6140: Programming Assignment 2

In this assignment we will look at two popular methods for classification: logistic regression and SVMs.

Please view this assignment as a chance to code, analyze the methods and subsequently document the observations. This assignment is to be completed individually (i.e., no teams). You have to create all deliverables yourself from scratch. In particular, you are not allowed to look at or copy someone else's code/text and modify it. If you use publicly available code/text, you need to cite the source in your report! Cheating and other acts of academic dishonesty will be referred to OSCCR (office of student conduct and conflict resolution) and the College of Computer Science.

All programming assignments must be completed in **Python**. You will find a zip file with some useful code and data in the Resources section of the Piazza course page. You can use the code in zip file or external libraries to read files and plot results. However, you shouldn't use external libraries to implement your main algorithm. You need to make sure your code is **runnable** before you submit it. We will grade mainly based on your document submission, so make sure you have a clear explanation in the .pdf file including plots, running results, and explanations.

To submit your solutions,

1. Use your CCIS Github account and access the repository named ml-<your_name>
2. Create a new folder for this assignment: (e.g. PA2)
3. Push your solutions for this assignment to that folder.
4. Your submission should include one .py file and one .pdf file. Please name these files with your name (e.g. Kechen_Qin_PA2.pdf).

Please submit your solution by the due date shown online (usually two weeks from release). Late assignments will be penalized by 10% for each day late. For example, if you turned in a perfect programming assignment two days late, you would receive an 80% instead of 100%.

1 Logistic Regression

We have given you three data sets (in the `data` folder); the name of each data set indicates whether it is to be used as a "training" set or as a "validation" set. Note that for consistency with SVMs, we will label all the data with $y^{(i)} \in \{-1, 1\}$. See the discussion in Appendix about how to reformulate the logistic regression model for this case; equations $\sum_{n=1}^{N} E_{LR}(y_n t_n) + \lambda \|w\|^2$ and $E_{LR}(yt) = \ln(1 + \exp(-yt))$ define the relevant log likelihood function.

1. Implement logistic regression using one of the minimizers you experimented with in Homework 1 (your own gradient descent or one of the other optimization methods you tried). Include a quadratic (L2) regularizer on the weights. Write a prediction function that can be used to "predict" an input point, that is, compute the posterior probability $P(y \mid x, w)$. Use this to report a count of the number of mistakes on the training data.

2. Test your implementation on the data sets provided. Report the behavior of your algorithm when $\lambda = 0$ (no regularization) on the training and validation data sets. Report the behavior of your algorithm when λ increases on the training and validation data sets. Report the general trends in what you find; comment specially on performance on non-separable data.

3. Extend your logistic regression implementation to handle second order (polynomial) basis functions, in particular, you should have basis functions for 1, for x_i for $i = 1, \ldots, D$ and for $x_i x_j$ for $i = 1, \ldots, D$, $j = i, \ldots, D$, where D is the dimension of the input data. Report the performance of the generalized regression on the training and data set. Illustrate the effect of regularization and explain what you see.

2 Support vector machine implementation

1. Implement a linear SVM with slack variables. You should implement both the primal and the dual form; compare the results to verify that you have implemented them correctly. See the separate file `optimizers.txt` on how to install and call the optimizers. Write a prediction function that can be used to "predict" an input point, that is, compute the functional margin of the point.

2. Run your SVM implementation on the training and validation data sets we gave you (all three). Report your results and explain what you find.

3 Kernel SVM

1. Extend your dual SVM implementation to use kernels. Hint: this is a relatively small change. Implement the prediction function. Test a second order polynomial kernel and a Gaussian

kernel on the training and validation data (all three). You should **not** have to modify your basic dual solution when you change kernels. Show (and explain) your results both graphically and by reporting mistakes for several values of C and the Gaussian kernel variance. Compare to your logistic regression results.

