Cryptocurrency Investment Strategy Estimator using Historical Price Data

Kaan Yeşildal

Bilkent University, 2017

TABLE OF CONTENTS

1) Problem Description

Financial prediction is a multidisciplinary subject studied by economists, statisticians and computer scientists. Many methods are developed to predict behavior of various financial instruments such as stocks, bonds, fiat currencies. With rise of machine learning and data science in the 21st century, financial prediction became an important field of interest of these disciplines. For the past 5 years, cryptocurrencies became a highly speculated financial instrument. The project focuses on application of machine learning methods for the most popular cryptocurrency Bitcoin (BTC) forecasting.

Cryptocurrencies has been an instrument of speculation from their inception. Although they are introduced as the financial infrastructure of future's economies, their real use has only been that of an agent of investment. As of today, there have been limited use for these currencies and the spike of their prices indicate that the reason for this high price range can be pure speculation. As of last 2 years the number of applications using bitcoin has been increasing at a steady rate but the price of bitcoin has increased 10x.

In the light of these information there is compelling evidence to believe that the oscillation in the price might only have countable number of reasons behind it. These can be news about the price and the industry, buyer and seller hype, developments of the infrastructure etc. However, the price of cryptocurrencies is highly volatile than any other fiat currency or any stock. Because we have vast amount of data about the volatile behavior of these currencies, we can isolate them from other effects that might affect the price in short term and investigate the behavior of the volatility to extract trader behavior about them.

2) Dataset Description

The dataset obtained and described in the Phase 1 Report includes the historical BTC market data at 1-min intervals for short-term prediction since the birth of BTC (13.09.2011). This early dataset contains Timestamp, Open-Close Price, Max Price, Min Price and Price by each minute. For Phase 2, it is decided to drop off all unnecessary data except closing price of each time sample which is a minute. By doing this the dimensionality of the input is reduced and the further calculations are simplified. However, for long term prediction we pre-process the 1-min interval dataset to transform it into 1-week interval dataset.
The dataset for both short-term and long-term prediction is divided into two parts training, and testing.

2.1) Training Data

The short-term training data is consist of the past 6 months of BTC price with 1-min intervals, in fact, 15.552.000 data(BTC price) instances will be utilized.

The long-term training data is consist of all the available data(about 5 years) of BTC price with 1-week intervals, in fact, 1825 data(BTC price) instances will be utilized.

2.2) Test Data

The test data will be consist of the past 2 days of BTC price with 1-min intervals, in fact, 604800 data(BTC price) instances is utilized.

The test data will be consist of the past 6 months of BTC price with 1-day intervals, in fact, 180 data(BTC price) instances will be utilized.

2.3) Feasibility of Dataset

If we analyze the ratio of number of training and test data: for short-term prediction it is 90 and for long-term prediction it is 10. It seems sufficient for many machine learning algorithms to be applied and it is also observed that professional financial prediction institutions utilize close ratios.

3) Dataset Preprocessing

Preprocessing of the dataset is one of the very important part of the project. It is decided to process important trends in the available dataset rather than processing bitcoin prices depending on the time series. This approach made serious impact on how we approach the problem. First of all, we get rid of the time series dependence of the project that is described in Phase 1 Report. Below, the preprocessing of the data and the reasoning behind it is explained.

3.1) Preprocessing for Regression and Classification Trees

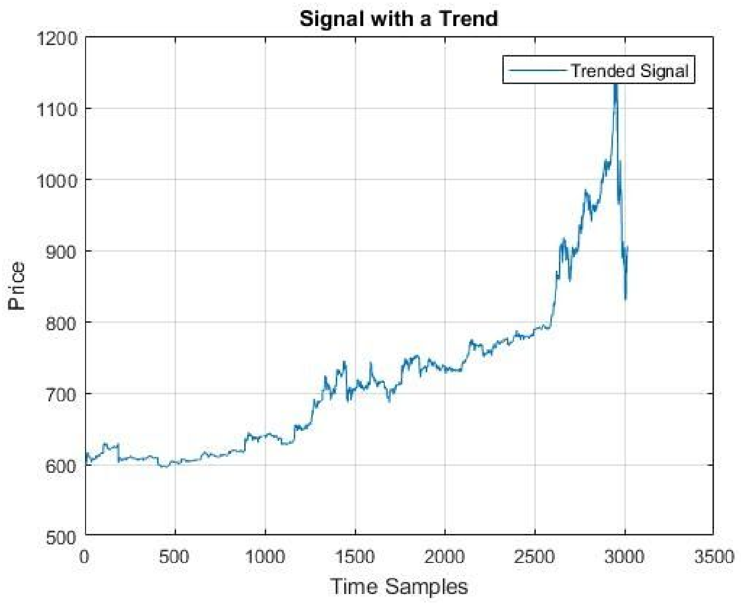
3.1.1) Trend Analysis in Financial Data

A trend analysis is an aspect of technical analysis that tries to predict the future movement of a stock based on past data. Trend analysis is based on that what has happened in the past time gives an idea of what is going to be happen in the future. There are three main types of trends: short-, intermediate- and long-term.

A trend is the general direction the market is taking during a specified period of time. Trends can be both upward and downward. While there is no specified minimum amount of time required for a direction to be considered a trend, the longer the direction is maintained, the more notable the trend.

3.1.2) The Method Developed for Finding Trends(Modified QRS Analysis)

To specify trends, positive peaks and negative peaks or holes must be find. However, the financial data is full of peaks and holes embedded into them. Note that it is also very noisy since there is a continuous change in the data. For instance, below there is an example BTC data from the dataset:



.

Figure 1: The Financial Data(Price vs Time) of 3000 Minutes

Here is the zoomed version of the BTC price data by 1 minute intervals, it is clear that the data is very noisy and it is not possible to take any trends without setting any threshold for finding peaks and holes.

