Chapter 1

Data mining for algorithmic asset management: an ensemble learning approach

Giovanni Montana[1] and Francesco Parrella[2]

Abstract Algorithmic asset management refers to the use of expert systems that enter trading orders without any user intervention. In particular, market-neutral systems aim at generating positive returns regardless of underlying market conditions. In this chapter we describe an incremental learning framework for algorithmic asset management based on support vector regression. The algorithm learns the fair price of the security under management by minimining a regularised ε-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 learning in non-stationary environments is addressed by adopting an ensemble learning strategy, 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 temporal data mining system after transaction costs.

1.1 Introduction

Financial asset management is generally executed using active and passive strategies. Active strategies assume that markets are not perfectly efficient and aim to exploit temporary inequilibria among securities. On the contrary, passive strategies are based on the assumptions that the market cannot be beaten in the long run, or that the high costs of active strategies, such as high transaction costs due to frequent readjustments, outweigh the potential benefits.

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 [20]. This is a very general class of investments that seeks to neutralise certain market risks by detecting market inefficiencies and taking offsetting long and short posi-

Imperial College London, Department of Mathematics g.montana@imperial.ac.uk · Imperial College London, Department of Mathematics f.parrella@imperial.ac.uk

tions[1]. The ultimate goal is to achieve positive returns independently of market conditions. Algorithmic trading systems implementing market-neutral strategies have recently gained popularity among professional investors. These systems generate the timing and the size of the orders (e.g. how many shares of a stock to hold on any given day) only on the basis of patterns and trends detected in financial data streams continuously imported through information providers. Given the open availability of electronic APIs (Application Programming Interfaces) and specifically designed communication protocols such as FIX (Financial Information eXchange), these systems communicate directly with the major exchanges and other electronic trading venues and execute orders in a completely autonomous way.

A specific class of market-neutral strategies that heavily relies on temporal data mining is referred to as *statistical arbitrage* [24, 20]. 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, rather than on forecasts of market direction. A simple instance of such relative-value strategies is given by a popular approach called *pairs trading* [35, 9]. 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 generalisation of pairs trading. In our setup, only one of the two dependent assets giving raise 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 utilises 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 and auxiliary 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. The data mining system extracts these dependencies in the form of hidden factors, and tracks them over time. As a result of this process, it will detect possible market inefficiencies, in a way that will be made clear in Section 1.3.

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 [21, 22] and clusters [11, 1]. These fast-paced developments are motivated by the pleatora of temporal data mining applications reaching well beyond the domain of algorithmic trading systems, and including disparate areas such as intrusion and fraud detection, web mining, surveillance video streams and many others. Recent developments include novel database architectures and paradigms such as CEX

[1] Short selling is the practice of borrowing a security from a broker for an immediate sale, with the understanding that it will be bought back later. Profits can be made if the security's price declines.

(Complex Event Processing) that discern patterns in streaming data, from simple correlations to more elaborated queries (see, for instance, [39]).

Notably, the scalability of these data mining methods is being challenged by the tremendous amount of data produced by all such applications, whose volume is increasing at unprecedented speed[2]. 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, without forgetting the previously acquired knowledge and, ideally, without having to access all the data that has been previously stored. To meet these requirements, our algorithmic asset management system build upon incremental algorithms that efficiently process data points as they arrive. In particular, in Section 1.3.3, we deploy a modified version of on-line support vector regression [26, 16] as a powerful function approximation device that can discover non-negligible divergences between the paired assets in real time.

Other than very large data volumes, streaming data are also characterised 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* [10, 22]. 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. This is due to the increased market efficiency experienced in the last few years and resulting from widespread use of algorithmic trading platforms. 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 (see, also, [31, 32]). The precise meaning of an *expert* and how to combine them will be the subject of Section 1.3.4. Experimental results obtained by applying our data mining system to historical data for an exchange-traded fund are presented in Section 1.4.

1.2 Backbone of the asset management system

In this section we outline the rationale behind the asset management 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 (e.g.

[2] For instance, *tick by tick* data are now routinely offered by financial provides for high-frequency trading applications, where a tick refers to a change in a stock's price from one single trade to the next. *Level 2 data* provide an even more comprehensive and in depth characterisation of the trading activity at any moment in time.

just before the markets close). 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.

