README.md

# CS230

***Description***: This NN uses social media data to predict price change among cryptocurrencies. The goal of the algorithm is to predict positive change with high accuracy, and predicts over the below-mentioned time periods.

First, the algorithm aggregates all the data from every currency to train a general model with very low variance. This model is used to initialize custom models for each currency. Each currency is then trained on it's own unique dataset. This approach takes advantage of the massive amount of data availible across all currencies for better training, yet still mantains the advantage of customization per currency. It also helps to avoid overfitting.

Additionally, once the primary model is trained, the customized models will be trivial to execute.

Please see the project paper in this repository for a more comprehensive description.

Some unique facets of this algorithm include:
***Data Clensing***: Instead of just mining tweets that contain mentions of the currency we're looking for, we're going to parse the tweets for spam and promotional tweets. Tweets with the objective of promoting currencies should not be analyzed. To clense the data in this way, we will use prebuilt algorithms by other engineers.

***Sentiment Analysis***: All tweets will be analyzed with a standard sentiment analysis, and their relative importance will be multiplied by their sentiment.

***Weighted Importance***: Tweets will be weighted by how popular they are, as a metric of retweets and favorites.

***Modified Time Data***: All the data will be mined per minute. We will then use weighted averages over time period m, to transform this data into a set of data. This will allow the algorithm to find the relationship between the last hour's tweets, and the change in the following minute. This creates a much more dynamic set of data. The same thing will be done with output, and the weighted average time period will simply be changed to predict for different time periods.

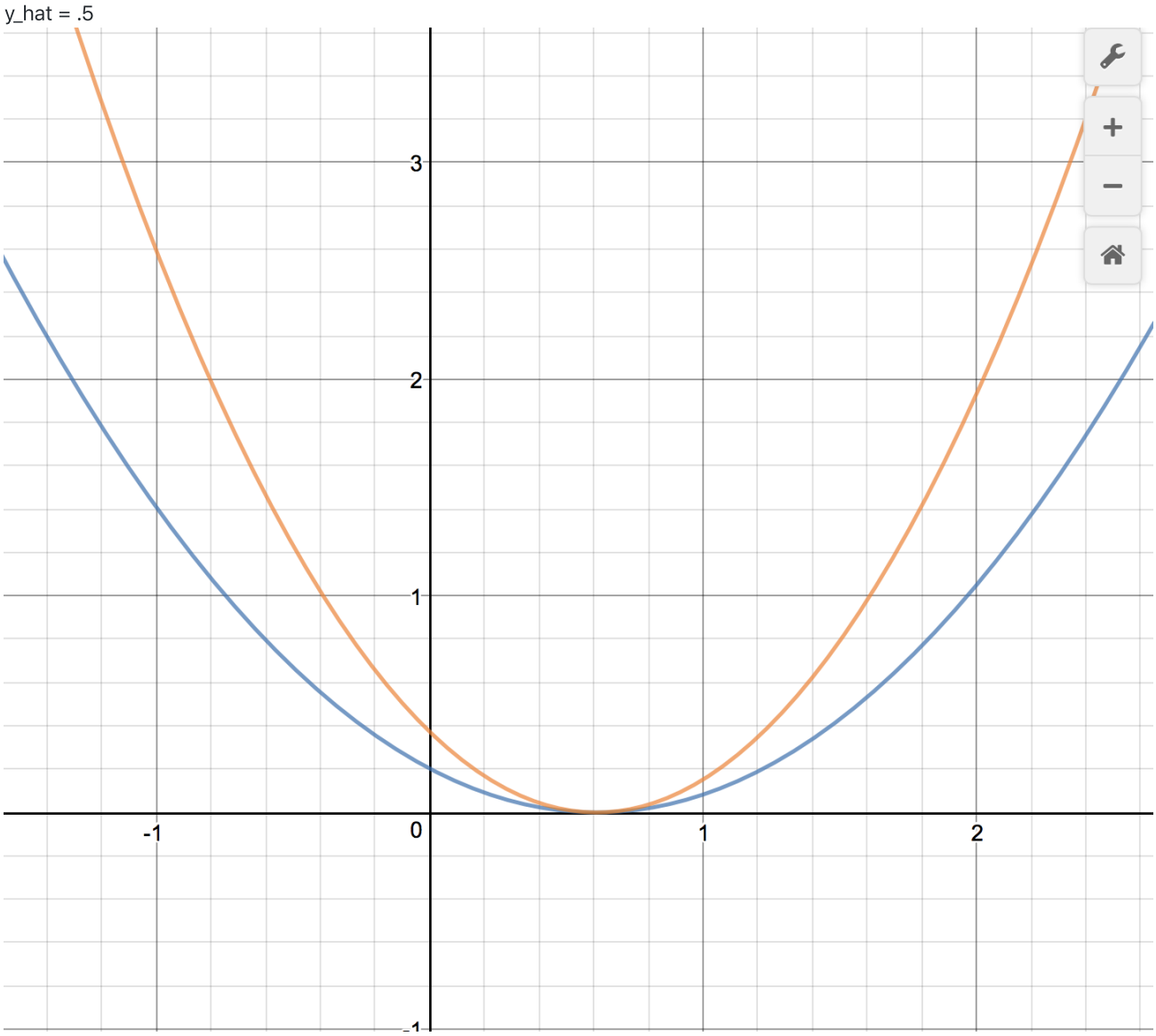
This modification is extremely important as it allows for significantly more training & label data. For example, a normal algorithm to predict a days price change of a specific currency will have 1000 data points relating to 1000 days. For those same 1000 days, my algorithm, has 144 million (1000*24*60*100) data points.

