On Developing a Financial Prediction System: Pitfalls and Possibilities

Stefan Zemke STEFAN.ZEMKE@DSV.SU.SE
Stockholm University and Royal Institute of Technology, Department of Computer and System Sciences, Forum 100, 164 40 Kista, Sweden

Abstract

A successful financial prediction system presents many challenges. Some are encountered over again, and though an individual solution might be system-specific, general principles still apply. Using them as a guideline might save time, effort, boost results, as such promoting project's success.

This paper remarks on a prediction system development stemming from author's experiences and published results. The presentation follows stages in a prediction system development: data preprocessing, prediction algorithm selection and boosting, system evaluation – with some commonly successful solutions highlighted.

1. Introduction

Financial prediction presents challenges encountered over again. The paper highlights some of the problems and solutions. A predictor development demands excessive experimentation: with data preprocessing and selection, the prediction algorithm(s), a matching trading model, evaluation and tuning – to benefit from the minute gains, but not fall into over-fitting. The experimentation is necessary since there are no proven solutions, but experiences of others, even failed, can speed the development.

The idea of financial prediction (and resulting riches) is appealing, initiating countless attempts. In this competitive environment, if one wants above-average results, one needs above-average insight and sophistication. Reported successful systems are hybrid and custom made, whereas straightforward approaches, e.g. a neural network plugged to relatively unprocessed data, usually fail (Swingler, 1994).

The individuality of a hybrid system offers chances and dangers. One can bring together the best of many approaches, however the interaction complexity hinders judging where the performance dis/advantage is coming from. This paper provides hints in major steps in a prediction system development based on author's experiments and published results.

The paper assumes some familiarity with machine learning and financial prediction. As a reference one could use (Hastie et al., 2001; Mitchell, 1997), including java code (Witten & Frank, 1999), applied to finance (Deboeck, 1994; Kovalerchuk & Vityaev, 2000). Non-linear analysis (Kantz & Schreiber, 1999), in finance (Deboeck, 1994; Peters, 1991). Ensemble techniques (Dietterich, 2000), in finance (Kovalerchuk & Vityaev, 2000).

2. Data Preprocessing

Before data is fed into an algorithm, it must be collected, inspected, cleaned and selected. Since even the best predictor will fail on bad data, data quality and preparation is crucial. Also, since a predictor can exploit only certain data features, it is important to detect which data preprocessing/presentation works best.

Visual inspection is invaluable. At first, one can look for: trend – if need to remove, histogram – redistribute, missing values and outliers, any regularities. There are financial data characteristics (Mantegna & Stanley, 2000) that differ from the normally-distributed, aligned data assumption in general data mining literature.

Outliers may require different considerations: 1) genuine big changes – of big interest to prediction, such data could even be multiplied to promote recognition; 2) jumps due to change a quality is calculated, e.g. stock splits; all previous data could be re-adjusted or a sin-

gle outlier treated as a missing value 3) outlier regularities could signal a systematic error.

Fat tails – extreme values more likely as compared to the normal distribution – is an established property of financial returns (Mantegna & Stanley, 2000). It can matter in 1) situations, which assume normal distribution, e.g. generating missing/surrogate data w.r.t. normal distribution will underestimate extreme values 2) in outlier detection. If capturing the actual distribution is important, the data histogram can be preferred to parametric models.

Time alignment – same date-stamp data may differ in the actual time as long as the relationship is kept constant. The series originating the predicted quantity sets the time – extra time entries in other series may be skipped, whereas missing in other series may need to be restored. Alternatively, all series could be converted to event-driven time scale, especially for intraday data (Dacorogna et al., 2001).

Missing values dealt with by data mining methods (Han & Kamber, 2001) (Dacorogna et al., 2001). If a miss spoils temporal relationship, restoration is preferable to removal. Conveniently all misses in the raw series are restored for feature derivation, alignment etc., skipping any later instances of undefined values. If data restorations are numerous, test if the predictor picks the inserted bias is advisable.

Detrending removes the growth of a series. For stocks, indexes, and currencies converting into logarithms of subsequent (e.g. daily) returns does the trick. For volume, dividing it by last k quotes average, e.g. yearly, can scale it down.

