Chapter 20  
Data Mining for Algorithmic Asset Management  

Giovanni Montana and Francesco Parrella  

Abstract Statistical arbitrage refers to a class of algorithmic trading systems implementing data mining strategies. In this chapter we describe a computational framework for statistical arbitrage based on support vector regression. The algorithm learns the fair price of the security under management by minimizing a regularized $\varepsilon$-insensitive loss function in an on-line fashion, using the most recent market information acquired by means of streaming financial data. The difficult issue of adaptive learning in non-stationary environments is addressed by adopting an ensemble learning approach, where a meta-algorithm strategically combines the opinion of a pool of experts. Experimental results based on nearly seven years of historical data for the iShare S&P 500 ETF demonstrate that satisfactory risk-adjusted returns can be achieved by the data mining system even after transaction costs.  

20.1 Introduction  

In recent years there has been increasing interest for active approaches to investing that rely exclusively on mining financial data, such as market-neutral strategies [11]. This is a general class of investments that seeks to neutralize certain market risks by detecting market inefficiencies and taking offsetting long and short positions, with the ultimate goal of achieving positive returns independently of market conditions. A specific instance of market-neutral strategies that heavily relies on temporal data mining is referred to as statistical arbitrage [11, 14]. Algorithmic asset management systems embracing this principle are developed to make spread trades, namely trades that derive returns from the estimated relationship between two statistically related securities.
An example of statistical arbitrage strategies is given by pairs trading [6]. The rationale behind this strategy is an intuitive one: if the difference between two statistically depending securities tends to fluctuate around a long-term equilibrium, then temporary deviations from this equilibrium may be exploited by going long on the security that is currently under-valued, and shorting the security that is over-valued (relatively to the paired asset) in a given proportion. By allowing short selling, these strategies try to benefit from decreases, not just increases, in the prices. Profits are made when the assumed equilibrium is restored.

The system we describe in this chapter can be seen as a generalization of pairs trading. In our setup, only one of the two dependent assets giving rise to the spread is a tradable security under management. The paired asset is instead an artificial one, generated as a result of a data mining process that extracts patterns from a large population of data streams, and utilizes these patterns to build up the synthetic stream in real time. The extracted patterns will be interpreted as being representative of the current market conditions, whereas the synthetic asset will represent the fair price of the target security being traded by the system. The underlying concept that we try to exploit is the existence of time-varying cross-sectional dependencies among securities. Several data mining techniques are being developed lately to capture dependencies among data streams in a time-aware fashion, both in terms of latent factors [12] and clusters [1]. Recent developments include novel database architectures and paradigms such as CEP (Complex Event Processing) that discern patterns in streaming data, from simple correlations to more elaborated queries.

In financial applications, data streams arrive into the system one data point at a time, and quick decisions need to be made. A prerequisite for a trading system to operate efficiently is to learn the novel information content obtained from the most recent data in an incremental way, slowly forgetting the previously acquired knowledge and, ideally, without having to access all the data that has been previously stored. To meet these requirements, our system builds upon incremental algorithms that efficiently process data points as they arrive. In particular, we deploy a modified version of on-line support vector regression [8] as a powerful function approximation device that can discover non-negligible divergences between the paired assets in real time. Streaming financial data are also characterized by the fact that the underlying data generating mechanism is constantly evolving (i.e. it is non-stationary), a notion otherwise referred to as concept drifting [12]. Due to this difficulty, particularly in the high-frequency trading spectrum, a trading system’s ability to capture profitable inefficiencies has an ever-decreasing half life: where once a system might have remained viable for long periods, it is now increasingly common for a trading system’s performance to decay in a matter of days or even hours. Our attempt to deal with this challenge in an autonomous way is based on an ensemble learning approach, where a pool of trading algorithms or experts are evolved in parallel, and then strategically combined by a master algorithm. The expectation is that combining expert opinion can lead to fewer trading mistakes in all market conditions.
20.2 Backbone of the Asset Management System

In this section we outline the rationale behind the statistical arbitrage system that forms the theme of this chapter, and provide a description of its main components. Our system imports \( n + 1 \) cross-sectional financial data streams at discrete time points \( t = 1, 2, \ldots \). In the sequel, we will assume that consecutive time intervals are all equal to 24 hours, and that a trading decision is made on a daily basis. Specifically, after importing and processing the data streams at each time \( t \), a decision to either buy or short sell a number of shares of a target security \( Y \) is made, and an order is executed. Different sampling frequencies (e.g. irregularly spaced intervals) and trading frequencies could also be incorporated with only minor modifications.

