MARKET PREDICTION WITH MACHINE LEARNING

JAN 2019

He Zhu

Introduction

In this report, I will discuss some popular machine learning techniques and their applications in Stock Market Prediction. The first chapter of this report is about feature engineering(factor construction); The second chapter is about Logistic Regression and its application in market prediction; The third chapter is the extension of Logistic Regression Technique, which talks about Softmax Regression, a multinomial form of Logistic Regression, and its application in market prediction. The fourth chapter is about Naive Bayes Classification Technique and its application in the prediction; The fourth part will focus on SVM(Support Vector Machine) and its application in the prediction; In the fifth chapter, Decision Tree will be applied to make prediction. In the last Chapter, I will try to consider multinomial prediction using softmax regression, Linear SVM.
The major mathematical derivation are based on many famous textbooks and notes, Andrew Ng's CS229 Notes, Paul Wilmott's Machine Learning: An Applied Mathematics Introduction and Mehryar Mohri's Foundation of Machine Learning are 3 major theoretical references of this report.

Contents

Chpater I.Feature Engineering

Feature engineering is usually the first step to create a predictive model, and feature engineering process usually includes following steps:(1) Create and select features based on our research target or business purpose (2) Fit our model using the features (3) Test the features' effectiveness (4)Repeat until model works well.

How do we create features, how do we select features and how do we combine features with our models will directly influence the quality of our model, and in this chapter, we will mainly focus on feature creation (Feature Extraction). Feature selection will be further developed in other chapters, where we apply our features to fit models. When we create features, we will try to dig out more features as we can, which will can help us reduce the risk our features missing. And dealing with the overfitting is the job of feature selection, which will be directly applied and discussed in other chapters

1.1 Return Features

In the first part of chapter I, we will first talk about some features extracted from stock return. Return is very basic attribute of a stock, it can be computed as following:

$$R_T = \frac{P_T - P_{T-\tau}}{P_{T-\tau}} \qquad (1)$$

Where P_t is the stock price of day t. And R_t represents the return of a stock from day $T-\tau$ to T. Another basic feature is the stock's volatility, and it can be calculated as the standard deviation of a stock's return:

$$\sigma_T = \sqrt{\frac{\sum_{t=T-\tau}^{T}(R_t - \overline{R})^2}{\tau - 1}} \qquad (2)$$

Where σ_T is the unconditional volatility of a stock from day $T-\tau$ to T. Because of an important feature of stock return that the long-term mean return of a stock $\overline{R}$ is usually near to 0, so we can express (2) in another form, which is more frequently used in quantitative finance world:

$$\sigma_T = \sqrt{\frac{\sum_{t=T-\tau}^{T} R_t^2}{\tau - 1}} \qquad (3)$$

.

Based on these two simple features, we can derive more features.

1.1.1 EWMA Volatility

EWMA (Exponential Weighted Moving Average) Volatility was invented by RiskMetrics, which tries to model the circumstance that the high-volatility days are usually followed with high-volatility days and low-volatility days tend to be followed by low-volatility days. We can express EWMA Volatility as followings:

$$\Sigma_T = \alpha[\sigma_T + (1-\alpha)\sigma_{T-1} + (1-\alpha)^2\sigma_{T-2} + ... + (1-\alpha)^{\tau-1}\sigma_{T-\tau-1}] \quad (4)$$

Where Σ_T represents the EWMA volatility of day T. By some tricky mathematical trick, we can convert (4) to be:

$$\Sigma_T = \alpha \cdot \sigma_T + (1-\alpha) \cdot \Sigma_{T-1} \qquad (5)$$

Where $T > 0$. With (5), we can have our pseudocode to compute EWMA Volatility like followings:

Algorithm 1 Calculate EWMA Volatility

```
Require: Prices are ascendent ordered by date
Require: $T > 0$ and $\sigma_T \geq 0$
 1: FUNCTION (Compute EWMA Volatility):
 2: $T = 0$, $\alpha = \frac{22}{23}$
 3: for $T < N$ do
 4:     if $T = 0$ then
 5:         $\Sigma_T = \sigma_T$
 6:     else
 7:         $\Sigma_T = \alpha \cdot \sigma_T + (1 - \alpha) \cdot \Sigma_{T-1}$
 8:     end if
 9:     $T = T + 1$
10: end for
11: RETURN $\Sigma_T$
```

As we can see when $t \to T$, the weight of σ_t becomes larger, and as $t \to T$, the weight of σ_t decreases exponentially.

1.1.2 Drift-Independent Volatility

Where $T > 0$. Drift-Independent Volatility is a new volatility estimator found by Dennis Yang and Qiang Zhang, which is independent of either drift motion or opening jumps, and this volatility estimator's variance is minimized among all estimators with the similar properties. Here I simply put the formula to compute the drift-independent volatility:

$$V_T = V_O + k \cdot V_C + (1-k)V_{RS} \qquad (6)$$

Where:

$$V_O = \frac{1}{n-1}\sum_{t=1}^{n}(o_t - \overline{o})^2 \qquad (7)$$

$$V_C = \frac{1}{n-1}\sum_{t=1}^{n}(c_t - \overline{c})^2 \qquad (8)$$

$$\overline{O} = \frac{1}{n}\sum_{t=1}^{n} O_t \ \textit{and} \ \overline{C} = \frac{1}{n}\sum_{t=1}^{n} C_t \qquad (9)$$

$$V_{RS} = \frac{1}{n}\sum_{t=1}^{n}[u_t(u_t - c_t) + d_t(d_t - c_t)] \qquad (10)$$

Where:

$$o_t : lnO_t - lnO_{t-1}, \ \textit{Normalized Open Price}$$

$$u_t : lnH_t - lnO_t, \ \textit{Normalized High Price}$$

$$d_t : lnL_t - lnO_t, \ \textit{Normalized Low Price}$$

$$c_t : lnC_t - lnO_t, \ \textit{Normalized Close Price}$$

Thus we can have pseudocode to compute Drift-Independent Volatility like followings:

Algorithm 2 Calculate Drift-Independent Volatility

```
Require: Prices are descendent ordered by date
 1: FUNCTION Compute Drift Independent Volatility(Prices):
 2: $N = length(\text{PRICES}) - 1$
 3: $K = \frac{0.34}{1.34+\frac{N+1}{N-1}}$
 4: $o = log(\text{OPENPRICES}[1{:}N]) - log(\text{OPENPRICES}[0{:}N\text{-}1])$
 5: $u = log(\text{HIGHPRICES}[1{:}N]) - log(\text{OPENPRICES}[1{:}N])$
 6: $d = log(\text{LOWPRICES}[1{:}N]) - log(\text{OPENPRICES}[1{:}N])$
 7: $c = log(\text{CLOSEPRICES}[1{:}N]) - log(\text{OPENPRICES}[1{:}N])$
 8: $V_O = \text{VAR.S}(o)$, $V_C = \text{VAR.S}(c)$
 9: $V_{RS} = \frac{u\cdot(u-c)+d\cdot(d-c)}{N}$
10: $V_T = V_O + K \cdot V_C + (1-K)V_{RS}$
11: RETURN $\sqrt{V_T}$
```

Now with this pseudocode, we can easily compute each period's unconditional drift-independent volatility by passing different period's price array into the function. We need to notice that in the line 3, we have:

$$k = \frac{0.34}{1.34 + \frac{n+1}{n-1}} \qquad (11)$$

It is because that when (11) holds, our estimator V_T is the minimum-variance unbiased variance estimator, which has been proved by the creators of drift-independent volatility.

1.1.3 Naive Sample Volatility

As I discussed before, we have many ways to interpret the simple volatility, and when we want to use daily return to describe the daily movement of stock price, we usually use the quadratic of a stock's daily return as the daily volatility estimator. Following is the way to compute this naive sample volatility:

Algorithm 3 Calculate Naive Sample Volatility

```
Require: Prices are descendent ordered by date
 1: FUNCTION Compute Sample Naive Volatility(Prices):
 2: $N = length(\text{PRICES}) - 1$
 3: $r = log(\text{CLOSEPRICES}[1{:}N]) - log(\text{CLOSEPRICES}[0{:}N\text{-}1])$
 4: $V_T = \sqrt{\frac{r^T\cdot r}{N-1}}$
 5: RETURN $V_T$
```

Surely, we can use (2) that $\sigma_T = \sqrt{\frac{\sum_{t=T-\tau}^{T}(R_t-\overline{R})^2}{\tau-1}}$ to compute sample naive volatility, but it is not a good approach to depict the volatility of the stock price but a good measure the measure the variation of stock daily return.

1.1.4 Lag Return

Lag Return can be regarded as a simple feature to depict the stock historical price movement, and its computation is very easy, the only thing we need to do is the alignment of features. For example, if you want to compute the lag return where $lag = \tau$, it means that you want to compute the daily return τ days before, the formula is:

$$R_{T-\tau} = \frac{P_{T-\tau} - P_{T-\tau-1}}{P_{T-\tau-1}} \qquad (12)$$

or

$$R_{T-\tau} = log(P_{T-\tau}) - log(P_{T-\tau-1}) \qquad (13)$$

The later one is under continuous compounding consideration. Followings is the pseudocode to compute a stock's lag return:

Algorithm 4 Calculate Lag Return

```
Require: Prices are descendent ordered by date
1: FUNCTION Compute Lag Return(Prices,Lags):
2: $lag = 0$, $R_{Lag}$ = new double[Lags]
3: $LagReturn = log(\textsc{ClosePrices}[1:N]) - log(\textsc{ClosePrices}[0:N-1])$
4: while $lag \leq lags$ do
5:     $R_{Lag}[lag] = LagReturn[lag]$
6:     $lag = lag + 1$
7: end while
8: RETURN $R_{Lag}$
```

Then we get get an array of a stock's lagged return, where the 1^{st} element is today's return, and 2^{nd} element is yesterday's return and so on. By compacting multiple arrays together, we can get a matrix, or a dataframe, representing many days' lagged return. Another importance is that we can extend Lag Naive Sample Volatility based on lag return, which may help us construct our predictive model.

1.2 Technical Indicator Features

Technical Indicator is a very popular method used to predict the stock/future/Forex market dynamics, which includes simple or advanced mathematical operation on historical market data. This part will focus on feature engineering of some very popular indicators in the market:RSI,Stochastic K, MACD, CCI, ATR and Accumulation/Distribution.

1.2.1 RSI

The RSI (Relative Strength Indicator) is a momentum factor which can be used to evaluate the power of overbought and oversold of a market in some periods. The formula to compute RSI is:

$$RSI = 100 - \frac{100}{1+RS} \qquad (14)$$

Where RS is the relative strength factor, which is computed by:

$$RS = \frac{\text{EWMA}(U,n)}{\text{EWMA}(D,n)} \qquad (15)$$

Where $\text{EWMA}(U,n)$ and $\text{EWMA}(D,n)$ are exponential weighted moving average percentage gain and exponential weighted percentage loss during a n-days period. More clearly:

$$U = I_{\{Close_t > Close_{t-1}\}} \cdot R_t \qquad (16)$$

$$D = I_{\{Close_t < Close_{t-1}\}} \cdot R_t \qquad (17)$$

And the notation $I_{\{\cdot\}} = 1$ if the expression in the curve bracket is true, and $I_{\{\cdot\}} = 0$ if the expression in the curve bracket is false.

The most popular interpretation of RSI is when its value is higher than or equal to 70, which means in the recent period, purchase power was so strong and the overbought occurred, so we should consider the stock price may go down; When RSI value is less than or equal to 30, which means in the recent period, sell power was so strong, so we should consider that the stock price may increase.

With this interpretation, when we apply RSI in our predictive model, we cannot only regard RSI value as a continuous feature, but overbought, oversold or oscillation also a discrete feature, which can be labelled as -1, 0,or -1. Pseudocode is attached:

Algorithm 5 Calculate RSI

```
Require: Prices are descendent ordered by date
FUNCTION Calculate RSI( ClosePrices, WindowLength=13):
Us = new int [WindowLength]
Ds = new int [WindowLength]
for WindowLength $\geq 0$ do
    if ClosePrices[WindowLength] > ClosePrices[WindowLength+1] then
        Us[WindowLength] $= log(\frac{\text{ClosePrices[WindowLength]}}{\text{ClosePrices[WindowLength+1]}})$
        Ds[WindowLength] $= 0$
    else if ClosePrices[WindowLength] < ClosePrices[WindowLength+1] then
        Ds[WindowLength] $= log(\frac{\text{ClosePrices[WindowLength]}}{\text{ClosePrices[WindowLength+1]}})$
        Us[WindowLength] $= 0$
    else
        Us[WindowLength] $= 0$, Ds[WindowLength] $= 0$
    end if
    WindowLength = WindowLength - 1
end for
RS $= \frac{\text{EWMA(Us)}}{\text{EWMA(Ds)}}$
RETURN $100 - \frac{100}{\text{RS}}$
```

Another we need to know is that to compute RSI, we can use different methods to get RS, for example, RS can not only obtained by using EWMA, but can also obtained by using simple Moving Average, and which one to choose all depends on our model purpose.

1.2.2 Stochastic Oscillator

Stochastic Oscillator is one another popular momentum indicator, it is computed like followings: v

$$K\% = \frac{Close_T - Low}{High - Low} \qquad (18)$$

Where $Close_T$ is the most recent close price, and the Low, High refer to the lowest and highest price during the period. The period length can be determined by us, and the most frequent one is 14-days. Sometimes we can use moving average technique to smooth stochastic oscillator indicator. Same as RSI, when its value is higher than or equal to the overbought threshold, for example 70, we believe it is time to sell this stock; when its value is less than or equal to the oversold threshold, for example 30, we believe it is time to buy this stock. Following is the pseudocode to compute stochastic oscillator:

1.2.3 CCI

CCI(Commodity Channel Index) is one another momentum indicator, which performs like RSI and stochastic oscillator. It also can be used to determine the overbought condition or oversold condition of a stock. The formula and pseudocode are followings:

Algorithm 6 Calculate Stochastic Oscillator

```
Require: Prices are descendent ordered by date
1: FUNCTION Calculate RSI( ClosePrices, WindowLength=14):
2: LOW = min(ClosePrices[0:WindowLength-1])
3: HIGH = max(ClosePrices[0:WindowLength-1])]
4: CLOSE = ClosePrices[0]
5: K = $\frac{\textsc{Close - Low}}{\textsc{High - Low}} \cdot 100$
6: RETURN K
```

$$CCI = \frac{\textsc{TypicalPrice}_T - \text{MA}}{0.015 \cdot \textsc{MeanDeviation}} \quad (19)$$

Where:

$$\textsc{TypicalPrice}_T = \frac{High_T + Low_T + Close_T}{3} \quad (20)$$

$$\text{MA} = \frac{\sum_{T-\tau}^{T} \textsc{TypicalPrice}_t}{\tau} \quad (21)$$

$$\textsc{MeanDeviation} = (\sum_{t=T-\tau}^{T} |\textsc{TypicalPrice}_T - \text{MA}|)/\tau \quad (22)$$

Algorithm 7 Calculate CCI

```
1: FUNCTION Calculate CCI( Prices, WindowLength=20):
2: $\textsc{TypicalPrices} = (\textsc{ClosePrices}[0]+\textsc{HighPrices}[0]+\textsc{LowPrices}[0])/3$
3: MA = $(\sum_{t=T-\tau}^{T} \textsc{TypicalPrice}_t)/\tau$
4: $\textsc{MeanDeviation} = \sum_{t=T-\tau}^{T} |\textsc{TypicalPrice}_T - \text{MA}|$
5: K = $\frac{\textsc{Close - Low}}{\textsc{High - Low}} \cdot 100$
6: return $(\textsc{TypicalPrice} - \text{MA})/(0.015 \cdot \textsc{MeanDeviation})$
```

When we use CCI as our trading signal, we adapt the almost same approach of RSI to use it. We also set an overbought threshold, for example 150, and an oversold threshold, for example -150, and we sell when the current price is higher than 150, and we short the stock when the RSI is higher than overbought threshold, we long the stock when the current CCI is less than oversold threshold.