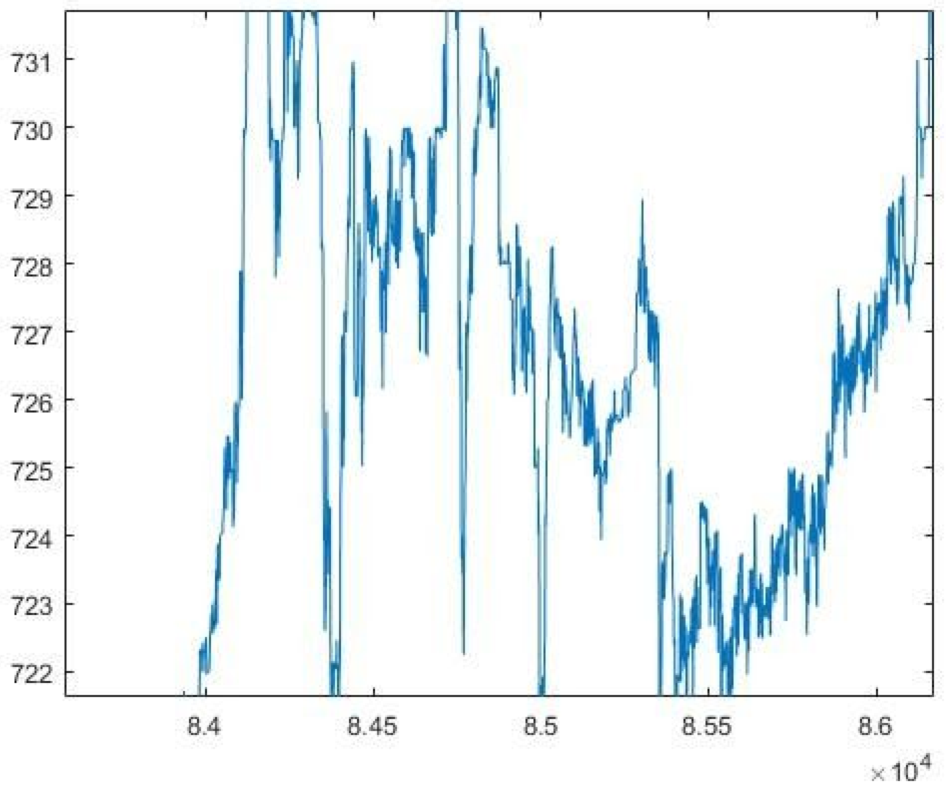


Figure 2: The Zoomed Financial Data(Price vs Time) of 3000 Minutes

By using prominence feature of MATLAB only the important peaks and holes are selected. This process is inspired by QRS analysis of biomedical signals which handles with very noisy data with spike-like features to extracted such as ECG signals. All in all, the obtained peaks and holes are shown below.