The imported data streams represent the prices of \( n + 1 \) assets. We denote by \( y_t \) the price of the security \( Y \) being traded by the system, whereas the remaining \( n \) streams, collected in a vector \( s_t = (s_{t1}, \ldots, s_{tn})^T \), refer to a large collection of financial assets and economic indicators, such as other security prices and indices, which possess some explanatory power in relation to \( Y \). These streams will be used to estimate the fair price of the target asset \( Y \) at each observational time point \( t \), in a way that will be specified below. We postulate that the price of \( Y \) at each time \( t \) can be decomposed into two components, that is \( y_t = z_t + m_t \), where \( z_t \) represents the current fair price of \( Y \), and the additive term \( m_t \) represents a potential mispricing. No further assumptions are made regarding the data generating process. Clearly, if the markets were always perfectly efficient, we would have that \( y_t = z_t \) at all times. However, when \( |m_t| > 0 \), an arbitrage opportunity arises. For instance, a negative \( m_t \) indicates that \( Y \) is temporarily under-valued. In this case, it is sensible to expect that the market will promptly react to this temporary inefficiency with the effect of moving the target price up. Under this scenario, an investor would then buy a number of shares hoping that, by time \( t + 1 \), a profit proportional to \( y_{t+1} - y_t \) will be made. Our system is designed to identify and exploit possible statistical arbitrage opportunities of this sort in an automated fashion. This trading strategy can be formalized by means of a binary decision rule \( d_t \in \{0, 1\} \) where \( d_t = 0 \) encodes a sell signal, and \( d_t = 1 \) a buy signal. Accordingly, we write

\[
   d_t(m_t) = \begin{cases} 
   0 & m_t > 0 \\
   1 & m_t < 0 
\end{cases} 
\]  

(20.1)

where we have made explicit the dependence on the current mispricing \( m_t = y_t - z_t \). If we denote the change in price observed on the day following the trading decision as \( r_{t+1} = y_{t+1} - y_t \), we can also introduce a \( 0 - 1 \) loss function \( L_{t+1}(d_t, r_{t+1}) = |d_t - 1_{(r_{t+1} > 0)}| \), where the indicator variable \( 1_{(r_{t+1} > 0)} \) equals one if \( r_{t+1} > 0 \) and zero otherwise. For instance, if the system generates a sell signal at time \( t \), but the security’s price increases over the next time interval, the system incurs a unit loss.

Obviously, the fair price \( z_t \) is never directly observable, and therefore the mispricing \( m_t \) is also unknown. The system we propose extracts knowledge from the large collection of data streams, and incrementally imputes the fair price \( z_t \) on the basis of the newly extracted knowledge, in an efficient way. Although we expect
20.3 Expert-based Incremental Learning

In order to extract knowledge from the streaming data and capture important features of the underlying market in real-time, the system recursively performs a principal component analysis, and extracts those components that explain a large percentage of variability in the $n$ streams. Upon arrival, each stream is first normalized so that all streams have equal means and standard deviations. Let us call $C_t = E(s_t s_t^T)$ the unknown population covariance matrix of the $n$ streams. The algorithm proposed by [16] provides an efficient procedure to incrementally update the eigenvectors of $C_t$ when new data points arrive, in a way that does not require the explicit computation of the covariance matrix. First, note that an eigenvector $g_t$ of $C_t$ satisfies the characteristic equation $\lambda_t g_t = C_t g_t$, where $\lambda_t$ is the corresponding eigenvalue. Let us call $\hat{h}_t$ the current estimate of $C_t g_t$ using all the data up to the current time $t$. This is given by $\hat{h}_t = \frac{1}{t} \sum_{i=1}^{t} s_i s_i^T g_i$, which is the incremental average of $s_i s_i^T g_i$, where $s_i s_i^T$ accounts for the contribution to the estimate of $C_i$ at point $i$. Observing that $g_i = h_i / ||h_i||$, an obvious choice is to estimate $g_i$ as $\hat{h}_{t-1} / ||\hat{h}_{t-1}||$. After some manipulations, a recursive expression for $\hat{h}_t$ can be found as
\[
\hat{h}_t = \frac{t - 1}{t} \hat{h}_{t-1} + \frac{1}{t} s_t s_t^T \frac{\hat{h}_{t-1}}{||\hat{h}_{t-1}||}
\]  
(20.2)