***Depth***: The aforementioned modefication of the data allows for a vastly expanded network. With millions of pieces of data the main training model can be much deeper. This allows for an examination of relationships previously uncapturable by Neural Networks. Though a very deep network is not in itsef a unique feature, the depth of this network in relation to the type of data is novel.

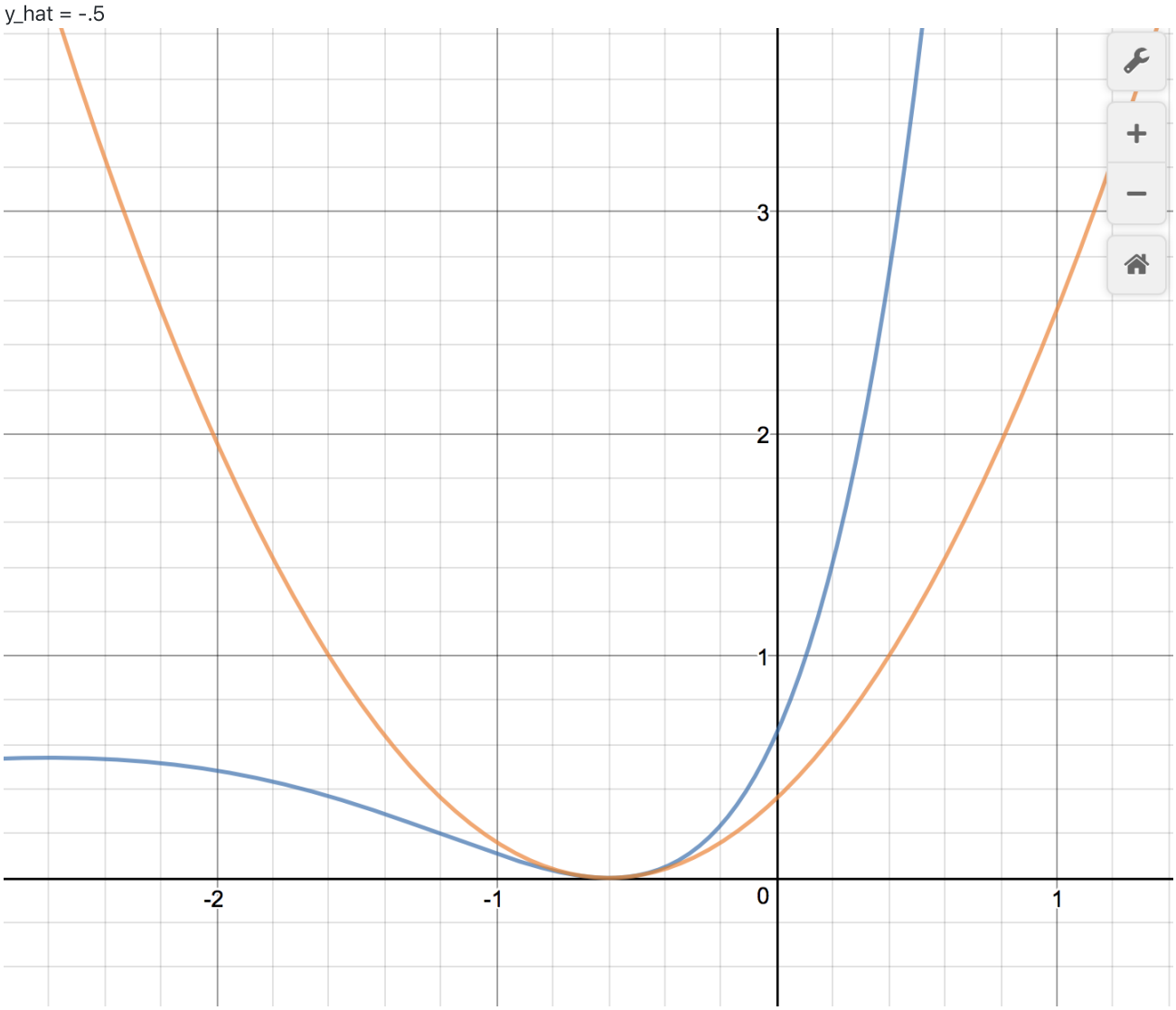
***Unique Loss Function***: I do not use a traditional loss function. The goal of this algorithm is to predict positive change. I do not care if the price of a currency is going to drop, because I cannot make money off a currency dropping. I do however care about the accuracy in predicting positive change. When the algorithm predicts positive change, it should be very confident. We can achieve this by creating a loss function that increases penalties when the actual change is negative. I created my own loss function that penalizes wrong negative choices greatly, but gives lower loss values to accurate positive predictions. This trains the model to predict positive change effectively.