1.2.4 ATR

ATR(Actual True Range) is a technical indicator measuring the stock price volatility beyond the scope of statistical views. When we compute ATR, we first need to compute TR (True Range):

$$TR = max[\ (High_T - Low_T), |High_T - Close_{T-1}|, |Low_T - Close_{T-1}|\ ] \quad (22)$$

$$= max(High_T, Close_{T-1}) - min(Low_T, Close_{T-1}) \quad (23)$$

so, we get ATR:

$$ATR = \frac{\sum_{t=T-\tau}^{T} TR_t}{\tau} \quad (24)$$

Formula (24) is usually used to calculate the first ATR, for other ATRs we use updating formula:

$$ATR_T = \frac{ATR_{T-1} \cdot (n-1) + TR_T}{n} \quad (25)$$

Algorithm 8 Calculate ATR

```
Require: 1.Prices are descendent ordered by date
Require: 2.TRs have been computed before
 1: FUNCTION Calculate ATR( TRs, WindowLength = 14):
 2: N = length(TRs), T = N-1, ATR = new double[N - windowLength]
 3: for $T \geq 0$ do
 4:     if T > N-WindowLength then
 5:         PASS
 6:     else if  T = N - WindowLength then
 7:         ATR[T] = $\frac{\text{SUM(TRs[T-windowLength:T-1])}}{\text{WindowLength}}$
 8:     else
 9:         ATR[T] = $\frac{\text{ATR[T-1]}\cdot\text{(n-1)+TRs[T]}}{\text{WindowLength}}$
10:     end if
11:     T = T-1
12: end for
13: RETURN ATR
```

Following is the pseudocode for ATR:

Usually when we apply ATR in real world, we usually divide today's ATR by today's price, which will give us a $ATR\%$:

$$ATR_T\% = \frac{ATR_T}{Price_T} \qquad (26)$$

When we use ATR indicator, we can just regard it as one another type of stock price volatility.

1.2.5 Accumulation/Distribution

Accumulation/Distribution indicator, or ADL, considers both the effect of stock price change and the effect of the trading volume:

$$ADL_T = ADL_{T-1} + CMFV \qquad (28)$$

where:

$$CMFV = \text{Current Money Flow volume} = \frac{(P_{Close} - P_{Low}) - (P_{High} - P_{Close})}{P_{High} - P_{Low}} \cdot V \qquad (29)$$

If the stock price increases but the ADL goes down, it means that the increase is not effective; If the stock price decreases but the ADL is uptrend, it means that here may be a reversal in stock price.The pseudocode for ADL is following:

Algorithm 9 Calculate ADL

```
Require: 1.Prices and Volume are descendent ordered by date
1: FUNCTION Calculate ADL( Prices,Volume):
2: N = length(Prices), T = N-1, ADL = new double[N]
3: ADL[T] = $\frac{(ClosePrices[T]-LowPrices[T])-(HighPrices[T]-ClosePrices[T])}{HighPrices[T]-LowPrices[T]} \cdot Volume[T]$
4: for $T \geq 1$ do
5:     ADL[T-1] = ADL[T] + $\frac{(ClosePrices[T-1]-LowPrices[T-1])-(HighPrices[T-1]-ClosePrices[T-1])}{HighPrices[T-1]-LowPrices[T-1]}$·Volume[T-1]
6:     T=T-1
7: end for
8: RETURN ADL
```

ADL helps us know how strong the buying or selling was during a period, for example, if a stock's close price is 100, its low price is 95, its high price is 103, and its volume of trade is 100, then its CMFV is: $((100-95)-(103-100))/(103-95)*100 = 25$, which means ADL will increase 25. So if the close price is more near to the low price and the volume of trade enlarges, ADL will have a significant drop.

1.2.6 MACD

MACD is one another momentum factor based on EWMA or MA prices. It is computed as following:

$$MACD_T = \text{EWMA}(ClosePrice, 12) - \text{EWMA}(ClosePrice, 26) \qquad (30)$$

Surely we can determine which moving average technique to choose or which price regime to use in our demand. It all depends on our model design. The EWMA$(Price, WindowLength)$'s computation is almost the same as the EWMA volatility's, so here I only display a simple pseudocode for MACD:

Algorithm 10 Calculate MACD

```
Require: 1.EWMA Prices are computed and descendent ordered by date
1: FUNCTION Calculate MACD( EWMA12,EWMA26):
2: MACD = EWMA12 - EWMA26
3: RETURN MACD
```

The most popular trading strategy using MACD indicator is that:If MACD line crosses above the signal line, a buying signal is generated; If MACD line crosses below the signal line, a selling signal is generated.

1.3 Deep in Features

Different from people's usual intuition on machine learning, the most important factor for a machine learning predictive model is not what model you select, what is called model selection; More frequently, the data quality and feature quality matter more for predictive accuracy. This part will first summarize some simple but important properties of our features given above; And then we will engineer, or create, more features from the existing features; Throughout this part, the feature scaling or normalization will also be applied.

1.3.1 How Do Our Features Perform

In this part, we will take MSFT as our example to analyse our features.

I.Lagged Return

Following figure 1 - figure 6 are the histogram and normal qq-plot of lagged return features. As we all know, they must be near to normal distribution but with fat tails. Its numerical properties will be given in the last part of this chapter.

II.Drift Independent Volatility

Following figure 7 - figure 12 are the histogram and normal qq-plot of lagged return features. We are very lucky to meet normal-like distribution again.

III.Naive Sample Volatility

Following figure 13 - figure 14 are the histogram and chi-square qq-plot of naive volatility. From them we can find that the naive volatility is near to chi-square distribution but with fat right tail and flat left

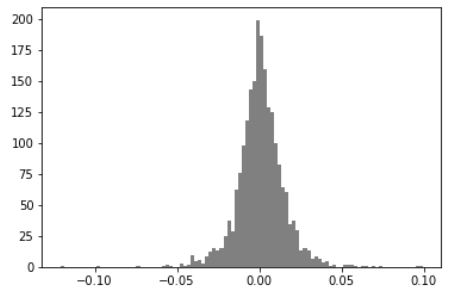


Figure 1: Lag=1 Return

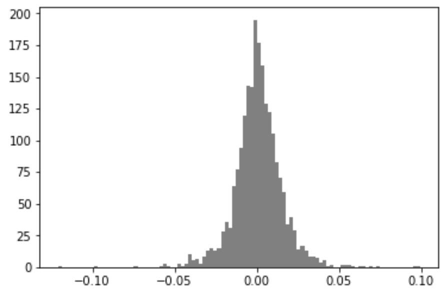


Figure 2: Lag=2 Return

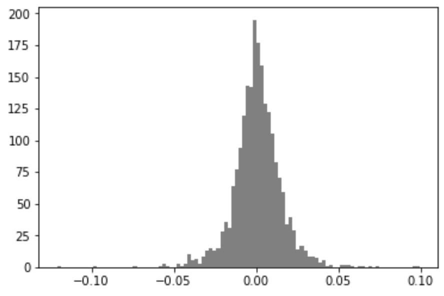


Figure 3: Lag=3 Return

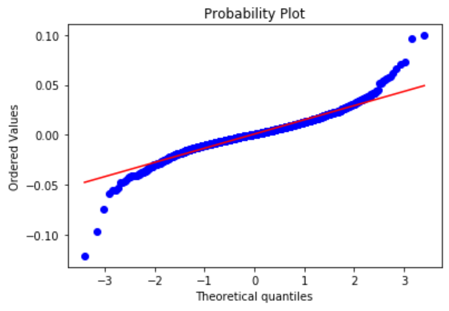


Figure 4: Lag=1 Return QQ-Plot

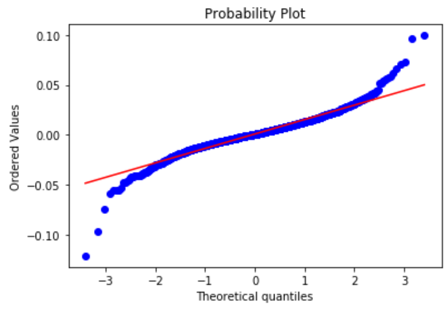


Figure 5: Lag=2 Return QQ-Plot

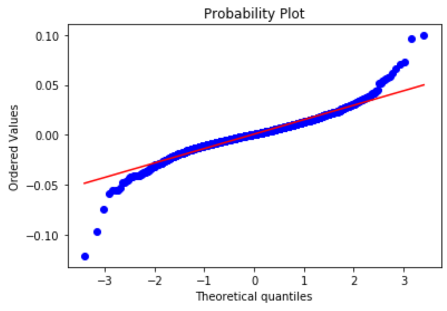


Figure 6: Lag=3 Return QQ-Plot

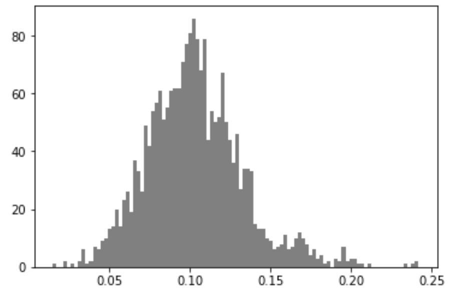


Figure 7: 5-day DI STD

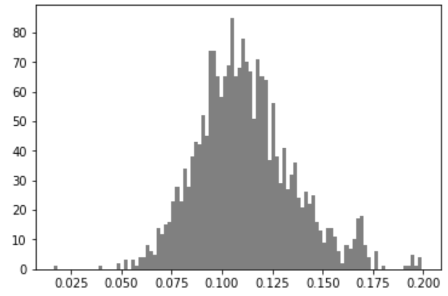


Figure 8: 13-day DI STD

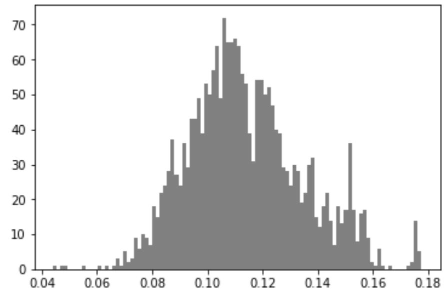


Figure 9: 23-day DI STD

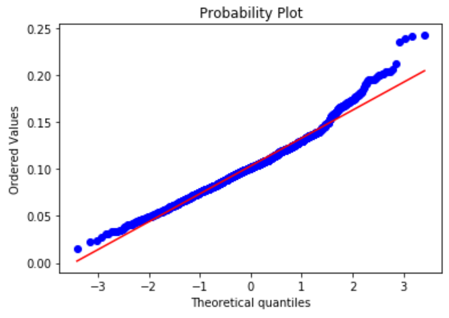


Figure 10: 5-day DI STD QQ-Plot

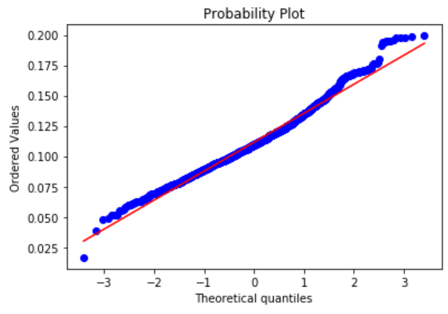


Figure 11: 5-day DI STD QQ-Plot

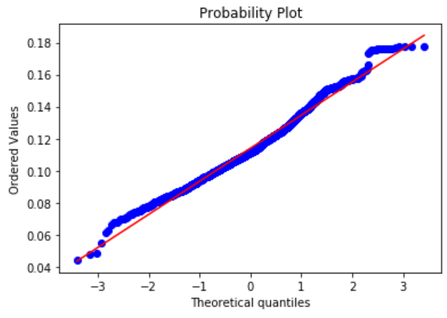


Figure 12: 5-day DI STD QQ-Plot

tail. It is very natural to interpret, because naive volatility is only the square root of return's quadratic. So naive volatility inherits the property from return.

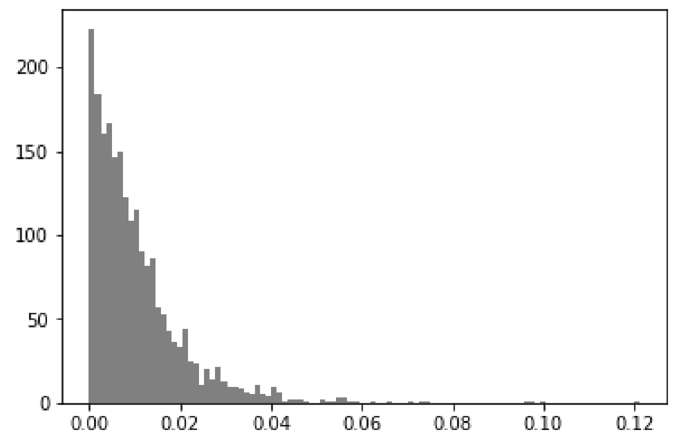


Figure 13: Naive Volatility

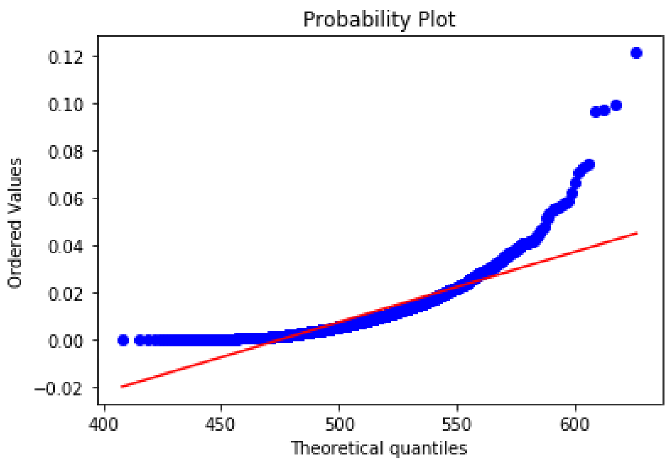


Figure 14: Naive Volatility Chi-Sqaure QQ-Plot

IV.EWMA Volatility

Following figure 15 - figure 20 are the histograms and log-normal qq-plots of naive EWMA. Amazingly we find that though the naive volatility has no direct relationship with normal distribution, the EWMA volatility perfectly follows the log-normal distribution.

V.RSI

Following figure 21 - figure 26 are the histograms and normal qq-plots of RSI.Again this technical indicator also display a strong normality, which is our favourite.

VI.Stochastic Oscillator

Following figure 27 - figure 32 are the histograms and normal qq-plots of CII(Stochastic Oscillator). CCI also performs a normality.

VI.ATR

Following figure 33 - figure 38 are the histograms and log-normal qq-plots of ATR. ATR nearly follows a log-normal distribution.

VI.Accumulation/Distribution

Following figure 39 is the time series of ADL(Accumulation/Distribution). ADL is relatively special compared to other features, but it still has indirect link to normal distribution, which will be discussed later.

VII.MACD

Following figure 40-45 are the histograms and qq-plots of MACD. MACD still has some normality, at least it is symmetrically distributed and in the bell shape.