Specifically, 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 price 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 *misprising*. No further assumptions are made regarding the data generating process. If the markets were always perfectly efficient, we would have that $y_t = z_t$ at all times. Clearly, when $|m_t| > 0$, an arbitrage opportunity arises[3]. 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. Analogously, a positive m_t would suggest to short sell a number of shares, at time t, in accordance with the expectation that the price of Y will be decreasing over the following time interval. Our system is designed to identify and exploit possible statistical arbitrage opportunities of this sort in an automated fashion.

The simple trading strategy above can be formalised 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

$$d_t(m_t) = \begin{cases} 0 & m_t > 0 \\ 1 & m_t < 0 \end{cases} \tag{1.1}$$

where we have made explicit the dependence on the current misprising $m_t = y_t - z_t$. 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

$$L_{t+1}(d_t, r_{t+1}) = |d_t - 1_{(r_{t+1}>0)}| \tag{1.2}$$

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. On the other

[3] Before transaction costs. Costs will be discussed in Section 1.4.

hand, when a daily profit is made (from either a buy or short sell signal), no losses are recorded.

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. The number of data streams imported by the system can be quite large, usually in the order of a few hundred to a few thousand. Although we expect some streams to have high explanatory power, most streams will carry little signal and will mostly contribute to generate noise. Furthermore, when n is large, we expect several streams to be highly correlated over time, and highly dependant streams will provide redundant information. To cope with both of these issues, the system extracts knowledge in the form of a feature vector x_t, dynamically derived from s_t, that captures as much information as possible at each time step. We require for the components of the feature vector x_t to be in number less than n, and to be uncorrelated with each other. Effectively, during this step the systems extracts informative patterns while performing dimensionality reduction.

As soon as the feature vector s_t is extracted, the pattern enters as input of a non-parametric regression model that provides an estimate of the fair price of Y at the current time t. The estimate of z_t is denoted by

$$\hat{z}_t = f_t(x_t;\phi) \tag{1.3}$$

where $f_t(\cdot;\phi)$ is a time-varying and flexible function depending upon the specification of a hyperparameter vector ϕ. With the current $\hat{z}_t$ at hand, an estimated mispricing $\hat{m}_t$ is computed and used to determine the trading rule (1.1). The major difficulty in setting up this learning step lies in the fact that the true fair price z_t is never made available to us, and therefore it cannot be learnt directly. To cope with this problem, we use the observed price y_t as a surrogate for the fair price. As will be made clear in Sections 1.3.2 and 1.3.3, the hyperparameter vector ϕ plays a key role in the fair price estimation driven by (1.3). Proper choices of ϕ can generate sensible estimates $\hat{z}_t$, and therefore realistic mispricing $\hat{m}_t$. In our framework, the quality of the estimated misprisings will be evaluated in term of losses of form (1.2) associated to the trading decision, as well as other financial indicators more directly related to profits and losses generated by the system.

We have now identified a number of practical issues that will have to be addressed next: (a) how to recursively extract and update the feature vector x_t from the the streaming data, (b) how to specify and recursively update the pricing function $f_t(\cdot;\phi)$, and finally (c) how to select the hyperparameter vector ϕ. As anticipated, the key aspect of the proposed system, discussed in Section 1.3.4, is an ensemble learning approach that, effectively, removes the difficult issue of parameter tuning and yields a (nearly) parameter-free asset management system.

1.3 Incremental learning methods

1.3.1 Incremental feature extraction

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 normalised 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 [38] 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 λ_t is the corresponding eigenvalue. Let us call $\widehat{h}_t$ the current estimate of $C_t g_t$ using all the data up to the current time t. This is given by

$$\widehat{h}_t = \frac{1}{t} \sum_{i=1}^{t} s_i s_i^T g_i \tag{1.4}$$

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 $g_t = h_t/||h_t||$ (where $||\cdot||$ indicates the Euclidean norm), an obvious choice is to estimate g_t as $\widehat{h}_{t-1}/||\widehat{h}_{t-1}||$. After plugging in this estimator in (1.4), we obtain

$$\hat{h}_t = \frac{1}{t} \sum_{i=1}^{t} s_i s_i^T \frac{\widehat{h}_{i-1}}{||\widehat{h}_{i-1}||} \tag{1.5}$$

where $\widehat{h}_0$ is initialised by equating it to s_1, the first direction of data spread. Finally, Eq. (1.5) can be rearranged to obtain an equivalent expression that only uses $\widehat{h}_{t-1}$ and the most recent data point s_t, that is

$$\widehat{h}_t = \frac{t-1}{t}\widehat{h}_{t-1} + \frac{1}{t} s_t s_t^T \frac{\widehat{h}_{t-1}}{||\widehat{h}_{t-1}||} \tag{1.6}$$

Observe how the weights $(t-1)/t$ and $1/t$ control the influence of old values in determining the current estimates. Full details related to the computation of the subsequent eigenvectors can be found in [38].

