Brandon Luong

Assignment 2

Project Machine Learning using Bitcoin's data has been going smoothly. I chose to get information from a bitcoin exchange called "GDAX." The data is from Dec 1 2014 - Oct 3 2018. I choose to stick with the daily information on Bitcoin, ticker (BTC) because the amount of data on the shorter time frames would be too overwhelming to work with. I also noticed that because BTC's volatility is higher than most stocks that choosing a larger time frame would smooth out a lot of the volatility. The dataset has over 1400 entries so I figured it was substantial enough to get good enough information without requiring too much computing power.

So after choosing the time frame and picking the exchange, the next thing to do was to examine the data. I plotted it and noticed immediately even the daily prices were too volatile and would not make for a good graph. I then decided to try to calculate the moving averages, I chose to stick with the 10 and 22, I chose those averages as they are the most common moving averages when tracking BTC. I put all the data into Python's library Pandas and calculated the simple moving average, (SMA) of the 10 and 22. I then appended these to the original data and exported a new CSV called "modified.csv."

I graphed the SMA 10 using Matplotlib, another Python library. The graph did look a lot smoother but I thought predictions using this model would still not produce the optimal results. I then tried another value for price which is the natural log (LN) and this made a much smoother and less volatile curve. I exported this new file out as "modified_w_ln.csv." From there I did a simple linear regression using gradient descent using only the time and LN of the prices. My initial values for theta0 and theta1 were 0 and I set the learning rate to .5. I ran this for 1500 iterations batch gradient descent.

The initial results were very poor and my Python IDE gave an error of overflow. To debug, I decided to print the cost function to make sure that the cost was both decreasing, and that it was converging. The cost was increasing rapidly so I knew the learning rate was set too high. I played around with different values between .00000001 - .1 and finally settled on .000002. The reason I chose this specific number was it was the highest learning rate I was able to obtain while still having the cost decrease and converging.

I let the gradient descent run a few times and tweaked a couple of values, but noticed that the cost

and the prediction it produced was still fairly inaccurate. I decided to graph the prediction and saw that though the slope was changing rapidly the y intercept was not. This probably had to do with me not normalizing and feature scaling the initial parameters values. I helped the algorithm along by setting the initial values theta 0 and theta 1 manually. I played around with it a lot and eventually settled on theta 0 = 4.8 and theta 1 = .5. I ran the descent for 1500 iterations and got a final output of: {theta0 : 4.799491281419004, theta1 : 0.0032453101562049543}. I ran these numbers which were LN against the actual closing high and still the predictions were fairly accurate.

A lot of the time of the project was spent learning about the different libraries in Python and playing with different parameters. The next steps I need to do is: figure out what other features I want to include, normalize the data, and to do feature scaling. So far gradient descent is working fairly well so I will be sticking with that algorithm. The current cost of the prediction - actual is -0.007520522761847905 so this is a good starting point.