Once the first \( k \) eigenvectors are extracted, recursively, the data streams are projected onto these directions in order to obtain the required feature vector \( x_t \). We are thus given a sequence of paired observations \( (y_1, x_1), \ldots, (y_t, x_t) \) where each \( x_t \) is a \( k \)-dimensional feature vector representing the latest market information and \( y_t \) is the price of the security being traded.

Our objective is to generate an estimate of the target security’s fair price using the data points observed so far. In previous work [9, 10], we assumed that the fair price depends linearly in \( x_t \) and that the linear coefficients are allowed to evolve smoothly over time. Specifically, we assumed that the fair price can be learned by recursively minimizing the following loss function

\[
t - 1 \sum_{i=1}^t (y_i - w_t^T x_i) + C (w_{i+1} - w_i)^T (w_{i+1} - w_i)
\]  
(20.3)

that is, a penalized version of ordinary least squares. Temporal changes in the time-varying linear regression weights \( w_t \) result in an additional loss due to the penalty term in (20.3). The severity of this penalty depends upon the magnitude on the regularization parameter \( C \), which is a non-negative scalar: at one extreme, when \( C \) gets very large, (20.3) reduces to the ordinary least squares loss function with time-invariant weights; at the other extreme, as \( C \) is small, abrupt temporal changes in the estimated weights are permitted. Recursive estimation equations and a connection to the Kalman filter can be found in [10], which also describes a related algorithmic asset management system for trading futures contracts. In this chapter we depart from previous work in two main directions. First, the rather strong linearity assumption is released so as to add more flexibility in modelling the relationship between the extracted market patterns and the security’s price. Second, we adopt a different and more robust loss function. According to our new specification, estimated prices \( f_t(x_t) \) that are within \( \pm \varepsilon \) of the observed price \( y_t \) are always considered fair prices, for a given user-defined positive scalar \( \varepsilon \) related to the noise level in the data. At the same time, we would also like \( f_t(x_t) \) to be as flat as possible. A standard way to ensure this requirement is to impose an additional penalization parameter controlling the norm of the weights, \( ||w||^2 = w^T w \). For simplicity of exposition, let us suppose again that the function to be learned is linear and can be expressed as \( f_t(x_t) = w^T x_t + b \), where \( b \) is a scalar representing the bias. Introducing slack variables \( \xi_t, \xi_t^* \) quantifying estimation errors greater than \( \varepsilon \), the learning task can be casted into the following minimization problem,

\[
\min_{w_t, b_t} \frac{1}{2} w_t^T w_t + C \sum_{i=1}^t (\xi_i + \xi_i^*)
\]  
(20.4)
that is, the support vector regression framework originally introduced by Vapnik [15]. In this optimization problem, the constant $C$ is a regularization parameter determining the trade-off between the flatness of the function and the tolerated additional estimation error. A linear loss of $|\xi_i| - \varepsilon$ is imposed any time the error $|\xi_i|$ is greater than $\varepsilon$, whereas a zero loss is used otherwise. Another advantage of having an $\varepsilon$-insensitive loss function is that it will ensure sparseness of the solution, i.e. the solution will be represented by means of a small subset of sample points. This aspect introduces non negligible computational speed-ups, which are particularly beneficial in time-aware trading applications. As pointed out before, our objective is learn from the data in an incremental way. Following well established results (see, for instance, [5]), the constrained optimization problem defined by Eqs. (20.4) and (20.5) can be solved using a Lagrange function,

$$
L = \frac{1}{2} w_t^T w_t + C \sum_{i=1}^t (\xi_i + \xi_i^+) - \sum_{i=1}^t \left( \eta_i \xi_i + \eta_i^+ \xi_i^+ \right) - \sum_{i=1}^t \alpha_i (\varepsilon + \xi_i - y_i + w_t^T x_i + b_t) - \sum_{i=1}^t \alpha_i^+ (\varepsilon + \xi_i^+ + y_i - w_t^T x_i - b_t)
$$