4 Appendix

(Source: Bishop 7.1.2)

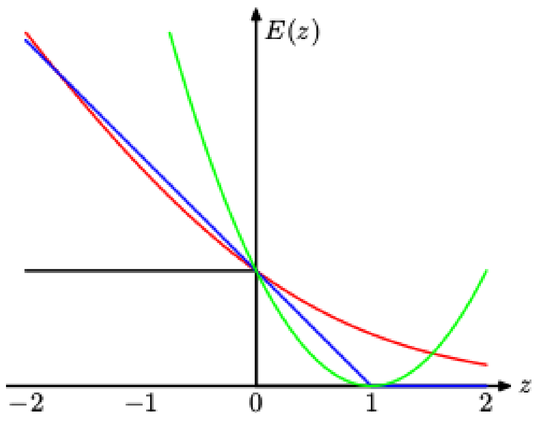

Figure 1: Plot of the 'hinge' error function used in support vector machines, shown in blue, along with the error function for logistic regression, rescaled by a factor of $1/\ln(2)$ so that it passes through the point (0, 1), shown in red. Also shown are the misclassification error in black and the squared error in green.

As with the separable case, we can re-cast the SVM for nonseparable distributions in terms of the minimization of a regularized error function. This will also allow us to highlight similarities, and differences, compared to the logistic regression model.

We have seen that for data points that are on the correct side of the margin boundary, and which therefore satisfy $y_n t_n \geqslant 1$, we have $\xi_n = 0$, and for the remaining points we have $\xi_n = 1 - y_n t_n$. Thus the objective function $C \sum_{n=1}^{N} \xi_n + \frac{1}{2}||w||^2$ can be written (up to an overall multiplicative constant) in the form

$$\sum_{n=1}^{N} E_{SV}(y_n t_n) + \lambda ||w||^2$$

where $\lambda = (2C)^{-1}$, and$E_{SV}(\cdot)$ is the hinge error function defined by

$$E_{SV}(y_n t_n) = [1 - y_n t_n]_+$$

where $[\cdot]_+$ denotes the positive part. The hinge error function, so-called because of its shape, is plotted in Figure 1. It can be viewed as an approximation to the misclassification error, i.e., the error function that ideally we would like to minimize, which is also shown in Figure 1.

When we considered the logistic regression model, we found it convenient to work with target variable $t \in 0, 1$. For comparison with the support vector machine, we first reformulate maximum likelihood logistic regression using the target variable $t \in \{-1, 1\}$. To do this, we note that $p(t = 1|y) = \sigma(y)$ where $y(x) = w^T \phi(x) + b$, and $\sigma(y)$ is the logistic sigmoid function $\sigma(a) = \frac{1}{1+exp(-a)}$. It follows that $p(t = -1|y) = 1 - \sigma(y) = \sigma(-y)$, where we have used the properties of the logistic sigmoid function, and so we can write

$$p(t|y) = \sigma(yt)$$

From this we can construct an error function by taking the negative logarithm of the likelihood function that, with a quadratic regularizer, takes the form

$$\sum_{n=1}^{N} E_{LR}(y_n t_n) + \lambda ||w||^2$$

where

$$E_{LR}(yt) = ln(1 + exp(-yt))$$

For comparison with other error functions, we can divide by ln(2) so that the error function passes through the point (0, 1). This rescaled error function is also plotted in Figure 1 and we see that it has a similar form to the support vector error function. The key difference is that the flat region in $E_{SV}(yt)$ leads to sparse solutions.

Both the logistic error and the hinge loss can be viewed as continuous approximations to the misclassification error. Another continuous error function that has sometimes been used to solve classification problems is the squared error, which is again plotted in Figure 1. It has the property, however, of placing increasing emphasis on data points that are correctly classified but that are a long way from the decision boundary on the correct side. Such points will be strongly weighted at the expense of misclassified points, and so if the objective is to minimize the misclassification rate, then a monotonically decreasing error function would be a better choice.