Grammar-guided Feature Extraction for Time Series Classification

Damian Eads, Karen Glocer, Simon Perkins, and James Theiler
Los Alamos National Laboratory, Los Alamos, NM 87544
{eads,glocer,s.perkins,jt}@lanl.gov

Abstract

We present a flexible, general-purpose technique for generating time series classifiers. These classifiers are two-stage algorithms; each consists of a set of feature extraction programs, used for transforming the time series into a vector of descriptive scalar features, and a back-end classifier (such as a support vector machine) which uses these features to predict a label. We use grammars to constrain the set of valid feature extraction programs and to provide a mechanism for incorporating domain expertise. We test our algorithm on a variety of problems, and compare its performance against conventional classifiers such as a Support Vector Machine (SVM) and Fisher Linear Discriminant (FLD).

1 Feature Extraction

Feature extraction is a process for generating numerical descriptions of data instances. The task of choosing appropriate features is notoriously domain-specific. Automated techniques for feature extraction exist but they often fall short of what a domain expert can suggest. A manual approach involves identifying physical characteristics and deriving mathematical expressions to generate numerical measures to describe them. In practice, the manual design of feature extractors is an incremental and often tedious process. Features are added or removed or in other ways tweaked until the desired performance is achieved. This can consume significant time and resources. Our aim is to enable the expert to give advice to the automated feature extractor, but for the computer to do all the grunt work. Grammars provide a mechanism for incorporating domain knowledge at whatever level of detail is available, and providing a framework within which the automated feature extractor can search for pertinent features.

1.1 Automated Feature Extraction

An automated approach to feature extraction using Genetic Programming (GP) was used by Harvey *et al.*[6] to classify pixels in multispectral images. They used GP to evolve feature extraction algorithms composed of primitive image processing operators (e.g. edge detectors, texture energy, morphological operations, etc.). The images produced by these algorithms were fed into a pixel-by-pixel linear classifier. The extracted features incorporated spatial information for each pixel to augment its spectral profile.

Previously, we developed a machine learning algorithm we called ZEUS for generating

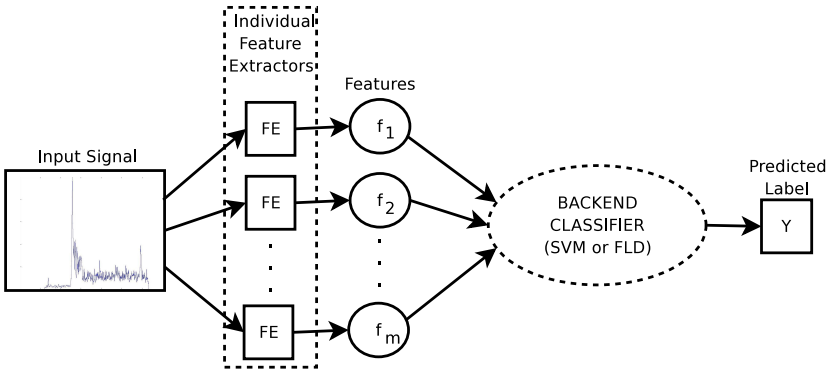

Figure 1: A ZEUS solution is a time series classifier. It consists of a set of feature extractors (the dashed box), and a back-end classifier (the dashed circle). When applied to a time series, the time series is fed into each individual feature extractor (labeled FE), and each produces at least one numerical descriptor of the input signal. These features are then fed into a back-end classifier and a predicted label (Y) results.

feature extractors for time series classification [3, 4]. We evaluated its performance on a FORTE lightning classification task. We have since extended our approach to incorporate the use of grammars to guide the extraction of a richer and more systematic set of features for classification.

As Figure 1 illustrates, a solution consists of two parts: a set of feature extractors (programs composed of primitive signal processing operators) which generate scalar features from the training data, and a back-end classifier which combines these features to predict a label. ZEUS iteratively refines its classifier until a stopping condition is met. Upon completion, ZEUS provides a classifier to categorize new data of the same form as the input data.