1.3.2 More Features(1):Discrete Trading Signal

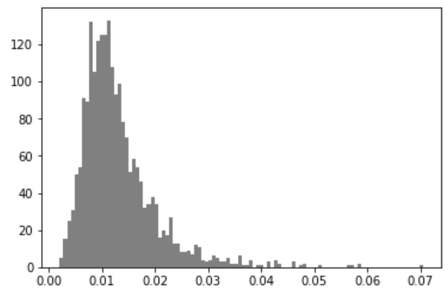


Figure 15: 5-day EWMA STD

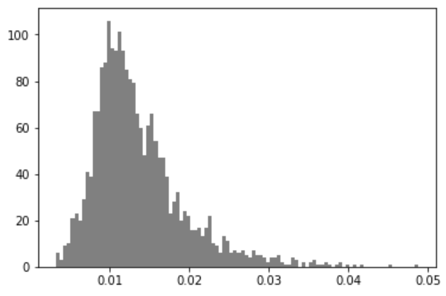


Figure 16: 12-day EWMA STD

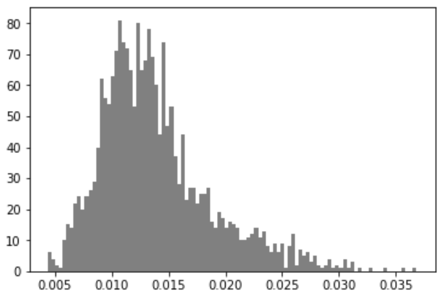


Figure 17: 23-day EWMA STD

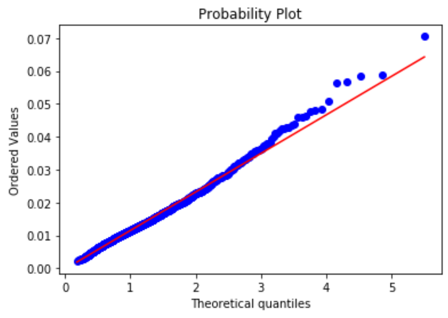


Figure 18: 5-D EWMA STD QQ

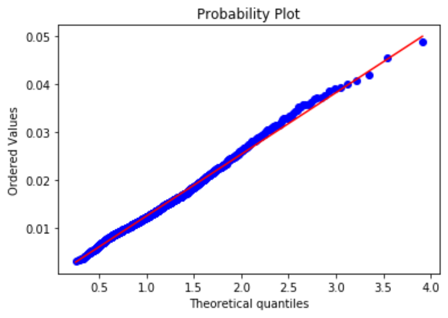


Figure 19: 12-D EWMA STD QQ

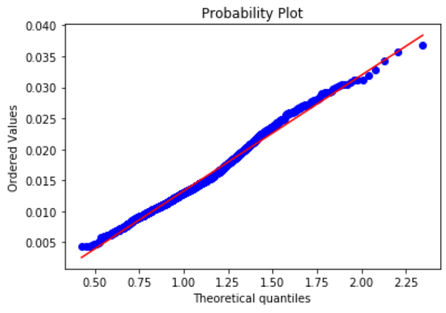


Figure 20: 23-D EWMA STD QQ

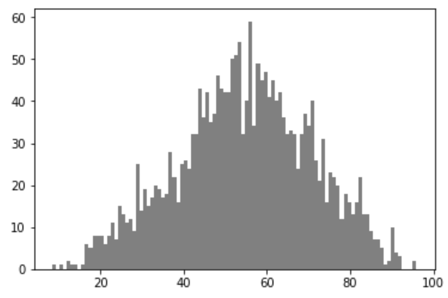


Figure 21: 13-day RSI

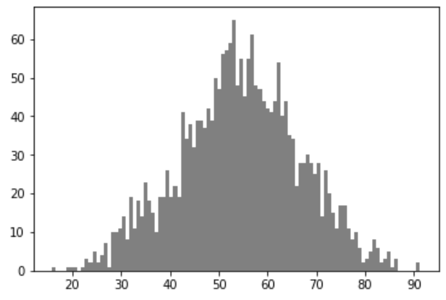


Figure 22: 23-day RSI

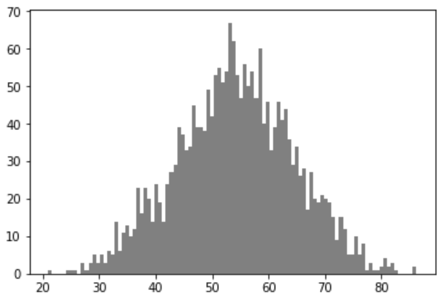


Figure 23: 33-day RSI

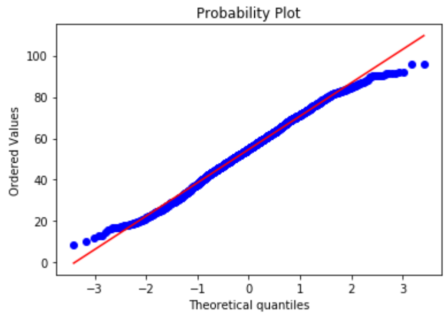


Figure 24: 13-day RSI QQPlot

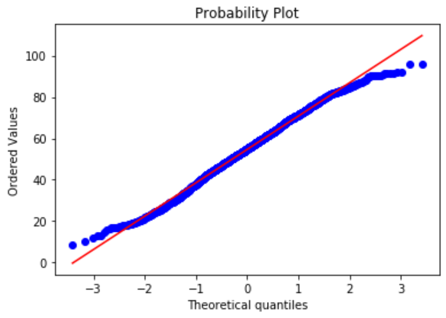


Figure 25: 23-day RSI QQPlot

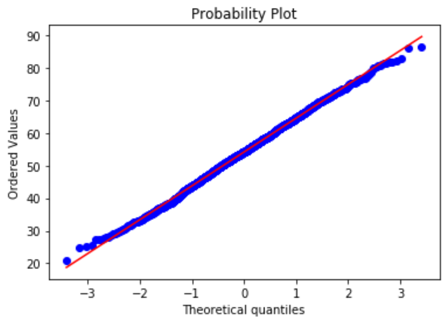


Figure 26: 33-day RSI QQPlot

In this part, we will work on feature transformation, which means we will transform our old features in to new features, and we will use all features to make market prediction.

We need to transform ATR, EWMA Volatility into logarithm form, and then it will nearly follow normal distrbution. We call new features to be Log-ATR, log-EWMA Volatility. Then we need to do some special transformation on naive volatility, though we know this factor cannot be an effective predictor:

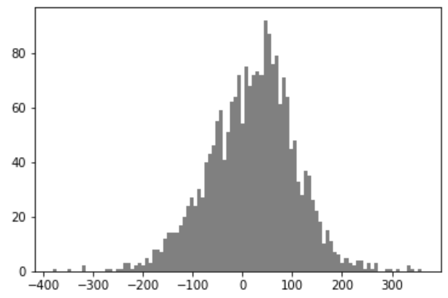


Figure 27: 13-day CCI

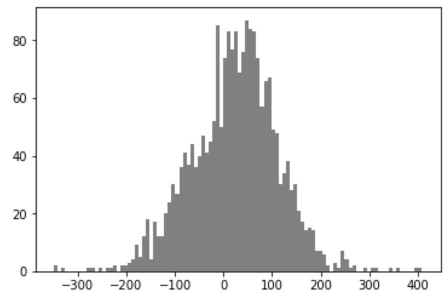


Figure 28: 23-day CCI

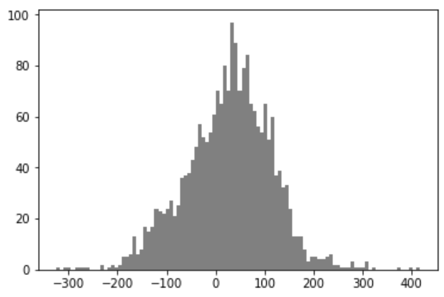


Figure 29: 33-day CCI

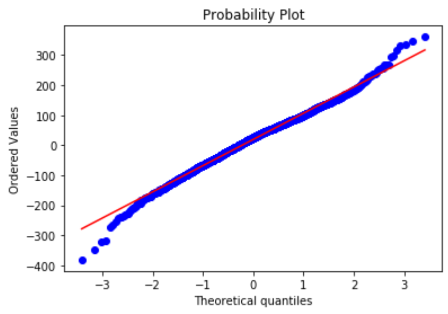


Figure 30: 13-day CCI QQPlot

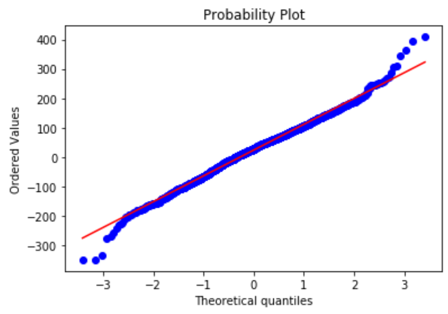


Figure 31: 23-day CCI QQPlot

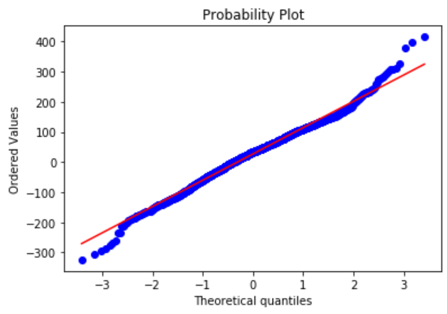


Figure 32: 33-day CCI QQPlot

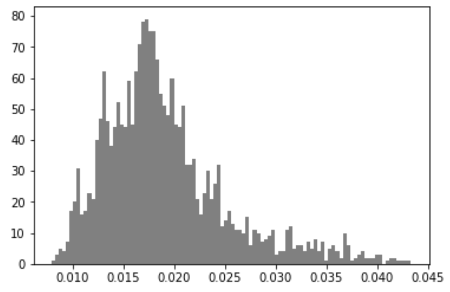


Figure 33: 13-day ATR

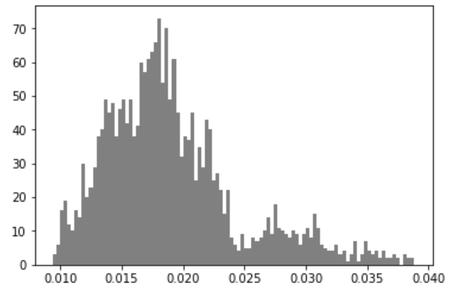


Figure 34: 23-day ATR

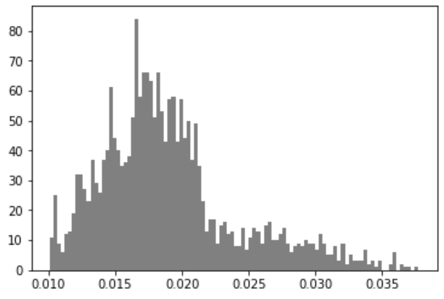


Figure 35: 33-day ATR

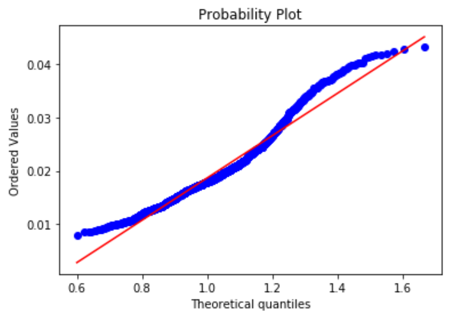


Figure 36: 13-day ATR QQPlot

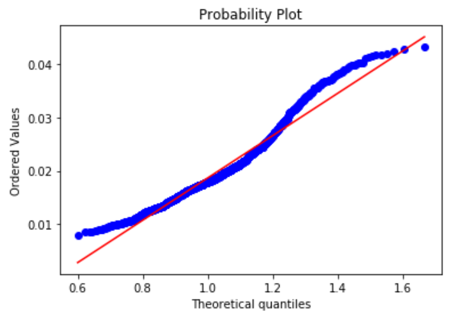


Figure 37: 23-day ATR QQPlot

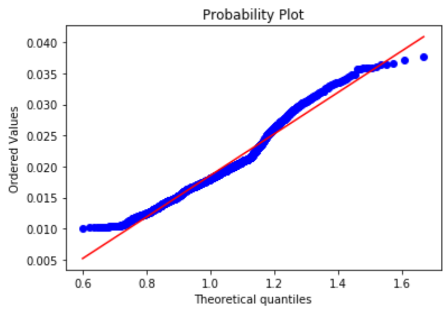


Figure 38: 33-day ATR QQPlot

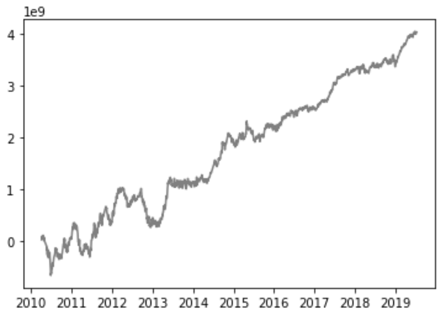


Figure 39: ADL Series

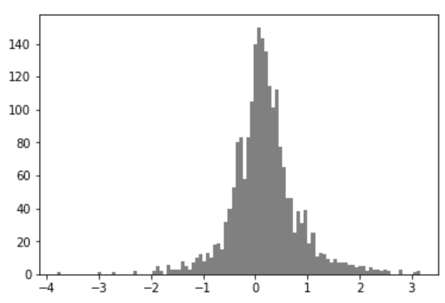


Figure 40: MACD 5:13

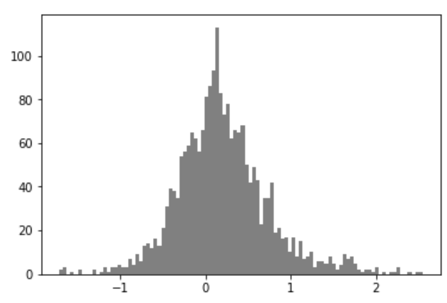


Figure 41: MACD 13:23

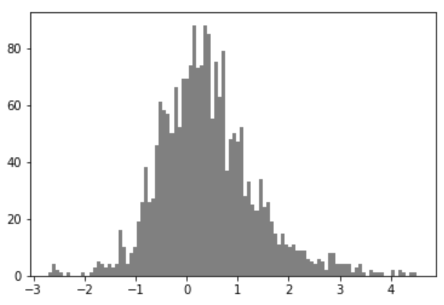


Figure 42: MACD 13-33

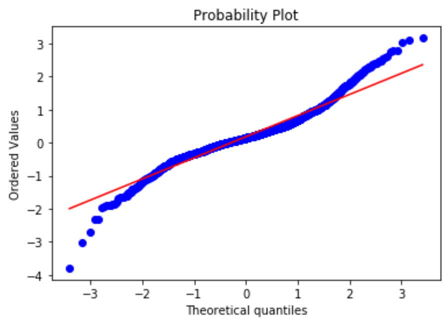


Figure 43: MACD 5:13 QQPlot

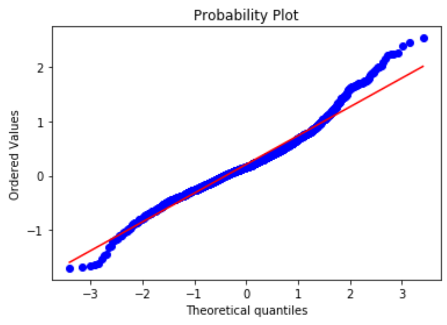


Figure 44: MACD 13:23 QQPlot

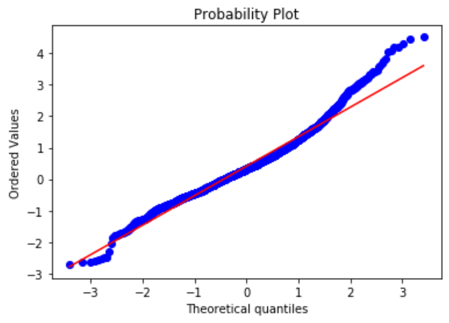


Figure 45: MACD 13:33 QQPlot

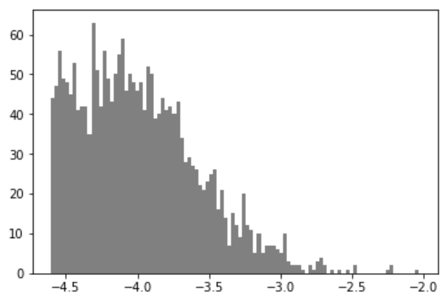


Figure 46: Cox-Box Transformed Naive Volatility

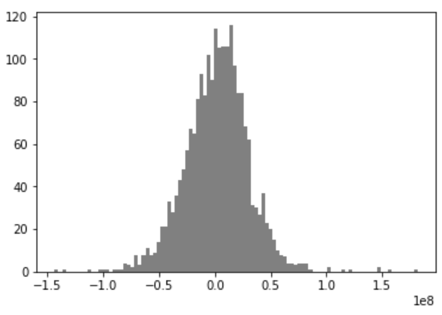


Figure 47: ADL1(1st-Order Diff)

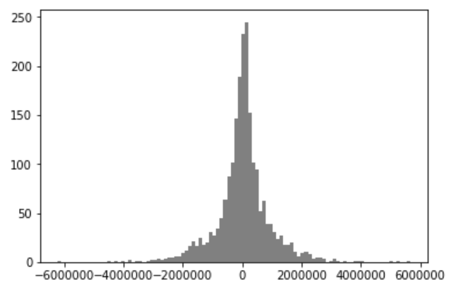


Figure 48: ADL2(ADL1/Close)

$$\hat{ATR} = \frac{x^{\lambda} - 1}{\lambda} \qquad (31)$$

This transformation is called Box-Cox Transformation, which can compress the higher values and expands smaller values. After this operation with $\lambda = 0.25$, we get its distribution like figure 46, which still don't have a good statistical property. But we can still use this kind of features in some distribution

MACD(T1:T2)		
<10% Quantile : >90% Quantile	<20%Quantile : >80% Quantile	<30% Quantile : >70 Quantile
3:-3	2 : -2	1:-1
RSI(T1)		
<10% Quantile : >90% Quantile	<20%Quantile : >80% Quantile	<30% Quantile : >70% Quantile
3:-3	2:-2	1:-1
CCI(T1)		
<10% Quantile : >90% Quantile	<20% Quantile : >90% Quantile	<30% Quantile : >70 Quantile
3:-3	2:-2	1:-1
ADL1/ADL2		
<10% Quantile : >90% Quantile	<20% Quantile : >80% Quantile	<20% Quantile : 80% Quantile
-3:3	-2:2	-1:1
ATR(T1)		
<10% Quantile : >90% Quantile	<20% Quantile : >80% Quantile	<30% Quantile : >70% Quantile
-3:3	-2:2	-1:1

independent models.

