GRAMMAR BASED FEATURE GENERATION FOR TIME-SERIES PREDICTION

A thesis submitted in fulfilment of the requirements for the degree of Master of Philosophy in the School of Electrical and Information Engineering at the University of Sydney

Anthony Mihirana de Silva

November 2013

UNIVERSITY OF SYDNEY

Abstract

Faculty of Engineering
School of Electrical and Information Engineering

Master of Philosophy

by Anthony Mihirana de Silva

The application of machine learning techniques to predict time-series continues to attract considerable attention due to the difficulty of the prediction problems compounded by the non-linear and non-stationary nature of the real world time-series. The performance of machine learning techniques, among other things, depends on suitable engineering of features. This thesis proposes a systematic way for generating suitable features using context-free grammar. The notion of grammar families as a compact representation to generate a broad class of features is exploited. Implementation issues and ways to overcome them are explained in detail. A number of feature selection criteria is investigated and a hybrid feature generation and selection algorithm using grammatical evolution is proposed. The proposed approaches are demonstrated by predicting the closing price of major stock market indices, peak electricity load and net hourly foreign exchange client trade volume. The widely and commonly employed features in practice (in previous work) for electricity and financial time-series are explored. These features are considered as a basis for comparison with the features generated and selected by the proposed framework. Other model-based approaches and naive approaches are also used as benchmarks. It is shown that the generated features can improve results, while requiring no domain-specific knowledge. The proposed method is used to determine suitable features to use in predicting previously unexplored foreign exchange client trade volume and the capabilities of the approach in automatically engineering appropriate features is highlighted. The proposed method can be applied to a wide range of machine learning architectures and applications to represent complex feature dependencies explicitly when machine learning cannot achieve this by itself.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisor, A/Prof. Philip Leong for his guidance, inspiring ideas, encouragement, precious time in reviewing my work and always expecting high standards.

I am thankful to Dr. Ahmed Pasha and Dr. Richard Davis for their valuable comments and discussions, as well as for reviewing my manuscripts which helped me to produce a well organized thesis.

I am extremely grateful to my colleague Farzad Noorian for his generous support which helped me to complete this work in a timely manner.

Finally, I thank my parents for their love, sacrifices and continuous support.

Contents

List of Figures

List of Tables

To Amma and Thaththa

Chapter 1

Introduction

1.1 Motivation and Aims

Rapid advances in data collection technology have enabled businesses to store massive amounts of data. Data mining algorithms are used to analyse such stored data to reveal previously unknown strategic business information in the form of hidden patterns and trends which are not apparent.

A time-series is a sequence of data recorded at successive points in time and is a common occurrence in finance, energy, signal processing, astronomy, resource management, economics and many other fields. Time-series prediction is a form of data mining that predicts future behaviours by analysing historical data.

Machine learning (ML) is concerned with teaching computers to make predictions or behaviours based on information extracted from data. Supervised ML methods have been extensively applied to a range of time-series prediction problems including financial [1–4], energy markets [5–7], control system/signal processing and resource management (see [8, 9] for a comprehensive list of applications). The inherent non-linear and non-stationary nature of real-world time-series makes ML methods (simply referred to as learners) more appealing than model-based approaches [10–13].

Time-series prediction is a supervised learning task. The learner is presented with the training samples $(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)$, where $\mathbf{x}_j \in \mathbb{R}^N$ is a vector of features. For a regression task, $y_j \in \mathbb{R}$ is the target output and for a classification task, $y_j \in \mathbb{Z}$ is the class label. The learner's objective is to find structure in the training samples to infer a general function (a hypothesis) that can predict y_k for previously unseen $\mathbf{x}_k$. The generalization is achieved by searching through a hypothesis space $\mathcal{H}$, for a hypothesis that best fits the training samples. In a binary classification example, the target hypothesis $h : \mathbb{Z} \rightarrow \{0, 1\}$ is a discriminant boundary that maximises the classification accuracy, for support vector and neural network regression, the hypothesis $h : \mathbb{R}^n \rightarrow \mathbb{R}$ is a function that minimises the root mean squared error, and so on.

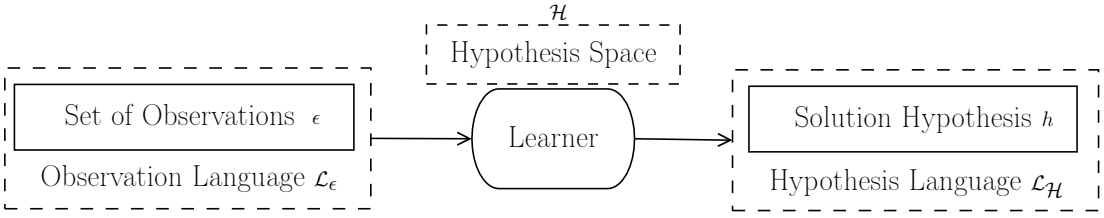

FIGURE 1.1: Language terminology (as depicted in [14]).

As shown in Fig. 1.1, the training samples (observations) ϵ can be specified using an observation language $\mathcal{L}_\epsilon$, and a hypothesis $h \in \mathcal{H}$ can be specified by a language $\mathcal{L}_H$. $\mathcal{L}_\epsilon$ is the notation used to represent the training samples and $\mathcal{L}_H$ is the notation used by the learner to represent what it has learned, e.g. for a neural network $\mathcal{L}_\epsilon$ is the notation used to represent the training patterns and $\mathcal{L}_H$ is the notation used to represent weights. The performance of supervised ML methods depends strongly on the formalism in which the solution hypothesis is represented using $\mathcal{L}_H$. The features used in $\mathcal{L}_\epsilon$ and $\mathcal{L}_H$ are identical which means that one way to achieve a better formulation of $\mathcal{L}_H$ is a better representation of $\mathcal{L}_\epsilon$. Therefore, an important research area is to investigate techniques to expand the feature space used to represent $\mathcal{L}_\epsilon$ and select features that maximize the performance of a particular ML architecture under consideration.

The main aim of this thesis is to develop a flexible framework that can transform the initial feature space of $\mathcal{L}_\epsilon$ to a much larger feature space containing thousands of feature combinations with different parameters and then extract feature subsets from this new space that can produce better predictions. By applying the proposed framework on

real-world datasets, the thesis asks the question "Can we improve the performance of a particular ML architecture by expanding and then reducing the feature space to better formalize the representation of the solution hypothesis?"

1.2 Feature Generation

Feature generation enriches the hypothesis language with additional constructed and derived features [15]. A goal of feature generation is to represent feature dependencies explicitly when the ML algorithm cannot accomplish this by itself. A ML algorithm might be able to elucidate hidden dependencies in data by itself but this can be supplemented in practice by proper feature engineering to enhance performance. This section sheds some light on the relationship between the features and the innate machinery of a kernel based ML algorithm.

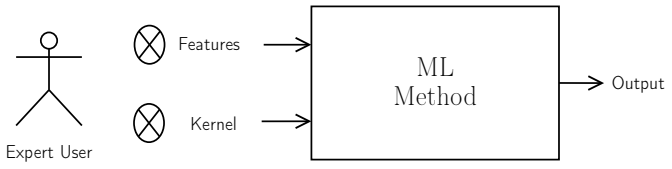

FIGURE 1.2: Tuning knobs of a kernel based ML algorithm.

The performance of a kernel based ML algorithm can be enhanced by, (i) kernel optimizations, e.g. type of kernel (only the classical kernels are considered, i.e. linear, polynomial, Gaussian and sigmoid), kernel parameters (ii) input feature space manipulation by feature engineering. These tasks can be visualized as tuning knobs of a kernel based ML algorithm available to a human expert as in Fig. 1.2. Ideally, kernel optimization and feature space manipulation should be concurrent tasks of the human expert but this is infeasible due to the large search space. Therefore, in practice, it is required to strike a correct balance between kernel optimization and feature engineering. For time-series prediction, the additional burden of empirically evaluating parametrized features, e.g. a moving average of a lagged time-series as a feature, and determining for each feature (i) which lags to use (ii) what look-back periods to use (iii) what are the moving-widow sizes to use and (iv) what are the best feature combinations to use, can

have a great impact on ML algorithm performance. This thesis decouples kernel optimization and feature engineering. The same philosophy can be extended to non-kernel based ML algorithms.

1.3 Approach

The work proposes to use context-free grammar (CFG) as a framework for defining rules that are sequentially invoked to generate a large feature space. Expert suggested features are expanded by a range of operators used by the grammar framework (see Fig. 1.3). The grammars are formally defined using the Backus-Naur form (BNF) notation, customizable and organized into grammar families. A user with a better understanding of the time-series under consideration, i.e. an experienced user or a domain expert, can define focused grammars to generate features that are more suitable.

The grammars defined in this thesis generate thousands of features. This large feature space is mined to extract best features for the learner under consideration. Standard filter and wrapper feature selection (FS) techniques are explored and a novel hybrid FS algorithm which uses grammatical evolution (GE) is proposed as well.

If the solution hypothesis formulated using the expert defined features is h_1 and the solution hypothesis formulated using a feature subset discovered through the proposed framework is h_2, which solution hypothesis is better, h_1 or h_2? The proposed technique is applied on real-world time-series to answer this question and the results are discussed.

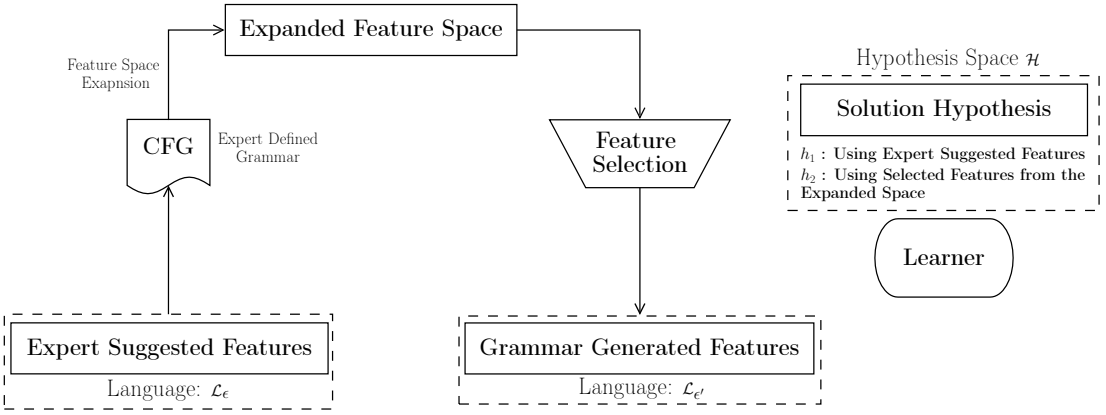

FIGURE 1.3: The research question. Which hypothesis is better? h_1 or h_2?

1.4 Contributions

The first contribution of this thesis is the implementation of a novel framework to systematically generate suitable features using CFGs. The notion of grammar families is introduced. Implementation issues affecting the feature space size, feature informativeness, computational time, framework flexibility are studied in detail and ways to overcome them proposed. Some of the grammar families use the concept of *technical indicators* [16], as used in finance. These are formulae that identify patterns and market trends in financial markets which are developed from models for price and volume. The work is unique in that no previous attempts have been made to automatically generate informative features in the form of technical indicator formulae for time-series prediction. Other proposed grammar families capture information using wavelet transformations, history windows and other operators.

Depending on the configuration of the grammar framework, irrelevant and redundant features can be produced. FS eliminates such features, thereby improving the performance and speed of ML algorithms [17]. Most ML techniques are designed to find the relevant features but an additional pre-processing step is often required prior to the application of ML techniques to get the best performance. Furthermore, using all of the generated features in a learning task is impractical and leads to over-fitting. As the second contribution of this thesis, different FS filters like information gain, symmetric uncertainty, correlation, maximum-relevance-minimum-redundancy (mRMR), Relief and wrapper techniques such as sequential forward and backward selection and genetic algorithms are compared to investigate the effectiveness of the selected algorithms in mining large feature spaces.

As the third contribution, the proposed feature generation framework is extended as a hybrid FS and feature generation framework by using a modified version of GE [18]. GE is presented as a convenient way to avoid selective feature pruning. This extended system is flexible in that an expert user can also suggest known feature subsets, e.g. feature subsets that are known to work well, and the system attempts to discover feature subsets that give better predictions. To the best of the author's knowledge, this is the first time GE is applied as a feature discovery technique for time-series prediction.

The software was developed using the R programming language and is made publicly available since there is no existing R package for GE (available at [19]).

Furthermore, the thesis also explores financial time-series and electricity peak load prediction using features employed in standard practice (in previous work). These results are considered as a basis for comparison with the proposed framework.

Finally, foreign exchange client trade volume time-series prediction is also investigated, the proprietary nature of such data having resulted in little published work. The capability of the framework in automatically identifying appropriate types of features to use for such previously unexplored datasets is highlighted.

1.5 Thesis Structure

Chapter 2 presents the necessary background in time-series prediction using ML techniques within the thesis. Several FS techniques are discussed and the CFG and support vector machine theory is presented. Chapter 3 contains details on how CFG is used as a feature generation framework and descriptive implementation details of the proposed feature generation framework. Chapter 4 continues to present how FS is performed on the expanded feature space to mine for "good" features. This chapter also describes how the learner is applied to predict time-series by utilizing the sliding window technique, cross-validation and parameter tuning. Chapter 5 presents the research results and discusses the results. Finally, the research question is answered and future research directions are suggested in Chapter 6. Wavelet theory is deferred to Appendix A. Some production rule sequences that generate interesting features are provided in Appendix B and supplementary results are in Appendix C.

Chapter 2

Background

This chapter establishes the theoretical foundations on which the research in this thesis is based. Specifically, the areas covered are: time-series prediction, application of machine learning (ML) techniques to time-series prediction, the importance of feature selection, feature selection techniques, context-free grammar (CFG), genetic algorithms (GAs), grammatical evolution (GE) and support vector machines (SVMs). The work presented in this thesis is an amalgamation of these research fields.

2.1 Time-series Prediction

A time-series is a sequence of vectors (or scalars) recorded at successive points in time. A vector sequence produces a multivariate time-series and a scalar sequence produces a univariate time-series. The sequence $x(t_0), x(t_1), \ldots, x(t_{i-1}), x(t_i), x(t_{i+1}), \ldots$ is a univariate time-series of the recorded variable x.

Time-series have many application domains ranging from economics to engineering. Daily share prices, daily foreign exchange rates, yearly company profits are some examples of financial time-series. Daily temperature measurements, sunspot activity and daily average rainfall are some time-series arising from physical phenomena. Other

types of time-series also exist in disciplines such as astronomy, physics, marketing and geography. Fig. 2.1 depicts the time-series of yearly number of sunspots, well-studied EUNITE daily electricity load [20] and closing price of the Standard & Poor's 500 (GSPC) stock index.

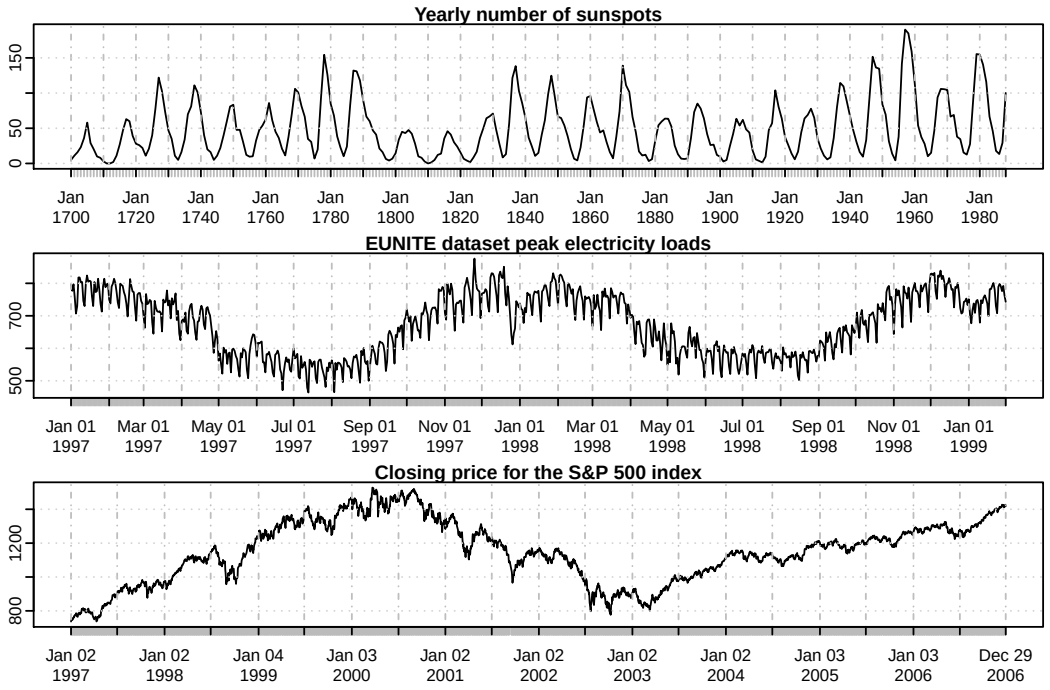

FIGURE 2.1: 3 real world time-series.

Time-series can be recorded at regular or irregular intervals and the recorded values can be discrete or continuous. A series of events occurring randomly in time is a special type of time-series known as a point process, e.g. the dates of major earthquakes. In this research, the time-series of interest are financial time-series of daily stock index prices, daily peak electricity load time-series and foreign exchange client trade volume time-series (irregular).

The objective of a time-series prediction task at time t is to estimate the value of x at some future time, $\hat{x}[t+s] = f(x[t], x[t-1], \ldots, x[t-N])$, $s > 0$ is called the horizon of prediction, e.g. for one-step ahead predictions $s = 1$. Fig. 2.2 shows the prediction of a time-series using autoregressive integrated moving average (ARIMA) model. The years 1973-1979 are the historical data and the years 1980-1981 are the predictions.

The horizon of prediction is 2 years of daily values ($s = 730$ days) and the prediction confidence limits are also shown.

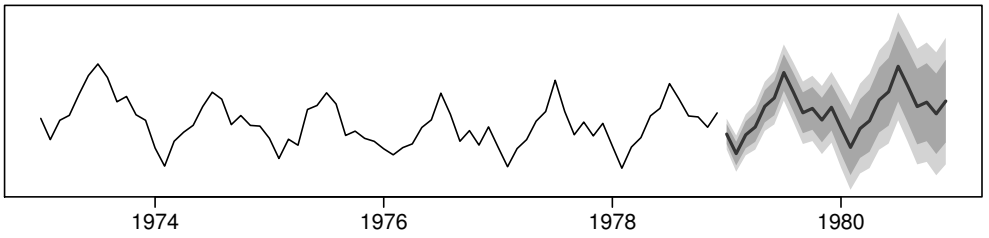

FIGURE 2.2: Time-series prediction: predicting the daily values using ARIMA.

Hyndman and Athanasopoulos [21] discuss three general model based approaches to time-series prediction: explanatory models, time-series models and mixed models.

Explanatory models use predictor variables commonly referred to as features in the ML terminology. Suppose that the prediction task is to predict the AUD/USD exchange rate ($E_{AUD/USD}$) for the next month. An explanatory model can be of the form, $E_{AUD/USD} = f$(Australian interest rate, US interest rate, strength of economy, gross domestic product, foreign relations, error).

If $E_{AUD/USD}$ is denoted by E, a simple *pure time-series model* of the form $\mathrm{E}_{t+1} = f(\mathrm{E}_t, \mathrm{E}_{t-1}, \mathrm{E}_{t-2}, \mathrm{E}_{t-3}, \ldots, \text{error})$ can be constructed. Such models only use information from the variable to be predicted and makes no attempt to discover the factors affecting its behaviour, i.e. it will extrapolate trend and seasonal patterns but ignore all other external covariates such as Australian interest rate, US interest rate, strength of economy, gross domestic product, foreign relations etc.

A mixed model can be formed as a combination of explanatory and pure time-series models, e.g. $\mathrm{E}_{t+1} = f(\mathrm{E}_t, \mathrm{E}_{t-1}, \mathrm{E}_{t-2}, \mathrm{E}_{t-3}, \ldots,,$ Australian interest rate, US interest rate, strength of economy, gross domestic product, foreign relations, error).

Pure time-series models are in much wider use than the explanatory models because it is difficult to understand and accurately model the complex relationships between the explanatory variables and the target variable. Many time-series models are based

on linear methods [22] in which the output variable depends linearly on its own previous values. Real world time-series are often non-linear and non-stationary. Non-linear approaches such as non-linear autoregressive processes, bilinear models and threshold models are widely used for time-series modelling. Generalized autoregressive conditional heteroskedasticity (GARCH) model is another non-linear time-series model used to represent the changes of variance over time (heteroskedasticity). In this thesis we use 2 pure time-series models, ARIMA and exponential smoothing state space model (ETS) for the purposes of comparison.

The drawback of model based approaches is that usually a priori assumption of the underlying distribution of data is required for model parameter estimation. ML techniques can alleviate this issue and cope with the inherent non-linear and non-stationary nature of real world time-series.

2.1.1 Machine Learning Techniques for Time-series Prediction

Real world time-series prediction problems are often complex and have interdependencies that are not clearly understood. The inherent non-linear and non-stationary nature makes ML prediction techniques more suitable than model-based approaches. The distinction of ML algorithms from the time-series model based approaches is the aspect of learning (training). For a neural network, learning represents the optimization of the network weights. For a SVM, this is the construction of the hyperplane in a high dimensional space. These optimizations are done using a set of training samples, i.e. known outcomes of the problem for given conditions. The final data modelling goal is not to memorize known patterns but to generalize from prior examples, i.e. be able to estimate the outcomes of the problem under unknown conditions. The ML method has to extract from the training samples, something about the distribution to allow it to produce useful predictions in new, unseen cases. A ML task starts with the construction of a dataset. Fig. 2.3 illustrates the steps involved with the training phase of a ML algorithm.

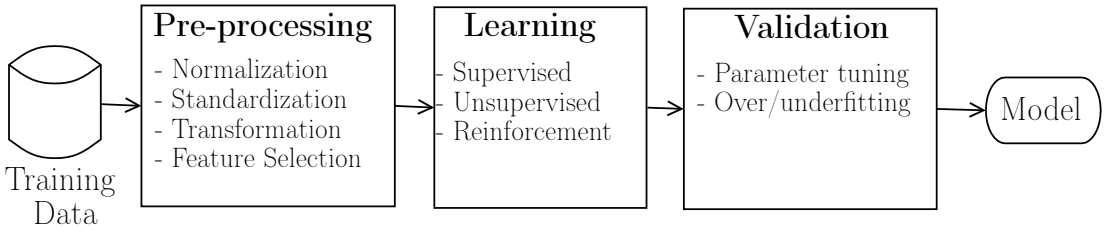

FIGURE 2.3: Training phase of a machine learning method.

- **Pre-processing:** Data pre-processing includes simple operations like data cleaning, normalization, standardization and more advanced operations like feature transformation and feature selection (FS). Pre-processing can be time-consuming but has shown to produce significantly better results due to better representation of the solution hypothesis. Usually, features are considered individually for simple pre-processing tasks. Any outliers are removed and non-available data samples are replaced with a mean value, closest value, regressed value or the most common value. The features are often scaled to the range [-1, 1] or [0, 1] to ensure that each feature is independently normalized using min-max or z-score normalization [23]. The concepts of feature transformation (sometimes referred to as feature construction) and FS are discussed later in this chapter.

- **Learning:** This is the core aspect of a learning method. Unsupervised learning involves discovering structure in unlabelled data, e.g. clustering algorithms. On the other hand, supervised learning infers a mapping function that maps input features to outputs (or labels), e.g. regression and classification algorithms. A reinforcement learning agent learns by interacting with its environment and observing the feedback of these interactions which in a way mimics human behaviour. It involves trial and error to maximize a certain reward, e.g. in robotics applications. This thesis is only concerned with supervised learning algorithms.

- **Validation:** Cross-validation is a way of measuring the performance of a ML method before the algorithm is deployed as a real world application. It is easy to overfit a model by including too many degrees of freedom which will lead to poor generalization for unseen data. Therefore, validation sets are used for model optimization such as parameter-tuning and is described in Sec. 4.2.2.

2.2 Application of Machine Learning Techniques in Time-series Prediction

It was already mentioned that real world time-series are often non-linear and non-stationary. For a time-series $x[t]$, the auto-covariance is given by $\gamma(x[t+\tau], x[t]) = cov(x[t+\tau], x[t])$, $\forall\ t$ and lag τ. $x[t]$ is stationary if two conditions are satisfied. The first condition is the expectation of $x[t]$ is finite and does not depend on t, i.e. $E(x[t]) = \mu_{x[t]} = \mu < \infty$. The second condition is for each lag τ, the auto-covariance does not depend on t, i.e. $\gamma(x[t+\tau], x[t]) = \gamma_\tau$. In simple terms, the statistical properties of a stationary time-series are time-invariant. In a non-stationary time-series, the distribution of the time-series changes over time causing changes in the dependency between the input and output variables. An effective learning algorithm should have the potential to take this into account. In this research, the sliding window (or the rolling window) technique for one-step ahead predictions was used as described in Chapter 4 to create new prediction models when new data becomes available.

If the time-series to be predicted is denoted by $y[t]$ and the features by $x_1[t], x_2[t], \ldots, x_k[t]$, each feature is a time-series by itself. In one-step ahead predictions, the objective is to predict the value of $y[t+1]$ using the previously unseen feature vector $x_1[t], x_2[t], \ldots, x_k[t]$. ML techniques can be applied for time-series prediction as a classification task or a regression task. For regression tasks, the target variable is $y[t]$ or an appropriately smoothed version of $y[t]$ [24]. For classification tasks, a discrete target variable is usually constructed from a continuous target variable $y[t]$. For example, in a daily stock closing price prediction task, if the closing price $C[t]$ has an upward directional change $C[t+1] - C[t] \geq 0$, $y[t] = +1$, and if the price has a downward directional change $C[t+1] - C[t] < 0$, $y[t] = -1$, i.e. $y[t] \in \{-1, +1\}$. For both regression and classification, appropriate features need to be constructed.

