Machine Learning Linear Classification Report

Christian Eid

December 2021

1. Linearly Separable Data

The goal of these machine learning algorithms is to correctly classify data. The data we want to classify consists of a vector with two values

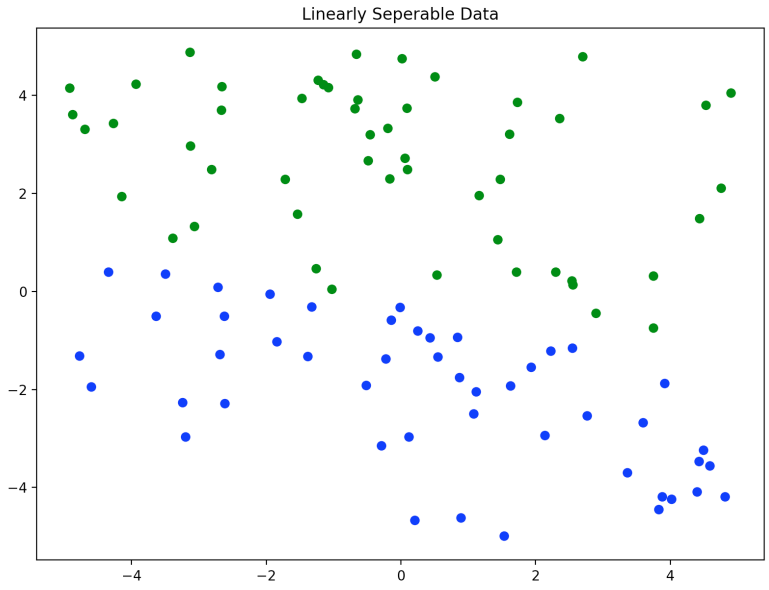
$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

We can generate a random weight vector

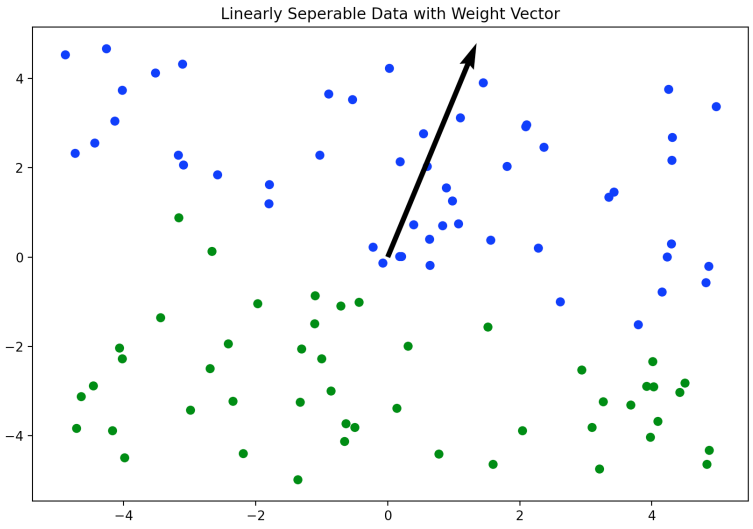
$$w = \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix}$$

By adding a 1 in x such that $x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$, we can do the dot product of x and w. If that dot product is greater than 0, then we classify the data as a 1. If it is less than 0, we classify it as a 0. (Or as -1, depending on the algorithm we use to learn.) The label is y. So, if $x^T w > 0$ then $y = 1$ By doing this for every training data, we have created a set of linearly separable data. The goal of our algorithms is to learn a weight vector w that correctly classifies the data. In doing so, we can predict the classification for any new training instance.

If we were to plot the x values using x_1 and x_2 as our axes, and color the point depending on the weight vector classification, we would obtain a graph that looks like the the following.



If we were to plot the weight vector that correctly classifies the data, it would show us that the line perpendicular to that weight vector will classify the data.



2. Loss Functions

We can measure the success of our algorithm's weight vector by using a loss function, which uses the predicted value of y and the actual value to calculate a loss. The negative gradient of the loss function tells us how to change our weight vector to better classify the data, since the negative gradient points to the lowest loss.

In this program, I use three loss functions: Least Squares, Cross Entropy, and Soft Max.

3. Gradient Descent and Algorithms

The gradient descent algorithm generates a random weight vector, and uses the negative gradient of a loss function to gradually change the value of the weight vector until the loss is minimized. In addition to gradient descent, I use the perceptron learning algorithm, which uses misclassified data to learn a weight vector until all data is correctly classified. I also use linear programming to learn the data.

4. Data

I generated 100 random training samples and classified then with a random weight vector. I then used these 100 training samples in the 5 machine learning algorithms. I repeated this process 100 times, keeping track of the time each algorithm took, the success rate, and the number of iterations.