Then, we need to transform ADL Series to be stationary, and we can do it by obtaining its 1st-order difference. Then we get its distribution to be like figure 47, which has a beautiful normality. Economically, this is just the Current Money Flow Volume, which has a better statistical property. We call this factor ADL1(Don't worry, a table will be given for depicting of all the features).

We can divide ADL1 by stock's daily close price, and we can also get a very beautiful fat-tailed bell-like distribution in figure 48, we call it ADL2.

Another feature we want to create is the MACD_Signal, which follow the basic usage of MACD: If MACD line crosses above the signal line, a buying signal is generated; If MACD line crosses below the signal line, a selling signal is generated. Here I make buying signal to be +1.5 and selling signal to be -1.5, and others to be 0.

At last, we should normalize all the features except the MACD_Signal, and then all of them are with mean = 0 and standard deviation = 1. The feature names do not change. Here is one thing we need to notice, before we normalize our features, we need to divide MACD by close prices, because as the stock prices change, the scale of MACD also changes, we need to divide it by close prices to make scale stable.Now we can generate more discrete signal based on our current features. The rules are summarized in the table before.

1.3.3 More Features(2):PCA for Continuous Features

When we work on discrete trading signal I mentioned before, we have an embedded logic that if many RSI indicators with different window length trigger a overbought or overbought signal, our model may give us a negative prediction, and we can regard that the more signals are given, more accurate our prediction will be. But when we work on continuous features, multi-linearity or correlation between features matter much, which will directly influence the computation cost and prediction accuracy of our model. So we must do some dimensionality reduction, and PCA (Principal Component Analysis) for this purpose. Following is major steps to complete PCA:

$$X = \begin{bmatrix} X^{(1)^T} \\ . \\ . \\ . \\ X^{(n)^T} \end{bmatrix} \qquad (32)$$

Where X is a feature matrix and

$$X^{(i)^T} = [X_1^{(i)} ... X_d^{(i)}] \qquad (33)$$

Where each element in this vector is the value of a feature. So our matrix is a n by d matrix. Then we can standardized our feature matirx X based on our model design. In this project, we need to standardize it. After standardization:

$$\Sigma = Z^T Z \qquad (34)$$

Where Σ is the covariance matrix of feature matrix X, and Σ must be symmetric and positive semi-definite. Then we need to do eigendecomposition on Σ using following rule:

$$\Sigma = PDP^{-1} \qquad (35)$$

Where P is the matrix of eigenvector and D is a diagonal matrix whose elements on the diagonal are the eigenvalues. Now we need to pay more attention to P and D: Because the eigenvalue on the i-th column points to the i-th column in the matrix P, so we can reorder P in the descendent order of D's eigenvalues, then we can get a new matrix of eigenvector $\hat{P}$, then :

$$\hat{X} = X\hat{P} \qquad (36)$$

Where $\hat{X}$ is our seeking principal component matrix, whose columns are called principal components. Principal components are linearly independent of each other, and they are positioned in the principal component matrix in the order of their explanatory power of variance. Following is the scree plot describing the change of explained variance percentage with the addition of principal components:

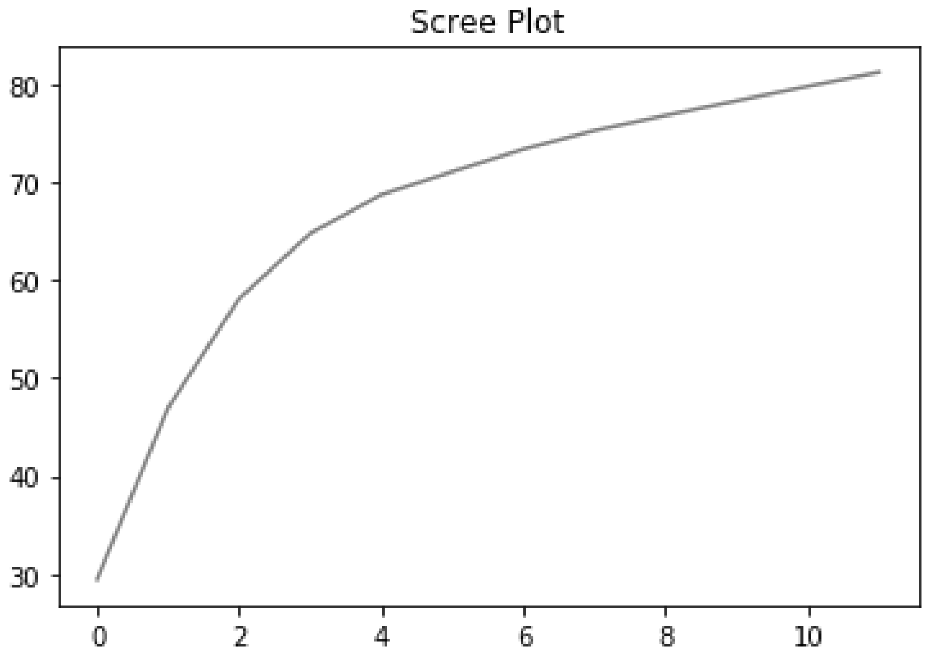


Figure 49: PCA Scree Plot

Before the PCA decomposition, my model have 76 features, and many features are overlapped by others, for example, RSI(13) may have strong linear relationship with RSI(23), or MACD(13) may have strong linear relationship with MACD(23), but after PCA decomposition, our dimensionality can reduced to 12 and 80% variance can be explained by these 12 features , which largely reduces the computation complexity of our models. But we need to notice that because of PCA operation, our independent variables will be more abstract because they become the linear combination of our raw features.

1.3.4 Feature Summary and Engineering Output

In the last part of this chapter I will have a quick review on our feature classification and then engineer the output (dependent variable) quickly.

We can classify our features into three part:

(1) 78 Raw Features: Lagged Return 1 $\sim$ Lagged Return 10; RSI-6, RSI-9, RSI-13, RSI-16, RSI-19, RSI-23, RSI-33; CCI-6, CCI-9, CCI-13, CCI-16, CCI-19, CCI-23, CCI-33; EWMA Volatility-6, EWMA Volatility-9, EWMA Volatility-13, EWMA Volatility-16, EWMA Volatility-19, EWMA Volatility-23, EWMA Volatility-33; Drift-Independent Volatility-6, Drift-Independent Volatility-9, Drift-Independent Volatility-13, Drift-Independent Volatility-16, Drift-Independent Volatility-19, Drift-Independent Volatility-23, Drift-Independent Volatility-33, MACD(5:13), MACD(5:23), MACD(5:33), MACD(5:13), MACD(13:23), MACD(13:33), MACD(13:66), MACD(23:33), MACD(23:44), MACD(23:66), MACD(33:44), MACD(33:66), MACD(33:88); ADL, ADL1, ADL2; Log EWMA Volatility-13, Log EWMA Volatility-23, Log EWMA Volatility-33; Log ATR-13, Log ATR-23, Log ATR-33; Typical Price-6, Typical Price-9, Typical Price-16; Moving Average Typical Price-6, Moving Average Typical Price-9, Moving Average Typical Price-16;

(2) 32 Discrete Signals: MACD Signal(5:13), MACD Signal(5:23), MACD Signal(5:33), MACD Signal(5:13), MACD Signal(13:23), MACD Signal(13:33), MACD Signal(13:66), MACD Signal (23:33), MACD Signal(23:44), MACD Signal(23:66), MACD Signal(33:44), MACD Signal(33:66), MACD Signal(33:88); RSI-6 Signal, RSI-9 Signal, RSI-13 Signal, RSI-16 Signal, RSI-19 Signal, RSI-23 Signal, RSI-33 Signal; CCI-6 Signal, CCI-9 Signal, CCI-13 Signal, CCI-16 Signal , CCI-19 Signal, CCI-23 Signal, CCI-33 Signal; ADL1 Signal, ADL2 Signal; ATR-6 Signal, ATR-9 Signal, ATR-13 Signal, ATR-16 Signal, ATR- 19 Signal, ATR-23 Signal; ATR-33 Signal;

(3)PCA Decomposed Continuous Features: 12 Linear Independent Abstract Features generated by Principal Component Analysis.

Now we can turn our focus into output engineering. We have 3 ways to engineer our output:

(1)Binarization: Return can be positive or negative and sometime may be zero(the probability is almost zero). So we can simply binarize our price change into 2 status: 1 means price increases and -1 means price decreases.

(2)Discretization: Discretization is the generalized form of binarization. While we convert our return to be 0 and 1 in binarization, we can convert our return into multiple values. In this report, we use following rules: When the return is between -0.1 **standard deviation** $\sim +0.1$ **standard deviation**, we convert the return to be 0. If the return is between -1 **standard deviation** ~ -0.1 **standard deviation** or between 0.1 **standard deviation** $\sim$ 1 **standard deviation**, we convert the return to be -1 or 1; If the return is between -1.5 **standard deviation** ~ -1 **standard deviation** or between 1 **standard deviation** $\sim$ 1.5 **standard deviation**, we convert the return to be -2 or 2; Otherwise, return is converted to be -3 or 3.

(3)Raw Return: We do not engineer our return and we build our model to fit continuous outpits.

Chpater II.Logistic Regression with Its Application in Stock Price Prediction

Logistic Regression usually refers to a binary classification learning algorithm in the machine learning, in this chapter, we will focus on this model and its application in stock price prediction

We will have a brief review on key mathematical context of logistic regression and its regularization. Secondly we will develop our predictive model; Finally we will backtest our model using cross validation and correlation analysis.

2.1 Quick Review on Logit Regression

2.1.1 Logistic Regression

Logistic is a *supervised learning* algorithm for *binary classification*, where we usually make y can only be assigned with two values, 0 and 1. We can understand logistic regression in multiple aspects.

For example you can understand logistic regression by regarding it as the *transformation* from the OLS Linear Regression into $[0, 1]$;

More generally, you can understand it as a special form of the *posterior* if some conditional distributional $P(x|y)$ is from *exponential family*, where x means input features, and y is output, 0 or 1. For example, we can prove that one another classification method *Gaussian Discriminative Analysis* can be converted into a logistic regression form mathematically. We can even prove that logistic regression can be constructed using *Generalized Linear Model* where we usually assume the conditional distribution of output y given x follows *Bernoulli Distribution*, which is one of exponential family distribution.

All in all, we can derive and understand logistic in many ways, but the most important part we need to keep in mind that Logistic Regression Model is a robust model without strong distribution assumptions and a model from more generalized models.

Let us first have an easy impression on Logistic Regression Model's mathematical formula:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \qquad (37)$$

As I mentioned before, logistic regression can be regarded as a transformation form of linear regression into value $[0, 1]$, here as we can see, the value $-\theta^T x$ is actually a linear regression prediction. If $-\theta^T x$ is 0, we can say that the predicted $-\theta^T x$ given a training set x is on the decision boundary, and our $h_\theta(x) = g(\theta^T x) = 0.5$, so we cannot determine which classification to put this sample.

For the same reason we can conclude that:

as $-\theta^T x \to \infty$, our $h_\theta(x) \to 0$ and our predicted point will be deeper located in the class labeled with 0.

as $-\theta^T x \to -\infty$, our $h_\theta(x) \to 1$ and our predicted point will be deeper located in the class labeled with 1.

We can also regard $h_\theta(x)$ as a pseudo-distribution

when $h_\theta(x) > 0.5$, we just say that the probability that the sample belongs to class 1 is larger, we simply put it into class 1, thus giving output $y = 1$

when $h_\theta(x) < 0.5$, we just say that the probability that the sample belongs to class 1 is smaller, we simply put it into class 0, thus giving output $y = 0$

So we can summarize that:

$$y = \begin{cases} 1, & \text{if } h_\theta(x) > 0.5; \\ 0, & \text{if } h_\theta(x) < 0.5; \end{cases} \qquad (38)$$

The most usual way we use to fit parameters to a model is to use *Maximum Likelihood Estimation*. For estimation of parameters of logistic regression, we will use this method, too. But this time we will just stop by giving a Likelihood Function of Logistic Regression and the way by which we can approximate our parameters will only be mentioned but not developped.

First, based on our discusson before, it is very natural to assume that:

$$p(y = 1|x; \theta) = h_\theta(x)$$

$$p(y = 0|x; \theta) = 1 - h_\theta(x)$$

It will be a good habit to compact them together, then we get:

$$p(y|x; \theta) = (h_\theta(x))^y((1 - h_\theta(x))^{1-y} \qquad (39)$$

Second, a very important assumption is needed. We assume that here are n training examples were independently generated. Only based on this strong assumption, we can product each sample's probability density function together to get our Maximum Likelihood Function. Now let us write it down:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{n} p(y^{(i)}|x^{(i)}; \theta) \\ &= \prod_{i=1}^{n} (h_\theta(x^{(i)}))^{y^{(i)}}(1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

then we can easily get our log-likelihood function:

$$\begin{aligned} l(\theta) &= logL(\theta) \\ &= \sum_{i=1}^{n} y^{(i)} logh(x^{(i)}) + (1 - y^{(i)})log(1 - h(x^{(i)})) \qquad (40) \end{aligned}$$

Now if we want to fit our θ, we only need to *maximize* $l(\theta)$. There are many methods can be used to maximize this function, such as Batch Gradient Descent Algorithm, Stochastic Gradient Descent Algorithm, Newton Methods and more.

