Evolutionary Data Selection for Enhancing Models of Intraday Forex Time Series

Michael Mayo

University of Waikato, Hamilton, New Zealand
mmayo@waikato.ac.nz
WWW home page: http://www.cs.waikato.ac.nz/~mmayo/

Abstract. The hypothesis in this paper is that a significant amount of intraday market data is either noise or redundant, and that if it is eliminated, then predictive models built using the remaining intraday data will be more accurate. To test this hypothesis, we use an evolutionary method (called Evolutionary Data Selection, EDS) to selectively remove out portions of training data that is to be made available to an intraday market predictor. After performing experiments in which data-selected and non-data-selected versions of the same predictive models are compared, it is shown that EDS is effective and does indeed boost predictor accuracy. It is also shown in the paper that building multiple models using EDS and placing them into an ensemble further increases performance. The datasets for evaluation are large intraday forex time series, specifically series from the EUR/USD, the USD/JPY and the EUR/JPY markets, and predictive models for two primary tasks per market are built: intraday return prediction and intraday volatility prediction.

Keywords: intraday, forex, steady state, genetic algorithm, instance selection, data mining, return prediction, volatility prediction

1 Introduction

This paper is concerned with using evolutionary algorithms to strengthen predictive models of market behaviour, in order to make them more robust to the noise typical in financial time series (especially intraday series). A new algorithm, Evolutionary Data Selection (EDS), is introduced, that uses a model building algorithm in conjunction with the available training data to find an optimal subset of that data. The search for the optimal subset is evolutionary, and the fitness measure used is the performance of the model built from the subset when tested against all the training data *not* included in the subset (which does not, of course, include the final testing data). At the conclusion of the search, the ultimate best subset is used to build the final predictive model. It is hypothesised that this method is helpful in eliminating noise and repetition from the training data that would otherwise confound the prediction models.

To test the algorithm, we have focussed on intraday (specifically, hourly) time series. The choice of intraday rather than daily series was made primarily because smaller time frames increase the number of samples with which to

test predictions; therefore the statistics should have greater confidence, and also smaller predictive gains (from a trader's point of view) become much more significant intraday when there are multiple opportunities to trade compared to a daily time frame where there would be only a single opportunity to trade per day. It is also known that intraday series often exhibit non-zero autocorrelations (see, for example, Sewell [9] who gives an overview and Breedon [1] who has studied start and end of day effects in the forex markets). The aim of this research is to build models that attempt to capture these small but repeatedly measurable effects, as well as other more general intraday trends.

The evaluation of the approach is carried out against three different forex datasets. The markets used are the Eurodollar (EUR/USD), the Japenese Yen market (USD/JPY), and the Euro/Yen cross (EUR/JPY), with intraday price data ranging from 2002 to 2011. There are two prediction tasks per market: *return prediction*, which attempts to model the direction of the market over the near hour, and *volatility prediction*, which models the square of the return. Both tasks are important for understanding the markets.

It should be noted that the concept of subset selection from the training data is not new; in fact, there are several methods that use a mixture of nearest neighbour and evolutionary techniques to achieve this (and Cano [3] gives an overview in the context of general data mining). Most recently, in the financial modelling field, Larkin and Ryan [5] proposed a method that uses one evolutionary computation method (specifically, a genetic algorithm) for subset selection from the training data, and a second different method (genetic programming) for building the actual prediction model given the optimal subset. The work described here differs from that work in two main ways. Firstly, evolutionary methods are used here only for selecting the training data, wheras a machine learning techniques are utilized for making the predictions. This leverages the strength of disparate approaches with a single system and should increase robustness. Secondly, Larkin and Ryan's approach also extends the selection of cases to the test set; this results in a considerable variation in the number of trades for different strategies (from a lower limit of 30 to in the order of thousands of trades). In contrast, the models developed here are used to predict *every* data point in the test set and therefore we expect that whilst the overall average accuracy may be lower, the results should be more significant and comparable due to a much higher number of samples.

A third and final main point of difference between this work and others is that besides the basic EDS method described here, we also propose an ensemble method in which multiple runs of EDS are combined to a produce a single predictor. This is shown to further enhance accuracy.

2 Evolutionary Data Selection Algorithm

The EDS algorithm described here is derived from the well known steady-state genetic algorithm [6] in which a fixed population P of individuals is evolved. Each

individual $p \in P$ has a fitness value $F(p)$, and the aim is to find the individual p^* that maximises F.

