UNIVERSITY OF CALIFORNIA, SAN DIEGO

Automated Feature Design for Time Series Classification by Genetic Programming

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Structural Engineering

by

Dustin Yewell Harvey

Committee in charge:

Professor Michael Todd, Chair
Professor Sanjoy Dasgupta
Professor Charles Farrar
Professor Tara Javidi
Professor Hyonny Kim

2014

The Dissertation of Dustin Yewell Harvey is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____
Chair

University of California, San Diego

2014

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

1-NN	1-Nearest Neighbor
AR	AutoRegressive
DSD	Dynamics, Stationarity, and Distribution validation problem
DTW	Dynamic Time Warping
FFT	Fast Fourier Transform
FN	False Negative
FP	False Positive
GP	Genetic Programming
IGDT	Info-Gap Decision Theory
MLP	Multi-Layer Perceptron
PDF	Probability Density Function
ROC	Receiver Operating Characteristic
RMS	Root Mean Square
SHM	Structural Health Monitoring
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
TSC	Time Series Classification
TSF	Time Series Forest
UCR	University of California, Riverside
WGN	White, Gaussian Noise

LIST OF SYMBOLS

x_i	Time series sample
i	Index reserved for time series
F	Fitness
E	Classification error
Q	Quadratic loss
δ	Fitness component weighting factor
N	Number of instances in training data
N_{min}	Number of instances in smallest class in training data
N_{max}	Number of instances in largest class in training data
k	Number of folds in cross-validation procedure
R	Number of repetitions in cross-validation procedure
μ	Population size
ρ	Number of individuals in parent pool
λ	Number of individuals in offspring
p	Function parameter
σ	Standard deviation
α	Uncertainty level parameter
r	Radius of uncertainty model

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to take this opportunity to formally thank the many people who have guided and supported me in my academic, professional, and personal life. First and foremost, I would like to acknowledge my professor and advisor, Dr. Michael Todd. I could not have asked for a more beneficial or enjoyable four years in graduate school and attribute the majority of credit for the experience to the engaging environment fostered by Dr. Todd.

I would further like to acknowledge Dr. Todd and the other members of my dissertation committee: Dr. Farrar, Dr. Kim, Dr. Dasgupta, and Dr. Javidi. Your time and valuable input both in preparation of this manuscript and throughout my research are greatly appreciated.

I would like to acknowledge all of the students I have interacted with through the UCSD SHM research group. In particular, thanks goes to Scott Ouellette, Colin Haynes, Eric Flynn, and Stuart Taylor for always taking the time to provide a sounding board for my ideas. Your support and friendship are greatly appreciated.

Special thanks to my many mentors over the past decade that have led me on a wonderful journey to this point. Dr. Patricia Brackin and Dr. Phillip Cornwell from Rose-Hulman Institute of Technology and Dr. Charles Farrar from the Engineering Institute at Los Alamos National Laboratory, each of you has counseled me through a critical transition in my life, and I am very grateful for your guidance. Dr. Keith Worden from University of Sheffield also deserves special recognition for many enlightening research discussions and his contributions as my co-author.

I would like to acknowledge my wonderful family and friends without whom I would never have reached this point. To my parents, Dale and Janet, I can never hope to repay your unwavering support nor express the depth and breadth of my gratitude. To my sister, Jessica, never underestimate the worth of the solace you have provided in my most difficult moments.

Last but not least, to my wife, Jennie, thank you for your patience, support, and sacrifice over the last three years. I am so excited to see what the future brings and share it with you.

Portions of Chapters 1, 2, 3, and 6 have been published in IEEE Transactions on Evolutionary Computation, Dustin Harvey and Michael Todd, 2014. The title of this paper is "Automated Feature Design for Numeric Sequence Classification by Genetic Programming". The dissertation author was the primary investigator and author of this paper.

Portions of Chapters 1, 2, 5, and 6 have been submitted for publication in Data Mining and Knowledge Discovery, Dustin Harvey and Michael Todd, 2014. The title of this paper is "Automated Feature Design for Time Series Classification". The dissertation author was the primary investigator and author of this paper.

A portion of Chapter 4 has been published in Smart Materials and Structures, Dustin Harvey and Michael Todd, 2014. The title of this paper is "Structural Health Monitoring Feature Design by Genetic Programming". The dissertation author was the primary investigator and author of this paper.

A portion of Chapter 7 has been published in Proceedings of SPIE Smart Structures/NDE conference, Dustin Harvey, Keith Worden, and Michael Todd, 2014. The title of this paper is "Robust Evaluation of Time Series Classification Algorithms for Structural Health Monitoring". The dissertation author was the primary investigator and author of this paper.

This work was supported by the Department of Defense (DoD) through the National Defense Science and Engineering Graduate Fellowship (NDSEG) Program.