As an example application, the approach used by Kim [25] to predict the daily directional change of the Korean composite stock index (KOSPI) was to use technical indicators as features. The first feature x_1 was the technical indicator MACD (moving average convergence divergence). The feature x_2 was the indicator disparity and feature

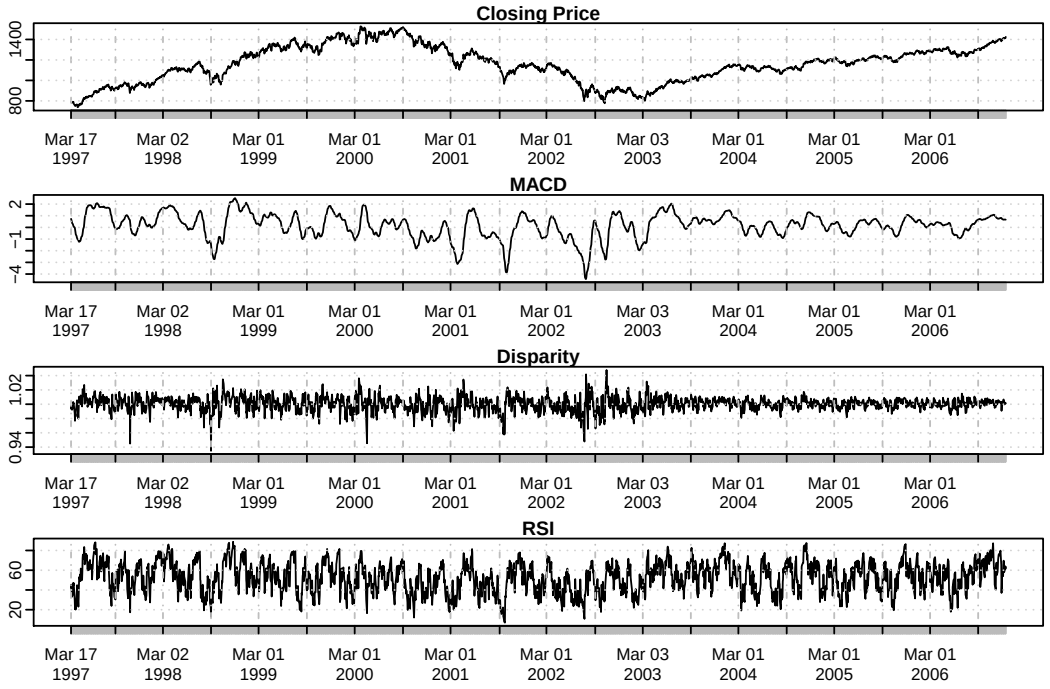

FIGURE 2.4: Next day's closing price is the target and MACD, Disparity and RSI are used as features.

x_3 was the indicator RSI. Fig. 2.2 shows the target (closing price) and 3 of the features (MACD, disparity and RSI). Kim used a total of 12 technical indicators with 1637 training samples forming a feature matrix of 1637×12. This produced the feature matrix $\mathcal{X}$ with $k = 12, n = 1637$. $\mathcal{X} =$

$$\begin{pmatrix} x_1[t-n] & x_2[t-n] & \cdots & x_k[t-n] \\ x_1[t-(n+1)] & x_2[t-(n+1)] & \cdots & x_k[t-(n+1)] \\ \vdots & \vdots & \ddots & \vdots \\ x_1[t-2] & x_2[t-2] & \cdots & x_k[t-2] \\ x_1[t-1] & x_2[t-1] & \cdots & x_k[t-1] \end{pmatrix}$$

The ML algorithm was trained with the feature matrix $\mathcal{X}$. 581 unseen feature vectors were presented to the model to get the predictions. For example, the unseen vector $x_1[t], x_2[t], \ldots, x_k[t]$ was used to predict the target variable $y[t+1]$, the vector $x_1[t+1], x_2[t+1], \ldots, x_k[t+1]$ was used to predict the target variable $y[t+2]$, and so on. It was necessary to ensure that at any time in predicting the value of $y[t+1]$, the feature matrix had no access to information beyond time t. Such a phenomenon which will lead to unrealistically good predictions and is referred to as "peeking" within this thesis. The techniques used to ensure that we avoid peeking are presented in Sec. 4.4.

The performance of ML techniques in predicting time-series, among other things, depends on a suitable crafting of feature vectors that contain information about the target signal. In most cases, these are selected by experienced users and domain experts. The next section explains the concept of FS using computational methods.

2.3 Feature Selection

FS is the process of choosing a subset of features that improves the ML method performance. Formally, for N data samples with M features in each data sample, FS problem is to find from the M-dimensional observation space R^M, a subspace of m features R^m that best predict the target. This is achieved by reducing the number of features to remove irrelevant, redundant, or noisy information from the feature set.

FS primarily aids in alleviating the curse of dimensionality and speeding up the learning task. In practice, it also helps in optimizing data collection methodologies by identifying which data to collect. Therefore, FS is a key pre-processing step (see Fig. 2.3) to improved predictions.

In a typical ML problem, there is usually an optimal number of features that provide the minimum error (highest accuracy). Because the total number of subspaces is 2^M, finding an optimal feature subset is usually intractable [26] and many problems related to FS have been shown to be NP-hard [27]. Alternatively, many sequential and random searches have been proposed.

FS strategies are essentially twofold; (i) ranking the relevant features according to different criteria and (ii) collectively choosing a feature subset [28]. Feature ranking assigns a weight for each feature and feature subset selection evaluates different feature combinations. In general, filter models are used as feature ranking strategies and wrapper and embedded models are used as feature subset selection strategies. These models are explained next.

2.4 Feature Selection Models

This section describes the filter and wrapper models in detail and briefly explains the embedded models. The evolutionary FS technique proposed in Sec. 3.5 can be considered as a hybrid approach that employs both filter and wrapper models.

2.4.1 Filter Models

A filter model makes use of intrinsic characteristics in the data for feature ranking. The feature ranking criterion can be based on dependency measures, distance measures and consistency measures. In this research, the commonly used filters include the information gain (dependency), maximum-relevance-minimal-redundancy (dependency), Pearson's correlation (dependency) and the Relief algorithm (distance). By evaluating these measures between the feature and the target variable the features can be assigned a weight and they can be ranked.

- **Information Gain:** Also referred to as the divergence between two random sequences (X) and (Y) with probabilistic densities $p(x)$, $p(y)$ and $p(x,y)$ is given by $I(x,y) = \iint p(x,y) \log \frac{p(x,y)}{p(x)p(y)}\, dx\, dy$. This is a measure of the information lost when X is used to predict Y. Information measures are widely-used in statistics and quantify the dissimilarity between time-series.

- **Maximum-relevance-minimal-redundancy (mRMR):** mRMR selects good features according to maximal statistical dependence criterion based on mutual information [29]. Maximum relevance features are assessed by Eq. 2.1 where S is a feature set of m features such that $|S| = m$, x_i is an individual feature and c is the target.

$$\max\ D(S,c),\ D = \frac{1}{|S|} \sum_{x_i \in S} I(x_i, c) \tag{2.1}$$

  Although highly relevant features are selected, there is a high probability of redundancy. Therefore, mRMR uses a minimal redundancy criterion to select mutually

exclusive features by choosing features that satisfy Eq. 2.2.

$$\min R(S),\ R = \frac{1}{|S|^2} \sum_{x_i, x_j \in S} I(x_i, x_j) \tag{2.2}$$

The criterion that combines the above two constraints is called "maximum-relevance-minimal-redundancy" criterion below.

$$\max \Phi(D, R),\ \Phi = D - R \tag{2.3}$$

Incremental search methods are used to find the near-optimal features defined by $\Phi(\cdot)$. Starting from an initial feature set S_{m-1} with $m-1$ features, the objective is to select the m^{th} feature from the remaining set $\{X - S_{m-1}\}$ by selecting the feature which maximizes $\Phi(\cdot)$. The incremental algorithm is given in Eq. 2.4.

$$\max_{x_j \in X - S_{m-1}} \left[I(x_j, c) - \frac{1}{m-1} \sum_{x_i \in S_{m-1}} I(x_j, x_i) \right] \tag{2.4}$$

- **Correlation:** Given two random sequences (X) and (Y) with means μ_X and μ_Y respectively, the Pearson's correlation coefficient at lag τ is, $\rho_{x,y}(\tau) = \frac{\sum_k (X_k - \mu_X)(Y_{k+\tau} - \mu_Y)}{\sqrt{\sum_k (X_k - \mu_x)^2} \sqrt{\sum_k (Y_{k+\tau} - \mu_Y)^2}}$. Direct feature-target correlation by itself is however considered as a poor technique because a high feature-target correlation can still be an irrelevant feature (referred to as spurious correlations).

- **Relief Score:** Distance based filter models such as Relief ranks features that distinguish classes based on how well a feature can separate classes. The original Relief algorithm proposed by Kira and Rendell [30] is a two-class filtering algorithm. A training sample R_i is chosen randomly and the Euclidean distance to all other instances calculated. The algorithm then proceeds to choose the nearest hit (same class) H and the nearest miss (different-class) M based on the calculated distances. If there are a features, the feature weight is recursively updated for each feature A using Eq. 2.5. The whole process is repeated m times and the average weight W[A] is retained.

$$W[A] = W[A] - \text{diff}(A, R_i, H)/m + \text{diff}(A, R_i, M)/m \tag{2.5}$$

The weighting function in Eq. 2.5 assigns a higher weight for features that discriminate classes while penalizing the weight of the features that do not contribute to class separability. RRelief (regression relief) [31] was developed later which was able to operate on regression problems. Since regression problems do not involve classes, RRelief uses a probability measure which is modelled by the target variable values of instances belonging to different classes.

Another filter model type is the consistency. Consistency models find a minimum number of features that distinguish between the classes as consistently as the the full set of features do. An "inconsistency" arises when multiple training samples have the same feature values but different class labels, e.g. set cover [32].

In the presence of multiple interacting features, the individual feature relationship with the target can be insignificant but if such interacting features are considered in combination, a strong relationship to the target variable *may* be identified. In other words, "the m best features are not the best m features" [29]. Many efficient FS algorithms assume feature independence, ignoring the interacting features. Jakulin and Bratko [33] define an interacting feature subset as an irreducible whole: "a whole is reducible if we can predict it without observing all the involved variables at the same time". Therefore, different feature combinations are often evaluated using wrapper models. For large dimensional feature spaces, a filter model is usually applied first to reduce the feature space dimensionality followed by the application of a wrapper model.

2.4.2 Wrapper Models

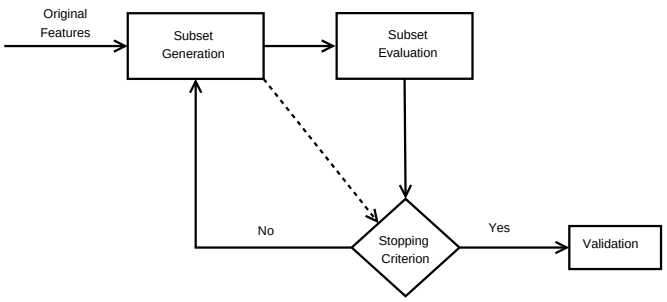

FIGURE 2.5: A general wrapper model [27].

A wrapper model involves a feature evaluation criterion, a search strategy and a stopping criterion as depicted in Fig. 2.5. Unlike a filter model, a wrapper model utilizes the performance of the learning algorithm as the evaluation criterion. This ensures that the selected feature subset performs optimally for the ML algorithm [34]. Wrappers are computationally expensive since it requires the ML algorithm to be executed in every iteration and are usually used for low dimensional datasets.

Search strategies generate different feature combinations to traverse through the feature space (generating feature subsets for evaluation). A widely used stopping criterion is to stop generating and evaluating new feature subsets when adding or removing features does not make any improvements [27]. The sequential steps of a wrapper model are given in Algorithm 1.

Algorithm 1 Wrapper Approach

Input
- X ▷ Training data with N features, $|X| = n$
- N ▷ Number of features (feature space dimensionality)
- n ▷ Number of training examples
- A ▷ Learning Algorithm
- SC ▷ Stopping criterion
- S_0 ▷ Initial subset of features

Output S_{best} ▷ Selected feature subset

$S_{best} \leftarrow S_o$ ▷ Assign the current best feature subset as the initial subset
$\gamma_{best} \leftarrow eval(S_0, X, A)$
repeat
  $S \leftarrow SearchStrategy(X)$ ▷ Generate next subset to be evaluated
  $\gamma \leftarrow eval(S, X, A)$
  if $\gamma \geq \gamma_{best}$ **then**
    $\gamma_{best} \leftarrow \gamma$
    $S_{best} \leftarrow S$
  end if
until SC is reached **return** S_{best}

For N features there exist 2^N potential subsets. For even modest values of N, an exhaustive search over this huge space is intractable. The feature search therefore

plays an important role in wrappers. Exponential, sequential and randomized searches are described next.

2.4.2.1 Feature Search

- **Exponential search:** An exponential search returns the optimal subset. Although the order of the search space is $\mathcal{O}(2^N)$, the search need not be exhaustive, i.e. heuristics can be introduced to reduce the search space without compromising the chances of discovering the optimal subset [27], e.g. branch and bound and beam search.

- **Sequential search:** Sequential forward feature selection (SFFS), sequential backward feature elimination (SBFE) and bidirectional selection are greedy search algorithms that add or remove features one at a time [27]. SFFS initiates with an empty set and SBFE initiates with a full set whereas a bidirectional search initiates a SFFS and SBFE simultaneously ensuring that the features selected by SFFS are never eliminated by SBFE. The drawback of SFFS is that it could fail to eliminate redundant features generated in the search process. SBFE has the drawback of not being able to re-evaluate feature usefulness together with other features once a feature is removed. Plus-L minus-R selection (LRS) search attempts to resolve these issues. The worst case search space of sequential search is in the order of $\mathcal{O}(N^2)$.

- **Randomized search:** In practical applications, the feature space may contain thousands of features, e.g. bioinformatics, text, natural language processing applications and the work in this thesis. In such applications, it is not feasible to search the entire feature space. Randomized search trades off optimality of the solution for efficiency by searching only a sampled portion of the feature space. GAs have been used as a guided randomized FS technique [35, 36] and is discussed in depth in Sec. 2.6.

In contrast to filter and wrapper models, embedded models do not separate learning from FS. Embedded models are specifically catered for certain ML algorithms and applications. SVM-RFE is an embedded FS method proposed in [37] for gene selection in cancer classification. Nested subsets of features are selected in a sequential backward elimination manner, which starts with all the features and removes one feature at a time.

Method	**Complexity**	**Advantages**	**Disadvantages**
Exhaustive	$\mathcal{O}(2^N)$	High accuracy	Computationally expensive
Sequential	$\mathcal{O}(N^2)$	Simple	Less flexible with backtracking
Randomized	$\mathcal{O}(N \log N)$	Users can trade-off between accuracy and speed, avoids trapping in local optima	Low accuracy

TABLE 2.1: Search criterion comparison.

At each step, the coefficients of the weight vector of a linear SVM are used to compute the feature ranking score. Embedded techniques are complex and application specific. In this research we do not use any embedded techniques for FS.

A different approach to eliminate redundant and irrelevant features and reduce the feature space is dimensionality reduction. Linear techniques such as principal component analysis (PCA) perform a linear mapping of the data to a lower dimensional space in such a way, the variance of the data in the low-dimensional representation is maximized. Kernel principal component analysis (KPCA) and manifold techniques are able to perform non-linear dimensionality reduction.

2.5 Principal Component Analysis

PCA is mathematically defined as an orthogonal linear transformation that maps the data to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on. In practice, PCA is calculated via SVD (singular value decomposition).

The application of PCA is described for a feature matrix $\mathbf{X}_{p\times N}$, i.e. there are p features and N samples. The feature matrix here has features as rows and the data samples as columns (different from the feature matrix $\mathcal{X}$ in Sec. 2.2). The feature matrix is first centred by subtracting the means from each feature, $\mathbf{X} = [(\mathbf{x_1} - m_1), \ldots, (\mathbf{x_N} - m_N)]$. Then, SVD is performed on $\mathbf{X}_{p\times N}$.

$$\mathbf{X} = \mathbf{U}_{p\times p}\mathbf{D}_{p\times p}(\mathbf{V}_{N\times p})^T \tag{2.6}$$

$\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices and $\mathbf{D}$ is a diagonal matrix. The scatter matrix $\mathbf{S}$ can be written as,

$$\mathbf{S} = \mathbf{X}\mathbf{X}^{\mathbf{T}} = \mathbf{U}\mathbf{D}^2\mathbf{U}^T \tag{2.7}$$

The eigenvectors of $\mathbf{S}$ are the columns of $\mathbf{U}$ and the eigenvalues are the diagonal elements of $\mathbf{D}^2$. By taking only a few significant eigenvalue-eigenvector pairs such that $d \ll p$. A new reduced dimension representation: $\tilde{x_i} = m + \mathbf{U}_{p\times d}(\mathbf{U}_{p\times d})^T(x_i - m)$ is formed.

In this research, the number of features is much greater than the number of training samples, i.e. $p \gg N$. This is because thousands of features are generated for a smaller number of training samples. In such cases, the SVD of $\mathbf{X}^{\mathbf{T}}$ is computed,

$$\mathbf{X}^{\mathbf{T}} = \mathbf{V}_{N\times N}\mathbf{D}_{N\times N}(\mathbf{U}_{p\times N})^T \tag{2.8}$$

$$\mathbf{X} = \mathbf{U}_{p\times N}\mathbf{D}_{N\times N}(\mathbf{V}_{N\times N})^T \tag{2.9}$$

By choosing only a few significant eigenvalue-eigenvector pairs such that $d < N$: $\tilde{x_i} = m + \mathbf{U}_{p\times d}(\mathbf{U}_{p\times d})^T(x_i - m)$.

The first principal component accounts for as much variance as possible in the data followed by other principal components. The assumption behind PCA is that the feature matrix is jointly normally distributed. Non-linear dimensionality reduction techniques like kernel PCA do not have this assumption.

2.6 Genetic Algorithms

A number of evolutionary approaches have been applied to FS. An initial population is created and evolved to traverse through the feature space. The best individuals of the population are selected using different criteria and the ML algorithm performance using the feature subset represented by an individual is used to assess the fitness of an individual. GAs have been used as a wrapper technique, thus introducing a search mechanism to avoid enumerating the entire space. A simple approach is to encode features in the chromosome, e.g. the chromosome 01001000 could mean that the 2^{nd} and 5^{th} features are selected [38, 39]. Huang and Wang [40] used a chromosome with

parameters of a SVM (the penalty parameter C, and the kernel parameter γ). The resultant solution provided the best features and the appropriate SVM parameters. Genetic programming (GP) for FS has been used where a GP based online FS algorithm which evolved a population of classifiers was used to choose accurate classifiers with the minimum number of features [41]. Multi-population GP has also been used in [42] to develop a method for simultaneous feature extraction and selection.

2.6.1 Canonical Genetic Algorithms

Canonical GA works on n-tuple of binary strings b_i of length l. The n-tuple is termed as the population and the bits of each string are considered to be genes of an individual chromosome. A chromosome is represented as a binary string, with each consecutive group of n-bits creating a *codon*. A group of codons is called a *gene*. An individual chromosome may contain more than one gene. The operations performed on the population are selection, crossover and mutation as illustrated in Fig. 2.6.

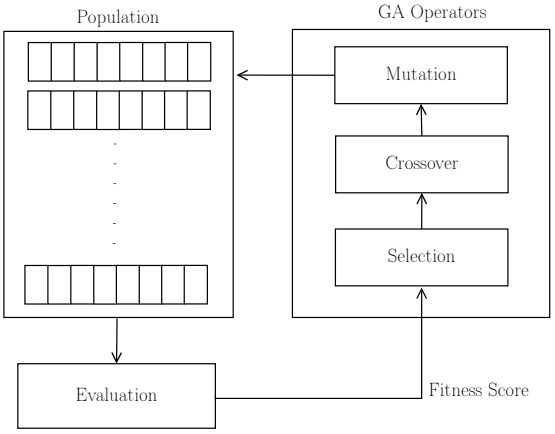

FIGURE 2.6: The evolutionary process and associated operators.

The initial population is created by randomly setting each bit of a chromosome to 1 or 0. The individuals are then evaluated based on a given fitness function $\phi(\cdot)$. Each individual represents a solution for the formalized problem. The objective function $\phi(b_i)$ gives the fitness score of the individual which has to be maximized. The better scoring individuals are deemed to have the best genes hence they are retained and the low scoring chromosomes are discarded from the population and replaced with

new chromosomes. These well scoring individuals are termed elite individuals. The probability for an individual b_i to be selected for recombination is proportional to the relative fitness score given by $\phi(b_i)/\sum_{j=0}^{n}\phi(b_j)$.

The crossover is performed on 2 randomly selected individuals. In canonical GA the traditional crossover operator is the single-point crossover. A crossover point in the binary string is chosen at random and the string sections of the two parents are exchanged. The mutation operator randomly flips single bits on a specific chromosome under a defined mutation probability. Mutation is necessary to maintain genetic diversity from one generation of a population to the next. The evolutionary process is repeated until a given termination criterion is satisfied. Integer GAs which is an extension of canonical GAs is explained in Sec. 4.3.2.

2.7 Grammatical Evolution

Grammatical evolution (GE) can be used to evolve complete programs in an arbitrary language and like other evolutionary programming techniques such as GP, evolves a population towards a certain goal. The difference between GE and GP is that GE applies evolutionary operators on binary strings which are then converted using the defined grammar to the final program; on the other hand GP directly operates on the actual program's tree structure [18]. In GE, a suitable grammar developed for the problem at hand is first specified in Backus-Naur form (BNF). Chapter 4 proposes a hybrid FS and feature generation algorithm using a modified version of GE. In order to describe GE, context-free grammar (CFG) is brought in the next section.

2.7.1 Context-free Grammar

A CFG is a simple mechanism to generate patterns and strings using hierarchically organized production rules [43]. Using the Backus-Naur form (BNF), a formal notation for CFG [44], a CFG can be described by the tuple $(\mathcal{T},\mathcal{N},\mathcal{R},\mathcal{S})$ where $\mathcal{T}$ is a set of terminal symbols and $\mathcal{N}$ is a set of non-terminal symbols with $\mathcal{N}\cap\mathcal{T}=\varnothing$. The non-terminal symbols in $\mathcal{N}$ and terminal symbols in $\mathcal{T}$ are the lexical elements used in

specifying the production rules of a CFG. A non-terminal symbol is one that can be replaced by other non-terminal and/or terminal symbols. Terminal symbols are literals that symbols in $\mathcal{N}$ can take. A terminal symbol cannot be altered by the grammar rules $\mathcal{R}$, a set of relations (also referred to as production rules) in the form of $\mathcal{R} \rightarrow \alpha$ with $\mathcal{R} \in \mathcal{N}$, $\alpha \in (\mathcal{N} \cup \mathcal{T})$. $\mathcal{S}$ is the start symbol $\mathcal{S} \in \mathcal{N}$. If the grammar rules are defined as $\mathcal{R} = \{x \rightarrow xa, x \rightarrow ax\}$, a is a terminal symbol since no rule exists to change it, whereas x is a non-terminal symbol. A language is context-free if all of its elements are generated based on a context-free grammar. If $\mathcal{S}$ is the starting symbol and we define a CFG, $\mathcal{T} = \{a, b\}$, $\mathcal{N} = \{\mathcal{S}\}$ and $\mathcal{R} = \{\mathcal{S} \rightarrow a\mathcal{S}b, \mathcal{S} \rightarrow ab\}$, $\mathcal{L} = \{a^n b^n | n \in \mathbb{Z}^+\}$ is a context-free language. An example grammar in BNF notation is given in Table 2.2.

2.7.2 Generating Features using GE

In this section, the grammar in Table 2.2 is used to demonstrate how features can be generated using GE. Consider the individual [166|225|180|132|187|219|179|249] in which the integer numbers represent codon values of 8 bits, i.e. the individual is 64 bits in length. The codon values are used to select production rules from the example grammar definition to generate features. The usual mapping function used is the MOD rule defined as; (codon integer value) MOD (number of rules for the current non-terminal), where MOD is the modulus operator (%).

$\mathcal{N} = \{expr, op, coef, var\}$
$\mathcal{T} = \{\div, \times, \texttt{+}, \texttt{-}, \texttt{V1}, \texttt{V2}, \texttt{C1}, \texttt{C2}, \texttt{(}, \texttt{)}\}$
$\mathcal{S} = <expr>$

Production rules : $\mathcal{R}$

$\langle expr \rangle$	::=	$(\langle expr \rangle)\langle op \rangle(\langle expr \rangle)$	(1.a)
		$\mid \langle coef \rangle \times \langle var \rangle$	(1.b)
$\langle op \rangle$	::=	$\div \mid \times \mid$ + $\mid$ -	(2.a), (2.b), (2.c), (2.d)
$\langle coef \rangle$	::=	C1 $\mid$ C2	(3.a), (3.b)
$\langle var \rangle$	::=	V1 $\mid$ V1	(4.a), (4.b)

TABLE 2.2: An example grammar in BNF notation.

Now it is shown how the example chromosome above generates a symbolic feature expression using the grammar provided in Table 2.2, which is later evaluated as a numerical feature. Using the start symbol $\mathcal{S} =$ <expr>, there are 2 production rules to choose

from, (1.a), (1.b). The MOD operation on the current codon becomes (166)MOD(2) which yields 0, hence choose rule (1.a). The successive application of rules in Table 2.3 shows how a feature expression is generated by the example chromosome.

MOD	Rule	Current element state
166 MOD 2	0	(<expr>)<op>(<expr>)
225 MOD 2	1	(<coef>×<var>)<op>(<expr>)
180 MOD 2	0	(c_1 ×<var>)<op>(<expr>)
132 MOD 2	0	(c_1 × v_1)<op>(<expr>)
187 MOD 4	3	(c_1 × v_1) ÷ (<expr>)
219 MOD 2	1	(c_1 × v_1) ÷ (<coef>×<var>)
179 MOD 2	1	(c_1 × v_1) ÷ (c_2 × <var>)
249 MOD 2	1	(c_1 × v_1) ÷ (c_2 × v_2)

TABLE 2.3: Production of a terminal element.

The derivation sequence above can be maintained as a derivation tree. The derivation tree is then reduced to the standard GP syntax tree as in Fig 2.7.

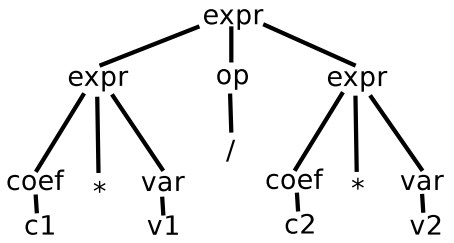

FIGURE 2.7: Feature generation tree.

GE uses standard concepts in canonical GAs to evolve the chromosomes thereby generating new derivation sequences, new feature expressions and new numerical features. Chapter 4 presents a more detailed description on how GE can generate much complex feature expressions.

For a given individual, the resultant feature can be a terminal ($\mathcal{T}$) or a non-terminal ($\mathcal{N}$) element. If the individual runs out of codons without producing a terminal element as above, it is wrapped around and the codons are reused from the beginning. This could lead to incomplete individuals if the mapping never produces a terminal element, which is addressed by introducing a limit on the allowed chromosome wrappings and returning a poor fitness score. An in-depth explanation on the usage of GE can be found in the original GE paper [18].

2.8 Support Vector Machines

This section explains the basic concept of SVM (see [45] for an in-depth explanation) which is the main ML algorithm used in this research. SVMs were first proposed for binary classification which finds an optimal hyperplane with a maximum margin which separates labelled data into two classes. This is achieved by finding a hyperplane in a higher dimension by transforming the original feature space using a kernel function (known as the kernel trick). The optimum (maximum margin) hyperplane gives the maximum separation between the two classes. If the training examples are labelled as $\{\mathbf{x_i}, y_i\}$, $i = 1, 2, \ldots, n$, $y_i \in \{-1, +1\}$ and $\mathbf{x}_i \in \mathbb{R}^d$. The linear hyperplane can then be represented by, $y_i = \mathbf{w} \cdot \mathbf{x} + b$, where $\mathbf{w} \cdot \mathbf{x}$ is the inner product between the weights ($\mathbf{w}$) and the training samples ($\mathbf{x}$), and b is the bias. Soft margin SVM was introduced to accommodate classifying training data when there is no clean hyperplane to separate the training data. The objective of soft margin SVM was to split the training data as cleanly as possible while still maximising the distance to the closest cleanly split data samples. In a similar manner to the soft-margin classification approach, SVM regression seeks to optimize the generalization bounds given for regression using an error tolerance parameter, ϵ.

