Fundamentals of Model-based Security Screener

Xugang Ye

Abstract

We present some latest development on the data-driven and model-based security recommendation system. We aim to forecast both index funds and individual stocks. For both ETFs and stocks, the goal is to generate a model-based algorithmic screener. Our methodology is different from existing empirically rule-based screening systems. Our screener is the result of the big data integration and model prediction.

1. Introduction

With the application of the Hadoop distributed file system (HDFS) and MapReduce framework, it becomes more convenient to model very large data of financial markets and parallelly generate signals and predictions for individual ETFs/stocks. In practice, the prevalent way of transforming multi-channel financial data into security recommendation list is to give each key financial variable a threshold value and use multiple variables to select potentially good/bad securities from pool. Although this method is easy to implement, it may encounter some issues. One is that a security’s price movement does not necessarily positively/negatively correlated with some key financial variables. For example, when an earning reports beats estimates, it’s highly likely that the jump is already built into the price and current outperformance does not necessarily indicate the future outperformance. Although a upward trend of the analysts’ expectation might suggest a long-run bullishness, a counterargument can be that current market sentiment might boost analysts’ expectation to the point beyond a firm can reach so that the long-run bullishness won’t be able to hold. Based on this and much more, our goal is to systematically build mathematical/statistical models to find possible causal relations between the fundamental variables and price movements. Essentially, in a way as simple as possible, we are interested in predicting the chance that a security’s return will exceed a level in a horizon. In technical notations, suppose the security’s price is observed as $s_t$ at time $t$ and in a horizon from time $t$ to $t + h$, there are $k$ sequential prices $s_{t+t_1}, \ldots, s_{t+t_k}$ to be observed. Let $r_{t+t_j} = \frac{s_{t+t_j}}{s_t} - 1$ for $j = 1, \ldots, k$ and denote $r^q_t$ as the $q$-quantile of $r_{t+t_1}, \ldots, r_{t+t_k}$ for some $q > \frac{1}{2}$. We want to estimate the hitting probability $P(r^q_t > r_+)$ for some $r_+ > 0$ based on the information obtained by time $t$. Assume $0 < P(r^q_t > r_+) < 1$ and let $\hat{x}^{(t)}$ be the feature vector obtained by time $t$, our goal is to find the functions $F: R^n \rightarrow (0,1)$ such that $P(r^q_t > r_+) = F(\hat{x}^{(t)})$. Once we have such a function with certain prediction power that is better than random guessing, the securities in the pool can be ordered by their projected likelihoods of outperformance and the historical validation accuracies. This in turn renders us a model-based, algorithmic screener. In the following sections, we introduce our basic predictive models, describe our approach of validation and calibration, illustrate our rank-ordering based portfolio construction, our feature processing tips, and our preliminary results.
2. Predictive Models

2.1. Fitting Label

This method leads to the classification models. The most common one is the logistic regression

\[ F(\hat{x}^{(t)}) = \frac{\exp(\alpha + \beta^T \hat{x}^{(t)})}{1 + \exp(\alpha + \beta^T \hat{x}^{(t)})}. \]  

(1)

To estimate the parameters \( \alpha, \beta \), one needs a training data \( \{\hat{x}^{(t-h-\tau+1)}, y_{t-h-\tau+1}: \tau = 1, \ldots, l\} \), where \( y_{t-h-\tau+1} = I(r_{t-h-\tau+1} > r_+), I(\cdot) \) is indicator function, and \( l \) is the width of the training window.

One way to derive \( \alpha, \beta \) is to maximize the objective function:

\[ H(\alpha, \beta) = \sum_{\tau=1}^{l} (y_{t-h-\tau+1} \cdot \hat{y}_{t-h-\tau+1} + (1 - y_{t-h-\tau+1}) \cdot (1 - \hat{y}_{t-h-\tau+1})), \]  

(2)

where \( \hat{y}_{t-h-\tau+1} = \frac{\exp(\alpha + \beta^T \hat{x}^{(t-h-\tau+1)})}{1 + \exp(\alpha + \beta^T \hat{x}^{(t-h-\tau+1)})} \). This objective function represents the total “true-positive” and “false-negative”, it is concave with respective to \( \alpha, \beta \).

2.2. Fitting Value

The second method is to fit \( r_t^q \) directly. The general model can be expressed as

\[ r_t^q = f(\hat{x}^{(t)}) + \varepsilon_t, \]  

(3)

where \( \varepsilon_t \sim N(0, \sigma_{\varepsilon_t}^2) \) is the uncorrelated Gaussian error. Hence

\[ F(\hat{x}^{(t)}) = \Phi\left(\frac{f(\hat{x}^{(t)}) - r_+}{\sigma_{\varepsilon_t}}\right). \]  

(4)

In the simplest case, \( f \) is linear and the problem is linear regression. The least-square minimization method can be applied to the training data \( \{\hat{x}^{(t-h-\tau+1)}, r_{t-h-\tau+1}^q: \tau = 1, \ldots, l\} \). And \( \sigma_{\varepsilon_t} \) can be estimated from the fitting residuals.

2.3. Fitting Trend

The third method is to assume that the security’s price follows some stochastic process. For example, the geometric Brownian motion (GBM) can be considered. According to GBM, the security’s price is a continuous stochastic process governed by the stochastic differential equation

\[ dS_t = \mu S_t dt + \sigma S_t dz_t, \]  

(5)
where $S_t$ is the price at time $t$, $z_t$ is a standard Wiener process, the constant $\mu$ is the percentage drift and the constant $\sigma$ is the percentage volatility. The GBM has the properties $E \left( \ln \left( \frac{S_{t+\Delta t}}{S_t} \right) \right) = \left( \mu - \frac{1}{2} \sigma^2 \right) \Delta t$ and $Var \left( \ln \left( \frac{S_{t+\Delta t}}{S_t} \right) \right) = \sigma^2 \Delta t$. Hence the moment estimates of $\mu$ and $\sigma$ can be obtained from observed price series.

In our problem domain, we assume a constant percentage drift $\mu_{t:t+h}$ and a constant percentage volatility $\sigma_{t:t+h}$ in the horizon from time $t$ to time $t + h$. $\mu_{t:t+h}$ can be assumed to be explainable by $f(\hat{x}^{(t)})$ and fitting $\mu_{t:t+h}$ using linear function is the simple linear regression problem. For $\sigma_{t:t+h}$, the simplest estimation is to use the observed prices in last horizon to obtain the moment estimate of $\sigma_{t:0}$, and then use it as the estimate of $\sigma_{t:t+h}$. According to (5), we have the discrete equation

$$s_{t+t_j} = \left( 1 + \mu_{t:t+h} t_j + \sigma_{t:t+h} \epsilon_{t_j} \sqrt{t_j} \right) s_t,$$

(6)

where $\epsilon_{t_j}$’s are uncorrelated standard normal random variables. Let the error $e_t$ of fitting $\mu_{t:t+h}$ be the Gaussian $N(0, \sigma_{e_t}^2)$ and $e_t$ is independent of $\epsilon_{t_j}$, then (6) can be written as

$$s_{t+t_j} = \left( 1 + f(\hat{x}^{(t)}) + e_t \right) t_j + \sigma_{t:t+h} \epsilon_{t_j} \sqrt{t_j} s_t.$$

(7)

Note that $\sigma_{e_t}$ could be estimated from the residuals of the historical fitting. By (7), one can simulate the price path $s_{t+t_1}, \ldots, s_{t+t_k}$ to obtain the return path $\tau_{t+t_1}, \ldots, \tau_{t+t_k}$. Suppose $N$ Monte Carlo replicates are generated, then $P(\tau_t^q > r_+)$ can be estimated as $\frac{1}{N} \sum_{i=1}^{N} I \left( (\tau_t^q)^{(i)} > r_+ \right)$, where $(\tau_t^q)^{(i)}$ is the $q$-quantile of the $i$-th path.