The loss function is as follows: X = Prediction Y = Actual L(X, Y) = (Y - X)^2 • e^(sigmoid(Y)•x - Y)

Images of the updated loss function versus a standard least squared loss are below: X Axis = Prediction Y Axis = Loss


y_hat = .5
3
2
1
0
-1
-1
1
2


y_hat = 0
3
2
1
-1
0
1
2




***Test Metric***: To test the algorithm's performance, negative predictions do not matter as they will be ignored in trading. Additionally, the only predictions that matter are those which are most confident. If my algorithms makes 500 predictions, there only needs to be one or two great predictions. Therefore, the test metric is two-fold, and designed to optmize for best-case predictions.

The first test metric, for the general model, is calculating the percent of predictions that are correct within one standard deviation of the norm.

The second test metric, is on each currency's customized model, is calculating the percent of positive predictions that are correct within one standard deviation of the norm, and the percent of negative predictions that are negative.

## Data

### Sources

A) Aggregated&Sentimented Tweet data

B) Aggregated&Sentimented Tweet data only for top crypto influencers

C) Aggregated&Sentimented Reddit data

D) Aggregated&Sentimented Reddit data for top influencers

E) Aggregated&Sentimented News data for top publications

G) Previous price behavior

H) Trade volume
All data is per minute. Ideally, we're looking for ~10 Million Tweets, ~100k Reddit posts, ~100k News articles

## Preprocessing

The process for preprocessing data is as follows:
Source A -
Step 1: Scrape twitter for tweets that mention a major currency

Step 2: Store each tweet with the tweet text, the number of likes + 2 * favorites and the timestamp

Step 3: Scale likes and favorites by time. This can be approximated by the function ~ Yarctan(1.5sqrt(x)) - 1/(1 + e^(-.1x)), where x is the time and Y is the scale factor necessary to match the rewtweet #
**This step is only really necessary for live testing on incoming data. No need to use it on mass data mining.

Step 4: Setup Twitter data into a matrix described by the columns: [tweet text, scaled retweets, time stamp ]

Step 5: Reduce noise by iterating through the matrix, and using a pretrained n-gram model to remove tweets which are likely promotional, offers, or bots [6]. Remove these tweets from the tweet matrix.