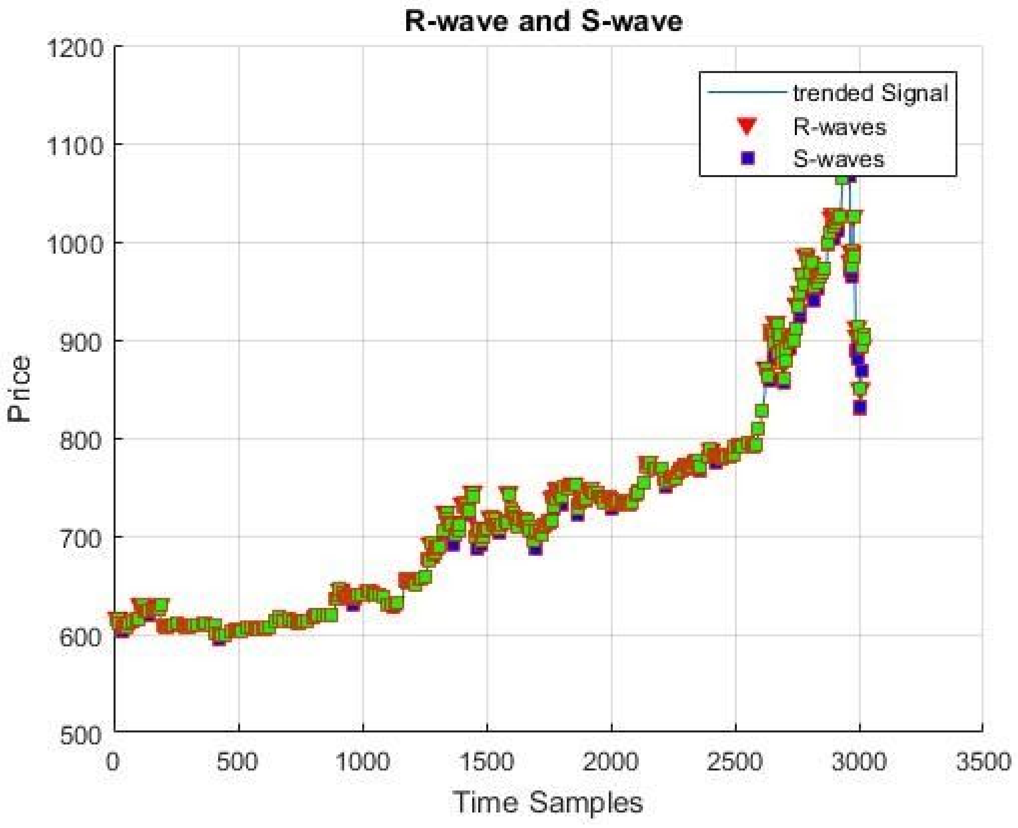


Figure 3: QRS Analysis of Financial Data of 3000 Minutes

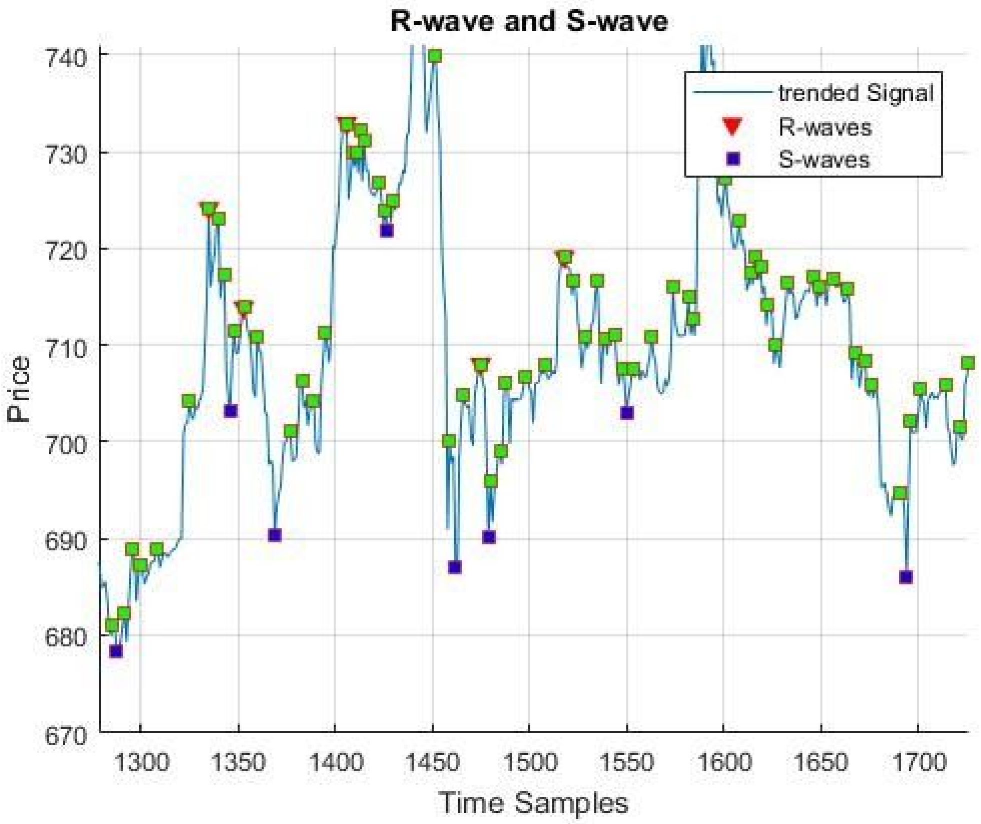


Figure 4: Zoomed QRS Analysis of Financial Data of 3000 Minutes

Above R-waves(red spots) indicates an important peak in the data,in fact, peak of many peaks or partial global maxima. S-waves(blue spots) are vice versa. Furthermore, Q-waves(green spor) indicates a lot of local maximas and minimas together between R and S-waves.

Later, we calculate slopes and euclidian distances between a R-wave(red spots) and the closest Q-wave(green spot) to each side. Similarly, calculating slopes and euclidian distances between a R-wave(red spot) and the closest Q-wave(green spot) to each side.

It is planned that by finding important peaks and holes and recording the relation between them, the trends in financial data can be obtained well. Therefore, our input data becomes a two dimensional data consisting of slope and magnitude of trends found. Note that, our size of data changes according to different financial data since there are different number of trends embedded into it.

3.2) Preprocessing for Neural Networks

Neural networks are utilized for binary classification that indicates the BTC price will either increase or decrease. First of all, whole price data is normalized for ANN to process. As mentioned, the dataset is composed of price with one minute intervals. This time only daily closing prices are taken. It is obvious that this decreases the available data significantly. To compansate it, framing is applied. In fact, 1 month, 30-day data frames are taken by sliding them one forward at each time. For instance, first frame consists of 1st to 30th daily prices, second frame consists of 2nd to 31th daily prices and so on. Therefore, we created (Frame Size)x(Number of Samples) matrix. Frame size which is 30 is the data dimensionality and the number of samples are the new size of the dataset to be implemented.

The labeling is done by the following procedure: Last element of ith frame and last element of the i+1 th frame are compared and decision made according to increase or decrease in one month. These comparisons' results are labeled with -1 for decrease and 1 for increase. -1 and 1 is selected since we will be using tanh function at output layer of neural networks that is described below.

4) Implemented Methods and Results

4.1) K Nearest Neighborhood Classifier(KNN)

In this method, the task is to convert time series price data to the typ eof data that can be classified by regular classification methods. To accomplish this task, we used the ECG signal QRS analysis as given above. This allowed us to determine the peaks in the data which are restricted by the prominence values we choose. For classification, we deterimined the R points as minimum %0.5 increase in the price. S points as %0.5 decrease in the price and Q point as %0.25 percent increase in the price. These consistent peak assignment allowed us to linearize the data.

The linearization is done by connecting Q, R and S points. For the peak values we determined above, we have only one Q point between R and S points. So we connected every R points with Q points and Q points by S points consecutively.

This allowed us to linearize the time series data in such a way that the same behaviour in the price is represented by the same positions for Q,R,S points. This allows us to use the linearized lines as features to implement classification.

We saved every line with attributes such as starting time point, ending time point, slope, and variance.

The variance is calculated by again using Q points. In this case, the minimum increase in the price is decreased to %0.10 for the Q points to represent the variance in a line instance. The number of %0.10 Q points between a line is saved as the lines variance.

After we save the time series data as linear lines with the given attributes. We determined features for the classification task.

For the price of bitcoin, we can assume that the next behaviour of the price is affected by the behaviour of the price which is at the time instance closest to the one we are looking at. This is caused because the bitcoin buyers and sellers are speculating mostly according to the price behaviour unlike any other security such as stocks and bonds where many other aspects affect the price in a given day. So for a given peak in the price, the corrections are more predictable.

So in the light of this information, we create our feature set as the 3 previous linearized time instances. So a single instance of data consists of the magnitude difference, the time difference and the variance of the line. This creates instances with 9 features.

The next task is to determine the interval of classification. For the decision tree we are creating, we need to create our training data with instance similar to each other in terms of magnitude. So for a given time instance , we chose the time points which when linearized, the magnitude of the previous 3 linearized time instances are smaller or bigger than %20 of the previous 3 corresponding linear lines.

