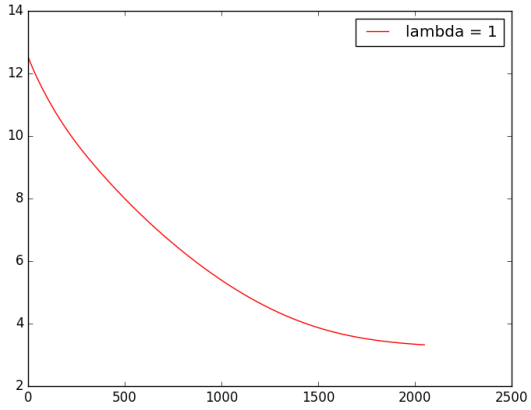
$$\frac{1}{1 + e^{-x}} \tag{3}$$

1.1 Optimizing with Gradient Descent

To investigate how L_2 regularization affected the logistic regression we tried $\lambda = 0$ and $\lambda = 1$. We decided not to penalize the bias term in the weight vector. We found that with $\lambda = 1$ the weight vector decreased in every iteration of the algorithm until it converged to its optimal value. We believe this is because for most iterations, the quickest way to decrease the objective function is to decrease the norm of the weight vector, because it is penalized by a quadratic factor. With $\lambda = 0$, the opposite happened. The norm of the weight increased in every iteration until it converged to its optimal value. Unregularized logistic regressions attempt to make the weight vector as large as possible because that makes the sigmoid function steeper, which in turn increases the log likelihood of the data. Our obersvations agree with our intuition that regularization makes the weight vector smaller.



(a) Graph of Norm of weight vector with lmbda = 0



(b) Graph of Norm of weight vector with lmbda = 0

Figure 1: Pictures of animals

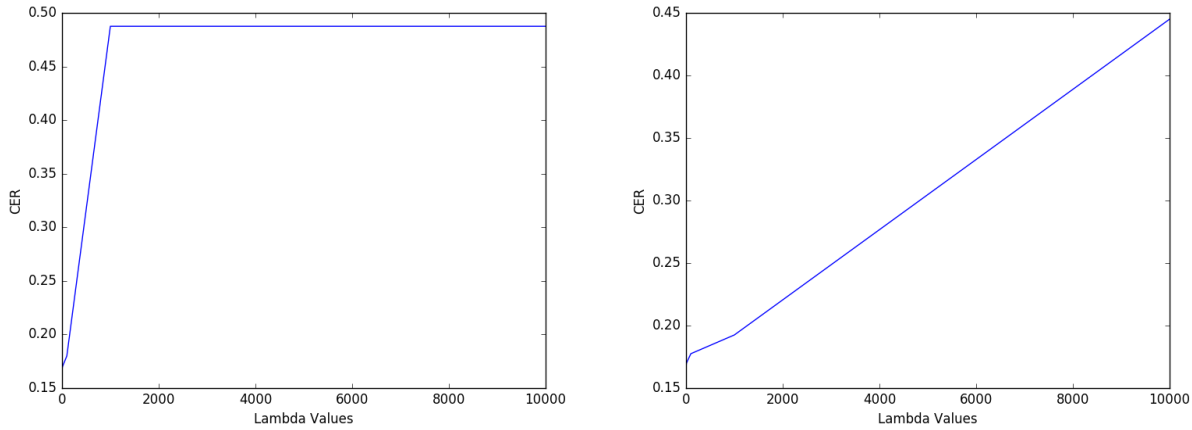
1.2 Section1??

You should be able to concisely represent trends for those configurations. For example, you could have one plot showing the effect of the error rate for different regularizers and lambdas, a good example of changing the decision boundary, and general observations, and maybe a plot about the weights. Perform your experiments and see what makes the most sense.

1.3 L1 vs L2 Norm

Two common metrics used to penalize the size of the weight vector are the L1 and L2 norm. The norms are defined as follows: (L1 is sum of absolute values). We tested the effect of each of these norms and different lambda values on the Classification Error Rate, decision boundary, and the weights of the weight vector. We used scikit-learn's implementation of Logistic Regression, which also penalizes the bias term. To allow for unpenalized bias, the logistic regression has an additional parameter, intercept$_{s}caling, whichisafactorbywhichtheinterceptismultipliedbytooffsettheeffectsoftheregularizatio$ to 10^5.

Effect on Classification Error Rate: In general, we found that the regularization seemed to have a larger effect on the CER when using the L1 norm then when using the L2 norm. The L2 norm logistic regression was able to maintain low CER's even with high lambda values. The L1 norm, on the other hand, often drove the weight vector to zero, which increased the CER. For most data sets, the L2 norm achieved a better CER than the L1 norm when set to the same lambda values. When both weight vectors were nonzero, their CER's usually only differed by a percentage point. The difference between the CER's grew dramatically whenever the L1 norm drove one of the weights in the weight vector to zero. The two models generally behaved that way except for the last data set, in which the data was linearly inseparable. There, the L1 and L2 norm achieved very similar CER scores for every lambda value we tested.