3. Validation and Calibration

The validation accuracies at time $t$ can be computed from the observed labels $(y_{t-h-i+1}, \ldots, y_{t-h})$ and the predicted values $(\hat{y}_{t-h-i+1}, \ldots, \hat{y}_{t-h})$. That is

$$H_{t-h-l+1:t-h} = \frac{1}{l} \sum_{i=1}^{l} (y_{t-h-i+1} \cdot \hat{y}_{t-h-i+1} + (1 - y_{t-h-i+1}) \cdot (1 - \hat{y}_{t-h-i+1})),\quad (8)$$

$$p_1^{(t-h-l+1:t-h)} = \frac{\sum_{i=1}^{l} (y_{t-h-i+1} \cdot \hat{y}_{t-h-i+1})}{\sum_{i=1}^{l} y_{t-h-i+1}},$$

(9)

$$p_2^{(t-h-l+1:t-h)} = \frac{\sum_{i=1}^{l} (1 - y_{t-h-i+1}) (1 - \hat{y}_{t-h-i+1})}{\sum_{i=1}^{l} (1 - y_{t-h-i+1})},$$

(10)

where $H_{t-h-l+1:t-h}$ is the total hit rate, $p_1^{(t-h-l+1:t-h)}$ is the hit rate among the “true” and $p_2^{(t-h-l+1:t-h)}$ is the hit rate among the “false”. Let $X_t = I(\text{“predict } r_t^q > r_+ \text{”})$ and $Y_t = I(\text{“true } r_t^q > r_+ \text{”})$. Define $p_1^{(t)} = P(X_t = 1|Y_t = 1)$ and $p_2^{(t)} = P(X_t = 0|Y_t = 0)$. By computing $P(X_t = 1)$ via conditioning on $Y_t$, we have
\( P(X_t = 1) = (p_1^{(t)} + p_2^{(t)} - 1)P(Y_t = 1) + 1 - p_2^{(t)}, \) \hspace{1cm} (11)

which describes the fundamental relation between the subjective probability \( P(X_t = 1) \) and the objective probability \( P(Y_t = 1) \). If \( p_1^{(t)} + p_2^{(t)} \neq 1 \), then solving (11) for \( P(Y_t = 1) \) yields

\[ P(Y_t = 1) = \frac{P(X_t = 1) + p_2^{(t)} - 1}{p_1^{(t)} + p_2^{(t)} - 1}, \] \hspace{1cm} (12)

which is the formula of calibrating \( P(X_t = 1) \). By plugging (9) and (10) into (12), we have a calibrated estimator of \( P(X_t = 1) \) as

\[ \hat{P}(Y_t = 1|l) = \frac{P(X_t = 1) + p_2^{(t-h+l+1:t-h)} - 1}{p_1^{(t-h+l+1:t-h)} + p_2^{(t-h+l+1:t-h)} - 1}. \] \hspace{1cm} (13)

By considering various \( l \), we have an aggregated estimator

\[ \hat{P}(Y_t = 1|l) \propto \sum_l \hat{P}(Y_t = 1|l) \cdot P(l). \] \hspace{1cm} (14)

where \( P(l) \) is the prior on \( l \), a simple choice is \( P(l) = 0 \) if \( p_1^{(t-h-l+1:t+h)} + p_2^{(t-h-l+1:t+h)} \) is close to 1; \( P(l) = 1 \) otherwise. In general, the prior can be trained from history in a Bayesian way.

It should be noted that the quantity \( p_1^{(t)} + p_2^{(t)} \) measures whether the prediction is in the same direction with the truth or not. \( p_1^{(t)} + p_2^{(t)} > 1 \) implies same direction; \( p_1^{(t)} + p_2^{(t)} < 1 \) implies opposite direction; and \( p_1^{(t)} + p_2^{(t)} = 1 \) means randomness.

4. Rank ordering screener

To illustrate how a rank ordering screener works, suppose at a time point, the true desired probability for a security \( i \) is \( \pi_i \) and its estimate is \( \hat{\pi}_i \). Let \( G_i \) be the gain when the security’s return exceeds target and \( L_i \) be the loss when the target is missed. A typical example is call option. When the security’s price is above the strike price, there is a gain that can be realized. If the security’s price falls below the strike price while expiration, there would be a cost of the option’s premium. The expected net gain is therefore

\[ g_i = G_i\pi_i\hat{\pi}_i - L_i(1 - \pi_i)\hat{\pi}_i. \] \hspace{1cm} (15)

Assume that the difference between \( \hat{\pi}_i \) and \( \pi_i \) is random and Gaussian, i.e. \( \hat{\pi}_i = \pi_i + e_i \), \( e_i \sim N(0, \sigma_i^2) \). Then

\[ g_i = (G_i + L_i)\pi_i^2 - L_i\pi_i + [(G_i + L_i)\pi_i - L_i]e_i. \] \hspace{1cm} (16)

Taking expectation over \( e_i \) yields

\[ E_{e_i}(g_i) = (G_i + L_i)\pi_i^2 - L_i\pi_i. \] \hspace{1cm} (17)
In order for \( E_{e_i}(g_i) > 0 \), we must require \( \pi_i > \frac{L_i}{G_i + L_i} \). Note that as \( \pi_i > \frac{L_i}{G_i + L_i} \), \( E_{e_i}(g_i) \) is monotonically increasing with respect to \( \pi_i \). This simply says the larger the value of \( \pi_i \) than \( \frac{L_i}{G_i + L_i} \) the preferable the security \( i \) it is. Taking variance over \( e_i \) yields
\[
Var_{e_i}(g_i) = [(G_i + L_i)\pi_i - L_i]^2\sigma_i^2, \tag{18}
\]
which simply says the risk comes from the uncertainty of the security itself and the error of the prediction model.

Suppose we have selected \( n \) securities \( 1, \ldots, n \), with weights \( w_1, \ldots, w_n \) nonegative and summing up to \( 1 \), then the net gain of the portfolio is \( g = \sum_{i=1}^{n} w_i g_i \). Assume the prediction errors are uncorrelated. Then the expectation and variation respectively are
\[
E(g) = \sum_{i=1}^{n} w_i [(G_i + L_i)\pi_i^2 - L_i\pi_i] \tag{19}
\]
and
\[
Var(g) = \sum_{i=1}^{n} w_i^2 [(G_i + L_i)\pi_i - L_i]^2\sigma_i^2. \tag{20}
\]
The variance shows the diversification of the risk that comes from the uncertainty of the securities’ prices and the errors of the model predictions.

Practically, a portfolio can be constructed by first screening out those securities with prediction accuracies (calculated from validation) above a pre-defined threshold, and then ranking them by the estimates \( \hat{\pi}_i \)'s, and then choose those in a top percentage. The weights in the simplest case are uniform. In general, the weights are found by solving quadratic programming.

### 5. Feature processing

The task of feature processing is to reduce the potential chances of numerical problems and overfitting. The most commonly encountered issues include the range scale, co-linearity, and dimensionality. Empirical studies have shown that directly throwing the values of features and targets into the model training often leads to disastrous results. Hence feature processing is a must. Perhaps the easiest first step is to remove the variables that are either empty or have no variation. If different variables have very different scales of ranges, some numerical troubles will happen later on. It’s recommended that variables should be scaled into the same range. The simplest method of doing so is to divide by the maximum absolute value. To decorrelate the variables, the eigen value decomposition or singular value decomposition of the covariance/correlation matrix can be applied. And the projection matrix thus yielded can be used to reduce the dimensionality of the original variables. The new variables are the linear combinations of the original variables and called principal components. Using principal components also helps reducing the noise. If there are too many original variables, two problems could arise. One is that finding the eigen value decomposition or singular value decomposition of the covariance/correlation matrix can be very costly because the dimension...
of the matrix is too high. The other is that even if the eigen value decomposition or singular value decomposition can be handled for large matrix, it’s still possible that the number of principal components is greater than the number of training data points. A practical way of dealing with these problems is to randomly choose a subset of the original variables at first and then find the principal components of the selected variables. A prediction can be made based on the training that uses thus generated principal components. This can be done many times and the final prediction is the average. This idea is very similar to the well-known method called random forest, which also manipulates the input features. It should be noted that too many input features usually contain a lot of redundant ones.