After we create the dataset this way, we used decision tree with regression to classify the data in subcategories of these 9 features. This allows us to calculate the smallest mean squared error given by the most deep leaf of this regression tree. The target feature for this regression tree is the duration of current linearized instance we are using.

By trying to estimate longitude of the duration, we are trying to find out when will this linear behaviour will end. This will allow us to sell before the price goes down or buy before the price goes up.

By using the regression tree method, we are trying to find the mean line duration of the instances belonging to the tree leaf which our current time instance fits and also minimizes

$$RSS \ + \ a * complexity$$

where RSS is the corresponding MSE of the leaf node, and complexity is the depth of the leaf node.We know that as the MSE decreases, complexity increases. The task is to optimize alpha to have the minimum error with minimum complexity.

After finding the optimal node, our estimation is the mean value of the target feature (duration) of instances belonging in the same node.

Results :

These results are achieved using 1440 time instances serepated by 100 minutes, starting from minute 5000.

The mean accurray of the duration estimation for increasing behaviour:

%73.24

This means that for a time instance which is on a increasing line, we can estimate how long that line will increase at this accuracy.

The mean accurray of the duration estimation for decreasing behaviour:

%71.50

This means that for a time instance which is on a decreasing line, we can estimate how long that line will decrease at this accuracy.

Earnings : The earnings for total of 505511 minutes of increasing behaviour in the price are in average %0.0094 per minute.

4.2) Neural Networks

Neural Networks are artificial models of real neural systems. They consists of different layers such as input layer, output layer and hidden layers in between them. The feedforward neural networks do not have any feedback connections so they do not have memory.

4.2.1) Single Layer Perceptron:

A single layer perceptron is the simplest form of the neural networks. They consist of only input and output layer and one activation function. They are good models for classification tasks but they can classify only linearly seperable data.

The SLP that we designed has 30 inputs and one output. The network is trained for 10 times of the data size in fact there is 10 epochs. Learning rate is selected as 0.001 .The activation function is selected as hyperbolic tangent activation function. Perceptron learning law is utilized.

We applied our normalized and framed train data onto the perceptron by randomizing the input at each forward pass. The performance is validated by applying the test data with the trained weights. The results are compared with the corresponding test labels to calculate accuracy. For financial prediction it is assumed that task anything above %50 accuracy which is base success is accepted as successful. Accuracy is calculated for every test frame and the corresponding correct output which we know. The mean accuracy for every test is around %55 and the highest that we get is %63.63. The MSE of training is also plotted to see the network is converging or not.

4.2.2) Feed Forward Neural Network with Regularization:

We also implemented multi layer network to see if we can increase accuracy that is mentioned above. The same training and test data is applied with randomization. The neural network that we designed has 4 layers. There are 30 neurons at the input and 2 hidden layers and one at the output.he network is trained for 10 times of the data size in fact there is 10 epochs. Learning rate is selected as 0.001 .All activation functions are selected as hyperbolic tangent activation function. Stochastic gradient descent algorithm is utilized.

After several tests, it is found that there are vanishing gradients. These effects outputs in a bad way by making them very close to zero. Therefore, we apply a lasso alike regularization to the weights to get rid of them. We also tried to apply autoencoder but we cannot implemented it in time.

We applied our normalized and framed train data onto the the network by randomizing the input at each forward pass. The performance is validated in a same way that we do in the perceptron. The mean accuracy for perceptron was %55. For the 4 layer MLP, the highest that we get is %57.

4.3) K Nearest Neighborhood Classifier(KNN)

KNN is a supervised learning algorithms. İn fact, while there are a given labelled dataset consisting of training observations (x,y)(x,y) and would like to record the relationship between x- y. Our objective is to train for a function f:X→Y so that given an unknown observation x, f(x) are able to predict the corresponding output y.

Note that the KNN classifier is non-parametric as well as instance-based algorithm.

- Non-parametric means it makes no explicit assumptions about the functional form of h, avoiding the modelling error for the distribution of the data. For instance, suppose the input data is highly non-Gaussian but the model chosen assumes it is in a Gaussian formation. In that case, KNN would make very inaccurate predictions.
- Instance-based means that the algorithm does not explicitly learn a model. Instead, it chooses to memorize the training instances which are subsequently used as “knowledge” for the prediction phase. When we ask it to predict a label given an input), the algorithm utilizes the corresponding training instances to give an output.

The algorithm works as the following: For a new data instance, by searching through the whole training dataset for the K most similar instances also called the neighbors and summarizing the output variable for those K instances. For regression which is our task this may be the mean output variable, in classification this might be the most commonly mode class value. To find out which of the K instances in the training dataset are most similar one to new input distance measurement is made. For real valued input variables such as BTC Price, the most popular distance measurement method is to find Euclidean distance just like we utilize when we preprocess the data.

5. CONCLUSION

We applied methods such as knn classifier ,regression tree, single layer perceptron and 4 layer neural network. The most successful method is regression tree with %73 average accuracy. The biggest challenge in this project was preprocessing the time series data into dataset and time instances with features we can use in our methods.