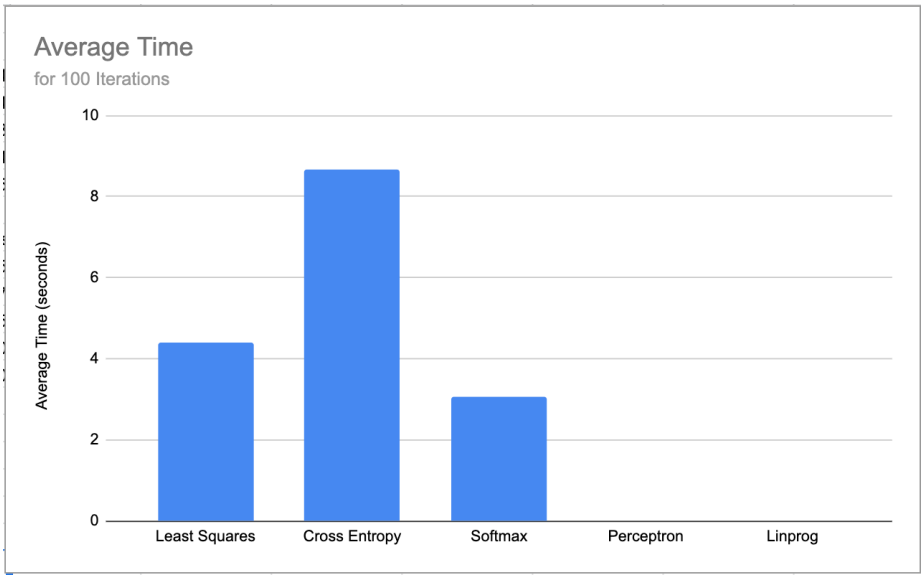
Test 1: Time and Success Rate for 100 iterations

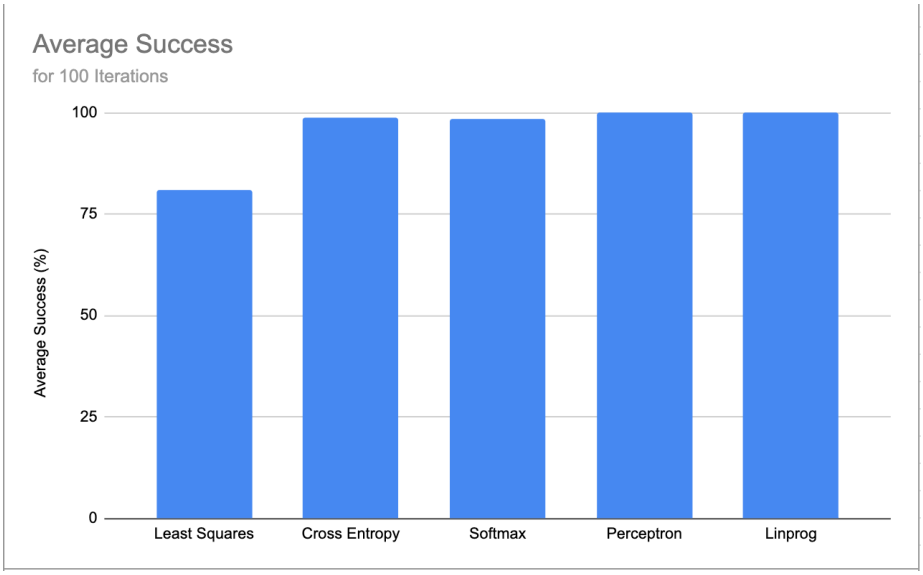
In this test, I ran the gradient descent algorithms for a consistent 100 iterations. I ran the perceptron and linprog as normal. The step size for the gradient descents were set to 0.1.

Algorithm	Time (s)	Success Rate (%)
Least Squares	4.392525024	81.05
Cross Entropy	8.65665353999999	98.7
Soft Max	3.0731506798	98.55
Perceptron	0.00794812739999581	100
Linprog	0.0107406322500022	100

Note that the perceptron and linprog algorithms will always result in a success rate of 100 percent, since the stopping condition is that it correctly classifies all the data. On average, the perceptron takes less time than linprog. The gradient descent algorithms all take substantially more time, and result in lower success rates. Gradient descent using the soft max loss function takes the least amount of time, and results in a highly accurate success rate. Cross Entropy loss function takes a substantial amount of time at an average of 8 seconds, and yields a similarly high success rate. Lastly, using the least squares loss function, gradient descent takes an average of 4.39 seconds, which is slightly higher than soft max, but yields a low success rate. The below plots show the success rate

and time each algorithm took.