6. Implementation

The following diagram illustrates the data and work flow in our implementation. The basic idea is the parallel computing supported by the HDFS and MapReduce framework. At the computing level of the reducers, each single node is like the usual single computer. The computing codes (C/C++/Matlab/R/Python) are developed and debugged locally (usually in a very small scale) before being sent to Hadoop run.

![Diagram of data and work flow](image)

Figure 1. Data and work flow
7. Preliminary Results

We present a preliminary result of applying our methodology to a Thomson Reuters raw data set that contains 2963 securities. The target data contains 1500 securities that are selected by market cap. The performance of our preliminary model-based screener is shown in Figure 2.

![Figure 2. Model performance compared with S&P 500, from 09/20/2007 to 12/29/2010](image)

In the back-test, we predict the probability that the 95%-quantile of the security’s prices in next 30 trading days will exceed 120% of the current price using more than three hundred variables empirically created. They are built upon the basic information including prices, volumes, outstanding shares, dividends, earnings, earning estimates, profits, debts, and assets etc. The scale of the target data is $1500 \times 1000 \times 300$. At the computing level of the reducers, we applied logistic regression, linear regression, and geometric Brownian motion. We ensemble models of various parameters settings after validation and calibration. We screen out those securities with validation accuracies less
than 55% and select the top 5% from the rest by the values of target probability prediction. We assume 0.1% transaction fee in a single buy or sell. If a security is bought, it is assumed to be bought once; and if it is sold, it is sold out. When buying multiple securities, we assume that the cash is equally distributed after the commission is paid. Note that if a long position on some day is followed by a long position on the next day, there won’t be a transaction; if a long position is followed by a cash position on the next day, there will be a sell transaction; if a cash position is followed by a long position on the next day, there will be a buy transaction. Also, when both buy and sell happen in same day, we assume the sell transactions happen at first, then the cash after commission payment is equally distributed to purchase the items.

The portfolio construction and management in practice is much more complicated. Although our experimental setting is too simple, the purpose of our demo is however to show the potential of the model-based security screener in assisting portfolio management.

8. Discussion and future direction

It's widely known that the real performance is usually worse than the back-testing if the performance is measured in a relatively long period like a year or more. There could be many reasons. The most agreed one is that market is essentially a place where countless participants compete, bid, and ask. It in itself evolves quickly so that the past patterns do not necessarily exits in the future. When a pattern is detected, it usually has been around for some time already and it may be late for corresponding actions. For this reason, many economists support at least the semi-strong form of the random walk hypothesis that future stock returns cannot be predicted from past returns even together with other publicly available information such as corporate statements and analyst reports. Regarding the back-testing performance, it's possible that the outperformance is due to the over-mining of the historical data. It's very common that model-building won't complete after one shot. It's a process in which there are many trials. Even the validation is a special training because it plays its part in forming the final aggregated model for making prediction. In this sense, the overfitting is inevitable.

Then why we still want to mining the historical data for the purpose of predicting the future returns, at the cost of building complicated data platforms and computing engines? The answer is that it's not self-evident that making predication based on large-scale information integration, rigorous mathematical derivation, and high-performance computing is useless compared with traditional ways of asset management. Moreover, the most important merit of the machine learning is to display to people the unusual digest and visualization of the past information, which have never existed before. Based on this point and maybe more, we recommend two future work. One is to establish a sequential out-of-sample evaluation approach. The approach should be able to adjust the previously used model system when the real-time performance begins deteriorating. The other is to make the model prediction and performance explainable so that any clue that has been caught by model should be immediately available to human experts.
Appendix A. Bayesian Regression and Model Selection

A.1. Bayesian Regression

\( Y_{new} \): new target
\( \tilde{x}^{(new)} \): new feature values
\( D = \{(y_i, \tilde{x}^{(i)}), i = 1, ..., n\} \)
\( \tilde{\theta} \): model parameters

\[
P(Y_{new} | \tilde{x}^{(new)}, D) = \int P(Y_{new} | \tilde{x}^{(new)}, \tilde{\theta}, D) P(\tilde{\theta} | \tilde{x}^{(new)}, D) d\tilde{\theta}
= \int P(Y_{new} | \tilde{x}^{(new)}, \tilde{\theta}) P(\tilde{\theta} | D) d\tilde{\theta}
\propto \int P(Y_{new} | \tilde{x}^{(new)}, \tilde{\theta}) P(\tilde{\theta} | \hat{y}) d\tilde{\theta},
\]

where \( P(\tilde{\theta} | \hat{y}) = \prod_{i=1}^n P(y_i | \tilde{x}^{(i)}, \tilde{\theta}) \) and \( P(\tilde{\theta} | \hat{y}) \) is prior.

Two optimization problems

1) Treat \( \tilde{\theta} \) as a point → maximum likelihood estimation
\( \tilde{\theta}^* = \text{argmax}_{\tilde{\theta}} P(\tilde{\theta} | \hat{y}) = \text{argmax}_{\tilde{\theta}} \sum_{i=1}^n \ln P(y_i | \tilde{x}^{(i)}, \tilde{\theta}) \).
\( P(Y_{new} | \tilde{x}^{(new)}, D) = P(Y_{new} | \tilde{x}^{(new)}, \tilde{\theta}^*) \).

2) Treat \( \tilde{\theta} \) as from a distribution → model averaging
\( \tilde{\theta}^* = \text{argmax}_{\tilde{\theta}} P(\tilde{\theta} | \hat{y}) = \text{argmax}_{\tilde{\theta}} \int P(\tilde{\theta} | \hat{y}) d\tilde{\theta} = \text{argmax}_{\tilde{\theta}} \int (\prod_{i=1}^n P(y_i | \tilde{x}^{(i)}, \tilde{\theta})) P(\tilde{\theta} | \hat{y}) d\tilde{\theta} \).
\( P(Y_{new} | \tilde{x}^{(new)}, D) \propto \int P(Y_{new} | \tilde{x}^{(new)}, \tilde{\theta}) (\prod_{i=1}^n P(y_i | \tilde{x}^{(i)}, \tilde{\theta})) P(\tilde{\theta} | \hat{y}) d\tilde{\theta} \).

A.2. Bayesian Model Selection

\( M_j \): the \( j \)-th model

\[
P(Y_{new} | \tilde{x}^{(new)}, D) = \sum_{j=1}^k P(Y_{new} | \tilde{x}^{(new)}, M_j, D) P(M_j | \tilde{x}^{(new)}, D)
= \sum_{j=1}^k P(Y_{new} | \tilde{x}^{(new)}, M_j) P(M_j | D)
\propto \sum_{j=1}^k P(Y_{new} | \tilde{x}^{(new)}, M_j) P(D | M_j) P(M_j),
\]

where \( P(D | M_j) = \prod_{i=1}^n P(y_i | \tilde{x}^{(i)}, M_j) \) and \( P(M_j) \) is the model prior.

If the data \( D \) is divided into the training part \( D^{(t)} \) that \( M_j \) is trained from and the validation part \( D^{(v)} \), then

\[
P(Y_{new} | \tilde{x}^{(new)}, D)
\]
Appendix B. Simple Methods for Two Class Classification

Features: $\mathbf{x}$.

Label: $y \in \{0, 1\}$.

Task: find $P(Y = 1|\mathbf{x})$ based on a training data $\mathbf{D}^{(0)} = \{(\mathbf{x}^{(1)}, y_1), ..., (\mathbf{x}^{(T)}, y_T)\}$, where $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(T)}$ are the points in feature space and $y_1, ..., y_T$ are observed labels, assuming $0 < \sum_{t=1}^{T} y_t < T$.