VITA

2009 B.S. in Mechanical Engineering, Rose-Hulman Institute of Technology

2012 M.S. in Structural Engineering, University of California, San Diego

2014 Ph.D. in Structural Engineering, University of California, San Diego

PUBLICATIONS

*Work that has been incorporated into this dissertation

Peer Reviewed Journal Articles

*D.Y. Harvey, M.D. Todd, Automated feature design for time series classification, Data Mining and Knowledge Discovery, in review. 2014.

*D.Y. Harvey, M.D. Todd, Structural health monitoring feature design by genetic programming, Smart Materials and Structures, 23, 9. 2014, 095002.

*D.Y. Harvey, M.D. Todd, Automated feature design for numeric sequence classification by genetic programming, Evolutionary Computation, IEEE Transaction on, pre-print. 2014, doi:10.1109/TEVC.2014.2341451.

Selected Conference Proceedings

*D.Y. Harvey, K. Worden, M.D. Todd, Robust evaluation of time series classification algorithms for structural health monitoring, Proc. SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring, San Diego, California. 2014, 90640K.

D.Y. Harvey, M.D. Todd, Automated near-optimal feature extraction using genetic programming with application to structural health monitoring problems, Proc. Proceedings of International Workshop on Structural Health Monitoring, Palo Alto, California. 2013.

D.Y. Harvey, M.D. Todd, Automated extraction of damage features through genetic programming, Proc. SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring, San Diego, California. 2013, 86950J.

D.Y. Harvey, M.D. Todd, Automated selection of damage detection features by genetic programming, Proc. International Modal Analysis Conference XXXI, Garden Grove, California. 2013.

M. Todd, D. Harvey, D. Gregg, B. Fladung, P. Blelloch, K. Napolitano, "Structural system Testing and model correlation": an industry-university collaborative course in structural dynamics, Proc. International Modal Analysis Conference XXXI, Garden Grove, California. 2013.

D.Y. Harvey, M.D. Todd, Cointegration as a data normalization tool for structural health monitoring applications, Proc. SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring, San Diego, California. 2012, 834810.

D. Harvey, M. Todd, Symbolic dynamics-based structural health monitoring, Proc. International Modal Analysis Conference XXX, Jacksonville, Florida. 2012.

E.B. Flynn, S. Kpotufe, D. Harvey, E. Figueiredo, S. Taylor, D. Dondi, et al., SHMTools: a new embeddable software package for SHM applications, Proc. SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring, San Diego, California. 2010, 764717.

FIELDS OF STUDY

Major Field: Structural Engineering

Studies in Structural Dynamics
Professor Michael Todd

Studies in Advanced Structural Behavior
Professor Charles Farrar

Studies in Digital Signal Processing
Professor William Hodgkiss

Studies in Statistical Pattern Recognition

ABSTRACT OF THE DISSERTATION

Automated Feature Design for Time Series Classification by Genetic Programming

by

Dustin Yewell Harvey

Doctor of Philosophy in Structural Engineering

University of California, San Diego, 2014

Professor Michael Todd, Chair

Time series classification (TSC) methods discover and exploit patterns in time series and other one-dimensional signals. Although many accurate, robust classifiers exist for multivariate feature sets, general approaches are needed to extend machine learning techniques to make use of signal inputs. Numerous applications of TSC can be found in structural engineering, especially in the areas of structural health monitoring and non-destructive evaluation. Additionally, the fields of process control, medicine, data analytics, econometrics, image and facial recognition, and robotics include TSC problems.

This dissertation details, demonstrates, and evaluates Autofead, a novel approach to automated feature design for TSC. In Autofead, a genetic programming variant evolves a

population of candidate solutions to optimize performance for the TSC or time series regression task based on training data. Solutions consist of features built from a library of mathematical and digital signal processing functions. Numerical optimization methods, included through a hybrid search approach, ensure that the fitness of candidate feature algorithms is measured using optimal parameter values. Experimental validation and evaluation of the method is carried out on a wide range of synthetic, laboratory, and real-world data sets with direct comparison to conventional solutions and state-of-the-art TSC methods. Autofead is shown to be competitively accurate as well as producing highly interpretable solutions that are desirable for data mining and knowledge discovery tasks. Computational cost of the search is relatively high in the learning stage to design solutions; however, the computational expense for classifying new time series is very low making Autofead solutions suitable for embedded and real-time systems.

Autofead represents a powerful, general tool for TSC and time series data mining researchers as well as industry practitioners. Potential applications are numerous including the monitoring of electrocardiogram signals for indications of heart failure, network traffic analysis for intrusion detection systems, vibration measurement for bearing condition determination in rotating machinery, and credit card activity for fraud detection. In addition to the development of the overall method, this dissertation provides contributions in the areas of evolutionary computation, numerical optimization, digital signal processing, and uncertainty analysis for evaluating solution robustness.