In the steady-state approach, each time step involves two "tournaments" in which T random individuals from P are selected. In each tournament, one winner is selected, the winner being the individual in the tournament with the highest fitness. Because two tournaments are performed per time step, there will be two winners, p_1 and p_2, which are then crossed-over to produce a new offspring o. This offspring may also be mutated with probability M.

After an offspring is generated, its fitness value is computed. If $F(o)$ exceeds either $F(p_1)$ or $F(p_2)$, then o replaces the parent with lowest fitness. In this way, evolution proceeds stochastically towards a population of higher fitness individuals. If no global improvements are made after N time steps, then the search finishes and the individual p^* with the highest fitness $F(p^*)$ is returned.

Clearly, the selection pressure can be adjusted by increasing or decreasing the T parameter: a smaller value of T will give weaker-fitness individuals a greater probability of producing offspring, whereas a higher T will result in only the strongest individuals being consistently selected. The choice of T needs to be made carefully, as too high a value may result in a greater chance of the GA converging on a local rather than a global or near-global maxima, and too low a value may unnecessarily prolong the search.

The specifics of how the individuals are represented, along with the fitness function, the crossover and the mutation operators, are now described.

Individual Representation Each individual is represented as a dataset of *instances*, where an instance is defined as a vector of features that includes a single class value to predict. In this work, the datasets are subsets of the training data that is being used to build a predictive model. When the GA is initialised, the initial individuals are simply random subsets of the training data.

An additional parameter I is furthermore defined that governs the size of the initial subsets: each instance from the training data is included in an initial individual with probability I. Thus, the smaller the value of I, the smaller the size of the initial subsets/individuals; conversely the larger the value of I, the greater the size of the starting subsets/individuals.

Canonically speaking, this representation corresponds to a bit string of length equal to the number of instances in the data, where a "1" indicates the presence of an instance in the set, and a "0" the absence. (It was not efficient, however, to use this representation directly in the implementation.)

Section 3.1 discusses the exact datasets and instances that were generated in the experiments.

Fitness Function Once an individual is defined, a model can then be built using only the instances in the subset as training data. The fitness of the individual is therefore defined as the model's accuracy, once the model is built, on those instances *not* included in the subset of selected instances. In other words, the unselected instances from the training data are used as proxy test data.

Crossover Operator A very simple crossover operator is implemented. Since both parents are defined as sets of instances, we assign to the offspring each instance (excluding duplicates) from the parents with probability 50%. This ensures that, on average, offspring will be about the same size as the parents.

Mutation Operator Mutation, if activated with probability M after an offspring is generated, removes a random 1% of the instances from the set. This method of mutation was designed because we were interested minimising the quantity of data used to train the model. Minimal data has several benefits, including simpler models and faster training times.

Other approaches allow subsets to grow as well as reduce in size, but include a term biasing for small subset sizes in the fitness function. This leads to a more complex fitness function with multiple components (i.e. accuracy as well as subset size) which must somehow then be combined, usually via the introduction of yet another parameter. In contrast, the opposite approach taken here starts with subsets as large as possible (defined by the I parameter) and gradually reduces their size via mutations. Thus, maximum subset size is strictly controlled.

Another reason for this approach is that it is important to keep the test sets large: if the test sets become too small (which can happen if an individual becomes too large), then random variations in the test instances will have greater impact on the accuracy and therefore the fitness, leading to weak individuals being scored unjustly highly.

Note that both crossover and mutation will never remove all the instances from an individual; subsets have a hard-coded minimum size of one.

3 Evaluation

In this section, an evaluation of the EDS method is described. We used data from three forex markets in this evaluation: EUR/USD, the USD/JPY and the EUR/JPY markets, and there were two basic prediction tasks: hourly return forecasts, and hourly volatility forecasts. In all the experiments, the EDS parameters are population size $P = 20$, no improvement limit $N = 200$, mutation rate M of 5%, tournament size $T = 2$ and initial subset size $I = 0.5$.

3.1 Data Preparation

The raw data for our evaluations are cleaned intraday forex time series for the three markets obtained from the Pi Trading Corporation [8]. Each series consists of open, high, low and closing prices for each *minute* that the given market was open where the price changed (i.e. there are no records where price did not change for the current minute), from 21 October 2002, 2am (EST), through to 18 March 2011, 5pm (EST). The total dataset sizes, therefore, are: 2,880,844 records for the EUR/USD market; 2,903,531 records for the USD/JPY market; and 3,009,328 records for the EUR/JPY market.