2.1.2 Regularization

Regularization is a technique used to solve the problem of over-fitting in machine learning models, which can help us make a trade-off between variance and bias. Simply speaking, bias can be the error the model does not catch and the variance can be the error caused by over-fitting, and regularization help us get an effective model free from over-fitting problems. We usually use 2 regularization techniques when it comes to logistic regression, one is called L1, or Lasso Regression, and another one is called L2, or Ridge Regression. Simply speaking, these 2 techniques both help us avoid from making our coefficients being too large, thus prevent us from making model over-fitting:

$$l(\hat{\theta}) = \sum_{i=1}^{n} y^{(i)} logh(x^{(i)}) + (1 - y^{(i)})log(1 - h(x^{(i)})) + \frac{\lambda}{2}\theta^T\theta \qquad (41)\textbf{L2: Ridge Regression}$$

$$l(\hat{\theta}) = \sum_{i=1}^{n} y^{(i)} logh(x^{(i)}) + (1 - y^{(i)})log(1 - h(x^{(i)})) + \frac{\lambda}{2}1^T|\theta| \qquad (41)\textbf{L1: Lasso Regression}$$

We can easily find no matter which regularization we choose, they both regularize large weight coefficients, or in other words, those peculiar features. Our target here is still $\mathbf{MAX}(l(\hat{\theta}))$.

2.2 Logit Model Development and Validation

In this part, we will start modelling and validating our model. We have created 3 feature sets in the last chapter and we will fit those sets one by one.
When developing our models, we should consider how much data should be used as training data and how much data should be used as testing data; do we need to reshuffle our data; we should use which regurgitation technique. How do we develop our models will directly influence our model performance. After model development, we need to do model validation. One of the most popular model validation technique is called Cross-Validation; We have other model validation techniques like AUC, ROC, Confusion matrix and so on.

2.2.1 Review on Cross Validation

Cross Validation is kind of out-of-sampling testing technique used to backtest how accurately our predictive model perform. Conventionally, our dataset is usually divided into two parts, training part and testing part, and cross validation is to test how good will our model perform in testing set with estimation from training set.

The basic steps of cross validation are

(1)First, we partite our data into complementary sets.Strictly speaking, there are no limit of set number, and we can partite our data set into thounsands part for backtesting if our dataset is large enough.

(2)Second, we choose one set as our training set to train our model

(3)Third, Test our model by running it on multiple different model

(4)Put our backtesting result together and average our backtesting result

There are many types of cross-validation techniques we usually use. Roughly we can seperate them into two types.

The first type is Exhaustive Cross-Validation techniques, which try to train and test our model on all possible subsets.

The second type is Non-Exhaustive Cross-Validation, which can be regarded as an approximation and simplicity of exhaustive validation, because we don't want to catch all possible validation possibilities.

In this report, I will use a validation method called $k - fold\ crossvalidation$ method. This method requires us to split our data set into K number of consecutive sections, and (k-1) subsets are used to train the model and the remained one is used to test our data. We will have k validation rounds and in each round, one subset plays the role of testing set and others plays the roles of training set. At last we can compute our Final Accuray by average our individual accuracy ratio.

We need to notice that cross validation techniques have some drawback, for example, it will reduce the volume of training set which may cause models become more biased; Some if we decided to use cross validation, we should make sure our dataset is big enough.

2.2.2 Discrete Features Case

The models based on discrete features are not very effective. The figure 50 is the AUC of the model using L2 regularization and the data is reshuffled; The figure 51 is the AUC of the model using L2 regularization and the data is not reshuffled; The figure 52 is the AUC of the model using L1 regularization and the data is reshuffled; The figure 53 is the AUC of the model using L1 regularization and the data is not reshuffled. All above models' 2-folds cross validation scores are near to 50%, 5-folds cross validation score is near to 47%. We can say that our models are almost random models, which cannot help us predict the market movement effectively; But we can still find that when the data is reshuffled, our prediction can be a little more accurate, but the point is that in real world, market data cannot be reshuffled.

2.2.3 Raw Features Case

When we work on raw features, our model can be a little more robust especially when it comes to population set. Figure 54 $\sim$ 57 are the plots describing the accuracy of models. Compared to the models with discrete data, this type of model performs better. All above models' 2-folds cross validation score is near to 51%, 5-folds cross validation scores are near to 47%. There is no surprise that raw features perform better than those with discrete features, because logistic regression is design to fit continuous data.

2.2.4 PCA Features Case

When we fit our models in principal components, our models do not perform as good as the models with raw features. Their AUC curves are again near random classifier lines. But they perform better than discrete features based models. We need to notice that when we allow reshuffling and use L2 regularization, PCA features based model performs good in both test set and population set. All above models' 2-folds cross validation scores are near to 50%, 5-folds cross validation scores are still near to 47%. (Pictures are in the page 23)

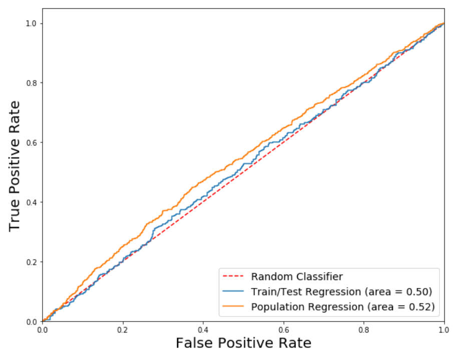


Figure 50: Discrete Features Logit L2 AUC: Shuffle=True

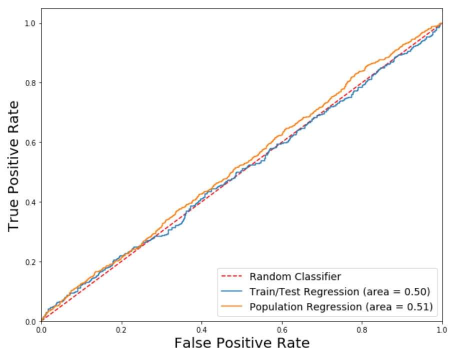


Figure 51: Discrete Features Logit L2 AUC: Shuffle=False

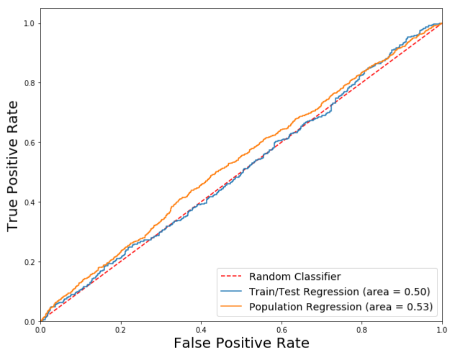


Figure 52: Discrete Features Logit L1 AUC: Shuffle=True

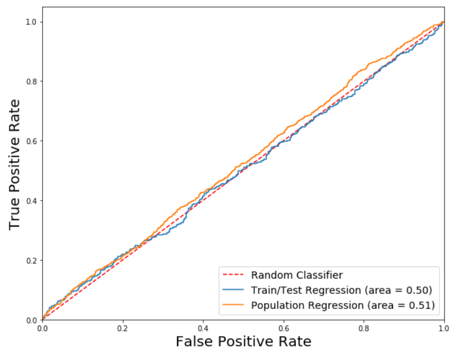


Figure 53: Logit L1 AUC: Shuffle=False

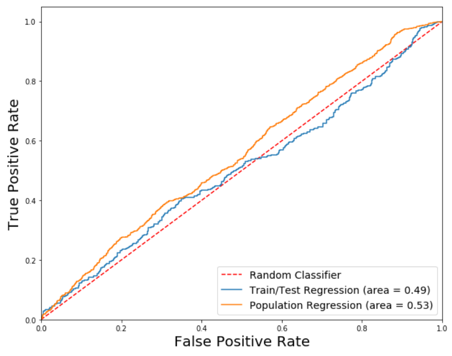


Figure 54: Raw Features Logit L2 AUC: Shuffle=False

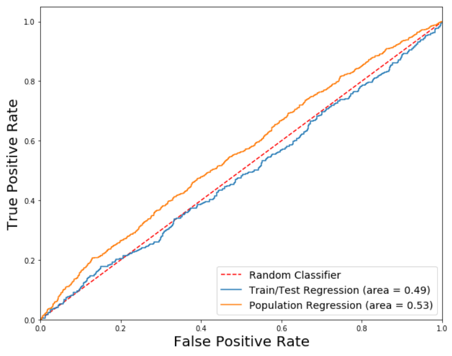


Figure 55: Raw Features Logit L2 AUC: Shuffle=True

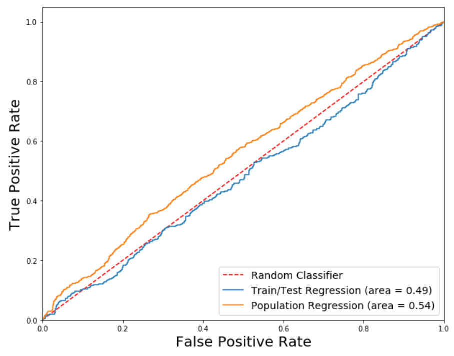


Figure 56: Raw Features Logit L1 AUC: Shuffle=True

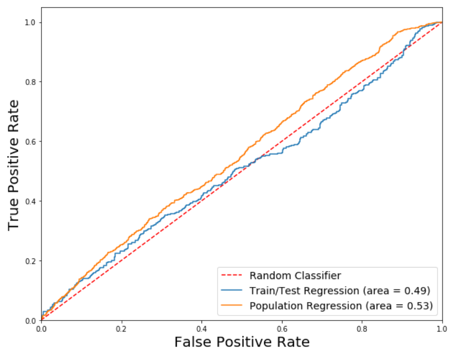


Figure 57: Raw Features Logit L1 AUC: Shuffle=False

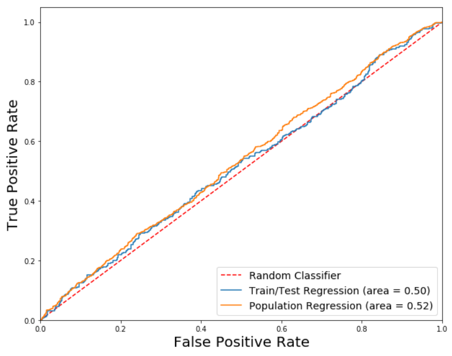


Figure 58: PCA Features Logit L1 AUC: Shuffle=True

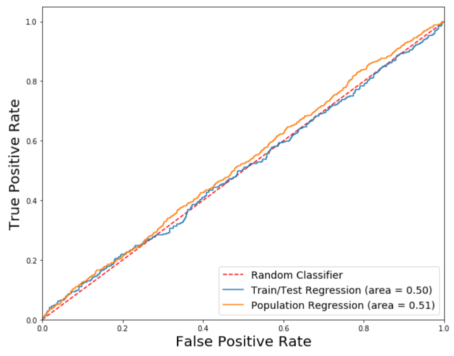


Figure 59: PCA Features Logit L1 AUC: Shuffle=False

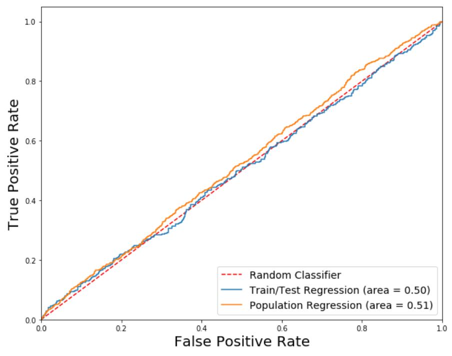


Figure 60: PCA Features Logit L2 AUC: Shuffle=False

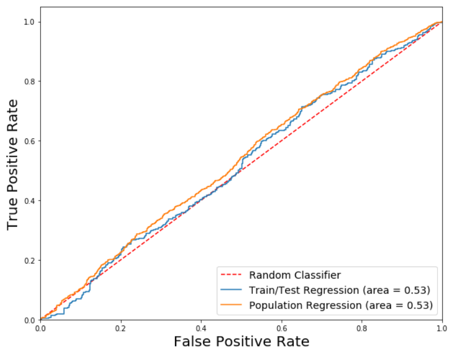


Figure 61: PCA Features Logit L2 AUC: Shuffle=True

Chpater III.Naive Bayesian Classifier with Its Application in Stock Price Prediction

3.1 Quick Review on Naive Bayesian

Now we will introduce a new classifier to predict our stock price movement, it is called Naive Baysian Classifier. This algorithm is relatively broad, and it includes many sub-algorithms, I think scikit-learn official websites can give us a very good category: 1. Gaussian Naive Bayes 2. Multinomial Naive Bayes 3. Complement Naive Bayes 4. Bernoulli Naive Bayes

Before we start talking about these algorithms, it is better to have an overview first. Compared to Logistic Regression, which is a kind of Discriminative Learning Algorithm, Bayesian Classifier is a kind of Generative Learning Algorithm.

Briefly, if you want to label output directly using input, we say that you are using Discriminative Learning Algorithm. Using logistic regression as an example, we usually predict the output probability distribution directly as follows:

$$p(y|x;\theta) = h_\theta(x)^y(1-h_\theta(x))^y$$

we can easily find that there is no explicit assumption on x's distribution, and we input x and then directly get our target probability distribution of y conditional on x.

Different from the Discriminative Learning Algorithm, Generative Learning Algorithm requires us to model the conditional probability of x conditional on y, and then we pick the nearest y to our sample, mathematically, it can be expressed as:

we first compute

$$p(x|y), p(y), and\ p(x)$$

then we get

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)} \qquad (42)$$

For better explaining how our model is going, we will concentrate on 2 most frequently used Naive Bayes:Gaussian Bayes and Bernoulli Bayes.

3.2 Gaussian Naive Bayesian Classifier

Gaussian Naive Bayes Classifier is a generative classifier which models $p(x|y)$ and limits :

$$x|y = 0 \sim N(\mu_0, \Sigma)$$

$$x|y = 1 \sim N(\mu_1, \Sigma)$$

$$y \sim Bernoulli(\phi)$$

Explicitly:

$$p(y) = \phi^y(1-\phi)^{1-y}$$

$$p(x|y=0) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)) \qquad (43)$$

$$p(x|y=1) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} exp(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)) \qquad (44)$$

Note that here we implictly assume that there is only one covariance matrix Σ

we need to notice that

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

where $p(x)$ is a known unconditional distribution, which is a constant, so if we want to maximize the likelihood function of $p(y|x)$, we can just maximize $p(x|y)p(y) = p(x,y)$

Then we can get the log-likelihood of joint distribution of x and y :

$$l(\phi, \mu_0, \mu_1, \Sigma) = log \prod_{i=1}^{n} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \qquad (45)$$

$$= log \prod_{i=1}^{n} p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma)p(y^{(i)}; \phi) \qquad (46)$$

Then we can get:

$$\phi = \frac{1}{n}\sum_{i=1}^{n} 1\{y^{(i)} = 1\} \qquad (47)$$

$$\mu_0 = \frac{\sum_{i=1}^{n} 1\{y^{(i)} = 0\}x^{(i)}}{\sum_{i=1}^{n} 1\{y^{(i)} = 0\}} \qquad (48)$$

$$\mu_1 = \frac{\sum_{i=1}^{n} 1\{y^{(i)} = 1\}x^{(i)}}{\sum_{i=1}^{n} 1\{y^{(i)} = 1\}} \qquad (49)$$

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T \qquad (50)$$

As the Gaussian Naive Bayesian Classifier requires, only after we make our data be continuous and nearly following a Gaussian distribution can we start using this model, so we have to exclude discrete signals here. Firstly, let us consider the conditional distribution of our raw features: RSI, MACD, Lagged Return, CCI, Log-EWMA Volatility, drift-independent volatility, ADL1, ADL2, Log-ATR are the features which meet the requirement. They are conditionally nearly following a normal distribution on last days' return. Secondly, we need to consider the conditional distribution of PCA features, though many of them are not normal distributed but their distributions are still like a bell, so we can loosely regard all the PCA features qualified. Actually, we can use other statistical techniques to test their normality but I would not like to expand too much. Following figure 62 $\sim$ 67 are some examples of feature's conditional distribution.

