We proceed by saving all the data for each of the 60 Excel files into Python.

Processing Data.py

Each File corresponds to a day of transactions for either Spot (BTC/USD) or the corresponding Futures contract.

There are 31 files for spot and 29 for futures

For each day( either for spot or futures) we do the following:

* For each tick we find:
  + Time difference from midnight(00:00:00) of that particular date
  + Time difference between the current tick and the previous tick
  + Extract the bid price
  + The bid-ask spread
  + The price difference between current and previous tick
  + The bid ask spread difference between current and previous tick
  + The total volume traded (bid price\*bid quantity+ ask\_price\*ask\_quantity)
  + The IBS\_i(defined in Tsay’s book)
  + The A\_i (price change?1:0)
  + The D\_i(DPt>0?1:-1)
  + The Si (abs(Price change)/0.5) the absolute value of Dpt in ticks
  + The mean of bid and ask price
  + The difference between current tick’s volume and last’s one
  + Which one minute time interval belongs the tick(based on first calculation)
  + Which five minute time interval belongs the tick(based on first calculation)
  + Bid volume
  + Ask volume
  + Bid price
  + Ask price
  + Adjusted price( bid\_price\*bid\_qty+ask\_price\*ask\_qty)/(bid\_qty+ask\_qty)
  + Delta Bid Volume (current ticks Bid Volume – previous tick’s Bid Volume)
  + Delta Ask Volume (current ticks Ask Volume – previous tick’s Ask Volume)
  + Delta Bid Price (current ticks Bid Price – previous tick’s Bid Price)
  + Delta Ask Price (current ticks Ask Price – previous tick’s Ask Price)
  + Delta Adjusted Price
  + Bid Quantity
  + Ask Quantity
* for each one minute interval and for each five minute interval we find(daily):
  + number of price listings in one minute interval
  + first in the list price of the iii-th minute
  + last in the list price of the iii-th minute
  + Prices of one minute interval
  + list of prices of one minute interval
  + number of bid transactions in one minute interval
  + number of ask transactions in one minute interval
  + Total number of transactions in one minute interval
  + Price change of the minute
  + Return of the minute
  + Ai of the minute
  + Di of the minute
  + Si of the minute
  + Ni of the minute (# Transactions no price change between ticks)
  + Volume of the minute

Working with Data:

For the tick data, I saved the IBS, The Dt between two consecutive ticks, the bid-ask spread, the Di, the Volume of the tick, the ask price, the Ai and the Si in ticks(The Dprice in ticks). I cleaned this data by excluding the outliers. As outliers I defined the data where the absolute value of the price change with respect to the previous tick exceeded 64 tick sizes(In Bitmex the tick size is 0.5 USD).

Creating Features:

Based on this cleaned data we created the necessary features for predicting the model on tick size:

IBS, dt, Bid-Ask, Di, Volume, Ask ,Ai , Dpt

Statistics that we got:

We got the mean number of transactions per minute.

We found the autocorrelation function between the number of transaction per minute and their lags.

Passing the features:

We have almost ticks. For every 9,000 ticks we fit the model for predicting the price changes from tick to tick. We do the same for the one minute and five minute intervals, that is by having the features of the minute or of the five minute, we predict the price changes from minute to minute or from a five minute interval to a five minute interval.

Models for price changes:

We use three models for price changes: The Ordered Probit model, the Decomposition model and the Price Change and Duration Model:

**Ordered Probit Model:**

The equation is:

where corresponds to the explanatory features and is the parameters vector. The features are Dt, the first three lags of the price changes in ticks,the first three lags of IBS and the first three lags of the product

Fitting the model:

We have 129 categories ranging from -64 to 64 the change of ticks size. This can be considered as a categorical variable. The  values range from -64.5 to 64.5 with step size one. This means that in the process of fitting if the Y measured, the tick size changed value, is between -64.5 and -63.5 we are going to set and =-63.5

The likelihood product is

We use the Nelder-Mead algorithm to fit for the parameters. We apply the minimization as many times as needed so that the ratio

We left 20% of the training data out and we used as a test for the model that we developed:

**Decomposition Model:**

The equation is:

We want to fit p\_i, the probability that there will be a price change, δ\_i, the probability there will be an upwards price change provided there will be a price change as well as the tick size of the price change.

Thus the conditional probability is:

Now the number of ticks by which the price will increase or decrease follows a geometric distribution. We require the number of ticks to be at list one:

So follows a geometric distribution, that

P(Y=k)=

We will have that

The log likelihood operand of the log-likelihood sum is:

We are also planning to work with the Price Change and Duration Model

**Price Change and Duration model:**

*The number of ticks follows a Poisson distribution*

For both up and down movements. We fit each one of those equations individually.

**Trading Strategies:**

We developed signals based on the predictions on the ΔPrice derived from the Logistic Regression, the Ordered Probit Model and the Decomposition Model.

Our signals have constant Dt even though the data does not and Dt=1,5 minutes.

For each time interval we find the Dp(Last price minus first price),Ni, Si,Di, as well as the number of bid transactions ,the number of ask transactions the total number of transactions the total volume of the time interval, the mean Bid-Ask of the interval as well as the IBS of the interval. We then fit the model in the 80% of the data and we make our predictions in the out of sample 20% of the data.

**Trading Strategies based on the Decomposition Model:**

Firstly we fit the Decomposition Model either for the one minute space intervals(approximately 40,000 data points) or for the 5-minute space interval( approximately 8,000 data points). We pick up the first 80% of the data, we fit the model and then we apply the strategy on the remaining 20% of the data.

Decomposition model strategy based on 1,5 minutes interval.

The training dataset is composed of 6500 data points for the five minutes case whereas the test dataset is composed of 1625 elements. We use as features the first three lags of Ai, Di and Si.

In particular, we fit a Logistic regression to predict whether Ai=0,1(whether there is a price change or not), then on the data where there is price change in the observed value we fit another Logistic Regression to see if we have an upward or downward movement and then: For those that in observed data we have upward movement we use a maximum likelihood of a geometric distribution where the λ is a function of Si, to predict the Si-1. We consider that Si-1 follows a Geometric distribution. After fitting the equations we make the predictions. For the Si, the predicted value is

Important portions of the Python code:

**Fitting:**

*theta=[0.2,-0.3,-0.3,-0.3]*

*t0 = time()*

*resultsPos = op.minimize(self.likelihoodSumGeometricPos, theta, method='nelder-mead')*

*print "done in %0.3fs" % (time() - t0)*

*t0 = time()*

*resultsNeg = op.minimize(self.likelihoodSumGeometricNeg, theta, method='nelder-mead')*

*print "done in %0.3fs" % (time() - t0)*

*self.thetaPos=resultsPos.x*

*self.thetaNeg=resultsNeg.x*

*def likelihoodSumGeometricPos(self,theta):*

*n=self.YThirdRegression.shape[0]*

*summ=0*

*for lll in range(0,n):*

*# print(((0.01\*self.XThirdRegression[lll,:])) )*

*Thelambda=(1.0/(1.0+ np.exp(-np.dot(theta,self.XThirdRegression[lll,:])) ))*

*if(1-Thelambda>0):*

*summ=summ+(self.YThirdRegression[lll]-1)\*np.log(1-Thelambda)*

*return -summ*

*def likelihoodSumGeometricNeg(self,theta):*

*n=self.YFourthRegression.shape[0]*

*summ=0*

*for lll in range(0,n):*

*Thelambda=(1.0/(1.0+ np.exp(-np.dot(theta,self.XFourthRegression[lll,:])) ))*

*if(1-Thelambda>0):*

*summ=summ+(self.YFourthRegression[lll]-1)\*np.log(1-Thelambda)*

*return -summ*

**Predicting:**

*Predictions=np.zeros((int(self.YFirstRegression\_test.shape[0])))*

*for lll in range(0,int(self.YFirstRegression\_test.shape[0]) ):*

*PredictedA=int(self.Logistic1.Logistic.predict(self.TheTest[lll:lll+1,(52,25,26,27)] ) )*

*if(PredictedA>0):*

*PredictedD=int(self.Logistic2.Logistic.predict(self.TheTest[lll:lll+1,(52,29,30,31)] ) )*

*if(PredictedD>0):*

*Thelambda\_u=(1.0/(1.0+ np.exp(-np.dot(self.thetaPos,0.01\*self.TheTest[lll,(52,33,34,35)]))))*

*#print(self.thetaNeg)*

*Predictions[lll]=np.log((1.0-Thelambda\_u)/Thelambda\_u)*

*else:*

*Thelambda\_d=(1.0/(1.0+ np.exp(-np.dot(self.thetaNeg,0.01\*self.TheTest[lll,(52,33,34,35)]))))*

*#print(self.thetaNeg)*

*Predictions[lll]=-np.log((1.0-Thelambda\_d)/Thelambda\_d)*

*else:*

*Predictions[lll]=0*

*self.Predictions=Predictions*

Based on this model we predict the amount the price of Bitcoin changes in ticks.

The prediction size is 1625:

The accuracy score per se is not that promising(0.2%) yet the order of magnitude of the predictions is promising. Furthermore if we take into account only the directions and not the size, the accuracy score increases to 52%.

For the one minute case the accuracy score is 0.8% but if we check for the directions that increases to 50.3% So even in the one minute interval space the directions are more than half accurate.

The training dataset is composed of 32829 elements for the 1 minute case whereas the test dataset is composed of 8208 elements.

Results’ Table: Decomposition model results applied on Five Minute intervals:

Equations:

,logit of probability of price change

,logit of probability of upward price

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| -1.82625913 | 3.53049234 | 2.22069971 | 2.90388839 |
|  |  |  |  |
| 0.031248 | -0.00287477 | -0.12643271 | -0.05386956 |
|  |  |  |  |
| -43.91736214 | -30.24406213 | -2.49914971 | -31.28090851 |
|  |  |  |  |
| -21.9 | -16.4778 | -0.006186 | -15.90 |

Results’ Table: Decomposition model results applied on Five Minute intervals:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| -1.04354611 | 2.28621537, | 1.47955969 | 1.65365277 |
|  |  |  |  |
| 0.03129157 | 0.12526689 | 0.06449228 | 0.01410344 |
|  |  |  |  |
| -43.91736214 | -2.49914971 | -30.24406213 | -31.28090851 |
|  |  |  |  |
| -43.91736214 | -30.24406213 | -2.49914971 | -31.28090851 |

|  |  |
| --- | --- |
| Accuracy 5-minute | 0.52369230769230768 |
| Accuracy 1-minute | 0.50341130604288498 |

Comparison of first 100 realizations(out of 1625) versus predictions in number of ticks for the 5-minutes case:

Realizations:

133., 117., 235., 95., -104., 152., -48., 89., 2.,

74., 78., 155., -23., -144., 398., -655., -23., 69.,

-173., -340., 87., -162., -59., -582., 320., 24., 143.,

-119., 70., 272., 27., -140., 144., 136., -73., 59.,

167., -31., -316., 310., -192., -543., 127., -44., -299.,

-23., 307., -93., 333., 206., -18., -141., 30., -37.,

142., -281., 8., 32., -150., -58., 101., 149., -78.,

20., 85., 20., -140., -92., -5., 16., 192., -13.,

45., -107., -34., 398., 169., -53., -84., 186., 84.,

50., -172., -219., 309., 116., 120., 79., 114., 8.,

-231., -24., -106., 16., 110., -35., 37., 168., -94., -27.