This minute time-scale data was then processed into hourly data series, by combining all records from the same hour into a single record giving the open, high, low and closing prices of a market for the entire hour. Again, hours where there was no change in price (and therefore no minute records) were excluded. This reduced the number of records by a factor of about 60, making the data much more wieldy for further investigation.

The hourly time series was then converted into a series of instances. One instance was produced for every hourly record in our processed series. The features used for every instance are defined as follows:

- the 24 previous hourly returns $rtn_{h-23}..rtn_h$
- the 24 previous hourly volatilities $vol_{h-23}..vol_h$
- the nominal day of week (a value from $0...6$)
- the nominal hour of the day, (a value from $0...23$)
- a class value to predict

The return and volatility values are defined by Equations 1 and 2. Return is normalized against the opening price for the hour in order to account for different price scales, and the scaling factor of 100.0 is applied to turn the values into percentages. Note that rtn_h is directional and can be positive or negative, whereas vol_h is never negative.

$$rtn_h = 100.0 \times \frac{close_h - open_h}{open_h} \tag{1}$$

$$vol_h = (rtn_h)^2 \tag{2}$$

The class value to predict was defined dependent on the prediction task. Recall that there are two prediction tasks: return prediction and volatility prediction. In the case of return prediction, that aim is to predict the *sign* of the return (positive or negative) of the market over the next hour; in the case of volatility prediction, the purpose is to predict whether volatility will increase or decrease over the next hour.

Let rtn_{h+1} and vol_{h+1} be the return and volatility of the next hour. Then the class variable for the prediction classes can be defined by Equations 3 and 4 respectively.

$$class_{h,rtn} = \begin{cases} 1 & \text{if } rtn_{h+1} \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

$$class_{h,vol} = \begin{cases} 1 & \text{if } vol_{h+1} > vol_h \\ -1 & \text{otherwise} \end{cases} \tag{4}$$

There are no instances generated for any hour where the price does not change. That is, if the open, high, low and close price for an hour are all equal, then that hour is skipped, and the next instance will represent the next hour in which there is a price change.

Table 1. Predictive models used in the experiments.

Model	Description
0R	Simple majority class classifier
RT-1	Single node decision tree, built using the REPTree algorithm [4] with depth limited to 1
RT	Full decision tree, built using the REPTree algorithm [4]
SMO	Sequential Minimal Optimization algorithm [7], a support vector machine learner with a linear kernel
RF	Random Forest algorithm [2] with an ensemble size of 10 random trees

After following this process, three datasets consisting of 52,092; 52,071; and 52,102 instances for the EUR/USD, USD/JPY and EUR/JPY markets respectively were generated. Each dataset comprised 50 features and a class variable, and two versions were generated for each market, one for each of the prediction classes. This resulted in six different versions of the three datasets.

The next step in data preparation was to split the data into training and testing portions. Because financial time series are known to be non-stationary, it cannot be expected that the distribution of features and the relationship between the features and the class should be uniform over the entire period (from 2002 to 2011) covered by the data. To account for this, each dataset was divided into five 10,000 instance "segments", with the final 2,000 or so instances being discarded. The segments were maintained in chronological order, and it was decided that models should be evaluated in not one but four "sliding window" experiments. That is, rather than training a model on the first 50% of the data and testing on the following 50% of data, the better approach of training the model on the first segment (in chronological order) then testing on the second segment; training the model on the second segment and testing on the third; and so on, was utilized. This allowed four evaluations per prediction algorithm, and the resulting four accuracies could then be averaged to produce a more reliable overall estimate.

To summarise, six datasets were produced (two per market, for two different prediction tasks), and each dataset was divided into four train/test experiments. This meant that there would be $6 \times 4 = 24$ experiments per predictive model.

3.2 Base Classifiers

We considered a number of predictive models in our experiments, ranging from the very simple to the state-of-the-art. Two versions of each predictive model were considered: one version *without* the EDS algorithm (the control version); and another version *with* the EDS algorithm (the experimental condition).

The specific predictive models we chose are shown in Table 1. We used implementations of these models from WEKA version 3.7.3 [4], and all settings other than those specified in the Table are WEKA defaults.

The choices of the models can be explained as follows. The 0R model is simple and ignores the features in the datasets; instead, it simply predicts the

Table 2. Performance of models with and without EDS for the return prediction task, by dataset.

