Homework 3 - Machine Learning

Catherine Baker

January 2024

Contents

1 Logistic Regression with Gradient Descent

We are given that the logistic regression function is:

$$f(x) = \frac{1}{1 + e^{-(\beta_0 + x^T\beta)}} \tag{1}$$

and the loss function is:

$$L(x, y) = log(1 + e^{-y(\beta_0 + x^T\beta)}) \tag{2}$$

1.1 Gradient Descent

1.1.1 Partial Derivatives

Let us compute the derivative of the loss function with respect to β. Because we have a nested function (an outer log function with an inner linear function) we must use the chain rule.

$$L'(x, y) = \frac{\partial L}{\partial \beta} log(1 + e^z) = \frac{e^z}{1 + e^z}$$

where

$$z = -y(\beta_0 + x^T\beta)$$

Now we take the derivative of z with respect to both β and β_0:

$$\frac{\partial z}{\partial \beta}(-y(\beta_0 + x^T\beta)) = -yx^T$$

$$\frac{\partial z}{\partial \beta_0}(-y(\beta_0 + x^T\beta)) = -y$$

By the Chain Rule, we should now multiply this inner derivative by our outer derivative and plug in for our z value:

$$\frac{\partial L}{\partial \beta} = -\frac{yx^T e^{-y(\beta_0 + x^T\beta)}}{1 + e^{-y(\beta_0 + x^T\beta)}}$$

$$\frac{\partial L}{\partial \beta_0} = -\frac{y e^{-y(\beta_0 + x^T\beta)}}{1 + e^{-y(\beta_0 + x^T\beta)}}$$

1.1.2 Code

To avoid overflow errors, we had to use the sigmoid function in our code to rewrite the gradient calculation:

$$\frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}$$

rewritting our above derivatives into:

$$\frac{\partial L}{\partial \beta} = -\frac{yx^T}{1 + e^{y(\beta_0 + x^T\beta)}}$$
$$\frac{\partial L}{\partial \beta_0} = -\frac{y}{1 + e^{y(\beta_0 + x^T\beta)}}$$

Please see CatherineBakerQ1A.py to view the code that created Figure 1.

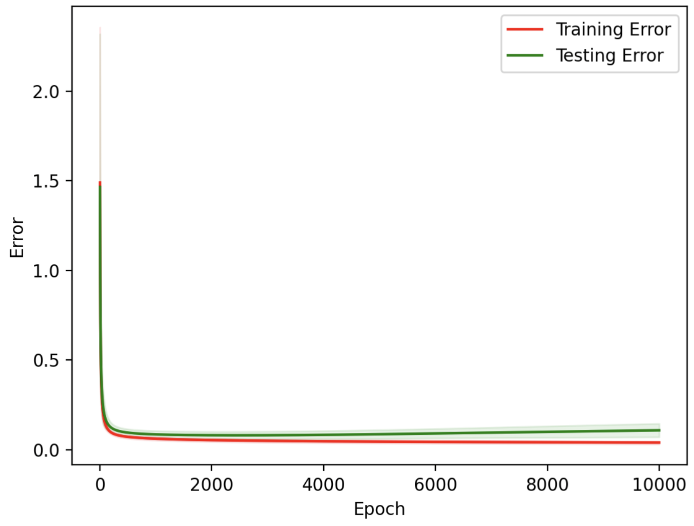

Figure 1: The plot produced by my CatherineBakerQ1A.py code. The red line represents the training data error and the green line is the test error. The standard deviation is show as the lighter fill lines.

It is worth noting that our Figure 1 shows a higher standard deviation and overall error on our testing data than the training data. Since our model weights were built by the training data this is to be expected, but could indicate overfitting. However, because the overall error is low for both testing and training, we remain confident that the model did not significantly overfit.

1.2 Built-in Logistic Regression

In Figure 2, the increase in error from train to test may lead us to believe the model overfit to the data. There is also a large amount of variability in our data which leads us to question the exactness of our implementation.