6. APPENDIX

```
% Main

clc;
clear all;
close all;

% Piece
money = 0;
k = 0;
k1 = 0;
duration = 0;
mean = 0;
mean1 = 0;
[big_linearray, big_wave_array] =  piecewise_linear_regression();
time = 24*60;
indexes = zeros(1,time);
wins =zeros(1,length(indexes));
offset = 5000;
for i=1 : time
indexes(i) =100*i;
end

indexes = indexes + offset;
% Apply Reg Tree
for i=1:length(indexes)
[earn_time, beginning,win_percentage, endd, guess, earning,not_lose_percentage,durr] =
the_last_trial(indexes(i), big_linearray,big_wave_array);
money = money + earning;
wins(i) = win_percentage;
notlose(i) =not_lose_percentage;
duration = duration + durr;
end
% Test
for i = 1:length(wins)
   if wins(i) ~= 0
      mean = mean + wins(i) ;
      k = k + 1;
   end
end

mean = mean /k;

for i = 1:length(notlose)
   if notlose(i) ~= 0
      mean1 = mean1 + notlose(i) ;
      k1 = k1 + 1;
   end
end
```

```
mean1 = mean1 /k1;




% win_mean =mean(wins);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [earn_time, beginning,win_percentage, endd, guess, earning,not_lose_percentage,durr] =
the_last_trial(indice, big_linearray, big_wave_array)


% indice = 17900;
num = 0;
goback = 5;

%Find the index of the linear lines starting just before the desired time
%point
[c, index] = min(abs(big_linearray(:,2)-indice));
if c <big_linearray(index,2)
   num = index+1;
end

knn_array = big_linearray(num-goback+1:num,:); %save the last "goback" number of lines in an array
% searchdegree = zeros(length(knn_array),2); % create an array to put degree intervals of the values
searchmagnitude = zeros(length(knn_array),2); % create an array to put degree intervals of the values

% find the interval of search for every line.
for i=1:length(knn_array)
   if knn_array(i,3)>0
searchmagnitude(i,1) = knn_array(i,3)*3/5;
searchmagnitude(i,2) = knn_array(i,3)*6/5;
   elseif knn_array(i,3)<0
searchmagnitude(i,2) = knn_array(i,3)*3/5;
searchmagnitude(i,1) = knn_array(i,3)*6/5;
   end
end

knn_array =[knn_array searchmagnitude];% add the interval of search to the next column of every
line
new_knn_array= zeros(length(big_linearray),goback);
indice_array = zeros(length(big_linearray),1);

% find the lines with the magnitude between the given values closest to the
% target

for z=1:length(big_linearray)

   if  big_linearray(z ,3) <= knn_array(goback,7) &&  big_linearray(z,3) >=  knn_array(goback,6)
      new_knn_array(z,1) = 1;
```

```
    end


end


for z=2:length(big_linearray)+1
%    if new_knn_array(z-1,1) == 1
    if big_linearray(z-1,3)<= knn_array(goback-1,7) &&  big_linearray(z-1,3) >=
knn_array(goback-1,6)
        new_knn_array(z,2) = 1;

     end
%    end
end


for z=3:length(big_linearray)+2
%    if new_knn_array(z-1,1) == 1
    if big_linearray(z-2,3)<= knn_array(goback-2,7) &&  big_linearray(z-2,3) >=
knn_array(goback-2,6)
        new_knn_array(z,3) = 1;

     end
%    end
end


for z=4:length(big_linearray)+3
%    if new_knn_array(z-1,1) == 1
    if big_linearray(z-3,3)<= knn_array(goback-3,7) &&  big_linearray(z-3,3) >=
knn_array(goback-3,6)
        new_knn_array(z,4) = 1;

     end
%    end
end

for z=5:length(big_linearray)+4
%    if new_knn_array(z-1,1) == 1
    if big_linearray(z-4,3)<= knn_array(goback-4,7) &&  big_linearray(z-4,3) >=
knn_array(goback-4,6)
        new_knn_array(z,5) = 1;

     end
%    end
end
```

```
%%% knn classfier

%% eulidian distance
count = 0;
newcount = 1;


for i =1:length(new_knn_array)
        if new_knn_array(i,1) == new_knn_array(i,2) && new_knn_array(i,1) ==1
        count =2;
        if new_knn_array(i,2) == new_knn_array(i,3)&& new_knn_array(i,2) ==1
        count =3;
        if new_knn_array(i,3) == new_knn_array(i,4)&& new_knn_array(i,3) ==1
        count =4;
        if new_knn_array(i,4) == new_knn_array(i,5)&& new_knn_array(i,4) ==1
        count =5;
        end
        end
        end

        end
    countarray(i) = count;
    count = 0;
end

for i=1:length(big_linearray)
 if countarray(i) >3


    categorical(i,1) = big_linearray(i-1,3);%magnitude
    categorical(i,2) = big_linearray(i-1,2)-big_linearray(i-1,1) ;%duration
    categorical(i,3) = big_linearray(i-1,5);

    categorical(i,4) = big_linearray(i-2,3);%magnitude
    categorical(i,5) = big_linearray(i-2,2)-big_linearray(i-2,1) ;%duration
    categorical(i,6) = big_linearray(i-2,5);

    categorical(i,7) = big_linearray(i-3,3);%magnitude
    categorical(i,8) = big_linearray(i-3,2)-big_linearray(i-3,1) ;%duration
    categorical(i,9) = big_linearray(i-3,5);

    categorical(i,10) = big_linearray(i,3);%magnitude
    categorical(i,11) = big_linearray(i,2)-big_linearray(i,1) ;%duration
    categorical(i,12) = big_linearray(i,5);


 end
end
% categorical= normc(categorical);

categorical = categorical(any(categorical,2),:);
magnitude_categorical = categorical(:,11);