This technique only requires the choice of the total number of eigenvectors to retain, say $k \leq n$. 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. Our specific choice of k will be discussed in Section 1.4.

1.3.2 Asset pricing: the choice of a loss function

We are now given a sequence of paired observations $(y_1,x_1),\dots,(y_t,x_t)$ where each x_t is a k-dimensional feature vector representing the latest market information (i.e. it contains the first k principal components at time t) 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 [19, 18], 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 minimising the following loss function

$$\mathscr{C}_{FLS} = \sum_{i=1}^{t-1}(y_i - w_i^T x_i) + C(w_{i+1} - w_i)^T(w_{i+1} - w_i) \tag{1.7}$$

which leads to a penalised version of ordinary least squares (see, for instance, [12]). Temporal changes in the time-varying linear regression weights w_t result in an additional loss due to the penalty term in (1.7). The severity of this penalty depends upon the magnitude on the regularisation parameter C, which is a non-negative scalar: at one extreme, when C gets very large, (1.7) 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 [19], 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 ε related to the noise level in the data. The corresponding ε-insensitive loss function to be minimised is defined as

$$\mathscr{C}_\varepsilon = |y_t - f_t(x_t)|_\varepsilon = \max(0, |y_t - f_t(x_t)| - \varepsilon) \tag{1.8}$$

and is illustrated in Figure 1.1. Therefore, estimation errors less than ε will simply be ignored, whereas error larger than ε introduce additional linear losses[4].

At the same time, we would also like $f_t(x_t)$ to be as flat as possible. One standard way to ensure this requirement is to impose an additional penalisation parameter controlling the norm of the weights, $||w_t||^2 = w_t^T w_t$. For simplicity of exposition, let us suppose again that the function to be learned is linear and can be expressed as $f_t(x_t) = w_t^T x_t + b_t$, where b_t is a scalar. The generalisation to the non-linear case will be introduced in the following section. Introducing slack variables ξ_t, ξ_t^* quantifying

[4] More general loss functions penalising errors larger than ε in non-linear ways, such as the quadratic ε-sensitive loss, could also be considered (see, for instance, [7]).

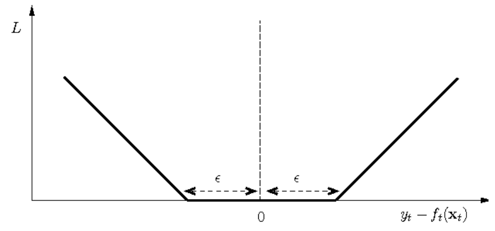

Fig. 1.1 The linear ε-insensitive loss function.

estimation errors greater than ε, the learning task can be casted into the following minimisation problem,

$$\min_{w_t, b_t} \frac{1}{2} w_t^T w_t + C \sum_{i=1}^{t} (\xi_i + \xi_i^*) \tag{1.9}$$

$$s.t. \begin{cases} -y_i + (w_i^T x_i + b_i) + \varepsilon + \xi_i \geq 0 \\ y_i - (w_i^T x_i + b_i) + \varepsilon + \xi_i^* \geq 0 \\ \xi_i, \xi_i^* \geq 0, \; i = 1, \ldots, t \end{cases} \tag{1.10}$$

that is, the support vector regression framework originally introduced by Vapnik [34]. In this optimisation problem, the constant C is a regularization parameter determining the trade-off between the flatness of the function and the tolerated additional estimation error. Again, a linear loss of $|\xi_t| - \varepsilon$ is imposed any time the error $|\xi_t|$ is greater than ε, whereas a zero loss is used otherwise.

Another motivation for considering the ε-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.

1.3.3 The incremental SVR algorithm

Various approaches for incremental learning with support vector machines have been proposed in the literature, for both classification [5, 14] and regression problems [16, 36, 17, 8][5]. Here we describe a modified version of the approach proposed in [16]. Following well established results (see, for instance, [7]), the constrained optimisation problem defined by Eqs. (1.9) and (1.10) can be solved using a Lagrange function,

[5] A handful of real-world applications are slowly appearing, such a real-time prediction of traffic flow [28] and air pollutant levels [37]. To the best of our knowledge, this is the first application of incremental SVR in computational finance and, specifically, algorithmic asset management.

