Skidmore College

CS 316: Machine Learning
Fall 2023: Lab 4

Travis McVoy

January 17, 2024

Abstract

In this lab we aim to find a classifier for heart's disease. We will use resting beats per minute and cholesterol levels as training data for a perceptron learning algorithm and several logistic regression algorithms via gradient descent.

1 Gradient Descent

We begin by examining the results using various loss functions and gradient descent.

1.1 Least Squares

For reasons unknown, the gradient descent algorithm did not update at all when using least squares. One can verify this for themselves using the `testGD()` function and setting the loss type equal to 1. I tried increasing the step size, and even with a step size of 1 billion, there is no update to the loss. I suppose it could be the case that the surface formed by the loss function is effectively a flat plane with zero gradient everywhere. Other than that, I have no explanation as to why least squares never updates. Fortunately, we have other methods besides least squares.

1.2 Cross Entropy

Cross entropy is not really an improvement from least squares because it returned a loss that is "not a number" after a single iteration. It might be the case that our data needs to be rescalled. If we print out the training data, the vast majority of our instances contain

numbers that are at least 100, which means when caclulate the sigmoid function, we are calculating e^{-x} where $x \geq 100$. Such a number could cause problems. Even though we are taking the log of the sigmoid function, we aren't using a symbolic math solver, so the sigmoid function is calculated first, then logs are taken. Hence, the machine is probably taking the logarithm of a number close to 0, which will cause everything to blow up.

1.3 Soft Max

Soft max is our saving grace. Soft max will run and update. However, it is imperitive we use a small step size. I tried 0.05 initially, and the loss osciallted between small and large values. Once I decreased the step size to 0.0001, we began to see desireable updates.

1.4 Limitations of Gradient Descent

I actually struggled to get gradient descent working at all. When converting between nd arrays and tensors, there can be problems with conflicting data types. In particular, when trying to compute the dot product, I frequently came across an error in which one of the data types was a long and one was a double. I first tried casting my weight tensor with `wt.long()` or `wt.double()` but I then ran into a new error: the machine seemed to "forget" that I required the gradient for my weight tensor. The trick is to require the double format when initializing the weight with the `dtype = torch.double` keyword argument. This is not the end of conflicting type problems, though.

When testing our solutions, we again compute the dot product between a weight vector and our testing data. The testing data, for me, was still an nd array and the weight vector is obviously a tensor. We could convert the test data to tensors, but there are multiple test data arrays so I tried converting the tensor to a numpy array. Apparently one needs to make use of the `detach()` method or the conversion won't work properly. Don't ask me why, I don't know the answer.

In addition to type erros, gradient descent has efficiency problems. We learned in the last lab that gradient descent is incredibly slow. Before I even bothered to get meaningful statistics with gradient descent, I tried using the perceptron and concluded the data isn't linearly separable (which, I believe, matches Kaelen's results). I then decided that getting enough evidence to argue that the data is indeed not linearly separable is better than doing tests with soft max.

2 Perceptron

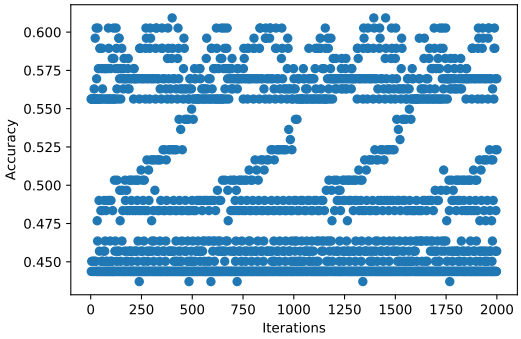
When working with the perceptron, I found that it was quite common to get accuracy that is not much better (or even worse) than just flipping a coin and assigning labels randomly. To verify that this is not a coincidence, I examined the number of iterations. Could it be that we found a decent solution and rushed right past it? I susspect not. I took 100 samples in two different ranges. We examine both separately.