categorical1 = categorical(:,1:9);

recategorical = [categorical1 magnitude_categorical];
tree = fitrtree(categorical1,magnitude_categorical);


my_values_array5 = [knn_array(5,3) knn_array(5,2) - knn_array(5,1) knn_array(5,5) ];

my_values_array4 = [knn_array(4,3) knn_array(4,2) - knn_array(4,1) knn_array(4,5) ];
my_values_array3 = [knn_array(3,3) knn_array(3,2) - knn_array(3,1) knn_array(3,5) ];
my_values_array2 = [knn_array(2,3) knn_array(2,2) - knn_array(2,1) knn_array(2,5) ];
my_values_array1 = [knn_array(1,3) knn_array(1,2) - knn_array(1,1) knn_array(1,5) ];

my_values= [my_values_array4 my_values_array3 my_values_array2 my_values_array1 ];
% my_values.Properties.VariableNames = {'x1' 'x2' 'x3' 'x4' 'x5' 'x6' 'x7' 'x8' 'x9'};
node_error = tree.NodeError;
node_mean = tree.NodeMean;
node_prob = tree.NodeProbability;
parenthood = tree.Parent;
cut_predictor = tree.CutPredictor;
cut_point = tree.CutPoint;
num_of_nodes = tree.NumNodes;
tree_children = tree.Children;
new_cut_predictor = zeros(length(cut_predictor),1);
for i = 1:length(cut_predictor)
    if strcmpi(cut_predictor{i,1},'x1')
        new_cut_predictor(i,1) = 1;
    end
    if strcmpi(cut_predictor{i,1},'x2')
        new_cut_predictor(i,1) = 2;
    end
    if strcmpi(cut_predictor{i,1},'x3')
        new_cut_predictor(i,1) = 3;
    end
    if strcmpi(cut_predictor{i,1},'x4')
        new_cut_predictor(i,1) = 4;
    end
    if strcmpi(cut_predictor{i,1},'x5')
        new_cut_predictor(i,1) = 5;
    end
    if strcmpi(cut_predictor{i,1},'x6')
        new_cut_predictor(i,1) = 6;
    end
    if strcmpi(cut_predictor{i,1},'x7')
        new_cut_predictor(i,1) = 7;
    end
    if strcmpi(cut_predictor{i,1},'x8')
        new_cut_predictor(i,1) = 8;
    end
    if strcmpi(cut_predictor{i,1},'x9')
        new_cut_predictor(i,1) = 9;
    end


end
next_node = 1;
stop = 0;
node_info = {};
add_matrix = [next_node node_error(next_node) node_mean(next_node) node_prob(next_node)];
node_info =[node_info add_matrix] ;

nan_values = isnan(cut_point);
while stop == 0

            if  nan_values(next_node) == 1
                stop = 1;
            end

             if(new_cut_predictor(next_node) ~= 0)
            if  my_values(1,new_cut_predictor(next_node)) < cut_point(next_node)
                    next_node = tree_children(next_node,1);
                    add_matrix = [next_node node_error(next_node) node_mean(next_node)
node_prob(next_node)];
                    node_info =[node_info add_matrix] ;


            elseif my_values(1,new_cut_predictor(next_node)) > cut_point(next_node)
                    next_node = tree_children(next_node,2);
                    add_matrix = [next_node node_error(next_node) node_mean(next_node)
node_prob(next_node)];
                    node_info =[node_info add_matrix] ;




            end

             end
```

```
end

for n=1:length(node_info)-1
efficiency1 = node_info(1,n);
efficiency1 = cell2mat(efficiency1);
efficiency2 = node_info(1,n+1);
efficiency2 = cell2mat(efficiency2);

eff(n,1) = efficiency2(1,2) - efficiency1(1,2);
eff(n,2) = n;
end


% last_count = 0;
% if length(eff) > 1
%     for i =1: length(eff)-1
%         if eff(i,1) <= eff(i+1,1)
%             last_count = last_count + 1;
%     end
%
% end
% end
win_percentage = 0;
not_lose_percentage =0;
earning = 0;
estimation_time = node_info(1,end);
estimation_time = cell2mat(estimation_time);
guess = estimation_time(1,3);
durr = 0;
durr = knn_array(5,2)-knn_array(5,1);
my_break = knn_array(5,1) + guess;
if(knn_array(5,3) > 0)
if my_break <= knn_array(5,2);
   win_percentage =  guess/ (knn_array(5,2) -  knn_array(5,1));
elseif my_break > knn_array(5,2)
   if my_break - knn_array(5,2) / (knn_array(5,2) -  knn_array(5,1)) < 0.66
   win_percentage = ((my_break - knn_array(5,2))/ (knn_array(5,2) -  knn_array(5,1)));
   else
   win_percentage = 0;
   end
end
end

if(knn_array(5,3) < 0)
if my_break <= knn_array(5,2)
   not_lose_percentage =  guess/ (knn_array(5,2) -  knn_array(5,1))
elseif my_break > knn_array(5,2)
   not_lose_percentage =  0

end
```

```
end




earn_time = guess;
beginning = knn_array(5,1)
endd = knn_array(5,2)

 if knn_array(5,3) > 0
    earning = abs(knn_array(5,4))*(my_break - indice)

end
 end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [big_linearray, big_wave_array] =  piecewise_linear_regression()
    btcdata = readtable('a.xlsx');
    price = cellfun(@str2double,btcdata{:,1});
    price_withTrend = price(1:end);
    t = 1:length(price_withTrend);

    figure
    plot(t,price_withTrend)
    title('Signal with a Trend')
    xlabel('Time Samples');
    ylabel('Price')
    legend('Trended Signal')
    grid on

    % Finding QRS-Complex
    [~,locs_Rwave] = findpeaks(price_withTrend,'MinPeakProminence',3);