The objective of support vector regression (SVR) is to estimate the function $f(x) = \mathbf{w} \cdot \mathbf{x_i} + b$ using linear regression in a higher dimensional space. Similar to the soft-margin constructed in SVM classification, the so called ϵ-insensitive loss function in Eq. 2.10 is adapted for SVM regression. ϵ is the error tolerance and only deviations larger than ϵ are considered as errors.

$$L_\epsilon[y - f(x)] = \begin{cases} 0, & \text{if } |y - f(x)| \leq \epsilon \\ |y - f(x)| - \epsilon, & \text{otherwise} \end{cases} \tag{2.10}$$

The function $f(x)$ is estimated by solving the following minimization problem. N is the number of training samples.

$$\min_w \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{N}\sum_{i=1}^{N} L_\epsilon[y - f(x_i)] \tag{2.11}$$

The problem is transformed into a constrained convex optimization problem by employing slack variables ξ_i and ξ_i^*.

$$\begin{aligned} \underset{w}{min} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & f(x_i) - y \leq \epsilon + \xi_i \\ & y - f(x_i) \leq \epsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 \text{ for } i = 1, 2, \ldots, N. \end{aligned} \tag{2.12}$$

To solve the optimization problem, Lagrange multipliers are added to the condition equations, and the problem is restated in its dual form.

$$\begin{aligned} \underset{w}{max} \quad & -\epsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)y_i - \frac{1}{2}\sum_{i,j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)(x_i, x_j) \\ \text{subject to} \quad & 0 \leq \alpha_i, \alpha_i^* \leq C \\ & \sum_{i=1}^{M}(\alpha_i - \alpha_i^*) = 0. \end{aligned} \tag{2.13}$$

where α_i, α_i^* are the Lagrange multipliers. Only the non-zero values of Lagrange multipliers are required to predict the regression line, and their corresponding data samples are called support vectors. The data samples inside the ϵ-tube have Lagrange multipliers as zero, and do not contribute to the regression. Fig. 2.8 aids in visualising the tolerance error, the function $f(x)$ to be regressed and the slack variables with the training data samples.

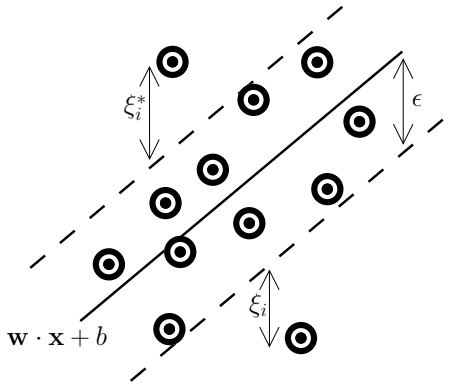

FIGURE 2.8: Epsilon tube with slack variables and selected data points.

The data samples are transformed to a higher dimension by a transformation function $\Phi(x)$, i.e. (x_i, x_j) in Eq. 2.13 can be substituted with $\Phi(x_i) \cdot \Phi(x_j)$ which is defined as the kernel function $K(x_i, x_j)$. The Gaussian radial basis function (RBF) is used throughout this research which is defined with the kernel parameter γ as $K(x_i, x_j) = exp(-\gamma \|x_i - x_j\|^2)$.

The weights can be re-expressed as, $w = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\Phi(x_i)$ and the function to be estimated can be expressed as $f(x) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)K(x_i, x_j) + b$, where l is the number of support vectors. Once the number of support vectors, Lagrange multipliers, and the bias are determined from the training samples, previously unseen testing data can be regressed using the function $f(x)$.

It is important to choose the correct SVM and kernel parameters for a particular task at hand since the performance of the ML algorithm depends highly on the parameter values. "The parameter C controls the trade-off between the margin and the size of the slack variables ($C = \infty$ leads to hard margin SVM)" [46]. On the other hand, ϵ governs the width of the ϵ-insensitive zone. This can affect the number of support vectors used to construct the ϵ regression function. For higher values of ϵ, fewer support vectors are selected and there is a higher chance of over-fitting.

2.9 Summary

This chapter explained time-series prediction, application of machine learning techniques to time-series prediction, the importance of feature selection, feature selection techniques, context-free grammar, genetic algorithms, grammatical evolution and support vector machines which are essential to understand the thesis. The following chapters utilize the theory presented in this chapter to construct a feature generation and selection framework for time-series prediction.

Chapter 3

Grammar Based Feature Generation

Instead of manually constructing feature vectors, the methodology proposed in this chapter can automatically generate a large pool of analytical features; including such manually constructed features. Although the proper selection of analytical features is of utmost importance, it is often overlooked and it is doubtful if the best possible models are constructed by manually selected features. Context-free grammars (CFGs) are used as a systematic way of generating suitable features. The notion of grammar families as a compact representation to generate a broad class of features is explained. Special attention is given to feature pruning and implementation issues arising as a result of processing a large number of features. Furthermore, grammatical evolution (GE) is proposed as a convenient technique to combine feature generation and feature selection (FS) without selective feature pruning.

3.1 Feature Generation using Context-free Grammars

Automated feature generation is the process of generating numerical descriptions of some data instances. A manual approach involves identifying certain characteristics of the time-series under consideration and to derive mathematical formulae to generate

numerical measures to describe them [47]. As suggested in Chapter 1, one aim of this thesis is to enable experts to provide guidelines to the automated feature generator but for the computer to do all the tedious work of generating and selecting appropriate feature subsets for the prediction task at hand.

3.1.1 Related Work on Grammar Based Feature Generation

Automated feature generation is an interesting research area since it may systematically produce and select better feature combinations than those designed by humans. It has been used mostly in signal classification. A general framework for function-based feature generation using context-free grammars was first proposed by Markovitch and Rosenstein [48]. Such grammars are used in linguistics to describe sentence structure and words of a natural language and in computer science to describe the structure of programming languages [43]. Markovitch and Rosenstein generated features strongly related to the target using decision trees. Unfortunately, the technique is only suitable for problems where the features are apparent from the problem definition. Eads et. al [47] and Pachet and Roy [49] addressed supervised time-series classification using standard genetic programming to discover a set of fundamental signal processing operations via a grammatical structure. Both these works conclude that conventional classifiers trained using raw data as features can be outperformed by training the same classifiers with grammar generated features. Standard genetic programming was also used by Ritthof et. al [15] to combine feature generation and feature selection, and applied to the interpretation of chromatography time-series. Ritthof et al. used arithmetic operators in their grammar to expand the feature space while Eads et al. extracted time-series information using operators such as the mean, delay, derivative, integral, etc. [50] used genetic programming with a set of mathematical transformation operators, e.g. sin, cos, +, -, sqrt, etc. to produce features of the raw vibration signals from a rotating machine (fault classification). The same framework was applied to breast cancer diagnosis as well [51]. Both works reported improved classification accuracies. Islamaj et al. [52] proposed a feature generation algorithm for analysing splice-site reduction in biology. An improvement of around 6% compared with using well-known features was reported. Krawiec and Bhanu [53] used a co-evolutionary feature generation approach

where multiple populations were evolved simultaneously. Primary operators applicable to images, e.g. different filters, image norms, scalar operators etc. were defined and the best operator sequences (or processing steps embedded in chromosomes) were used for synthetic aperture radar (SAR) image recognition. The approach proved to be robust in different operating conditions.

The proposed framework in this chapter uses a range of such operators to expand the feature space. The parameters of these operators are selected appropriately to maximize the prediction performance of the ML algorithm under consideration. Previous work has given less attention to appropriate feature selection techniques of a feature generation framework to select robust features which is extremely important in predicting non-stationary time-series hence Chapter 4 of the thesis is dedicated for feature selection. Comparisons to using well-known features is also missing in literature. The thesis also proposes GE as a hybrid technique for feature generation and selection. [54] used GE to select features for detecting epileptic oscillations within clinical intracranial electroencephalogram (iEEG) recordings of patients with epilepsy. GE has also been used in computational finance, credit rating & corporate failure prediction, music and robot control applications (see [55] for a survey). None of these works however formulate a well organized general framework that is extensible and customizable. Furthermore, none of the works have applied their techniques to financial time-series prediction using technical indicators or electricity load time-series prediction or foreign exchange client trade volume time-series predication.

3.1.2 Proposed Framework

This work uses CFGs to systematically guide the generation of a pool of candidate features and define (hierarchical) grammar structures with different layers to guide the feature generation flow. Unlike genetic algorithms in feature construction [56], a CFG framework facilitates visualization of the feature generation flow which helps to monitor the features generated. The layered organization of operators used to generate features is shown in Table 3.1. At time k, the base layer consists of the observed variables $x_k^{(1)}, \ldots, x_k^{(m)}$ and the derived variables $f_k^{(1)}, \ldots, f_k^{(l)}$. Using the notation in

Table 3.4, O, H, L, C are the observed variables and M, U, D are the derived variables in the financial time-series prediction context.

User defined layers		e.g. EMA($c_k^{(i)}$, n)
Combinatorial and transformation layer elements $c_k^{(i)}$ are parsed to the higher layers		
Combinatorial and transformation layer	Fractional combinations	e.g. $(b_k^{(i)} - b_k^{(j)})/b_k^{(j)}$
	Additive combinations	e.g $b_k^{(i)} \pm b_k^{(j)}$
	Base operators	e.g. $log(b_k^{(i)})$
	Running operators	e.g. $func(b_k^{(i)})$ (see Table 3.2)
Base layer elements $b_k^{(i)}$ are parsed to the transformation layer		
Base layer	Derived variables	$f_k^{(1)}, \ldots, f_k^{(l)}$
	Observed variables	$x_k^{(1)}, \ldots, x_k^{(m)}$

TABLE 3.1: Layered organization of operators for feature generation.

The transformation operators in the combinatorial and transformation layer consists of base operators and running operators. The base operators are simple operators such as the first difference and the absolute value of a variable. Running operators use sliding windows of length n to identify local features. By varying n, long or short history information can be captured systematically. The combinatorial operators fuse information across variables to produce more features which are parsed to the user defined layers, and are defined in Table 3.2. Domain transformations (e.g. wavelets, Fourier) are also useful in constructing informative features. For example, the wavelet transformation has been used for multi-resolution analysis of stock data to capture information on different time-scales that is not obvious from the original time-series [57]. Such transformed variables can easily be incorporated to the framework by including them in the base layer as derived variables. Additionally, transformations can be defined in a higher layer as well (preferably in the combinatorial and transformation layer).

The operators defined in Table 3.2 suffice to generate a broad class of features. In the next section, it is explained how the proposed framework can be used to generate simple forms of features from a grammar based on wavelet components for electricity load time-series prediction.

Base operators		**Running operators** (window size n)	
diff(x, n)	$x_k - x_{k-n}$	sma(x)	simple moving average
log(x)	natural log	wilder(x)	Wilder exponential moving average
delt(x)	$(x_k - x_{k-1})/x_k$	ema(x)	exponential moving average
abs(x)	$\|x_k\|$	wma(x)	weighted moving average
lag(x, n)	x_{k-n}	max(x)	maximum value
sin(x)		min(x)	minimum value
cos(x)		sd(x)	standard deviation
...		sum(x)	summation
		meandev(x)	mean deviation
		skewness(x)	skewness
		kurtosis(x)	kurtosis
		median(x)	median
		histwin(x)	a history window (a lag sequence)

TABLE 3.2: Base operators and running operators.

3.2 Wavelet Based Grammar for Electricity Load Time-series Prediction

In order to demonstrate how a feature is generated, the grammar in Table 3.3 which was designed to generate features to predict peak electricity load time-series is used. This grammar is expressed in Backus-Naur form (BNF) notation. The operator notation is as in Table 3.2 and `E` is the half-hourly electricity load time-series. $\mathtt{D}^1$, $\mathtt{D}^2$, $\mathtt{D}^3$ and $\mathtt{S} = \mathtt{S}^3$ are the 3-level wavelet decomposition components of `E`. Wavelet transformation background theory is deferred to Appendix A.

$\mathcal{T}$ = {`abs`, `delt`, `diff`, `lag`, `sma`, `sd`, `meandev`, `histwin`, `E`, $\mathtt{D}^1$, $\mathtt{D}^2$, $\mathtt{D}^3$, `S`, `n`, `k`, `(`, `)`}
$\mathcal{N}$ = {*expr*, *base-var*, *pre-op*, *base-op*, *var*}
$\mathcal{S}$ = <*expr*>

$\mathcal{R}$ Production rules

⟨*expr*⟩	::=	`histwin`(⟨*base-var*⟩, `n`)	(1.a)
		| ⟨*pre-op*⟩(⟨*base-var*⟩, `n`)	(1.b)
		| ⟨*base-var*⟩	(1.c)
⟨*base-var*⟩	::=	⟨*base-op*⟩(⟨*var*⟩)	(2.a)
		| `lag`(⟨*var*⟩, `k`)	(2.b)
		| ⟨*pre-op*⟩(⟨*var*⟩, `n`)	(2.c)
⟨*pre-op*⟩	::=	`sma` | `sd` | `meandev`	(3.a), (3.b), (3.c)
⟨*base-op*⟩	::=	`delt` | `diff` | `abs`	(4.a), (4.b), (4.c)
⟨*var*⟩	::=	`E` | $\mathtt{D}^1$ | $\mathtt{D}^2$ | $\mathtt{D}^3$ | `S`	(5.a), (5.b), (5.c), (5.d), (5.e)

TABLE 3.3: Wavelet based grammar for peak electricity load time-series prediction.

Recall that a CFG can be described by $(\mathcal{T}, \mathcal{N}, \mathcal{R}, \mathcal{S})$ as explained in Sec 2.7.1. The production rules $\mathcal{R}$ for this grammar are organized into 5 groups. Group 1 has 3 rules (1.a)-(1.c), group 2 has 3 rules (2.a)-(2.c) and so on. The total number of production rules is 17, (1.a)-(5.e) ($|\mathrm{R}| = 17$). Each production rule has a head (left hand side), a non-terminal symbol in $\mathcal{N}$, that is assigned by the string of symbols in the body (right hand side). Multiple production rules in the same group are delimited by the pipe "|". There are 5 non-terminal symbols ($\mathcal{N}$) which are denoted in the production rules by $<\cdot>$. A feature generated by each rule corresponds to a particular layer in Table 3.1. Rule (2.b) adds different lags of observed (`E`) and derived variables (`D`^{1},`D`^{2}, `D`^{3} and `S`) to the generated feature pool. Rule group 2 can be considered as producing features in the combinatorial and transformational layer. Rule (1.a) produces user defined layer features by applying different operators on variable combinations. The rules (1.b) and (1.c) act as mere linking rules that invoke other rules.

As described in Sec. 2.7.1, to generate a specific feature, the feature generation sequence is initiated with the start symbol $\mathcal{S} =$ <*expr*>. The production rules are sequentially invoked on the left-most non-terminal to generate features. The generation of a simple feature is illustrated first.

The production rule sequence (1.b)→(3.a)→(2.b)→(5.a) generates `sma(lag(E, k),n)` which is a running operator in the combinatorial and transformation layer, e.g. for `n`=10 and `k`=0, the feature can be interpreted as the simple moving average of peak load for 10 days. This rule invoking sequence is compactly represented in Fig. 3.1. The circles represent the leftmost non-terminals on which the rules are invoked and the rule number is indicated below the arrows.

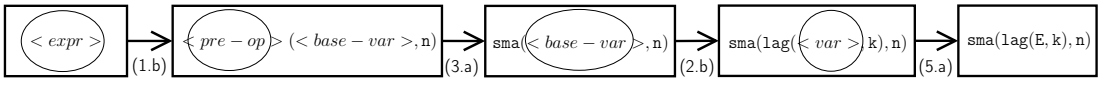

FIGURE 3.1: Step-wise generation of exponential moving average as a feature from the wavelet grammar in Table 3.3.

This can also be illustrated by the following notation.

Invoking rule (1.b): <*expr*> ::= <*pre-op*>(<*base-var*>, n)

Invoking rule (3.a): (<*pre-op*>) ::= `sma`

Invoking rule (2.b): <*base-var*> ::= `lag`(<*var*>, `k`)

Invoking rule (5.a): <*var*> ::= `E`

It is illustrated that rule sequences like above generate primary features such as moving averages. This section demonstrated how simple features are generated from the proposed framework. More complex feature generation sequences to generate technical indicator type formulae are brought later in this chapter.

3.3 Grammar Families

By defining well organized compact grammar families (instead of a single grammar), the number of generated features can be significantly reduced. More importantly, the generated features can now be be more informative and interpretable because they are systematically organized by a human expert. Once all the features are generated, feature selection and dimensionality reduction techniques can be applied to select the best features.

This is illustrated by introducing the grammar families designed to generate *technical indicators*. If a single grammar was designed to generate all the technical indicators surveyed, the number of features will be extremely large. Grammar families were designed to generate standard technical indicators along with many other potentially new technical indicators. Technical indicators are described in the next section.

3.3.1 Technical Indicators

Technical indicators are formulae that identify patterns and market trends in financial markets [58] which are developed from models for price and volume. The seminal work of Edwards et al. [59] on technical analysis still remains in use to the present. Technical indicators can be broadly classified as trend, momentum, volatility and volume indicators. A trend analysis studies price charts using the moving average filters. The moving average filter gives smooth price estimates and identifies overall trend patterns. The exponential moving average (EMA), simple moving average (SMA) and weighted

moving average (WMA) filters are some of the standard trend indicators. Momentum measures the variation of price in a given time. Momentum indicators identify overbought and oversold positions and start of new trends. Rate of convergence (ROC), relative strength index (RSI) and average directional index (ADX) are some commonly used momentum indicators. Volatility indicators like Bollinger bands (BB) identify the uncertainty in the market via statistical variance of price movements. Volume indicators identify the volumes of trade that have the potential to cause market movements, and the money flow index (MFI) is one such example. Table 3.5 summarizes some well-known and standard trend, momentum, volatility and volume indicators and its notation is in Table 3.4.

Symbol	Interpretation
O	Opening price of the current day
H	Highest price of the day
L	Lowest price of the day
C	Closing price of the day
V	Traded volume for the day
M	Typical (average) price, $(\mathrm{H_k} + \mathrm{L_k} + \mathrm{C_k})/3$
U	Upward price change, $max(0, \mathrm{C_k} - \mathrm{C_{k\text{-}1}})$
D	Downward price change, $min(0, \mathrm{C_k} - \mathrm{C_{k\text{-}1}})$
F	Money flow at a given time, $\mathrm{M} \times \mathrm{V}$
F^+	Positive money flow, $max(0, \mathrm{F}_k - \mathrm{F}_{k-1})$
F^-	Negative money flow, $min(0, \mathrm{F}_k - \mathrm{F}_{k-1})$
H^+	Highest high price of the day, $max(H_{k-i})_{i=0}^{n-1}$
L^-	Lowest low price of the day, $min(L_{k-i})_{i=0}^{n-1}$
i^+	Days elapsed since the last highest price, $\arg\max_i \mathrm{P}_{k-i}$, $i = 0, 1, \ldots, n-1$
i^-	Days elapsed since the last lowest price, $\arg\min_i \mathrm{P}_{k-i}$, $i = 0, 1, \ldots, n-1$

TABLE 3.4: Symbol notation. The subscript k denotes the current day.

3.3.2 Generating Technical Indicators using Grammar Families

This section demonstrates how a set of grammar families generate a broad class of features with the technical indicators summarized in Table 3.5 as particular cases. 7 grammar families generate all the technical indicators in Table 3.5 as particular cases. The CFG based framework is flexible in that, (i) the number of grammar families and the organization of the production rules can be adapted (ii) the user is able to design a sufficiently large grammar to capture as much information as possible with a manageable feature space and (iii) the user can incorporate domain knowledge by

Indicator name	Acronym	Formula	Parameters
Simple moving average	SMA	$\frac{1}{n}\sum_{i=0}^{n-1} P_{k-i}$	$n = 5, 15, 30$
Weighted moving average	WMA	$\frac{1}{n}\sum_{i=0}^{n-1}(n-i)P_{k-i}$	$n = 15, 5, 30$
Exponential moving average	EMA	$\sum_{i=0}^{n-1}\alpha(1-\alpha)^i P_{k-i}$	$n = 5, 15, 30$ $\alpha = 2/(n+1)$
Disparity	DIS	$\mathrm{P_k}/\mathrm{ema}(\mathrm{P}_k, n)$	$n = 5, 10$
Bias	BIAS	$(\mathrm{P_k} - \mathrm{sma}(\mathrm{P_k}, n))/n$	$n = 5, 10$
Bollinger bands	BB	$\mathrm{sma}(\mathrm{P}_k, n) \pm 2\sigma$	$n = 15$
Chaikin volatility	-	$\mathrm{sma}(\mathrm{H}_k - \mathrm{L}_k, n) \pm 2\sigma$	$n = 15$
Average true range	ATR	$max(\mathrm{H}_k - \mathrm{L}_k, \lvert\mathrm{H}_k - \mathrm{C}_{k-n}\rvert, \lvert\mathrm{L}_k - \mathrm{C}_{k-1}\rvert)$	-
Money flow index	MFI	$(1 + \mathrm{R})/\mathrm{R}$	$n = 14$
Aroon indicator (Up, Down)	-	$(n - i^+)/n$, $(n - i^-)/n$	$n = 6, 12, 24$
Rate of convergence	ROC	$(\mathrm{C}_k - \mathrm{C}_{k-n})/\mathrm{C}_{k-n}$	$n = 1$
Commodity channel index	CCI	$(\mathrm{M}_k - \mathrm{sma}(\mathrm{M}_k, n))/0.015\bar{\sigma}$	$n = 15$
Relative strength index	RSI	$\mathrm{RS}/(1 + \mathrm{RS})$	$n = 14$
Moving average convergence divergence	MACD	$\mathrm{ema}(\mathrm{C}_k, n_1) - \mathrm{ema}(\mathrm{C}_k, n_2)$, $n_1 > n_2$	$n_1 = 26, n_2 = 12$
Momentum	MOM	$\mathrm{C}_k - \mathrm{C}_{k-n}$	$n = 4$
William's indicator	R	$(\mathrm{H}^+_{k-n} - \mathrm{C}_k)/(\mathrm{H}^+_{k-n} - \mathrm{L}^-_{k-n})$	$n = 15$
Stochastic oscillator	K	$(\mathrm{C}_{k-n} - \mathrm{L}^-_{k-n})/(\mathrm{H}^+_{k-n} - \mathrm{L}^-_{k-n})$	$n = 14$
Stochastic indicator	D	$\mathrm{sma}(\mathrm{K}(n_1), n_2)$	$n_1 = 15, n_2 = 5$
Slow stochastic indicator	Slow D	$\mathrm{sma}(\mathrm{D}(n_1, n_2), n_3)$	$n_1 = 15, n_2, n_3 = 3$
Close location value	CLV	$((\mathrm{C}_k - \mathrm{L}_k) - (\mathrm{H}_k - \mathrm{C}_k))/(\mathrm{H}_k - \mathrm{L}_k)$	-
Price Oscillator	OSCP	$(\mathrm{sma}(\mathrm{P}_k, n_1) - \mathrm{sma}(\mathrm{P}_k, n_2))/\mathrm{sma}(\mathrm{P}_k, n_1)$	$n_1 = 5, n_2 = 10$
Accumulation/Distribution Oscillator	ADO	$(\mathrm{H}_k - \mathrm{C}_{k-1})/(\mathrm{H}_k - \mathrm{L}_k)$	-

TABLE 3.5: Standard trend, volatility and volume indicators.

Additional Notation: Money ratio $\mathrm{R} = \Sigma_0^{n-1}\mathrm{F}_i^+/\mathrm{F}_i^-$. $\mathrm{RS} = \mathrm{ema}(\mathrm{U}, n)/\mathrm{ema}(\mathrm{D}, n)$.

choosing appropriate derived variables and production rules. The grammar families 1 and 2 are brought in Tables 3.6,3.7 and the rest in Appendix B Tables B.1-B.5.

Family 1

$\mathcal{N}$ = {L1, L2, L3}
$\mathcal{T}$ = {− , ÷, `lag`, `sma`, `meandev`, `sum`, `H`, `L`, `C`, `M`, `n`, `k`, `N`, (,) }
$\mathcal{S}$ = {L3}

Production rules : $\mathcal{R}$

$\langle L3 \rangle$	::=	$(\langle L2 \rangle) \div (\texttt{lag}(\langle L2 \rangle, \texttt{k}))$ | $(\langle L2 \rangle) \div (\langle L2 \rangle)$	(1.a), (1.b)
	|	$((\langle L2 \rangle) - (\langle L2 \rangle)) \div \texttt{N}$ | $\langle L2 \rangle$	(1.c), (1.d)
$\langle L2 \rangle$	::=	$\langle L1 \rangle - \texttt{lag}(\langle L1 \rangle, \texttt{k})$ | $\langle L1 \rangle - \texttt{sma}(\langle L1 \rangle, \texttt{n})$	(2.a), (2.b)
	|	$\texttt{meandev}(\langle L1 \rangle, \texttt{n})$ | $\texttt{sum}(\langle L1 \rangle, \texttt{n})$ | $\langle L1 \rangle$	(2.c), (2.d), (2.e)
$\langle L1 \rangle$	::=	`H` | `L` | `C` | `M`	(3.a), (3.b), (3.c), (3.d)

TABLE 3.6: Grammar family 1.

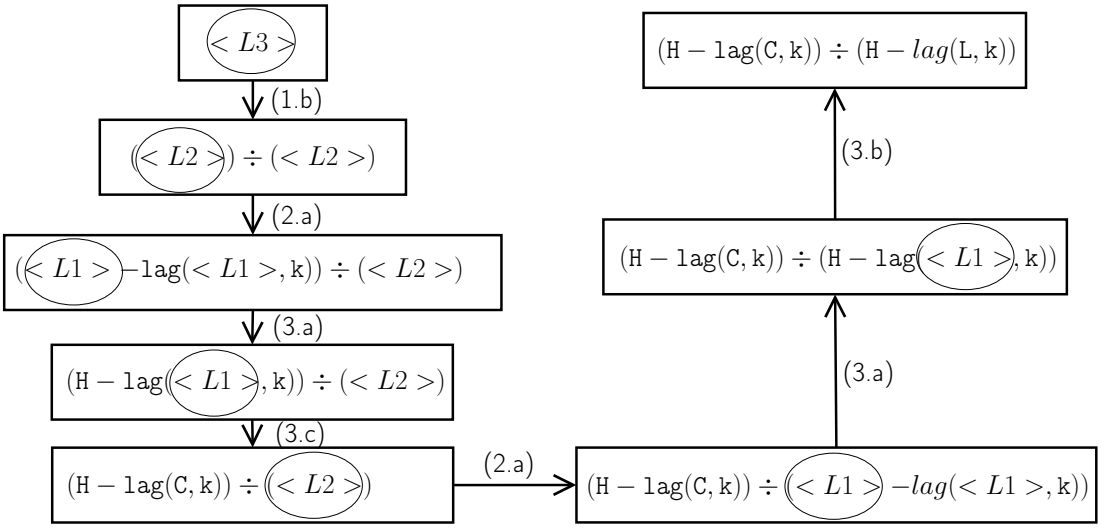

FIGURE 3.2: Step-wise generation of ADO indicator. The circles indicate the current non-terminals on which the rules are invoked and the rule number is indicated.

The grammar family 1 is used to illustrate the generation of standard technical indicators A/D oscillator, CLV, bias and ROC. Fig 3.2 shows the production rules invoked to generate the technical indicator A/D oscillator and the intermediate states produced in the process. The start symbol for grammar family 1 is <*L3*>. By invoking rule (1.b) on <*L3*>, the intermediate non-terminal element (<*L2*>) ÷ (<*L2*>) is produced. Since there are 2 individual non-terminal elements in this intermediate element, the leftmost non-terminal is always chosen (hence circled in figure). Fig. 3.2 can be understood in

this manner and the resultant terminal element can be verified to be the A/D oscillator formula provided in Table 3.5 with `C` lagged by $k = 0$ and `L` lagged by $k = 1$, i.e. $(\mathtt{H} - \mathtt{lag}(\mathtt{C}, 1)) \div (\mathtt{H} - \mathtt{lag}(\mathtt{L}, 0))$. The generation of CLV and ROC by grammar family 1 is deferred to Appendix B.