The predictions are given by in case of upward move and

in case of a downward move.

Predicions:

-39.67947238, 100.44322966, -41.6195098 , -60.1078495 ,

-34.5007644 , -54.74746228, 78.72612684, -24.68290191,

76.10296171, -8.19028232, -26.5715449 , -13.39447968,

-37.53698454, -16.42734519, 93.05083571, 131.60392544,

-131.0821451 , 148.26275433, 226.77233163, -32.38896482,

129.17634073, 89.36458839, -81.00832982, 49.5461832 ,

228.60918534, 130.22095975, -96.78255398, -74.69188686,

-23.65459504, 69.31570443, -63.97451559, -15.82099569,

-66.56152553, 55.93527798, -44.91006848, -35.16488271,

62.64958513, -39.35406259, -14.72364805, 149.02426357,

111.79016094, -82.14737402, 266.4336148 , 112.47885976,

-93.86176016, 131.69529907, 28.63136527, 187.39316523,

-19.22141434, 199.50849886, -48.97912872, -56.17398251,

107.9718197 , 18.66675688, -28.7490375 , 53.69469979,

-52.41678128, 54.30019934, -50.19606806, -26.2104554 ,

31.73934494, 79.35654596, -34.00433301, -29.14880631,

55.04587649, -26.63525624, -6.70159001, -36.81168469,

38.01870207, 48.04386636, 34.18161687, -32.65289679,

-4.91840993, 74.43323537, -19.92122543, 27.47265376,

155.13082391, -33.5001711 , -72.27964203, 80.03347053,

75.37129645, -27.43527449, -38.05344654, -41.92736313,

86.61266144, 153.16962609, -54.19263965, -69.15770949,

-31.69810668, -38.09916646, -14.11221969, -56.41932492,

15.97325703, 105.35657406, 15.43474029, -35.20892656,

-8.53846238, 46.91317836, -33.47214571, -21.60482139

The order of magnitude is the same

For the case of one minute:

Realizations:

22, -30, -3, 70, 22, -4, -11, 6, 2, 2, 2,

50, -37, -12, 57, 2, 0, -187, -95, 42, -66, 6,

21, 3, 43, 52, 67, -170, 8, 88, -77, -26, 23,

85, 82, 21, 0, 4, 9, 18, -17, 6, -29, 1,

-17, 17, 21, 23, -131, 17, 33, 51, -77, 9, -40,

39, -5, -37, -23, 58, 82, 55, 226, 12, 2, 102,

22, 31, 43, -60, -39, -4, 7, -65, -37, 7, 25,

-14, 112, 6, 1, 39, 28, 122, -12, -31, 24, -20,

-2, 47, -36, 32, 0, -35, -89, -21, 20, -47, -149,

-41

Predictions:

5, 10, 17, -8, 12, 23, 29, -10, 5, 5, 2, 1, 2,

17, -20, 17, 21, 18, -57, -33, 88, -51, 33, 23, 8, 8,

15, 31, 41, 73, 58, 31, -37, 32, 17, 35, 52, 32, 7,

1, 4, 9, 11, -14, 11, -14, 6, 11, 12, 17, 47, 47,

17, 28, 39, -36, 16, -15, -23, -9, 20, 27, 44, 48, 86,

74, 7, 32, 39, 17, 24, -27, -21, 14, -21, -15, 32, 14,

10, 15, 38, 37, 3, 13, 24, 46, -48, 14, -16, -9, 8,

-13, 26, 21, -21, -28, -19, 35, -21, -52

Ordered Probit Model( Multinomial Logistic Regression)

We use the multinomial logistic regression model.

In this case we use 11 categorical values- examples.

Pr(Category=k|Info)=,

To assign the categories we have to run through the parameters which are the thresholds-borders of each category.

We are going to choose four categories only. The boundaries are:

In particular if the price change in ticks is less than -1 tick then we assign category 1, if it is between -1 and 0 ticks we assign the category 2, if the price change is between 0 and one ticks we assign the category 3 and if the price change is greater than one tick we assign the category 4.

For the 5-minutes case:

The accuracy of this model in the out of sample space is 50.4%.

For this for categories the coefficients are:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
| Category 1 | -9e-06 | 3.51853616e-04 | 3.19393663e-04 | -8.99819119e-05 | -6.49634327e-04 | -5.18595605e-04 | -4.43622632e-04 | 1.65715226e-10 | 1.03820794e-09 | -7.87994010e-09 |
| Category 2 | 3.39948943e-05 | -6.32083567e-05 | 2.48323721e-05 | 5.42916312e-05 | 7.05528120e-04 | 5.00727995e-04 | 5.19793007e-04 | 5.29345727e-10 | -5.50792803e-10 | -7.12891406e-09 |
| Category 3 | -1.74169096e-05 | -4.79323802e-05 | -1.68723456e-05 | 1.05657059e-05 | 6.13531065e-04 | 5.40649797e-04 | 3.64711216e-04 | -5.45082895e-10 | -7.25769786e-10 | 2.23500368e-08 |
| Category 4 | -6.78041037e-06 | -2.40712879e-04 | -3.27353690e-04 | 2.51245748e-05 | -6.69424857e-04 | -5.22782187e-04 | -4.40881591e-04 | -1.49941843e-10 | 2.38397982e-10 | -7.34108303e-09 |

We present now the first 100 out of 1625 predictions:

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3

and the first 100 out of the 1625 realizations:

3., 3., 3., 3., 1., 3., 1., 3., 3., 3., 3., 3., 1.,

1., 3., 1., 1., 3., 1., 1., 3., 1., 1., 1., 3., 3.,

3., 1., 3., 3., 3., 1., 3., 3., 1., 3., 3., 1., 1.,

3., 1., 1., 3., 1., 1., 1., 3., 1., 3., 3., 1., 1.,

3., 1., 3., 1., 3., 3., 1., 1., 3., 3., 1., 3., 3.,

3., 1., 1., 1., 3., 3., 1., 3., 1., 1., 3., 3., 1.,

1., 3., 3., 3., 1., 1., 3., 3., 3., 3., 3., 3., 1.,

1., 1., 3., 3., 1., 3., 3., 1., 1.

For the one minute case:

We fit the model in 32,829 samples.

We make the predictions in 8208 samples

Again same separation of categories, the accuracy now is 48.8%

For this for categories the coefficients are:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
| Category 1 | -2.37470056e-05 | -6.27557463e-04 | -1.98539219e-04 | -9.20712850e-05 | -2.21812188e-03 | -1.46276304e-03 | -1.18562120e-03 | -5.75816489e-09 | 4.12083926e-09 | -9.13468582e-09 |
| Category 2 | 1.18515642e-04 | 9.81078084e-05 | 1.85616246e-05 | 5.45261731e-05 | 2.54723792e-03 | 2.84237467e-04 | 9.04711728e-05 | -6.06497626e-09 | 2.37864566e-09 | 8.88021977e-10 |
| Category 3 | -7.91279637e-05 | 5.85538028e-05 | 1.76238573e-04 | 2.61973115e-05 | 2.26579528e-03 | 2.38156403e-03 | 2.38006967e-03 | 1.31016250e-08 | -2.51141001e-09 | 1.47606484e-08 |
| Category 4 | -1.56406722e-05 | 4.70895852e-04 | 3.73902134e-06 | 1.13478004e-05 | -2.59491133e-03 | -1.20303846e-03 | -1.28491965e-03 | -1.27834931e-09 | -3.98797732e-09 | -6.51393936e-09 |

First 100 predicted out of 8208

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3,

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3

First 100 out of 8208 realised:

3., 1., 1., 3., 3., 1., 1., 3., 3., 3., 3., 3., 1.,

1., 3., 3., 2., 1., 1., 3., 1., 3., 3., 3., 3., 3.,

3., 1., 3., 3., 1., 1., 3., 3., 3., 3., 2., 3., 3.,

3., 1., 3., 1., 3., 1., 3., 3., 3., 1., 3., 3., 3.,

1., 3., 1., 3., 1., 1., 1., 3., 3., 3., 3., 3., 3.,

3., 3., 3., 3., 1., 1., 1., 3., 1., 1., 3., 3., 1.,

3., 3., 3., 3., 3., 3., 1., 1., 3., 1., 2., 3., 1.,

3., 2., 1., 1., 1., 3., 1., 1., 1.

Logistic Regression:

In the logistic Regression modelling we predict whether we will have positive or less or equal to zero returns.

Pr(Category=k|Info)=,

We use the same explanatory variables as before.

For the 5 minute case:

-4.73439997e-10

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
| Category 1 | -1.43577716e-15 | 2.35667477e-14 | 2.49628934e-14 | -6.49417046e-15 | 6.36233496e-14 | 6.96713347e-14 | 7.10172396e-14 | 7.46342724e-10 | 9.89816845e-10 | -4.7e-10 |

For the five minutes case the accuracy is 48%

The first 100 predicted values using Logistic regression(determine whether up or down)

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,

1, 1, 1, 1, 1, 0, 1, 1

0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1.,

1., 0., 1., 1., 0., 1., 1., 0., 1., 1., 1., 0., 0.,

0., 1., 0., 0., 0., 1., 0., 0., 1., 0., 0., 1., 1.,

0., 1., 1., 0., 1., 1., 1., 0., 1., 0., 0., 1., 1.,

0., 1., 0., 1., 0., 0., 1., 1., 0., 0., 1., 0., 0.,

0., 1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 0., 1.,

1., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 1.,

1., 1., 0., 0., 1., 0., 0., 1., 1

Those are the out of sample predictions.

For the case of one minute:

The accuracy in predictions is 47.57%

The coefficients of the Logistic Regression Model are:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |
| Category 1 | -1.12326029e-06 | -1.92657630e-05 | -4.97315373e-06 | -1.66428302e-06 | 1.25991528e-04 | 1.14149328e-04 | 1.39073745e-04 | -1.09607914e-09 | 7.68860660e-09 | 2.35710575e-10 |

The first one hundred predictions

1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,

1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,

1, 1, 1, 1, 1, 1, 1, 1

The first 100 realizations of the out of sample data:

0., 1., 1., 0., 0., 1., 1., 0., 0., 0., 0., 0., 1.,

1., 0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 0., 0.,

0., 1., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0.,

0., 1., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0., 0.,

1., 0., 1., 0., 1., 1., 1., 0., 0., 0., 0., 0., 0.,

0., 0., 0., 0., 1., 1., 1., 0., 1., 1., 0., 0., 1.,

0., 0., 0., 0., 0., 0., 1., 1., 0., 1., 1., 0., 1.,

0., 0., 1., 1., 1., 0., 1., 1., 1.

**Alphas:**

The alphas that are developed are based on the predictions derived from the Logistic Regression Model, the Ordered Probit Model and the Decomposition Model.