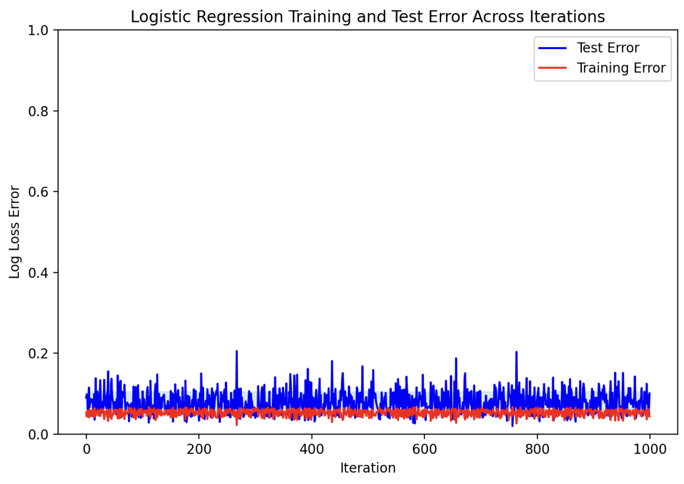

Figure 2: The plot produced by my CatherineBakerQ1B.py code. The blue line represents testing error while the red line represents training error.

2 Perceptron

See CatherineBakerQ2.py for the code that produced Figures 3 and 4.

Both of our plots produced similarly appropriate deision boundaries. The boundary on Figure 4 is seemingly better fit to the data points, but this may be due to the width of the margin between the red and blue points being smaller. Our code reported the following square of the radius of the ball R and geometric margin y for datasets 1 and 2:

Data 1:
R^2: 40224.711050749
y^2: 2.691223336254419
Ratio of R^2 to y^2: 14946.626877400968
Training completed in 2 iterations.
Testing error: 0.0

Data 2:
R^2: 38566.11232948361
y^2: 0.0027386701552385524
Ratio of R^2 to y^2: 14082058.131649775
Training completed in 4 iterations.
Testing error: 0.0

The values for R^2 are both high indicating that our data in both sets (and slightly more for dataset 1) is widely spread out. Additionally the datasets

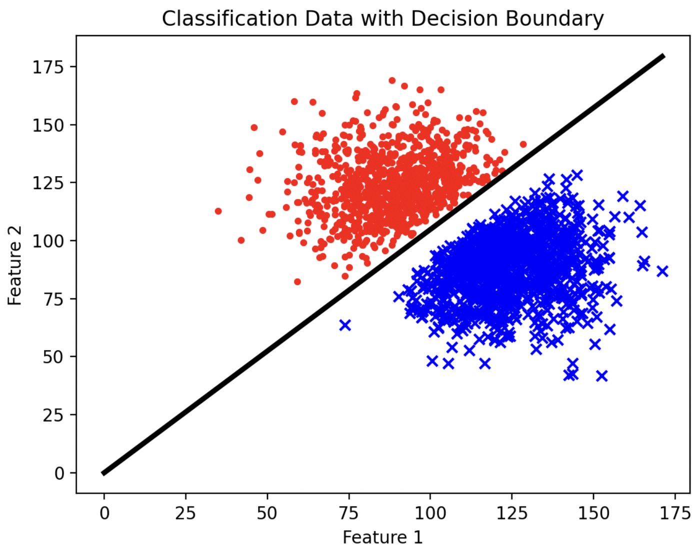

Figure 3: The plot produced by my CatherineBakerQ2.py code for data 1's X and y. The black line represents the decision boundary. The red dots are points where $y = +1$. The blue X's are points where $y = -1$.

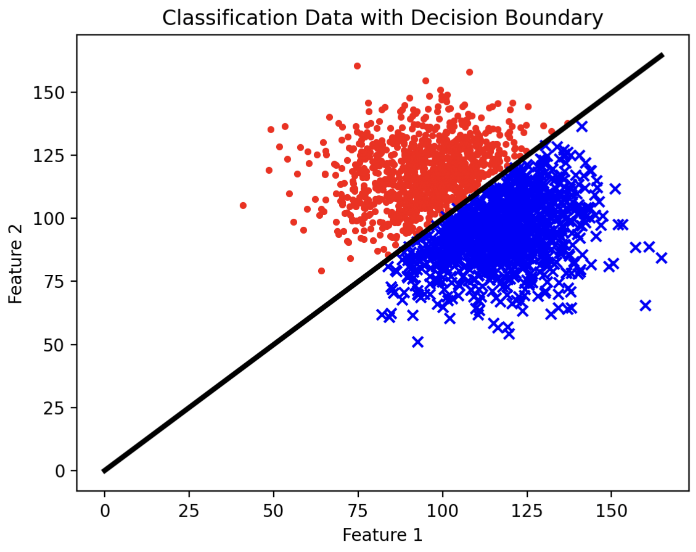

Figure 4: The plot produced by my CatherineBakerQ2.py code for data 2's X and y. The black line represents the decision boundary. The red dots are points where $y = +1$. The blue X's are points where $y = -1$.

reported similarly small, but different y^2 values. Dataset 1 (larger y^2 value) seemed to have its closest points farther from the margin than dataset 2. This is shown on our graphs and may be due to the wider spread of points (R^2 value) in dataset 1.