Family 2

$\mathcal{N}$ = {L1, L2, L3, L4}
$\mathcal{T}$ = {− , ÷, `lag`, `sma`, `ema`, `wma`, `H`, `L`, `C`, `M`, `delt`, `diff`, `n`, `k`, (,) }
$\mathcal{S}$ = {L4}

Production rules : $\mathcal{R}$

$\langle L4 \rangle$	::= $(\langle L3 \rangle) \div (\langle L3 \rangle) \mid (\langle L3 \rangle - \langle L3 \rangle) \mid \langle L3 \rangle$	(1.a), (1.b), (1.c)
$\langle L3 \rangle$	::= $\mathtt{ema}(\langle L2 \rangle, \mathtt{n}) \mid \mathtt{sma}(\langle L2 \rangle, \mathtt{n}) \mid \mathtt{wma}(\langle L2 \rangle, \mathtt{n})$	(2.a), (2.b), (2.c)
	$\mid \mathtt{sma}(\mathtt{ema}(\langle L2 \rangle, \mathtt{n}), \mathtt{n}) \mid \langle L2 \rangle$	(2.d), (2.e)
$\langle L2 \rangle$	::= $\mathtt{diff}(\langle L1 \rangle) \mid \mathtt{delt}(\langle L1 \rangle) \mid \mathtt{lag}(\langle L1 \rangle, \mathtt{k})$	(3.a), (3.b), (3.c)
$\langle L1 \rangle$	::= $\mathtt{H} \mid \mathtt{L} \mid \mathtt{C} \mid \mathtt{M}$	(4.a), (4.b), (4.c), (4.d)

TABLE 3.7: Grammar family 2.

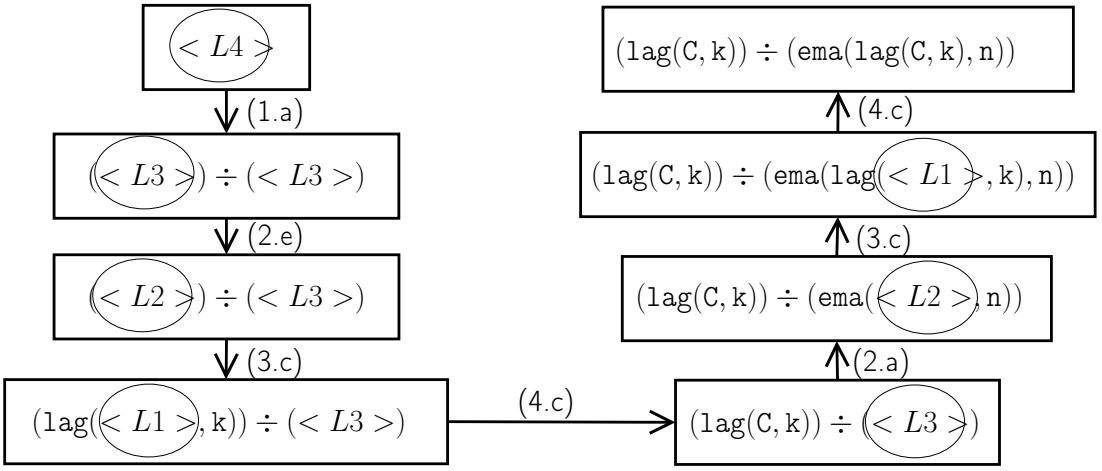

FIGURE 3.3: Step-wise generation of disparity indicator. The circles indicate the current non-terminals on which the rules are invoked and the rule number is indicated.

In a similar fashion, Fig. 3.3 illustrates how the technical indicator disparity is generated using the family 2. The final terminal element, $(\mathtt{lag}(\mathtt{C}, \mathtt{k})) \div (\mathtt{ema}(\mathtt{lag}(\mathtt{C}, \mathtt{k}), \mathtt{n}))$ provides the disparity with `C` lagged by $k = 0$ as can be verified with the disparity formula in Table 3.5. All 7 grammar families were defined to generate one or more technical indicators in Table 3.5 along with many other custom technical indicators. Some other technical indicator generation sequences are presented in Appendix B. Table 3.8 indicates which technical indicators are generated by each grammar family.

Family #	Standard technical indicators generated
1	CLV, CCI, ROC, ADO, Bias, Lagged prices
2	EMA, SMA, WMA, Lagged prices, Disparity, MACD, SD, OSCP
3	R, K, D, Slow D
4	Aroon
5	RSI, MFI
6	BB, Chakin volatility
7	Volume related indicators

TABLE 3.8: Standard technical indicators generated by each grammar family.

The base layer variables H^+, L^- in grammar family 3 rely on a look-back period of n which does not affect the symbolic feature expression but affects the numerical features. This is similarly applicable to the base layer variables in grammar family 4, i^+ and i^-. Notice the variable n used in the interpretation of H^+, L^-, i^+ and i^- in Table 3.4. Therefore, 3 look back-periods of $n = 6, 12, 24$ days were used for grammar family 3 and 4 to generate 3 sets of numerical features from each grammar family.

3.4 Implementation

This section discusses the practical aspects of the feature generation framework and tactics adopted in the implementation.

3.4.1 Pruning Strategies

Some of the generated features are parametrized by the window-size n and lag k. The parameter values used were $n = 5, 15, 30$ and $k = 0, 1, 2, 3, 4, 5, 6$. This means that a feature expression can produce multiple numerical features , e.g. the feature expression `EMA(H,n)` generates 3 numerical features `EMA(H,5)`, `EMA(H,15)` and `EMA(H,30)`. Clearly, the language size explodes by having a larger range for the parameters n and k. Appropriate feature pruning can be used to retain only the top ranking features. An alternative approach is to generate a very large number of features and then choose a subset of the features. Unfortunately, the larger the search space, the harder it becomes to mine for better features. A computationally more efficient approach is to limit the

number of features generated in the first place. This can be achieved by introducing pruning mechanisms to ensure the number of features remain within a manageable range. In the following, the primary pruning strategies are discussed.

(i) Feature pruning can be implicitly achieved by carefully designing the grammar structure. The first step towards this was to introduce grammar families in Sec. 3.3. By defining well organized compact grammar families (instead of a single grammar), the number of generated features can be significantly reduced.

(ii) Limiting the number of production rules in each rule group can also avoid the feature space becoming too large. Constructing focused grammar families for different groups of technical indicators was used to achieve this.

(iii) Grammar family 4 in Table B.2 has separate production rules <*L1*> and <*L2*> for the numerator and denominator terms. This reduces the number of permutations in comparison to using a single rule to generate fractional features.

(iv) Avoiding invalid combination of terms in a production rule avoids generation of meaningless features, e.g. price cannot be added to volume or time and so on.

Despite following the strategies mentioned above, it was found that the number of features generated was still too large to carry out the necessary computations in a reasonable time. Furthermore, it was understood that the computational issues that arise due to memory limitations should be addressed in the feature generation framework implementation because the feature matrices were too large to be processed as single large matrices. Each grammar family produces a different number of feature expressions which are brought in Table 3.9 totalling 24284. With 1900 days under consideration, the feature matrix size was 1900×24284 for financial time-series.

Grammar Family #	# of feature expressions
1	5852
2	10440
3	1675
4	480
5	5516
6	21
7	300
Total	24284

TABLE 3.9: Number of feature expressions generated by each family.

3.4.2 Implementation Issues

This section discusses the implementation issues and secondary pruning strategies involved therein. Fig. 3.4 depicts the feature generation flow which was used in this research to generate features. Because of the design of grammar families, there were some feature expressions that were duplicated, i.e. some feature expressions were generated by multiple grammar families. These feature expressions were removed from the final feature expression list by simply searching for repeated formulae in the list.

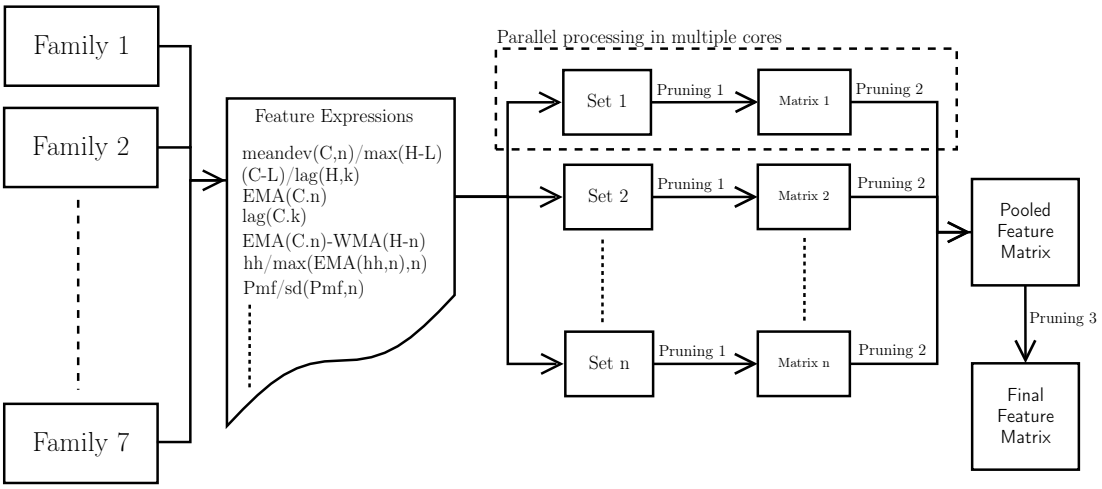

FIGURE 3.4: Feature generation flow.

Each feature expression in the list was evaluated by substituting numerical values for the symbols of the expression. All possible n, k combinations of a particular feature expression were also considered. The feature expression list was divided into 20 sets of roughly 1000 features in each set. The feature expressions in each set were evaluated in a parallel environment using multiple cores in reasonable time since features can be independently evaluated.

For feature expressions having more than 10 permutations, "Pruning 1" (see Fig. 3.4) only selects the 10 best numerical features based on the information gain. If the number of permutations of a particular feature expression was less than 10, all permutations were selected.

It was also observed that some features had very low variance and not helpful in a regression. The feature matrices after the first prune were scaled and the features with low variance were removed by "Pruning 2".

All feature matrices were combined to form a large pool of features and the final feature matrix was achieved by further pruning. This final pruning, "Pruning 3", was carried out to remove features that were reciprocals of each other e.g. the feature generation framework treats `EMA(H,5)` and 1/`EMA(H,5)` as 2 distinct features but it is unclear if these 2 features add any additional information to the learning task. The MD5 hash for these 2 features are the same and this was exploited to keep only one of the features in the final matrix.

Continuing with the methodology of feature generation, ways of automatically generating only "good" feature subsets were also investigated. GE combines feature generation with feature selection by using the fitness function of the underlying GA to assess feature subset goodness using a wrapper approach. It is also shown that more degrees of freedom can be allowed in grammar by allowing recursive rules (see next section). This means that GE does not need any explicit feature pruning. Feature generation using GE is entailed in the next section.

3.5 Feature Generation using GE

In this section, GE is proposed as a novel feature subset selection process. The grammar families in Table 3.10 were designed to generate *trend*, *momentum* and *volatility* type indicators for financial/electricity load time-series prediction and an initial population is evolved using GE to generate feature subsets that are better suited for prediction task at hand. Note the recursive rule (1.a) in the momentum family.

The theoretical background of GE was explained in Sec. 2.7. Fig. 3.5 illustrates how the momentum grammar family maps a binary chromosome to a feature, namely the MACD technical indicator. The binary chromosome has 8 codons and each codon is converted to an integer resulting in the integer chromosome. The GE technique can also be directly used with integer chromosomes but using binary chromosomes leads to

Moving average grammar family

$\mathcal{N}$ = {*expr*, *der-var*, *base-var*, *pre-op*, *base-op*, *var*}
$\mathcal{T}$ = {`delt`, `diff`, `ema`, `sma`, `wma`, `max`, `min`, `H`, `L`, `C`, `n`, `(`, `)` }
$\mathcal{S}$ = <*expr*>

Production rules : $\mathcal{R}$

⟨*expr*⟩	::=	⟨*der-var*⟩	(1.b)
		| ⟨*base-var*⟩	(1.c)
⟨*der-var*⟩	::=	⟨*pre-op*⟩(⟨*base-var*⟩, `n`)	(2.a)
⟨*base-var*⟩	::=	⟨*base-op*⟩(⟨*var*⟩)	(3.a)
		| ⟨*pre-op*⟩(⟨*var*⟩, `n`)	(3.b)
		| ⟨*var*⟩	(3.c)
⟨*pre-op*⟩	::=	`ema` | `sma` | `wma` | `max` | `min`	(4.a), (4.b), (4.c), (4.d), (4.e)
⟨*base-op*⟩	::=	`delt` | `diff`	(5.a), (5.b)
⟨*var*⟩	::=	`H` | `L` | `C`	(6.a), (6.b), (6.c)

Momentum grammar family

$\mathcal{N}$ = {*expr*, *var-op*, *op*, *var*}
$\mathcal{T}$ = {÷, `-`, `delt`, `lag`, `ema`, `H`, `L`, `C`, `n`, `k`, `(`, `)` }
$\mathcal{S}$ = <*expr*>

Production rules : $\mathcal{R}$

⟨*expr*⟩	::=	(⟨*expr*⟩)⟨*op*⟩(⟨*expr*⟩) | ⟨*var-op*⟩	(1.a), (1.b)
⟨*var-op*⟩	::=	`lag`(⟨*var*⟩, `k`) | `ema`(⟨*var*⟩, `n`)	(2.a), (2.b)
⟨*op*⟩	::=	÷ | `-`	(3.a), (3.b)
⟨*var*⟩	::=	`H` | `L` | `C` | `delt(H)`	(4.a), (4.b), (4.c), (4.d)

Volatility grammar family

$\mathcal{N}$ = {*expr*, *var-op*, *op*, *var*}
$\mathcal{T}$ = {`+`, `-`, `abs`, `ema`, `sd`, `meandev`, `H`, `L`, `C`, `n`, `(`, `)` }
$\mathcal{S}$ = <*expr*>

Production rules : $\mathcal{R}$

⟨*expr*⟩	::=	`ema`(⟨*var-op*⟩, `n`) `+` `sd`(⟨*var-op*⟩)	(1.a)
		| `ema`(⟨*var-op*⟩, `n`) `-` `sd`(⟨*var-op*⟩)	(1.b)
⟨*var-op*⟩	::=	`abs`(⟨*var*⟩) | `meandev`(⟨*var*⟩) | ⟨*var*⟩	(2.a), (2.b), (2.c)
⟨*var*⟩	::=	`H-L` | `H-C` | `C-L`	(3.a), (3.b), (3.c)
		| `H` | `L` | `C`	(3.d), (3.e), (3.f)

TABLE 3.10: High, low and close value based multi-family grammar.

better genetic diversity. The start symbol is $\mathcal{S}$=<*expr*>. The first codon value is 222. Since there are 2 production rules (1.a), (1.b) that the non-terminal element <*expr*> can take, (codon integer value) MOD (number of rules for the current non-terminal) is evaluated as 222%2 = 0 which points to rule (1.a). The derivation sequence of the figure can be understood in this manner noting that the left-most non-terminal is always expanded.

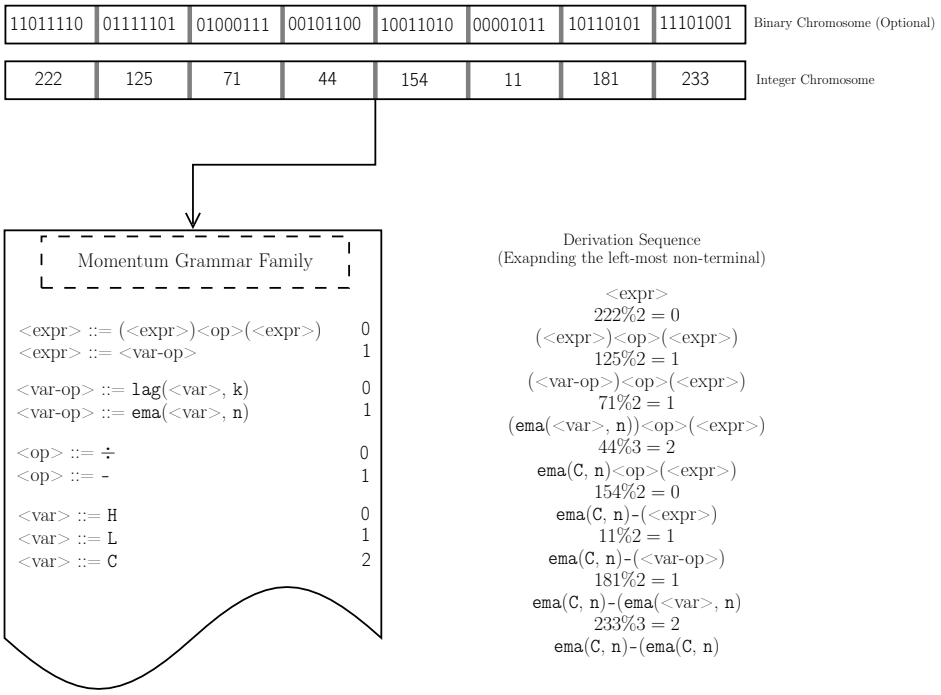

FIGURE 3.5: GE mapping from a linear binary (or integer) chromosome.

It was expected that a good mix of trend, momentum and volatility type formulae could result in better predictions. Therefore, gene partitions of individual chromosomes were mapped into different grammar families. Fig. 3.6 depicts such an examplary mapping of 5 features to 3 grammar families. The binary form of a chromosome is represented as a set of stacked genes for clarity. Each n bit long individual was dissected to N gene partitions where each gene was considered a feature and GE was independently applied on each partition, e.g. for an individual of 1280 bits with 10 features $n = 1280$, $N = 10$.

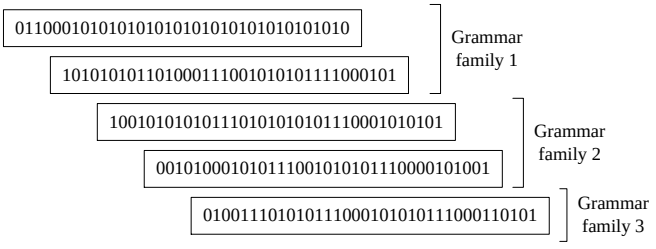

FIGURE 3.6: Gene partition mapping example for 5 features.

It was already mentioned that GE uses standard genetic operators. Individuals were selected for recombination using the fitness proportionate selection (also known as roulette

wheel selection). Random mutation was employed and a modified version of multiple-point crossover in Fig. 3.7 was used. The stacked gene was partitioned based on randomly selected cross-over points and the sections were interchanged to produce the children. This crossover ensured that inter-grammar-family crossover is avoided. Elite individuals were passed to the next generation without mutation or crossover.

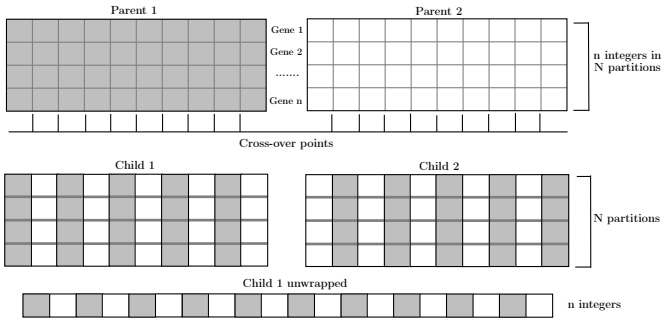

FIGURE 3.7: Proposed multiple-point crossover.

The unique advantage of using GE is the ability to exploit the GE fitness function as an inherent mechanism to penalize bad feature subsets and to evolve the population towards "good" feature subsets that lead to better predictions. Different parameters can be controlled to get feature subsets with desired properties (feature complexity, number of parameters in the features, feature expression character length etc.). This crafting of GE fitness function as a wrapper is described in Sec. 4.3.2.

3.6 Summary

This chapter described the core functionality of the feature generation framework. The mechanism in which the rule sequences were invoked to generate feature expressions was illustrated for simple and complex feature expressions, i.e. technical indicators. The issue of large feature spaces was brought up and selective feature pruning strategies used to contain the most informative features was presented. It was highlighted that the fitness function of grammatical evolution can be used as an inherent mechanism to penalize bad feature subsets hence selective feature pruning can be avoided. The importance of feature selection was highlighted and the next chapter presents how the generated feature space was mined to seek better features.

Chapter 4

Feature Selection

The previous chapter presented the proposed approach to generate a large number of features using an expert defined grammar framework. This chapter proceeds to investigate ways to explore such large feature spaces to extract the best features for prediction, i.e. feature selection (FS). Several FS and feature extraction techniques were explored to determine the best approach to discover "good" feature subsets for particular ML algorithms.

4.1 Common Input Features

When applying ML techniques for a time-series prediction task, the input features are usually chosen by domain experts. Different application domains adopt features of different style, e.g. financial time-series prediction applications use technical indicators, electricity load time-series prediction applications use a history window, temperature, calendar information etc. The common input features for financial time-series and electricity load time-series prediction were surveyed and are presented in Tables 4.1 and 4.2 respectively. These common features were used as a basis to construct grammar structures for each application. By doing so, it was possible to generate commonly used features and many other potentially "good" feature subsets in a large candidate feature

pool. It can be noticed that similar input features are used with different learners in different studies with seldom changes in input feature formulation choice.

4.1.1 Stock Index Time-series Prediction

The analysis of financial time-series is of critical importance to a wide range of business such as corporate banks, broker firms, individual investors and other organisations. Such analysis is used for the hedging of risk, speculative and algorithmic trading, portfolio management, planning and other activities. The accuracy of financial predictions and the speed at which the predictions can be obtained is of great interest. The efficiency and complexity of financial markets however, makes reliable learning of financial time-series an extremely challenging problem.

Seminal work by Fama [75], proposes the random walk hypothesis on stock-market patterns and concludes that "The main conclusion will be that the data seem to present consistent and strong support for the model. This implies, of course, that chart reading, though perhaps an interesting pastime, is of no real value to the stock market investor". Many subsequent studies agree with this theory while an equal number disagree. Lo et al. [76] uses non-parametric kernel regression and conclude that "several technical indicators do provide incremental information and may have some practical value". It has always been a controversial topic and it is generally believed that some predictability can be achieved over some time-periods. Based on this belief, many studies are still conducted on pure financial time-series prediction.

The purpose of this research is to investigate the success achieved by using effective, but seemingly unconventional feature combinations which are not apparent to human experts. Hence, the thesis does not attempt to make advances in the state-of the art financial time-series prediction systems. Nevertheless, it is straightforward for any state-of-the art financial prediction system to make use of the proposed approach as a pre-processing stage to select feature subsets.

A standard approach in financial time series literature is to pick a set of standard technical indicators and/or external economic factors as input features to machine

Technical indicators used as features	# of TIs	Data	Method	Reference
K, D, Slow D, MOM, ROC, R, ADO, DIS, OSCP, RSI, CCI	13	KOSPI	SVM	[25]
O, H, L, C, V, RSI, WMA	7	N225, TAIEX	ICA + SVM	[60]
SMA, RSI, PSY, MOM, D, VR, OBV, DIS, ROC	9	KOSPI	GA	[61]
C, K, D, Slow D, ROC, MOM, SMA, σ, σ ratio, EMA, MACD, ADO, DIS, OSCP, CCI, RSI	18	S&P 500	SVM + GA	[62]
SMA, BIAS, RSI, K, D, MACD, PSY, V	8	TSEC Stocks	K-means + FDT + GA	[63]
SMA, EMA, Projection oscillator, MACD, Qstick, TRIX, etc.	54	DJI, S&P 500	ANN	[64]
SMA, RSI, K, D, MACD, R, PSY, $\Gamma^{\pm}$, BIAS, VR, A ratio, B ratio	13	FITX	SOM + SVM	[65]
Returns, differences of returns, oscillators, SMA, EMA	42	BEL 20	KPCA + RBF-NN	[66]
EMA, ADO, K, RSI, ROC, Price accelerations	10	Stocks	ABFO + BFO	[67]
ΔC, BB, RSI, K, M, CLV, MFI, R	10	Stocks	KPCA + SVM	[68]
OBV, SMA, BIAS, PSY, Average stock yield, C	12	Stocks	AFSA + NN	[69]
SMA, C	6	Forex	ARIMA + ANN	[70]

TABLE 4.1: Common technical indicators used as input features in financial time-series prediction.

OBV = On-balance volume, VR = Volume ratio, TRIX is the percentage change of the triple smoothed moving average of the closing price, QStick is another oscillator and PSY = psychological stability of investors. $\Gamma^{\pm}$ are the directional movement indicators. KOSPI = Korea composite stock price index, N225 = Japanese Nikkei index, TAIEX = Taiwan stock exchange capitalization weighted stock index, S&P 500 = US standard & poor's index, TSEC = Taiwan stock exchange corporation, DJI = Dow Jones index, FITX = Taiwan index futures, BEL 20 = Belgian stock index. ICA = Independent component analysis, GA = Genetic algorithm, FDT = Fuzzy decision tree, ANN = Artificial neural network, SOM = Self-organizing map, KPCA = Kernel principal component analysis, RBF-NN = Radial basis function neural network, ASFA = Adaptive fish swarm algorithm, ABFO = Adaptive bacterial foraging optimization and ARIMA = Autoregressive integrated moving average.

Features	Method	Reference
Previous hour load history windows	SVM	[5, 71]
Previous week's hourly load information, temperature information, calendar information	SVM	[7]
Previous load information and differenced values, Daubechies wavelets (Multi resolution analysis of hourly load upto 168 hours ago) and differenced values, temperature information	ANN	[72]
Wavelets (Multi-resolution analysis of hourly load upto 500 hours ago), temperature information	ANN, GA	[73]
Empirical Mode Decomposition (Multi-resolution analysis of hourly load), temperature information	SVM	[74]

TABLE 4.2: Common input features used in short-term electricity load time-series prediction.

learning (ML) algorithms. Table 4.1 presents the symbols of technical indicators used, number of technical indicators used and method in 12 studies. [25, 60, 65] used standard technical indicators as features in the support vector machine (SVM) to predict financial time-series. In a similar fashion, standard technical indicators have also been used as features in neural networks [25, 64, 70]. [66, 68] used kernel principal component analysis (KPCA) to reduce the dimensionality of the feature space formed by standard technical indicators and used SVM and neural network to predict stock prices. Other techniques such as SVMs optimized using genetic algorithm (GA) [61], neural networks optimized using adaptive fish swarm algorithm (AFSA) [69], boosted experts using GA [61], adaptive bacterial foraging optimization (ABFO) [67] have also been used to model financial time-series. All these applications manually selected a set of standard technical indicators as features for different ML techniques.

In the above-mentioned works, the authors identify a subset of the standard technical indicators that are deemed informative. In each case, the technical indicator used, the number of technical indicators selected and more importantly, the technical indicator parameters used is not the same with some overlap between different works. However, the choice is generally ad-hoc. It would therefore be useful to have a framework that can automatically generate interesting feature combinations by systematically guiding the generation of a pool of candidate features that have a meaningful interpretation.

4.1.2 Electricity Load Demand Time-series Prediction

Accurate load prediction plays a major role in distribution system investments and electricity load planning and management strategies. Bunn and Farmer [77] pointed that a 1% increase in prediction error implied a £10 million increase in operating costs. While overestimation results in an unnecessary spinning reserve and undesired excess supply, underestimation causes failure in providing sufficient reserve and implies high costs per peaking unit. Therefore, it is desirable to predict electricity load demand accurately. It is shown that using decomposed signals is more effective in ML based electricity load prediction than using raw time-series signals [78]. Fourier series can be used to decompose electricity load time-series but it is far useful when the time-series is stationary. Empirical mode decomposition (EMD) and wavelet transforms are the most popular approaches. In predicting non-stationary time-series, such multi-resolution decomposition techniques can be used for elucidating complex relationships [79, 80]. The wavelet transform can produce a good local representation of a signal in both time and frequency domain and is not restrained by the assumption of stationarity. EMD is an alternative to wavelet transform but EMD based feature generation is not adapted in this thesis. History windows and first differences of the raw time-series and decomposed components seem to be popular expert hand picked features in electricity load time-series prediction (see Table 4.2).