The procedure is as follows. We collect all the ticks and we assign them to their corresponding one or five minute interval. We find the intervals’ statistics so now we reduce our data points significantly from ticks to minute intervals or five minute intervals. We fit our model to the first 80% of the intervals and then we apply the prediction for the rest of the 20% percent of the time intervals. That is if we have a granularity in minutes we fit the model in 32,829 samples and we make the predictions, develop the alpha, on the remaining 8208 samples, the alpha size is 8208 we trade every minute.

I acknowledge that we need to develop a more dynamic model using the ticks.

That is we need to fit the model to all the ticks of the first minute and then start making the predictions of the second minute and apply our alpha. Then based on the realizations of the second minute fit again the model and predict for the third minute and vice versa. There was not time to implement this strategy so I resorted to the simpler version. I will be happy to continue with that if you want me to.

**Finding the Statistics of the Alphas:**

To evaluate the performance of our alphas we have to calculate the total PnL as well as the Mean returns and the Standard Deviation of the returns to get the Sharpe ratio.

At the x-th minute after the inception of our trading we update the PnL as:

At the x-th minute after the inception of our trading we find the portfolio returns of the x-th minute as:

We find the mean of the above quantity as well as the standard deviation thereof.

The Sharpe Ratio is the mean of the minute returns over the standard deviation of the minute returns.

**Alphas:**

**Assuming that shorting is allowed**

All the cumulative statistics are either per minute or per five minutes.

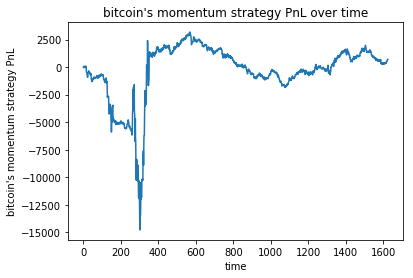
Alpha 1:

Alpha on Five minute granularity

Based on the predictions of the Decomposition model(we predict price change in ticks) we assign as

alpha= np.sign(self.predictions[x]/10.0)

where x the x-th 5 minute interval after the end of sampling period. In sampling period we do not develop the Alpha.



Very big drawdown, bad performance.

Cumulative Statistics:

Cumulative Statistics:

|  |  |
| --- | --- |
| Total PNL | 7.15500000e+02 |
| Mean Portfolio Return per 5-minute | -2.77316488e-05 |
| Mean Square Portfolio Return per 5-minute | 4.24206185e-04 |
| Standard deviation | 0.0205962476218 |
| Sharpe Ratio | -1.34644181e-03 |

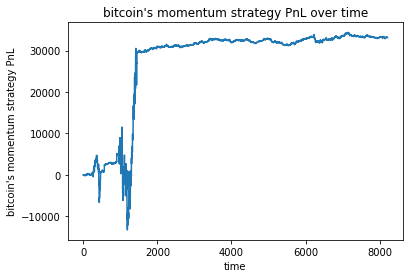
Alpha 2:

Alpha on One minute granularity

Based on the predictions of the Decomposition model(we predict price change in ticks) we assign as

alpha= np.sign(self.predictions[x]/10.0)

where x the x-th minute interval after the end of sampling period. In sampling period we do not develop the Alpha.



The performance of this alpha is shown above.

Cumulative statistics:

Cumulative Statistics:

|  |  |
| --- | --- |
| Total PNL | 3.31515000e+04 |
| Mean Portfolio Return per 5-minute | 1.93477378e-04 |
| Mean Square Portfolio Return per 5-minute | 3.85149560e-04 |
| Standard deviation | 0.0196242739071 |
| Sharpe Ratio | 9.85908466e-03 |

Alpha 3:

Alpha on Five minute granularity

Based on the predictions of the Ordered Probit Model(we predict whether the price change will be less than minus one tick ,Category 1, between minus one tick and zero ticks, Category 2, between zero ticks and 1 ticks, Category 3, and greater than one tick, Category 4, we assign the Alpha as follows.

for x in range(0, n):

if(self.predictions[x]==1):

self.alpha[x]=-1

elif(self.predictions[x]==2):

self.alpha[x]=-0.5

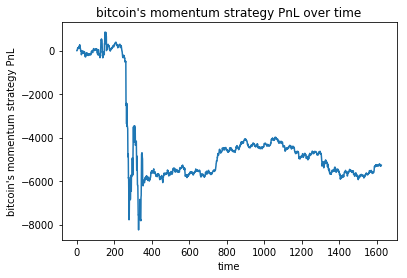
elif(self.predictions[x]==3):

self.alpha[x]=0.5

elif(self.predictions[x]==4):

self.alpha[x]=1

where x the x-th 5 minute interval after the end of sampling period. In sampling period we do not develop the Alpha.



The performance is dreadful.

Cumulative Statistics:

|  |  |
| --- | --- |
| Total PNL | -5.27025000e+03 |
| Mean Portfolio Return per 5-minute | -1.50965650e-04 |
| Mean Square Portfolio Return per 5-minute | 1.17817223e-04 |
| Standard deviation | 0.0108533143569 |
| Sharpe Ratio | -1.39096358e-02 |

Alpha 4

Alpha on One minute granularity

Based on the predictions of the Ordered Probit Model(we predict whether the price change will be less than minus one tick ,Category 1, between minus one tick and zero ticks, Category 2, between zero ticks and 1 ticks, Category 3, and greater than one tick, Category 4, we assign the Alpha as follows.

for x in range(0, n):

if(self.predictions[n]==1):

self.alpha[x]=-1

elif(self.predictions[n]==2):

self.alpha[x]=-0.5

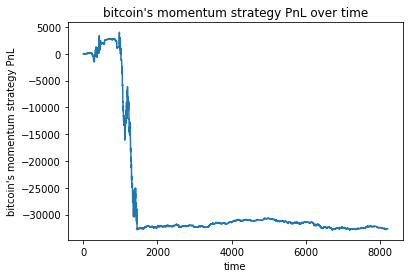
elif(self.predictions[n]==3):

self.alpha[x]=0.5

elif(self.predictions[n]==4):

self.alpha[x]=1

where x the x-th minute interval after the end of sampling period. In sampling period we do not develop the Alpha.



Very bad performance.

Cumulative Statistics:

|  |  |
| --- | --- |
| Total PNL | -3.26540000e+04 |
| Mean Portfolio Return per 5-minute | -1.90563344e-04 |
| Mean Square Portfolio Return per 5-minute | 1.01167539e-04 |
| Standard deviation | 0.0100564021765 |
| Sharpe Ratio | -1.89494554e-02 |

Alpha 5:

Alpha on Five minute granularity

Based on the predictions of the Logistic Regression Model(we predict whether the price change greater than 0 or equalt or less than zero

for x in range(0, n):

if(self.predictions[x]>0):

if(self.alpha[x-1]>0):

self.alpha[x]=self.alpha[x-1]

else:

self.alpha[x]=1

else:

if(self.predictions[x]<0):

if(self.alpha[x-1]<0):

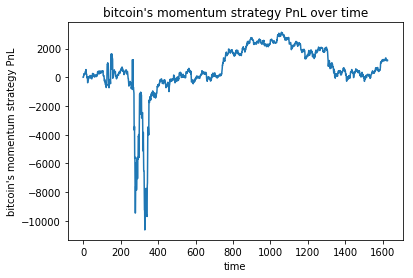
self.alpha[x]=self.alpha[x-1]

else:

self.alpha[x]=-1

where x the x-th 5 minute interval after the end of sampling period. In sampling period we do not develop the Alpha.

The performance is as follows:



The time in the abscissa is the index of the five minute intervals.

Cumulative Statistics:

|  |  |
| --- | --- |
| Total PNL | 1.17750000e+03 |
| Mean Portfolio Return per 5-minute | 2.59969873e-04 |
| Mean Square Portfolio Return per 5-minute | 4.22508252e-04 |
| Standard deviation | 0.0205533614579 |
| Sharpe Ratio | 1.26485331e-02 |

To find per day we multiply the returns by 288 and the standard deviation by sqrt(288).

Alpha 6

Alpha on One minute granularity

Based on the predictions of the Logistic Regression Model(we predict whether the price change greater than 0 or equal or less than zero

for x in range(0, n):

if(self.predictions[x]>0):

if(self.alpha[x-1]>0):

self.alpha[x]=self.alpha[x-1]

else:

self.alpha[x]=1

else:

if(self.predictions[x]<0):

if(self.alpha[x-1]<0):

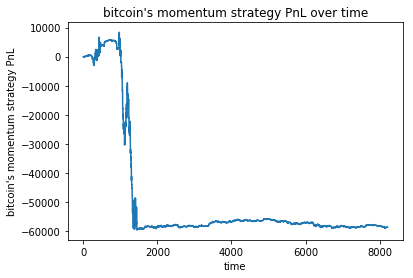
self.alpha[x]=self.alpha[x-1]

else:

self.alpha[x]=-1

self.alpha=self.alpha.astype(np.float) where x the x-th minute interval after the end of sampling period. In sampling period we do not develop the Alpha.

In this example we created the same alpha as we did in the five minutes sample and the performance is disastrous:



Cumulative Statistics:

|  |  |
| --- | --- |
| Total PNL | -5.86420000e+04 |
| Mean Portfolio Return per 5-minute | -3.19310701e-04 |
| Mean Square Portfolio Return per 5-minute | 3.83321396e-04 |
| Standard deviation | 0.019575991340 |
| Sharpe Ratio | -1.63113426e-02 |

**No shorting allowed:**

Alpha 7:

Alpha on Five minute granularity

Based on the predictions of the Decomposition model(we predict price change in ticks) we assign as

for x in range(0, n):

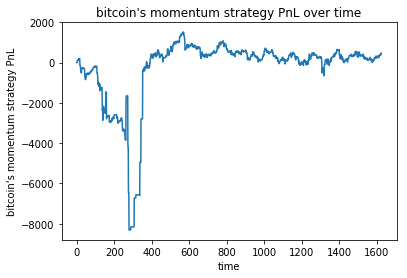
if(np.sign(self.predictions[x]/10.0)<0):

self.alpha[x]=0

else:

self.alpha[x]=1

where x the x-th 5 minute interval after the end of sampling period. In sampling period we do not develop the Alpha.



Very big drawdown, bad performance.

Cumulative Statistics:

|  |  |
| --- | --- |
| Total PNL | 4.79500000e+02 |
| Mean Portfolio Return per 5-minute | 9.85892318e-05 |
| Mean Square Portfolio Return per 5-minute | 1.42525190e-04 |
| Standard deviation | 0.0119379843435 |
| Sharpe Ratio | 8.25844874e-03 |

A little bit better than the allow shorting strategy.

Alpha 8:

Alpha on One minute granularity

Based on the predictions of the Decomposition model(we predict price change in ticks) we assign as

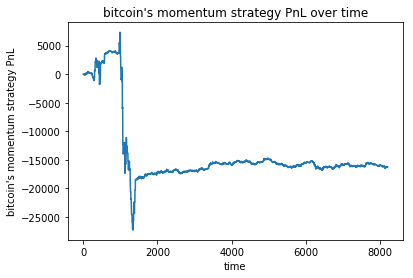
for x in range(0, n):

if(np.sign(self.predictions[x]/10.0)<0):

self.alpha[x]=0

else:

self.alpha[x]=1

where x the x-th minute interval after the end of sampling period. In sampling period we do not develop the Alpha. 