Noise minimally at price discretisation level is prevalent; especially low volume markets should be dealt with suspicion. Discretisation of series into few (< 10) categories (Gershenfeld & Weigend, 1993) along noise cleaning could be evaluated against predictions quality. Simple cleaning: for each series value, find its nearest neighbor based on surrounding values, and then substitutes the value by an average of the original and those from the neighbors (Kantz & Schreiber, 1999). Other operations limiting noise: averaging, instance multiplication, sampling – mentioned below.

Normalization. Discretization – mapping the original values to fewer (new) ones – e.g. positive to 1 and other to -1 – is useful for noise reduction and for nominal input predictors. Subsequent predictor training with input discretized into decreasing number of values can estimate noise – prediction accuracy could increase (Kohavi & Sahami, 1996) once difference between discretized values exceeds noise, to decline later after rough discretization ignores important data distinctions.

Redistribution – changing the frequency of some values in relation to others – can better utilize available range, e.g. if daily returns were linearly scaled to (-1, 1), majority would be around 0.

Normalization brings values to a certain range, minimally distorting initial data relationships. SoftMax norm increasingly squeezes extreme values, linearly mapping middle, e.g. middle 95% input values could be mapped to [-0.95, 0.95], with bottom and top 2.5% nonlinearly to (-1,-0.95) and (0.95, 1) respectively. Normalization should precede feature selection, as non-normalized series may confuse the process.

Series to instances conversion is required by most learning algorithms expecting as an input a fixed length vector. It can be a delay vector derived from series, a basic technique in nonlinear analysis (Kantz & Schreiber, 1999), $\mathbf{v}_t = (series_t, series_{t-delay}, .., series_{t-(D-1)*delay})$. The $delay$ can be the least giving zero autocorrelation, when applied to the series. Such vectors with the same time index t – coming from all input series – appended give an instance, its coordinates referred to as features or attributes.

Data multiplication can be done on many levels. The frequency of a series can be increased by adding (Fourier) interpolated points (Gershenfeld & Weigend, 1993).

Instances can be cloned with some features supplemented with Gaussian noise, 0-mean, deviation between the noise level already present in the feature/series, and the deviation of that series. This can be useful when only few instances are available for an interesting type, e.g. instances with big return. Such data forces the predictor to look for important characteristics ignoring noise – added and intrinsic. Also, by relatively increasing the number of interesting cases, training will pay more attention to their recognition.

Including more series can increase the number of features. A simple test what to include, is to look for series significantly correlated to the predicted one. More difficult is to add non-numerical series, however, adding a text filter for keywords in news can bring substantial advantage.

Indicators are series derived from others, enhancing some features of interest, such as trend reversal. Over the years traders and technical analysts trying to predict stock movements developed the formulae

(Murphy, 1999), some later confirmed to pertain useful information (R. Sullivan & White, 1999). Indicator feeding into a prediction systems is important due to 1) averaging, thus noise reduction, present in many indicator formulae, 2) providing views of the data suitable for prediction. Common indicators follow.

MA, Moving Average, is the average of past k values up to date. Exponential Moving Average, $EMA_n = weight * series_n + (1 - weight) * EMA_{n-1}$.

Stochastic (Oscillator) places the current value relative to the high/low range in a period: $\frac{series_n - \text{low}(k)}{\text{high}(k) - \text{low}(k)}$, low($k$) – the lowest among the k values preceding n, k often 14 days.

MACD, Moving Average Convergence Divergence, difference of short and long-term exponential moving averages, 8 and 17, or 12 and 26 days used.

ROC, Rate of Change, ratio of the current price to price k quotes earlier, k usually 5 or 10 days.

RSI, Relative Strength Index, relates growths to falls in a period. RSI can be computed as positive changes (i.e. $series_i - series_{i-1} > 0$) sum divided by all absolute changes sum, taking last k quotes; k usually 9 or 14 days.