As we did in the logistic regression, we still make the return less than or equal to 0 to be -1 and others to be 1; And then we can get our model. When we make test size to be 50% of all the data, we get the AUC in the figure 68; Our 2-folds cross validation score is 50.4% and 5-folds cross validation score is 49.3%. And we can get its confusion matrix I in the bottom of page 26, and we can conclude that no matter which model we use Gaussian Bayesian Classifier is still not a good enough model for modelling the market movement with these features

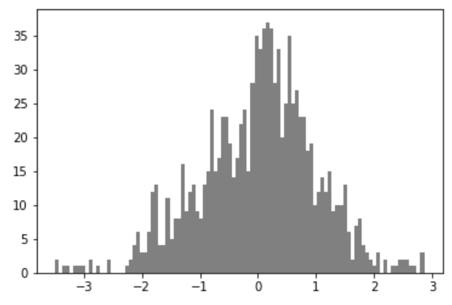


Figure 62: conditional MACD(13,23)

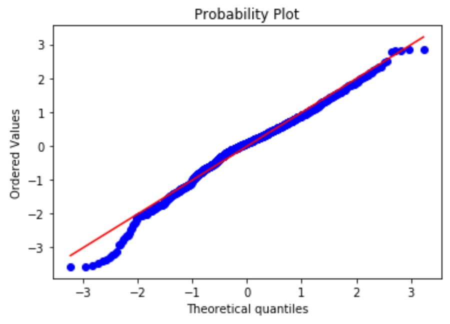


Figure 63: QQPlot:conditional MACD(13,23)

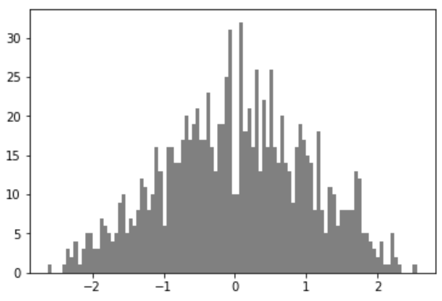


Figure 64: Conditional RSI13

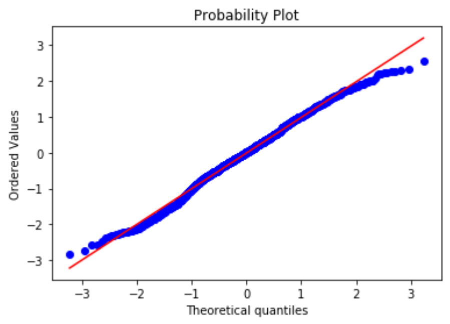


Figure 65: QQPlot Conditional RSI

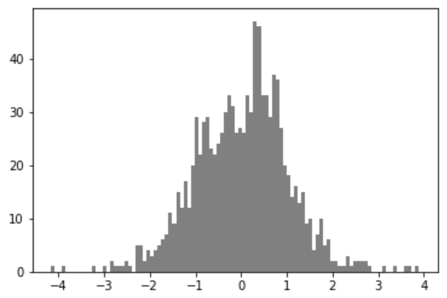


Figure 66: Conditional CCI

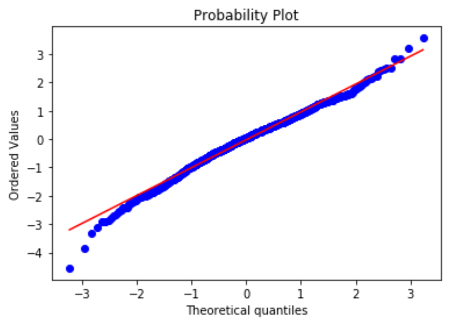


Figure 67: QQPlot Conditional CCI

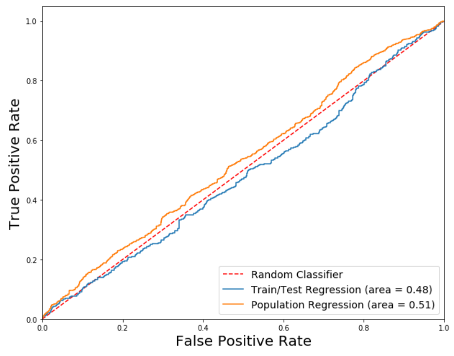


Figure 68: AUC of Gaussian Naive Bayes

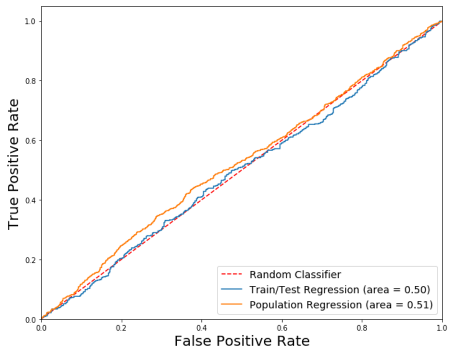


Figure 69: AUC of Bernoulli Naive Bayes

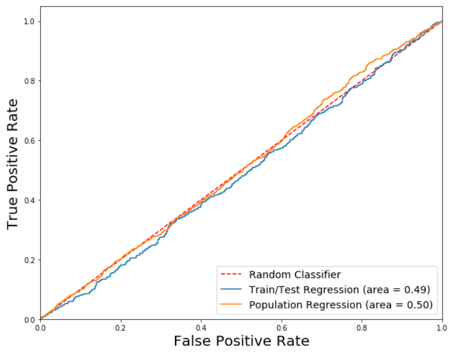


Figure 70: AUC of Multinomial Naive Bayes

3.3 Multinomial and Bernoulli Naive Bayes Classifier

3.3.1 Implementation of Bernoulli Naive Bayes Classifier

Suppose we have a feature vector with k-dimension as follow:

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

This vector's element can only have two value, 1 or 0.

Confusion Matrix I	Predicted Positive	Predicted Negative
Actual Positive	231	377
Actual Negative	295	398

Confusion Matrix II	Predicted Positive	Predicted Negative
Actual Positive	235	267
Actual Negative	264	318

As we did in other generative algorithm, we should model :

$$p(x|y)$$

We need a strong but very useful assumption that:

$$x_i's \text{ are conditionally independent given y}$$

which means

$$p(x_i|y, x_j) = p(x_j|y)$$

So we can have:

$$P(x_1, ..., x_n|y)$$

$$= p(x_1|y)p(x_2|y, x_1)p(x_3|x_1, x_2)...p(x_n|y, x_1, ..., x_{n-1}) \qquad (51)$$

$$= p(x_1|y)...p(x_n|y)$$

$$= \prod_{n}^{j=1} p(x_j|y)$$

Let us turn back to model $p(x|y)$
As we did before,

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

where $p(y)$ is know so we can regard it as constant, so if we want to maximize the likelihood function of $p(x|y)$, we only need to maximize the log-likelihood function of $p(y|x)p(x) = p(x, y)$. So now our problem is just to finding the parameters of our model, they are

$$p(x_j = 1|y = 1)$$

$$p(x_j = 0|y = 1)$$

$$p(x_j = 1|y = 0)$$

$$p(x_j = 0|y = 0)$$

$$p(y)$$

Because

$$p(x_j = 1|y = 1) + p(x_j = 0|y = 1) = 1$$

$$p(x_j = 1|y = 0) + p(x_j = 0|y = 0) = 1$$

So we only need to make $\phi_{j|y=1} = p(x_j = 1|y = 1)$,$\phi_{j|y=0} = p(x_j = 1|y = 0)$, $\phi_y = p(y = 1)$ and then

$$p(x_j = 0|y = 1) = 1 - \phi_{j|y=1}$$

and

$$p(x_j = 0|y = 0) = 1 - \phi_{j|y=0}$$

So we can only parameterize our model using three parameters:
$phi_{j|y=1}$, $\phi_{j|y=0}$, ϕ_y
Now we can compute the joint likelihood of data is :

$$L(\phi_y, \phi_{j|y=0}, \phi_{j|y=1}) = \prod_{i=1}^{n} p(x^{(i)}, y^{(i)}) \qquad (51)$$

Maximize it with respect to ϕ_y, $\phi_{j|y=0}$ and $\phi_{j|y=1}$, then we get:

$$\phi_{j|y=1} = \frac{\sum_{i=1}^{n} 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^{n} 1\{y^{(i)} = 1\}} \qquad (52)$$

$$\phi_{j|y=1} = \frac{\sum_{i=0}^{n} 1\{x_j^{(i)} = 0 \wedge y^{(i)} = 1\}}{\sum_{i=1}^{n} 1\{y^{(i)} = 0\}} \qquad (53)$$

$$\phi_y = \frac{1\{y^{(i)} = 1\}}{n} \qquad (54)$$

Now when I want to implement Bernoulli Naive Bayesian Classifier, I first need to generate my discrete features again: Because in Bernoulli Naive Bayesian Classifier, the numbers should be binary. So I make a conversion rule that if a feature value is less than or equal to the median, it should be converted to 0; if a feature value is larger than the median, it should be converted to 1. In this model, we will consider following indicator types: RSI, CCI, ATR, ADL1, ADL2 and Lagged Return, totally 33 features.

Figure 69 is the AUC curve of Bernoulli Naive Bayesian; Confusion Matrix II is the confusion matrix for Bernoulli Naive Bayesian; Its 2-folds cross validation score is 50.2% and 5-folds cross validation score is 43.87%. So we still cannot have a robust model for market prediction.

3.3.2 Implementation of Multinomial Naive Bayes Classifier

If we regard Bernoulli distribution as a subclass of multinomial distribution, then we can also regard Bernoulli Naive Bayesian classifier as the subclass of multinomial naive Bayesian classifier: Multinomial Naive Bayesian Classifier works on multinomially distributed data. When we use this Naive Bayesian Classifier, we need to discretize our raw continuous features into discrete signals, and fortunately the features we engineered at the end of the first chapter can be directly used in this model, because those features are indeed following multinomial distributions.(Remember to make all the features be non-negative: $x := x + abs(min(X))$). Figure 71 is the AUC of this Multinomial Naive Bayesian Classifier. Confusion matrix III is the confusion matrix of this model. The 2-folds cross validation score is 50.04% and 5-folds cross validation score is 49.80%. So though we make our features more complex and detailed, multinomial naive Bayesian classifier still performs as poor as Bernoulli naive Bayesian classifier.

Confusion Matrix III	Predicted Positive	Predicted Negative
Actual Positive	264	344
Actual Negative	309	384

Chpater IV.Support Vector Machine with Its Application in Stock Price Prediction

3.1 Quick Review on Support Vector Machine

SVM is one of the most famous supervised learning algorithm as a binary classifier. Following is a quick review on SVM basic technique: Assume we have a function $h_{\omega,b}(x)$:

$$h_{\omega,b}(x) = g(\omega^T x + b) \qquad (55)$$

where $g(z) = 1$ if $z \geq 0$, $g(z) = -1$ if $z \leq 0$

So we can say that the function $h_{\omega,b}(x)$ can directly output a binary classification result based on our input. Then we define functional margin for training example (x, y):

$$\gamma = y(w^T x + b) \qquad (56)$$

Where $y = \pm 1$, it represents the real output. $w^T x + b$ represents will be negative or positive, which represents the position of the predicted point, it may be on one side of the hyperplane.Usually a large functional margin represents a confident and a correct prediction. But this margin notation has a big drawback that we can change it by only scaling the ω and b at the same time, for example from $(\omega, \underline{)}$ to $(2\omega, 2\underline{)}$. As we can see, though the scale of functional margin γ changes, but anything else keeps unchanged, this scaling operation is totally meaningless. So we need do some normalization as following:

$$\omega := \frac{\omega}{||\omega||} \qquad (57.1)$$

and

$$b := \frac{b}{||b||} \qquad (57.2)$$

Now we assume that here is a point M on the one side of hyperplane, with the coordinates vector x; and it is vertical to the hyperplane with the cross point N. Because ω is exactly the vector which is perpendicular to the hyperplane, so we can write the point B as :

$$x - \gamma \frac{\omega}{||\omega||} \qquad (58)$$

where γ is the distance between point A to the hyperplane, because the point B is perpendicular to A on the hyperplane

$$w^T x + b = 0$$

, then we can have

$$w^T (x - \gamma \frac{\omega}{||\omega||}) + b = 0 \qquad (59)$$

Then:

$$\gamma = (\frac{\omega}{||\omega||})^T x + \frac{b}{||\omega||} \qquad (60)$$

At last, we have our Geometric Functional Margin:

$$\gamma = y\left(\left(\frac{\omega}{||\omega||}\right)^T x + \frac{b}{||\omega||}\right) \qquad (61)$$

If our input is a dataset with many pairs of input and output $\{(x^{(i)}, y^{(i)})\ i = 1, .., m$, we define the geometric margin of our training examples as:

$$\gamma = \min_{i=1,...,m} \gamma^{(i)} \qquad (62)$$

Now with (62) we clearly know what we want is to find a a decision boundary, or hyperplane, to maximize the margin, and we can represent this maximization problem as:

$$\max_{\gamma,\omega,b} \gamma$$
$$s.t. \quad y^{(i)}(\omega^T x^{(i)} + b) \geq \gamma, i = 1, ..., m \qquad (63)$$
$$||\omega|| = 1$$

We indeed have some techniques to solve this optimization problem, such as transformation and Lagrange Duality, but we do not need to do here. For more detailed steps, please reference: http://cs229.stanford.edu/nc notes3.pdf.

One another important term in data science, especially in the SVM, is feature mapping, which is usually mentioned with one another term, the kernel. Here I will have a brief review on feature mapping; For kernel, we can just regard it as a convenient approach for us to complete our feature mapping. Actually we have met feature mapping many times before, some times we can just regard it as feature engineering. I hope following examples can help you have a review on it:

(1)Price to Return:

$$\text{Price} = \begin{bmatrix} \text{Open} \\ \text{Close} \\ \text{High} \\ \text{Low} \end{bmatrix} \rightarrow \begin{bmatrix} \text{Lagged Return} \\ \text{EWMA Return} \\ \text{Quadractic of Return} \\ \text{ATR} \end{bmatrix}$$

(2)Technical Indicator to Signal:

$$\text{Tech Indicator} = \begin{bmatrix} \text{RSI} \\ \text{MACD} \\ \text{ATR} \\ \text{CCI} \end{bmatrix} \rightarrow \begin{bmatrix} \text{RSI Signal} \\ \text{CCI Signal} \\ \text{ATR Signal} \\ \text{MACD Signal} \end{bmatrix}$$

So in the example(1), we regard the price as raw data, and we map them into a new feature space by some basic operations. In the example(2), we map technical as raw data into a new feature space. So intuitively we can just regard feature mapping just as feature engineering.

And the kernel method is just one particular feature mapping technique that maps raw data into higher-order polynomial feature space, and with kernel method, the computation complexity can be significantly reduced.

4.2 Linear Support Vector Machine

For a data $(x^{(i)}, y^{(i)})$ whose features $x^{(i)}$ are 2-dimensional and can be separated by a straight line, or whose features $x^{(i)}$ are 3-dimensional and can be separated by a plane, then we can say that the data is linearly separable; The linear support vector machine technique tries to model this kind of data.

4.2.1 Implementation on Discrete Signal

Linear support vector machine can work on discrete variables. Here we do not map our features into a new space, or in other word, because I think the original discrete signals have enough economic meaning, more wrapping on the data is redundant. At the same time, I remove MACD signals from features. Confusion matrix IV is for this model. According to k-folds cross validation, Linear SVM performs relatively good compared to any other model displayed before. Its performance is in the table Linear SVM Discrete above. As we can see, the confusion matrix table validation is a once-validation technique, and the result is not usually random; According to table Linear SVM Discrete, we find that as the margin softness increase, which means the model can tolerate some misclassification in the training set, our model performs better; And when the margin softness is high to 1000, the model's performance converges to be stable.

4.2.2 Implementation on Continuous Signal

When we work on raw continuous features, we find that (1) the model performs worse than the model with discrete signals; (2) the model accuracy cannot be improved a little by removing lagged return features. The Confusion Matrix V and Confusion Matrix VI are for the model with and without lagged return features with softness = 1e3; The table Linear SVM Continuous(1) is for the model with lagged return features, and the Linear SVM(2) is for the model without return features. So we can conclude that the raw features can not provide any useful information which can be directly used by Linear SVM; Compared to this model, the discrete signals may indeed catch some effective factors influencing the market.