4.1.3 Better Feature Combinations than Common Input Features?

From Tables 4.1 and 4.2 it can be seen that applications tend to use a similar set of features over and over with ad-hoc parameter values. As brought in Chapter 1, we were intrigued by the question, "can we find feature combinations better than a human expert selected features for a given ML architecture?". It can be argued that having a better observation language can lead to a better hypothesis language leading to a better solution hypothesis (see Chapter 1). By searching for better input feature combinations generated via an expert defined grammar framework, the thesis answers this question under certain conditions.

4.2 Data Partitioning for Time-series Prediction

In a typical application of ML to time-series prediction, it is common practice to divide the time-series into training, validation, and testing (out-of-sample) sets. The training set is used to construct a model. The model is supposed to fit the samples in the training set. The validation set is used to evaluate the generalization ability of the trained model. The model parameters are tuned such that the model performs satisfactorily on the validation set. The final evaluation of the model is based on its performance on the testing set. No changes to the model should be made by repeatedly performing a series of train-validation-test steps and adjusting the model parameters and features based on the model performance on the testing set. This is a form of "peeking" (see Sec. 4.4).

FIGURE 4.1: Training, validation and testing samples in a typical ML application.

The training set is the largest in size and the validation set is usually 10% - 30% of the training set size. The testing set should have sufficient samples to evaluate the trained model (in this research the same sample size is used for both the validation and the testing set). The testing set should consist of the most recent contiguous observations.

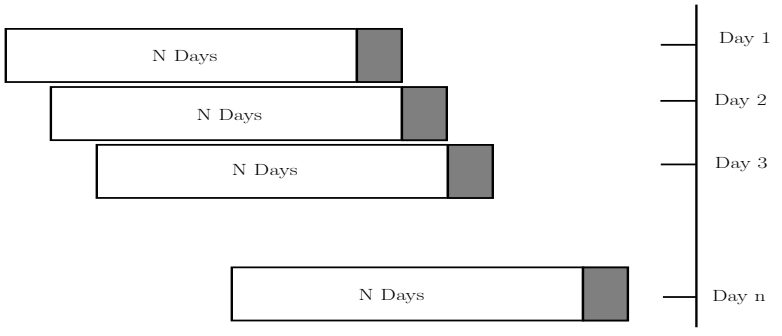

FIGURE 4.2: The sliding window approach for daily model retraining.

A more rigorous approach called the sliding or moving window (also known as walk-forward testing) is a form of online training because the model is frequently retrained.

The number of samples in the testing set determines the retraining frequency of model. For one-step ahead predictions, this means that the model can be retrained after every prediction. The data is divided into a series of overlapping training-validation-testing sets. The typical training-validation-testing concept is still present, but now only the most recent observations are used to construct models. This is depicted in Fig. 4.2 for daily model retraining. Frequent retraining is more time consuming but allows the models to adapt more quickly to the non-stationarity in time-series.

4.2.1 Data Preprocessing

The feature matrix is "cleaned" before being used by any ML model. The missing values are interpolated, the infinite values are replaced with a finite maximum threshold and the outliers are usually removed.

The training and validation sets are scaled together since the purpose of the testing set is to determine the ability of the network to generalize. However, the testing set should not be scaled with either the training or validation sets since this biases the integrity of the validation set as a final and independent check on the model.

Self-organizing map (SOM) makes use of a set of prototype vectors representing the data set and performs a "topology preserving projection" of the prototypes from input space to a low dimensional grid [81]. This ordered grid can be used as a visualization surface to identify clusters in the original data in an unsupervised manner. This has been exploited in many time-series prediction applications such as electricity load time-series prediction to identify similar data patterns, e.g. if the objective is to predict the electricity load in January, a SOM can be used to identify periods (months) that show similar load patterns to January. Training a model only on similar data patterns can not only lead to better prediction but also reduces the model training time.

4.2.2 Model Selection and Parameter Tuning

Model selection and parameter search is critical to the performance of any ML algorithm. For a support vector machine (SVM), the kernel function is such a parameter that should be chosen to maximize the performance. Radial basis function (RBF) which non-linearly maps the samples to the high-dimensional space is generally known to work well in many applications. The RBF with the kernel parameter γ, i.e. $K(x_i, x_j) = exp(-\gamma\|x_i - x_j\|^2)$, where x_i, x_j are the input training vectors, $i, j = 1, 2, \ldots, m$ was found to work well. For a SVM using a RBF kernel, the parameters C, γ needs to be tuned where C is the penalty term (see Sec. 2.8). Improper parameter selection can lead to over/under fitting [3]. Cross-validation via parallel grid-search, genetic algorithms, random search, heuristics search and inference of model parameters within the Bayesian evidence framework are some parameter search techniques. The performance of different parameter combinations is assessed by the learner performance, e.g. mean squared error.

The experiments in this thesis use parameter evaluation via a parallel grid-search on the validation data. Parameter tuning using the validation data prevents the over-fitting problem. The final performance of the learner is evaluated using the best parameters in the validation phase.

K-fold cross validation, 2-fold cross validation, leave-one-out cross validation and repeated random sampling cross validation are some popular cross validation techniques. Time-series cross validation is slightly different because the data are not independent and leaving an observation out does not remove all the associated information due to the correlations with other observations. Time-series cross validation was done as follows as suggested by Hyndman [82],

1. Fit the model to the data $y_1, \ldots, y_t$ and let $\hat{y}_{t+1}$ denote the prediction of the next observation. Then compute the error RMSE as $e_t^* = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
2. Repeat step 1 for $t = m, \ldots, n-1$ where m is the minimum number of observation needed for fitting the model
3. Compute the average RMSE from $e_{m+1}^*, \ldots, e_n^*$

4.3 Feature Selection Techniques

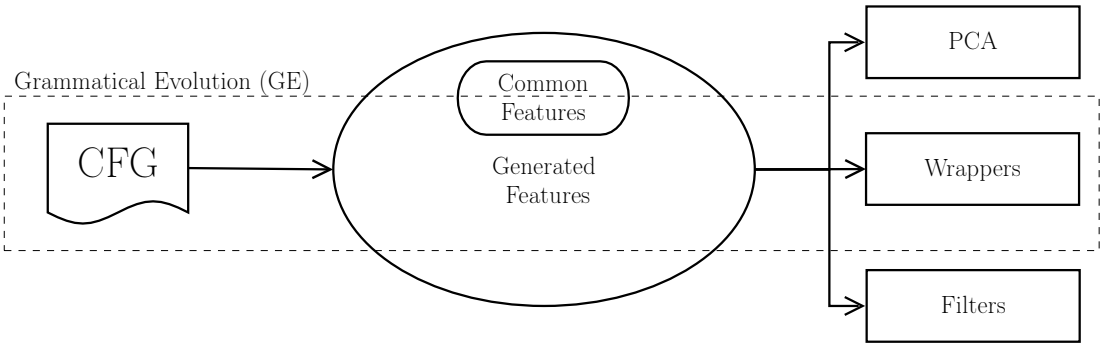

FIGURE 4.3: Different feature selection techniques explored.

This section presents the FS techniques investigated in the research. As mentioned in Chapter 3, the CFG based framework generates a large number of features parametrized by the look-back period N, window-size n and lag k. Some features have no parameters while others have one or both. It is important to select the appropriate (N, n, k) for each feature to obtain the most informative features, e.g. some specific lags (k) of a feature can have much important information than other lags of the same feature, a feature with a shorter window (n) can have better predictive power than the same feature with a longer window.

As shown in Fig. 4.3, a range of FS techniques were explored to compare the performance. The filter FS techniques (see Sec. 2.4.1) used were information gain, maximum-relevance-minimum-redundancy (mRMR), correlation and Releif. The individual feature goodness for each feature was assessed against the target variable, e.g. the closing price of stock index time-series. Once the features were ranked, an appropriate number of features was used for the prediction task. The advantage of filters is that they are extremely fast compared to wrapper based FS. As already mentioned, filter ranked features can have redundancies since similar features have equal rankings (weights). Furthermore, individual feature goodness does not translate to better predictive power when used in ML algorithms. Therefore, it is hard to determine the optimal number of features to use. PCA and wrapper approaches which are described in Sec. 4.3.1 and 4.3.2 lead to better results. The use of grammatical evolution (GE) as a wrapper based hybrid feature generation and selection technique is explained in Sec. 4.3.2.

4.3.1 Dimensionality Reduction using PCA

Principal component analysis (PCA) has been proposed as an effective FS technique in many applications. The generated features were first ranked according to different filter criteria and the number of features to perform PCA on was chosen based on a sharp cut-off threshold. PCA was applied on this chosen number of features. Different numbers of principal components accounting for different threshold in the variance of original data were used as the features and the performance was compared.

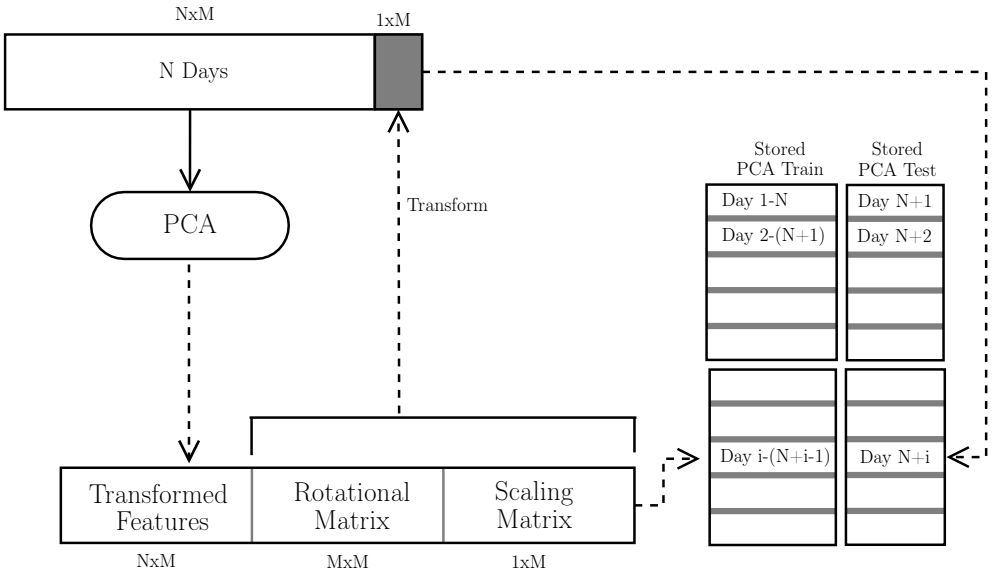

FIGURE 4.4: PCA transformations for the i^{th} day.

PCA for large matrices is a time-consuming process. When using the sliding window technique for PCA predictions m models should be constructed, where m is the testing set size. If the size of the parameter grid is 100×100, the total number of models becomes $100 \times 100 \times m$ and an equal number of PCAs should be performed. Since the calculations are required to be performed within a reasonable time, the PCA components (transformed feature matrix, rotational matrix and scaling matrix) on each training window was stored and reused as illustrated in Fig. 4.4. The testing day features were scaled and rotated using the rotational and scaling matrices of the training data as shown. This approach requires only m PCAs to be performed hence less time-consuming.

4.3.2 Feature Selection using Integer Genetic Algorithms

The generic details of canonical GAs were presented in Sec. 2.6.1 and this section elaborates specificities of adopting integer GA for FS.

Fig. 4.5 shows the proposed architecture for integer GA based FS. Chromosomes are selected from the population which are then mapped to feature subsets using the defined CFG. These subsets are evaluated and the best subsets are selected by a wrapper approach using a particular learner architecture with fixed parameter settings. The population is evolved such that the learner architecture has a minimum cross validation error.

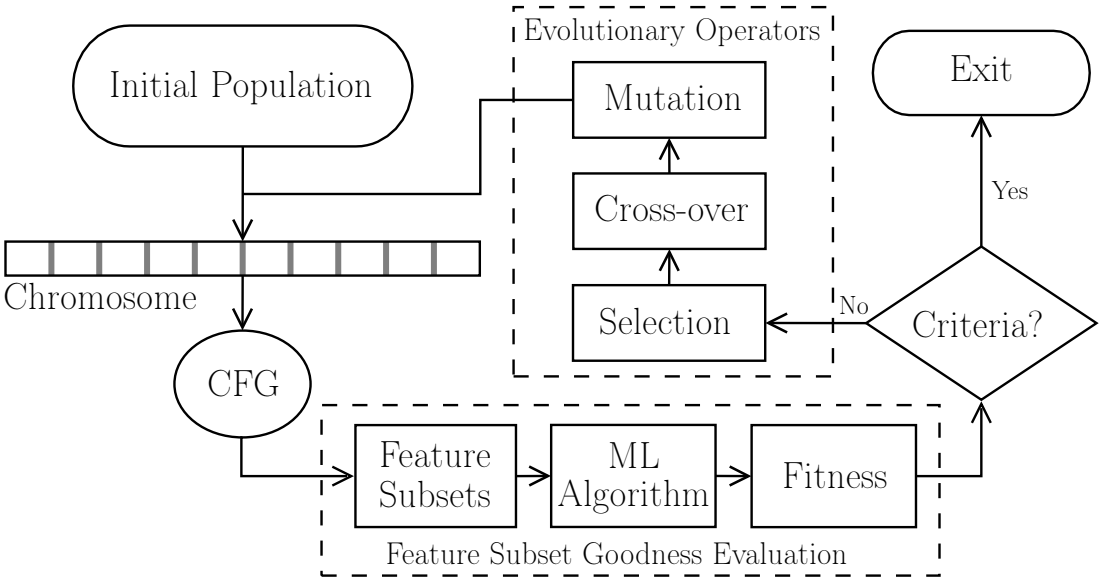

FIGURE 4.5: System architecture for integer GA based FS.

The initially generated feature space space was too large for the search algorithms to converge in a reasonable time. Therefore, when GE was not used as a hybrid feature generation and selection technique, the feature space was first shrunken by using mRMR criterion to rank and select top features. This shrunken feature space was used to seek better performing feature subsets using the wrapper approach.

In order to quickly discover better feature subsets, specific chromosomes were placed in the initial population which were known to work well in general, e.g. standard technical indicators in financial time-series prediction. This is possible since the rule sequence to

generate a specific technical indicator (or a feature in general) is known. This ensures that the initial population is healthy and encourages the generation of high performing feature subsets. The rest of the population consists of randomly generated individuals.

Consider the integer chromosome [302|237|2|451|...|871|23|657|77|549]. Each gene represents a feature number. For example if the feature space size was 1000 features, the chromosome says to select the features 302, 237, 2 and so on. Integer GA mutation was done according to the criterion $x(1 + \texttt{rand}(-0.05, 0.05))$, where x is the current codon and `rand` was a function which generated a random number between -0.05 and 0.05 (as opposed to random bit flipping in canonical GA). Cross-over and selection criteria were unchanged. The feature subset size was varied by varying the chromosome length, i.e. the number of genes. Because of the inherent random nature of the GA algorithm it is usually run multiple times before drawing any conclusions. At least 10 trials were performed in the work involved with this thesis and a final result was reported as the best, worst and the average performance for the 10 runs as in standard practice.

4.3.3 Wrapper based Feature Subset Evaluation

This section explains wrapper based FS and in particular how the GE fitness function can be crafted to select good feature subsets. The feature subset evaluation steps used in electricity load time-series prediction using GE are shown in Algorithm 2. The individuals with the lowest score were deemed to be the best. $e(Y_k)$ is the MAPE (mean absolute percentage error), given by $\frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i-\hat{y}_i}{y_i}\right|$ where n is the number of predictions, y is the target and $\hat{y}$ is the predicted value. To choose robust features providing good generalization, time-series cross validation was used (see Sec. 4.2.2). The lines 6-8 correspond to time-series cross validation and N was chosen to be 5, i.e. 5-fold time-series cross validation. $q(\cdot)$ is an assessment of the symbolic feature expression complexity, taking into account the length, the number of operations and pre-operations. Since GE generated feature subsets can include non-terminal expressions, $c(\cdot)$ was calculated based on the number of non-terminal elements N_{NT} in subset Y_k (a subset containing many non-terminal elements was considered as poor). When GE was not used, the genetic feature subset evaluation steps are specified in Algorithm 3.

Algorithm 2 Feature subset evaluation procedure using GE

Input current integer chromosome

1: Extract individual genes from the current chromosome
2: Map genes into grammar families (if necessary)
3: Derive symbolic features
4: Generate numeric feature subset Y_k for each time stamp
5: Append calendar information to Y_k (see Chap. 5)
6: **for** i in 1:N **do**
7: err[i] = ML algorithm MAPE in validation sample i
8: **end for**
9: Calculate the average MAPE $e(Y_k) = \texttt{mean}(\text{err})$
10: Calculate the final score $J(Y_k) = e(Y_k) + q(Y_k) + c(N_{NT})$

Output $J(Y_k)$

Algorithm 3 Feature subset evaluation procedure

Input current symbolic features

1: Generate numeric feature subset Y_k for each time stamp
2: Append selected expert suggested features to Y_k
3: **for** i in 1:N **do**
4: err[i] = ML algorithm performance metric in validation sample i
5: **end for**
6: Calculate the average performance metric $e(Y_k) = \texttt{mean}(\text{err})$

Output $e(Y_k)$

4.4 Avoiding Peeking

In this thesis, the term "peeking" refers to scenarios where future data is mistakenly used to get an insight to the future behaviour. This section explains 2 techniques that were implemented to ensure that this mistake is avoided.

(i) It was ensured that in prediction or parameter tuning, the implementation function had no access to future data. The safest way to avoid peeking when predicting x_{t+1} is to

ensure that all training samples are time-stamped and the prediction/tuning function to return an error/terminate if it encounters any samples with time-stamps after x_t. The wavelet decomposition should also be carried out in an incremental manner to avoid peeking as described in Appendix A.

(ii) Even if the time-stamp check is performed, there is still the risk of peeking involved in feature construction. It was ensured that a particular feature is constructed with only past values. A validity test was devised to confirm this. This test compared features constructed from all data up to time t with features constructed from all data up to time $t + k$ for some interval k. The numeric evaluation of the constructed features in two cases should be identical for the period t. If this is not the case, it implies that in constructing features for the period t, the data in interval k has been used in some way.

4.5 Summary

This chapter investigated commonly and widely used features in financial and electricity load time-series prediction. The steps involved is data partitioning for time-series prediction was also explained. The rest of the chapter described how the feature selection techniques explained in Chapter 2 were used in practice for filter and wrapper based feature selection. Since the proposed framework involves the generation of a large pool of features, there is a high probability of redundant and irrelevant features. Therefore, feature selection is equally important as feature generation. The proposed hybrid feature selection and generation algorithm using grammatical evolution was described as a technique to avoid selective feature pruning by crafting the fitness function to penalize *bad* feature subsets. Now that the theory and methodology is covered, the next chapters present and analyse the results to conclude the thesis.

Chapter 5

Results

This chapter presents how the performance of selected learning algorithms is affected by using various input feature combinations with different parameter values. The results are discussed under three sections for stock market index time-series, electricity load demand time-series and foreign-exchange client trade volume time-series. The effectiveness of the feature subsets selected using the proposed approach was compared to the performance of the same algorithms trained using commonly (and widely) used input features and other benchmarks. By "good" features, a reference is made to features that are "good for a particular ML algorithm architecture/configuration" because it is difficult to define universally good features.

5.1 System Performance on Predicting Stock Indices

A stock market index reflects the movement average of many individual stocks rather than the movement of a single stock. It can be believed that stock market indices are easier to model and predict than individual stocks, which can be very volatile.

Numerous studies, some which are summarized in Table 4.1 use technical indicators as input features for prediction. This relies on the past events of the time-series captured using technical indicators repeating to produce reliable predictions and is known as

technical analysis. This section explores the effectiveness of using such standard technical indicators as features and attempts to discover numeric feature combinations with different parameter values that can give better predictions.

The system performance was assessed on the daily closing prices of leading financial stock indices in Table 5.1. The data were downloaded from "Yahoo! Finance" using the `quantmod` package in R [83] and comprise of daily recordings between 16 October 1998 and 19 June 2006 (1900 trading days). The data preprocessing and parameter tuning was carried out as described in Chapter 3.

Symbol	Index Name	Listed Exchange
AORD	All Ordinaries Index	Australia Stock Exchange
FTSE	FTSE-100 Index	London Stock Exchange
GDAXI	DAX Index	Germany Stock Exchange
GSPC	S&P-500 (Standard and Poor's) Index	-
HSI	Hang Seng Index	Hong Kong Stock Exchange
TWII	Taiwan Weighted Stock Index	Taiwan Stock Exchange
NDX	NASDAQ-100 Index	NASDAQ Stock Market
N225	NIKKEI 225 Index	Osaka Securities Exchange
SSEC	Shanghai Stock Exchange Composite Index	Shanghai Stock Exchange
SSMI	Swiss Market Index	Six Swiss Exchange

TABLE 5.1: Major world financial indices.

The predictability of indices was first assessed using widely used standard technical indicators in Table 3.5 with the exception of SMA and WMA because EMA is much popular. The grammar families in Tables 3.6,3.7, B.1-B.5 were used to generate a large pool of features and different feature selection (FS) criteria in Sec. 2.3 were used to select *custom technical indicator* combinations with different parameter values. For a fair comparison, the number of features used in both cases was the same, 25 features.

Support vector regression (SVR) and back-propagation neural network (BPNN) were used as ML methods and the experiment configuration details are in Table 5.2. SVM with a Gaussian kernel was found to work best and the parameters were selected by a grid search parallelized on multiple cores using the `snowfall` package in R [84] (see Appendix C Sec. C.2 for selected parameters). The SVM was implemented using the `e1071` package [85]. For the BPNN, a 3 layer architecture with 8 hidden nodes was implemented [3, 25, 64, 66, 69, 70] using the Neural Network Toolbox in MATLAB®.

Prediction horizon	One-day ahead
Training period	1998-10-16 to 2004-11-11 (1500 days)
Validation period	2004-11-12 to 2005-09-01 (200 days)
Testing period	2005-09-02 to 2006-06-19 (200 days)
Support vector regression + Gaussian kernel	
SVM cost range (C)	$\{1, 5i, i = 1, 2, \ldots, 40\}$
SVM kernel hyper-parameter range (γ)	100 values spread out between $\{0.00001, 1\}$
SVM hyper-parameter (ϵ)	$0.1, 0.01, 0.001$
Back-propagation neural network	
Number of layers	3 (input, output and hidden)
Number of nodes in hidden layer	8
Other learning parameters	Optimized for best validation performance *

TABLE 5.2: Financial index prediction experiment configuration.

* The learning rate, momentum and other learning parameters were automatically decided by the MATLAB Neural Network Toolbox to get the best performance by using the least number of epochs.

The steps involved in the financial time-series prediction experiments are in Table 5.3.

(1) Select a ML algorithm, e.g. SVM
(2) Select well-known (and widely used) technical indicators (25 standard technical indicators as described above)
(3) Use the validation data to select an appropriate kernel (linear, polynomial and Gaussian kernels were considered) for the 25 standard technical indicators
(4) Optimize kernel parameters for the 25 standard technical indicators - Result 1
(5) Generate grammar features
(6) Apply pruning strategies to prune the feature space
(7) Use mRMR to shrink the grammar feature space
(8) Use integer genetic algorithm (GA) with fixed parameter valued SVM as a wrapper to select feature subsets from the shrunken space
(9) Re-optimize the kernel for the integer GA selected feature subset - Result 2

TABLE 5.3: Steps in the financial time-series prediction experiments.

To gauge the performance of the ML techniques, a comparison was made with the naive approaches such as the previous close, exponential moving average (ema) with windows size $p = 5, 10, 15$, and traditional model-based approaches: exponential time-series smoothing (ETS) and autoregressive integrated moving average (ARIMA). The performance was assessed based on the RMSE (root mean squared error) $= \sum_{i=1}^{n}(a_i - p_i)^2$ and MAE (mean absolute error) $= \sum_{i=1}^{n} |a_i - p_i|$ for validation and test periods, a_i is the

actual value and p_i is the predicted value. The results are first summarized for GSPC in Table 5.4. The integer GA FS proved computationally too burdensome for the BPNN and hence omitted.

	GSPC Result Comparison	Validation		Test	
		RMSE	MAE	RMSE	MAE
1	Naive 1 - Previous Close, C_{k-1}	-	-	7.61	5.88
2	Naive 2 - ema$(C, 5)$	-	-	9.39	7.67
3	Naive 3 - ema$(C, 10)$	-	-	11.36	9.49
4	Naive 4 - ema$(C, 15)$	-	-	12.95	10.82
5	ARIMA (1, 1, 1) *	-	-	7.58	5.87
6	ETS *	-	-	7.60	5.87
	BPNN				
	Input Features				
7	Standard Technical Indicators	7.82	6.35	7.86	6.02
9	Grammar - PCA **	7.73	6.13	7.69	6.01
10	Grammar - Releif	8.14	6.64	8.05	6.35
11	Grammar - Correlation	7.96	6.37	8.34	6.40
12	Grammar - Info. Gain	7.75	6.18	7.69	6.02
13	Grammar - mRMR	7.69	6.07	7.60	5.84
	SVM				
	Input Features				
14	Standard Technical Indicators	7.73	6.25	7.54	5.93
15	Grammar - PCA **	7.78	6.32	8.10	6.41
16	Grammar - Releif	7.97	6.69	8.03	6.29
17	Grammar - Correlation	8.24	6.14	7.68	6.08
18	Grammar - Info. Gain	7.64	6.06	7.51	5.81
19	Grammar - mRMR	7.73	6.08	7.51	5.82
20	Grammar - GA Best	**7.63**	**5.98**	**7.45**	**5.80**
21	Grammar - GA Average ***	7.71	6.07	7.49	5.83
22	Grammar - GA Worst	7.75	6.11	7.51	5.86

Table 5.4: Performance of different techniques on GSPC.

* The parameters for the ETS and ARIMA models were chosen using the `forecast` package [86] (see Appendix C C.1 for implementation details).

** For fair comparison only the first 25 principal components were used.

*** Feature subset selection using GA was repeated 10 times with different subset initializations. The results were averaged over the 10 runs.

Based on the result for GSPC the following observations were made.

- SVM and BPNN were able to outperform naive approaches and model based approaches ARIMA and ETS

- The reduction in the errors using the SVM was more pronounced compared to the BPNN
- SVM and BPNN using grammar features selected using integer GA, information gain and mRMR were found to reduce the validation and test errors compared to the same methods using standard technical indicators
- SVM using integer GA selected features was able to outperform all other approaches
- Results were consistent for RMSE and MAE in general

It can be stated that the grammar framework performed well for GSPC in comparison to model based, naive approaches and ML methods with standard technical indicators. It is evident that the generated feature space contains feature combinations with some additional information not captured by the commonly used standard technical indicators. This implies that better solution hypothesis can be formulated by grammar generated features provided that "good" feature subsets for the particular ML algorithm are selected. FS therefore plays a critical role. Based on the above observations, the results for other indices are compactly presented in Table 5.5. Only the RMSE is reported and the BPNN and SVM performance is evaluated for the standard technical indicators and selected grammar features using mRMR and integer GA. ARIMA, ETS, AR(1) and EMA = ema(C, p=5) results are also shown. Based on this empirical study involving a range of stock indices, the following can be stated. The results were consistent when the metric was changed to MAE.