Performance is measured by the accuracy on a test data $\mathbf{D}^{(1)} = \{(\mathbf{x}^{(T+1)}, y_{T+1}), ..., (\mathbf{x}^{(T+N)}, y_{T+N})\}$, that is

$$H(\hat{y}_{T+1}, ..., \hat{y}_{T+N}, y_{T+1}, ..., y_{T+N}) = \frac{1}{N} \sum_{t=1}^{N} I(\hat{y}_{T+1} = y_{T+1})$$

(1.1)

1. Logistic Regression

Model:

$$P(Y = 1|\mathbf{x}; \vartheta) = \frac{\exp(\alpha + \beta^T \mathbf{x})}{1 + \exp(\alpha + \beta^T \mathbf{x})}, \quad \vartheta = (\alpha, \beta).$$

(B.2)

Data likelihood:

$$P(\mathbf{D}^{(0)}|\vartheta) = \prod_{t=1}^{T} P(y_t|\mathbf{x}^{(t)}; \vartheta) = \prod_{t=1}^{T} \left[ \frac{\exp(\alpha + \beta^T \mathbf{x}^{(t)})}{1 + \exp(\alpha + \beta^T \mathbf{x}^{(t)})} \right]^{y_t} \left[ \frac{1}{1 + \exp(\alpha + \beta^T \mathbf{x}^{(t)})} \right]^{1-y_t}.$$  \hspace{1cm} (B.3)

Maximum likelihood estimation:

$$\hat{\vartheta}^* = \arg \max_{\alpha, \beta} \prod_{t=1}^{T} \left[ \frac{\exp(\alpha + \beta^T \mathbf{x}^{(t)})}{1 + \exp(\alpha + \beta^T \mathbf{x}^{(t)})} \right]^{y_t} \left[ \frac{1}{1 + \exp(\alpha + \beta^T \mathbf{x}^{(t)})} \right]^{1-y_t}.$$  \hspace{1cm} (B.4)

2. Probit Regression

Model:

$$P(Y = 1|\mathbf{x}; \vartheta) = \Phi(\alpha + \hat{\beta}^T \mathbf{x}), \quad \vartheta = (\alpha, \hat{\beta}).$$

(B.5)

where $\Phi$ is the cumulative distribution of the standard normal. The motivation comes from a latent variable $Z = \alpha' + \hat{\beta}^T \mathbf{x} + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. And $Y$ can be viewed as the indicator for whether $Z > 0$. Note that
which is as same as (B.5) after letting $\alpha = \alpha'/\sigma$ and $\beta = \beta'/\sigma$.

**Data likelihood:**

$$P(D(0) | \tilde{\theta}) = \prod_{t=1}^{T} P(y_t | \bar{x}^{(t)}; \tilde{\theta}) = \prod_{t=1}^{T} [\Phi(\alpha + \beta^T \bar{x}^{(t)})]^{y_t} [1 - \Phi(\alpha + \beta^T \bar{x}^{(t)})]^{1-y_t}. \quad (B.7)$$

**Maximum likelihood estimation:**

$$\tilde{\theta}^* = \arg\max_{\alpha, \beta} \prod_{t=1}^{T} [\Phi(\alpha + \beta^T \bar{x}^{(t)})]^{y_t} [1 - \Phi(\alpha + \beta^T \bar{x}^{(t)})]^{1-y_t}. \quad (B.8)$$

3. **Projection method**

**Projector:** $\bar{w}$

**Projection:** $\bar{w}^T \bar{x}$

**Decision boundary:** $c$

**Model:**

$$P(Y = 1 | \bar{x}; \tilde{\theta}) = \begin{cases} \frac{\sum_{t=1}^{T} I(\bar{w}^T \bar{x}^{(t)} > c) y_t}{\sum_{t=1}^{T} I(\bar{w}^T \bar{x}^{(t)} > c)} & \text{if } \bar{w}^T \bar{x} > c; \\ \frac{\sum_{t=1}^{T} I(\bar{w}^T \bar{x}^{(t)} < c) y_t}{\sum_{t=1}^{T} I(\bar{w}^T \bar{x}^{(t)} < c)} & \text{if } \bar{w}^T \bar{x} < c, \end{cases} \quad (B.9)$$

where $\tilde{\theta} = (c, \bar{w})$.

The two centers of the two classes of feature points are

$$\bar{\mu}^{(1)} = \frac{\sum_{t=1}^{T} y_t \bar{x}^{(t)}}{\sum_{t=1}^{T} y_t}, \quad (B.10)$$

which is the mean of the feature points labeled "1", and

$$\bar{\mu}^{(2)} = \frac{\sum_{t=1}^{T} (1-y_t) \bar{x}^{(t)}}{\sum_{t=1}^{T} (1-y_t)}, \quad (B.11)$$

which is the mean of the feature points labeled "0". Hence the decision boundary can be estimated as $\hat{c} = \bar{w}^T (\bar{\mu}^{(1)} + \bar{\mu}^{(2)})/2$, \quad (B.12)

To infer $\bar{w}$, consider the Fisher's criterion

$$J(\bar{w}) = \left( \frac{\bar{w}^T S_B \bar{w}}{\bar{w}^T S_B \bar{w}} \right), \quad (B.13)$$

where

$$S_B = (\bar{\mu}^{(2)} - \bar{\mu}^{(1)})(\bar{\mu}^{(2)} - \bar{\mu}^{(1)})^T \quad (B.14)$$

is the between-class covariance matrix and
\[ S_W = \sum_{t=1}^{T} \left[ y_t \left( \tilde{x}^{(t)} - \tilde{\mu}^{(1)} \right) \left( \tilde{x}^{(t)} - \tilde{\mu}^{(1)} \right)^T + \left( 1 - y_t \right) \left( \tilde{x}^{(t)} - \tilde{\mu}^{(2)} \right) \left( \tilde{x}^{(t)} - \tilde{\mu}^{(2)} \right)^T \right] \]  
(B.15)

is the within-class covariance matrix.

The maximizer \( \tilde{w}^* \) of \( J(\tilde{w}) \) is the eigen vector corresponding to the largest eigenvalue of the matrix \( S_W^{-1}S_B \). In fact letting the gradient of \( J(\tilde{w}) \) be zero yields \( S_W^{-1}S_B \tilde{w} = J(\tilde{w})\tilde{w} \). Also note that \( S_B \) is symmetric, \( S_W \) is symmetric and positive definite, and they can be diagonalized simultaneously (\( Q^T S_W Q = I, Q^T S_B Q = \Lambda \)). In the simple case of two classes, there is a simpler way to find the eigenvector, that is \( \left( \tilde{\mu}^{(2)} - \tilde{\mu}^{(1)} \right) \left( \tilde{\mu}^{(2)} - \tilde{\mu}^{(1)} \right)^T \tilde{w} \propto S_W \tilde{w} \) yields \( \tilde{w} \propto S_W^{-1} \left( \tilde{\mu}^{(2)} - \tilde{\mu}^{(1)} \right) \). Finally \( \tilde{\xi} \) and \( \tilde{w}^* \) give an estimate of (B.9) as

\[
\hat{p}(y = 1|\tilde{x}; \tilde{\theta}) = \begin{cases} 
\frac{\sum_{t=1}^{T} \left( w^{T} \tilde{x}^{(t)} \right)}{\sum_{t=1}^{T} \left( w^{T} \tilde{x}^{(t)} > \tilde{c} \right)} & \text{if } w^{T} \tilde{x} > \tilde{c} \; ; \\
\frac{\sum_{t=1}^{T} \left( w^{T} \tilde{x}^{(t)} < \tilde{c} \right)}{\sum_{t=1}^{T} \left( w^{T} \tilde{x}^{(t)} < \tilde{c} \right)} & \text{if } w^{T} \tilde{x} < \tilde{c} .
\end{cases}
\]  
(B.16)

Data likelihood:

\[
\hat{p}(D^{(0)}|\tilde{\theta}) = \prod_{t=1}^{T} \left[ \hat{p}(y_t|\tilde{x}^{(t)}; \tilde{\theta}) \right]^{y_t} \left[ 1 - \hat{p}(y_t|\tilde{x}^{(t)}; \tilde{\theta}) \right]^{1-y_t} .
\]  
(B.17)

4. Bayesian Classifier

Prior: \( \pi = P(Y = 1) \)

Likelihoods: \( f_1(\tilde{x}) = P(\tilde{x}|Y = 1), f_0(\tilde{x}) = P(\tilde{x}|Y = 0) \)

Posterior: \( P(Y = 1|\tilde{x}) = \frac{P(\tilde{x}|Y = 1)P(Y = 1)}{P(\tilde{x}|Y = 1)P(Y = 1) + P(\tilde{x}|Y = 0)P(Y = 0)} = \frac{f_1(\tilde{x})\pi}{f_1(\tilde{x})\pi + f_0(\tilde{x})(1-\pi)} \).  
(B.18)

It’s trivial to estimate \( \pi \) as \( \hat{\pi} = \frac{1}{T} \sum_{t=1}^{T} y_t \). The main task is to infer \( f_1(\tilde{x}) \) and \( f_0(\tilde{x}) \) from the data \( D^{(0)} \). The simplest form is the naive Bayes in which \( f_0(\tilde{x}) = \prod_{j} P(x_j|y) \) and each \( P(x_j|y) \) can take a non-parametric form. If there exists collinearity among the features, the principal component analysis could be used to handle that. If there are enough many samples (\( T \) is large), then finite mixture models can be applied to \( f_1(\tilde{x}) \) and \( f_0(\tilde{x}) \). But the inference usually resort to the EM algorithm or Gibbs sampling.