The performance of this alpha is shown above.

Cumulative statistics:

Cumulative Statistics:

|  |  |
| --- | --- |
| Total PNL | -1.62560000e+04 |
| Mean Portfolio Return per 5-minute | -8.89605547e-05 |
| Mean Square Portfolio Return per 5-minute | 1.04872740e-04 |
| Standard deviation | 0.0102403528103 |
| Sharpe Ratio | -8.68725486e-03 |

Alpha 9:

Alpha on Five minute granularity

Based on the predictions of the Ordered Probit Model(we predict whether the price change will be less than minus one tick ,Category 1, between minus one tick and zero ticks, Category 2, between zero ticks and 1 ticks, Category 3, and greater than one tick, Category 4, we assign the Alpha as follows.

for x in range(0, n):

if(self.predictions[x]==1):

self.alpha[x]=0

elif(self.predictions[x]==2):

self.alpha[x]=0

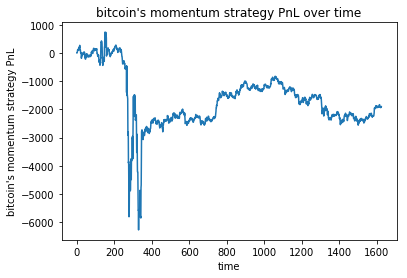
elif(self.predictions[x]==3):

self.alpha[x]=0.5

elif(self.predictions[x]==4):

self.alpha[x]=1

where x the x-th 5 minute interval after the end of sampling period. In sampling period we do not develop the Alpha.



The performance is dreadful.

Cumulative Statistics:

|  |  |
| --- | --- |
| Total PNL | -1.90225000e+03 |
| Mean Portfolio Return per 5-minute | 1.19846414e-05 |
| Mean Square Portfolio Return per 5-minute | 9.62998431e-05 |
| Standard deviation | 0.00981324102981 |
| Sharpe Ratio | 1.22127250e-03 |

Alpha 10

Alpha on One minute granularity

Based on the predictions of the Ordered Probit Model(we predict whether the price change will be less than minus one tick ,Category 1, between minus one tick and zero ticks, Category 2, between zero ticks and 1 ticks, Category 3, and greater than one tick, Category 4, we assign the Alpha as follows.

for x in range(0, n):

if(self.predictions[n]==1):

self.alpha[x]=0

elif(self.predictions[n]==2):

self.alpha[x]=0

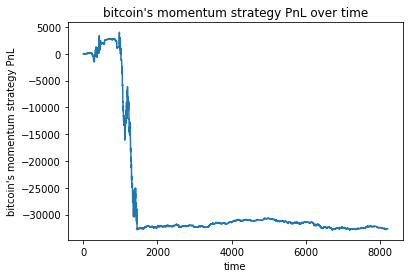
elif(self.predictions[n]==3):

self.alpha[x]=0.5

elif(self.predictions[n]==4):

self.alpha[x]=1

where x the x-th minute interval after the end of sampling period. In sampling period we do not develop the Alpha.



Very bad performance.

Cumulative Statistics:

|  |  |
| --- | --- |
| Total PNL | -3.10300000e+04 |
| Mean Portfolio Return per 5-minute | -1.73534337e-04 |
| Mean Square Portfolio Return per 5-minute | 9.65834602e-05 |
| Standard deviation | 0.00982615621688 |
| Sharpe Ratio | -1.76604496e-02 |

Alpha 11:

Alpha on Five minute granularity

Based on the predictions of the Logistic Regression Model(we predict whether the price change greater than 0 or equalt or less than zero

for x in range(0, n):

if(self.predictions[x]>0):

if(self.alpha[x-1]>0):

self.alpha[x]=self.alpha[x-1]

else:

self.alpha[x]=1

else:

if(self.predictions[x]<0):

if(self.alpha[x-1]==0):

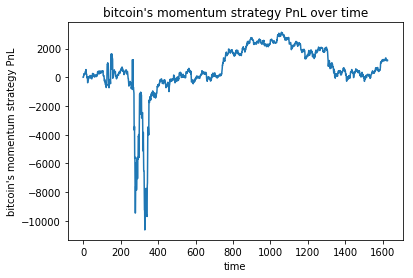
self.alpha[x]=self.alpha[x-1]

else:

self.alpha[x]=0

where x the x-th 5 minute interval after the end of sampling period. In sampling period we do not develop the Alpha.

The performance is as follows:



The time in the abscissa is the index of the five minute intervals.

Cumulative Statistics:

|  |  |
| --- | --- |
| Total PNL | 1.17750000e+03 |
| Mean Portfolio Return per 5-minute | 2.59969873e-04 |
| Mean Square Portfolio Return per 5-minute | 4.22508252e-04 |
| Standard deviation | 0.0205533614579 |
| Sharpe Ratio | 1.26485331e-02 |

To find per day we multiply the returns by 288 and the standard deviation by sqrt(288). Same performance.

Alpha 12

Alpha on One minute granularity

Based on the predictions of the Logistic Regression Model(we predict whether the price change greater than 0 or equal or less than zero

for x in range(0, n):

if(self.predictions[x]>0):

if(self.alpha[x-1]>0):

self.alpha[x]=self.alpha[x-1]

else:

self.alpha[x]=1

else:

if(self.predictions[x]<0):

if(self.alpha[x-1]==0):

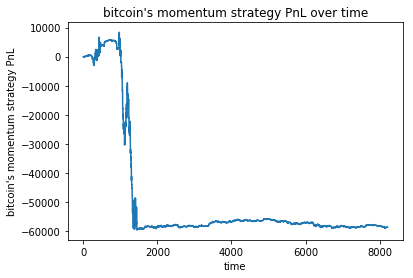
self.alpha[x]=self.alpha[x-1]

else:

self.alpha[x]=0

self.alpha=self.alpha.astype(np.float) where x the x-th minute interval after the end of sampling period. In sampling period we do not develop the Alpha.

In this example we created the same alpha as we did in the five minutes sample and the performance is disastrous:



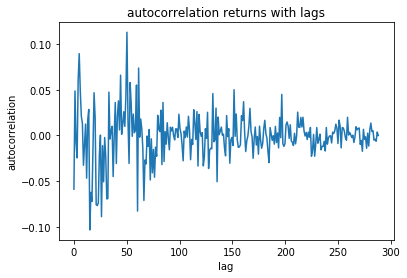
Cumulative Statistics:

|  |  |
| --- | --- |
| Total PNL | -5.86420000e+04 |
| Mean Portfolio Return per 5-minute | -3.19310701e-04 |
| Mean Square Portfolio Return per 5-minute | 3.83321396e-04 |
| Standard deviation | 0.019575991340 |
| Sharpe Ratio | -1.63113426e-02 |

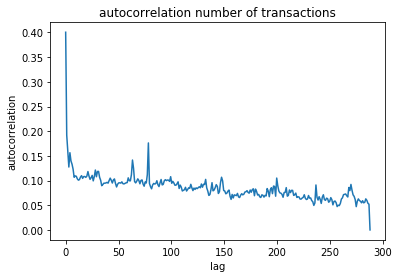
Same performance

Below we found some statistics( autociorrelation functions and means for some measures in x-minute intervals).

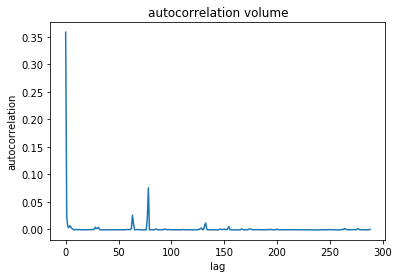
For the five minutes returns this is the autocorrelation function



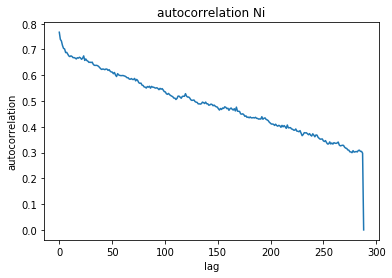
For the number of transactions this is the autocorrelation function:



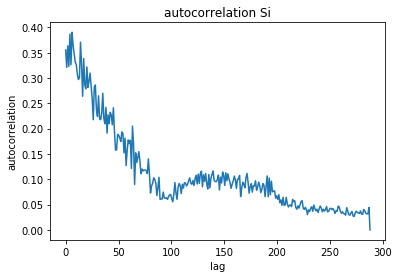
For the volume this is the ACF:



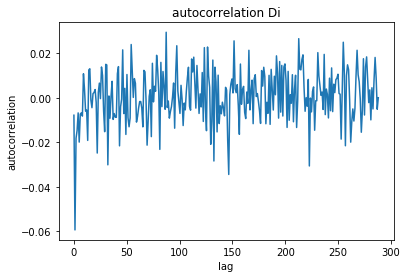
For the Ni(number of transactions with zero change)



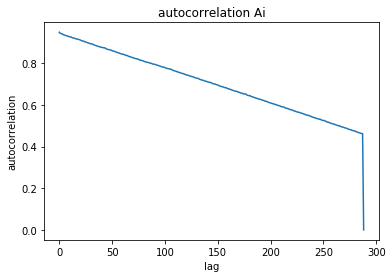
For the Si:



For the Di:

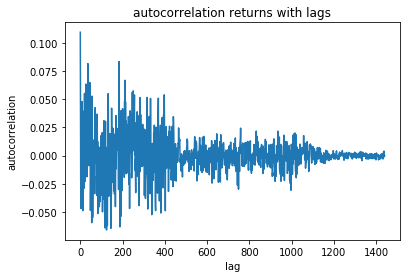


Autocorrelation Ai

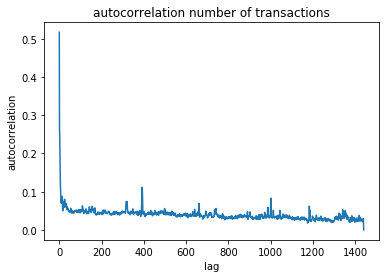


For the one minute case:

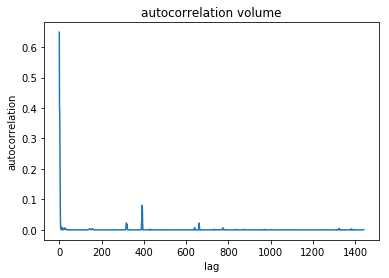
Returns



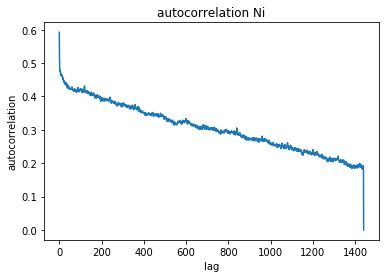
Number of Transactions:



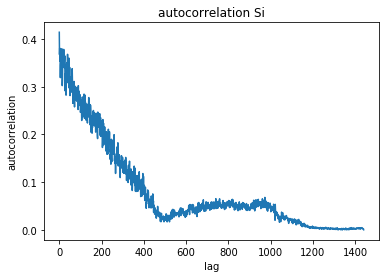
Volume:



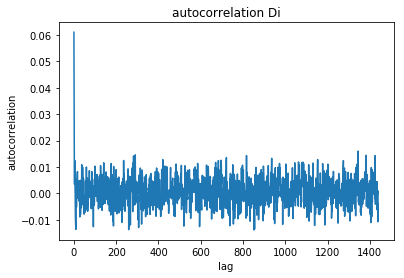
Ni

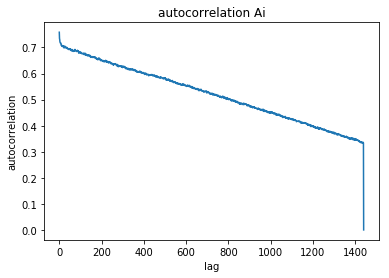


Si:



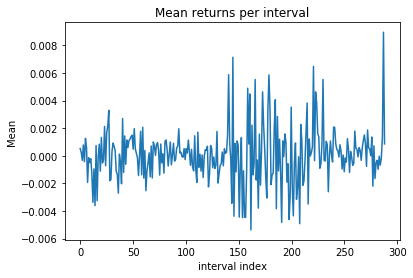
Di



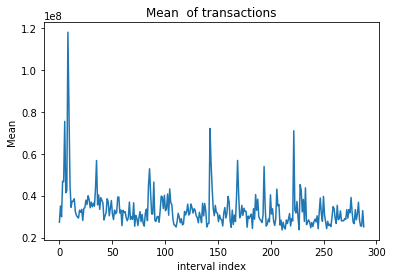


Five minutes interval indices means:

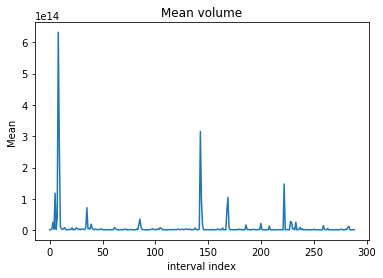
Returns:

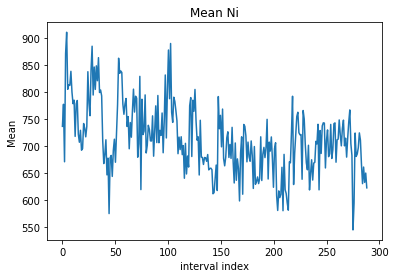


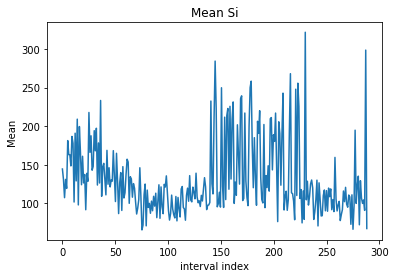
Transactions:

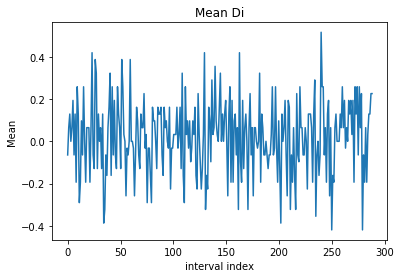


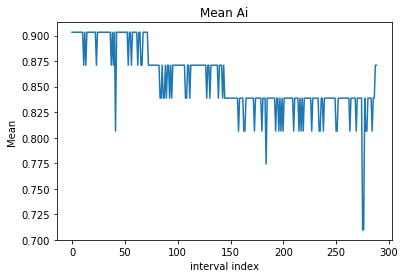
Volume:



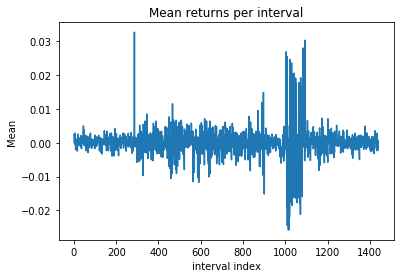


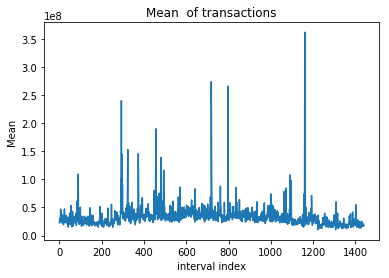


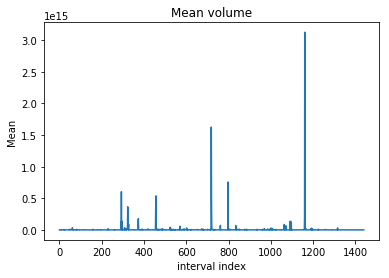


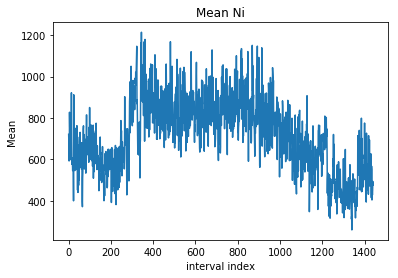


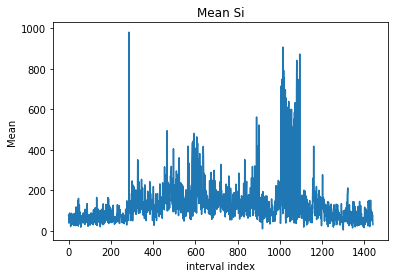
One minute interval means:

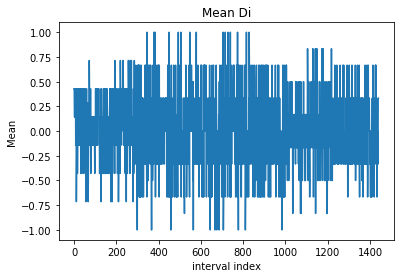


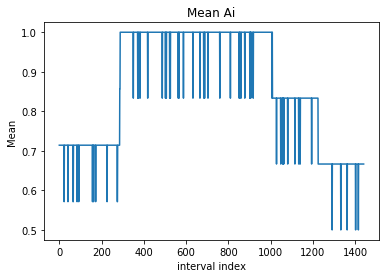












import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.gridspec as gridspec

import math

import scipy.linalg as lin

import scipy

from time import time

import scipy.optimize as op

FeaturesAndLabels=np.load('C:\\Aris\\ud120-projects\\BitcoinFuturesHF\\WFeaturesLabelsIntervals.npz')

OneMinuteFeatures\_train=FeaturesAndLabels['a']

OneMinuteFeatures\_test=FeaturesAndLabels['b']

OneMinuteLabels\_train=FeaturesAndLabels['c']

OneMinuteLabels\_test=FeaturesAndLabels['d']

FiveMinuteFeatures\_train=FeaturesAndLabels['e']

FiveMinuteFeatures\_test=FeaturesAndLabels['f']

FiveMinuteLabels\_train=FeaturesAndLabels['g']

FiveMinuteLabels\_test=FeaturesAndLabels['h']

class LogisticRegressionModel:

def \_\_init\_\_(self, features\_train,labels\_train,features\_test,labels\_test,Dt,PolarValue1,PolarValue2):#Dt one or five

#17,18,19,lags Dy

#25,26,27, lags Ai

#29,30,31 lags Di

#33,34,35 lags Si

#37,38,39 lags Ni

#41,42,43 lags Vi

#45,46,47 lags Bid-Ask\_i

#49,50,51 lags IBS\_i

self.features\_train = features\_train[:,(17,18,19,25,26,27,29,30,31,33,34,35,37,38,39,41,42,43,45,46,47,49,50,51)]

self.labels\_train = labels\_train

self.features\_test = features\_test[:,(17,18,19,25,26,27,29,30,31,33,34,35,37,38,39,41,42,43,45,46,47,49,50,51)]

self.labels\_test= labels\_test

self.Dt=(float(Dt)\*60.0)/100.0

self.PolarValue1=PolarValue1

self.PolarValue2=PolarValue2

#0,1,2,lags Dy

#3,4,5, lags Ai

#6,7,8 lags Di

#9,10,11 lags Si

#12,13,14 lags Ni

#15,16,17 lags Vi

#18,19,20 lags Bid-Ask\_i

#21,22,23 lags IBS\_i

self.features\_train[:,(15,16,17)]=self.features\_train[:,(15,16,17)]/10000 #divide the volume over 10000

self.features\_train[:,(15,16,17)]=(np.power(self.features\_train[:,(15,16,17)],0.5)-1.0)/0.5

self.features\_train[:,(15,16,17)]=np.multiply(self.features\_train[:,(15,16,17)],self.features\_train[:,(21,22,23)])#multiplying with IBS

def fit(self):

from sklearn import linear\_model

self.My\_features=np.zeros(((int(self.features\_train.shape[0])),10))

self.My\_features[:,0]=self.Dt

self.My\_features[:,1:10]=self.features\_train[:,(0,1,2,21,22,23,15,16,17)]

self.My\_features\_test=np.zeros(((int(self.features\_test.shape[0])),10))

self.My\_features\_test[:,0]=self.Dt

self.My\_features\_test[:,1:10]=self.features\_test[:,(0,1,2,21,22,23,15,16,17)]

self.labels\_train[np.where(self.labels\_train>=0)]=self.PolarValue1

self.labels\_train[np.where(self.labels\_train<0)]=self.PolarValue2

self.labels\_train=self.labels\_train.astype(int)

print(self.labels\_train.shape[0])

print(self.labels\_train)

self.Logistic=linear\_model.LogisticRegression()

self.Logistic.fit(self.My\_features,self.labels\_train)

def predict(self):

self.labels\_test[np.where(self.labels\_test>=0)]=self.PolarValue1

self.labels\_test[np.where(self.labels\_test<0)]=self.PolarValue2

self.pred=self.Logistic.predict(self.My\_features\_test)

from sklearn.metrics import accuracy\_score

#self.Accuracy=accuracy\_score(self.labels\_test,self.predict)

class MultinomialLogisticRegressionModel:

def \_\_init\_\_(self, features\_train,labels\_train,features\_test,labels\_test,Dt,PolarValues):#Dt one or five

#17,18,19,lags Dy

#25,26,27, lags Ai

#29,30,31 lags Di

#33,34,35 lags Si

#37,38,39 lags Ni

#41,42,43 lags Vi

#45,46,47 lags Bid-Ask\_i

#49,50,51 lags IBS\_i

self.features\_train = features\_train[:,(17,18,19,25,26,27,29,30,31,33,34,35,37,38,39,41,42,43,45,46,47,49,50,51)]

self.labels\_train = labels\_train

self.features\_test = features\_test[:,(17,18,19,25,26,27,29,30,31,33,34,35,37,38,39,41,42,43,45,46,47,49,50,51)]

self.labels\_test= labels\_test

self.Dt=(float(Dt)\*60.0)/100.0

self.PolarValues=sorted(PolarValues)