- In general, the grammar features reduce the validation and test errors in the ML techniques. For example, for NDX, the reduction is by (1.52, 2.82) by using the BPNN with mRMR and (0.49, 0.78) for the SVM using integer GA. Similarly, for HSI, the best reduction is by (1.80, 2.57) using the BPNN and (7.77, 1.59) using the SVM. The grammar features reduced the validation and test errors for GSPC, HSI, SSMI, SSEC, FTSE, N225, NDX and TWII using the BPNN (8/10 indices) and GSPC, HSI, SSMI, FTSE, N225, NDX and AORD (7/10 indices) using the SVM.

	HSI		SSMI		SSEC	
Method	Validation	Test	Validation	Test	Validation	Test
ARIMA	-	138.95	-	47.96	-	19.78
ETS	-	139.95	-	47.13	-	19.78
AR(1)	-	140.24	-	47.60	-	19.71
EMA	-	183.51	-	61.42	-	28.16
BPNN (TIs)	100.00	155.43	34.07	49.22	16.57	20.86
BPNN (Grammar - mRMR)	99.71	140.55	33.56	43.98	16.33	19.02
SVM (TIs)	107.48	136.70	36.04	46.26	16.50	**18.03**
SVM (Grammar - GA Avg.)	99.71	**135.11**	33.55	**46.17**	15.79	18.54
SVM (Grammar - GA Best)	97.30	133.71	33.53	45.73	16.64	18.27
SVM (Grammar - GA Worst)	101.33	137.12	33.57	46.55	16.77	18.92

	FTSE		N225		NDX	
Method	Validation	Test	Validation	Test	Validation	Test
ARIMA	-	34.33	-	207.07	-	14.35
ETS	-	33.54	-	207.13	-	14.05
AR(1)	-	33.71	-	207.21	-	**13.98**
EMA	-	42.63	-	275.20	-	14.75
BPNN (TIs)	26.79	36.19	90.09	211.94	15.79	16.90
BPNN (Grammar - mRMR)	25.48	34.61	85.80	221.79	14.14	15.78
SVM (TIs)	26.34	40.27	94.29	203.89	14.89	15.15
SVM (Grammar - GA Avg.)	25.30	**32.64**	86.49	**203.30**	14.40	14.37
SVM (Grammar - GA Best)	24.77	32.27	85.96	202.70	14.22	14.04
SVM (Grammar - GA Worst)	26.11	33.37	86.49	204.35	15.68	14.75

	GDAXI		TWII		AORD	
Method	Validation	Test	Validation	Test	Validation	Test
ARIMA	-	44.19	-	69.52	-	29.97
ETS	-	44.08	-	69.46	-	29.98
AR(1)	-	44.18	-	69.41	-	29.97
EMA	-	58.26	-	96.55	-	38.81
BPNN (TIs)	31.77	47.73	47.97	71.02	22.43	54.18
BPNN (Grammar - mRMR)	36.69	51.37	46.88	70.94	21.22	73.98
SVM (TIs)	31.04	**43.70**	48.61	**66.85**	26.06	29.23
SVM (Grammar - GA Avg.)	29.94	44.49	46.99	67.13	21.53	**28.61**
SVM (Grammar - Best)	27.45	43.93	46.39	67.00	21.87	27.95
SVM (Grammar - GA Worst)	30.12	46.28	47.03	67.41	21.04	30.14

TABLE 5.5: RMSE for validation and test data for major stock indices using the ARIMA, ETS, AR(1), EMA, BPNN using technical indicators (TIs) and grammar feature subsets selected using mRMR and SVM using TIs and grammar feature subsets selected using mRMR and integer GA.

- The reduction in the errors using the SVM was more pronounced compared to the BPNN. Furthermore, the accuracy of the estimates was higher for the SVM in most cases. The superior performance of SVM might be due to better generalization achieved by structural risk minimization as opposed to empirical risk minimization in BPNN [3]. The poor generalization is clearly observed in BPNN for AORD. Additionally, the large number of free parameters in BPNN (hidden layers, number of hidden nodes, learning rate, momentum, epochs, transfer functions and weight initialization methods) requires extensive empirical parameter estimation to achieve optimal results.
- For NDX, the model-based approach AR(1) showed superior performance (1/10 indices). The superior performance of model based approaches in test is most likely due to some structure in the time-series where there is dependence on the history. This is supported by the RMSE which is considerably smaller compared to other indices which seems to suggest that such models provide a reasonable fit to the data. However, it can be observed that the performance of the model-based methods is only marginally better than the ML techniques.
- In general, integer GA performed the best followed by mRMR and information gain.

The SVM using grammar features was unable to outperform the SVM using standard technical indicators for SSEC, GDAXI and TWII. Although the difference is marginal at test errors 0.51, 0.79 and 0.28, there can be 3 major reasons.
(i) The technical indicators have captured most of the information hence the grammar features do not add any additional information
(ii) The FS criteria have failed to select *good* features that are still *good* in test period
(iii) The parameter tuning is far from optimal

Considering that the approach worked well for all other indices producing lower validation errors, the most probable reason is that the FS criteria has selected less informative features. The time-series is highly non-stationary hence the *goodness* of features selected using the validation period might have changed considerably in the test period. It is likely that a selected subset will not be optimal over the entire validation and test

interval. Based on the belief that the features might have been outdated, the same experiment was conducted again but with adaptive FS once every 20 days as opposed to one-time FS using a validation period of 200 days. The adaptive FS process is depicted in Figure 5.1. The test period was kept the same and adaptive FS was done by using integer GA as the FS technique. Only the SVM was used.

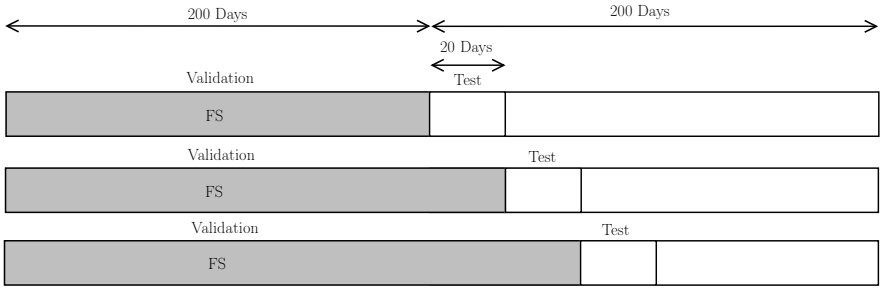

FIGURE 5.1: Adaptive FS using the sliding window technique.

Furthermore, the best performance for NDX using ML techniques was 14.37 using SVM with integer GA based FS which is poor compared to the AR(1)/ ETS results of 13.98/14.05. Therefore, adaptive FS was performed on SSEC, GDAXI, TWII and NDX to see if the results can be improved. The results for the 4 selected indices are brought up in Table 5.6.

Index	ETS	ARIMA	TIs	Grammar	Adaptive Grammar
SSEC	19.78	19.78	18.03	18.54	**17.96**
GDAXI	44.08	44.19	**43.70**	44.49	44.45
TWII	69.46	69.52	**66.85**	67.13	67.88
NDX	**14.05**	14.35	15.15	14.37	14.30

TABLE 5.6: RMSE for test period using ARIMA, ETS and SVM with features as TIs, grammar features selected only once and adaptive grammar FS using integer GA.

It is seen that the errors can be improved by adaptive FS. The SSEC test error in fact outperformed the standard TI indicators (making 8/10 indices in favour of grammar features). Adaptive FS in this manner is not possible if a fixed set of standard technical indicators and parameter values are used as features. It is expected that by repeating feature subset selection as often as necessary, the accuracy of the ML techniques can be further improved. The interval over which selection needs to be repeated will need to be

empirically decided. Although this technique is time consuming and computationally expensive, real-world applications can exploit cloud resources for fast calculations.

	Feature	Freq.		Feature	Freq.
1	**Disparity**	32	33	(lag(M, k))-(sd(lag(H, k), n))	14
2	C-(sd(lag(C, k), n))	24	34	M-(sma(ema(diff(M), n), n))	14
3	C-(sd(lag(M, k), n))	23	35	sma(L, n) + 2*(sd(L, n))	14
4	C-(sd(lag(L, k), n))	22	36	C-(H-lag(L, k)))/n	13
5	H-(sd(lag(C, k), n))	21	37	C-(H-sma(L, n)))/n	13
6	L-(sd(lag(L, k), n))	21	38	C-(sd(lag(H, k), n))	13
7	sd(diff(H), n))/(sd(delt(L), n))	21	39	H-(sd(lag(M, k), n))	13
8	(C-(M-lag(C, k)))/n	20	40	(M-(L-lag(H, k)))/n	13
9	(C-(M-lag(C, k)))/n	19	41	(sd(diff(C), n))/(sd(delt(C), n))	13
10	**Bias**	18	42	(sd(diff(C), n))/(sd(delt(M), n))	13
11	L-(sd(lag(H, k), n))	18	43	(sd(diff(L), n))/(sd(delt(M), n))	13
12	(M-(H-lag(H, k)))/n	18	44	sma(M, n) + 2*(sd(M, n))	13
13	(C-(M-lag(H, k)))/n	17	45	C-(sma(diff(H), n))	12
14	H-(sd(lag(L, k), n))	17	46	C-(sma(ema(diff(M), n), n))	12
15	L-(sd(lag(M, k), n))	17	47	H-(sma(diff(H), n))	12
16	(L-(H-lag(L, k)))/n	17	48	L-(sd(lag(C, k), n))	12
17	(sd(diff(H), n))/(sd(delt(M), n))	17	49	M-(sd(lag(L, k), n))	12
18	**Lower Bollinger Band**	17	50	(L-(H-lag(C, k)))/n	12
19	(C-(meandev(M, n)))/n	16	51	(M-(L-lag(C, k)))/n	12
20	(H-(L-lag(H, k)))/n	16	52	(sd(diff(M), n))/(sd(delt(L), n))	12
21	M-(sd(lag(M, k), n))	16	53	**Upper Bollinger Band**	12
22	M-(sma(ema(diff(L), n), n))	16	54	**Aroon**	11
23	(C-(H-lag(C, k)))/n	15	55	(C-(H-lag(M, k)))/n	11
24	C-(sd(diff(C), n))	15	56	(C-(M-lag(L, k)))/n	11
25	C-(sma(ema(diff(L), n), n))	15	57	(H-(meandev(L, n)))/n	11
26	L-(sd(diff(C), n))	15	58	**Lagged closing price**	11
27	sma(L, n) - 2*(sd(L, n))	15	59	H-(sd(diff(L), n))	11
28	(C-(H-lag(H, k)))/n	14	60	H-(sd(lag(H, k), n))	11
29	(C-(L-lag(C, k)))/n	14	61	M-(sd(lag(C, k), n))	11
30	(C-(L-lag(H, k)))/n	14	62	M-(sma(ema(diff(H), n), n))	11
31	(C-(L-sma(L, n)))/n	14	63	(sum(L, n))/(max(i^+, n))	11
32	**CLV**	14	64	sma(H, n) + 2*(sd(H, n))	11

TABLE 5.7: Technical indicators and selected grammar feature frequency.

As a by-product of the empirical study, potential candidates for technical indicators that consistently improve validation and test errors compared to the standard technical indicators can be identified. For a given stock index, FS using the GA method was repeated 10 times with different initialization and a histogram was constructed. The results for 4 indices are shown in Table C.4 Appendix C. Of the grammar generated features, the standard technical indicators (TIs) are indicated in bold. In each case only a very small number of standard TIs account for the grammar generated features. For example, for GSPC only 1 was selected. For SSMI no standard TIs were selected and for FTSE only 2 standard TIs were selected.

Table 5.7 shows the frequency of a grammar generated feature selected by aggregating the results for the 10 indices considered in Table 5.1. It is found that only 7 of the

25 standard TIs appear in the top 64 and the list is dominated by grammar generated features. The actual `n` and `k` values are not shown so that the general formulae can be seen. The other frequently selected TIs were K (9), ROC (8), OSCP (8), Slow K (8), SMA (8), ATR (6), R (6), ADO (4) and Chaikin volatility (4). In this manner, appropriate parametrized feature combinations can be identified for the time-series under consideration for a particular algorithm.

5.2 System Performance on Predicting Peak Electricity Load Data

The EUNITE dataset [20] has been extensively used as a benchmark to test load prediction algorithms. It consists of half-hourly recordings in the years 1997 and 1998, and also holiday and temperature information. It was originally published for a competition to predict the peak daily load for January 1999 using half-hourly recordings in the years 1997 and 1998. The performance metric used in electricity load time-series prediction, mean absolute percentage error (MAPE) $= \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$ where y_i, $\hat{y}_i$ are the actual and predicted values respectively was used for evaluation.

Calendar information was encoded as 7 binary values to represent the day-of-week and a single binary value for holidays as the competition winners in [71]. A self organising map (SOM) was used to identify data clusters using a peak load history window of 7 days, temperature and calendar information as SOM inputs. This showed a very strong seasonality, dividing the data into *cold* and *hot* seasons (season 1 and 2 in Fig. 5.2 respectively). Based on the SOM result, only the data from season 1 months were used to construct models in the experiments, e.g. to predict Jan. 1999, the model was constructed using Jan. 1997 - Mar. 1997, Oct. 1997 - Mar. 1998 and Oct. 1998 - Dec. 1998 data.

Two different experiments were performed on the dataset. In the first trial, the wavelet based grammar in Table 3.3 was used in a month-ahead manner of prediction. Here, after predicting a load value for January 1$^{\text{st}}$, 1999, this predicted value was used with the historical values before January 1$^{\text{st}}$ to predict January 2$^{\text{nd}}$, 1999. This process

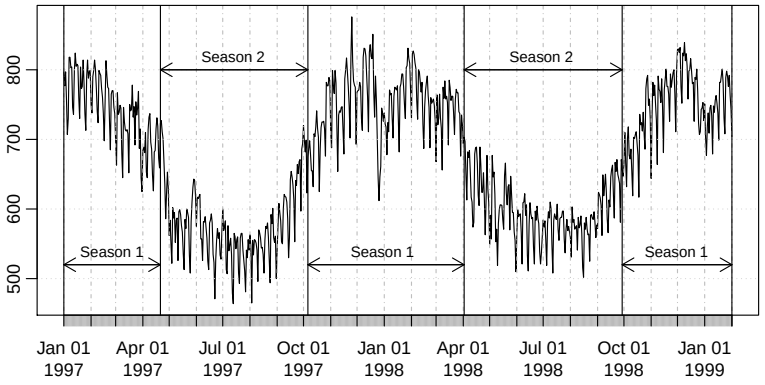

FIGURE 5.2: Clustering days to different seasons using a SOM.

was continued until the predicted load value of January 31st was obtained. For the second approach, the grammar in Table 3.10 was used. This required the previous day's high, low and close electricity loads (HLC), and was suited to a day-ahead prediction approach. Therefore, the actual historical HLC values were used in predicting each day. The usage of HLC values in electricity load prediction is unconventional but conceptually justified because the data is better represented.

For initial performance comparison, ARIMA and ETS models were chosen as analytical methods. SVM and kernel recursive least squares (KRLS) algorithm were used as kernel based ML methods. Table 5.8 summarizes the results of the different techniques used for different feature subsets on different months. It was observed that the best results were produced by a radial basis function (RBF) kernel.

Based on the features used in previous load prediction work, the performance of 5 domain-expert suggested feature subsets was compared in Table 5.8. It was observed that for a particular feature subset, the result stability for different months is poor, e.g. the feature subset $H_k, \Delta H_k, \Delta D_i^1, \Delta D_i^2, \Delta D_i^3, \Delta S_i (\forall i \in \{k-1, ..., k-6\})$ performed remarkably well only in Jan. 1999 but the average accuracy was inferior to other subsets considered. It can be suspected that some related work on the EUNITE dataset have used specific subsets in a trial and error fashion to report good performance only on the *test* dataset, Jan. 1999.

Grammatical evolution (GE) was used as the FS method as described in Sec. 4.3.3. GE has no selective feature pruning and the GE fitness function is crafted to evaluate feature subset goodness. As KRLS requires one less parameter (σ and λ) than support

vector regression (C, γ and ϵ) consuming less hyper-parameter tuning time, KRLS was chosen for feature subset evaluation as explained in Algorithm 2 Sec. 4.3.3. The optimal parameters for the KRLS feature subset evaluator, $\sigma = 256$ and $\lambda = 0.0625$ were chosen by 10-fold time-series cross validation using a peak load history window of 7 days and encoded calendar information as features. A chromosome of the proposed GE algorithm comprised of 24 codons and 25 genes, i.e. a maximum of 25 features per subset with each feature represented by 24 codons making the integer chromosome size 600. The population size was 48 and 100 generations were iterated. Roulette wheel parent selection technique was used and the proposed multi-point crossover technique in Fig. 3.7 was used. The best 2 chromosomes in each generation were considered to be elite individuals.

10 GE trials were performed and the results are presented at the bottom of Table 5.8. The HLC based grammar produced significantly better results for each of the 3 months (1.15%, 1.73% and 1.39%), and even the worst performing HLC feature subset outperformed all other methods for average day-ahead predictions. Although the best wavelet feature subset (1.81%) outperformed other domain-expert suggested feature subsets for day-ahead predictions, the average performance of each month (1.96%) was poor. However, its performance for month-ahead prediction was promising. This leads to state that the HLC grammar incorporates more information about the daily variations of load and significantly improves day-ahead predictions while the wavelet grammar elucidates long-term trends and hence is more suited for month-ahead predictions. The best one-day ahead predictions are plotted against the actual values and the cumulative MAPE is also shown in Fig. 5.3.

Table 5.9 compiles selected features from all GE trials filtered using the maximum-relevance-minimum-redundancy (mRMR) criterion. While some features are obvious and straightforward (e.g. Δ`H`, `H`, `delt(H)`, `histwin(H, 14)`), many others are not so apparent to human experts.

Method and Features	Month-ahead MAPE %				Day-ahead MAPE %			
	Jan. 1999	Dec. 1998	Nov. 1998	Avg.	Jan. 1999	Dec. 1998	Nov. 1998	Avg.
Last Year's Data	2.29	4.79	3.22	3.43	2.29	4.79	3.22	3.43
ARIMA	2.08	4.97	3.45	3.50	2.60	2.55	2.71	2.62
ETS	1.87	4.66	3.47	3.33	2.11	2.48	2.10	2.23
Linear SVM with $H_i(\forall i \in \{k, ..., k-6\})$	2.32	3.84	1.82	2.66	1.62	2.39	1.99	2.00
Polynomial SVM with $H_i(\forall i \in \{k, ..., k-6\})$	2.22	3.50	1.82	2.52	1.72	2.39	1.99	2.03
Radial SVM with $H_i(\forall i \in \{k, ..., k-6\})$	1.96	2.87	1.75	2.19	1.68	2.19	1.81	**1.89**
Polynomial KRLS with $H_i(\forall i \in \{k, ..., k-6\})$	3.93	4.71	3.46	4.03	3.91	4.62	3.38	3.97
Radial KRLS with $H_i(\forall i \in \{k, ..., k-6\})$	1.67	3.06	1.76	**2.16**	1.61	2.31	1.85	1.92
Using radial KRLS with feature subsets								
$H_i(\forall i \in \{k, ..., k-6\})$ + Temp.	3.59	3.05	2.06	2.90	2.25	2.52	1.86	2.21
$H_i(\forall i \in \{k, ..., k-6\})$	1.67	3.06	1.76	**2.16**	1.61	2.31	1.85	1.92
D_k^1, D_k^2, D_k^3, S_k	1.88	3.25	1.70	2.27	1.62	2.05	1.84	**1.84**
$H_k, \Delta H_k, \Delta D_k^1, \Delta D_k^2, \Delta D_k^3, \Delta S_k$	1.64	3.49	1.96	2.36	1.62	2.08	1.93	1.88
$H_k, \Delta H_k, \Delta D_i^1, \Delta D_i^2, \Delta D_i^3, \Delta S_i(\forall i \in \{k-1, ..., k-6\})$	1.55	3.98	2.56	2.70	1.25	2.61	2.25	2.04
HLC grammar - best of each month	-	-	-	-	1.15	1.73	1.39	**1.48**
HLC grammar - average of each month	-	-	-	-	1.34	1.89	1.56	1.60
HLC grammar - worst of each month	-	-	-	-	1.62	2.03	1.68	1.68
Wavelet grammar - best of each month	1.42	2.45	1.35	**1.84**	1.61	1.93	1.66	1.81
Wavelet grammar - average of each month	1.82	2.89	1.54	2.08	1.77	2.31	1.79	1.96
Wavelet grammar - worst of each month	2.42	3.32	1.76	2.31	1.90	2.89	1.98	2.26

TABLE 5.8: Kernel method performance comparison for different feature subsets on 3 different months.

	HLC features	Wavelet features
1	(|C-L|-|L|)/|C-L|	D_1
2	(|L|-|H-L|)/L	delt(H)
3	(ema(ΔC, 2))/(ema(delt(C), 2))	ΔD_1
4	(ema(H, 5))/(min(ema(H, 2), 2))	ΔH
5	delt(H)	ΔS
6	delt(L)	H
7	ΔH	lag(D_3,5)
8	ema(C, 3)	lag(delt(H), 2)
9	ema(C-L, 5) + sd(H-L, 3)	lag(ΔD_1, 1)
10	ema(D, 3)	lag(ΔD_3, 4)
11	ema(ema(H, 5), 3)	lag(meandev(D_1, 14), 3)
12	ema(H, 7)	lag(sd(H, 7), 3)
13	ema(H, 7) - sd(H-C, 2)	lag(S, 6)
14	ema(H-C, 2) + sd(H, 14)	lag(sma(D_1, 7), 6)
15	ema(lag(D,7), 3)	histwin(ΔH, 7)
16	ema(lag(U, 3), 3)	lag(sma(H, 5), 4)
17	lag(H, 7)	histwin(H, 14)
18	lag(U, 1)	meandev(D_1,14)
19	max(H, 14)	sd(D_1, 14)
20	max(L, 3)-lag(H, 7)	sd(H, 14)
21	max(lag(H, 1), 3)	S
22	min(D, 5)	sma(D_1, 14)
23	min(ΔC, 5)	sma(D_1,7)
24	sd(H, 5)-lag(D, 7)	sma(D_2,3)
25	sma(ΔL, 2)	sma(D_3,3)

TABLE 5.9: Selected features from 10 GE runs.

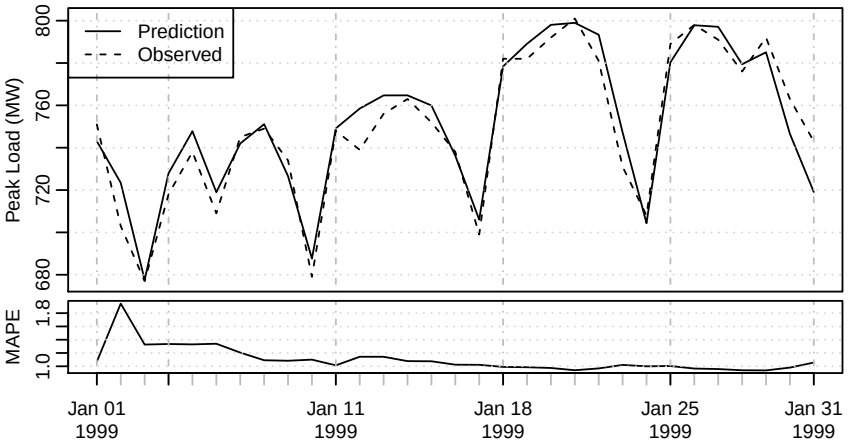

FIGURE 5.3: The best one-day ahead prediction values versus the actual values (MAPE 1.15%).

Method	MAPE (%)	Year
SVM (EUNITE Competition Winner)	1.95	2004
SVM-GA	1.93	2006
Autonomous ANN	1.75	2007
Floating search + SVM	1.70	2004
MLP-NN + Levenberg-Marquardt	1.60	2008
Auto-regressive recurrent ANN	1.57	2006
Local prediction framework + SVR	1.52	2009
Feedforward ANN	1.42	2005
Best GE result	**1.42**	-
SOFNN + Bi-level optimization	1.40	2009
LS-SVM + chaos theory	1.10	2006
MLP-NN + Differential Evolution	1.02	2011

TABLE 5.10: Benchmark results on the EUNITE dataset (Compiled from [87, 88]).

The best results of proposed methods in literature which are summarized in Table 5.10 consistently use a window of past days' peak load, temperature and calendar information. The main differences between these methods is the data partitioning technique, choice of the ML algorithm and its training method. There is some ambiguity in these results in that in the original competition, the temperature results of the testing period as well as the actual load data was not available. After the competition, some researchers might have used this information in order to improve their results and the original competition winner's result has been *claimed* to be outperformed. Furthermore, some works do not state if it's day-ahead or month-ahead predictions.

5.3 System Performance on Predicting Foreign Exchange Client Trade Volume

The foreign exchange market enables global firms to easily convert currencies crucial for trade and investment. Banks are an important part of this market, brokering for small institutes and big corporations with minimal costs thus creating specialised services for international traders. With the floating nature of exchange rates, traders and brokers are exposed to financial risk when trading this market. Exchange rate fluctuations can cause profit and loss to both brokers and traders alike. While speculative traders are attracted to this risk, corporations hedge to reduce their risk. Different financial instruments are created to reduce risk exposure of financial bodies but to remain competitive

in the already efficient foreign exchange market, banks are seeking to exploit machine intelligence to better hedge their risk. Given the exchange rate and customers history, can the risk exposure of a bank be reduced using ML techniques? The literature on industrial bank/broker hedging is scarce due to the proprietary nature of in-house developed techniques. In a hedging system, the prediction engine is a major component. This section explains the implementation of a grammar based prediction engine for foreign exchange client trade volume prediction.

Data Preprocessing : The dataset contained transaction details for different currency pairs. The majority of the transactions were AUD/USD. These were spot transactions and the data were irregular. The AUD/USD transactions were aggregated to form hourly net trade volume time-series V. Firstly, net trade volume prediction task was formulated as a binary classification problem. $V > 0$ meant that the net trade volume was "buy" and vice versa. The out-sample test results for 6 months are presented in Appendix C Table C.3. The first observation is that the data is unbalanced (1460 net buys and 604 net sells) for the 6 months considered. Therefore, the balanced accuracy is a more suitable measure to assess the model performance and the 6 month average is 51.06%. It was then attempted to deal with a more balanced dataset and a threshold *Thresh* was decided to construct classification labels as,

$$class = \begin{cases} -1 & \text{V}< -Thresh \text{ (1085 samples)} \\ 0 & -\text{Thresh}<\text{V}<Thresh \text{ (959 samples)} \\ +1 & \text{V}>Thresh \text{ (660 samples)} \end{cases}$$

Candidate Features : Unlike stock market index and electricity load data, previous knowledge of the types of features to use was unavailable. Therefore, after manually picking features the best results were achieved by using the following features.