Lastly, the ratio for dataset 1 is much higher (1000 times higher) than that for dataset 2. This indicates that while there was a smaller spread of points, it was more difficult to classify the margin with a decision boundary than in dataset 1. This makes sense to us as our dataset 1 had a much wider space between the groups of circles and x's than in dataset 2 making it more linearly separable.

3 Support Vector Machine

3.1 Properties of Kernel

3.1.1 Proving Kernel Symmetry

To say the kernel is symmetric is to say $K(x_1, x_2) = K(x_2, x_1)$. We can prove this is true by recognizing that the operations the Kernel performs do not rely on the order of the inputs (distance measures, dot products, e.g. $K(x_1, x_2) = x_1^T x_2 = x_2^T x_1$, etc.). Because the operations do not rely on the variable input order, the order does not matter and therefore makes the Kernel symmetric.

3.1.2 Proving a Semi-Positive Definite A

We are given that:

$$A(i,j) = K(x_i, x_j) = (1 + x_i x_j)^2 \tag{3}$$

and we want to prove that for a given n dimensional vector **f**, it holds that:

$$\mathbf{f'Af} \geq 0 \tag{4}$$

To show this let's expand the kernel function:

$$A(i,j) = K(x_i, x_j) = (1 + x_i x_j)^2$$
$$\Rightarrow = 1 + (x_i x_j)^2 + 2x_i x_j$$

and rewrite Equation 4 to multiply values over a summation rather than matrix-vector multiplication:

$$\mathbf{f'Af} = \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} f_i A(i,j) f_j$$
$$\Rightarrow = \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} f_i (1 + (x_i x_j)^2 + 2x_i x_j) f_j$$
$$\Rightarrow = \Sigma_{i=1}^{n} \Sigma_{j=1}^{n} (f_i + (x_i x_j)^2 f_i + 2x_i x_j f_i) f_j$$

$$\Rightarrow = \Sigma_{i=1}^{n}\Sigma_{j=1}^{n}(f_i f_j + f_i(x_i x_j)^2 f_j + 2 f_i x_i x_j f_j)$$

$$\Rightarrow = \Sigma_{i=1}^{n}\Sigma_{j=1}^{n} f_i f_j + \Sigma_{i=1}^{n}\Sigma_{j=1}^{n} f_i(x_i x_j)^2 f_j + \Sigma_{i=1}^{n}\Sigma_{j=1}^{n} 2 f_i x_i x_j f_j$$

By separating the summations we can easily see how the first and second terms must always be non-negative as they are both the sum of squares. The third term involves $2(x_i x_j)$, which contributes to the overall sum non-negatively when squared (as in Equation 3).

Thus we have shown that with the given Kernel function, **f'Af** will be the sum of non-negative values resulting in a value that is always greater than or equal to 0

3.2 Soft-Margin Linear SVM

3.2.1 $C = 0$

If $C = 0$ then we only care about the size of the margin (minimizing $w^T w$). In this case, there is no penalty for margin violations and any or all points could be selected as support vectors (as the margin may be wildly inaccurate. The only support vectors we would need to use in 1 dimension are the two points bordering the decision boundary the closest, so at minimum these would be selected.

Since are data is clearly separable in one dimension between points -1 and 0, they would function as our 2 support vectors surrounding the decision boundary, or point for one dimension, in between them. This could lead to margin violations or underfifting, but is probably fine given our easily separable one-dimensional data.

3.2.2 $C \to \infty$

When $C \to \infty$, it means that margin violations and data points close to the margin are heavily penalized by the model, making it sensitive. In this case our model will likely use any points within or in violation of the margin as support vectors while having a very narrow and selective margin.