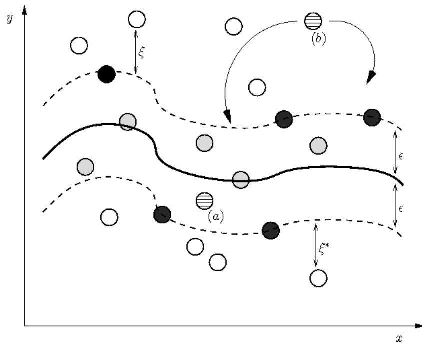

Fig. 1.2 A schematic illustration of the ε-insensitive tube for a non-linear regression function. Each data point, represented by a circle, is classified as belonging to either one of the three sets of Eq. (1.14): the error set $\mathscr{E}$ (grey-filled), the support set $\mathscr{S}$ (black-filled), and the remaining set $\mathscr{R}$ (white-filled).

$$\begin{aligned} L =& \frac{1}{2} w_t^T w_t + C \sum_{i=1}^{t} (\xi_i + \xi_i^*) - \sum_{i=1}^{t} (\eta_i \xi_t + \eta_i^* \xi_i^*) \\ & - \sum_{i=1}^{t} \alpha_i (\varepsilon + \xi_i - y_t + w_t^T x_t + b_t) - \sum_{i=1}^{t} \alpha_i^* (\varepsilon + \xi_i^* + y_t - w_t^T x_t - b_t) \end{aligned} \tag{1.11}$$

where $\alpha_i, \alpha_i^*, \eta_i$ and η_i^* are the Lagrange multipliers, and have to satisfy positivity constraints, for all $i = 1, \ldots, t$. The partial derivatives of (1.11) with respect to w, b, ξ and ξ^* are required to vanish for optimality. By doing so, each η_t can be expressed as $C - \alpha_t$ and therefore can be removed (analogously for η_t^*) . 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

$$f_t(x_t) = \sum_{i=1}^{t} \theta_i x_i^T x_i + b_i \tag{1.12}$$

where each coefficient θ_i has been defined as the difference $\alpha_i - \alpha_i^*$. The dual optimisation problem leads to another Lagrangian function, and its solution is provided by the Karush-Kuhn-Tucker (KTT) conditions, whose detailed derivation in this context can be found in [23]. After defying the *margin function* $h_i(x_i)$ as the difference $f_i(x_i) - y_i$ for all time points $i = 1, \ldots, t$, the KKT conditions can be expressed in terms of $\theta_i, h_i(x_i), \varepsilon$ and C, as follows

$$\begin{array}{ll} h_i(x_i) \geq +\varepsilon & \theta_i = -C \\ h_i(x_i) = +\varepsilon & \theta_i \in [-C, 0] \\ h_i(x_i) \in [-\varepsilon, +\varepsilon] & \theta_i = 0 \\ h_i(x_i) = -\varepsilon & \theta_i \in [0, C] \\ h_i(x_i) \leq -\varepsilon & \theta_i = C \end{array} \tag{1.13}$$

Now, observe that each data point (x_i, y_i) in the training set, is associated with a margin function $h_i(x_i)$ and a parameter vector θ_i. Hence, all the historical data points up to the current time t can be classified as belonging to each one of the following three auxiliary sets,

$$\begin{aligned}
\mathscr{S} &= \{i \mid (\theta_i \in [0, +C] \wedge h_i(x_i) = -\varepsilon)\ \vee (\theta_i \in [-C, 0] \wedge h_i(x_i) = +\varepsilon)\} \\
\mathscr{E} &= \{i \mid (\theta_i = -C \wedge h_i(x_i) \geq +\varepsilon) \vee (\theta_i = +C \wedge h_i(x_i) \leq -\varepsilon)\} \qquad (1.14) \\
\mathscr{R} &= \{i \mid \theta_i = 0 \wedge |h_i(x_i)| \leq \varepsilon\}
\end{aligned}$$

Our definition (1.14) differs from the one suggested in [16], where the three sets above are defined without any reference to the margin function. In [23] we argue that a sequential learning algorithm adopting the original definitions of [16] will not always satisfy the KKT conditions.