- 6 external features that can affect the trading behaviour of small and medium scale foreign exchange traders, e.g. T-bill rates, interest rates, bond prices, first difference of the AUD/USD foreign exchange rate, etc. obtained from SIRCA [89]
- A client trading volume history window of 16 hours, i.e. lag(V,1), ..., lag(V,16)

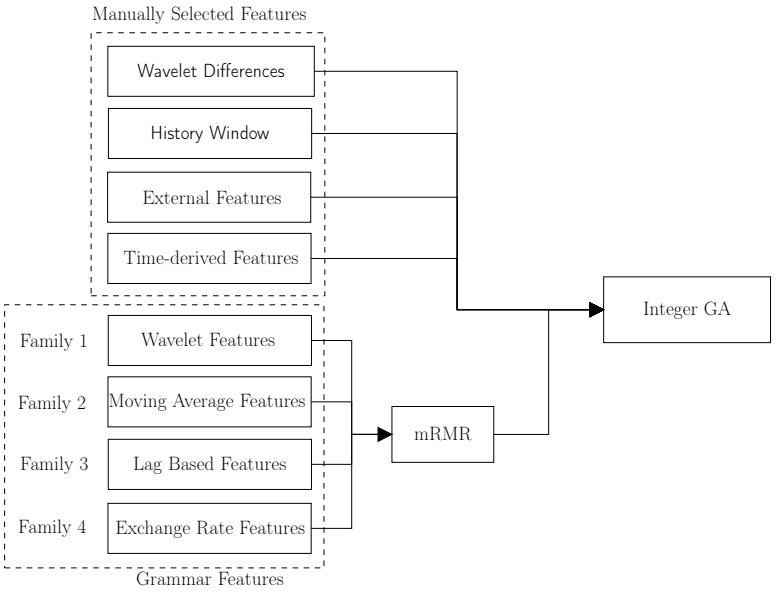

FIGURE 5.4: Features considered for client trade volume prediction. Grammar features were supplementary to the manually selected features.

- Time-derived features : Month of the year (1-12), day of the week (1-7), day of the month (1-31) and hour of day(0-23)
- First difference of the trade volume wavelets, Δ(D1), Δ(D2), Δ(D3) and Δ(S)

The grammar features considered with varying lags k and moving window lengths and look-back periods n for predicting foreign exchange client trade volume were,
(i) Wavelet features: Grammar features derived from the 3-level decomposition of the client trade volume time-series for different n, k
(ii) Moving average features: Grammar features derived from a combination of moving averages applied on the client trade volume time-series for different n
(iii) Lag-based features: Grammar features derived using different lags, maximum and minimum values for different n, k values of the client trade volume time-series
(iv) Exchange rate related features: Technical indicator type grammar features using the exchange rate which can be believed to be a driving factor of client trades

From each of the 4 grammar families, 10 features were first selected using mRMR to create a pool of 40 features and the binary GA wrapper was used to select feature subsets from this pool as in Fig 5.4. The resultant best chromosome encoding represented the selected features and the number of '1's in the chromosome was the feature subset

size. Adaptive FS was performed weekly. SVM was used with a RBF kernel and the parameters were also re-optimized weekly. The predictions were done in a rolling-window manner to get day-ahead predictions. The SVM parameters were optimized.

Confusion Matrix: Columns represent predicted class and rows represent actual class, e.g. for a binary classification task, $\left(\begin{smallmatrix}15 & 33\\ 56 & 45\end{smallmatrix}\right)$, TN = 15, FP = 33, FN = 56 and TP = 45.

The out-sample test results for 8 months are presented in Table 5.12 and for 6/8 months, the grammar improved the 3-class classification accuracies. The overall balanced accuracy improved from 38.03% to 38.91% and the overall hit ratio improved from 42.05% to 42.86%. A closer inspection of the confusion matrix showed that the class -1 prediction accuracy remained the same at 696/1085 while class 0 accuracy improved from 357/959 to 372/959 and class +1 accuracy improved from 84/660 to 91/660.

	Feature	Freq.	Family
1	$i^-(4)$	13	3
2	`sma(Δ(F),16) - ema(Δ(F),4)`	11	4
3	$(i^-(8)$ - $i^-(4))/4$	11	3
4	$i^+(4)$	10	3
5	$i^-(4)$`/(lag(min(V,16), 16)`	9	3
6	$i^+(16)$`/(lag(min(V,16), 4)`	9	3
7	$(i^-(16)$ - $i^-(4))/4$	8	3
8	`ema(lag(F,8),4)/sma(lag(F,8),4)`	8	4
8	`sma(Δ(V),32)/ema(Δ(V),4)`	8	2
10	`sma(Δ(F),32)/sma(Δ(F),4)`	8	4
11	`wma(Δ(F),4)/ema(Δ(F),4)`	8	4
12	$(i^-(16)$ - $i^-(8))/4$	8	3
13	$(i^+(4)$ - $i^-(8))/4$	8	3
14	`(min(V,16)-min(V,8))/4`	8	3
15	`(min(V, 4)-V))/4`	7	3
16	`(min(V,8)-min(V,4))/4`	7	3
17	$i^+(16)$`/(lag(min(V,16), 16)`	6	3
18	$i^-(8)$	6	3
19	`V/min(V,16)`	6	3
20	`(min(V,16)-min(V,4))/4`	5	3
21	`((min(V,16)-V))/4`	5	3
22	$i^-(4)$`/(lag(min(V,16), 4)`	4	3
23	$i^-(8)$`/(lag(min(V,16), 16)`	4	3
24	$i^+(8)$`/(min(V,16)`	4	3
25	$(i^-(16)$ - $i^-(8))/4$	4	3
26	`Δ(V) - sma(Δ(V),8)`	4	2
27	`ema(Δ(V),16) - wma(Δ(V),16)`	4	2
28	`sma(Δ(V),16)/ema(Δ(V),16)`	4	2
29	$i^-(16)$	4	2
30	`(wma(`$\lvert V\rvert$`, 4))/(wma(Δ(V), 4))`	4	2

TABLE 5.11: Selected grammar feature frequency for different families.

The grammar based prediction system also enabled the identification of the best features to use with the selected ML algorithms. The parameters of the moving averages, look-back periods of maximum, minimum values and the appropriate lags to use can also be empirically decided in this manner, e.g. a window-size of 4 was associated with a large number of features which means that recent history is more important. The best features discovered for the optimized SVM are in Table 5.11. `V` is the net trade volume and i^+, i^- are the days elapsed since the last highest net trade volume and the last lowest net trade volume. `F` is the AUD/USD foreign exchange rate.

An interesting observation is the absence of wavelet based grammar features on the top 30 features. Therefore, when attempting to predict this type of data, the effectiveness of wavelets should be further investigated. Net trade volume lag-based features were dominant and the exchange rate seemed to influence trade volumes as well.

5.4 Summary

This section applied the proposed feature engineering algorithm to predict stock index time-series, electricity load time-series and foreign exchange client trade volume time-series. The non-stationary and non-linear nature of these time-series makes prediction a challenging task. The feature subsets generated by the proposed framework were able to improve over carefully crafted manual features *in general*. By supplementing the manually selected features with the grammar features, further improvements were achieved. Although the improvements were marginal in some cases, it is a significant achievement provided that the time-series considered are hard to predict in the first place. The best parametrized feature combinations to use with the particular ML algorithms were identified along with their dominant lags, most useful window-sizes and look back periods, which is of utmost importance when predicting time-series. Such feature parameters can not be automatically elucidated by standard kernel functions or manual feature crafting. The importance of feature engineering in real world prediction problems was realized. The proposed framework was used to harness the best performance from established ML algorithms and standard kernels in a practical sense. The results justify the usage of automatic feature generation because it can improve the results over human expert defined parametrized feature combinations.

Month	Method	Confusion Matrix	Balanced Accuracy	Hit Ratio
1	Without Grammar	84 41 16 55 41 17 56 31 11	35.69	38.64
	With Grammar	91 31 19 61 35 17 50 30 18	**37.96**	**40.91**
2	Without Grammar	80 31 18 57 33 29 60 9 19	37.11	39.29
	With Grammar	77 36 16 57 43 19 56 15 17	**38.38**	**40.77**
3	Without Grammar	65 63 2 53 87 0 31 50 1	**37.79**	**43.47**
	With Grammar	63 62 6 52 87 1 28 53 1	37.27	42.90
4	Without Grammar	168 4 0 94 8 0 57 5 0	35.17	**52.38**
	With Grammar	158 14 0 86 16 0 57 5 0	**35.85**	51.79
5	Without Grammar	100 32 6 68 31 10 62 19 8	36.63	41.37
	With Grammar	97 29 12 64 31 14 57 20 12	**37.40**	**41.67**
6	Without Grammar	94 11 30 62 11 39 67 7 31	**36.32**	**38.64**
	With Grammar	84 18 33 63 12 37 57 15 33	34.79	36.65
7	Without Grammar	59 60 14 64 60 13 40 20 6	32.42	37.20
	With Grammar	64 56 13 59 69 9 40 21 5	**35.35**	**41.07**
8	Without Grammar	46 58 3 31 86 10 26 36 8	40.71	46.05
	With Grammar	62 38 7 39 79 9 40 25 5	**42.43**	**58.03**
1-8	Without Grammar	696 300 89 484 357 118 399 177 84	38.03	42.05
	With Grammar	696 284 105 481 372 106 385 184 91	**38.91**	**42.86**

TABLE 5.12: Out-sample results (%) using binary GA to predict the client trade volume classification for 8 months using SVM.

Chapter 6

Conclusion

The performance of machine learning (ML) techniques is highly dependant on the formalism in which the solution hypothesis is represented. The features used to formalize this hypothesis should be engineered carefully for optimal performance. This is usually done by domain experts which often leads to good results. This thesis investigated if an automatic feature generation framework that can generate expert suggested features and many other parametrized features can be used to improve the performance of ML methods in time-series prediction.

The feature generation framework using context-free grammars (CFGs) was proposed in Chapter 3. A key feature was to enable experts to provide guidelines to the system but for the computer to do all the grunt work of feature engineering. Grammar families were proposed as a way to organize and constrain the feature space. Pruning strategies that can be used to eliminate features without compromising the effectiveness of the feature generation were discussed. The merits of the proposed framework are, (i) parametrized features can be engineered in a systematic manner based on time-series dynamics (non ad-hoc parametrization) (ii) the user is able to design a sufficiently large grammar to capture as much information as possible with a manageable feature space (iii) the user can incorporate domain knowledge by choosing appropriate derived variables and production rules. It was realized that not only the expansion but mining the generated feature space to discover better feature subsets is also important.

Chapter 4 presented how different feature selection (FS) criteria were explored to mine the expanded feature space. The wrapper based techniques were found to work better than filter based approaches. Integer genetic algorithms, maximum-relevance-minimal-redundancy and information gain criteria performed the best.

Grammatical evolution was proposed as a hybrid feature generation and selection algorithm. This hybrid algorithm combines the feature generation and selection phases and avoids the need for selective feature pruning strategies. The fitness function of the genetic algorithm was crafted to penalize *bad* feature subsets and select *good* feature subsets using a wrapper system.

The empirical evaluation on real world financial and electricity load time-series proved that the proposed approach can lead to improvements of the performance of particular ML algorithms that use certain expert suggested feature subsets. The expert suggested features for financial time-series prediction were chosen as widely used standard technical indicators. For electricity load time-series, different features used in previous works such as history windows, wavelet transforms and differenced values were chosen as expert suggested features.

The classification accuracy of foreign exchange client trade volume time-series was also improved by using grammar families. Because of the absence of literature on predicting such time-series, features engineered manually and exogenous features were defined as expert features.

Although the proposed approach is not *guaranteed* to produce better out-sample results, it can certainly be used as a technique to craft features. The best approach is to begin with expert suggested features, and use the system generated feature combinations as supplementary feature subsets to add value to the predictor. An experienced user can monitor features that consistently rank in the top and consider them as *good* features. The feature parameters and the optimal feature subsets are subject to change due to the non-stationary nature of real world time-series hence it is intuitive that an expert system can assist a human expert in selecting the best feature combinations automatically.

In a nutshell, the proposed framework is a systematic approach to generate a rich class of features from an expert designed set of rules that can yield considerable improvement in the performance of any ML technique in a practical sense for critical applications.

Future Work

The grammar developed in this work was designed to generate only a limited feature space. For example, the grammar for stock market index time-series generated only a limited set of technical indicators. Advanced conditional technical indicators can be generated by combining probabilistic context-free grammar and genetic programming. The grammar can also be applied to other transformations of the time-series, e.g. the grammar for electricity load time-series can have other effective transformations such as empirical mode decomposition (EMD).

In addition, robust FS for non-stationary time-series using ensemble approaches and non-linear dimension reduction techniques can also be investigated further. It was discovered that wrapper approaches are better than filter approaches hence advanced FS algorithms should be developed that will ensure that the selected feature subsets are robust for non-stationary time-series which will aid in minimizing the the gap between the best and worst performing feature subsets, e.g. in Table 5.8.

It was also discovered that adaptive FS can lead to better results. This can be supplemented with adaptive parameter optimization to account for the non-stationary nature of time-series. Although this is time-consuming, small improvements can yield benefits in critical applications such as financial time-series prediction.

The proposed approach is general and is not limited to time-series investigated in the thesis. Other time-series can be explored and appropriate grammars can be developed. By using the proposed approach, an insight to the feature combinations that work well with particular ML algorithms can be discovered hence the experts can use the proposed feature generation framework in any application to supplement their own set of hand picked features.

The approach is computationally expensive hence certain experiment parameters were restricted, e.g. number of generations in genetic algorithm, tuning parameter grid size etc. A cloud based parallel implementation can speed up the feature generation and selection process in real world applications.

Finally, as in Fig.6.1, the approach can be viewed as an abstraction layer to reduce the effort associated with finding the best feature-ML architecture-parameter combination which is the most time-consuming aspect in developing a ML system. The system can be used to automate tuning and feature engineering to enhance the performance of existing ML systems, conveniently.

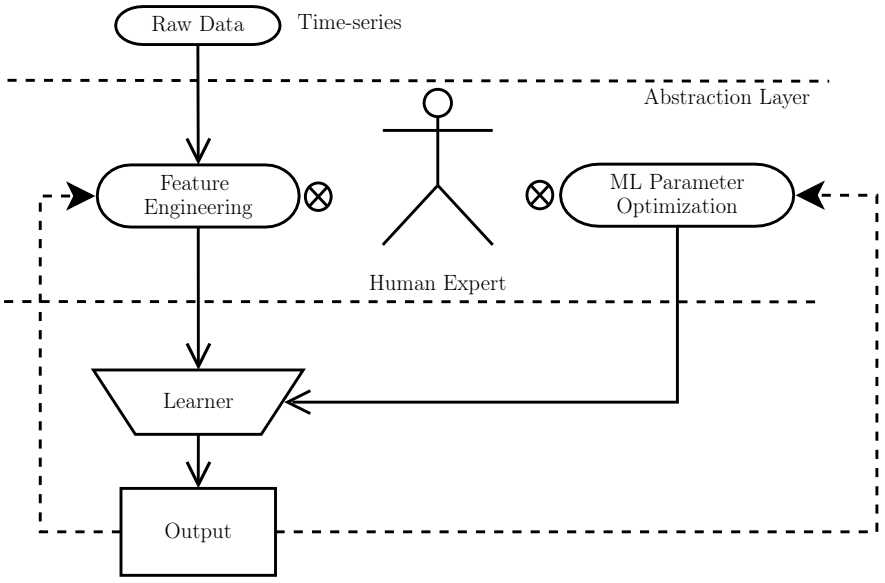

FIGURE 6.1: The proposed system viewed as an abstraction layer.

Appendix A

Wavelet Transformations

Discrete Wavelet Transform (DWT)

This appendix is loosely based on the presentation outline in [90].

DWT decomposes a time-series into a set of orthonormal basis functions (wavelets) producing a set of wavelet coefficients. Each coefficients captures information at different frequencies at distinct times. A function f can be expanded using a mother wavelet function Ψ as,

$$f(t) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} w_{jk} 2^{j/2} \Psi(2^j t - k) \tag{A.1}$$

The functions $\Psi(2^j t - k)$ are mutually orthogonal. The coefficient w_{jk} conveys information about the behaviour of f at a scale around 2^{-j} near time $k \times (2^{-j})$. Application of DWT for time-series analysis suffers from a lack of translation invariance which is undesired. The highly redundant maximal overlap discrete wavelet transform (MODWT) is used to address this issue.

Maximal Overlap DWT (MODWT) and À Trous Filtering

When MODWT is applied on an N sample input time-series it will produce N samples for each resolution level aligning with the original time-series in a meaningful way. For a time-series X with N samples, the j$^{\text{th}}$ level MODWT wavelet ($\tilde{W}_j$) and scaling ($\tilde{V}_j$)

coefficients are,

$$\tilde{W}_{j,t} = \sum_{l=0}^{L_{j-1}} \tilde{h}_{j,l} X_{t-l \bmod N} \tag{A.2}$$

$$\tilde{V}_{j,t} = \sum_{l=0}^{L_{j-1}} \tilde{g}_{j,l} X_{t-l \bmod N} \tag{A.3}$$

where $\tilde{h}_{j,l} = h_{j,l}/2^{(j/2)}$ are the MODWT wavelet filters and $\tilde{g}_{j,l} = g_{j,l}/2^{(j/2)}$ are the MODWT scaling filters. Hence the additive decomposition or the multi-resolution analysis (MRA) is expressed by,

$$X = \sum_{j=1}^{J_0} \tilde{D}_j + \tilde{S}_{J_0} \tag{A.4}$$

where,

$$\tilde{D}_{j,t} = \sum_{l=0}^{N-1} \tilde{h}_{j,l} \tilde{W}_{j,t+l \bmod N} \tag{A.5}$$

$$\tilde{S}_{j,t} = \sum_{l=0}^{N-1} \tilde{g}_{j,l} \tilde{V}_{j,t+l \bmod N} \tag{A.6}$$

A set of coefficients D_j with N samples can be obtained at each scale j. These coefficients capture the local fluctuations of X at each scale. S_{J_0} captures the overall "trend" of X. Adding D_j to S_{J_0}, for $j = 1, 2, \ldots, J_0$, gives an increasingly more accurate approximation of the now decomposed X. This additive reconstruction allows to predict each wavelet sub-series (D_j, S_{J_0}) separately and add the individual predictions to generate an aggregate prediction.

In time-series prediction, the MODWT should be performed incrementally where a wavelet coefficient at a position n is calculated from the samples at positions less than or equal to n, but never larger. If this is not adhered it is a form of peeking. The À Trous filtering scheme [91] briefly described below is used to do this. Consider a signal $X(1), X(2), \ldots, X(n)$, where n is the present time point and perform the following steps:

1. For index k sufficiently large, carry out the MODWT transform (A.5), (A.6) and (A.7) on $X(1), X(2), \ldots, X(n)$.

2. Retain the coefficient values as well as the smooth values for the kth time point only: $D_1(k), D_2(k), \ldots, S_5(k)$. The summation of these values gives $X(k)$.

3. If $k < n$, set $k = k + 1$ and go to Step 1. This process produces an additive decomposition of the signal $X(k), X(k+1), \ldots, X(n)$, which is similar to the À Trous wavelet transform decomposition on $X(1), X(2), \ldots, X(n)$.

Appendix B

Production rules for some standard technical indicators

This appendix presents the feature generation flow for the technical indicators R, Aroon, RSI and Chaikin volatility respectively using grammar families 3, 4, 5 and 6.

Family 3

$\mathcal{N}$ = {L1, L2, L3}
$\mathcal{T}$ = {− , ÷, `lag`, `sma`, `meandev`, `sum`, H_h, L_l, `C`, `n`, `k`, (,) }
$\mathcal{S}$ = {L3}

Production rules : $\mathcal{R}$

$\langle L3 \rangle$	::=	$(\langle L2 \rangle) \div (\langle L2 \rangle)$ | `sma`$(\langle L2 \rangle$, `n`$)$ | $\langle L2 \rangle$	(1.a), (1.b), (1.c)
$\langle L2 \rangle$	::=	$\langle L1 \rangle -$ `lag`$(\langle L1 \rangle$, `k`$)$ | `sma`$(\langle L1 \rangle$, `n`$)$	(2.a), (2.b)
	|	`meandev`$(\langle L1 \rangle$, `n`$)$ | `sum`$(\langle L1 \rangle$, `n`$)$ | $\langle L1 \rangle$	(2.c), (2.d), (2.e)
$\langle L1 \rangle$	::=	H^+ | L^- | `C`	(3.a), (3.b), (3.c)

Table B.1: Grammar family 3.

Family 4

$\mathcal{N}$ = {L1, L2, L3, L4, L5}
$\mathcal{T}$ = {− , ÷, `lag`, `ema`, `sma`, `meandev`, `sum`, `H`, `L`, `C`, `n`, `k`, i^+, i^-, (,) }
$\mathcal{S}$ = {L5}

Production rules : $\mathcal{R}$

$\langle L5 \rangle$	::= ($\langle L3 \rangle$ ÷ $\langle L4 \rangle$) | ($\langle L3 \rangle$ ÷ `N`) | $\langle L4 \rangle$	(1.a), (1.b), (1.c)
$\langle L4 \rangle$	::= `ema`($\langle L1 \rangle$, `n`) | `sum`($\langle L1 \rangle$, `n`) | `max`($\langle L1 \rangle$, `n`)	(1.d), (1.e)
	| `min`($\langle L1 \rangle$, `n`) | ($\langle L1 \rangle$) ÷ `N` | $\langle L1 \rangle$	(2.a), 2.b), (2.c)
$\langle L3 \rangle$	::= $\langle L2 \rangle$ − `ema`($\langle L2 \rangle$, `n`) | `ema`($\langle L2 \rangle$, `n`) | `meandev`($\langle L2 \rangle$, `n`)	(3.a), (3.b)
	| `sum`($\langle L2 \rangle$, `n`) | `max`($\langle L2 \rangle$, `n`) | `min`($\langle L2 \rangle$, `n`) | $\langle L1 \rangle$	(3.c), (3.d), (3.e), (3.f)
$\langle L2 \rangle$	::= `H` | `L` | `C`	(4.a), (4.b), (4.c)
$\langle L1 \rangle$	::= i^+ | i^-	(5.a), (5.b)

TABLE B.2: Grammar family 4.

Family 5

$\mathcal{N}$ = {L1, L2, L3}
$\mathcal{T}$ = {− , ÷, `lag`, `ema`, `sma`, `meandev`, `sum`, `H`, `L`, `C`, `n`, `k`, i^+, i^-, (,) }
$\mathcal{S}$ = {expr}

Production rules : $\mathcal{R}$

$\langle L5 \rangle$	::= $\langle L3 \rangle$ ÷ ($\langle L3 \rangle$+$\langle L4 \rangle$) | $\langle L3 \rangle$ ÷ ($\langle L3 \rangle$−$\langle L4 \rangle$)	(1.a), (1.b), (1.c)
$\langle L4 \rangle$	::= `ema`($\langle L1 \rangle$, `n`) | `sum`($\langle L1 \rangle$, `n`) | `meandev`($\langle L1 \rangle$, `n`) | `max`($\langle L1 \rangle$, `n`)	(2.a), (2.b)
	| `min`($\langle L1 \rangle$, `n`) | `delt`($\langle L1 \rangle$)	(2.c), (2.d), (2.e)
$\langle L3 \rangle$	::= `ema`($\langle L2 \rangle$, `n`) | `sum`($\langle L2 \rangle$, `n`) | `meandev`($\langle L2 \rangle$, `n`) | `max`($\langle L2 \rangle$, `n`)	(3.a), (3.b)
	| `min`($\langle L2 \rangle$, `n`) | `delt`($\langle L2 \rangle$)	(3.c), (3.d), (3.e)
$\langle L2 \rangle$	::= F^- | `D`	(4.a), (4.b)
$\langle L1 \rangle$	::= F^+ | `U`	(5.a), (5.b)

TABLE B.3: Grammar family 5.

Family 6

$\mathcal{N}$ = {L1, L2, L3}
$\mathcal{T}$ = {− , ÷, `lag`, `ema`, `sma`, `meandev`, `sum`, `H`, `L`, `C`, `n`, `k`, i^+, i^-, (,) }
$\mathcal{S}$ = {L4}

Production rules : $\mathcal{R}$

$\langle L4 \rangle$	::= ($\langle L1 \rangle$ − $\langle L3 \rangle$) ÷ ($\langle L2 \rangle$ − $\langle L3 \rangle$) | $\langle L2 \rangle$ | $\langle L3 \rangle$	(1.a), (1.b), (1.c)
$\langle L3 \rangle$	::= `sma`($\langle L1 \rangle$, `n`) - 2× `sd`($\langle L1 \rangle$, `n`)	(2.a)
$\langle L2 \rangle$	::= `sma`($\langle L1 \rangle$, `n`) + 2× `sd`($\langle L1 \rangle$, `n`)	(3.a)
$\langle L1 \rangle$	::= `H` | `L` | `C` | `H-L` | `H-C` | `C-L`	(4.a), (4.b), (4.c), (4.d), (4.e), (4.f)

TABLE B.4: Grammar family 6.

Family 7

$\mathcal{N}$ = {L1, L2, L3}
$\mathcal{T}$ = {− , ÷, `lag`, `sma`, `H`, `L`, `C`, `M`, (,) }
$\mathcal{S}$ = {expr}

Production rules : $\mathcal{R}$

$\langle L3 \rangle$	::=	$(\langle L2 \rangle) \div (\langle L2 \rangle) \mid (\langle L2 \rangle - \langle L2 \rangle) \mid \langle L2 \rangle$	(1.a), (1.b), (1.c)
$\langle L2 \rangle$	::=	$\texttt{ema}(\langle L1 \rangle, \texttt{n}) \mid \texttt{sma}(\langle L1 \rangle, \texttt{n}) \mid \texttt{wma}(\langle L1 \rangle, \texttt{n})$	(2.a), (2.b), (2.c)
		$\mid \texttt{sma}(\texttt{ema}(\langle L1 \rangle, \texttt{n}), \texttt{n}) \mid \langle L1 \rangle$	(2.d), (2.e)
$\langle L1 \rangle$	::=	$\texttt{lag}(\texttt{V}, \texttt{k})$	(3.a)

TABLE B.5: Grammar family 7.

Fig. B.1 shows how the indicator R is generated. In Chapter 3.3.2, it was explained that features were generated with 3 values $n = 6, 12, 24$ for H^+, L^-, i^+ and i^-.

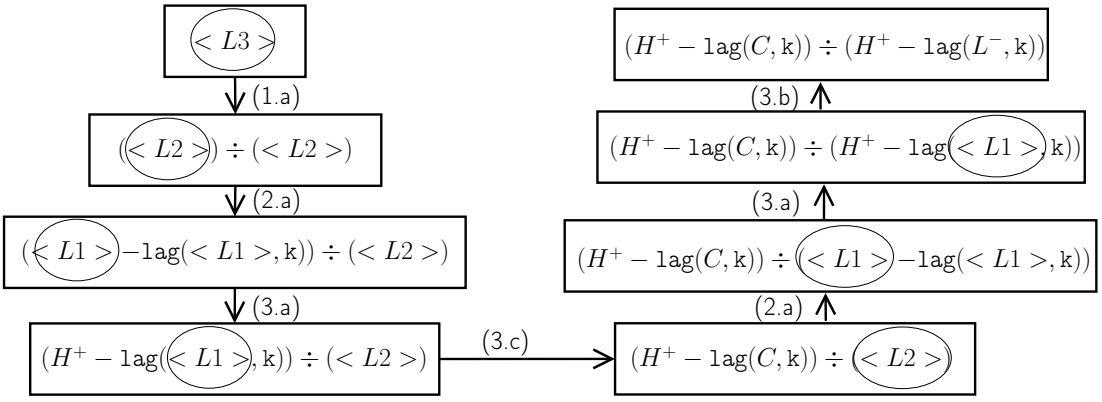

FIGURE B.1: Feature generation flow for the technical indicator R.

Aroon type technical indicators can be easily generated as follows. Although the original Aroon up indicator is $(N - i^+)/N$ it is understood that the information is captured by simple division i^+/N, hence the subtraction from 1 is ignored.

Invoking rule (1.b) : <*L5*> ::= (<*L3*>) ÷ `N`

Invoking rule (3.f) : <*L3*> ::= <*L1*>

Invoking rule (5.a) : (<*L1*>) ::= i^+

Invoking rule (5.b) : (<*L1*>) ::= i^-

Fig. B.2 shows how the indicator RSI is generated. By defining RS = ema(U, n)/ema(D, n), RSI=RS/(1+RS). By expanding this RSI = ema(U, n)/(ema(U, n) + ema(D, n)).