	EUR/USD	USD/JPY	EUR/JPY	Average
0R	51.41	50.68	50.20	50.76
RT-1	51.57	50.58	50.50	50.88
EDS+RT-1	51.95	50.75	**50.72**	51.14
RT	51.61	50.41	50.29	50.77
EDS+RT	51.97	**51.06**	50.64	**51.22**
SMO	52.09	50.74	50.32	51.05
EDS+SMO	**52.15**	51.04	50.32	51.17
RF	51.40	50.79	50.20	50.80
EDS+RF	51.12	50.49	50.60	50.74

majority class from the training data. This was chosen as a baseline with which to compare the other classifiers: any model should in theory perform better than 0R if the features are informative.

The RT-1 model is a single level decision tree, otherwise known as a stump, because it selects only a single feature with which to make a prediction. If RT-1 turns out to be the best model for any prediction task, then this finding implies that only one feature (e.g. hour of the day, or previous hour's return) is in fact relevant and the other features are noise. Thus, the performance of RT-1 can be thought of as a second, stronger, baseline for comparison. On the other hand, the RT model is a full decision tree: if RT performs better than RT-1, it implies that *more* than one feature is important, and that patterns in the data exist.

Decision tree models like RT and RT-1 are important because they are interpretable models, which means that humans can inspect them and determine the rules being used to make the predictions. This is important for traders and scientists interested in understanding markets.

The remaining algorithms in the Table, namely RF and SMO, however, are not interpretable: the former is an ensemble of decision trees, and the latter is a mathematical function based on the "support vectors" that it finds in the training data. These algorithms represent two of the state-of-the-art methods in machine learning and are included in our experiments because they should, empirically speaking, give the most accurate results.

3.3 Experiment 1: EDS Prediction

In this experiment, the performance of each prediction model described in Table 1 was compared with and without the EDS algorithm. Recall that each dataset was divided into five 10,000 hour segments, and each model was trained and tested four times. Thus, the results shown in Tables 2 and 3 represent an average over all four experiments.

Starting with results of the return prediction task (Table 2), a number of observations can be made.

Table 3. Performance of models with and without EDS for the volatility prediction task, by dataset.

	EUR/USD	USD/JPY	EUR/JPY	Average
0R	50.78	50.48	50.20	50.49
RT-1	70.30	71.86	70.55	70.90
EDS+RT-1	**70.32**	71.89	**70.69**	**70.97**
RT	70.21	70.47	69.90	70.19
EDS+RT	70.14	**71.94**	70.64	70.91
SMO	56.09	64.41	56.96	59.15
EDS+SMO	61.46	55.37	63.50	60.11
RF	68.26	68.49	67.81	68.18
EDS+RF	68.46	69.44	67.91	68.60

Firstly, the EDS algorithm boosts the performance of every model in nearly every case. This is an encouraging result showing that the inituition behind the method is suitable for intraday financial data.

Secondly, return prediction on a "next hour" basis is very difficult; most of the models achieve only a very small excess over 50%, which would be expected by random guessing. However, they do *consistently* achieve more than 50%, which does imply that there is at least some kind of pattern to learn.

It should be noted at this point that it is very unlikely that higher probability patterns exist; if they did, they would have been exploited by traders already. Also note that a very small excess accuracy (e.g. 1%) can translate into a very significant "edge" if the number of samples is very high, as they are in these experiments. That is, across 40,000 tests, an accuracy of 51% implies that there were 400 more correct predictions than incorrect predictions.

In terms of which market predictability on an intraday basis, it appears that EUR/USD is the most predictable with accuracies achieving 52.15%. On the other hand, the EUR/JPY market is the least predictable, with no model accuracy exceeding 50.72%.

The performances of the different classifiers are also interesting. In both the EUR/USD and USD/JPY cases, classifiers other than 0R and RT-1 achieve the best performance. This implies that patterns encompassing more than a single feature exist in the data. In the case of EUR/USD, SMO performs the best, whilst for USD/JPY, the decision tree is the best performer. The decision tree RT also happens to be the best overall average performer.

For the volatility prediction task, the results shown in Table 3 are quite different. Although the frequency of the classes are balanced (0R achieves little over 50%), the typical classifier can achieve approximately 65-70% accuracy. However, unlike the previous case, the stronger classifiers perform no better than the weaker classifier RT-1 in two out of the three cases. This implies that only a single feature may be needed for volatility prediction, at least for those two markets.

Table 4. Performance of ensembles of EDS-learned models for the return prediction task, by dataset.