#0,1,2,lags Dy

#3,4,5, lags Ai

#6,7,8 lags Di

#9,10,11 lags Si

#12,13,14 lags Ni

#15,16,17 lags Vi

#18,19,20 lags Bid-Ask\_i

#21,22,23 lags IBS\_i

self.features\_train[:,(15,16,17)]=self.features\_train[:,(15,16,17)]/10000 #divide the volume over 10000

self.features\_train[:,(15,16,17)]=(np.power(self.features\_train[:,(15,16,17)],0.5)-1.0)/0.5

self.features\_train[:,(15,16,17)]=np.multiply(self.features\_train[:,(15,16,17)],self.features\_train[:,(21,22,23)])#multiplying with IBS

def fit(self):

from sklearn import linear\_model

self.My\_features=np.zeros(((int(self.features\_train.shape[0])),10))

self.My\_features[:,0]=self.Dt

self.My\_features[:,1:10]=self.features\_train[:,(0,1,2,21,22,23,15,16,17)]

self.My\_features\_test=np.zeros(((int(self.features\_test.shape[0])),10))

self.My\_features\_test[:,0]=self.Dt

self.My\_features\_test[:,1:10]=self.features\_test[:,(0,1,2,21,22,23,15,16,17)]

self.reassignValues()

self.labels\_train=self.labels\_train.astype(int)

print(self.labels\_train.shape[0])

print(self.labels\_train)

self.Logistic=linear\_model.LogisticRegression(multi\_class='multinomial',solver ='newton-cg')

self.Logistic.fit(self.My\_features,self.labels\_train)

def predict(self):

self.pred=self.Logistic.predict(self.My\_features\_test)

from sklearn.metrics import accuracy\_score

self.Accuracy=accuracy\_score(self.labels\_test,self.pred)

def reassignValues(self):

categValues=range(1,len(self.PolarValues)+1)

# if( (len(self.PolarValues)%2)!=0 ):

# categValues=range(- int((len(self.PolarValues)-1)/2), int((len(self.PolarValues)-1)/2)+1)

# else:

# categValues=range(- int((len(self.PolarValues))/2), int((len(self.PolarValues))/2)+1)

# categValues.remove(0)

# if( (len(self.PolarValues)%2)!=0 ):

# categValues=range(- int((len(self.PolarValues)-1)/2), int((len(self.PolarValues)-1)/2)+1)

# else:

# categValues=range(- int((len(self.PolarValues))/2), int((len(self.PolarValues))/2)+1)

# categValues.remove(0)

for lll in range(0,len(self.labels\_train)):

if(self.labels\_train[lll]<self.PolarValues[0]):

self.labels\_train[lll]=categValues[0]

elif(self.labels\_train[lll]>self.PolarValues[-1]):

self.labels\_train[lll]=categValues[-1]+1

else:

for kkk in range(0,len(self.PolarValues)-1):

if( (self.labels\_train[lll]>=self.PolarValues[kkk]) & (self.labels\_train[lll]<=self.PolarValues[kkk+1]) ):

self.labels\_train[lll]=categValues[kkk+1]

for lll in range(0,len(self.labels\_test)):

if(self.labels\_test[lll]<self.PolarValues[0]):

self.labels\_test[lll]=categValues[0]

elif(self.labels\_test[lll]>self.PolarValues[-1]):

self.labels\_test[lll]=categValues[-1]

else:

for kkk in range(0,len(self.PolarValues)-1):

if( (self.labels\_test[lll]>=self.PolarValues[kkk]) & (self.labels\_test[lll]<=self.PolarValues[kkk+1]) ):

self.labels\_test[lll]=categValues[kkk+1]

class GeneralLogisticRegressionModel:

def \_\_init\_\_(self, features\_train,labels\_train,features\_test,labels\_test,Dt,PolarValue1,PolarValue2):#Dt one or five

self.features\_train = features\_train

self.labels\_train = labels\_train

self.features\_test = features\_test

self.labels\_test= labels\_test

self.Dt=(float(Dt)\*60.0)/100.0

self.PolarValue1=PolarValue1

self.PolarValue2=PolarValue2

def fit(self):

from sklearn import linear\_model

self.PolarValue3=0

if(self.PolarValue2==-1):

self.PolarValue2=0

self.PolarValue3=-1

self.labels\_train[np.where(self.labels\_train>0)]=self.PolarValue1

self.labels\_train[np.where(self.labels\_train<=0)]=self.PolarValue2#self.PolarValue2

self.labels\_train=self.labels\_train.astype(int)

self.Logistic=linear\_model.LogisticRegression()

self.Logistic.fit(self.features\_train,self.labels\_train)

def predict(self):

self.labels\_test[np.where(self.labels\_test>=0)]=self.PolarValue1

self.labels\_test[np.where(self.labels\_test<0)]=self.PolarValue2#self.PolarValue2

self.pred=self.Logistic.predict(self.features\_test)

from sklearn.metrics import accuracy\_score

self.Accuracy=accuracy\_score(self.labels\_test,self.pred)

class DecompositionModel:

def \_\_init\_\_(self, my\_features\_train,labels\_train,my\_features\_test,labels\_test,Dt,PolarValues1,PolarValues2):#Dt one or five

self.PolarValues1=PolarValues1

self.PolarValues2=PolarValues2

self.Dt=Dt

features\_train=np.zeros(( int(my\_features\_train.shape[0]),int(my\_features\_train.shape[1])+1 ))

features\_test=np.zeros(( int(my\_features\_test.shape[0]),int(my\_features\_test.shape[1])+1 ))

features\_train[:,-1]=1

features\_test[:,-1]=1

for lll in range(0,int(my\_features\_test.shape[1])):

features\_train[:,lll]=my\_features\_train[:,lll]

features\_test[:,lll]=my\_features\_test[:,lll]

self.TheTest=features\_test

#first regression Ai with respect to Ai lags

self.XFirstRegression=features\_train[:,(52,25,26,27)] #Ai lags

self.XFirstRegression\_test=features\_test[:,(52,25,26,27)] #Ai lags

self.YFirstRegression=features\_train[:,24] #Ai

self.YFirstRegression\_test=features\_test[:,24] #Ai

#second regression Di with respect to Di lags where Ai=1

X=features\_train[:,24]

X\_test=features\_test[:,24]

A=features\_train[np.where(X!=0),:]

B=features\_test[np.where(X\_test!=0),:]

A=A[0]

B=B[0]

self.XSecondRegression=A[:,(52,29,30,31)]

self.XSecondRegression\_test=B[:,(52,29,30,31)]

self.YSecondRegression=A[:,28]

self.YSecondRegression\_test=B[:,28]

# print(self.XSecondRegression.shape)

# print(self.YSecondRegression.shape)

# print(self.XSecondRegression\_test.shape)

# print(self.YSecondRegression\_test.shape)

#third regression S\_i with respect to Si lags where Di>0

A=features\_train[np.where(features\_train[:,28]>0),:]

B=features\_test[np.where(features\_test[:,28]>0),:]

A=A[0]

B=B[0]

self.XThirdRegression=A[:,(52,33,34,35)]

self.XThirdRegression\_test=B[:,(52,33,34,35)]

self.YThirdRegression=A[:,32]#self.YFirstRegression[np.where(labels\_train>0)]

self.YThirdRegression\_test=B[:,32]#self.YFirstRegression\_test[np.where(labels\_test>0)]

#fourth regression S\_i with respect to Si lags

A=features\_train[np.where(features\_train[:,28]<0),:]

B=features\_test[np.where(features\_test[:,28]<0),:]

A=A[0]

B=B[0]

self.XFourthRegression=A[:,(52,33,34,35)] #self.XFirstRegression[np.where(labels\_train<0)]

self.XFourthRegression\_test=B[:,(52,33,34,35)]

self.YFourthRegression=A[:,32]#self.YFirstRegression[np.where(labels\_train<0)]

self.YFourthRegression\_test=B[:,32] #self.YFirstRegression\_test[np.where(labels\_test<0)]

def fit(self):

from sklearn import linear\_model

# print(np.unique(self.YFirstRegression))

# print(np.unique(self.YSecondRegression) )

#First Regression

self.Logistic1=GeneralLogisticRegressionModel(self.XFirstRegression,self.YFirstRegression,self.XFirstRegression\_test,self.YFirstRegression\_test,self.Dt,self.PolarValues1,self.PolarValues2)

self.Logistic1.fit()

self.Logistic1.predict()

#Second Regression

self.Logistic2=GeneralLogisticRegressionModel(self.XSecondRegression,self.YSecondRegression,self.XSecondRegression\_test,self.YSecondRegression\_test,self.Dt,self.PolarValues1,self.PolarValues2)

self.Logistic2.fit()

self.Logistic2.predict()

theta=[0.2,-0.3,-0.3,-0.3]

t0 = time()

resultsPos = op.minimize(self.likelihoodSumGeometricPos, theta, method='nelder-mead')

print "done in %0.3fs" % (time() - t0)

t0 = time()

resultsNeg = op.minimize(self.likelihoodSumGeometricNeg, theta, method='nelder-mead')

print "done in %0.3fs" % (time() - t0)

self.thetaPos=resultsPos.x

self.thetaNeg=resultsNeg.x

def likelihoodSumGeometricPos(self,theta):

n=self.YThirdRegression.shape[0]

summ=0

for lll in range(0,n):

# print(((0.01\*self.XThirdRegression[lll,:])) )

Thelambda=(1.0/(1.0+ np.exp(-np.dot(theta,self.XThirdRegression[lll,:])) ))

if(1-Thelambda>0):

summ=summ+(self.YThirdRegression[lll]-1)\*np.log(1-Thelambda)

return -summ

def likelihoodSumGeometricNeg(self,theta):

n=self.YFourthRegression.shape[0]

summ=0

for lll in range(0,n):

Thelambda=(1.0/(1.0+ np.exp(-np.dot(theta,self.XFourthRegression[lll,:])) ))

if(1-Thelambda>0):

summ=summ+(self.YFourthRegression[lll]-1)\*np.log(1-Thelambda)

return -summ

def Predict(self):

#self.TheTest[:,(52,25,26,27)] #Ai lags

#self.TheTest[:,(52,29,30,31)] #Di lags

#self.TheTest[:,(52,33,34,35)] #Si-1 lags

Predictions=np.zeros((int(self.YFirstRegression\_test.shape[0])))

for lll in range(0,int(self.YFirstRegression\_test.shape[0]) ):

PredictedA=int(self.Logistic1.Logistic.predict(self.TheTest[lll:lll+1,(52,25,26,27)] ) )

if(PredictedA>0):

PredictedD=int(self.Logistic2.Logistic.predict(self.TheTest[lll:lll+1,(52,29,30,31)] ) )

if(PredictedD>0):

Thelambda\_u=(1.0/(1.0+ np.exp(-np.dot(self.thetaPos,0.01\*self.TheTest[lll,(52,33,34,35)]))))

#print(self.thetaNeg)

Predictions[lll]=np.log((1.0-Thelambda\_u)/Thelambda\_u)

else:

Thelambda\_d=(1.0/(1.0+ np.exp(-np.dot(self.thetaNeg,0.01\*self.TheTest[lll,(52,33,34,35)]))))

#print(self.thetaNeg)

Predictions[lll]=-np.log((1.0-Thelambda\_d)/Thelambda\_d)

else:

Predictions[lll]=0

self.Predictions=Predictions

#self.Accuracy2=accuracy\_score(self.YThirdRegression\_test,B)

class PriceChangeAndDurationModel:

def \_\_init\_\_(self, features\_train,labels\_train,features\_test,labels\_test,NiClean\_train,NiClean\_test):

self.features\_train = features\_train

self.labels\_train = labels\_train

self.features\_test = features\_test

self.labels\_test= labels\_test

self.NiClean\_train=NiClean\_train

self.NiClean\_test=NiClean\_test

def fitDt(self):

bnds = ((None, None), (None, None), (None, None), (0.01, None))

#results =op.minimize(PCD.likelihoodSumSiDiPos, theta, method='SLSQP',bounds=bnds)

theta=scipy.rand(4)

FirstLikelihood=self.likelihoodSumDt(theta)

t0 = time()

results = op.minimize(self.likelihoodSumDt, theta, method='SLSQP',bounds=bnds)

print "done in %0.3fs" % (time() - t0)

theta=results.x

FirstLikelihoodNew=self.likelihoodSumDt(theta)

count=0

while (abs((FirstLikelihoodNew-FirstLikelihood)/abs(FirstLikelihood+0.01))>0.1):

FirstLikelihood=self.likelihoodSumDt(theta)

t0 = time()

results = op.minimize(self.likelihoodSumDt, theta, method='SLSQP',bounds=bnds)

print "done in %0.3fs" % (time() - t0)

count=count+1

print(count)

theta=results.x

FirstLikelihoodNew=self.likelihoodSumDt(theta)

self.thetaDt=results.x

return self.thetaDt

def likelihoodSumDt(self,theta):

n=int(self.features\_train.shape[0])

Summ=0

P=4\*math.atan(1)

for x in range(1, n):

#ln(Dti),S2-i

A=np.array([1.0, np.log(self.features\_train[x-1,9]), self.features\_train[x-1,0]])

B=np.array([theta[0],theta[1],theta[2]])

Xbeta= np.dot(A,B)

Yobserved=np.log(self.features\_train[x,9]) #ln(Dti)

operand= np.log(1.0/theta[3]/np.sqrt(2\*P))-pow(Yobserved-Xbeta,2.0)/2.0/theta[3]/theta[3]

if(operand!=0):

Summ=Summ+operand

return -Summ

def fitNiZer(self):

theta=scipy.rand(2)

FirstLikelihood=self.likelihoodSumNiZer(theta)

t0 = time()

results = op.minimize(self.likelihoodSumNiZer, theta, method='nelder-mead')

print "done in %0.3fs" % (time() - t0)

theta=results.x

FirstLikelihoodNew=self.likelihoodSumNiZer(theta)

count=0

while (abs((FirstLikelihoodNew-FirstLikelihood)/abs(FirstLikelihood+0.01))>0.1):

FirstLikelihood=self.likelihoodSumNiZer(theta)

t0 = time()

results = op.minimize(self.likelihoodSumNiZer, theta, method='nelder-mead')

print "done in %0.3fs" % (time() - t0)

count=count+1

print(count)

theta=results.x

FirstLikelihoodNew=self.likelihoodSumNiZer(theta)

self.thetaNiZer=results.x

return self.thetaNiZer

def likelihoodSumNiZer(self,theta):

#NiClean\_train

n=int(self.features\_train.shape[0])

Summ=0

for x in range(0, n):

A=np.array([1.0, np.log(self.features\_train[x,9])])

B=np.array([theta[0],theta[1]])

Xbeta= np.dot(A,B)

operand=np.log(np.exp(Xbeta)/(1.0+np.exp(Xbeta)) )

Summ=Summ+operand

return -Summ

def fitNiPos(self):

theta=scipy.rand(2)

FirstLikelihood=self.likelihoodSumNiPos(theta)

t0 = time()

results = op.minimize(self.likelihoodSumNiPos, theta, method='nelder-mead')

print "done in %0.3fs" % (time() - t0)

theta=results.x

FirstLikelihoodNew=self.likelihoodSumNiPos(theta)

count=0

while (abs((FirstLikelihoodNew-FirstLikelihood)/abs(FirstLikelihood+0.01))>0.1):

FirstLikelihood=self.likelihoodSumNiPos(theta)

t0 = time()

results = op.minimize(self.likelihoodSumNiPos, theta, method='nelder-mead')

print "done in %0.3fs" % (time() - t0)

count=count+1

print(count)

theta=results.x

FirstLikelihoodNew=self.likelihoodSumNiPos(theta)

self.thetaNiPos=results.x

return self.thetaNiPos

def likelihoodSumNiPos(self,theta):

#NiClean\_train

n=int(self.features\_train.shape[0])

Summ=0

for x in range(0, n):

A=np.array([1.0, np.log(PCD.features\_train[x,9])] )

B=np.array([theta[0],theta[1]])

Xbeta= np.dot(A,B)

Thelambda=(np.exp(Xbeta))/(1.0+np.exp(Xbeta))

if(Thelambda==0):

operand=(PCD.NiClean\_train[x]-1.0)\*np.log(1.0-Thelambda)

elif(Thelambda==1):

operand=np.log(Thelambda)

else:

operand=np.log(Thelambda)+ (PCD.NiClean\_train[x]-1.0)\*np.log(1.0-Thelambda)

Summ=Summ+operand

return -Summ

def fitDi(self):

theta=scipy.rand(4)

FirstLikelihood=self.likelihoodSumDi(theta)

t0 = time()

results = op.minimize(self.likelihoodSumDi, theta, method='nelder-mead')

print "done in %0.3fs" % (time() - t0)

theta=results.x

FirstLikelihoodNew=self.likelihoodSumDi(theta)

count=0

while (abs((FirstLikelihoodNew-FirstLikelihood)/abs(FirstLikelihood+0.01))>0.1):

FirstLikelihood=self.likelihoodSumDi(theta)

t0 = time()

results = op.minimize(self.likelihoodSumDi, theta, method='nelder-mead')

print "done in %0.3fs" % (time() - t0)

count=count+1

print(count)

theta=results.x

FirstLikelihoodNew=self.likelihoodSumDi(theta)

self.thetaDi=results.x

return self.thetaDi

def likelihoodSumDi(self,theta):

n=int(self.features\_train.shape[0])

P=4\*math.atan(1)

Summ=0

for x in range(1, n): #Di\_lag, ln(Dt)

A=np.array([1.0, (self.features\_train[x,16]), np.log(self.features\_train[x,9])])

B=np.array([theta[0],theta[1],theta[2]])

Xbeta= np.dot(A,B)

sigma=np.exp(theta[3]\*abs(self.features\_train[x,16]+self.features\_train[x,17]+self.features\_train[x,18]))

Yobserved=(self.features\_train[x-1,16]) #Di

operand= (1.0/sigma/np.sqrt(2\*P))\*np.exp(-pow(Yobserved-Xbeta,2.0)/2/sigma/sigma)

Summ=Summ+np.log(operand)

return -Summ

def fitSiDineg(self):

theta=scipy.rand(4)

#cons= ({'type': 'ineq', 'fun': lambda x: -x[0] })

FirstLikelihood=self.likelihoodSumSiDiNeg(theta)

t0 = time()

results = op.minimize(self.likelihoodSumSiDiNeg, theta, method='nelder-mead')

print "done in %0.3fs" % (time() - t0)

theta=results.x

FirstLikelihoodNew=self.likelihoodSumSiDiNeg(theta)

count=0

while (abs((FirstLikelihoodNew-FirstLikelihood)/abs(FirstLikelihood+0.01))>0.1):

FirstLikelihood=self.likelihoodSumSiDiNeg(theta)

t0 = time()

results = op.minimize(self.likelihoodSumSiDiNeg, theta, method='nelder-mead')

print "done in %0.3fs" % (time() - t0)

count=count+1

print(count)

theta=results.x

FirstLikelihoodNew=self.likelihoodSumSiDiNeg(theta)

self.thetaSiDiNeg=results.x

return self.thetaSiDiNeg

def likelihoodSumSiDiNeg(self,theta):

n=int(self.features\_train.shape[0])

Summ=0

for x in range(1, n): #Di\_lag, ln(Dt)

A=np.array([1.0, (self.NiClean\_train[x]), np.log(self.features\_train[x,9]),self.features\_train[x,0]])

B=np.array([theta[0],theta[1],theta[2],theta[3]])

Xbeta= np.dot(A,B)

Thelambda=np.exp(Xbeta)

if(PCD.features\_train[x-1,0]==0):

divisor=1.0

else:

divisor=scipy.special.gamma(abs(PCD.features\_train[x-1,0]))

if(abs(Thelambda)<0.001):

operand=-Thelambda-np.log(divisor)

else:

operand=-Thelambda+abs(PCD.features\_train[x-1,0]-1.0)\*np.log(Thelambda)-np.log(divisor)

#operand=-Thelambda+abs(PCD.features\_train[x-1,0]-1.0)\*np.log(Thelambda)-np.log(divisor)

#operand=np.log(np.exp(-Thelambda)\*pow(Thelambda,abs(self.features\_train[x-1,0]-1.0))/divisor )

Summ=Summ+operand

return(-Summ)

def fitSiDiPos(self):

theta=scipy.rand(4)

FirstLikelihood=self.likelihoodSumSiDiPos(theta)

t0 = time()

results = op.minimize(self.likelihoodSumSiDiPos, theta, method='nelder-mead')

print "done in %0.3fs" % (time() - t0)

theta=results.x

FirstLikelihoodNew=self.likelihoodSumSiDiPos(theta)

count=0

while (abs((FirstLikelihoodNew-FirstLikelihood)/abs(FirstLikelihood+0.01))>0.1):

FirstLikelihood=self.likelihoodSumSiDiPos(theta)

t0 = time()

results = op.minimize(self.likelihoodSumSiDiPos, theta, method='nelder-mead')

print "done in %0.3fs" % (time() - t0)

count=count+1

print(count)

theta=results.x

FirstLikelihoodNew=self.likelihoodSumSiDiPos(theta)

self.thetaSiDiPos=results.x

return self.thetaSiDiPos

def likelihoodSumSiDiPos(self,theta):

n=int(self.features\_train.shape[0])

Summ=0

for x in range(1, n): #Di\_lag, ln(Dt)

A=np.array([1.0, (self.NiClean\_train[x]), np.log(self.features\_train[x,9]),self.features\_train[x,0]])

B=np.array([theta[0],theta[1],theta[2],theta[3]])

Xbeta= np.dot(A,B)

