6.867 HW2: Binary Classification Methods

October 26, 2015

When making predictions and models, sometimes we would rather have discrete labels instead of continuous values. For example, if we were looking exam results, we might only care if students pass or fail. Alternatively, we may want to study electricity patterns, and at any given time lights can either be on or off. This idea of discrete labels is known as classification and in the case that there are only two options, binary classification. For the following work, we will consider the class labels $\{-1, +1\}$.

1 Logistic Regression

The first method we will use to implement binary classification will be logistic regression. The use of logistic functions, specifically the sigmoid, allow us to map continuous values of input values to a corresponding probability between 0 and 1, which then can be used to select a discrete label. Considering the maximum likelihood estimation, logistic regression can be written as

$$\text{MLE}(w) = \sum_{i=1}^{n} \log(1 + e^{-y^{(i)} w^T \Phi(x^{(i)})}) \tag{1}$$

Where w are the weights defining our model, $x(i), y(i)$ given data points, and Φ the basis functions from which we construct our model. Letting the basis function be a simple linear formulation, $\Phi(x) = \begin{bmatrix} 1 & x_1 & x_2 & ... \end{bmatrix}^T$, the above equation can be simplified as

$$\text{NNL}(w) = \sum_{i=1}^{n} \log(1 + e^{-y^{(i)}(w_0 + w^T x^{(i)})}) \tag{2}$$

However to prevent overfitting, we also want to introduce a regularization term into the logistic regression error function. Specifically we will consider a quadratic regularizer, which results in the finalized error function,

$$E_{LR}(w) = \text{NNL}(w) + \lambda w^T w \tag{3}$$

Unlike some of the regression formulations we looked at last time, such as ridge or linear, there is no closed from solution to minimizing the error. Thus, we must employ optimization methods to select the correct weights. As far as optimization methods,we used a simple gradient descent.

Once weights are calculated from minimizing the error on the training data our classifier is simply defined as,

$$h = \text{sign}(w^T x + w_0) \tag{4}$$

Which simply creates a hyperplane in which for any input, x. This classification returns -1 if $w^T x + w_0 < 0$ and +1 if $w^T x \geq 0$. The boundary decision is marked by $.5 = \sigma(w^T x)$ and is shown below for the case where there is no regularizer.

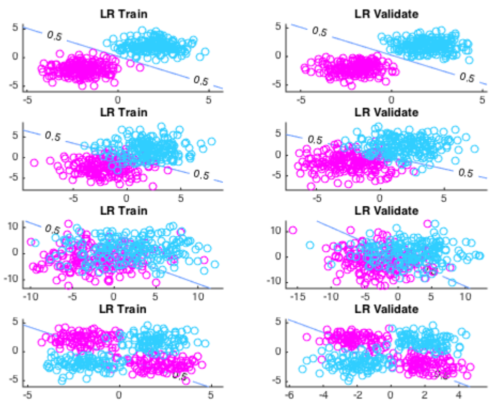

Figure 1: Labelled test and validation data for stdev1,stdev2, stdev4 and nonsep data sets. Also shown is the decision boundary

Also considered was the error from mislabelled points, which was calculated for both the test and validation sets. Of course, the more easily separable the data, the lower the error. These results are shown in Table 1

Table 1: Training and Validation Errors for different regularizers, λ

dataset	$\lambda = 0$	$\lambda = 10$	$\lambda = 1000$
stdev1	[0,0]	[0,0]	[0,0]
stdev2	[36, 29]	[36,30]	[41, 26]
stdev4	[105, 99]	[105, 99]	[106, 98]
nonsep	[239, 247]	[243, 247]	[233, 228]

As the magnitude of w increases the steepness of the hyperplane increases. If w goes to ∞ the separator becomes a step function. The regularizer helps to limit this by penalizing the magnitude of the weights. It also helps prevent overfitting. So as λ is increased the training error might increase since the cost is less influenced on the data error, NLL, and more influenced by the weights.

We tested this logistic regression for $\lambda = \{0, 10, 1000\}$ and for the clearly separable and non separable case, there was no effect. For the stdev2, and stdev4 cases the training error increases but the validation error decreases. These results are shown in Table 1