In the case of our one-dimensional data, the points closest to the margin will definitely be used as support vectors (-1 and 0). Our model may include more points depending on the determined margin width. If it assumes a third support vector, this will most likely be at the data point 1 since their are more positive than negative data points.

3.3 Code

3.3.1 Filling in SVM_soft.py

See SVM_soft.py and associated files in the svm folder to view my implemented code.

3.3.2 Testing C Values

See svm_discrim_func.py to view my code edits. See CatherineBakerQ3.py to view the code that runs this function. It imports the data, calls SVM_soft(), and calls ssvm_plot() for all given values of C.

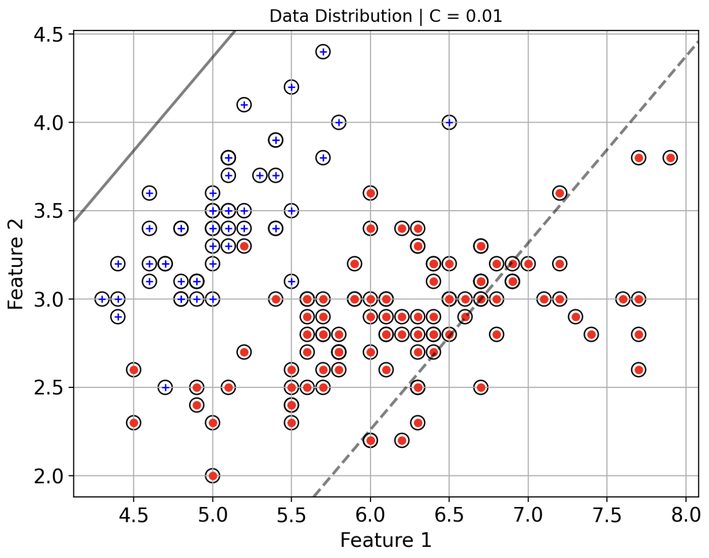

Figure 5: The plot produced by my CatherineBakerQ3.py code using SVM_soft() and ssvm_plot(). For this plot, C = 0.01.

Throughout our graphs we see our margin and decision boundary come closer to linearly separating the red circles and blue crosses as C increases. This makes sense given our knowledge about C. C is our slack penalty variable and the higher it is, the higher penalty slack variables will receive making our margin smaller, removing support vectors, and making the decision boundary more exact. If you set C too high then the model could overfit to the data, so it's important to find a comfortable C value.

As our C increases we see less and less slack variables included in the margin which allows our margin to narrow it's area creating a more accurate decision boundary line. Our highest C = 100 is likely a bit overfit to our data while our lowest C = 0.01 is clearly underfit to the data with a margin much too large. Our optimal choice of C hear depends on the needs of our data but could easily be C = 1, 10.

3.3.3 Test Error

Please see CatherineBaker3C.py to view the code for this problem. We report the following test error from our script:

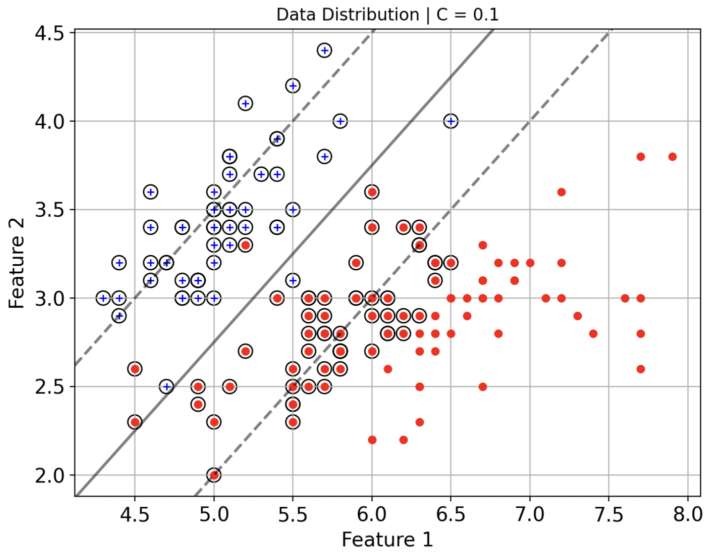

Figure 6: The plot produced by my CatherineBakerQ3.py code using SVM_soft() and ssvm_plot(). For this plot, C = 0.1.