    [~,locs_Swave] = findpeaks(-price_withTrend,'MinPeakProminence',2);
    [~,locs_Qwave] = findpeaks(price_withTrend,'MinPeakProminence',2);
    [~,locs_lilQwave] = findpeaks(price_withTrend,'MinPeakProminence',1);

    figure
    hold on
    plot(t,price_withTrend)
    plot(locs_Rwave,price_withTrend(locs_Rwave),'rv','MarkerFaceColor','r')
    plot(locs_Swave,price_withTrend(locs_Swave),'rs','MarkerFaceColor','b')
    plot(locs_Qwave,price_withTrend(locs_Qwave),'rs','MarkerFaceColor','g')
    %  plot(locs_lilQwave,price_withTrend(locs_lilQwave),'rv','MarkerFaceColor','b')


    grid on
    legend('trended Signal','R-waves','S-waves','Q-waves','lilQwave')
    xlabel('Time Samples')
    ylabel('Price')
    title('Q-waves, R-wave and S-wave in Noisy Signal')
```

```
   Anew=sortrows(locs_Qwave,1);
   Bnew=sortrows(locs_Swave,1);
   Cnew=sortrows(locs_Rwave,1);

   big_wave_array = [Anew;Bnew(:,1);Cnew(:,1)];
   big_wave_array = sortrows(big_wave_array);

   big_wave_array = unique(big_wave_array,'rows'); % RETURN
%%%%%%%%%%%%%%%%%%%%%%%%%

   count = 0;
   deletion = zeros(length(big_wave_array),1);

   % find the duplicate values for y axis
   for i=1:length(big_wave_array)-1
       if price_withTrend(big_wave_array(i)) == price_withTrend(big_wave_array(i+1))
         deletion(i+1) = 1;
       end
   end

   % remove the duplicate values for y axis from index array

   big_wave_array(any(deletion==1,2),:)=[];

   big_linearray(1,1) = 1;

   for i = 1:length(big_wave_array)-1
      big_linearray(i+1,1) = big_wave_array(i,1);
   end

   for i = 1:length(big_wave_array)
      big_linearray(i,2) = big_wave_array(i,1);
   end

   big_linearray(1,3) = price_withTrend(big_linearray(1,2));

   for i = 1:length(big_wave_array)
      big_linearray(i,3) = price_withTrend(big_linearray(i,2)) - price_withTrend(big_linearray(i,1));
   end

   for i = 1:length(big_wave_array)
      big_linearray(i,4) = big_linearray(i,3)/(big_linearray(i,2)- big_linearray(i,1));
   end

   %add the number of smaller Q points to the matrix as a variance
   for k=1:length(big_linearray)
      for i =1:length(locs_lilQwave)
        if big_linearray(k,1) < locs_lilQwave(i) && big_linearray(k,2) > locs_lilQwave(i)
            count = count +1;
        end
        end
        big_linearray(k,5) = count; % RETURN %%%%%%%%%%%%%%%%%%%%%%%%%
        count = 0;
    end
end % END FUNC



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [rss, error, split ,newcat1 ,newcat2] =  regtree(recategorical)
% [r t] = size(recategorical);
%
% for i  = 1:t-1
%  srtc = sortrows(recategorical,i);
%     for k = 1:length(recategorical)
%         ols1 = sum(srtc(1:k,10))/k;
%         ols2 = sum(srtc(k:length(recategorical),10))/(length(recategorical) - k + 1);
%         for m=1:k
%                 euclidian1 = euclidian1 + (srtc(m,10) - (ols1')).^2;
%
%         end
%
%         for n=k+1:length(recategorical)
%                 euclidian2 = euclidian2 +(srtc(n,10) - (ols2')).^2;
%
%         end
%
% %
%   rss1 = sqrt(sum(euclidian1,2));
% %     rss2 = sqrt(sum(euclidian2,2));
%
%     rss(k,i) =rss1 + rss2;
%     end
%     srtc = recategorical;
%     euclidian1= 0;
%     euclidian2 = 0;
% end
% rss(find(rss==0)) = NaN;
%
% % rss = rss([1:4]==1000000,:);
% [minRSS1,minRSS2] = min(rss);
% [minminRSS1, indRSS1] = min(minRSS1);
% split_column = indRSS1(1,1);
% split_row = minRSS2(1,indRSS1(1,1));
%
%
% cat = sortrows(recategorical,split_column);
% newcat1 = cat(1:split_row,:);
% newcat2 = cat(split_row+1:end,:);
% split (1,1)= split_column;
% split (2,1)= split_row;
% error = minminRSS1;
% end

function [rss, err, split_row, split_column ,newcat1 ,newcat2, bound_value, cat_size_1, cat_size_2] =
regtree(recategorical)
[r t] = size(recategorical);

for i  = 1:t-1
 srtc = sortrows(recategorical,i);
    for k = 1:length(recategorical)
        ols1(k,i) = sum(srtc(1:k,10))/k;
    end
end

for i  = 1:t-1
 srtc = sortrows(recategorical,i);
    for k = 1:length(recategorical)
          ols2(k,i) = sum(srtc(k:length(srtc),10))/(length(recategorical) - k + 1);
    end
end

for i  = 1:t-1
 srtc = sortrows(recategorical,i);
for n = 1:length(srtc)
    srtc1 = srtc(1:n,10);
    srtc2 = srtc(n:length(srtc),10);
    srtc1 = srtc1 - ols1(n,i);
    srtc1 = srtc1.^2;
    srtc2=  srtc2 - ols2(n,i);
    srtc2 = srtc2.^2;
    error1(n,i) = (sum(srtc1)).^2;
    error2(n,i) = (sum(srtc2)).^2;
end
end

error = error1 + error2;
rss = error;
[minRSS1,minRSS2] = min(rss);
[minminRSS1, indRSS1] = min(minRSS1);
split_column = indRSS1(1,1);
split_row = minRSS2(1,indRSS1(1,1));


cat = sortrows(recategorical,split_column);
newcat1 = cat(1:split_row,:);
newcat2 = cat(split_row+1:end,:);
cat_size_1 = length(newcat1);
cat_size_2 = length(newcat2);

bound_value = cat(split_row,split_column);
```

```
err = rss(split_row,split_column);
end
```