(20.6)

where $\alpha_i, \alpha_i^+, \eta_i$ and $\eta_i^+$ are the Lagrange multipliers, and have to satisfy positivity constraints, for all $i = 1, \ldots, t$. The partial derivatives of (20.6) with respect to $w, b, \xi$ and $\xi^+$ are required to vanish for optimality. By doing so, each $\eta_i$ can be expressed as $C - \alpha_i$ and therefore can be removed (analogously for $\eta_i^+$). Moreover, we can write the weight vector as $w_t = \sum_{i=1}^t (\alpha_i - \alpha_i^+) x_i$, and the approximating function can be expressed as a support vector expansion, that is

$$
f_t(x_i) = \sum_{i=1}^t \theta_i x_i^T x_i + b_i
$$

(20.7)

where each coefficient $\theta_i$ has been defined as the difference $\alpha_i - \alpha_i^+$. The dual optimization problem leads to another Lagrangian function, and its solution is provided by the Karush-Kuhn-Tucker (KKT) conditions, whose derivation in this context can be found in [13]. After defying the margin function $h_t(x_i)$ as the difference $f_t(x_i) - y_i$ for all time points $i = 1, \ldots, t$, the KKT conditions can be expressed in terms of $\theta_i, h_t(x_i), \varepsilon$ and $C$. In turn, each data point $(x_i, y_i)$ can be classified as belonging to each one of the following three auxiliary sets,
and an incremental learning algorithm can be constructed by appropriately allocating new data points to these sets [8]. Our learning algorithm is based on this idea, although our definition (20.8) is different. In [13] we argue that a sequential learning algorithm adopting the original definitions proposed by [8] will not always satisfy the KKT conditions, and we provide a detailed derivation of the algorithm for both incremental learning and forgetting of old data points.

In summary, three parameters affect the estimation of the fair price using support vector regression. First, the \( C \) parameter featuring in Eq. (20.4) that regulates the trade-off between model complexity and training error. Second, the parameter \( \epsilon \) controlling the width of the \( \epsilon \)-insensitive tube used to fit the training data. Finally, the \( \sigma \) value required by the kernel. We collect these three user-defined coefficients in the hyperparameter vector \( \phi \). Continuous or adaptive tuning of \( \phi \) would be particularly important for on-line learning in non-stationary environments, where previously selected parameters may turn out to be sub-optimal in later periods. Some variations of SVR have been proposed in the literature (e.g. in [3]) in order to deal with these difficulties. However, most algorithms proposed for financial forecasting with SVR operate in an off-line fashion and try to tune the hyperparameters using either exhaustive grid searches or other search strategies (for instance, evolutionary algorithms), which are very computationally demanding.

Rather than trying to optimize \( \phi \), we take an ensemble learning approach: an entire population of \( p \) SVR experts is continuously evolved, in parallel, with each expert being characterized by its own parameter vector \( \phi^{(e)} \), with \( e = 1, \ldots, p \). Each expert, based on its own opinion regarding the current fair value of the target asset (i.e. an estimate \( z_r^{(e)} \)) generates a binary trading signal of form (20.1), which we now denote by \( d_t^{(e)} \). A meta-algorithm is then responsible for combining the \( p \) trading signals generated by the experts. Thus formulated, the algorithmic trading problem is related to the task of predicting binary sequences from expert advice which has been extensively studied in the machine learning literature and is related to sequential portfolio selection decisions [4]. Our goal is for the trading algorithm to perform nearly as well as the best expert in the pool so far: that is, to guarantee that at any time our meta-algorithm does not perform much worse than whichever expert has made the fewest mistakes to date. The implicit assumption is that, out of the many SVR experts, some of them are able to capture temporary market anomalies and therefore make good predictions.