1.2 Human-Readable Code

The classifier (or regressor) that ZEUS produces takes the form of MATLAB code that can be integrated into a user's standalone application. The first part of this code is a set of MATLAB expressions for extracting features from the time series data. The second part takes these features and classifies them with a back-end classifier such as a linear discriminant. Human-readable algorithms provide insight into the physical and descriptive characteristics of time series, and permit an expert to more easily incorporate domain knowledge.

1.3 Dimensionality Reduction

Extracting a set of scalar features from time series also serves the purpose of reducing the dimensionality of the data. In our experiments, it was not uncommon to attain decent performance with only 5 scalar features generated from time series consisting of thousands of values. This is useful when computing a Fisher Linear Discriminant as it is significantly cheaper to compute a covariance matrix for a lower dimensional data set. Since SVMs can handle high dimensional spaces very well, the benefits of dimensionality reduction are less clear. However, the benefits of feature extraction are realized with SVMs; e.g. ZEUS can produce feature extractors which are invariant to offset shifts.

Table 1: Parameters

NAME	DEFAULT	DESCRIPTION
`init_f`	15	INITIAL NUMBER OF FEATURES
`min_f`,`max_f`	5,35	MINIMUM, MAXIMUM NUMBER OF FEATURES
`iter`	`1000`	NUMBER OF HILL-CLIMBING ITERATIONS.
`des_f`	NONE	DESIRED FITNESS (STOPPING CONDITION)
`backend`	`svm`	BACK-END CLASSIFIER
`C`	`1000`	SVM ERROR PENALTY
`rr`	0.1	RIDGE REGULARIZATION COEFFICIENT, λ
`grmfile`	NONE	FILE CONTAINING THE GRAMMAR

2 General Algorithm

The objective of ZEUS is to find a good solution for a given time series classification problem. In contrast to ordinary GP which evolves a population of solutions, ZEUS only maintains a single solution, which consists of a set of feature extractors and a linear backend.

The user provides ZEUS with choices for parameters (see Table 1) and a training set. At the start of learning, an initial set S of size `init_f` is randomly generated using the specified grammar.

For each iteration, ZEUS may add, remove, or mutate any of the feature extractors in S; each decision and feature extractor has an equal probability of being chosen. (Features are not added or removed if that would lead to fewer than `min_f` or more than `max_f` features). After a change is made, a back-end classifier (specified by `backend`) is trained with the new set of features. ZEUS is a random mutation hill-climber [5] – the new feature set is kept if it results in an increase in classification accuracy, or a negligible change in accuracy but a decrease in the time to run S. This serves two purposes: first, less complex solutions are less likely to overfit [8], and second, it provides a mechanism for escaping local minima. This loop continues for either `iters` iterations, or until a classification accuracy of `des_f` is attained.

2.1 Back-end Classifier

The back-end classifier applies a linear discriminant to the set S of extracted features. To classify an instance $\mathbf{x} \in \mathbb{R}^n$, we project it onto a hyperplane and threshold to give us a predicted label $\hat{y}(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b) \in \{-1, 1\}$. The find the weight vector $\mathbf{w}$, we employ either a Fisher Linear Discriminant (FLD) or a Support Vector Machine (SVM). The FLD is based on maximizing the ratio of between-class variance and within-class variance [2]; to guard against overfitting, we employ ridge regularization on the within-class covariance [7]. The SVM uses a linear hinge loss function with a quadratic normalization [1]; we use the implementation by Ma *et al* [11].

All input patterns are normalized to have unit magnitude ($\mathbf{x}_i^T\mathbf{x}_i = 1$) before training or applying an SVM model. The features are scaled to have zero mean and unit variance.

To train a multi-class model of ℓ classes with labels $1, 2, \ldots, \ell$, the problem is broken up into $\ell - 1$ binary classification problems: class 2 against all others, class 3 against all others, *etc.* This gives us $\ell - 1$ weight vectors $\mathbf{w}_2, \mathbf{w}_3, \ldots, \mathbf{w}_\ell$ and biases $b_2, b_3, \ldots, b_\ell$. The decision value of ith discriminant for a given pattern $\mathbf{x} \in \mathbb{R}^n$ is given by thresholding its projection,

$$d(\mathbf{x}, i) = \mathbf{w}_i^T\mathbf{x} + b_i. \tag{1}$$

To classify an instance $\mathbf{x}$, we assign the label that corresponds to the highest decision value

Table 2: Primitive operators. Some take numeric parameters that are not shown.