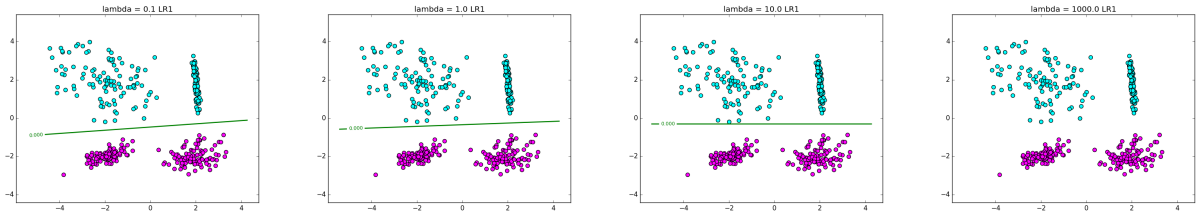


(a) Classification Error Rate of L1 norm as a function of Lambda(b) Classification Error Rate of L2 norm as a function of Lambda

Figure 2: Pictures of animals

Effect on Decision Boundary As mentioned in the previous section, the L1 norm tends to drive the weights of the weight vector to zero as lambda increases. As a result, there were several instances where the decision boundary was simply the x axis CHECK THIS. In general, we found that decreasing lambda changed the decision boundary in a direction where it could better separate the training data. This was true for all of the data sets where the data was linearly separable. For the data set in which the data was linearly inseparable, the decision boundary never changed, regardless of the lambda value.

Effect on Weights Higher lambda values decreased the weights of the weight vector in both models. Higher lambda values affected the L1 model a lot more, as it was more likely to have zero valued weights. The L2 model was more likely to have very small, but still nonzero weights.



(a) A gull (b) A gull2 (c) A tiger (d) A mouse

Figure 3: Pictures of animals

1.4 Optimal Parameters

We evaluated the performance of both norms on different data sets using the same lambda values as before. We found that when using L1 regularization, a lambda value of 0.1 achieved the best performance on both the validation and the test sets. With L2 regularization the optimal value for lambda was 1. Both of these values can be modified by a factor of 10 without significantly hurting the accuracy. With these values, the two models achieved nearly identical results on all data sets, the difference in performance was never more than half a percentage point.

1.5 Section2

2 SUPPORT VECTOR MACHINES

Support Vector Machines are supervised learning models that work by finding a dividing hyperplane between the training data while maximizing the gap between the training data and the decision boundary. This is to help the classifier generalize better and makes it more robust to noise. Assuming the data is linearly separable, finding this dividing hyperplane amounts to solving the quadratic program

$$\begin{aligned} \min \quad & \frac{1}{2}||w||^2 \\ \text{s. t.} \quad & y^i(w^T x^i + b) \geq 1, 1 \leq i \leq n \end{aligned} \tag{4}$$

where w is the vector perpendicular to the dividing hyperplane and $\frac{1}{||w||}$ is the size of the margin.

If the data is almost but not completely linearly separable, we can still model the data with an SVM by introducing slack variables when solving for a classifier. We allow the training points to be misclassified by some amount e and the goal is to maximize the margin while minimizing the slack. This formulation is called C-SVM and the separating hyperplane can be found by solving the quadratic program below.

$$\begin{aligned} \min \quad & \frac{1}{2}||w||^2 + C\sum_i e_i \\ \text{s. t.} \quad & y^i(w^T x^i + b) \geq 1 - e_i, 1 \leq i \leq n \\ & e_i \geq 0, 1 \leq i \leq n \end{aligned} \tag{5}$$

2.1 Dual Formulation of C-SVM

This problem as stated above is difficult to implement with kernels functions. Fortunately, we convert the problem from the primal formulation stated above and instead implement its dual formuation.

$$\begin{aligned} \min \quad & \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j[\phi(x_i)^T\phi(x_j)] - \sum_{t=1}^{n}\alpha_t \\ \text{s. t.} \quad & 0 \leq a_t \leq C, \sum_{t=1}^{n}\alpha_t y_t = 0 \end{aligned} \tag{6}$$

To solve this linear program and get the αs, we used the solver in a python library called cvxopt. The solver finds solutions to quadratic programs of the form:

$$\begin{array}{ll} \min & \frac{1}{2}x^T Px + q^T x \\ \text{s. t.} & Gx \leq h \\ & Ax = b \end{array} \tag{7}$$

In our case, our matrices are

$$P = Diag(\overrightarrow{y})XX^T Diag(\overrightarrow{y})$$
$$q = -1 * \overrightarrow{1}$$
$$G = [I|-I]$$
$$h = [C * \overrightarrow{1} \,|\, \overrightarrow{0}]$$
$$A = \overrightarrow{y}$$
$$b = 0$$

$Diag(\overrightarrow{y})$ is a diagonal matrix with $y^{(i)}$s on the diagonal.