4.2.3 Implementation on PCA Features

Now we work on PCA features. The Confusion Matrix VII and Linear SVM PCA include validation result of this model. As we can see again, the model on PCA features is still a valid one. PCA features are directly from raw features, if the raw features based model is invalid, there is no surprise that PCA based model also performs weak. Because the main purpose of PCA is first the dimension reduction but not improving prediction accuracy.

4.3 Polynomial Kernel Support Vector Machine

Apart from linear kernel SVM, there are many other popular kernel method used in SVM models, such as polynomial kernel,radial basis function kernel and sigmoid kernel.Briefly, polynomial kernel method map low-order features into higher order features; radial basis function method introduces the concept of squared Euclidean distance to map features; Sigmoid kernel methods transform raw features into binary features.

Because these 3 kernel methods perform almost the same in this project, I only display the result of polynomial kernel method SVM in this report.

4.3.1 Implementation on Discrete Features

The Confusion Matrix VIII is for the polynomial SVM with discrete signal features, its softness = 1e3. According to confusion matrix, our model is still not valid; When we come to the table Poly SVM PCA, we find that our model may be relatively effective, because when folder number is 5, the accuracy ratios of models with softness 1e3 and 1e6 are both a little larger than the random model.

4.3.2 Implementation on Continuous Features

Confusion Matrix IX is for this model; Poly SVM Continuous is the table for this model. As we can see,this model is not good, and it may have a strong over-fitting problem, which we can usually meet in polynomial kernel cases.

4.3.3 Implementation on PCA Features

Confusion Matrix X is for this model; Poly SVM PCA is the table for this model. We can conclude that PCA based model again is not an effective one; There is no surprise, because the model based on raw continuous features is not effective, too. And as fold number increases, the accuracy ratio decreases.

Confusion Matrix IV	Predicted Positive	Predicted Negative
Actual Positive	370	238
Actual Negative	425	268

Linear SVM Discrete	Folders = 2	Folders = 3	Folders = 5
Margin Softness = 1	49.17%	49.54%	48.10%
Margin Softness = 1e3	51.24%	50.7835%	51.85%
Margin Softness = 1e6	51.24%	50.46%	52.08%

Confusion Matrix V	Predicted Positive	Predicted Negative
Actual Positive	316	292
Actual Negative	402	291

Confusion Matrix VI	Predicted Positive	Predicted Negative
Actual Positive	358	250
Actual Negative	426	267

Linear SVM Continuous(1)	Folders = 2	Folders = 3	Folders = 5
Margin Softness = 1	50.28%	44.88%	45.89%
Margin Softness = 1e3	48.84%	47.14%	49.53%
Margin Softness = 1e6	48.84%	47.187%	49.582%

Linear SVM Continuous(2)	Folders = 2	Folders = 3	Folders = 5
Margin Softness = 1	49.49%	44.37%	45.29%
Margin Softness = 1e3	50.07%	44.00%	49.03%
Margin Softness = 1e6	49.17%	50.87%	50.03%

Confusion Matrix VII	Predicted Positive	Predicted Negative
Actual Positive	173	435
Actual Negative	176	517

Linear SVM PCA	Folders = 2	Folders = 3	Folders = 5
Margin Softness = 1	50.74%	44.00%	45.29%
Margin Softness = 1e3	49.03%	50.78%	50.18%
Margin Softness = 1e6	49.12%	50.88%	50.00%

Confusion Matrix VIII	Predicted Positive	Predicted Negative
Actual Positive	193	451
Actual Negative	199	458

Poly SVM Discrete	Folders = 2	Folders = 3	Folders = 5
Margin Softness = 1	49.67%	50.14%	49.07%
Margin Softness = 1e3	51.66%	48.99%	53.09%
Margin Softness = 1e6	51.52%	48.57%	52.95%

Confusion Matrix IX	Predicted Positive	Predicted Negative
Actual Positive	248	396
Actual Negative	248	409

Poly SVM Continuous	Folders = 2	Folders = 3	Folders = 5
Margin Softness = 1	50.27%	44.88%	45.89%
Margin Softness = 1e3	48.84%	47.14%	49.54%
Margin Softness = 1e6	48.84%	47.187%	49.58%

Poly SVM PCA	Folders = 2	Folders = 3	Folders = 5
Margin Softness = 1	50.23%	47.70%	48.02%
Margin Softness = 1e3	50.19%	48.94%	47.93%
Margin Softness = 1e6	49.95%	48.99%	47.88%

Confusion Matrix X	Predicted Positive	Predicted Negative
Actual Positive	274	354
Actual Negative	305	368

Chapter V.Decision Tree with Its Application in Stock Price Prediction

5.1 Quick Review on Decision Tree

Usually in the classification problem, we have 2 or more features to describe an object, and we can predict what the object is by using SVM, Logit Regression, Naive Bayesian Classifier and other classifiers. Decision Tree is also a classifier which tries to help us do classification. Intuitively speaking, Decision Tree Classifier divides a input space into sub parts by lines(for 2-dimensional feature space) or hyperplane (for multi-dimensional feature space) iteratively in the form of flow chart until the output can be predicted or the recursive depth is met.

The strict mathematical steps to divide the feature space is briefly displayed: Assume we have a input space χ, and we divide χ into multiple disjoint regions R_i:

$$\chi = \bigcup_{i=0}^{n} R_i \qquad (64)$$

,

$$\text{Where } R_i \bigcap R_j = \emptyset \text{ if} i \neq j$$

Then our decision tree start a top-down, recursive, greedy partition. The steps are:(1) Split χ into 2 child subregions by thresholding on a single variable; (2)Split subregions into sub-subregions on one another feature (3)Split until a stop criterion is met (4) Repeat steps above. The mathematical notation of dividing a parent region R_p into 2 subregions is:

$$R_1 = \{X | X_j < t, X \in R_p\} \qquad (64.1)$$

$$R_2 = \{X | X_j \geq t, X \in R_p\} \qquad (64.2)$$

Where j is the feature index, $t \in R$ is a threshold.

Usually if we want to have a good split, we need to define a loss function, and in this model, entropy, especially the cross entropy will also be used; I will not talk about it more. Decision can work on both categorical variables and continuous variables. And some times we may want to set minimum leaf size, maximum depth or maximum number of nodes as stopping criteria.

5.2 Decision Tree Classifier Implementation

5.2.1 Implementation on Discrete Signal

Confusion Matrix XI	Predicted Positive	Predicted Negative
Actual Positive	301	328
Actual Negative	364	308

Confusion Matrix XII	Predicted Positive	Predicted Negative
Actual Positive	301	328
Actual Negative	364	308

Confusion Matrix XIII	Predicted Positive	Predicted Negative
Actual Positive	272	357
Actual Negative	266	406

Decision Tree Discrete	Folders = 2	Folders = 3	Folders = 5
Max Depth = 3	51.47%	50.37%	48.98%
Max Depth = 5	50.74%	50.83%	47.65%
Max Depth = 7	52.30%	48.99%	48.20%

Confusion Matrix XIV	Predicted Positive	Predicted Negative
Actual Positive	343	282
Actual Negative	330	346

Confusion Matrix XV	Predicted Positive	Predicted Negative
Actual Positive	280	347
Actual Negative	335	339

Confusion Matrix XVI	Predicted Positive	Predicted Negative
Actual Positive	504	126
Actual Negative	523	148

Decision Tree Continuous	Folders = 2	Folders = 3	Folders = 5
Max Depth = 3	49.03%	50.32%	46.%
Max Depth = 5	49.17%	50.74%	44.23%
Max Depth = 7	50.04%	49.03%	45.84%

We predefined the maximum depth to be 3, 5 and 10. Confusion Matrix XI, XII, XIII correspond to the different maximum depth; Table Decision Tree Discrete is for this model. As we can see, when the maximum depth is larger than 3, the accuracy ratio from confusion matrix is nearly 52%. And according to the Decision Tree Discrete Table, we find that when the folder number is equal to 2, all three models with the different maximum depth performs good. One thing we need to notice is that when we give the model the different random state number, our model will return a different classification result. It is because that the Decision Tree Classifier's underlying algorithm cannot give an convex function where we can find a globally optimal point, so with the different starting points and feature selections, we usually get a different output. This problem will be solved in the next part of this chapter.

5.2.2 Implementation on Continuous Signal

One of the strength of the decision tree classifier is that it can work on both discrete features and continuous features. We predefined the maximum depth to be 3, 5 and 10 again. Confusion Matrix XIV, XV, XVI correspond to those different cases ; The table Decision Tree Continuous is for this model. We can conclude that this kind of models perform worse than those with discrete features. It is nearly random. And same as the model with discrete features, this model will generate different results with different random state.

5.3 Ensemble Methods

Confusion Matrix XVII	Predicted Positive	Predicted Negative
Actual Positive	270	277
Actual Negative	375	379

Confusion Matrix XVIII	Predicted Positive	Predicted Negative
Actual Positive	269	270
Actual Negative	376	386

Confusion Matrix XIX	Predicted Positive	Predicted Negative
Actual Positive	300	300
Actual Negative	345	354

Tree Ada Discrete	Folders = 2	Folders = 3	Folders = 5
Max Depth = 3	52.12%	49.49%	48.99%
Max Depth = 5	50.60%	48.38%	48.53%
Max Depth = 7	49.86%	47.83%	49.35%

5.3.1 Quick Review on Ensemble Methods and Ada-booster

Ensemble Learning is a technique which combine many weak models to generate a relative strong model. With ensemble learning technique, a model's bias and variance can be somehow reduced. There are 2 major types of ensemble learning, one is Bagging and another one is boosting. The difference between these 2 methods is in which do they train the individual base models. When we use bagging methods, a dataset will be divided into multiple datasets, and our base models will be trained on these datasets in a parallel way, finally we congregate our results together; When we use boosting method, we train our model sequentially, the previous model's result will directly influence the result of the later model.

The most famous bagging technique we use to strengthen our decision tree classifier is random forest model. As other bagging techniques do, random forest also need dataset to be divided into multiple datasets, for example n; Then we will train n sub decision tree classifiers in those datasets; At last all the models will vote, and the majority vote will be the final prediction.

The most famous boosting technique we use to strengthen our decision tree classifier is Ada-boost technique. Briefly, Ada-boost learns from the previous model's misclassification by increasing the weight of misclassified data points. The general steps are (1)For n data points, we assign each data point with a initial weight $\frac{1}{n}$. (2)Train the first decision tree. (3) Calculate the tree's weighted error rate. $errorrate = \frac{\text{wrong prediction}}{\text{total prediction}}$. Usually the weight, the more the error will be weighted (4) Calculate weight of decision tree: $\omega_i = \alpha * log\frac{1-e}{e}$. A tree's decision power will be stronger if its weighted error rate is small. α is the learning rate constant (5) Update weight of misclassified data points: If correctly classified, weight keeps the same; If misclassified $\omega_{point} := \omega_{point} \cdot exp(\omega_{tree})$ (6) Repeat steps above until all the trees are iterated.

In this report we will work on decision tree model with Ada-boosting Technique.

5.3.2 Implementation on Discrete Features

We predefined the tree numbers to be 17; Again We predefined the maximum depth to be 3, 5 and 10; Confusion Matrix XVII, XVIII, XIX are corresponding confusion matrices for these different maximum depth. Table Tree Ada Discrete displays the cross validation score's relationship with fold number and max depth. According to the confusion matrices, we cannot conclude that the model is effective, because almost all of them are random; According to the table Tree Ada Discrete, we can find that when fold number is 2 and maximum depth is 3, our cross validation score is high to 52.12%. Another thing we need to notice is that compared to the model without ada-booster, this model's cross validation scores are more stable with less variance, so we can say that though ada-boosted decision tree is not a good prediction tool, but it can be a good tool to ensemble weak models.

5.3.3 Implementation on Continuous Features

Confusion Matrix XX	Predicted Positive	Predicted Negative
Actual Positive	292	283
Actual Negative	353	373

Confusion Matrix XXI	Predicted Positive	Predicted Negative
Actual Positive	307	345
Actual Negative	320	329

Confusion Matrix XXII	Predicted Positive	Predicted Negative
Actual Positive	316	283
Actual Negative	335	367

Tree Ada Continuous	Folders = 2	Folders = 3	Folders = 5
Max Depth = 3	49.72%	47.87%	46.72%
Max Depth = 5	50.23%	48.66%	47.55%
Max Depth = 7	49.21%	48.10%	46.86%

Now let us turn to the decision tree with ada-booster's implementation on continuous features. As we did in before, Confusion Matrix XX, XXI, XXII are confusion matrices corresponding to maximum depth 3,5,7. Table Tree Ada Discrete displays the cross validation score's relationship with fold number and max depth. Again, comparing this model with the model without ada-booster, we find that the prediction accuracy is still random, but the cross validation score becomes more stable with lower variance.

All in all, in this chapter, we have 3 important conclusion (1)Neither the vanilla decision tree nor the tree with the ada-boost technique can give us a stable prediction accuracy (2)Ada-booster cannot improve the prediction accuracy, but it can make the prediction more stable (3)In either the vanilla decision tree or the tree with the ada-boost technique, models with discrete features performs better than those with continuous features; Especially when max-depth = 3 and cross validation fold number ≤ 3, models with discrete features may indeed have prediction ability.

Chapter VI.Multinomial Prediction of Stock Price Movement

We have successfully implemented binary prediction of stock price, and in this chapter, I will implement multinomial classification. Before all, we need to categorize continuous return into discrete variables.

$$\text{return} \in (-\infty, -2 \cdot STD) \to -3$$

$$\text{Return} \in (-2 \cdot STD, -1 \cdot STD) \to -2$$

$$\text{Return} \in (-1 \cdot STD, -0.1 \cdot STD) \to -1$$

$$\text{Return} \in (-0.1 \cdot STD, 0.1 \cdot STD) \to 0$$

$$\text{Return} \in (0.1 \cdot STD, 1 \cdot STD) \to 1$$

$$\text{Return} \in (0.1 \cdot STD, 2 \cdot STD) \to 2$$

$$\text{Return} \in (2 \cdot STD, \infty) \to 3$$

6.1 Softmax Regression

Softmax Regression can be regarded as an generalized form of logit regression who extends binary output to be multinomial output. Usually when we want to derive logit regression or softmax regression, we can always derive them from generalized exponential family.

6.1.1 Implementation of Discrete Signals

Actually, when we backtest our time series predictive model, we should not usually concentrate on its cross validation result, because time series is a very particular type of data. Here, we use one another backtesting method: I use softmax regression model to predict the price movement direction and the change size, and then I multiply this prediction value by our estimated volatility for the next day. Here in this report, I use Lagged Volatility 1, Lagged Volatility 2, Lagged Volatility 3, EWMA Volatility 6, EWMA Volatility 9, Lagged Volatility 13 as our volatility estimation. I divided dataset in 2 parts with equal part, and the figure 71 and figure 72 display how well our model plays: It was very different from our empirical research that our model with vanilla lagged return surprisingly better fit the future movement of the stock price. And the EWMA Volatilitis all perform worse than vanilla lagged volatility. And as the window length increases, the EWMA volatility performs worse.