OPERATOR	DESCRIPTION
`auto_corr`	CROSS-CORRELATION OF THE SIGNAL WITH ITSELF
`bounds`	DEFINES WINDOW SPEC. GIVEN LOWER & UPPER BOUND
`define_window`	DEFINES WINDOW SPEC. GIVEN CENTER AND WIDTH
`grab_window`	RETURN A WINDOW OF A SIGNAL GIVEN A WINDOW SPEC.
`smooth`	SMOOTHS A SIGNAL USING A MEAN FILTER
`shift`	SHIFTS A WINDOW SPECIFICATION TO THE LEFT OR RIGHT
`minimum_index`	INDEX OF THE MINIMUM VALUE
`maximum_index`	INDEX OF THE MAXIMUM VALUE
`minimum`, `maximum`	MINIMUM, MAXIMUM VALUE OF A SIGNAL
`mean`, `median`	MEAN, MEDIAN VALUE OF A SIGNAL
`stddev`, `variance`	STANDARD DEVIATION, VARIANCE OF A SIGNAL
`skewness`, `kurtosis`	SKEWNESS, KURTOSIS OF A SIGNAL
`derivative`, `integrate`	APPROXIMATES THE DERIVATIVE, INTEGRAL OF A SIGNAL
`cross`	INDEX WHERE THE SIGNAL CROSSES A VALUE (E.G. MEAN)
`se_line`	RETURNS A LINE STRUCTURING ELEMENT
`erode`, `dilate`, `open`, `close`	STANDARD MORPHOLOGICAL OPERATORS

if it is positive, and 1 if all tests fail; thus, we have the predicted label

$$\hat{y}(\mathbf{x}) = \begin{cases} \arg\max_{i=2}^{\ell} d(\mathbf{x}, i) & \max_{i=2}^{\ell} d(\mathbf{x}, i) > 0 \\ 1 & \text{otherwise} \end{cases}. \tag{2}$$

3 Grammars

What operators help classify time series? The answer to this question depends on the problem at hand. A human who is familiar with the problem could help in this regard.

If it is unclear what operators should be excluded from the pool at the start of learning, one approach is the shotgun approach; *i.e.*, simply provide ZEUS with a rich and diverse set of operators. Unfortunately, this poses another problem: the more operators that are available to ZEUS, the larger the search space, and the greater potential for garbage solutions. The advantage of using grammars is that they enable one to constrain the space and incorporate domain knowledge into the automated search for a feature extractor.

The use of grammars to constrain Genetic Programming, a field that has since adopted the name Grammatical Evolution (GE), was considered by Ryan *et al* [17]. They employed a context-free grammar to define the set of valid programs for which the genetic program could generate. Before the introduction of grammars, GP was largely restricted to languages, such as LISP, whose syntax could not be violated by applying GP operators to its programs. GE has been successfully used on a number of problems [15, 18, 19].

In previous work, a Strongly-Typed-GP system proposed by Montana *et al.*[13] was used to ensure the validity of programs [3, 4]. This posed two problems. First, it was difficult to visualize the possible structures of candidate algorithms because they were defined by a flat set of operator prototypes. A grammar addresses this problem by providing hierarchical syntax for which to express algorithm structures. Second, the data type matching approach did not permit one to constrain the set of candidate solutions beyond that achieved by ensuring type validity. For example, programs were produced which would grab a window centered at a peak in a signal and then grab another window within that window. Our grammar prevents this from happening. In this way, grammars enable the user to remove program parts from the set of all possible program parts which are valid but are not likely

to be useful.

3.1 Motivation

Grammars can help speed up the learning process. This is achieved by restricting the solution space to programs that obey a grammar. Similar to how the English grammar rules define how a valid sentence is formed so that a comprehensible idea, however unusual, is conveyed, a ZEUS grammar defines the general flow of a well-formed feature extraction program.