Sampling. In my experiments with NYSE predictability, skipping 0.5 training instances with lowest weight (i.e. weekly return) enhanced predictions, similarly reported (Deboeck, 1994). The distribution (for returns approximated by lognormal) was such that the lowest-return half constituted only 0.2 of the cumulative return, and lowest 0.75 – 0.5 (Mantegna & Stanley, 2000). The improvement could be due to skipping noise-dominated small changes, and/or bigger changes ruled by a mechanism whose learning is distracted by the numerous small changes. Thus, while sampling, it might be worth under-representing small weight instances, missing value-filled, evident-outlier instances and older ones. The amount of data to train a model can be estimated (Walczak, 2001).

Bootstrap – with repetitions, sampling as many elements as in the original – and deriving a predictor for each such a sample, is useful for collecting various statistics (LeBaron & Weigend, 1994), e.g. performance, also ensemble creation or best predictor selection (e.g. via bumping), however not without limits (Hastie et al., 2001).

Feature selection can make learning feasible, as because of the curse of dimensionality (Mitchell, 1997) long instances demand (exponentially) more data. As always, feature choice should be evaluated together with the predictor, as assuming feature importance because it worked well with other predictors, may mislead.

Principal Component Analysis (PCA) and claimed better for stock data Independent Component Analysis (Back & Weigend, 1998), reduce dimension by proposing a new set of salient features.

Sensitivity Analysis trains predictor on all features and then drops those least influencing predictions. Many learning schemes internally signal important features, e.g. (C4.5) decision tree use them first, neural networks assign highest weights etc.

Heuristic such as hill-climbing or genetic algorithms operating on binary feature selection can be used not only to find salient feature subsets, but also – invoked several times – to provide different sets for ensemble creation.

Predictability assessment allows to concentrate on feasible cases (Hawawini & Keim, 1995). Some tests below are simple non-parametric predictors – prediction quality reflecting predictability, measured, e.g., by standard error to series standard deviation ratio.

Linear methods measure correlation between predicted and feature series – significant non-zero implying predictability (Tsay, 2002). Multiple features can be taken into account by multivariate regression.

Nearest Neighbor (Mitchell, 1997) offers a powerful local predictor. Distracted by noisy/irrelevant features, but if this ruled out, failure suggests that the most that can be predicted are general regularities, e.g. an outcome overall probability.

Entropy measures information content, i.e. deviation from randomness (Molgedey & Ebeling, 2000). This general measure, not demanding big amounts of data, and useful in discretisation or feature selection is worth familiarizing with.

Compressibility – the ratio of compressed to the original sequence length – shows how regularities can be exploited by a compression algorithm (which could be the basis of a predictor). An implementation: series digitized 4-bit values packed in pairs into byte array subjected to Zip compression (Zemke, 2002a).

Detrended Fluctuation Analysis (DFA) reveals long term correlations (self-similarity) even in non-stationary time series (N. Vandewalle & Ph.Boveroux, 1997). DFA is more robust, so recommended to Hurst analysis – a sensitive statistics of cycles, proper interpretation requiring experience (Peters, 1991).

Chaos and Lyapunov exponent test short-term determinism, thus predictability (Kantz & Schreiber, 1999). However, the algorithms are noise-sensitive and require long series, thus conclusions should be cautious.

Randomness tests like chi-square, can assess the likelihood that the observed (digitized) sequence is random. Such a test on patterns of consecutive digits could hint pattern no/randomness.

Non-stationarity test can be implemented by dividing data into parts and computing part i predictability based only on part j data. The variability of the measures (visual inspection encouraged), such as standard deviation, assesses stationarity.

A battery of tests could include linear regression, DFA for long term correlations, compressibility for entropy-based approach, Nearest Neighbor for local prediction, and a non-stationarity test.

3. Prediction Algorithms

Below, common learning algorithms (Mitchell, 1997) are discussed, pointing their features important to financial prediction.

Linear methods not main focus here, are widely used in financial prediction (Tsay, 2002). In my Weka (Witten & Frank, 1999) experiments, Locally Weighted Regression (LWR) – scheme weighting Nearest Neighbor predictions – discovered regularities in NYSE data (Zemke, 2002b). Also, Logistic – nonlinear regression for discrete classes – performed above-average and with speed. As such, regression is worth trying, especially its schemes more specialized to the data (e.g. Logistic to discrete) and as a final optimization – weighting other predictions (LWR).