NEURAL NETWORKS CODES

MAIN

```
%  Main

clc;
clear all;
close all;
%clear all;
btcdata = readtable('a.xlsx');
price = cellfun(@str2double,btcdata{:,1});
price = price(1:60:end);
N = length(price);
x = 1:N;

% Normalization of Price Data;
for i = 1:length(price)
   price(i)= (price(i)-min(price))./(max(price) - min(price));
end


% Framing
frame_size = 30;
[frames labels] = framing(price,frame_size);

% Seperate Train and Test Data
train_percent   = 80;
traindata      = frames(:,1:ceil(length(frames)*train_percent/100)-1);
trainlabels    = labels(1:ceil(length(frames)*train_percent/100)-1);
testdata       = frames(:,ceil(length(frames)*train_percent/100)-1:end);
testlabels     = labels(ceil(length(frames)*train_percent/100)-1:end);

% Single Layer Perceptron
perceptron_lr = 0.001;
perceptron_acc = perceptron(traindata, trainlabels, testdata, testlabels,perceptron_lr,frame_size);

% MLP
MLP_lr = 0.001;
MLP_acc = MLP(traindata,trainlabels,testdata,testlabels,MLP_lr,frame_size);
```

FRAMING

```
function [frames labels ] = framing(price,frame_size)

    % Get Frames
    no_of_frame = ceil(length(price) / frame_size)-1;
    frames = zeros (frame_size,length(price) - frame_size);
    for i = 1:length(price) - frame_size
        frames(:,i) = price(i:i+frame_size-1);
    end
    % frames = zeros (frame_size,no_of_frame);
    % for i = 1:no_of_frame-1
    %     frames(:,i) = price_daily(i*frame_size:(i+1)*frame_size-1);
    % end


    % Get Labels
    labels = zeros(1,length(frames)-1);
    for i = 1:length(frames)-1
        if (frames(frame_size,i) < frames(frame_size,i+1))
            labels(i) = 1;
        else
            labels(i) = -1;
        end
    end
    % for i = 1:length(frames)-1
    %     if (frames(frame_size,i) < frames(1,i+1))
    %         labels(i) = 1;
    %     else
    %         labels(i) = -1;
    %     end
    % end
end
```

PERCEPTRON

```
function acc = perceptron(traindata,trainlabels,testdata,testlabels,learning_rate,frame_size)
    tic;
    % Weight Init.
    w = zeros(1,frame_size+1);
     for idx=1:frame_size+1
         w(idx) = normrnd(0,0.01);
     end

    % Initials
    error_sum = 0;
    MSE = zeros(1,length(traindata));
    input = zeros(1,frame_size+1);

    % TRAIN
    for i=1:length(traindata)*10
        idx = randi(length(traindata),1);
        input(2:end) = traindata(:,idx)';
        input(1)    = 1; % Add Bias
        % Forward Prop
        net = sum(w.*input);
        out = tanh(net);

        % Calculate Error and Update Weights
        error = trainlabels(idx) - out;
        error_sum = error_sum +error^2;
        MSE(idx) = error_sum/idx;
        w = w + learning_rate*error*input* (1-out^2);
    end
    figure
    plot(MSE);
    title('MSE of Training');

    % TEST
    error_new = zeros(1,length(testdata));
    for idx = 1:length(testdata)-1
        input(2:end) = testdata(:,idx)';
        input(1)    = 1; % Add Bias
        % Forward Prop
        net = sum(w.*input);
        out = tanh(net);
        % Check Success Rate
        error_new(idx) = testlabels(idx) - sign(out);
    end
    acc = ((sum(error_new(:)==0))/(length(error_new))*100);
    elapsedTime = toc;
    disp(['Perceptron Success Rate: %' num2str(acc)]);
    disp(['Perceptron Training/Testing Time: ' num2str(elapsedTime)]);
end
```

MLP

```
function acc = MLP(traindata,trainlabels,testdata,testlabels,learning_rate,frame_size)
    tic;

    % Variables and Constants
    reg_param =[1,0.001,0.005,0.01,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.5,0.75,0];
    N = length(traindata)-1;

    % Network Specs.
    n1 = frame_size+1;
    n2 = frame_size+1;
    n3 = frame_size+1;
    nout = 1;
    input = zeros(1,frame_size+1);
    lr = learning_rate;
    total_epoch = 10;

    % Weight and Bias Initilization
```

```
shift = 0;
shift2 = 0;
w_1 = -shift + normrnd(0,0.001,length(input),n1);
w_2 = -shift + normrnd(0,0.001,n1,n2);
w_3 = -shift + normrnd(0,0.001,n2,n3);
w_4 = -shift2 + normrnd(0,0.001,n3,nout);
MSE = zeros(1,total_epoch);
for t = 1:15
    lambda = reg_param(t);
   for i=1:total_epoch
      error_acc = 0;
      cum_error = zeros(1,N);
      for k= 1:N
         idx = randi(length(traindata),1);
         %Forward Propagation
         input(2:end) = traindata(:,idx)';
         input(1)    = 1; % Add Bias
         x1  = tanh((input*w_1));
         x2  = tanh((x1*w_2));
         x3  = tanh((x2*w_3));
         o   = tanh((x3*w_4));

         % Back Propagation
         error = trainlabels(idx)-o;
         error_acc = error_acc +(error.^2);
         cum_error(k) = error_acc/k;

         % Finding Local Gradients
         f0 = (1-o.^2);
         Local_gradient_1 = f0.*error;
         f3 = (1-x3.^2);
         Local_gradient_2 = f3.*(w_4*Local_gradient_1);
         f2 = (1-x2.^2);
         Local_gradient_3 = f2.*(w_3*Local_gradient_2);
         f1 = (1-x1.^2);
         Local_gradient_4 = f1.*(w_2*Local_gradient_3);

         % Updating Weights
         w_4 = w_4 + lr*(x3'*Local_gradient_1' - lambda*w_4);
         w_3 = w_3 + lr*(x2*Local_gradient_2' - lambda*w_3);
         w_2 = w_2 + lr*(x1*Local_gradient_3' - lambda*w_2);
         w_1 = w_1 + lr*(input*Local_gradient_4' - lambda*w_1);
      end

      MSE(i) = mean(cum_error); % Last Value of MSE of Each Epoch
   end
   MSE_Regularized(i) = mean(MSE);
end
figure;
plot(MSE_Regularized);
title('MSE Through Different Regularization Parameters')
```

```
    % TEST
    error_new = zeros(1,length(testdata));
    for idx = 1:length(testdata)-1
        input(2:end) = testdata(:,idx)';
        input(1)     = 1; % Add Bias
        % Forward Prop
        x1  = tanh((input*w_1));
        x2  = tanh((x1*w_2));
        x3  = tanh((x2*w_3));
        o   = tanh((x3*w_4));
        % Check Success Rate
        error_new(idx) = testlabels(idx) - sign(o);
    end
    acc = ((sum(error_new(:)==1))/(length(error_new))*100);
    elapsedTime = toc;
    disp(['MLP Success Rate: %' num2str(acc)]);
    disp(['MLP Training/Testing Time: ' num2str(elapsedTime)]);
end
```