Step 6: Run sentiment analysis on each tweet, for a score between [-1, 1]. Use VADER by Hutto & Gilbert [7]. Append Matrix with these values. The matrix should now be: [tweet text, scaled retweets, time stamp, sentiment ]

Step 7: Split the data into two matrix. For matrix one, multiply the sentiment by the scaled retweets number to give a relative significance to each tweet. Append the matrix and removing superfluous data (everything but the time stamp). For matrix two, remove the tweet text and the scaled retweet metric. Each matrix should now be:
Matrix 1 - [Scaled signifiance, time stamp]
Matrix 2 - [Sentiment, time stamp]

Step 8: Combine tweets over each minute. Ie, anything tweeted in the same minute should be linearly-combined. For any minutes without tweets represented, the algorithm should just put 0 (though hopefully with enough data this should never be the case). The matrix are now vectors:
Vector A: [Aggregate scaled signifance/per minute]
Vectoer B: [Aggregate sentiment /per minute]

Step 9: Create time vectors with moving averages over each minute. Create a moving average for each time frame, and have each one be a new vector. This results in 12 permutations of vectors A and B.

Step 10: Remove the first 7 days of tweet data from each vector in order to keep time alignment between moving averages. This will also remove the need for a bias in the moving average.

Step 11: Divide each vector by the set average to keep values more normalized to train faster.

The finished product should be a set of 24 vectors which represent aggregate twitter sentiment for a single currency. The most recent month of tweet data should be placed into the test set, the rest should be for training. This process should be repeated for the top 100 major currencies. Each of those 24 vectors should be appended by each currency. It is CRITICAL that currency order is maintained through training, dev and test examples. If order is lost, the entire training algorithm FAILS.

Store all datasets for each currency seperately. They will each be individually used later.

Source B - Repeat all of steps A, but only include data from a hand-picked set of 30 influencers.

Source C - Repeat all of steps A, but substitute retweets for upvotes.

Source D - Repeat all of steps C, but on select threads

Source E - Repeat steps A, yet on only a set of ~40 news outlets that report on cryptocurrency. No need to reduce noise in step 5.

Source G - Take the bitcoin output data, shift it down by one. Execute steps 9 and 10 to create different time sets for training on different time periods. However, every training algorithm will only be trained for one time period.

Source H - Use common online sources to pull the readily availible data. Execute steps 9 and 10.

## Training Details

On the main training model parameters will be initialized using Xavier initialization. We will use RELU activation functions in order to preserve the importance of extreme change.

Furthermore, on the main training model we use a number of regulation techniques in order to lower variance.

On the custom training algorithms we will use less regulation techniques as overfitting is less of a concern.

The algorithm will start with 50 hidden layers with 50 nodes in each layer. I'll see how this works and adapt accordingly.

## Initial Steps

First run the general model on prediction for one minute. Then use transfer learning on each mini training set to get the predictions for each smaller currency for that training set.

Next, retrain the model for each time period. Each of these will produce a different weight matrix. Rerun each mini training set on each time period.

This will yield a dictionary of results. The dictionary will be: {Currency name, vector of prediction accuracy for each time period]}

Successful results will be evaluated by the highest test accuracy for any time period, on any currency.

## Time Periods

1 minute, 5 minutes, 15 minutes, 30 minutes, 1 hour, 2 hours, 4 hours, 6 hours, 12 hours, 24 hours, 2 days, 7 days

## Disadvantages to this approach

Because this approach optimizes for positive predictions and optomizes for false negatives, not false positives, it will likely miss some bull periods in the market.

## Sources

1. http://cs230.stanford.edu/files_winter_2018/projects/6940331.pdf
2. http://cs230.stanford.edu/files_winter_2018/projects/6927076.pdf
3. http://cs230.stanford.edu/files_winter_2018/projects/6929537.pdf
4. https://arxiv.org/abs/1610.09225
5. https://kth.diva-portal.org/smash/get/diva2:1110776/FULLTEXT01.pdf
6. http://stats.seandolinar.com/twitter-retweet-decay/
7. C.J. Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In Eighth Inter- national AAAI Conference on Weblogs and Social Media, 2014.