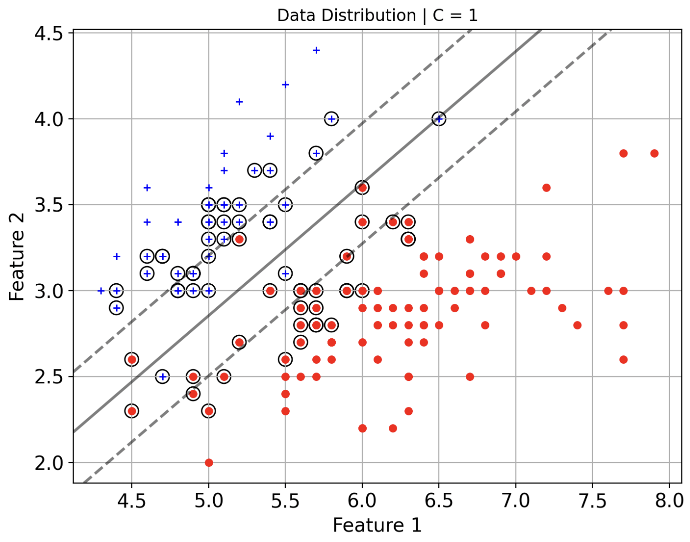

Figure 7: The plot produced by my CatherineBakerQ3.py code using SVM_soft() and ssvm_plot(). For this plot, C = 1.

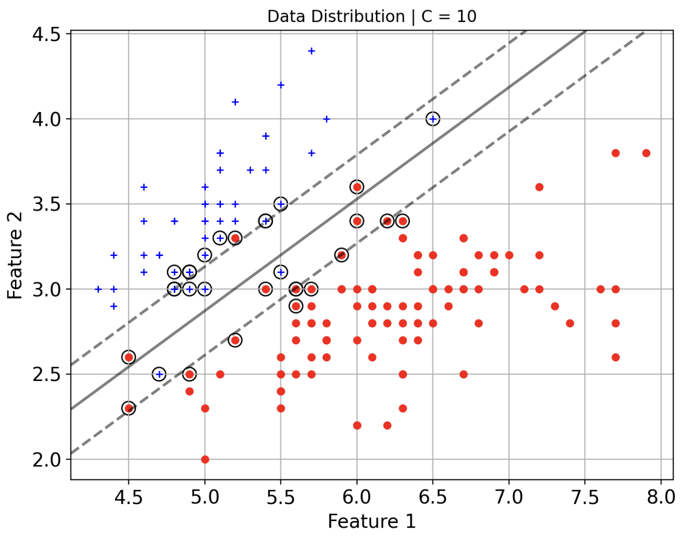

Figure 8: The plot produced by my CatherineBakerQ3.py code using SVM_soft() and ssvm_plot(). For this plot, C = 10.

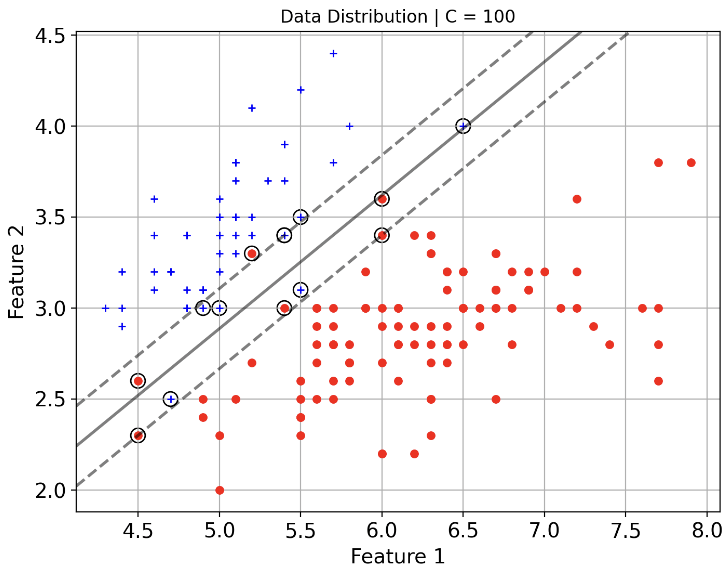

Figure 9: The plot produced by my CatherineBakerQ3.py code using SVM_soft() and ssvm_plot(). For this plot, C = 100.

Average test error: 0.03205645161290323
This is low enough for us to feel confident in our implementation.

4 Notes

I am using a late day for this assignment.