Chapter 1

Introduction

1.1. Time Series in Structural Engineering

Time series measurements are employed throughout structural engineering to characterize and quantify structure behavior. Structural response data typically consists of time histories of kinetic and kinematic quantities under various loading and operational conditions. Such measurements are conventionally used for fitting, updating, and validating structural models, often through conversion to the frequency domain. Design certification, risk assessment, and other tasks rely on accurate models; however, highly complex structures may not admit reliable physical modeling in which case direct methods to utilize time series data are required. In the areas of non-destructive evaluation and structural health monitoring, data-based approaches are increasingly common for determining local thickness, joint integrity, bond condition, and other structural health information. Many of these problems can be addressed using techniques from the more general areas of machine learning and time series classification.

1.2. Machine Learning

The field of machine learning encompasses methods and techniques for discovering, describing, and utilizing patterns in data. Machine learning is further divided based on input and output characteristics into classification, clustering, and regression. Regression methods produce numeric output, while classification and clustering deal with nominal or categorical output, i.e. classes. Furthermore, classification methods are applied to supervised learning problems where the input data includes corresponding class labels. When class information is unavailable, the problem is categorized as unsupervised learning, as in clustering.

Machine learning methods find application in a variety of fields including search engines, marketing and advertising, process control, econometrics, and medical diagnosis. As an example, a medical researcher may apply a variety of techniques to patient records with confirmed diagnoses of a specific disease in an effort to develop new diagnostic tools. The input data consists of categorical and numeric attributes, or features, of each patient such as age, weight, and results of various diagnostic tests. This example is representative of a classification problem since known examples, or instances, are available from each class of interest. The objectives of such studies are to develop or select a method that generalizes with maximum accuracy to unseen instances.

Together with data mining, the two fields constitute the more general area of pattern recognition. Whereas the primary objective in machine learning is accuracy, data mining employs many of the same tools in an effort to discover knowledge about the data-generating system. Jain, Duin, and Mao provide a summary of pattern recognition methods, issues, and examples [1]. Additional background can be found in books by Bishop [2] and Hastie, Tibshirani, and Friedman [3].

1.2.1. Classifiers and Learners

Machine learning for multivariate scalar features is a near-mature field in pattern recognition. Learning algorithms, or learners, build and fit classification models using training data. A classifier describes the assumptions, structure, and representation of a specific class of models that may be created from a variety of different learning algorithms. Many learners and associated classifiers are available that make certain assumptions about the classes leading to tradeoffs in accuracy, complexity, and computational cost. The assumptions are a form of bias in the classifier selection process. Low-bias methods make minimal assumptions allowing for a more flexible model, but requiring more instances for the training process to converge. Conversely, high-bias methods are more robust to variance in the training data and therefore often generalize better to new data provided the assumptions made are correct or reasonable for the problem at hand. For example, the common assumptions of feature independence and Gaussian feature distributions together create the popular Gaussian Naïve-Bayes classifier. This high-bias model works well on a wide variety of data sets and has very low computational costs, but it may perform poorly with features that are highly-correlated or far from Gaussian. Typically, designers select an appropriate algorithm after basic statistical analysis of the feature set through a trial-and-error process.

Other types of classifiers include instance-based methods, decision trees, ensemble methods, neural networks, support vector machines, and various other statistical models such as linear discriminant analysis. Instance-based, or lazy learning, classifiers have no associated learning stage; instead, a distance measure is computed between new data and all training instances then the new instance is labeled with the class containing the most similar (minimum distance) training instance. Decision trees use a hierarchical structure where each step in the labeling process involves thresholding of a single feature. Ensemble methods, or meta-

algorithms, employ multiple realizations of a classifier or different classifiers in conjunction with a voting scheme to combine results from the set of classifiers. Boosting and bagging are examples of ensemble schemes [3].

1.2.2. Evaluating Accuracy

A common issue in machine learning arises due to the finite size of data sets for training and evaluating solutions. Most learners require large instance counts to reach convergence of the classifier model. Additionally, accurate error rate estimation requires large instance counts, and the two steps of training and testing classifiers must be performed on mutually exclusive data sets to avoid overly optimistic in-sample error estimates. While accurate classifiers are always desirable, the ability to truthfully estimate error is equally important. Poor error estimates on training data in the solution design stage lead to loss of performance on test data through non-optimal selection of classifiers and parameters.

The most basic percentage split, or holdout, method separates available data into a training set and testing set to perform out-of-sample error estimation. More advanced procedures such as k-fold cross-validation, leave-one-out, and bootstrapping improve on the holdout method by generating multiple pairs of testing and training splits to better utilize finite data. Even with such sampling methods, estimation of generalization error from limited data is still a challenging task. Overfitting of training data is a common issue wherein a learner is able to effectively model the noise within a small training data set leading to solutions that do not generalize to new instances. The interested reader is directed to Japkowicz and Shah for more detailed discussion of the evaluation process and related issues [4].