Data likelihood:

\[
\hat{p}(D^{(0)}|\tilde{\theta}) = \prod_{t=1}^{T} \left[ \frac{f_1(\tilde{x}^{(t)})\pi}{f_1(\tilde{x}^{(t)})\pi + f_0(\tilde{x}^{(t)})(1-\pi)} \right]^{y_t} \left[ \frac{f_0(\tilde{x}^{(t)})(1-\pi)}{f_1(\tilde{x}^{(t)})\pi + f_0(\tilde{x}^{(t)})(1-\pi)} \right]^{1-y_t} .
\]  
(B.19)

5. Support Vector Machine

The basic idea is to find a hyper plane \( \tilde{w}^{T}\tilde{x} + b = 0 \) by solving the following constrained optimization problem:

\[
\text{Min } f(\tilde{w}, b, \tilde{\xi}) = \frac{\tilde{w}^{T}\tilde{w}}{2} + C \sum_{t=1}^{T} \xi_t \]  
(B.20)

Subject to \( 2(y_t - 0.5)(\tilde{w}^{T}\tilde{x}^{(t)} + b) \geq 1 - \xi_t, t = 1, ..., T; \)  
(B.21)

\[
\xi_t \geq 0, t = 1, ..., T, \]  
(B.22)
where $C > 0$ is the penalty parameter and $\xi_t$ ‘s are slack variables. For each $t$, the constraint (B.21) is $w^T \hat{x}^{(t)} + b \geq 1 - \xi_t$ if $y_t = 1$; or $w^T \hat{x}^{(t)} + b \leq -1 + \xi_t$ if $y_t = 0$. That is if the point $\hat{x}^{(t)}$ is labeled “1”, it is expected to be in the region $w^T \hat{x}^{(t)} + b \geq 1$; otherwise, it is expected to be in the region $w^T \hat{x}^{(t)} + b \leq -1$. Let $\hat{x}^{(t_1)}$ be a point in the region $w^T \hat{x}^{(t)} + b \geq 1$ and $\hat{x}^{(t_2)}$ be a point in the region $w^T \hat{x}^{(t)} + b \leq -1$. Consider $2 \leq |w^T (\hat{x}^{(t_2)} - \hat{x}^{(t_1)})| = \|w\|_2 d(\hat{x}^{(t_1)}, \hat{x}^{(t_2)}; \hat{\theta})$, where $d(\hat{x}^{(t_1)}, \hat{x}^{(t_2)}; \hat{\theta}) = \|w^T (\hat{x}^{(t_2)} - \hat{x}^{(t_1)})\| \|w\|_2$ quantities the separation between $\hat{x}^{(t_1)}$ and $\hat{x}^{(t_2)}$. We want to maximize $d(\hat{x}^{(t_1)}, \hat{x}^{(t_2)}; \hat{\theta})$. Since $d(\hat{x}^{(t_1)}, \hat{x}^{(t_2)}; \hat{\theta}) \geq \frac{2}{\|w\|_2}$, we can simply minimize $\|w\|_2$, therefore the objective function (B.20) contains the term $\frac{2}{\|w\|_2}$. For the second term $C \sum_{t=1}^{T} \xi_t$, it’s simply the penalty on those points that are in the wrong region. The solution to this constrained optimization problem gives a maximum soft margin separation. Given $\hat{\theta}$ and $\hat{\xi}$, which are estimated from the training data $D^{(0)}$, the estimate of $P(Y = 1|x; \hat{\theta})$ is

$$
\hat{P}(Y = 1|x; \hat{\theta}) = \begin{cases} 
\frac{\sum_{t=1}^{T} I(w^T \hat{x}^{(t)} + \hat{\xi} \geq 1)}{\sum_{t=1}^{T} I(w^T \hat{x}^{(t)} + \hat{\xi} = 1)} y_t & \text{if } w^T \hat{x} + \hat{\xi} \geq 1; \\
\frac{\sum_{t=1}^{T} I(w^T \hat{x}^{(t)} + \hat{\xi} = -1)}{\sum_{t=1}^{T} I(w^T \hat{x}^{(t)} + \hat{\xi} = -1)} y_t & \text{if } w^T \hat{x} + \hat{\xi} \leq -1.
\end{cases}
$$

The data likelihood is simply $\hat{P}(D^{(0)}|\hat{\theta}) = \prod_{t=1}^{T} \hat{P}(y_t|x^{(t)}; \hat{\theta})^{y_t} [1 - \hat{P}(y_t|x^{(t)}; \hat{\theta})]^{1-y_t}$.

Appendix C. Sample R-Query from Hadoop

```
# Commands/Scripts for querying data
#
# GetBigTableByDateRange
# Description:
#   GetBigTableByDateRange, which gets sample data of signals in
#   signals.list with filtering by id.list and date range from HDFS to
#   R, then join these signal sample data.
# Usage:
#   GetBigTableByDateRange(signals.list, id.list, date.range,date.type)
# Arguments:
#   signals.list: list type
#   id.list: list type
#   date.range: character type
#   date.type: numeric type
#
# Date: 02/12/2013
library(SigGMS);

# Output file name
go_out_filename = './test_file_tbl_01.csv';

# Stock ID
stock_ids = list();
stock_ids[[1]] = 'US6174464486';
stock_ids[[2]] = 'US46625H1005';
```
# Date range of the query
dt_range = '2005-07-01,2012-12-31';

# Raw varibles to query
raw_headers = c("sigdev_rgen_raw_adjopen_backward",
                "sigdev_rgen_raw_adjhigh_backward",
                "sigdev_rgen_raw_adjlow_backward",
                "sigdev_rgen_raw_adjclose_backward");
n_names = length(raw_headers);
raw_variables = list();
for (i in 1:n_names)
{
  raw_variables[[i]] = paste(raw_headers[i], ',', 0, sep=''); # 0 is the version number
}

# Read into R's data frame
my_tbl <- GetBigTableByDateRange(raw_variables, stock_ids, dt_range, 3);

names(my_tbl)[1] = 'ID';
names(my_tbl)[2] = 'Date';
for (i in 1:n_names)
{
  names(my_tbl)[i+2] = raw_headers[i];
}

print(my_tbl[1:10,]);
write.csv(my_tbl, file = out_filename, row.names=FALSE, quote = FALSE);

# The execution command
# R --no-save < data_query_02122013_tbl.R > ./query_tbl_log_01.out &

Appendix D. Sample R-Write to Hadoop

#!/usr/bin/env Rscript
f1 <- file.path('.', "mkt_trend.R")
if(file.exists(f1))
  source(f1)
f2 <- file.path('../', "mkt_trend.R")
if(file.exists(f2))
  source(f2)

value.list <-
list("sigdev_rgen_raw_adjopen_backward","sigdev_rgen_raw_adjhigh_backward","sigdev_rgen_raw_adjlow_backward","sigdev_rgen_raw_adjclose_backward")
type <- "right"
key.by <- "ID"