Let us suppose that the SVR has been trained with data points till time t, and a sequential update is needed as soon as a new observation pair (x_{t+1}, y_{t+1}) arrives into the system. At that point, a coefficient $\theta_{t+1}^{(j)}$ is assigned to the newly arrived sample, and an iterative procedure starts. The index j refers to the iteration number. Initially, $\theta_{t+1}^{(1)} = 0$. According to (1.14), if the new sample point qualifies as belonging to set $\mathscr{R}$, then no further steps are necessary because the KKT conditions are already satisfied. This case only arises in the first iteration when $|h(x_{t+1})|$ is less than ε, so that the latest data point does not contribute to the solution. This is schematically illustrated in Figure 1.2, where the new data point is denoted by (a). Figure 1.2 also gives a schematic representation illustrating the loss $\pm\xi$ incurred by data points lying outside the *tube* defined by $\pm\varepsilon$. If the condition above is not verified, the algorithm needs to gradually increase or decrease θ_{t+1} in a number of subsequent iterations so that, eventually, all data points (including the new one) will enter either one of the auxiliary sets. This is again illustrated in Figure 1.2, where the new data point is denoted by (b). However, perturbing the newly added parameter $\theta_{t+1}^{(j)}$ will affect all the remaining parameters, which may depart from the optimality established in the previous time point. Fortunately, it is possible to analytically characterise how these changes interact with each other, and search for the step-size that, at each iteration, introduces the smallest perturbation in the KKT conditions till optimality is reached again. Older data points, say older than m days, can also be easily removed by using a decremental version of this approach. In our application, we have set $m = 20$ (a full trading month) although m could be treated as an additional parameter to be tuned over time. Full details relating to the incremental and decremental algorithm can be found in [16], and our modified implementation is described in [23][6].

Finally, the linearity assumption can be released by means of suitable chosen kernel functions (see, for instance, [25]). Essentially, this amounts to replacing $x_t^T x_t$ in (1.12) with a kernel $K(x_i, x_j)$. For this application, we gave used a Gaussian kernel depending upon the specification of a width-parameter σ.

[6] All experiments described in Section 1.4 are based on our C++ implementation of the on-line SVR algorithm. The software will be made available upon request.

1.3.4 An ensemble learning approach

As established in previous sections, three parameters affect the estimation of the fair price using support vector regression. First, the C parameter featuring in Eq. (1.9) that regulates the trade-off between model complexity and training error. Second, the parameter ε controlling the width of the ε-insensitive tube used to fit the training data[7]. Finally, the σ value required by the kernel. We collect these three user-defined coefficients in the hyperparameter vector ϕ first introduced in Eq. (1.3).

Continuous tuning of ϕ 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. However, parameter optimisation based on highly dynamic data streams such as those representing financial instruments is notoriously a difficult task. In order to deal with these problems, adaptive versions of SVR have been proposed in the literature (e.g. in [29, 30, 4]). 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 better search strategies (for instance, evolutionary algorithms [27, 33]), which are very computationally demanding.

Rather than trying to optimise ϕ, we take an ensemble learning approach: an entire population of p SVR *experts* is continuously evolved, in parallel, with each expert being characterised by its own parameter vector $\phi^{(e)}$, with $e = 1, \dots, p$. Each expert, based on its own opinion regarding the current fair value of the target asset, i.e. an estimate $z_t^{(e)}$, generates a binary trading signal of form (1.1), 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 are related to sequential portfolio selection decisions [3, 6]. Since we make no assumptions about the quality or independence of the experts, we cannot hope to achieve any absolute level of quality in our predictions. Our main goal instead is 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. Our only assumption if 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 [15].

The WMV algorithm maintains a list of non-negative weights $\omega_1, \dots, \omega_p$, 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

[7] We note that the value of ε affect the number of support vectors used to construct the regression function. Larger values of ε yields fewer support vectors are selected. On the other hand, larger values of ε also result in flatter regression estimates.

computed, respectively, as

$$q_0 = \sum_{e:d_t^{(e)}=o} \omega_e \ , \ q_1 = \sum_{e:d_t^{(e)}=1} \omega_e$$

and the final trading decision taken by the WMV algorithm is

$$d_t^{(*)} = \begin{cases} 0 & \text{if } q_o > q_1 \\ 1 & \text{otherwise} \end{cases} \tag{1.15}$$

Each day the meta algorithm is told whether or not its last trade was successfull, using the loss function of Eq. (1.2). 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 β selected by the user, with $0 \leq \beta < 1$. That is, when an expert e makes as mistake, its weight is downgraded to $\beta \omega_e$. For a chosen $\beta > 0$, 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.

1.4 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, and are expected to become as common as mutual funds. In brief, ETFs are securities that combine elements of both index funds and stocks: like index funds, ETFs are pools of securities that track specific marker 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 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. On the contrary, an algorithmic system should be able to generate positive returns in *all* 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 (no foresight) by emulating the behaviour 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 documented 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, as in [21], but we do not discuss this direction any further here.