2 Support Vector Machine

Another implementation method of binary classification is the Support Vector Machine. In this classification method, we seek to divide the data into the different classifications with a clear a gap as possible. This gap or margin is defined as support vectors and takes advantage of the hinge loss. The simplest form of SVM, the primal hard form, is expressed as

$$\min_{w} ||w||^2 \tag{5}$$

$$s.t.\ \forall i, y^{(i)} w^T x^{(i)} \geq 1 \tag{6}$$

The hard form of SVM does not allow any leeway on the constraints, making it impossible to create a model if the data is not linearly separable. To increase the capability of the SVM, the soft SVM, includes a slack variable, ξ, For the straight forward primal form of soft SVM applied to the linear case, $\Phi(x) = \begin{bmatrix}1 & x_1 & x_2 & ...\end{bmatrix}^T$, the overall objective once combined with corresponding constraint is thus

$$\min_{w,w_0} (\frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \xi_i) \tag{7}$$

$$s.t.\ y^{(i)}(w^T x^{(i)} + w_0) \geq 1 - \xi_i \tag{8}$$

$$\xi_i \geq 0 \tag{9}$$

This formulation is not ideal, instead we translate this to the dual form which using an auxiliary variable, α to rewrite the SVM as a maximization and clearly shows the optimization as a maximum margin.

$$\max_{\alpha \in \mathbb{R}^n} (\sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)})^T x^{(j)}) \tag{10}$$

$$s.t.\ 0 \leq \alpha_i \leq C \tag{11}$$

$$\sum_{i=1}^{n} \alpha_i y^{(i)} = 0 \tag{12}$$

Instead of optimizing over the weights explicitly, we optimize over α from which the weights can be directly calculated as

$$w = \sum_{i=1}^{n} \alpha_i y^{(i)} x^{(i)} \tag{13}$$

$$w_0 = \frac{1}{M}(\sum_{j \in M} y^{(j)} - \sum_{i \in S} \alpha_i y^{(i)} (x^{(j)})^T x^{(i)}) \tag{14}$$

Where S is the set of support vectors, $S = \{i : 0 < \alpha_i \leq C\}$ and $M = \{i : 0 < \alpha_i < C\}$. However since there is some numerical residual resulting from the optimizations of α that won't exactly equal 0 or C we will consider the small tolerance, $\epsilon = 10^-5$ so that $S = \{i : \epsilon < \alpha_i \leq C - \epsilon\}$ and the same for M.

The MATLAB function *quadprog* minimizes the function $1/2x^T Hx + f^T x$ with the constraints $Ax <= b$ and $A_{eq}x = b_{eq}$. For this portion, we set

$$H = diag(y)xx^T diag(y)$$
$$f = -\mathbf{1}_{n \text{ x } 1}$$
$$A = \begin{bmatrix}-I_{n \text{ x } n} \\ I_{n \text{ x } n}\end{bmatrix}$$
$$b = \begin{bmatrix}\mathbf{0}_{n \text{ x } 1} \\ \mathbf{C}_{n \text{ x } 1}\end{bmatrix}$$
$$A_{eq} = y^T$$
$$b_{eq} = 0$$

Where the bold terms indicate matrixes of constant value of size denoted by the subscript and n is the length of the input data.

This yielded $\alpha = [0.5306, 0.0000, 0.3673, 0.1633]$ for the input $x = [1, 2; 2, 2; 0, 0; -2, 3]$ and $y =$

Table 2: Training and Validation Errors for $C = 1$

dataset	Test Error	Validation Error
stdev1	0	0
stdev2	38	32
stdev4	102	94
nonsep	118	121

$[1, 1, -1, -1]$. This makes intuitive sense because the non support vector, (2,2), has an α value of 0.

This SVM was applied to the data from problem 1, stdev1,stdev2, stdev4 and nonsep, test and validation sets. Initially we looked at the case where $C = 1$. The plotted results are shown in Fig. 2 and the corresponding errors in Table 2.

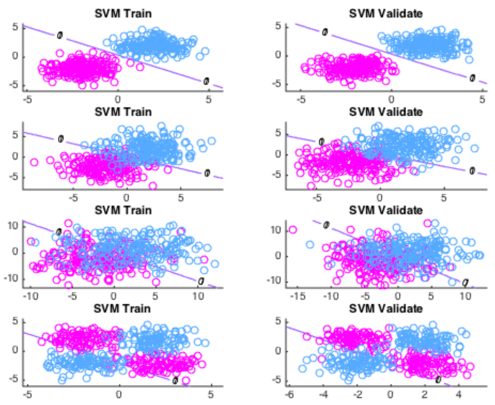

Figure 2: Labelled test and validation data for stdev1,stdev2, stdev4 and nonsep data sets when $C = 1$. Also shown is the decision boundary

Clearly the more overlapped the data is the more errors the classifier makes. However comparing with the results from Table 1, the SVM implemented here outperforms our logistic regression algorithm for stdev4 (102, 94 vs 105, 99) and nonsep (118, 121 vs 243,247).