SignalCalc <- function(data)
{
  # put your signal code here and return result
  # ...
  n_data = dim(data)[1]; # Number of rows in data
  m_data = dim(data)[2]; # Number of columns in data
  # Get values (note that the first column of data is ID, the second column of data is Date, the rest columns are values)
  x_data = list();
  for (i in 3:m_data)
  {
    #...
x_data[[i-2]] = as.numeric(as.character(data[,i]));
# x_data[[1]]: Adjusted open prices
# x_data[[2]]: Adjusted high prices
# x_data[[3]]: Adjusted low prices
# x_data[[4]]: Adjusted close prices
n_list = m_data-3+1;
adjclose = 4; # Point to the adjusted close prices
lengths_h = c(30,15);
values_q = c(95,90,85,80,75);
lengths_h = length(lengths_h);
values_q = length(values_q);
tgt_rtns = n_lengths_h*n_values_q;
tgt_rtns = matrix(NA,n_data,m_tgt_rtns);
# Generate the names of the variables
var_names = c();
for (i in 1:n_lengths_h)
{
    for (j in 1:n_values_q)
    {
        var_name = paste('xyertn_h', lengths_h[i], 'q', values_q[j], '_close2st', sep='');
        var_names = c(var_names,var_name);
    }
}
# Compute the target returns
k = 0;
for (i in 1:n_lengths_h)
{
    for (j in 1:n_values_q)
    {
        k = k + 1;
        for (t in 1:(n_data-lengths_h[i]))
        {
            x = c();
            for (i_list in 1:n_list)
            {
                x = c(x, x_data[[i_list]][(t+1):(t+lengths_h[i])]);
            }
            x_filtered = x[is.na(x) == FALSE];
            if (is.na(x_data[[adjclose]][t]) == FALSE) & (length(x_filtered) >= 1)
            {
                tgt_rtns[k] = quantile(x_filtered, 0.01*values_q[j],na.rm=TRUE) / x_data[[adjclose]][t] - 1.0;
            }
        }
    }
}
# End of computing the target returns
# Generate the output
k = 1;
data_out = cbind( data[,1:2], c(rep(var_names[k],n_data)), tgt_rtns[k] );
colnames(data_out)[3] = 'varname';
colnames(data_out)[4] = 'varvalue';
if (m_tgt_rtns > 1)
{
    for (k in 2:m_tgt_rtns)
    {
        data_out_k = cbind( data[,1:2], c(rep(var_names[k],n_data)), tgt_rtns[k] );
colnames(data_out_k)[3] = 'varname';
colnames(data_out_k)[4] = 'varvalue';
data_out = rbind(data_out,data_out_k);
    }
}
}  
return (data_out);
}

# StreamingMktTrendMain(value.list,type,key_by)

Appendix E. Sample local R-computing code
# Program: Market Direction V2.0
# Authors: Xugang Ye
# Spring 2013

# Note: Three steps
# Step 1: Find up-probs by PCA-linear-regression using rolling windows of lengths 50 to 90
# Step 2: Select and caliborate up-probs via p1 and p2 under rolling windows of lengths 40 to 90
# Step 3: Filter the caliborated results by comparing the profitabilities of the model and the market
#
# Note: use unbiased estimation of the standard deviation of the residual

# Extra libraries
library(multicore);

### Program Parameters
start_step_No = 1; # 1: start from begining, 2: start from the up-probs file already generated
print(paste('Step to start from', start_step_No));

in_dir = '../data';
out_dir = '../some_folder';
in_filename = 'original_signals.csv';
out_filename1 = '/tickerx_1up_probs_from';
out_filename2 = '/tickerx_1comb_preds2Msfs01_from';
out_filename3 = '/tickerx_levatbl2Msfs01_valid_from';

Date_format = "%Y-%m-%d";

MCORES = 10; print(paste('Number of CPU cores requested:', MCORES));

start_date = as.Date('2007-01-03'); # Start date of the up-probs
date_start = as.Date('2010-12-29'); # End date of the up-probs

eva_end_date = as.Date('2010-12-29'); # End date of evaluation period

my_tolerance = 1e-10;

train_win_min = 50; # Minimum length of rolling window for step 1
train_win_max = 90; # Maximum length of rolling window for step 1
train_win_list = c(train_win_min:train_win_max);
print('Lengths of rolling window for step 1:');
print(train_win_list);

PCA_contribution = 0.9; # Total PCs variance contribution

orig_target_info = 7;
first_signal_idx = orig_target_info + 1;
no_target_info = 4;
new_first_signal_idx = first_signal_idx + no_target_info;

T_win_min = 40; # Minimum length of rolling window for step 2
T_win_max = 90; # Maximum length of rolling window for step 2
T_win_list = c(T_win_min:T_win_max);
print('Lengths of rolling window for step 2:');
print(T_win_list);
delta_p = 1.0e-2; # Threshold for comparing p1+p2 with 1.0
comb_mtd = 2; # Which combining method to choose
print(paste('Combing method:',comb_mtd));

T_win_filter_min = 40; # Minimum length of rolling window for step 3
T_win_filter_max = 90; # Maximum length of rolling window for step 3
T_win_filter_list = c(T_win_filter_min:T_win_filter_max);
print('Lengths of rolling window for step 3:');
print(T_win_filter_list);

thres_hold = 0.0; # Decision threshold
### End Program Parameters

### Functions
#
LinReg_prediction <- function(x_test, x_train, y_train, train_win)
{ # Begin function
  # PCA transformation
cor_x = cor(rbind(x_train,x_test)); # Generate correlation matrix from [x_train;x_test]
eig_x <- eigen(cor_x);
Q = eig_x$vectors;
lam = eig_x$values;
sum_lam = sum(lam);
j = 1;
s = lam[j];
while (s/sum_lam < PCA_contribution)
{
  j = j + 1;
s = s + lam[j];
} #print(j);
x_train = x_train %*% Q[,1:j];
x_test = x_test %*% Q[,1:j];
# End PCA transformation
model_win <- lm(y_train ~ x_train); # Linear regression model
b_win = as.numeric(model_win$coefficients); # Retrieve the fitting line (as the combining weights)
s = sd(as.numeric(model_win$residuals)); # Retrieve the fitting line (as the combining weights)
s = sd(as.numeric(model_win$residuals));
c = as.numeric(model_win$residuals);
s = sum(e*c)/max(1, dim(x_train)[1] - dim(x_train)[2]);
s = sqrt(s);
y_test = sum( c(1,as.numeric(x_test)) * b_win );

# Compute the up-prob (for market-trend model)
p = pnorm( y_test/s, 0, 1 );
return (p);
} # End function

### Inputs:
#
x - up probs
y - target labels (1 = "up", 0 = "down ", 0.5 = "flat")
get_accuracies <- function(x,y)
{
smth = 0.01; # Smoothing factor
p = c(rep(0.0,3)); # To store p1, p2, and q(up prior)
n = length(y); # Number of market's ups and downs
n1 = sum(y); # Number of market's ups
p[1] = (sum(x*y) + smth) / (n1 + 2*smth); # The portion of being right in market's ups
p[2] = (sum((1.0-x)*(1.0-y)) + smth) / (n - n1 + 2*smth); # The portion of being right in market's downs
p[3] = (n1 + smth) / (n + 2*smth); # Market's up-prior

return (p);

# Inputs:
# x - up probs
# y - binary labels (1 = "up", 0 = "down", 0.5 = "flat")
# r - rtns
get_accuraciesM <- function (x,y,r)
{
  smth = 0.1; # Smoothing factor
  p = c(rep(0.0,3)); # To store p1, p2, and q(up prior)
  n = sum(abs(r)); # sum of market's gains and losses
  n1 = sum(y*r); # sum of market's gains
  p[1] = (sum(x*y*r) + smth) / (n1 + 2*smth); # The portion of gain in market's gains
  p[2] = (sum((1.0-x)*(1.0-y)*(-r)) + smth) / (n - n1 + 2*smth); # The portion of gain in market's losses
  p[3] = (n1 + smth) / (n + 2*smth); # Market's profitability

  return (p);
}

# This function is regression procedure that is implemented using multi-cores
rolling_regress <- function(train_win)
{
  up_probs = rep(NA,n_test);
  for (t in t_start:t_end)
  {
    t0 = max(1,t-train_win);
    x_train = as.matrix(my_data[t0:(t-1),J]);
    r = rOO_security[t0:(t-1)];
    r[t-t0] = rOC_security[t-1]; # Remove target leak
    r = as.matrix(r);
    x_test = as.matrix(my_data[t,J]);
  