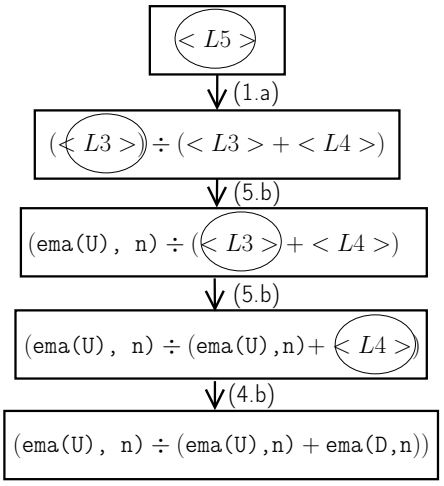

FIGURE B.2: Feature generation flow for the RSI indicator.

The upper band of Chaikin volatility is generated as follows.

Invoking rule (1.c) : <*L4*> ::= (<*L3*>)

Invoking rule (2.a) : <*L3*> ::= `sma`(<*L1*>, `n`) + 2×`sd`(<*L1*>, `n`)

Invoking rule (4.d) : <*L1*> ::= `H-L`

The lower band can be generated similarly by using rule (2.b) instead of (2.a).

Appendix C

Results

C.1 Model-based Approaches on Stock Indices

Autoregressive integrated moving average (ARIMA) and exponential time-series smoothing (ETS) were used as pure time-series model benchmarks. `auto.arima()`, `Arima()` and `ets()` functions in the `forecast` package in R [86] were used for implementation. ARIMA model order was automatically decided using `auto.arima()` for day-ahead predictions in the validation dataset. The error type, trend type and seasonal type of ETS models were automatically decided using `ets()`. Table C.1 gives the ARIMA model orders for the 10 indices.

GSPC	1,1,1	SSMI	4,1,5	N225	2,1,0	GDAXI	4,1,4	HSI	2,1,2
FTSE	4,1,4	SSEC	0,1,0	NDX	4,1,4	TWII	1,1,0	AORD	0,1,0

Table C.1: ARIMA model order for the 10 indices.

C.2 SVM Parameter Values

The C and γ parameter of the SVM were chosen using a parallelized grid search. The grid range is specified in Table 5.2. Table C.2 gives the optimal parameters selected for SVM using technical indicators as features.

Symbol	C	γ	Symbol	C	γ
AORD	190	$5.823249e-05$	TWII	140	0.00049476
FTSE	70	0.00049476	NDX	190	0.0002474412
GDAXI	200	$8.99928e-05$	N225	190	0.000109989
GSPC	110	0.00014141	SSEC	190	$7.614881e-05$
HSI	200	0.0001979628	SSMI	170	$8.249402e-05$

TABLE C.2: Parameters for SVM using TIs as features with $\epsilon = 0.01$.

C.3 Foreign Exchange Client Net Trade Volume Binary Classification

Evaluation Criteria : For unbalanced data sets, accuracy alone is not a good criterion for evaluating a model performance. Therefore a range of measures were evaluated. Define TP ≡ True Positive, TN ≡ True Negative, FP ≡ False Positive and TN ≡ True Negative. Accuracy ≡ Hit Ratio (HR) = (TP + TN)/(TP + FP + TN + FN). Sensitivity = TP/(TP+FN), Specificity = TN/(TN+FP) and Balanced Accuracy (BAC) = (Sensitivity + Specificity)/2. Precision = TP/(TP+FP), Recall = TP/(TP+FN) and F-score = 2×Precision×Recall/(Precision + Recall).

Evaluation Criterion	Month 1	Month 2	Month 3	Month 4	Month 5	Month 6
Confusion Matrix	210 30 97 15	206 26 87 17	235 13 97 7	261 8 65 2	234 5 96 1	225 7 109 11
Sensitivity	13.39	16.34	6.73	2.98	1.03	9.16
Specificity	87.50	88.79	94.76	97.03	97.91	96.98
Balanced Accuracy	50.44	52.57	50.74	50.01	49.47	53.07
Precision	33.33	39.53	35.00	20.00	16.67	61.11
Recall	13.39	16.35	6.73	2.98	1.03	9.17
F-Score	19.11	23.13	11.29	5.19	1.94	15.94
Hit Ratio	63.92	66.37	68.75	78.27	69.94	67.04

TABLE C.3: Out-sample results (%) using integer GA to predict the client trade volume direction for 6 months using SVM.

C.4 Dominant Features Providing Best Results in the SVM for Chosen Indices

(a) GSPC

Feature	Parameters	Freq.
$\sigma(\Delta H_k, 15)/\sigma(\delta L_k, 15)$		10
Disparity	$n = 5$	8
$(M_k - \Delta H_k)/15$		7
$\text{SMA}(L_k, n) + 2 \times \sigma(L_k, 30)$	$n = 5, 15, 30$	6
$(C_k - (M_k - H_{k-2}))/n$	$n = 5, 15$	5
$(H_k - (L_k - H_{k-2}))/n$	$n = 5, 15$	5
$C_k - \text{SMA}(\Delta H_k, 30)$		5
$H_k - \text{SMA}(\text{EMA}(\Delta C_k, 30))$		5
$\text{SMA}(C_k, 15) + 2 \times \sigma(C_k, n)$	$n = 5, 15, 30$	5
$\text{SMA}(H_k - \Delta C_k, n)$	$n = 15, 30$	5
$\text{SMA}(H_k, n) + 2 \times \sigma(H_k, 15)$	$n = 5, 15, 30$	5
$H_k - \sigma(M_{k-i}, 5)$	$i = 0, 2, 5, 6$	4
$H_{k-1} - \text{SMA}(\Delta H, 30)$		4
$L_k - \text{SMA}(\text{EMA}(\Delta L, n_1), n_2)$	$n_1 = 15, 30, n_2 = 5, 15, 30$	4
$\Sigma(L_k, n)/\max(i^+, 30)$	$n = 5, 15$	4
$C_k - \text{SMA}(\text{EMA}(\Delta L, n_1), n_2)$	$n_1 = 5, 15, n_2 = 15, 30$	4
H_{k-1}		3
$H_k - \text{WMA}(\Delta H_k, n_1)$	$n_1 = 5, 15$	3
$(L_k - (H_k - C_{k-i}))/n$	$n = 5, 15, i = 1, 2$	3
$(M_k - (L_k - C_{k-2}))/n$	$n = 5, 15$	3

(b) SSMI

Feature	Parameters	Freq.
$(C_k - (M_k - L_{k-i}))/n$	$n = 5, 15, i = 1, 2, 3, 5$	6
$(C_k - \bar{\sigma}(H_k, n_1))/n_2$	$n_1 = 5, 15, 30, n_2 = 5, 15$	6
$(C_k - (L_k - C_{k-i}))/n$	$n = 5, 15, i = 0, 1, 2, 3$	5
$(C_k - \sigma(M_{k-3}, n))$	$n = 5$	5
$\sigma(\Delta C_k, n_1)/\sigma(\delta M_k, n_2)$	$n_1 = 5, 15, 30, n_2 = 5, 15, 30$	5
$(C_k - (H_k - H_{k-i}))/n$	$n = 5, 15, i = 2, 3$	4
$C_k - \sigma(C_{k-i}, 5)$	$i = 3, 4, 5$	4
$\sigma(\Delta C_k, n_1)/\sigma(\delta C_k, n_2)$	$n_1 = 5, 15, n_2 = 5, 15$	4
$\Sigma(H_h(6), 5)/C_k$		4
$(C_k - (H_k - L_{k-i}))/n$	$n = 5, 15, i = 1, 3$	3
$(C_k - (H_k - \text{SMA}(L_k, n_1)))/n_2$	$n_1 = 5, 30, n_2 = 5, 15$	3
$(C_k - (L_k - H_{k-2}))/5$		3
$(C_k - (L_k - L_{k-i}))/n$	$n = 5, 15, i = 1, 3$	3
$(C_k - (L_k - M_{k-i}))/5$	$i = 2, 3$	3
$(C_k - (M_k - L_{k-2}))/15$		3
$(C_k - (M_k - \text{SMA}(H_k, n_1)))/n_2$	$n_1 = 5, 15, n_2 = 5, 15$	3
$(C_k - \bar{\sigma}(L_k, 30))/15$		3
$(C_k - \bar{\sigma}(M_k, n_1))/n_2$	$n_1 = 5, 30, n_2 = 5, 15$	3
$(H_k - (L_k, H_{k-i}))/15$	$i = 0, 2, 3$	3
$(H_k - \bar{\sigma}(C_k, n))/5$	$n = 5, 30$	3

(c) FTSE

Feature	Parameters	Freq.
$L_k - \sigma(H_{k-i}, n)$	$n = 5, 15, i = 1, 2, 4, 5$	6
$L_k - \sigma(M_{k-i}, n)$	$n = 5, 15, i = 0, 1, 3, 5$	6
$C_k - \sigma(L_{k-i}, 5)$	$i = 3, 5$	5
$C_k - \sigma(M_{k-i}, n)$	$n = 5, 15, i = 0, 2, 3, 6$	5
$M_k - \sigma(C_{k-i}, n)$	$n = 5, 15, i = 0, 1, 3, 6$	5
$C_k - (L_k - \text{SMA}(L_k, 30))/5$		4
$C_{k-i} - \Delta H_k$	$i = 0, 1, 2$	4
$C_k - \sigma(C_{k-i}, n)$	$n = 5, 15, i = 0, 3, 5$	4
$L_k - \sigma(C_{k-i}, n)$	$n = 5, 15, i = 3, 5, 6$	4
$M_k - \text{SMA}(\text{EMA}(\Delta L, n_1), n_2)$	$n_1 = 5, 15, n_2 = 5, 15, 30$	4
$(M_k - (L_k - H_{k-i}))/n$	$n = 5, 15, i = 1, 2, 3$	4
Bias		3
$(C_k - (H_k - H_{k-i}))/n$	$n = 5, 15, i = 1, 3$	3
$(C_k - (L_k - C_{k-i}))/15$	$i = 0, 2, 3$	3
$(C_k - \bar{\sigma}(C_{k-i}, n_1))/5$	$n_1 = 15, 30$	3
Disparity		3
$C_{k-i} - \sigma(\Delta H, 5)$	$i = 2, 3$	3
$C_k - \sigma(H_{k-i}, n)$	$n = 5, 15, i = 1, 3, 6$	3
$M_k - \sigma(L_{k-i}, 5)$	$i = 3, 4, 5$	3
$(L_k - (H_k - \text{SMA}(L_k, 5)))/n$	$n = 5, 15$	3

(d) HSI

Feature	Parameters	Freq.
$C_k - \sigma(M_{k-i}, n)$	$n = 5, 15, i = 2, 6$	4
$L_k - \sigma(H_{k-i}, n)$	$n = 5, 15, i = 1, 3, 6$	4
$(L_k - (H_k - L_{k-i}))/n$	$n = 5, 15, i = 0, 2, 3$	4
ROC		4
$\text{SMA}(M_k, n_1) + 2 \times \sigma(M_k, n_2)$	$n_1 = 5, 15, n_2 = 5, 15, 30$	4
Aroon		3
$(C_k - (M_k - H_{k-1}))/n$	$n = 5, 15$	3
$(C_k - \bar{\sigma}(H_k, 5))/n$	$n = 5, 15$	3
Disparity		3
$(H_k - \bar{\sigma}(L_k, n))/15$	$n = 5, 15$	3
$C_k - \sigma(\Delta M_k, 5)$		3
$C_k - \text{SMA}(\Delta C_k, n)$	$n = 5, 30$	3
$C_k - \text{SMA}(\Delta H_k, n)$	$n = 5, 30$	3
$H_k - \sigma(\Delta L_k, n)$	$n = 5, 15, 30$	3
L_{k-6}		3
$L_k - \sigma(L_{k-i}, 5)$	$i = 1, 2, 6$	3
$L_k - \sigma(L_{k-i}, n)$	$n = 5, 15, i = 1, 5, 6$	3
$M_k - \sigma(H_{k-i}, n)$	$n = 5, 15, i = 1$	3
$M_k - \sigma(L_{k-i}, n)$	$n = 5, 15, i = 1, 2, 6$	3
$M_k - \sigma(M_{k-i}, n)$	$n = 5, 15, i = 1, 2$	3

TABLE C.4: Feature frequency for 4 selected indices when using wrapper based GA for feature selection.

Bibliography

[1] James W Hall. Adaptive selection of US stocks with neural nets. *Trading on The Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets. New York: Wiley*, pages 45–65, 1994.

[2] W. Cheng, W. Wagner, and C.H. Lin. Forecasting the 30 year US treasury bond with a system of neural networks. *Journal of Computational Intelligence in Finance*, 4:10–16, 1996.

[3] F.E.H. Tay and L. Cao. Application of support vector machines in financial time series forecasting. *Omega*, 29(4):309–317, 2001.

[4] Z. Huang, H. Chen, C.J. Hsu, W.H. Chen, and S. Wu. Credit rating analysis with support vector machines and neural networks:a market comparative study. *Decision Support Systems*, 37(4):543–558, 2004.

[5] M. Mohandes. Support vector machines for short-term electrical load forecasting. *International Journal of Energy Research*, 26(4):335–345, 2002.

[6] T. Liang and A. Noore. A novel approach for short-term load forecasting using support vector machines. *International Journal of Neural Systems*, 14(05):329–335, 2004.

[7] M. Espinoza, J. Suykens, and B. De Moor. Load forecasting using fixed-size least squares support vector machines. *Computational Intelligence and Bioinspired Systems*, pages 488–527, 2005.

[8] N. Sapankevych and R. Sankar. Time-series prediction using support vector machines: A survey. *IEEE Computational Intelligence Magazine*, 4(2):24–38, 2009.

[9] B. Krollner, B. Vanstone, and G. Finnie. Financial time series forecasting with machine learning techniques: A survey. *European Symp. ANN: Computational and*

Machine Learning, 2010.

[10] P.J. Brockwell and R.A. Davis. *Introduction to time-series and forecasting*. Springer, 2002.

[11] J. Franke, W. Härdle, and C.M. Hafner. *Statistics of financial markets: An introduction*. Springer, 2008.

[12] R.S. Tsay. *Analysis of financial time series*, volume 543. Wiley-Interscience, 2005.

[13] T.L. Lai and H. Xing. *Statistical models and methods for financial markets*. Springer, 2008.

[14] Claude Sammut and Geoffrey I Webb. *Encyclopedia of machine learning*. Springer-Verlag New York Incorporated, 2011.

[15] O. Ritthof, R. Klinkenberg, S. Fischer, and I. Mierswa. A hybrid approach to feature selection and generation using an evolutionary algorithm. In *2002 U.K. Workshop on Computational Intelligence*, pages 147–154, 2002.

[16] C.D. Kirkpatrick and J.R. Dahlquist. *Technical analysis: the complete resource for financial market technicians*. Safari Books Online. FT Press Financial Times, 2007.

[17] Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502, 2005.

[18] M. O'Neill and C. Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, 2001.

[19] Anthony Mihirana de Silva and Farzad Noorian. R package for grammatical evolution, November 2013. URL `http://sydney.edu.au/engineering/electrical/cel/gramevol`.

[20] EUNITE. World-wide competition within the eunite network, February 2001. URL `http://neuron-ai.tuke.sk/competition/`.

[21] Robert J Hyndman and George Athanasopoulos. Forecasting: principles and practice, 2012.

[22] Chris Chatfield. *The analysis of time series: an introduction*, volume 59. Chapman and Hall/CRC, 2003.

[23] SB Kotsiantis, D Kanellopoulos, and PE Pintelas. Data preprocessing for supervised leaning. *International Journal of Computer Science*, 1(2):111–117, 2006.

[24] Li-Juan Cao and Francis EH Tay. Support vector machine with adaptive parameters in financial time series forecasting. *Neural Networks, IEEE Transactions on*, 14(6):1506–1518, 2003.

[25] Kyoung-jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1):307–319, 2003.

[26] Ron Kohavi and George H John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1):273–324, 1997.

[27] Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491 – 502, April 2005.

[28] Isabelle Guyon. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[29] H. Peng, Fulmi Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1226–1238, 2005.

[30] Kenji Kira and Larry A. Rendell. A practical approach to feature selection. In *Proceedings of the 9th international workshop on Machine learning*, ML92, pages 249–256, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[31] Marko Robnik-konja and Igor Kononenko. Theoretical and empirical analysis of relieff and rrelieff. *Machine Learning*, 53:23–69, 2003.

[32] M. Dash. Feature selection via set cover. In *Knowledge and Data Engineering Exchange Workshop, 1997. Proceedings*, pages 165 –171, nov 1997.

[33] Aleks Jakulin and Ivan Bratko. Testing the significance of attribute interactions. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 52–, New York, NY, USA, 2004. ACM.

[34] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the 11th International Conference On Machine Learning*, pages 121–129. Morgan Kaufmann, 1994.

[35] Kevin J. Cherkauer and Jude W. Shavlik. Growing simpler decision trees to facilitate knowledge discovery. In *KDD'96*, pages 315–318, 1996.

[36] H. Vafaie and K. De Jong. Genetic algorithms as a tool for restructuring feature space representations. In *Proceedings of 7th International Conference on Tools*

with Artificial Intelligence, pages 8 –11, nov 1995.

[37] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[38] Jihoon Yang and Vasant Honavar. Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems and Their Applications*, 13(2):44–49, 1998.

[39] Il-Seok Oh and Jin-Seon Lee and Byung-Ro Moon. Hybrid genetic algorithms for feature selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1424 –1437, nov. 2004.

[40] Cheng-Lung Huang and Chieh-Jen Wang. A ga-based feature selection and parameters optimizationfor support vector machines. *Expert Systems with Applications*, 31(2):231 – 240, 2006.

[41] D.P. Muni, N.R. Pal, and J. Das. Genetic programming for simultaneous feature selection and classifier design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 36(1):106 –117, feb 2006.

[42] Lin, Jung-Yi and Ke, Hao-Ren and Chien, Been-Chian and Yang, Wei-Pang. Classifier design with feature selection and feature extraction using layered genetic programming. *Expert Systems with Applications*, 34(2):1384–1393, 2008.

[43] M. Sipser. Context-free grammars. In *Introduction to the Theory of Computation*, chapter 2, pages 91–122. PWS Publishing, 1997.

[44] Donald E. Knuth. Backus normal form vs. Backus Naur form. *Commun. ACM*, 7 (12):735–736, December 1964.

[45] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[46] Nello Cristianini and John Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[47] D. Eads, K. Glocer, S. Perkins, and J. Theiler. Grammar-guided feature extraction for time series classification. In *Proceedings of the 9th Annual Conference on Neural Information Processing Systems*, 2005.

[48] Shaul Markovitch and Dan Rosenstein. Feature generation using general constructor functions. In *Machine Learning*, volume 49, pages 59–98. The MIT Press, 2002.

[49] F. Pachet and P. Roy. Analytical features: a knowledge-based approach to audio feature generation. *EURASIP Journal on Audio, Speech, and Music Processing*, 2009.

[50] Hong Guo, L.B. Jack, and A.K. Nandi. Feature generation using genetic programming with application to fault classification. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 35(1):89–99, 2005.

[51] Hong Guo and A.K. Nandi. Breast cancer diagnosis using genetic programming generated feature. In *Machine Learning for Signal Processing, 2005 IEEE Workshop on*, pages 215–220, 2005.

[52] Rezarta Islamaj, Lise Getoor, and W John Wilbur. A feature generation algorithm for sequences with application to splice-site prediction. In *Knowledge Discovery in Databases: PKDD 2006*, pages 553–560. Springer, 2006.

[53] K. Krawiec and B. Bhanu. Visual learning by coevolutionary feature synthesis. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 35 (3):409–425, 2005.

[54] Otis Smart, Ioannis G Tsoulos, Dimitris Gavrilis, and George Georgoulas. Grammatical evolution for features of epileptic oscillations in clinical intracranial electroencephalograms. *Expert systems with applications*, 38(8):9991–9999, 2011.

[55] Robert McKay, NguyenXuan Hoai, PeterAlexander Whigham, Yin Shan, and Michael ONeill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11:365–396, 2010.

[56] Hong Guo, Lindsay B Jack, and Asoke K Nandi. Feature generation using genetic programming with application to fault classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(1):89–99, 2005.

[57] Shian-Chang Huang and Tung-Kuang Wu. Integrating GA-based time-scale feature extractions with SVMs for stock index forecasting. *Expert Systems with Applications*, 35(4):2080–2088, Nov. 2008.

[58] R.W. Schabacker. *Stock market theory and practice*. BC Forbes, 1930.

[59] R.D. Edwards, J. Magee, and WHC Bassetti. *Technical analysis of stock trends*. CRC, 1948.

[60] Chi-Jie Lu, Tian-Shyug Lee, and Chih-Chou Chiu. Financial time series forecasting using independent component analysis and support vector regression. *Decision*

Support Systems, 47(2):115 – 125, 2009.

[61] Myoung-Jong Kim, Sung-Hwan Min, and Ingoo Han. An evolutionary approach to the combination of multiple classifiers to predict a stock price index. *Expert Systems with Applications*, 31(2):241–247, 2006.

[62] Lean Yu, Shouyang Wang, and Kin Keung Lai. Mining stock market tendency using ga-based support vector machines. In *Lecture Notes in Computer Science 3828*, pages 336–345, 2005.

[63] Robert K Lai, Chin-Yuan Fan, Wei-Hsiu Huang, and Pei-Chann Chang. Evolving and clustering fuzzy decision tree for financial time series data forecasting. *Expert Systems with Applications*, 36(2):3761–3773, 2009.

[64] Achilleas Zapranis. Testing the random walk hypothesis with neural networks. In *Artificial Neural Networks ICANN 2006*, volume 4132 of *Lecture Notes in Computer Science*, pages 664–671. Springer Berlin Heidelberg, 2006.

[65] Cheng-Lung Huang and Cheng-Yi Tsai. A hybrid SOFM-SVR with a filter-based feature selection for stock market forecasting. *Expert Systems with Applications*, 36(2):1529–1539, 2009.

[66] A. Lendasse, E. de Bodt, V. Wertz, and M. Verleysen. Non-linear financial time series forecasting - Application to the BEL-20 stock market index. *European Journal of Economic and Social Systems*, 14(1):81–91, 2000.

[67] Ritanjali Majhi, G. Panda, Babita Majhi, and G. Sahoo. Efficient prediction of stock market indices using adaptive bacterial foraging optimization and BFO based techniques. *Expert Systems with Applications*, 36(6):10097 – 10104, 2009.

[68] H. Ince and T.B. Trafalis. Kernel principal component analysis and support vector machines for stock price prediction. In *IEEE International Joint Conference on Neural Networks*, volume 3, pages 2053–2058, 2004.

[69] Wei Shen, Xiaopen Guo, Chao Wu, and Desheng Wu. Forecasting stock indices using radial basis function neural networks optimized by artificial fish swarm algorithm. *Knowledge-Based Systems*, 24(3):378 – 385, 2011.

[70] Joarder Kamruzzaman and Ruhul A Sarker. Forecasting of currency exchange rates using ANN: A case study. In *Proceedings of the 2003 International Conference on Neural Networks and Signal Processing*, volume 1, pages 793–797, 2003.

[71] Bo-Juen Chen, Ming-Wei Chang, et al. Load forecasting using support vector machines: A study on EUNITE competition 2001. *IEEE Transactions on Power Systems*, 19(4):1821–1830, 2004.

[72] A.J.R. Reis and A.P.A. da Silva. Feature extraction via multiresolution analysis for short-term load forecasting. *IEEE Transactions on Power Systems*, 20(1):189 – 198, 2005.

[73] Jingtao Yao and Chew Lim Tan. A case study on using neural networks to perform technical forecasting of forex. *Neurocomputing*, 34(14):79 – 98, 2000.

[74] L. Ghelardoni, A. Ghio, and D. Anguita. Energy load forecasting using empirical mode decomposition and support vector regression. *Smart Grid, IEEE Transactions on*, 4(1):549–556, 2013.

[75] Eugene F Fama. The behavior of stock-market prices. *The journal of Business*, 38(1):34–105, 1965.

[76] Andrew W Lo, Harry Mamaysky, and Jiang Wang. Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. *The Journal of Finance*, 55(4):1705–1770, 2000.

[77] D Bunn and E Dillon Farmer. Comparative models for electrical load forecasting. 1985.

[78] Chang-Jun Yu and Yuan-Yuan He and Tai-Fan Quan. Frequency spectrum prediction method based on emd and svr. In *Intelligent Systems Design and Applications, 2008. ISDA '08. Eighth International Conference on*, volume 3, pages 39–44, 2008.

[79] Zbigniew R Struzik. Wavelet methods in (financial) time-series processing. *Physica A: Statistical Mechanics and its Applications*, 296(1):307–319, 2001.

[80] D Benaouda, Fionn Murtagh, J-L Starck, and Olivier Renaud. Wavelet-based nonlinear multiscale decomposition model for electricity load forecasting. *Neurocomputing*, 70(1):139–154, 2006.

[81] Teuvo Kohonen. *Self-organizing maps*, volume 30. Springer, 2001.

[82] Rob J Hyndman. Why every statistician should know about cross-validation, October 2010. URL `http://robjhyndman.com/hyndsight/crossvalidation/`.

[83] Jeffrey A. Ryan. *quantmod: Quantitative Financial Modelling Framework*, 2013. URL `http://CRAN.R-project.org/package=quantmod`. R package version 0.4-0.

[84] Jochen Knaus. *snowfall: Easier cluster computing (based on snow).*, 2010. URL `http://CRAN.R-project.org/package=snowfall`. R package version 1.84.

[85] David Meyer, Evgenia Dimitriadou, Kurt Hornik, and Andreas Weingessel. *e1071: Misc Functions of the Department of Statistics, TU Wien*, 2012. URL `http://CRAN.R-project.org/package=e1071`. R package version 1.6-1.

[86] Rob J Hyndman et al. *forecast: Forecasting functions for time series and linear models*, 2013. URL `http://CRAN.R-project.org/package=forecast`. R package version 4.06.

[87] Jawad Nagi, Keem Siah Yap, Farrukh Nagi, Sieh Kiong Tiong, and Syed Khaleel Ahmed. A computational intelligence scheme for the prediction of the daily peak load. *Applied Soft Computing*, 11(8):4773 – 4788, 2011.

[88] M. Moazzami, A. Khodabakhshian, and R. Hooshmand. A new hybrid day-ahead peak load forecasting method for Iranian national grid. *Applied Energy*, 101:489 – 501, 2013.

[89] Sirca. Enabling financial research, October 2013. URL `www.sirca.org.au/`.

[90] Saif Ahmad, Ademola Popoola, and Khurshid Ahmad. Wavelet-based multiresolution forecasting. *University of Surrey, Technical Report*, 2005.

[91] Mark J Shensa. The discrete wavelet transform: wedding the a trous and mallat algorithms. *Signal Processing, IEEE Transactions on*, 40(10):2464–2482, 1992.

Publications

Full-length Conference Papers

Anthony Mihirana de Silva, Farzad Noorian, Richard I. A. Davis and Philip H. W. Leong, A Hybrid Feature Generation and Selection Algorithm for Electricity Load Prediction using Grammatical Evolution, In 12$^{\text{th}}$ IEEE International Conference on Machine Learning and Applications (ICMLA), *page to appear*, Dec. 2013.

Journal Papers

Anthony Mihirana de Silva, Richard I. A. Davis, Syed A. Pasha and Philip H. W. Leong, Forecasting Financial Time-series with Grammar Guided Feature Generation, Computational Intelligence, *Under review*.