Kernels

The multiplication $x^T x$ appears in multiple locations for out SVM calculations, both in the optimization of the α_i and in the calculation of w_0. The idea of kernel trick is that we can rewrite anything the terms there as a kernel function, $k(x, z)$ replacing any terms that can be expressed as inner products. Using this formulation the SVM can be written as

$$\max_{\alpha \in \mathbb{R}^n} (\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} k(x^{(i)}, x^{(j)}) \quad (15)$$

$$s.t.\ 0 \leq \alpha_i \leq C \quad (16)$$

$$\sum_{i=1}^{n} \alpha_i y^{(i)} = 0 \quad (17)$$

$$w = \sum_{i=1}^{n} \alpha_i y^{(i)} x^{(i)} \quad (18)$$

$$w_0 = \frac{1}{M} (\sum_{j \in M} y^{(j)} - \sum_{i \in S} \alpha_i y^{(i)} k(x^{(j)}, x^{(i)}) \quad (19)$$

With the decision boundary,

$$\sum_{i:\alpha_i > 0} \alpha_i y^{(i)} k(x^{(i)}, x) + w_0 = 0. \quad (20)$$

More importantly kernels are adaptable to a range of basis functions. It can be used for anything that can be written as an inner product. In particular we chose to look at the Gaussian Kernel

$$k(x, z) = \exp(-\frac{1}{2\sigma^2} ||x - z||_2^2) \quad (21)$$

Where σ is the bandwidth.

Trying the Guassian Kernel on the data sets already mention, with $C = .01, \sigma = .1$ the following results were obtained

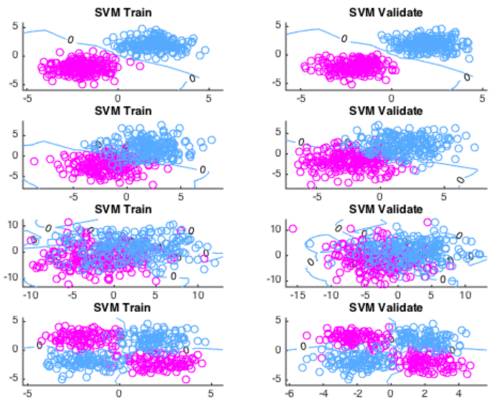

Figure 3: Labelled test and validation data for stdev1,stdev2, stdev4 and nonsep data sets when $C = .01, \sigma = .1$ with Gaussian Kernel. Also shown is the decision boundary

Table 3: Geometric Margin for varying C

dataset	$C = .01$	$C = .1$	$C = 1$	$C = 10$	$C = 100$
stdev1	.089	.013	$8.08 \cdot 10^{-}3$	$2.85 \cdot 10^{-}3$	$3.49 \cdot 10^{-}4$
stdev2	.088	.009	$2.40 \cdot 10^{-}3$	$3.72 \cdot 10^{-}4$	$4.01 \cdot 10^{-}5$
stdev4	.096	.010	$1.71 \cdot 10^{-}3$	$3.25 \cdot 10^{-}4$	$3.57 \cdot 10^{-}5$
nonsep	3.03	.300	$2.78 \cdot 10^{-}2$	$3.11 \cdot 10^{-}3$	$1.59 \cdot 10^{-}4$

Table 4: Number of Support Vectors for varying C

dataset	$C = .01$	$C = .1$	$C = 1$	$C = 10$	$C = 100$
stdev1	400	400	396	388	364
stdev2	400	400	386	346	302
stdev4	400	392	369	294	263
nonsep	400	398	375	332	307

Increasing σ leads to Fig. 4. Where one can see that the decision boundary has become more linear

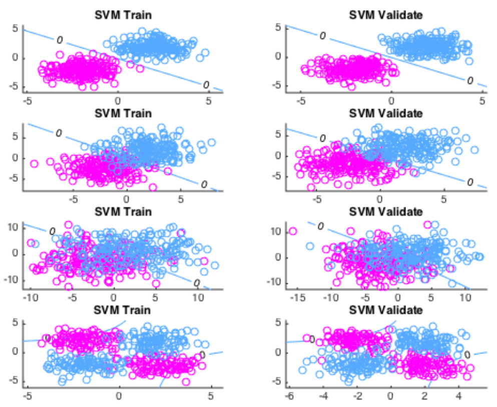

Figure 4: Labelled test and validation data for stdev1,stdev2, stdev4 and nonsep data sets when $C = .01, \sigma = 10$ with Gaussian Kernel. Also shown is the decision boundary

As shown in Fig. 3 The gaussian kernel performs a lot better on the non separable data.

Using a constant value of $\sigma = .1$ we investigated the effect of changing C on the geometric margin, $\frac{1}{||w||}$ and the number of support vectors, S. In particular we considered $C = \{.01, .1, 1, 10, 100\}$ and the results are shown in Tables 3 and 4

When C increases,the geometric margin decreases as does the number of support vectors. This is because the slack variables are being penalized more and more resulting in smaller margins. However when C becomes too large, the problem becomes hard SVM and it may not be possible to separate the data.

3 Titanic Data