2.1 The 1-2,000 examination

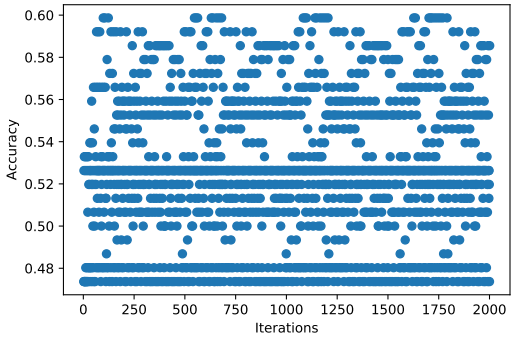
I first started by examing the training and testing accuracy when we have 1 iteration, 2 iterations, and so on up to 300 iterations. The upper bound 2,000 was not chosen arbitrarily. In lab 3 we found that the perceptron consistently terminated prior to 300 iterations when data

is linearly separable, but there were some outliers requiring iterations in the thousands. If the data is linearly separable, we would anticipate finding a solution in the 1-2,000 iterations range. Our results suggest the opposite, however:

Training:



Testing:



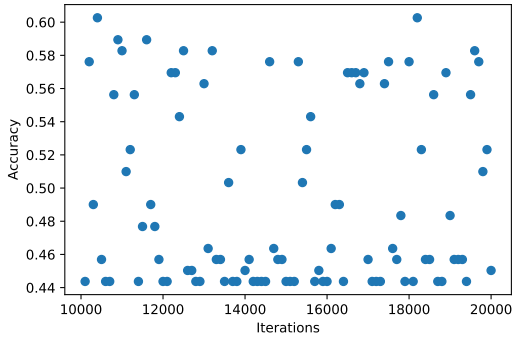
We regularly got accuracy readings lower than 50% and the highest we seemed to achieve was around 60%. Just to be safe, I continued my examination with more iterations.

2.2 The 10,000-20,000 examination

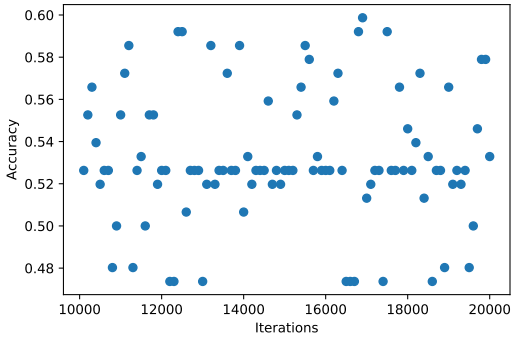
Since we didn't find anything all that great in the 1 to 2,000 range, I thought we should test two kinds of bigger numbers: bigger numbers that are still relatively close to 2,000 and numbers that are significantly larger than 2,000. We save the significantly larger numbers for the next section.

I chose to do fewer samples in the larger ranges, as learning time will only get longer and I was already fairly convinced the data wasn't linearly separable. I added 100 to the max number of iterations every increment for a total of 100 increments and got the following plots:

Training:



Testing:

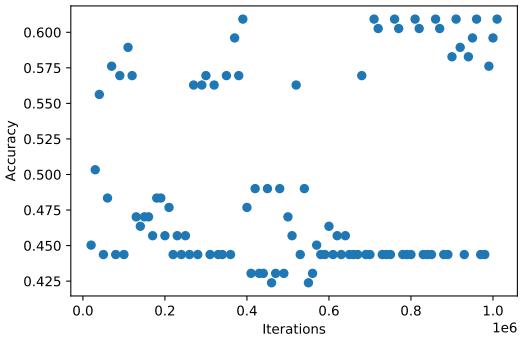


Everything still seems to indicate unseparable data. We conclude with one final examination.

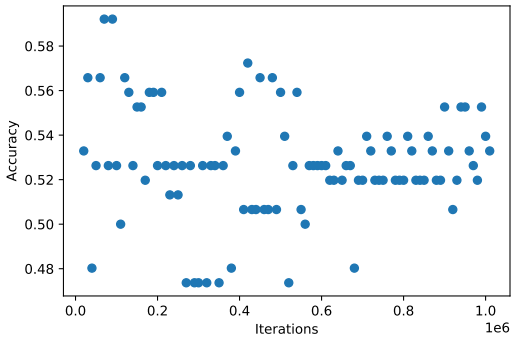
2.3 The 10,000-1,000,000 examination

For the final examination, I still used 100 increments, only they are even larger than before so we can hit the nice, clean number of 1 million iterations. Like before, the data seems to be unseparable:

Training:



Testing:



Interestingly enough, we did get some of our best training accuracy in the 10,000-1,000,000 range. Near 400,000 iterations and between 800,000 and 1,000,000 ranges there were several occurences of accuracy readings slightly above 60%. Better than flipping a coin, but still not great.

3 Conclusion

Our models are not strong enough to actually diagnose a patient. They are maybe strong enough to implement an app that tells a patient if they should schedule a vist to the doctor. It is perhaps worth asking if we would get better results if we tried replacing one of the variables with another type of medical metric. That is, it might be the case that we need to determine if our variables actually have any correlation with heart's disease.