Grammars express a language, that is, a set of strings. Grammars are most commonly used for parsing; however, in our study, we use them exclusively to generate programs. More specifically, grammars in ZEUS generate strings that can be evaluated in a MATLAB environment. The format of our grammar files is very similar to Backus Naur Form (BNF) notation commonly used in computation theory. In our grammars, the right-hand side (RHS) of a production rule is a parenthetical expression; the parenthesis govern the structure of the tree to be generated, or more simply, the placement of parenthesis in a MATLAB expression.

References to a primitive operator (*i.e.*, a MATLAB function) begin with a lowercase letter and references to productions are capitalized. An expression `*(a:s:b)`, when expanded, where `a` and `b` are numbers, generates a random number r between a and b (exclusive) such that $r = a + sk$. Note that `*(a:b)` is equivalent to `*(a:1:b)`. To expand an expression of the form `*{a1, a2, ..., an}`, where `ai` is a fixed constant, one of the constants is chosen at random.

```
TimeProc(X)       ::= TimeProc(Fourier(X)) | autocorr(X) | ...;
RandNum           ::= 0:10^-12:1;
RandFFTPoint      ::= *{64, 128, 256, 512};
Fourier(X)        ::= ifft(FourierProc(fft(X, PT = RandFFTPoint()), PT));
FourierProc(X, Y) ::= lowpass(X, *(1:Y/2)) | highpass(X, *(1:Y/2))
                  |   bandpass(X, *(1:Y/2), *(1:Y/2));
```

Figure 2: An example of a grammar for generating part of an algorithm for filtering a signal. Note that in practice this grammar would be part of a much larger grammar. The `Fourier` production is context-sensitive; *i.e.*, a production is situated between the terminals `fft` and `ifft`. The grammar also stipulates that time domain operators are to be used once an Inverse Fast Fourier Transform is performed.

3.1.1 Example 1: Context-Sensitivity

If we wish produce a program which can apply a Fast Fourier Transform (FFT) to a signal, we can use a grammar to ensure that only operators that work in the Fourier domain are applied to the result of an FFT. The grammar in Figure 2 does this. The `Fourier` production is context-sensitive, *i.e.*, `FourierProc`, is situated between the terminals `ifft` and `fft`. This means that the part of the program that is expanded between these two terminals is restricted to the rules that apply in that context. More specifically, only the band-math operators `lowpass`, `highpass`, and `bandpass` can be used after `fft` and before `ifft` are performed.

3.1.2 Example 2: Data Fusion and Segmentation

The grammar shown in Figure 3 is used to generate a feature extractor. When extracting features from signals, often a statistic is computed over only part of a signal; this enables us to compute *local* features. Choosing this part is called segmentation. The grammar

```
Feature            ::= mean(GrabWindow(GetData('Signal', *1:@num_signals)))
                       :  {if=@signal_available, prob=0.8}
                   |   mean_image(GetData('Image', *1:@num_images))
                       :  {if=@image_available};
GrabWindow(X)      ::= window(MeaningfulIndex(X), RandNum())
                   |   shift(window(MeaningfulIndex(X), MeaningfulIndex(X),
                                RandNum()), *(-@size/3):(@size/3));
MeaningfulIndex(X) ::= @user_index(*1:size(@user_index))
                       :  {if=defined(@user_index)}
                   |   maximum_peak(X) | ...;
```

Figure 3: A simple grammar for producing a feature extractor. The `if` statement shown after a production rule is use to impose a condition. If false, the rule cannot be chosen for expansion. The `@` symbol indicates a global variable. These variables are set when data is loaded into the ZEUS system. The `GrabWindow` production extracts part of a signal based on a location. This location is provided by `MeaningfulIndex`. This index can be one provided by the user or one based on a feature of the data, e.g. `maximum_peak`.

stipulates that when segmenting, a window is grabbed at some location. This location could be provided by the user, or it could be based on some feature of the data. The window can be centered at that location, or the bounds of the window could be specified by two indices. Optionally, once the window is defined, it could be shifted to the left or right. This way, we can capture the part of the signal that occurs before some location.

It is not uncommon for remote sensing devices to collect data in a variety of formats (e.g. signals, images, trigger locations, trigger times, images, geographic locations). A tool that performs data fusion knows how to parse through these data in different formats, and combine them in some sensible, legal way so that all available data are used, and learning is not restricted to one domain. We can use a grammar to achieve this kind of data fusion.