Table 1.1 Statistical and financial indicators summarising 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.

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

With the chosen grid of values for each one of the three key parameters[8], the pool comprises 2560 experts. The performance of these individual experts is summarised in Table 1.1, 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 very promising Sharpe Ratio of 1.13, 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.

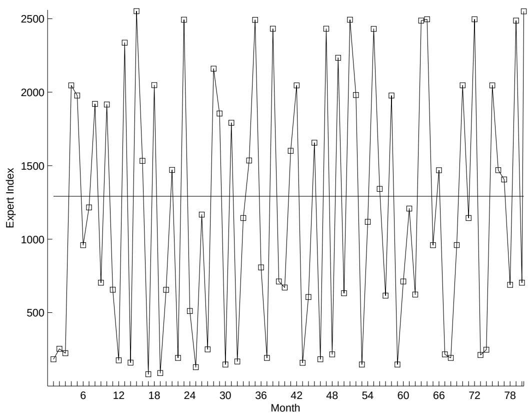

Fig. 1.3 Time-dependency of the best expert: each square represents the expert that produced the highest Sharpe ratio during the last trading month (22 days). The horizontal line indicates the best expert overall. Historical window sizes of different lengths produced very similar patterns.

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

[8] ε varies between 10^{-1} and 10^{-8}, while both C and σ vary between 0.0001 and 1000.

pool dynamically and quite rapidly varies across time. This important aspect can be appreciated by looking at the pattern reported in Figure 1.3, which identifies the best expert over time by looking at the the Sharpe Ratio generated by an expert in the last trading month. More specifically, Figure 1.4 illustrates the time-varying nature of the ε parameter corresponding to the best expert for a randomly chosen subperiod of daily data. From these results, it clearly emerges that the overall performance of the system may be improved by dynamically selecting or combining experts.

For comparison with the *Weighted Majority Voting* (WMV) algorithm of Section 1.3.4, 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 to 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 Share 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 1.5 reports on the Sharpe Ratio obtained by these two competing strategies, FBW 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.

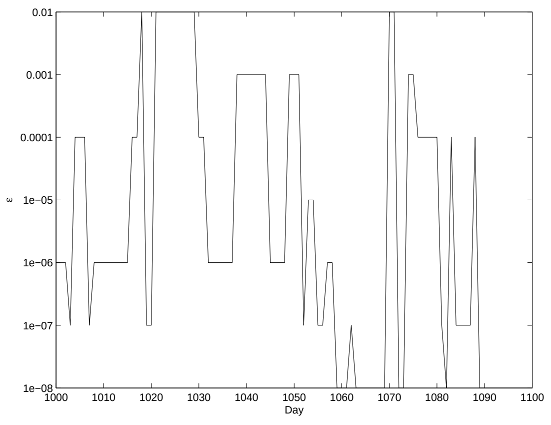

Fig. 1.4 Time-dependency of ε parameter: shown is the ε value corresponding to the best-performing expert for a subset of trading days in the sample.

The WMV algorithm only depends upon one parameter, the scalar β. Figure 1.6 shows that WMV always consistently outperforms the average expert regardless of the chosen β value. More surprisingly, for a wide range of β values, this algorithm also outperforms the best performing expert by a large margin (Figure 1.6). 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 1.7 compares the profits and losses obtained by our algorithmic trading system with B&H, and illustrates the typical market neutral behaviour of the active trading system. Overall, the correlation coefficient between the returns generated by WMV and the market returns is as low as 0.3.

Finally, we have attempted to include realistic estimates of transaction costs, and to characterise the statistical significance of these results. Only estimated and visible costs are considered here [13], 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 (`http://opentick.com`) have been used to estimate bid-ask spreads in terms of base points or *pbs*[9]. 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 1.2 assume an indicative estimate of 2 bps and a fixed commission fee ($10).

An important question to address is the following, how likely is it that the risk-adjusted returns produced by WMV were generate by chance only? In order to gain an understanding of the statistical significance of our 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 simulation. Based upon $10,000$ repetitions, this distribution has mean -0.012 and standard deviation 0.404 (data not shown), and in Table 1.2 we report p-values associated to the observed net Sharpe Ratios and computed with reference to this empirical distribution. For instance, 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 suggest that the data mining system does capture informative signals and produce statistically significant results. Application of our data mining system for active management of other ETFs generated comparable results (not shown here due to space constraints).