The specific expert combination scheme that we have decided to adopt here is the Weighted Majority Voting (WMV) algorithm introduced in [7]. The WMV algorithm maintains a list of non-negative weights \( \omega_1, \ldots, \omega_p \), one for each expert, and predicts based on a weighted majority vote of the expert opinions. Initially, all weights are set to one. The meta-algorithm forms its prediction by comparing the total weight
of the experts in the pool that predict 0 (short sell) to the total weight $q_1$ of the algorithms predicting 1 (buy). These two proportions are computed, respectively, as $q_0 = \sum_{e, d_t(e) = 0} \omega_e$ and $q_1 = \sum_{e, d_t(e) = 1} \omega_e$. The final trading decision taken by the WMV algorithm is

$$d_t(\times) = \begin{cases} 0 & \text{if } q_0 > q_1 \\ 1 & \text{otherwise} \end{cases} \tag{20.9}$$

Each day the meta algorithm is told whether or not its last trade was successful, and a $0 - 1$ penalty is applied, as described in Section 20.2. Each time the WMV incurs a loss, the weights of all those experts in the pool that agreed with the master algorithm are each multiplied by a fixed scalar coefficient $\beta$ selected by the user, with $0 < \beta < 1$. That is, when an expert $e$ makes a mistake, its weight is downgraded to $\beta \omega_e$. For a chosen $\beta$, WMW gradually decreases the influence of experts that make a large number of mistakes and gives the experts that make few mistakes high relative weights.

### 20.4 An Application to the iShare Index Fund

Our empirical analysis is based on historical data of an exchange-traded fund (ETF). ETFs are relatively new financial instruments that have exploded in popularity over the last few years. ETFs are securities that combine elements of both index funds and stocks: like index funds, they are pools of securities that track specific market indexes at a very low cost; like stocks, they are traded on major stock exchanges and can be bought and sold anytime during normal trading hours. Our target security is the iShare S&P 500 Index Fund, one of the most liquid ETFs. The historical time series data cover a period of about seven years, from 19/05/2000 to 28/06/2007, for a total of 1856 daily observations. This fund tracks very closely the S&P 500 Price Index and therefore generates returns that are highly correlated with the underlying market conditions. Given the nature of our target security, the explanatory data streams are taken to be a subset of all constituents of the underlying S&P 500 Price Index comprising $n = 455$ stocks, namely all those stocks whose historical data was available over the entire period chosen for our analysis. The results we present here are generated out-of-sample by emulating the behavior of a real-time trading system. At each time point, the system first projects the lastly arrived data points onto a space of reduced dimension. In order to implement this step, we have set $k = 1$ so that only the first eigenvector is extracted. Our choice is backed up by empirical evidence, commonly reported in the financial literature, that the first principal component of a group of securities captures the market factor (see, for instance, [2]). Optimal values of $k > 1$ could be inferred from the streaming data in an incremental way, but we do not discuss this direction any further here.
Table 20.1  Statistical and financial indicators summarizing the performance of the 2560 experts over the entire data set. We use the following notation: SR=Sharpe Ratio, WT=Winning Trades, LT=Losing Trades, MG=Mean Gain, ML=Mean Loss, and MDD=Maximum Drawdown. PnL, WT, LT, MG, ML and MDD are reported as percentages.

<table>
<thead>
<tr>
<th>Summary</th>
<th>Gross SR</th>
<th>Net SR</th>
<th>Gross PnL</th>
<th>Net PnL</th>
<th>Volatility</th>
<th>WT</th>
<th>LT</th>
<th>MG</th>
<th>ML</th>
<th>MDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best</td>
<td>1.13</td>
<td>1.10</td>
<td>17.90</td>
<td>17.40</td>
<td>15.90</td>
<td>50.16</td>
<td>45.49</td>
<td>0.77</td>
<td>0.70</td>
<td>0.20</td>
</tr>
<tr>
<td>Worst</td>
<td>-0.36</td>
<td>-0.39</td>
<td>-5.77</td>
<td>-6.27</td>
<td>15.90</td>
<td>47.67</td>
<td>47.98</td>
<td>0.72</td>
<td>0.76</td>
<td>0.55</td>
</tr>
<tr>
<td>Average</td>
<td>0.54</td>
<td>0.51</td>
<td>8.50</td>
<td>8.00</td>
<td>15.83</td>
<td>48.92</td>
<td>46.21</td>
<td>0.75</td>
<td>0.72</td>
<td>0.34</td>
</tr>
<tr>
<td>Std</td>
<td>0.36</td>
<td>0.36</td>
<td>5.70</td>
<td>5.70</td>
<td>0.20</td>
<td>1.05</td>
<td>1.01</td>
<td>0.02</td>
<td>0.02</td>
<td>0.19</td>
</tr>
</tbody>
</table>