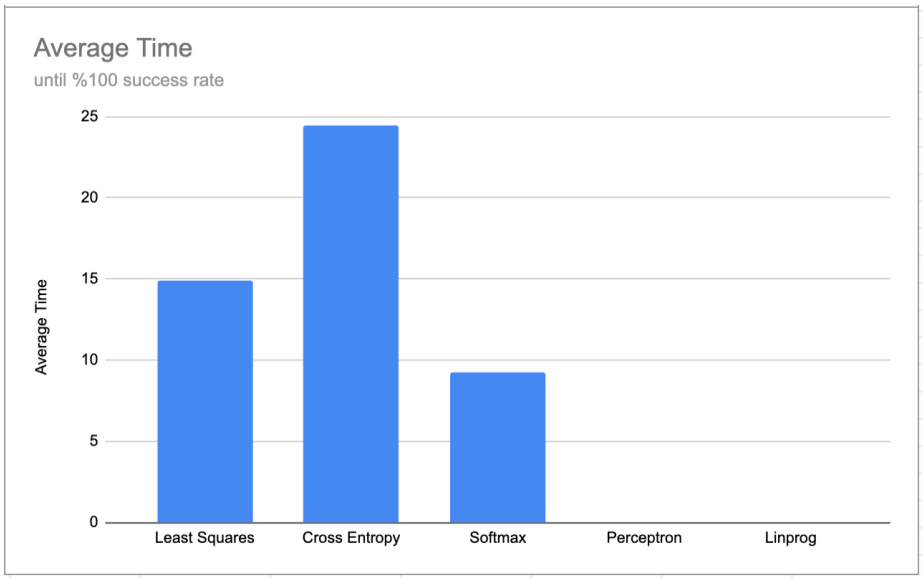
Test 2: Correctly Classifying the Data

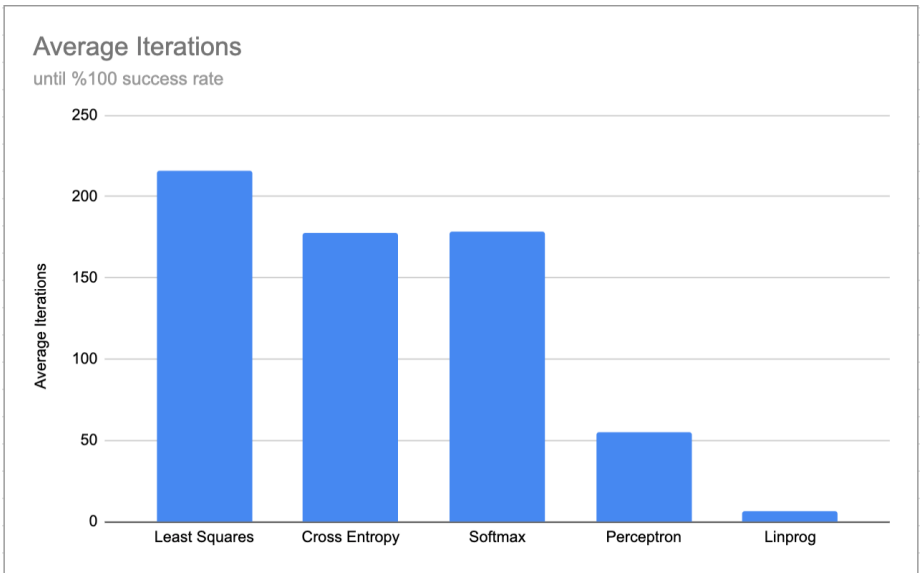
In this test, I ran the gradient descent algorithms until they correctly classified 100 percent of the data. I kept track of the time that each algorithm took, as well as the number of iterations that it went through. The step-size was increased to 1 for this test. If the gradient descent algorithms do not classify

100 percent correctly after 500 iterations, they are halted. They are stuck and will usually keep going. Note that the perceptron will be said to iterate every time the weight vector is updated. However, it loops through the data checking for misclassified instances many more times than that.

Algorithm	Time (s)	Iterations
Least Squares	14.9149388	215.78
Cross Entropy	24.44747901	177.92
Soft Max	9.215968336	178.5
Perceptron	0.01638689704	54.92
Linprog	0.01209723686	6.68

The data shows similar time trends to the previous test. Perceptron and linprog both take around 0.01 seconds to complete. The linprog iterates only an average of 6.5 times, which is very small compared to all the other algorithms. Perceptron updates the weight an average of 54.9 times, which is the second smallest. Perceptron loops through the training set an average of 1541.24 times, meaning it goes through that many training instances searching for misclassified. The gradient descent algorithms all take a significantly larger amount of time to correctly classify all instances. Soft max takes 9 seconds, which is the least. Least squares takes 14 seconds. Both of these algorithms occasionally reach 500 iterations without classifying all the data. In these cases, the algorithms will always result in a success rate of 99 percent. Cross entropy takes an average of 24 seconds, which is extremely long, and will sometimes fail to classify the data in the cases that it reaches 500 iterations. See below for a visual representation of the data .





5. Conclusion

In conclusion, the gradient descent algorithms take more time and lead to a lower success rate then the perceptron learning algorithm and linear programming algorithm. The perceptron and linprog will always succeed at classifying the data, and do so very quickly. If you want to correctly classify every instance using gradient descent, it will take very long. Alternatively, you can set a stopping condition to run faster, but miss some of the data. There is a trade-off for accuracy and speed, which is not present in the perceptron and linprog. Perceptron and linprog seem to be much more efficient at learning to classify data.