Neural Network (ANN) – seems the method of choice for financial prediction (Kutsurelis, 1998; Cheng et al., 1996). Backpropagation ANNs present the problems of long training and guessing the net architecture. Schemes training architecture along weights could be preferred (Hochreiter & Schmidhuber, 1997) (Kingdon, 1997), limiting under-performance due to wrong (architecture) parameter choice. Note, a failure of an ANN attempt, especially using a general-purpose package, does not necessitate prediction impossible. In my experiments, Voted Perceptron performance often compared with that of ANN, this could be a start, especially when speed is important, such as in ensembles.

C4.5, ILP – generate decision trees/if-then rules – human understandable, if small. In my experiments with Progol (Mitchell, 1997) – otherwise successful rule-learner – applied to NYSE data, rules (resembling technical) seldom emerged; Weka J48 (C4.5) tree-learner predictions have not performed; GA-evolved rules' performance was very sensitive to 'right' background predicates (Zemke, 1998). The conclusion being that, small rule-based models cannot express certain relationships and perform well with noisy/at times inconsistent financial data (Kovalerchuk & Vityaev, 2000). Ensembles of decision trees, can make up for the problems, but readability is usually lost. Rules can also be extracted from ANN, offering accuracy and readability (Kovalerchuk & Vityaev, 2000).

Nearest Neighbor (NN) does not create a general model, but to predict, it looks back for the most similar case(s) (Mitchell, 1997). Irrelevant/noisy features disrupt the similarity measure, so pre-processing is worthwhile. NN is a key technique is nonlinear analysis which offers insights, e.g. weighting more neighbors, efficient NN search (Kantz & Schreiber, 1999). Cross-validation (Mitchell, 1997) can also decide an optimal number of kNN neighbors. Ensemble/bagging NNs trained on different instance samples usually does not boost accuracy, though on different feature subsets might.

Bayesian classifier/predictor first learns probabilities how evidence supports outcomes, used then to predict new evidence's outcome. Though the simple scheme is robust to violating the 'naive' independent-evidence assumption, watching independence might pay off, especially as in decreasing markets variables become more correlated than usual. The Bayesian scheme might also combine ensemble predictions – more optimally than majority voting.

Support Vector Machines (SVM) are a relatively new and powerful learner, having attractive characteristics for time series prediction (Muller et al., 1997). First, it deals with multidimensional instances, actually the more features the better – reducing the need for (wrong) feature selection. Second, it has few parameters, thus finding optimal settings can be easier, one of the parameters referring to noise level the system can handle.

Performance improvement

Most successful prediction are hybrid: several learning schemes coupled together (Kingdon, 1997; Cheng et al., 1996; Kutsurelis, 1998; Kovalerchuk & Vityaev, 2000). Predictions, indication of their quality, biases, etc., fed into a (meta-learning) final decision layer. The hybrid architecture may also stem from performance improving techniques:

Ensemble (Dietterich, 2000) is a number of predictors of which votes are put together into the final predic-

tion. The predictors, on average, are expected above-random and making independent errors. The idea is that correct majority offsets individual errors, thus the ensemble will be correct more often than an individual predictor. The diversity of errors is usually achieved by training a scheme, e.g. C4.5, on different instance samples or features. Alternatively, different predictor types – like C4.5, ANN, kNN – can be used or the predictor's training can be changed, e.g. by choosing the second best decision, instead of first, building C4.5 decision tree. Common schemes include bagging, boosting and their combinations and Bayesian ensembles (Dietterich, 2000). Boosting is particularly effective in improving accuracy.

Note: an ensemble is not a panacea for non-predictable data – it only boosts accuracy of already performing predictor. Also, readability, efficiency are decreased.