This solver returns $\overrightarrow{x}$ which is our $\overrightarrow{\alpha}$ vector.

Testing this implementation, we created a dataset with four points. The positive examples were $(2,2),(2,3)$ and the negative examples where $(0,-1),(-3,-2)$. The solver returned

$$\overrightarrow{\alpha} = [1.54e-01, 7.02e-08, 1.54e-01, 9.15e-09] \tag{8}$$

Notice how α_1 and α_3 are much larger than the rest. This means that these two are the support vectors and they correspond to points $(2,2)$ and $(0,-1)$. The graph of the decision boundary for this small example is shown in figure ? and the support vectors are the points predicted.

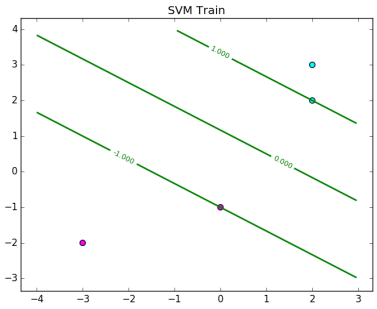
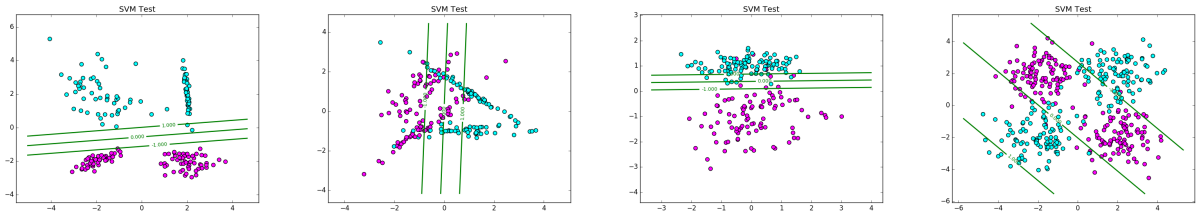


Figure 4: SVM with linear kernel on four data points.

2.2 2D Dataset Results

We used our implementation of SVM to classify 2D points and generated plots of the decision boundary and the margins. Figure ? shows SVM run on four different training sets with linear kernels and a slack penalty, C, of 1. It is clear the datasets 1 and 3 are meant to be separated by a linear function while datasets 2 and 4 are not. Thus only the separating decision boundary for 1 and 3 are reasonable. The table below shows the training, validation, and test classification error rates. Unsurprisingly, datasets 2 and 4 have terrible classifications rates.

	Training Error	Validation Error	Test Error
Dataset 1	0	0	0
Dataset 2	0.1775	0.18	0.195
Dataset 3	0.02	0.03	0.045
Dataset 4	0.3	0.305	0.3



(a) Dataset 1, C=1 (b) Dataset 2, C=1 (c) Dataset 3, C=1 (d) Dataset 4, C=1

Figure 5: Linear SVM perfomance of different datasets. Datasets 1 and 3 are relatively linearly separable so this SVM performs well on them. Dataset 2 and 4 are definitely not linearly separable so this SVM is not the right model for the data.

2.3 2D Dataset Results with Kernel Functions

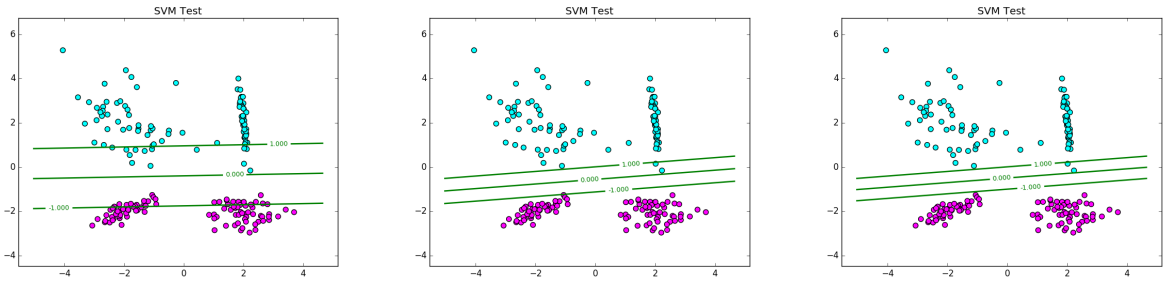
¡¡¡¡¡¡¡ HEAD Support Vector Machines can still be used to classify not linearly separable data. To do this, we need to either map the data points to higher dimensions or use a nonlinear kernel. To obtain a better classification for dataset 4, we used a Gaussian radial basis function (RBF) kernel with $\gamma = 1$. The result is shown in figure b of figure ? and it achieved a test error rate 4.75%.