	EUR/USD	USD/JPY	EUR/JPY	Average
Ens(EDS+RT)	51.48	**51.23**	50.73	51.15
Ens(EDS+SMO)	**52.57**	**51.23**	50.58	**51.46**
Ens(EDS+RF)	52.01	51.06	**50.77**	51.28

Table 5. Performance of ensembles of EDS-learned models for the volatility prediction task, by dataset.

	EUR/USD	USD/JPY	EUR/JPY	Average
Ens(EDS+RT)	71.48	72.31	71.03	71.61
Ens(EDS+SMO)	62.61	65.67	64.61	64.30
Ens(EDS+RF)	**71.89**	**72.79**	**71.85**	**72.18**

Despite this, the Table shows that EDS again boosts every model's performance most of the time. In fact,EDS+RT-1 is the overall best model for volatility prediction in every case.

3.4 Experiment 2: Ensemble Prediction

One well-known drawback of randomized algorithms such as the steady state genetic algorithm is its dependence on the random initial conditions: on some runs, a highly accurate model may be built because the random initial solutions were good, but on other runs, only weaker models may be found because the initial population was largely poor. In other words, randomized algorithms have a performance variance dependent on the random number seed.

One method proposed here to reduce this variance is to build multiple models using EDS, and then combine them together into a single model by averaging their predictions. This should, in theory, produce more reliable estimates because the weaker and stronger classifiers will average out in the final results. However, this method does turn models that were originally interpretable (such as the decision tree model, RT) into black-box classifiers, because now each ensemble consists of multiple models whose predictions are now being combined in a new "second stage" of the prediction process.

To investigate this idea, EDS was applied to each base model (RT, SMO, and RF) not once but ten times, producing ten prediction models. These models were then combined into an averaging ensemble, and the ensemble was used to make the predictions. We applied this to both the return and volatility prediction tasks, with exactly the same experimental train/test setup as previously described. The results are shown in Tables 4 and 5.

The first point to note from these Tables (specifically comparing Table 2 to Table 4 and Table 3 to Table 5) is that uniformly, an ensemble of EDS-filtered classifiers outperforms a single equivalent EDS classifier by itself. For example,

the accuracy of SMO on the EUR/USD return prediction task is increased by approximately 0.5% giving a new best accuracy. The best accuracies on the other markets are also increased, albeit not as much as in the case of EUR/USD. EUR/JPY still remains the most difficult market to predict with the ensemble only increasing the accuracy of the RF classifier to 50.77% from 50.72%.

An interesting point to note is that the best classifier for EUR/JPY is now RF, rather than the stump, which was the best performer in the previous experiment. This implies that the ensemble is able to detect a pattern involving more than a single attribute whereas the single EDS-tuned model by itself was unable to.

This observation is also true for the second volatility prediction experiment show in Table 5. In the previous experiment, no pattern beyond a single attribute was discovered for volatility prediction, and all the other models appeared to over-fit the data: hence, RT-1 uniformly performed with the highest accuracy. In the second experiment, however, the results are quite different: both the RT and RF models outperform RT-1, and the best accuracies are approximately 1% *at least* above the best results from that experiment.

4 Conclusion

To summarise, extensive experiments have shown that firstly, the EDS approach typically boosts the accuracy of the tested predictive models; and secondly, further accuracy increases are possible by using EDS to construct a diverse ensemble of models.

References

[1] Breedon, F. and Ranaldo A. "Intraday Patterns in FX Returns and Order Flow." Swiss National Bank Working Papers 2011-4 (2010)

[2] L. Breiman. "Random Forests". *Machine Learning* 45(1):5-32, 2001.

[3] Cano J., Herrera F., and Lozano M. "Using Evolutionary Algorithms as Instance Selection for Data Reduction in KDD: An Experimental Study." IEEE Transactions on Evolutionary Computation 7(6) pp. 561-575 (2003).

[4] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explorations*, vol. 11, no. 1, 2009.

[5] Larkin F. and Ryan C. Modesty is the Best Policy: Automatic Discovery of Viable Forecasting Goals in Financial Data. In Proc. EvoApplications 2010, Part II, pp. 202-211.

[6] Luke, S. "Essentials of Metaheuristics." Lulu, Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/ (2009)

[7] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods – Support Vector Learning*, B. Schölkopf, C. Burges, and A. Smola, Eds. MIT Press, 1998.

[8] Pi Trading Corporation, http://pitrading.com/

[9] Sewell, Martin. "Characterization of Financial Time Series." Research Note RN/11/01, Dept. of Computer Science UCL, (2011)