Genetic Algorithms (GAs) (Deboeck, 1994) explore novel possibilities, often not thought of by humans. Therefore, it is always worth keeping some decisions as parameters that can be (later) GA-optimized, e.g., feature preprocessing and selection, sampling strategy, predictor type and settings, trading strategy. GAs (typically) require a fitness function – reflecting how well a solution is doing. A common mistake is to define the fitness one way and to expect the solution to perform another way, e.g. if not only return but also variance are important, both factors should be incorporated into fitness. Also, with more parameters and GAs ingenuity it is easier to overfit the data, thus testing should be more careful.

Local, greedy optimization can improve an interesting solution. This is worth combining with a global optimization, like GAs, which may get near a good solution without reaching it. If the parameter space is likely nonlinear, it is better to use a stochastic search, like simulated annealing, as compared to simple uphill.

Pruning properly applied can boost both 1) speed – by skipping unnecessary computation, and 2) performance – by limiting overfitting. Occam's razor – among equally performing models, simpler preferred – is a robust criterion to select predictors, e.g. Network Regression Pruning (Kingdon, 1997), MMDR (Kovalerchuk & Vityaev, 2000) successfully use it. In C4.5 tree pruning is an intrinsic part. In ANN, weight decay schemes (Mitchell, 1997) reduce towards 0 connections not sufficiently promoted by training. In kNN, often a few prototypes perform better than referring to all instances – as mentioned, high return instances could be candidates. In ensembles, if the final vote is weighted, as in AdaBoost (Dietterich, 2000), only the highest-weighted predictors matter.

Tabu, cache, incremental learning, gene GA can accelerate search, allowing more exploration, bigger ensemble etc. Tabu search prohibits re-visiting recent point again – except for not duplicating computation, it forces the search to explore new areas. Caching stores computationally expensive results for a quick recall, e.g. (partial) kNN can be precomputed. Incremental learning only updates a model as new instances arrive, e.g. training ANN could start with ANN previously trained on similar data, speeding up convergence. Gene expression GAs optimize solution's compact encoding (gene), instead of the whole solution which is derived from the encoding for evaluation.

I use a mixture: optimizing genes stored in a tabu cache (logged and later scrutinized if necessary).

What if everything fails but the data seems predictable? There are still possibilities: more relevant data, playing with noise reduction/discretisation, making the prediction easier, e.g. instead of return, predicting volatility (and separately direction), or instead of stock (which may require company data) predicting index, or stock in relation to index; changing the horizon – prediction in 1 step vs. many; another market, trading model.

Trading model given predictions, makes trading decisions, e.g. predicted up – long position, down – short, with more possibilities (Hellstrom & Holmstrom, 1998). Return is just one objective, other include: minimizing variance, maximal loss (bankruptcy), risk (exposure), trade (commissions), taxes; Sharpe ratio etc. A practical system employs precautions against predictors non-performance: monitoring recent performance and signaling if it is below accepted/historic level. It is crucial in non-stationary markets to allow for market shifts beyond control – politics, disasters, entry of a big player. If the shifts cannot be dealt with, at least should be signaled before inflicting unreparable loss. This touches the subject of a bigger (money) management system, taking the predictions into account while hedging, but it is beyond the scope of this paper.

4. System Evaluation

Proper evaluation is critical to a prediction system development. First, it has to measure exactly the interesting effect, e.g. trading return, as opposed to prediction accuracy. Second, it has to be sensitive enough as to distinguish often minor gains. Third, it has to convince that the gains are no merely a coincidence.

Evaluate the right thing. Financial forecasts are often developed to support semi-automated trading (profitability), whereas the algorithms underlying those systems might have different objective. Thus, it is important to test the system performing in the setting it is going to be used, a trivial, but often missed notion. Also, the evaluation data should be of exactly the same nature as planned for real-life application, e.g. an index-futures trading performed for index data

used as a proxy for futures price, but real futures data degraded it. Some problems with common evaluation strategies (Hellstrom & Holmstrom, 1998) follow.

Accuracy – percentage of correct discrete (e.g. up/down) predictions; common measure for discrete systems, e.g. ILP/decision trees. It values instances equally, disregarding both instance's weight and accuracies for different cases, e.g. a system might get high score predicting the numerous small changes whereas missing the big few. Actually, some of the best-performing systems have lower accuracy than could be found for that data (Deboeck, 1994).