    # Compute the mean and std of each column of rbind(x_train, x_test)
    x_absmax = apply(abs(rbind(x_train, x_test)), 2, max);
    x_std = apply(rbind(x_train, x_test), 2, sd);
    j_select = which(x_std > my_tolerance);
  
    # Remove the columns with no variation
    x_train = x_train[,j_select];
    x_test = t(as.matrix(x_test[j_select]));
    x_absmax = x_absmax[j_select];
    x_std = x_std[j_select];
  
    # Scaling
    for (i in 1:(t-t0))
    {
      x_train[i,] = x_train[i,] / x_absmax;
    }
    x_test = x_test / x_absmax;

    up_probs[t-t_start+1] = LinReg_prediction(x_test, x_train, r, t-t0);
  }

# This function is combining procedure 2 that is implemented using multi-cores
rolling_pred_mtd2 <- function(T_win)
{
  rolling_predict = c(rep(NA,n_up_probs)); # To store the combined result under a length of rolling window
  for (t in (T_win + 1):n_up_probs)
  {
    rolling_predict[t] = 0.0;
    t0 = max(1,t-T_win);
    y = yOO[t0:(t-1)];
    y[t-t0] = yOC[t-1];
    r = rOO[t0:(t-1)];
    r[t-t0] = rOC[t-1];
    for (j in J)
    {
      x = up_probs_out[t0:(t-1),j];
      p = get_accuraciesM(x,y,r);
      p12 = p[1] + p[2];
      x_t = up_probs_out[t,j];

      # Capping
      p_ub = max(p[1],1.0-p[2]);
      p_lb = min(p[1],1.0-p[2]);
      if (x_t < p_lb)
      {
        x_t = p_lb;
      }
      else if (x_t > p_ub)
      {
        x_t = p_ub;
      }
      if (abs(p12 - 1.0) > delta_p)
      {
        rolling_predict[t] = rolling_predict[t] + (x_t + p[2] - 1.0)/(p12 - 1.0); # Add the calibrated one
      }
      else
      {
        rolling_predict[t] = rolling_predict[t] + 0.5; # Throw away the random one
      }
    }
    rolling_predict[t] = rolling_predict[t]/n_J;
  }

  return (rolling_predict);
}

# This function is filtering procedure
rolling_filter_hq <- function(T_win_filter)
{
  rolling_predict = c(rep(NA,n_up_probs));
  for (t in (T_win_max+1+T_win_filter):n_up_probs)
  {
    rolling_predict[t] = 0.0;
    t0 = max(1,t-T_win_filter);
    y = yOO[t0:(t-1)];

    # Capping
    p_ub = max(p[1],1.0-p[2]);
    p_lb = min(p[1],1.0-p[2]);
    if (x_t < p_lb)
    {
      x_t = p_lb;
    }
    else if (x_t > p_ub)
    {
      x_t = p_ub;
    }
    if (abs(p12 - 1.0) > delta_p)
    {
      rolling_predict[t] = rolling_predict[t] + (x_t + p[2] - 1.0)/(p12 - 1.0); # Add the calibrated one
    }
    else
    {
      rolling_predict[t] = rolling_predict[t] + 0.5; # Throw away the random one
    }
  }

  return (rolling_predict);
}
y[t-t0] = yOC[t-1];
r = rOO[0:(t-1)];
r[t-t0] = rOC[t-1];
for (j in 2:(n_T_wins+1))
{
  x = comb_pred_out[0:(t-1),j];
  p = get_accuraciesM(x,y,r);
  x_t = comb_pred_out[t,j];
  # Voting
  if ((h < p[3]) & (p[3] > 0.5))
  {
    rolling_predict[t] = rolling_predict[t] + p[3];
  }
  else
  {
    rolling_predict[t] = rolling_predict[t] + x_t;
  }
  rolling_predict[t] = rolling_predict[t]/n_T_wins;
}
return (rolling_predict);

cumu_rtn_without_cost <- function(x,r)
{
  n = length(r);
  rtn = 1.0;
  for (t in 1:n)
  {
    rtn = rtn*(1.0 + x[t]*r[t]);
  }
  rtn = rtn - 1.0;
  return (rtn);
}
### Main Script
## Data preperation
# Read the source data
in_data = data.frame(read.csv(paste(in_dir,in_filename,sep='/'),header=T,stringsAsFactors=F));
in_data[,c('dt')] = as.Date(in_data[,c('dt')], Date_format);
n_data = dim(in_data)[1];
v_data = dim(in_data)[2];
no_signals = v_data - first_signal_idx + 1;
print( paste('Number of rows in raw data:', n_data) );
print( paste('Number of columns in raw data:', v_data) );
print( paste('Number of NAs in raw data:', sum(is.na(in_data))) );
rOO_security = c(rep(NA,n_data));
rOC_security = c(rep(NA,n_data));
# Open to Open return (t+1 -> t+2)
rOO_security[1:(n_data-2)]=(in_data$open_imp[3:n_data]/in_data$open_imp[2:(n_data-1)])-1.0;
# Open to Close return (t+1 -> t+1)
rOC_security[1:(n_data-1)]=(in_data$clpx_imp[2:n_data]/in_data$open_imp[2:n_data])-1.0;

# Label of Open to Open return (t+1 -> t+2)
yOO_security = c(rep(NA,n_data));
yOO_security[yOO_security > 0] <- 1;
yOO_security[yOO_security == 0] <- 0.5;
yOO_security[yOO_security < 0] <- 0;

# Label of Open to Close return (t+1 -> t+1)
yOC_security = c(rep(NA,n_data));
yOC_security[yOC_security > 0] <- 1;
yOC_security[yOC_security == 0] <- 0.5;
yOC_security[yOC_security < 0] <- 0;

# Create new data frame for training
my_data=data.frame( in_data[,1:(first_signal_idx-1)] );
names(my_data)[1] = 'Date';
my_data[,first_signal_idx] <- rOO_security;
my_data[,first_signal_idx+1] <- yOO_security;
my_data[,first_signal_idx+2] <- rOC_security;
my_data[,first_signal_idx+3] <- yOC_security;
new_last_signal_idx = new_first_signal_idx + no_signals - 1;
my_data[,new_first_signal_idx:new_last_signal_idx] = in_data[,first_signal_idx:(first_signal_idx+no_signals-1)];
rm(in_data);

# Check NAs
x = as.matrix(is.na(my_data) == TRUE);
x_sum_col = apply(x,2,sum);
j_NA = which(x_sum_col == n_data);
original_names = names(my_data);
print( paste('NA-variables:',original_names[j_NA]) );
for (j in j_NA) {
  my_data[,original_names[j]] <- NULL;
}

v_data = dim(my_data)[2];
new_last_signal_idx = v_data;
print(paste('Check NAs after NA-removal:',sum(is.na(my_data[,new_first_signal_idx:new_last_signal_idx]) == TRUE)));

## Data preparation completed

## Generate up-probs or, if the up-probs file is already there, just read it
if (start_step_No == 1) # Step 1 has not been finished, do it
{
  t_start = 1;
  while (my_data[t_start,1] < start_date) {
    t_start = t_start + 1;
  }

  t_end = t_start;
  while (my_data[t_end,1] < end_date) {
    t_end = t_end + 1;
  }

  print(paste('Start date of step 1:', my_data[t_start, 1],t_start));
  print(paste('End date of step 1:', my_data[t_end, 1],t_end));
  print(paste('Number of test days in step 1:', n_test));
}
J = c(new_first_signal_idx:new_last_signal_idx); # Variable indices
n_J = length(J);

# Build the dataframe for storing the up-probs
up_probs_out = data.frame(my_data[t_start:t_end,c('Date')]);
names(up_probs_out)[1] = 'Date';
vnames = sapply(train_win_list, function(x) {paste('up_prob',x,sep='')});
up_probs_out[,vnames] <- NA;

# Regressing using multi-cores
n_train_wins = length(train_win_list);
mproc <- min(MCORES, n_train_wins);
pred_out <- mclapply(train_win_list, rolling_regress, mc.cores=mproc);
up_probs_out[,vnames] <- do.call(cbind, pred_out);