Thelambda=np.exp(Xbeta)

if(PCD.features\_train[x-1,0]==0):

divisor=1.0

else:

divisor=scipy.special.gamma(abs(PCD.features\_train[x-1,0]))

if(abs(Thelambda)<0.001):

operand=-Thelambda-np.log(divisor)

else:

operand=-Thelambda+abs(PCD.features\_train[x-1,0]-1.0)\*np.log(Thelambda)-np.log(divisor)

#operand=np.log( np.exp(-Thelambda)\*pow(Thelambda,abs(self.features\_train[x-1,0]-1.0))/divisor )

Summ=Summ+operand

return(-Summ)

def predict(self):

# self.thetaSiDiPos, self.thetaSiDiNeg, self.thetaDi,

# self.thetaNiPos, self.thetaNiZer, self.thetaDt

n=int(self.features\_test.shape[0])

self.prediction=np.zeros(n)

dt\_prediction=np.zeros(n)

dt\_prediction[0]=self.features\_test[0,9] #initial conditions

ProbNi0=np.zeros(n)

Ni\_prediction=np.zeros(n)

Ni\_prediction[0]=self.NiClean\_test[0] #initial conditions

Di\_prediction=np.zeros(n)

Di\_prediction[0]=self.features\_test[1,16] #initial conditions

Si\_prediction=np.zeros(n)

Si\_prediction[0]=self.features\_test[1,0] #initial conditions

for x in range(1, n):

dt\_prediction[x]=np.exp( self.thetaDt[0]+self.thetaDt[1]\*np.log(dt\_prediction[x-1])+self.thetaDt[2]\*Si\_prediction[x-1] )

ProbNi0[x]=np.exp(self.thetaNiZer[0]+self.thetaNiZer[1]\*np.log(dt\_prediction[x]) )/(1.0+ np.exp(self.thetaNiZer[0]+self.thetaNiZer[1]\*np.log(dt\_prediction[x]) ) )

A=np.array([1.0, np.log( dt\_prediction[x] )] )

B=np.array([self.thetaNiPos[0],self.thetaNiPos[1]])

Xbeta= np.dot(A,B)

Thelambda=(np.exp(Xbeta)/(1.0+np.exp(Xbeta)) )

Ni\_prediction[x]=1.0+(1.0-Thelambda)/Thelambda

A1=np.array([1.0, (Di\_prediction[x-1]), np.log(dt\_prediction[x])])

B1=np.array([self.thetaDi[0],self.thetaDi[1],self.thetaDi[2]])#the mean only

Di\_prediction[x]=np.sign( np.dot(A,B) )

if(Di\_prediction[x]>0):

A2=np.array([1.0, Ni\_prediction[x], np.log(dt\_prediction[x]),Si\_prediction[x]])

B2=np.array([self.thetaSiDiPos[0],self.thetaSiDiPos[1],self.thetaSiDiPos[2],self.thetaSiDiPos[3]])

Xbeta2= np.dot(A2,B2)

Si\_prediction[x]=Xbeta2#np.exp(Xbeta2)

else:

A2=np.array([1.0, Ni\_prediction[x], np.log(dt\_prediction[x]),Si\_prediction[x]])

B2=np.array([self.thetaSiDiNeg[0],self.thetaSiDiNeg[1],self.thetaSiDiNeg[2],self.thetaSiDiNeg[3]])

Xbeta2= np.dot(A2,B2)

Si\_prediction[x]=Xbeta2#np.exp(Xbeta2)

self.prediction[x]= Di\_prediction[x]\*Si\_prediction[x]

return self.prediction

Alpha Files’ code:

import os

import sys

import numpy as np

import csv

import matplotlib.pyplot as plt

import matplotlib.gridspec as gridspec

import math

import scipy.linalg as lin

import numpy as np

PredictionsAndRealisations=np.load('C:\\Aris\\ud120-projects\\BitcoinFuturesHF\\WPredictionsRealisations.npz')

LRM5Pred=PredictionsAndRealisations['a']

#LRM5Real=PredictionsAndRealisations['b']

LRM1Pred=PredictionsAndRealisations['c']

#LRM1Real=PredictionsAndRealisations['d']

OPM5Pred=PredictionsAndRealisations['e']

#OPM5Real=PredictionsAndRealisations['f']

OPM1Pred=PredictionsAndRealisations['g']

#OPM1Real=PredictionsAndRealisations['h']

DEC5Pred=PredictionsAndRealisations['i']

#DEC5Real=PredictionsAndRealisations['j']

DEC1Pred=PredictionsAndRealisations['k']

#DEC1Real=PredictionsAndRealisations['l']

FeaturesAndLabels=np.load('C:\\Aris\\ud120-projects\\BitcoinFuturesHF\\WFeaturesLabelsIntervals.npz')

OneMinuteFeatures\_test=FeaturesAndLabels['b']

FiveMinuteFeatures\_test=FeaturesAndLabels['f']

FiveMinutesTestReturns=FiveMinuteFeatures\_test[:,16]

OneMinutesTestReturns=OneMinuteFeatures\_test[:,16]

FiveMinutesTestDy=FiveMinuteFeatures\_test[:,12]

OneMinutesTestDy=OneMinuteFeatures\_test[:,12]

class FindingAlphaStatistics:

def \_\_init\_\_(self, predictions,TestDy,TestReturns):

self.predictions=predictions

self.TestDy=TestDy

self.TestReturns=TestReturns

self.n=int(self.TestDy.shape[0])

#Example of an alpha

def definingOrderedProbitAlpha(self):

n=self.n

self.alpha=np.zeros((n))

self.predictions=self.predictions.astype(int)

for x in range(0, n):

if(self.predictions[x]==1):

self.alpha[x]=-1

elif(self.predictions[x]==2):

self.alpha[x]=-0.5

elif(self.predictions[x]==3):

self.alpha[x]=0.5

elif(self.predictions[x]==4):

self.alpha[x]=1

self.alpha=self.alpha.astype(np.float)

return self.alpha

def definingLogisticRegressionAlpha(self):

n=self.n

self.alpha=np.zeros((n))

for x in range(0, n):

if(self.predictions[x]>0):

if(self.alpha[x-1]>0):

self.alpha[x]=self.alpha[x-1]

else:

self.alpha[x]=1

else:

if(self.predictions[x]<0):

if(self.alpha[x-1]<0):

self.alpha[x]=self.alpha[x-1]

else:

self.alpha[x]=-1

self.alpha=self.alpha.astype(np.float)

return self.alpha

def definingDecompositionModelAlpha(self):

n=self.n

self.alpha=np.zeros((n))

self.predictions=self.predictions.astype(int)

for x in range(0, n):

self.alpha[x]=np.sign(self.predictions[x]/10.0)

self.alpha=self.alpha.astype(np.float)

return self.alpha

def definingOrderedProbitAlpha2(self):

n=self.n

self.alpha=np.zeros((n))

self.predictions=self.predictions.astype(int)

for x in range(0, n):

if(self.predictions[x]==1):

self.alpha[x]=0

elif(self.predictions[x]==2):

self.alpha[x]=0

elif(self.predictions[x]==3):

self.alpha[x]=0.5

elif(self.predictions[x]==4):

self.alpha[x]=1

self.alpha=self.alpha.astype(np.float)

return self.alpha

def definingLogisticRegressionAlpha2(self):

n=self.n

self.alpha=np.zeros((n))

for x in range(0, n):

if(self.predictions[x]>0):

if(self.alpha[x-1]>0):

self.alpha[x]=self.alpha[x-1]

else:

self.alpha[x]=1

else:

if(self.predictions[x]<=0):

if(self.alpha[x-1]==0):

self.alpha[x]=self.alpha[x-1]

else:

self.alpha[x]=0

self.alpha=self.alpha.astype(np.float)

return self.alpha

def definingDecompositionModelAlpha2(self):

n=self.n

self.alpha=np.zeros((n))

self.predictions=self.predictions.astype(int)

for x in range(0, n):

if(np.sign(self.predictions[x]/10.0)<0):

self.alpha[x]=0

else:

self.alpha[x]=1

self.alpha=self.alpha.astype(np.float)

return self.alpha

def FindingAlphaStatistics(self):

n=self.n

X = np.linspace(0, n, n, endpoint=True)

print(X)

TheVector=np.zeros((6,1))

PnL=np.zeros((n,1))

MeanPortfolioReturn=np.zeros((n,1))

MeanSquarePortfolioReturn=np.zeros((n,1))

SharpeRatio=np.zeros((n,1))

DeltaPnL=np.zeros((n,1))

PortfolioReturn=np.zeros((n,1))

for kk in range(2,n):

TheVector=self.findingDeltaPnL(kk,TheVector)

PnL[kk]=TheVector[0]

MeanPortfolioReturn[kk]=TheVector[1]

MeanSquarePortfolioReturn[kk]=TheVector[2]

SharpeRatio[kk]=TheVector[3]

DeltaPnL[kk]=TheVector[4]

PortfolioReturn[kk]=TheVector[5]

gs = gridspec.GridSpec(1, 1)

ax1 = plt.subplot(gs[0, :])

ax1.set\_title('bitcoin\'s momentum strategy PnL over time')

ax1.set\_xlabel('time')

ax1.set\_ylabel('bitcoin\'s momentum strategy PnL')

plt.plot(X,PnL)

AlphaStatistics=np.zeros((n,6))

for kk in range(2,n):

AlphaStatistics[kk,0]=PnL[kk]

AlphaStatistics[kk,1]= MeanPortfolioReturn[kk]

AlphaStatistics[kk,2]= MeanSquarePortfolioReturn[kk]

AlphaStatistics[kk,3]=SharpeRatio[kk]

AlphaStatistics[kk,4]= DeltaPnL[kk]

AlphaStatistics[kk,5]= PortfolioReturn[kk]

return AlphaStatistics

def findingDeltaPnL(self,n,MyVector):

alpha=self.alpha

PnL=MyVector[0]

MeanPortfolioReturn=MyVector[1]

MeanSquarePortfolioReturn=MyVector[2]

SharpeRatio=MyVector[3]

DeltaPnL=alpha[n-2]\*(self.TestDy[n-1])

PnL=PnL+DeltaPnL

PortfolioReturn=float(alpha[n-2])\*self.TestReturns[n-1]

MeanPortfolioReturn=(float(n-1)/float(n))\*MeanPortfolioReturn+(1.0/float(n))\*PortfolioReturn

MeanSquarePortfolioReturn=(float(n-1)/float(n))\*MeanSquarePortfolioReturn+(1.0/float(n))\*PortfolioReturn\*PortfolioReturn

StandardDeviationPortfolioReturn=math.sqrt(MeanSquarePortfolioReturn-MeanPortfolioReturn\*MeanPortfolioReturn)

SharpeRatio=MeanPortfolioReturn/StandardDeviationPortfolioReturn

MyVector[0]=PnL

MyVector[1]=MeanPortfolioReturn

MyVector[2]=MeanSquarePortfolioReturn

MyVector[3]=SharpeRatio

MyVector[4]=DeltaPnL

MyVector[5]=PortfolioReturn

print("Mean Portfolio Return: ")

print(MeanPortfolioReturn)

print("Standard Deviation of Portfolio Return: ")

print(StandardDeviationPortfolioReturn)

return MyVector