Figure 3 shows a simple example of data fusion in action. It can extract features from images and signals. Note the use of `if` after several production rules. The production rule can only be expanded if the `if` condition is true. In this case, we can only extract a feature from an image or a signal if the user has provided such data. If an image is not available, the second production rule of `Feature` cannot be chosen for expansion.

The user can provide one or more time indices for each signal by setting a global variable in MATLAB called `user_index`. If this is done, the first production rule `MeaningfulIndex` can be expanded. If expanded, an index is chosen at random. If the user has provided two indices for each signal, e.g. the location of the smallest and largest peak, only one of them will be chosen in the expansion.

4 Experiments

We used seven data sets altogether. Three of them (GUN, TRACE, and SYNTHETIC CONTROL) were obtained from the UCR Time Series Data Mining Archive [9]. Three data sets (FORTE-2, FORTE-6, and FORTE-7) are from a satellite used to measure the RF radiation emitted from lightning strikes [14]. These time series are labeled according to the kind of lightning: FORTE-2 is labeled only according to intra-cloud versus cloud-to-ground; FORTE-6 is the same data with more specific labels; and FORTE-7 includes some data from an "outlier" class. Finally, the EDOTX-SYN data contains simulated lightning pulses from a ground sensor. The last four data sets and grammar file are available at the author's website: `http://nis-www.lanl.gov/~eads/nips05-data`. For consistency, the same generic grammar file was used for all experiments with ZEUS.

For each data set, we performed five experiments. The first two involve coupling ZEUS with either FLD or a linear SVM as the back-end classifier. We only used a linear classifier

Table 3: Experimental setup for each data set. There are ℓ classes and n time series records in each data set. Each of these records consists of d data points per channel, and k channels.

EXPERIMENTAL SETUP					
DATA SET	ℓ	n	d	k	TESTING/VALIDATION
GUN	2	200	150	1	10-FOLD C.V.
TRACE	4	200	200	1	10-FOLD C.V.
SYN. CONTROL	6	600	60	1	10-FOLD C.V.
FORTE -2	2	121	3811	1	10-FOLD C.V.
FORTE -6	6	121	3811	1	10-FOLD C.V.
FORTE -7	7	143	3811	1	10-FOLD C.V.
EDOTX-SYN	2	2000	2000	1	18000

Table 4: Results. The first two major columns report the error obtained when coupling a conventional classifier is and is not coupled with ZEUS. The best external baseline is reported if available. For the DTW baselines, leave-one-out cross-validation was used.

	ERROR WITH ZEUS		ERROR WITHOUT ZEUS			BASELINE	
DATA SET	FLD	LSVM	FLD	LSVM	SVM+RBF	RESULT	METHOD
GUN	1.5	2.5	9.5	5.5	5.5	1	DTW [16]
TRACE	0	0.5	28.5	8	17.5	0	DTW [16]
SYN. CONTROL	2	2.17	25.17	4	3.3	0.33	DTW [16]
FORTE -2	13.22	20.66	40.5	30.58	28.93		
FORTE -6	21.49	28.1	50.41	40.50	38.02		
FORTE -7	25.87	28.67	54.54	34.97	37.06	21.52	MSNFE [10]
						29.57	KFFE [10]
EDOTX-SYN	4.43	3.67	32.39	3.98	4.69		

backend for ZEUS because of the value of its lack of complexity as well as its decent performance on the data sets we tried. A linear SVM, SVM with a Radial Basis Function (RBF) kernel, and a FLD are trained on the raw data for the last three experiments. For each experiment, a regularization coefficient is adjusted through 10-fold cross-validation. When FLD is used, the ridge regularization coefficient λ takes on the values defined by the set $\{10^{-i} | i \in \{5, 4, 3, 2, 1\}\}$. For an SVM, the set of error penalties tried is defined by the set $\{10^{i} | i \in \{-1, 0, 1, 2, 3\}\}$. The lowest cross-validation error is reported for each experiment and data set except for EDOTX-SYN. In this case, we retrain with the regularization parameter that leads to the lowest validation error, and apply it to a test set consisting of $18,000$ instances and then report the test error. When training an SVM with an RBF kernel, we set gamma to the standard deviation of the entire training set.