# Output to .csv
write.csv(up_probs_out, file = paste(out_dir, out_filename1, train_win_min, 'to', train_win_max, '.csv', sep=''))

if (start_step_No == 2) # Step 1 has been finished, just read the up_probs file
{
  up_probs_file = paste(out_dir, out_filename1, train_win_min, 'to', train_win_max, '.csv', sep='');
  up_probs_out = data.frame(read.csv(up_probs_file, header=T, stringsAsFactors=F));
  up_probs_out[,c('X')] <- NULL;
  up_probs_out[,c('Date')] = as.Date(up_probs_out[,c('Date')], '%Y-%m-%d');
}

if (((start_step_No == 1) | (start_step_No == 2))
{
  n_up_probs = dim(up_probs_out)[1];
  v_up_probs = dim(up_probs_out)[2];

  print(paste('Start date of up-probs:', up_probs_out[1,c('Date')], 1));
  print(paste('End date of up-probs:', up_probs_out[n_up_probs,c('Date')], n_up_probs));
  print(paste('Number of rows in up-probs data:', n_up_probs));
  print(paste('Number of columns in up-probs data:', v_up_probs));

  t_start = 1;
  while (my_data[t_start,1] < up_probs_out[1,c('Date')])
  {
    t_start = t_start + 1;
  }

  t_end = t_start;
  while (my_data[t_end,1] < up_probs_out[n_up_probs,c('Date')])
  {
    t_end = t_end + 1;
  }

  yOO = yOO_security[t_start:t_end];
  yOC = yOC_security[t_start:t_end];
  rOO = rOO_security[t_start:t_end];
  rOC = rOC_security[t_start:t_end];

  J = c(2:v_up_probs);
  n_J = length(J); # Number of up-probs results

  # Build the dataframe for storing the combined predictions
  comb_pred_out = data.frame(up_probs_out[,c('Date')]);
names(comb_pred_out)[1] = 'Date';
vnames = sapply(T_win_list, function(x) {paste('mdl_prediction_valid',x,sep='')});
comb_pred_out[,vnames] <- NA;
vnames_agg = sapply(T_win_filter_list, function(x) {paste('mdl_agg_valid',x,sep='')});
comb_pred_out[,vnames_agg] <- NA;

# Combining using multi-cores
n_T_wins = length(T_win_list);
mproc <- min(MCORES, n_T_wins);
pred_out <- mclapply(T_win_list, rolling_pred_mtd2, mc.cores=mproc);
comb_pred_out[,vnames] <- do.call(cbind, pred_out);

# Filtering using multi-cores
n_T_wins_filter = length(T_win_filter_list);
mproc <- min(MCORES, n_T_wins_filter);
pred_out <- mclapply(T_win_filter_list, rolling_filter_hq, mc.cores=mproc);
comb_pred_out[,vnames_agg] <- do.call(cbind, pred_out);

# Final aggregation
agg_final = c(rep(NA, n_up_probs));
agg_start = T_win_max+1+T_win_filter_max;
agg_final[agg_start:n_up_probs] = apply(comb_pred_out[agg_start:n_up_probs,vnames_agg],1,sum);
comb_pred_out[,paste('agg_final',T_win_filter_min,'to',T_win_filter_max,sep='') ] = agg_final;

# Output to .csv
write.csv(comb_pred_out, file = paste(out_dir, out_filename2, T_win_min, 'to',  T_win_max, '.csv', sep=''));

x = agg_final[agg_start:n_up_probs];

print(paste('Threshold for decision = ',thres_hold));
x[x > 0.5+thres_hold] <- 1;
x[x < 0.5-thres Holden] <- -1;
x[(x >= 0.5-thres_hold)&(x <= 0.5+thres_hold)] <- 0;

# Evaluate the prediction results
r = rOO[agg_start:n_up_probs];
y = yOO[agg_start:n_up_probs];

# Compute the cumulative returns
market_cumu_rtn = cumu_rtn_without_cost(c(rep(1,length(r))),r);
model_cumu_rtn = cumu_rtn_without_cost(x,r);

# Compute the accuracies
p = get_accuracies(x,y);

# Generate the evaluation table
eva_start = agg_start;
eva_end = eva_start;
while (up_probs_out[eva_end,c('Date')] < eva_end_date)
{
    eva_end = eva_end + 1;
}

n_eva = eva_end - eva_start + 1;
mdl_eva_tbl = data.frame(c(T_win_list, T_win_filter_list, NA));
names(mdl_eva_tbl)[1] = '#_valid_days';
mdl_eva_tbl[,c('From')] <- up_probs_out[eva_start,c('Date')];
mdl_eva_tbl[,c('To')] <- up_probs_out[eva_end,c('Date')];
mdl_eva_tbl[,c('Decision_threshold')] <- thres_hold;
mdl_eva_tbl[,c('#_days')] <- n_eva;
mdl_eva_tbl[,c('p1')] <- NA;
mdl_eva_tbl[,c('p2')] <- NA;
mdl_eva_tbl[,c('q')] <- NA;
mdl_eva_tbl[,c('1-q')] <- NA;
mdl_eva_tbl[,c('p1+p2')] <- NA;
mdl_eva_tbl[,c('hit-rate')] <- NA;
mdl_eva_tbl[,c('market_cumu_rtn')] <- NA;
mdl_eva_tbl[,c('mdl_cumu_rtn')] <- NA;
mdl_eva_tbl[,c('#_days_noncash')] <- NA;
mdl_eva_tbl[,c('p1_noncash')] <- NA;
mdl_eva_tbl[,c('p2_noncash')] <- NA;
mdl_eva_tbl[,c('q_noncash')] <- NA;
mdl_eva_tbl[,c('1-q_noncash')] <- NA;
mdl_eva_tbl[,c('p1+p2_noncash')] <- NA;
mdl_eva_tbl[,c('hit-rate_noncash')] <- NA;

for (i in 1:(n_T_wins+n_T_wins_filter+1))
{
    x = comb_pred_out[eva_start:eva_end,i+1];
    x[x > 0.5+thres_hold] <- 1;
    x[x < 0.5-thres_hold] <- -1;
    x[(x >= 0.5-thres_hold)&(x <= 0.5+thres_hold)] <- 0;

    r = rOO[eva_start:eva_end];
    mdl_eva_tbl[,c('market_cumu_rtn')] = cumu_rtn_without_cost(c(rep(1,length(r))),r);
    mdl_eva_tbl[,c('mdl_cumu_rtn')] = cumu_rtn_without_cost(x,r);

    y = yOO[eva_start:eva_end];
    x[x==0] <- 0.5;
    x[x<0] <- 0;

    p = get_accuracies(x,y);
    mdl_eva_tbl[,c('p1')] = p[1];
    mdl_eva_tbl[,c('p2')] = p[2];
    mdl_eva_tbl[,c('q')] = p[3];
    mdl_eva_tbl[,c('1-q')] = 1.0 - p[3];
    mdl_eva_tbl[,c('p1+p2')] = p[1] + p[2];

    # Calculate the accuracies in non-cash days
    x1 = x[x != 0.5];
    y1 = y[x != 0.5];
    mdl_eva_tbl[,c('#_days_noncash')] = length(x1);
    p = get_accuracies(x1,y1);
    mdl_eva_tbl[,c('p1_noncash')] = p[1];
    mdl_eva_tbl[,c('p2_noncash')] = p[2];
    mdl_eva_tbl[,c('q_noncash')] = p[3];
    mdl_eva_tbl[,c('1-q_noncash')] = 1.0 - p[3];
    mdl_eva_tbl[,c('p1+p2_noncash')] = p[1] + p[2];
}
}
print(mdl_eva_tbl);
# Output to .csv
write.csv(mdl_eva_tbl, file = paste(out_dir, out_filename3, T_win_min, 'to', T_win_max, '.csv', sep=''));

} # End if ((start_step_No == 1) | (start_step_No == 2))

if ((start_step_No != 1) & (start_step_No != 2))
{
  print('No further clear instruction, program is stopped');
}

### End main Script