With the chosen grid of values for each one of the three key parameters ($\varepsilon$ varies between $10^{-1}$ and $10^{-8}$, while both $C$ and $\sigma$ vary between 0.0001 and 1000), the pool comprises 2560 experts. The performance of these individual experts is summarized in Table 20.1, which also reports on a number of financial indicators (see the caption for details). In particular, the Sharpe Ratio provides a measure of risk-adjusted return, and is computed as the ratio between the average return produced by an expert over the entire period, divided by its standard deviation. For instance, the best expert over the entire period achieves a promising 1.13 ratio, while the worst expert yields negative risk-adjusted returns. The maximum drawdown represents the total percentage loss experienced by an expert before it starts winning again. From this table, it clearly emerges that choosing the right parameter combination, or expert, is crucial for this application, and relying on a single expert is a risky choice.

Fig. 20.1  Time-dependency of the best expert: each square represents the expert that produced the highest Sharpe ratio during the last trading month (22 days). The horizontal line indicates the best expert overall. Historical window sizes of different lengths produced very similar patterns.

However, even if an optimal parameter combination could be quickly identified, it would soon become sub-optimal. As anticipated, the best performing expert in the pool dynamically and quite rapidly varies across time. This important aspect can be appreciated by looking at the pattern reported in Figure 20.1, which identifies the best expert over time by considering the Sharpe Ratio generated in the last trading month. From these results, it clearly emerges that the overall performance of the
Fig. 20.2 Sharpe Ratio produced by two competing strategies, *Follow the Best Expert* (FBE) and *Majority Voting* (MV), as a function of window size.

Fig. 20.3 Sharpe Ratio produced by *Weighted Majority Voting* (WMV) as a function of the $\beta$ parameter. See Table 20.2 for more summary statistics.

Fig. 20.4 Comparison of profit and losses generated by *Buy-and-Hold* (B&H) versus *Weighted Majority Voting* (WMV), after costs (see the text for details).
system may be improved by dynamically selecting or combining experts. For comparison, we also present results produced by two alternative strategies. The first one, which we call Follow the Best Expert (FBE), consists in following the trading decision of the best performing expert seen so far, where again the optimality criterion used to elect the best expert is the Sharpe Ratio. That is, on each day, the best expert is the one that generated the highest Sharpe Ratio over the last \( m \) trading days, for a given value of \( m \). The second algorithm is Majority Voting (MV). Analogously to WMV, this meta algorithm combines the (unweighted) opinion of all the experts in the pool and takes a majority vote. In our implementation, a majority vote is reached if the number of experts deliberating for either one of the trading signals represents a fraction of the total experts at least as large as \( q \), where the optimal \( q \) value is learnt by the MV algorithm on each day using the last \( m \) trading days. Figure 20.2 reports on the Sharpe Ratio obtained by these two competing strategies, FBE and MV, as a function of the window size \( m \). The overall performance of a simple minded strategy such a FBE falls well below the average expert performance, whereas MV always outperforms the average expert. For some specific values of the window size (around 240 days), MV even improves upon the best model in the pool.