Square error – sum of squared deviations from actual outputs – is a common measure in numerical prediction, e.g. ANN. It penalizes bigger deviations, however if sign is what matters this might not be optimal, e.g. predicting -1 for -0.1 gets bigger penalty than predicting +0.1, though the latter might trigger going long instead of short. Square error minimization is often an intrinsic part of an algorithm such as ANN backpropagation, and changing it might be difficult. Still, many such predictors, e.g. trained on bootstrap samples, can be validated according to the desired measure and the best picked.

Reliability – predictor's confidence in its forecast – is equally important and difficult to develop as the predictor itself (Gershenfeld & Weigend, 1993). A predictor will not always be confident – it should be able to express this to the trading counterpart, human or not. e.g. by an output 'undecided'. No trade on dubious predictions is beneficial in many ways: lower errors, commissions, exposure. In my experiments optimizing the reliability requirement, stringent values emerged – why to trade if the predicted move and confidence are low? Reliability can be assessed by comparing many predictions: coming from an ensemble, as well as done in one step and multiple steps fashion.

Performance measure (Hellstrom & Holmstrom, 1998) should incorporate the predictor and the (trading) model it is going to benefit. Some points: Commissions need to be incorporated – many trading 'opportunities' exactly disappear with commissions. Risk/variability – what is the value of even high return strategy if in the process one gets bankrupt? Data difficult to obtain in real time, e.g. volume, might mislead historic data simulations.

Evaluation bias resulting from the evaluation scheme and time series data, needs to be recognized. Evaluation similar to the intended operation can minimize performance estimate bias, though different tests can be useful to estimate different aspects, such as return, variance.

N-cross validation – data divided into N disjoint parts, $N-1$ for training and 1 for testing, error averaged over all N (Mitchell, 1997) – in the case of time series data, underestimates error. Reason: in at least $N-2$ out of the N train-and-test runs, training instances precede and follow the test cases unlike in actual prediction when only past in known. For series, window approach is more adept.

Window approach – segment ('window') of consecutive instances used for training and a following segment for testing, the windows sliding over all data, as statistics collected. Often, to save training time, the test segment consists of many instances. However, more than 1 instance overestimates error, since the training window does not include the data directly preceding some tested cases. Since markets undergo regime change in matter of weeks, the test window should be no longer than that, or the train window's fraction ($< 20\%$). To speed up training for the next test window, the previous window predictor could be used as the starting point while training on the next window, e.g. instead of starting with ANN random weights.

Evaluation data should include different regimes, markets, even data errors, and be plentiful. Dividing test data into segments helps to spot performance irregularities (for different regimes).

Overfitting a system to data is a real danger. Dividing data into disjoint sets is the first precaution: training, validation for tuning, and test set for performance estimation. A pitfall may be that the sets are not as separated as seem, e.g. predicting returns 5 days ahead, a set may end at day D, but that instance may contains return for day $D+5$ falling into a next set. Thus data preparation and splitting should be careful.

Another pitfall is using the test set more than once. Just by luck, 1 out of 20 trials is 95% above average, 1 out of 100, 99% above etc. In multiple test, significance calculation must factor that in, e.g. if 10 tests are run and the best appears 99.9% significant, it really is $99.9\%^{10} = 99\%$ (Zemke, 2000).

Multiple use can be avoided, for the ultimate test, by taking data that was not available earlier. Another possibility is to test on similar, not tuned for, data – without any tweaking until better results, only with predefined adjustments for the new data, e.g. switching the detrending preprocessing on.

Surrogate data is a useful concept in nonlinear system evaluation (Kantz & Schreiber, 1999). The idea is to generate data sets sharing characteristics of the original data – e.g. permutations of series have the same mean, variance etc. – and for each compute an interesting statistics, e.g. return of a strategy. To compare the original series statistics to those of the surrogates, there are 2 ways to proceed: 1) If the statistics is normally distributed, the usual one/two-sided test comparing to the surrogates' mean used. 2) If no such assumption, the nonparametric rank test can be used:

If α is the acceptable risk of wrongly rejecting the null hypothesis that the original series statistics is lower (higher) than of any surrogate, then $1/\alpha - 1$ surrogates needed; if all give higher (lower) statistics than the original series, then the hypothesis can be rejected. Thus, if predictor's error was lower on original series, than in 19 runs on surrogates, we can be 95% sure it is up to something.