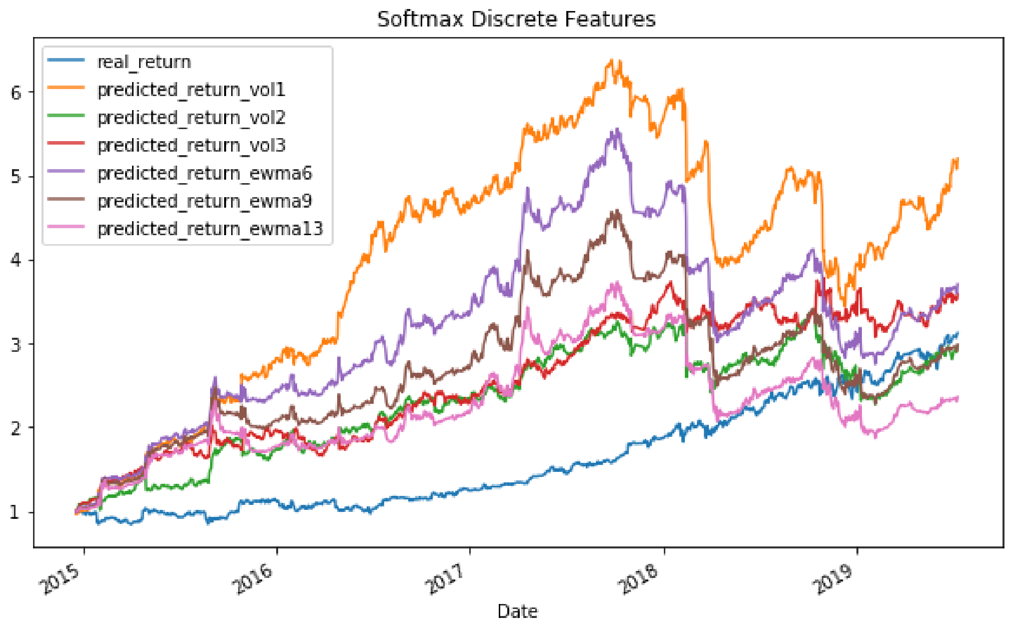
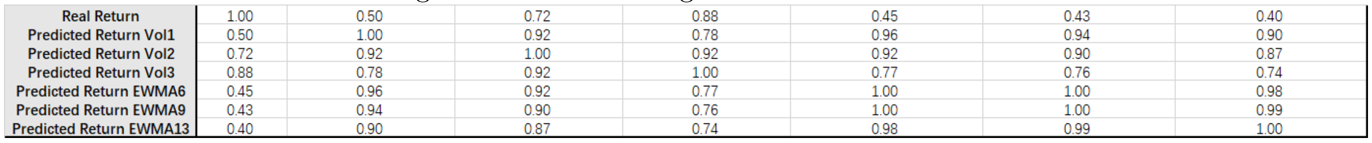


Figure 71: Softmax Regression Predicted NAV



Real Return	1.00	0.50	0.72	0.88	0.45	0.43	0.40
Predicted Return Vol1	0.50	1.00	0.92	0.78	0.96	0.94	0.90
Predicted Return Vol2	0.72	0.92	1.00	0.92	0.92	0.90	0.87
Predicted Return Vol3	0.88	0.78	0.92	1.00	0.77	0.76	0.74
Predicted Return EWMA6	0.45	0.96	0.92	0.77	1.00	1.00	0.98
Predicted Return EWMA9	0.43	0.94	0.90	0.76	1.00	1.00	0.99
Predicted Return EWMA13	0.40	0.90	0.87	0.74	0.98	0.99	1.00

Figure 72: Correlation Matrix for Softmax Regression:Discrete

Now we can say that though our model may perform bad in naive prediction of price movement prediction, but when it is combined with the prediction of the volatility and stock price path, our model again becomes relatively strong. Our model can catch the major tendency of stock price when it is combined with Lagged Volatility 3.

6.1.2 Implementation of Continuous Signals

Again we do the anything as we did in the last part but only replace the discrete signals with continuous features. Our model results are displayed in figure 73 and figure 74.

Again we find that our model performs worse when we use continuous features. First, according to plot, almost all the curves become messy and irregular. Second, according to correlation matrix in the figure 72, the curves predicted by Lagged Volatility 1 and EWMA Volatilities all become useless. But surprisingly we find that the volatility 3 still performs very strong. It again run against our old experience.

6.2 Linear SVM

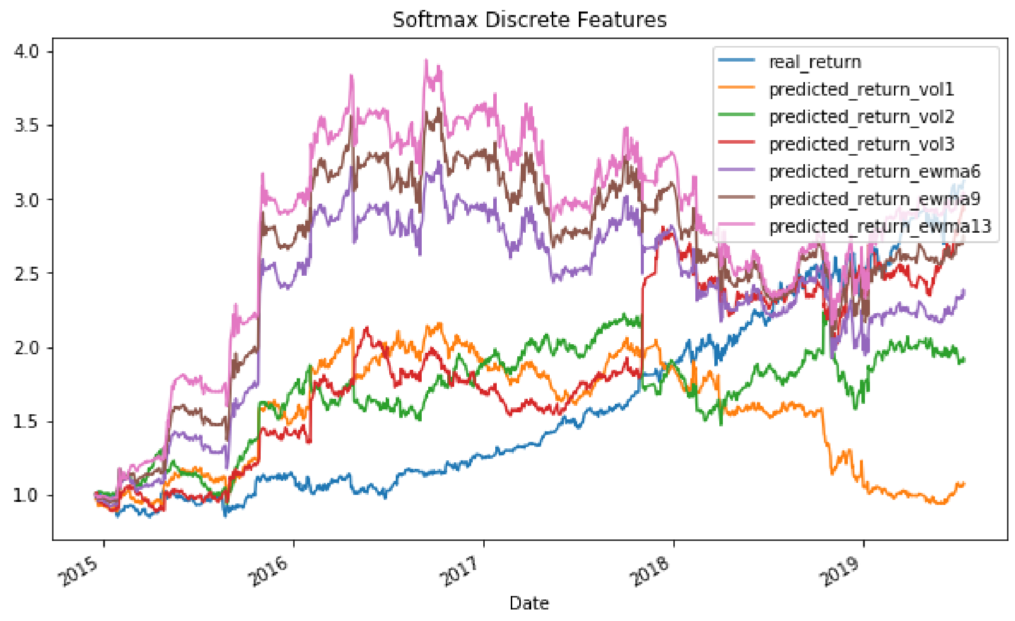
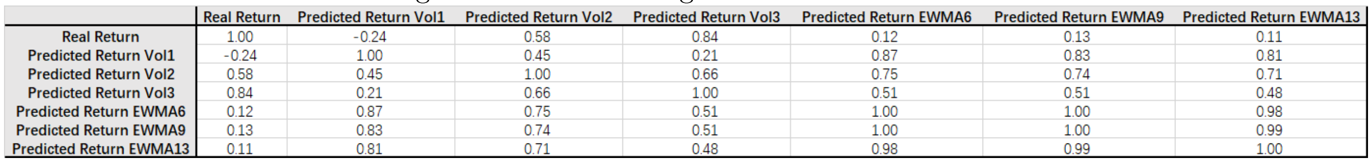


Figure 73: Softmax Regression Predicted NAV



	Real Return	Predicted Return Vol1	Predicted Return Vol2	Predicted Return Vol3	Predicted Return EWMA6	Predicted Return EWMA9	Predicted Return EWMA13
Real Return	1.00	-0.24	0.58	0.84	0.12	0.13	0.11
Predicted Return Vol1	-0.24	1.00	0.45	0.21	0.87	0.83	0.81
Predicted Return Vol2	0.58	0.45	1.00	0.66	0.75	0.74	0.71
Predicted Return Vol3	0.84	0.21	0.66	1.00	0.51	0.51	0.48
Predicted Return EWMA6	0.12	0.87	0.75	0.51	1.00	1.00	0.98
Predicted Return EWMA9	0.13	0.83	0.74	0.51	1.00	1.00	0.99
Predicted Return EWMA13	0.11	0.81	0.71	0.48	0.98	0.99	1.00

Figure 74: Correlation Matrix for Softmax Regression:Discrete

6.2.1 Implementation on Discrete Signals

Now let us use the same logic to make prediction using linear SVM. SVM cannot only handle discrete features but can also handle the multinomial prediction, or categorical prediction. The figure 75 and figure 76 display the performance of SVM model.

Again we amazingly find that the overall tendency of stock price is well predicted. Compared to softmax regression, all the volatility's predictive power are improved. Another thing we still need to notice is that Lagged Volatility 1 still performs bad and the Lagged Volatility 3 still performs very good. (Because of the problem of the page space, figure are attached at last page of the report)

6.2.1 Implementation on Continuous Signals

Things again change when we use continuous features to fit linear SVM. As you can see in the figure 77 and figure 78, it is very amazing that continuous features works here! Most of predictive curve close follow the tendency of the real stock price curve. Besides, we can find that this time Lagged Volatility 1 and Lagged Volatility 2 perform good than Lagged Volatility 3; And All other EWMA volatilities all perform strongly in this case. So we can conclude that for the different type of model, the features which performs weak in one model may be a strong feature in one another model. (Because of the problem of the page space, figure are attached at last page of the report)

Summary

1.In this report I developed multiple machine learning models and most of them are not effective enough to predict one-day stock price movement; 2.When predicting one-day stock price movement, the discrete signals generated by RSI, MACD, ATR and other technical indicators performs better than raw continuous features. 3. When predicting the overall stock price movement tendency, I used softmax regression and linear SVM, and the result is very robust. Lagged Return 3 is a very interesting feature which we may need to study further; And under different model, the effectiveness of EWMA volatility is different.

References

[1] Andrew Ng, Stanford University CS229 Notes
[2] Paul Wilmott, Machine Learning: An Applied Mathematics Introduction
[3] Mehryar Mohri, Foundations of Machine Learning
[2] ET Jaynes, Probability Theory: The Logic of Science
[3]Wikepedia Cross Validation (Statistics) https://en.wikipedia.org/wiki/Cross-validation$_($*statistics*)
[4]Springer, Encyclopedia of System Biology
[5]Data Driven Investor
[6]Max Kuhn Kjell Johnson, Applied Predictive Modeling
[7]James Douglas Hamilton , Time Series Analysis
[8]Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow
[9] Website: https://towardsdatascience.com/
[10]Website: https://medium.com/machine-learning-101/
[11]Website: https://scikit-learn.org/

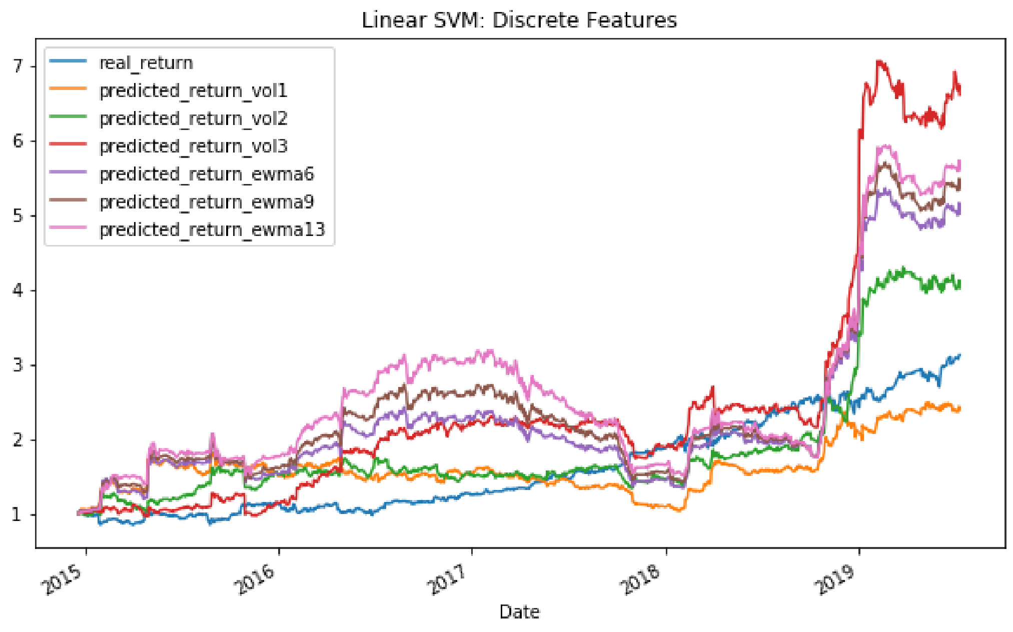


Figure 75: Linear SVM Predicted NAV

	Real Return	Predicted Return Vol1	Predicted Return Vol2	Predicted Return Vol3	Predicted Return EWMA6	Predicted Return EWMA9	Predicted Return EWMA13
Real Return	1.00	0.21	0.92	0.95	0.83	0.77	0.67
Predicted Return Vol1	0.21	1.00	0.19	0.18	0.41	0.38	0.36
Predicted Return Vol2	0.92	0.19	1.00	0.97	0.91	0.87	0.80
Predicted Return Vol3	0.95	0.18	0.97	1.00	0.91	0.88	0.80
Predicted Return EWMA6	0.83	0.41	0.91	0.91	1.00	0.99	0.96
Predicted Return EWMA9	0.77	0.38	0.87	0.88	0.99	1.00	0.99
Predicted Return EWMA13	0.67	0.36	0.80	0.80	0.96	0.99	1.00

Figure 76: Correlation Matrix for Linear SVM:Discrete

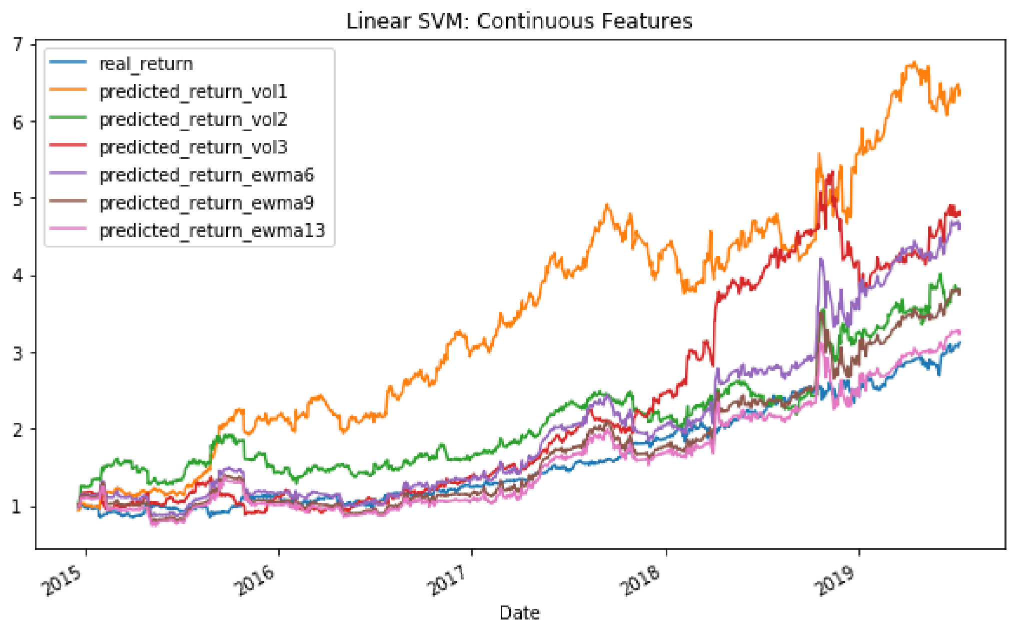


Figure 77: Linear SVM Predicted NAV

	Real Return	Predicted Return Vol1	Predicted Return Vol2	Predicted Return Vol3	Predicted Return EWMA6	Predicted Return EWMA9	Predicted Return EWMA13
Real Return	1.00	0.94	0.96	0.77	0.92	0.90	0.88
Predicted Return Vol1	0.94	1.00	0.93	0.76	0.98	0.96	0.94
Predicted Return Vol2	0.96	0.93	1.00	0.64	0.90	0.88	0.85
Predicted Return Vol3	0.77	0.76	0.64	1.00	0.80	0.80	0.80
Predicted Return EWMA6	0.92	0.98	0.90	0.80	1.00	1.00	0.99
Predicted Return EWMA9	0.90	0.96	0.88	0.80	1.00	1.00	1.00
Predicted Return EWMA13	0.88	0.94	0.85	0.80	0.99	1.00	1.00

Figure 78: Correlation Matrix for Linear SVM:Discrete