The WMV algorithm only depends upon one parameter, the scalar \( \beta \). Figure 20.3 shows that WMV always consistently outperforms the average expert regardless of the chosen \( \beta \) value. More surprisingly, for a wide range of \( \beta \) values, this algorithm also outperforms the best performing expert by a large margin (Figure 20.3). Clearly, the WMV strategy is able to strategically combine the expert opinion in a dynamic way. As our ultimate measure of profitability, we compare financial returns generated by WMV with returns generated by a simple Buy-and-Hold (B&H) investment strategy. Figure 20.4 compares the profits and losses obtained by our algorithmic trading system with B&H, and illustrates the typical market neutral behavior of the active trading system. Furthermore, we have attempted to include realistic estimates of transaction costs, and to characterize the statistical significance of these results. Only estimated and visible costs are considered here, such as bid-ask spreads and fixed commission fees. The bid-ask spread on a security represents the difference between the lowest available quote to sell the security under consideration (the ask or the offer) and the highest available quote to buy the same security (the bid). Historical tick by tick data gathered from a number of exchanges using the OpenTick provider have been used to estimate bid-ask spreads in terms of base points or bps\(^2\). In 2005 we observed a mean bps of 2.46, which went down to 1.55 in 2006 and to 0.66 in 2007. On the basis of these findings, all the net results presented in Table 20.2 assume an indicative estimate of 2 bps and a fixed commission fee ($10).

Finally, one may tempted to question whether very high risk-adjusted returns, as those generated by WMV with our data, could have been produced only by chance. In order to address this question and gain an understanding of the statistical significance of our empirical results, we first approximate the Sharpe Ratio distribution (after costs) under the hypothesis of random trading decisions, i.e. when sell and buy signals are generated on each day with equal probabilities, using Monte Carlo

\[ \text{bps} = 10000 \left( \frac{a-b}{m} \right) \]

\( a \) is the ask, \( b \) is the bid, and \( m \) is their average.
simulation. Based upon 10,000 repetitions, this distribution has mean $-0.012$ and standard deviation $0.404$. With reference to this distribution, we are then able to compute empirical p-values associated to the observed Sharpe Ratios, after costs; see Table 20.2. For instance, we note that a value as high as 1.45 or even higher ($\beta = 0.7$) would have been observed by chance only in 10 out of 10,000 cases. These findings support our belief that the SVR-based algorithmic trading system does capture informative signals and produces statistically meaningful results.

### Table 20.2 Statistical and financial indicators summarizing the performance of Weighted Majority Voting (WMV) as function of $\beta$. See the caption of Figure 20.1 and Section 20.4 for more details.

<table>
<thead>
<tr>
<th>$\beta$</th>
<th>Gross SR</th>
<th>Net SR</th>
<th>Gross PnL</th>
<th>Net PnL</th>
<th>Volatility</th>
<th>WT</th>
<th>LT</th>
<th>MG</th>
<th>ML</th>
<th>MDD</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>1.34</td>
<td>1.31</td>
<td>21.30</td>
<td>20.80</td>
<td>15.90</td>
<td>53.02</td>
<td>42.63</td>
<td>0.74</td>
<td>0.73</td>
<td>0.24</td>
<td>0.001</td>
</tr>
<tr>
<td>0.6</td>
<td>1.33</td>
<td>1.30</td>
<td>21.10</td>
<td>20.60</td>
<td>15.90</td>
<td>52.96</td>
<td>42.69</td>
<td>0.75</td>
<td>0.73</td>
<td>0.27</td>
<td>0.001</td>
</tr>
<tr>
<td>0.7</td>
<td>1.49</td>
<td>1.45</td>
<td>23.60</td>
<td>23.00</td>
<td>15.90</td>
<td>52.71</td>
<td>42.94</td>
<td>0.76</td>
<td>0.71</td>
<td>0.17</td>
<td>0.001</td>
</tr>
<tr>
<td>0.8</td>
<td>1.18</td>
<td>1.15</td>
<td>18.80</td>
<td>18.30</td>
<td>15.90</td>
<td>51.84</td>
<td>43.81</td>
<td>0.75</td>
<td>0.72</td>
<td>0.17</td>
<td>0.002</td>
</tr>
<tr>
<td>0.9</td>
<td>0.88</td>
<td>0.85</td>
<td>14.10</td>
<td>13.50</td>
<td>15.90</td>
<td>50.03</td>
<td>45.61</td>
<td>0.76</td>
<td>0.71</td>
<td>0.25</td>
<td>0.014</td>
</tr>
</tbody>
</table>

### References