Non/Parametric tests. Most statistical tests (Hastie et al., 2001) (Efron & Tibshirani, 1993) have preconditions. They often involve assumptions about sample independence and distributions – unfulfilled leading to unfounded conclusions. Independence is tricky to achieve, e.g. predictors trained on overlapping data are not independent. If the sampling distribution is unknown, as it usually is, it takes least 30, better 100, observations for normal distribution statistics.

If the sample is smaller than 100, nonparametric test are preferable, with less scope for assumption errors. The backside is they have less discriminatory power – for the same sample size (Heiler, 1999).

A predictor should significantly win (nonparametric) comparisons with naive predictors: 1) Majority predictor outputs the commonest value all the time, for stocks it could be the dominant up move, translating into the buy and hold strategy. 2) Repeat previous predictor for the next value issues the (sign of the) previous one.

Sanity checks involve common sense (Gershenfeld & Weigend, 1993). Prediction errors along the series should not reveal any structure, unless the predictor missed something. Do predictions on surrogate (permuted) series discover something? If valid, this is the bottom line for comparison with prediction on the original series – is it significantly better?

Putting it all together

To make the paper's less abstract, some author's choices in a NYSE index prediction system follow. The research (Zemke, 2002b) extends an earlier system (Zemke, 1998). The idea is to develop a 5-days return predictor, later on, to support a trading strategy.

Data used consists of 30 years of daily NYSE 5 indexes and 4 volume series. Data is plotted and some series visibly mimicking other omitted. Missing values are filled by a nearest neighbor algorithm, and the 5-days return series to be predicted computed. The index series are converted to logarithms of daily returns; the volumes divided by lagged yearly averages. Additional series are derived, depending on experiment, 10 and 15 days MA and ROC for indexes. Then all series are Softmax normalized to -1..1 and discretized to 0.1 precision. In between major preprocessing steps series statistics are computed: number of NaN, min and max values, mean, st. deviation, 1,2-autocorrelation, zip-compressibility, linear regression slope, DFA – tracing if preprocessing does what expected – removing NaN, trend, outliers, but not zip/DFA predictability. In the simplest approach, all series are then put together into instances with $D = 3$ and $delay = 2$. An instance's weight is corresponding time absolute 5-days return and instance's class – the return's sign.

The predictor is one of Weka (Witten & Frank, 1999) classifiers handling numerical data, 4-bit coded into a binary string together with: which instance's features to use, how much past data to train on (3, 6, 10, 15, 20 years) and what part of lowest weight instances to skip (0.5, 0.75, 0.85). Such strings are GA-optimized, with already evaluated strings cached and prohibited from costly re-evaluation. Evaluation: a predictor is trained on past data and used to predict values in a disjoint window, 20% size of the data, ahead of it; repeated 10 times with the windows shifted by the smaller window size. The average of the 10 period returns less the 'always up' return and divided by the 10 values st. deviation give a predictor's fitness.

5. Final Remarks

Financial markets, as described by multidimensional data presented to a prediction/trading system, are complex nonlinear systems – with subtleties and interactions difficult for humans to comprehend. This is why, once a system has been developed, tuned and proven performing on (volumes of) data, there is no space for human 'adjustments', except for going through the whole development cycle. Without stringent re-evaluation performance is likely hurt.

A system development usually involves a number of recognizable steps: data preparation – cleaning, selecting, making data suitable for the predictor; prediction algorithm development and tuning – for performance on the quality of interest; evaluation – to see if indeed the system performs on unseen data. But since financial prediction is very difficult, extra insights are needed. The paper has tried to provide some: data enhancing techniques, predictability tests, performance improvements, evaluation hints and pitfalls to avoid. Awareness of them hopefully will make predictions easier, or at least the realization that they cannot be done quicker.

References

Back, A., & Weigend, A. (1998). A first application of independent component analysis to extracting structure from stock returns. *Int. J. on Neural Systems, 8(4)*, 473–484.