2.4 Effect of C on Margin

The amount of penalty C given to the slack variables can have a large impact on the size of the margin. The small C is, the more the algorithm is willing to give slack to the data points. Thus the size of the margin will increase. As C goes to infinity, the boundary becomes more and more like Hard-SVM.

This relationship between C and the margin size is true for both linear and nonlinear kernels. In figure ?, the graphs with linear kernels are in order of increasing C. We can clearly see the margin is decreasing in size. In figure ?, the graphs with nonlinear kernels also have decreasing margin size with increasing C. As the margin decreases, the number of support vectors, that is the number of vectors that are within or on the the margin or the number of points with nonzero αs, also decreases. This is clear because less points fall on the wrong side of their margin boundary. Thus as C increases, the number of support vectors decrease.

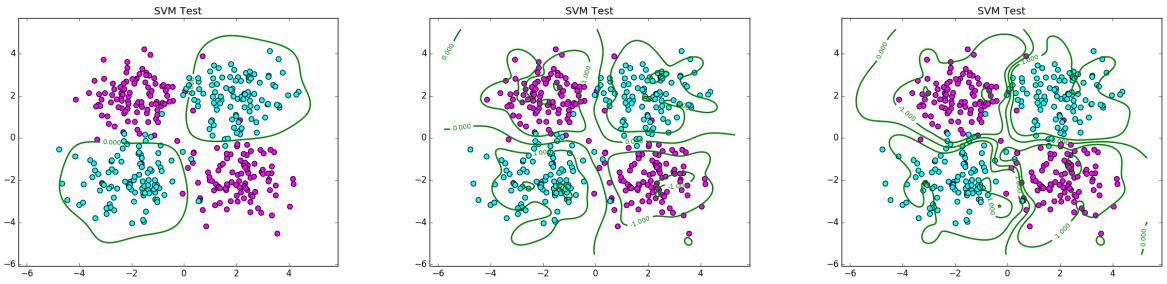
When optimizing and choosing a C, maximizing the geometric margin should not be the goal. The goal should instead be minimizing the error rate in the validation set. A large geometric margin is not related to a low error rate. In fact, a large geometric margin means lots of support vectors which leads to slow runtimes.



(a) Dataset 1, C=0.01 (b) Dataset 1, C=1 (c) Dataset 1, C=100

Figure 6: Decision boundary for SVM with linear kernel. The margin clearly gets smaller for large C

======= small lambdas penalize the w less, which let w get bigger ¿¿¿¿¿¿¿ f5a6e012cde3361dfe4b260a617d8c21b0f77a3b as Lambda — ¿ inf w ——-¿ 0 margin = 1/w ——¿ inf
Lambda —¿ 0 Empirical evidence: as lambda increases, the margin decreases



(a) Dataset 4, C=0.01 (b) Dataset 4, C=1 (c) Dataset 4, C=100

Figure 7: Decision boundary for SVM with Gaussian RBF kernel. In figure a, the margins are so big they almost do not show up in the graph. By figure c, they have shrunk considerably closer to the decision boundary.

3 SUPPORT VECTOR MACHINE WITH PEGASOS

Even though SVMs have some nice properties, it can be very inefficient to solve large SVMs. The Pegasos Algorithm can be used to solve a formulation of Soft-SVM that is equivalent to C-SVM with C = 1/(lmbda*n). One of the biggest advantages the Pegasos algorithm offers is very fast training times.

3.1 Linear SVM

We implemented the Pegasos algorithm to solve the objective function shown above and tested several values of lambda between $2^1 and 2^-10 and found that the margins hrank as lambda was increased$.

3.2 Kernelized Pegasos

The Pegasos algorithm can be easily modified to support Kernels. We implemented this modification and evaluated the performance of SVMs trained using Pegasos.

4 HANDWRITTEN DIGIT RECOGNITION WITH MNIST

Using the three classification algorithms we developed in this paper, we attempted to classify hand-written digits from the MNIST dataset and compared their performances. Since all the algorithms performed only two-way classification, our goal was to distinguish a certain set of digits for another set of digits. Images of digits from one set were labeled 1 and while digits from the other set were labeled -1.

The training sets used consisted of 200 images of each digit to be classified. Validation and test sets contained 150 for each digit. Each 28 X 28 image was treated as an vector of length 784. We tried the following classfication tasks, 1 vs 7, 3 vs 5, 4 vs 9, (0, 2, 4, 6, 8) vs (1, 3, 5, 7, 9) and also explored the effect of normalizing the image vectors on an algorithm ability to classify the digits. Normalizing a pixel means taking its 0-255 value and mapping it to the range $[-1, 1]$.

4.1 Logistic Regression vs Linear SVM

4.2 Effect of C on Margin