The YALE system uses genetic programming to generate feature extraction programs for time series classification [12]. The operator pool by YALE includes operators which, perform transformations into, and operate in, phase space. In future work, we would like to incorporate these operators into ZEUS and use grammars to define how they are used.

5 Conclusion

Grammars provide an intuitive and hierarchical framework for incorporating domain knowledge into an automated search for feature extractors. They also can explore the use of a larger set of operators while mitigating the cost of an increase in search space size. We tested our algorithm against conventional classifiers on seven data sets to demonstrate its performance. Our algorithm outperformed the conventional classifiers trained on the

raw data. The use of grammar-guided hill climbing for feature extraction has potentially two powerful benefits: it is generic to a wide variety of problem domains (e.g. computer vision, robotics, time series forecasting), and second, it enables the user to input domain knowledge, if available, to assist the learning algorithm in its search. Efforts are underway to show these effects more clearly by comparing results obtained with different grammars.

References

[1] C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[2] R. O. Duda, P. E. Hart, and D. C. Stork. *Pattern Classification*. John Wiley & Sons, New York, 2001.

[3] D. Eads, D. Hill, S. Davis, S. Perkins, J. Ma, R. Porter, and J. Theiler. Genetic algorithms and support vector machines for time series classification. *Proceedings of the SPIE*, 4787:74–85, 2002.

[4] D. R. Eads, S. J. Williams, J. Theiler, R. Porter, N. R. Harvey, S. J. Perkins, S. Brumby, and N. A. David. A multimodal approach to feature extraction for image and signal learning problems. *Proceedings of the SPIE*, 5200:79–90, 2003.

[5] S. Forrest and M. Mitchell. Relative building-block fitness and the building-block hypothesis. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 109–126. Morgan Kaufmann, San Mateo, CA, 1993.

[6] N. R. Harvey, J. Theiler, S. P. Brumby, S. Perkins, J. J. Szymanski, J. J. Bloch, R. B. Porter, M. Galassi, and A. C. Young. Comparison of genie and conventional supervised classifiers for multispectral image feature extraction. *IEEE Transactions on Geoscience and Remote Sensing*, 40:392–404, 2002.

[7] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, New York, 2002.

[8] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. *International Conference on Machine Learning*, 11:121–129, 1994.

[9] E. Keogh and T. Folias. The UCR time series data mining archive, 2002.

[10] J. Ma, J. Theiler, and S. Perkins. Two realizations of a general feature extraction framework. *Pattern Recognition*, 37:875–887, 2004.

[11] J. Ma, Y. Zhao, and S. Ahalt. The Ohio State University Support Vector Machine Package, 2002. `http://www.ece.osu.edu/~maj/osu_svm/`.

[12] I. Mierswa and M. Katharina. Automatic feature extraction for classifying audio data. *Machine Learning Journal*, 58:127–149, 2005.

[13] D. J. Montana. Strongly typed genetic programming. Technical Report #7866, 10 Moulton Street, Cambridge, MA 02138, USA, 7 1993.

[14] K. Moore, P. Blain, S. Briles, and R. Jones. Classification of RF transients in space using digital signal processing and neural network techniques. *Proceedings of the SPIE*, 2492:995–1006, 1997.

[15] M. O'Neill and C. Ryan. *Grammatical Evolution*. Kluwer Academic Publishers, Boston, 2003.

[16] C. A. Ratanamahatana and E. Keogh. Everything you know about dynamic time warping is wrong. In *10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining Workshop on Temporal Data Mining*, pages 50–60, 2004.

[17] C. Ryan, J. J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. *Proceedings of EuroGP 98*, pages 83–96, 1998.

[18] A. Tsakonas, V. Aggelis, I. Karkazis, and G. Dounias. An evolutionary system for neural logic networks using genetic programming and indirect encoding. *Journal of Applied Logic*, 2:349–379, 2004.

[19] M. Wong and K. Leung. *Data Mining Using Grammar Based Genetic Programming and Applications*. Kluwer Academic Publishers, Boston, 2000.