Cheng, W., Wagner, L., & Lin, C.-H. (1996). Forecasting the 30-year u.s. treasury bond with a system of neural networks.

Dacorogna, M., Gencay, R., Muller, U., Olsen, R., & Pictet, O. (2001). *An introduction to high-frequency finance*. Academic Press.

Deboeck, G. (1994). *Trading on the edge*. Wiley.

Dietterich, T. G. (2000). Ensemble methods in machine learning. *Multiple Classifier Systems* (pp. 1–15).

Efron, B., & Tibshirani, R. (1993). *An introduction to the bootstrap*. Chapman & Hall.

Gershenfeld, N., & Weigend, S. (1993). *The future of time series: Learning and understanding*. Addison-Wesley.

Han, J., & Kamber, M. (2001). *Data mining. concepts and techniques*. Morgan Kaufmann.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning. data mining, inference and prediction*. Springer.

Hawawini, G., & Keim, D. (1995). *On the predictability of common stock returns: World-wide evidence*. North Holland.

Heiler, S. (1999). A survey on nonparametric time series analysis.

Hellstrom, T., & Holmstrom, K. (1998). *Predicting the stock market* (Technical Report). Univ. of Umeøa, Sweden.

Hochreiter, S., & Schmidhuber, J. (1997). Flat minima. *Neural Computation, 9*, 1–42.

Kantz, H., & Schreiber, T. (1999). *Nonlinear time series analysis*. Cambridge Univ. Press.

Kingdon, J. (1997). *Intellignet systems and financial forecasting*. Springer.

Kohavi, R., & Sahami, M. (1996). Error-based and entropy-based discretization of continuous features. *Proc. of Second Itn. Conf. on Knowledge Discovery and Data Mining* (pp. 114–119).

Kovalerchuk, B., & Vityaev, E. (2000). *Data mining in finance: Advances in relational and hybrid methods*. Kluwer Academic.

Kutsurelis, J. (1998). Forecasting financial markets using neural networks: An analysis of methods and accuracy.

LeBaron, B., & Weigend, A. (1994). Evaluating neural network predictors by bootstrapping. *Proc. of Itn. Conf. on Neural Information Processing*.

Mantegna, R., & Stanley, E. (2000). *An introduction to econophysics: Correlations and complexity in finance*. Cambridge Univ. Press.

Mitchell, T. (1997). *Machine learning*. McGraw Hill.

Molgedey, L., & Ebeling, W. (2000). *Local order, entropy and predictability of financial time series* (Technical Report). Institute of Physics, Humboldt-University Berlin, Germany.

Muller, K.-R., Smola, A., Rtsch, G., Schlkopf, B., Kohlmorgen, J., & Vapnik, V. (1997). Using support vector machines for time series prediction.

Murphy, J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Prentice Hall.

N. Vandewalle, M., & Ph.Boveroux (1997). Detrended fluctuation analysis of the foreign exchange markets. *Proc. Econophysics Workshop, Budapest*.

Peters, E. (1991). *Chaos and order in the capital markets*. Wiley.

R. Sullivan, A. T. A., & White, H. (1999). Data-snooping, technical trading rule performance and the bootstrap. *J. of Finance*.

Swingler, K. (1994). *Financial prediction, some pointers, pitfalls and common errors* (Technical Report). Centre for Cognitive and Computational Neuroscience, Stirling Univ., UK.

Tsay, R. (2002). *Analysis of financial time series*. Wiley.

Walczak, S. (2001). An empirical analysis of data requirements for financial forecasting with neural networks.

Witten, I., & Frank, E. (1999). *Data mining: Practical machine learning tools and techniques with java implementations*. Morgan Kaufmann.

Zemke, S. (1998). Nonlinear index prediction. *Physica A, 269*, 177–183.

Zemke, S. (2000). Rapid fine tuning of computationally intensive classifiers. *Proceedings of AISTA, Australia*.

Zemke, S. (2002a). Compression as a quick measure of predictability. *Submitted*.

Zemke, S. (2002b). Weka for financial prediction. *Submitted*.