Table 1.2 Statistical and financial indicators summarising the performance of Weighted Majority Voting (WMV) as function of β. See the caption of Figure 1.1 and Section 1.4 for more details.

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

[9] A base point is defined as $10000\frac{(a-b)}{m}$, where a is the ask, b is the bid, and m is their average.

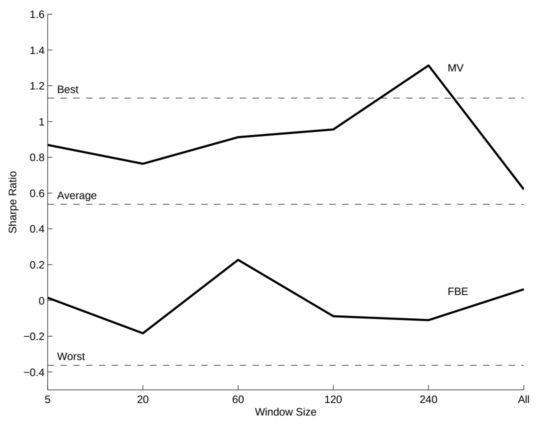

Fig. 1.5 Sharpe Ratio produced by two competing strategies, *Follow the Best Expert* (FBE) and *Majority Voting* (MV), as a function of window size.

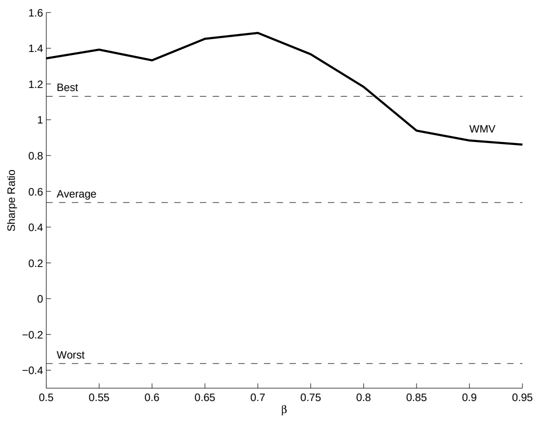

Fig. 1.6 Sharpe Ratio produced by *Weighted Majority Voting* (WMV) as a function of the β parameter. See Table 1.2 for more summary statistics.

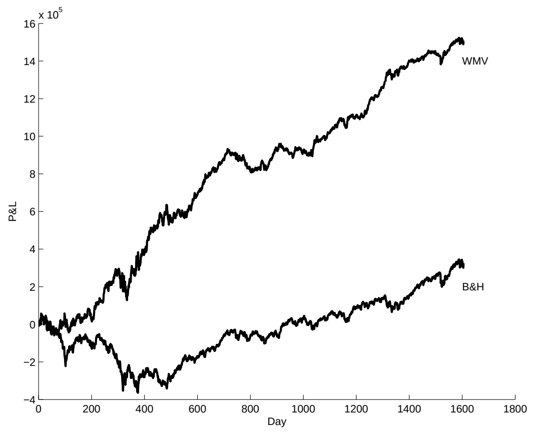

Fig. 1.7 Comparison of profit and losses generated by buy-and-hold (B&H) versus Weighted Majority Voting (WMV), after costs (see the text for details).

1.5 Conclusions

We have presented a temporal data mining system for algorithmic asset management. The system exploits potential price discrepancies between a target security's price and its estimated fair price. The price estimation is performed incrementally by means of on-line principal components and on-line support vector regression, using streaming financial data. Empirical evidence shows that an ensemble learning approach that combines expert opinion produces enhanced results, and provides a viable alternative to the difficult issue of parameter tuning in non-stationary environments. Since every expert deliberates in an autonomous way, the proposed meta algorithm is suitable for parallel implementations in systems with multiple processors.

References

1. C.C. Aggarwal, J. Han, J. Wang, and Yu P.S. *Data Streams: Models and Algorithms*, chapter On Clustering Massive Data Streams: A Summarization Paradigm, pages 9–38. Springer, 2007.
2. C. Alexander and A. Dimitriu. Sources of over-performance in equity markets: mean reversion, common trends and herding. Technical report, ISMA Center, University of Reading, UK, 2005.
3. A. Blum. *Online Algorithms*, chapter On-Line Algorithms in Machine Learning, pages 306–325. Springer Berlin / Heidelberg, 1998.
4. L. Cao and F. Tay. Support vector machine with adaptive parameters in financial time series forecasting. *IEEE Transactions on Neural Networks*, 14(6):1506–1518, 2003.
5. G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. volume 13, pages 409–123. Cambridge, 2001.
6. N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.
7. N. Cristianini and J. Shawe-Taylor. *An introduction to Support Vector Machines*. Cambridge University Press, 2000.
8. N. de Freitas, M. Milo, P. Clarkson, M. Niranjan, and A Gee. Sequential support vector machines. In *Proceedings of the 1999 IEEE Signal Processing Society Workshop*, pages 31 – 40, 1999.
9. R.J. Elliott, J. van der Hoek, and W.P. Malcolm. Pairs trading. *Quantitative Finance*, pages 271–276, 2005.
10. M. Gaber, A. Zaslavasky, and S. Krishnaswamy. *Data Streams: Models and Algorithms*, chapter A Survey of Classification Methods in Data Streams, pages 39–59. Springer, 2007.
11. S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003.
12. R. Kalaba and L. Tesfatsion. The flexible least squares approach to time-varying linear regression. *Journal of Economic Dynamics and Control*, 12(1):43–48, 1988.
13. R. Kissell, M. Glantz, and R. Malamut. A practical framework for estimating transaction costs and developing optimal trading strategies to achieve best execution. *Finance Research Letters*, 1(1):35–46, 2004.
14. P. Laskov, C. Gehl, and S. Kruger. Incremental support vector learning: analysis, implementation and applications. *Journal of machine learning research*, 7:1909–1936, 2006.

15. N. Littlestone and M.K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–226, 1994.
16. J. Ma, J. Theiler, and S. Perkins. Accurate on-line support vector regression. *Neural Computation*, 15:2003, 2003.
17. M. Martin. On-line support vector machine regression. In *13th European Conference on Machine Learning*, 2002.
18. G. Montana, K. Triantafyllopoulos, and T. Tsagaris. Data stream mining for market-neutral algorithmic trading. In *In Proceedings of the ACM Symposium on Applied Computing*, 2008.
19. G. Montana, K. Triantafyllopoulos, and T. Tsagaris. Flexible least squares for temporal data mining and statistical arbitrage. *Expert Systems with Applications*, 2008.
20. J. G. Nicholas. *Market-Neutral Investing: Long/Short Hedge Fund Strategies*. Bloomberg Professional Library, 2000.
21. S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 697 – 708, 2005.
22. S. Papadimitriou, J. Sun, and C. Faloutsos. *Data Streams: Models and Algorithms*, chapter Dimensionality reduction and forecasting on streams, pages 261–278. Springer, 2007.
23. F. Parrella and G. Montana. A note on incremental support vector regression. Technical report, Imperial College London, 2008.
24. A. Pole. *Statistical Arbitrage. Algorithmic Trading Insights and Techniques*. Wiley Finance, 2007.
25. J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
26. A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.
27. R. Stoean, D. Dumitrescu, M. Preuss, and C. Stoean. Evolutionary support vector regression machines. In *Proceedings of the Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2006.
28. H. Su, L. Zhang, and S. Yu. Short-term traffic flow prediction based on incremental support vector regression. In *Third International Conference on Natural Computation*, 2007.
29. F. Tay and L. Cao. ε-descending support vector machines for financial time series forecasting. *Neural Processing Letters*, 15:179–195, 2002.
30. F. Tay and L. Cao. Modified support vector machines in financial time series forecasting. *Neurocomputing*, 48:847861, 2002.
31. N. Towers and N. Burgess. *Developments in Forecast Combination and Portfolio Choice*, chapter A meta-parameter approach to the construction of forecasting models for trading systems, pages 27–44. Wiley, 2001.
32. N. Towers and N. Burgess. *Neural Networks and the Financial Markets*, chapter Learning Trading startegies for Imperfect markets, pages 95–108. Springer, 2002.
33. B. Üstün, W. J. Melssen, M. Oudenhuijzenb, and L.M.C. Buydensa. Determination of optimal support vector regression parameters by genetic algorithms and simplex optimization. *Analytica Chimica Acta*, 544:292305, 2005.
34. V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.
35. G. Vidyamurthy. *Pairs Trading*. Wiley Finance, 2004.
36. W. Wang. An incremental learning strategy for support vector regression. *Neural Processing Letters*, 21:175–188, 2005.
37. W. Wang, C. Men, and W. Lu. Online prediction model based on support vector machine. *Neurocomputing*, 71:550–558, 2008.
38. J. Weng, Y. Zhang, and W. S. Hwang. Candid covariance-free incremental principal component analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):1034–1040, 2003.
39. E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 407 – 418, 2006.