1.2.3. Feature Sets

The input data for standard machine learning tasks is represented as feature, or attribute, sets. Each instance contains a feature vector that can include nominal, ordinal, and numeric features. The terms feature selection, construction, transformation, and extraction all refer to an intermediate step of creating an improved, often reduced size, feature set from the original data. Feature selection simply identifies a subset of the original features that minimizes information loss. Feature construction, transformation, and extraction attempt to recombine the original features linearly or non-linearly into new features. Principal component analysis, factor analysis, and neural networks are popular techniques in the area of feature construction [1]. The optimal, minimal-basis feature set retains all the information present in the original data that is relevant to the classification problem within an orthogonal feature space.

1.2.4. Higher Dimension Inputs

Increasingly, researchers and analysts find themselves inundated with high-dimensional data for use in classification tasks. Here, dimensionality refers to the number of independent variables required to represent sampled data. Time series and spectra are one-dimensional signals while images require two spatial dimensions. As such, feature vectors are considered zero-dimensional but multivariate in the case of multiple features. Example applications involving one-dimensional signals include the monitoring of electrocardiogram signals for indications of heart failure, network traffic analysis for intrusion detection systems, vibration measurement for bearing condition determination in rotating machinery, and credit card activity for fraud detection. In such applications, domain knowledge primarily drives the selection and design of pre-processing steps, features, models, and learning algorithms that constitute a pattern recognition system for high-dimensional data. Ever-decreasing costs of sensing, data

acquisition, and data storage hardware have led to many applications where such data is abundantly available, but domain knowledge is very limited.

Feature-based methods for high-dimensional data require an additional step of feature design. Feature design is a challenging task, especially in the absence of sufficient domain knowledge, since one-dimensional signals have no explicit features and one cannot enumerate all possible features [5]. Here, we differentiate between feature design and methods for feature construction, feature extraction, or feature transformation based on the presence of order-dependent operations. To clarify, an automated, evolutionary feature construction process with a one-dimensional signal input is described by Guo, Jack, and Nandi [6]. In this work, the input contains 33 ordered Fast Fourier Transform (FFT) bins that may contain relevant, dynamic information within their structure. However, no order-based operations are used to construct new features; therefore, each FFT bin is treated independently making the system more indicative of standard feature construction than feature design. Other high-level approaches exist that directly classify high-dimensional data without a separate step of measuring features, for example instance-based approaches that compute distance measures on the original input signals.

As an example, consider a critical machine in a manufacturing process that undergoes periodic maintenance but still experiences occasional bearing failures causing expensive downtime. If maintenance were performed as needed, downtime would be minimized compared to a time- or usage-based maintenance schedule or run-to-failure approach. A damage monitoring system based on pattern recognition would allow for such a condition-based maintenance approach. Traditionally, an expert in condition monitoring for rotating machinery would be consulted to design a damage monitoring system based on available literature and past experience with similar machines. An automated machine learning approach would allow an

unfamiliar engineer or technician to design a similar system from example data alone. First, the user would instrument the machine with a vibration sensor and periodically record time series. For supervised learning, records are then labeled as either from the healthy state or just prior to known failures. Finally, an automated feature design approach is applied to design optimal pre-processing steps and features for the specific machine and failure mode from the time series data set. The latter approach represents significant potential savings in development time and expense, improved performance of the damage monitoring system, and relaxation of the need for an expert.

1.3. Time Series Classification

Time series classification (TSC) methods discover and exploit patterns in time series and other numeric sequence data for a variety of applications. The field has developed rapidly over the last decade with dozens of new techniques and a wide variety of problems. With the ever-increasing deployment of sensors in industrial, defense, and civilian arenas, powerful automated tools are needed to maximize the utility of these rich data sets. TSC is an extension of standard classification of attributes and features to handle one-dimensional signals such as time series, spectra, and one-dimensional representations of shape and image data. Applications of TSC can be found in the areas of process control, medicine, structural health monitoring, data analytics, econometrics, and robotics as well as image, shape, and facial recognition from one-dimensional representations. In a review on the related topic of time series data mining, Fu notes that “the fundamental problem is how to represent the time series data,” either through a transformation to an improved data space, computation of a feature vector, or in another form [7]. Xing, Pei, and Keogh provide an overview of the general field of sequence classification for both numeric and symbolic data [5]. In comparison to standard classification tasks, Xing et al.

cite the difficulty of sequence data as a lack of explicit features, high-dimensionality, and additional complexity of building interpretable classifiers. Further review of time series representations, distance and similarity measures, indexing, classification, and clustering is found in the work of Esling and Agon [8].

Xing et al. categorize the various approaches to sequence classification as either model-based, distance-based, or feature-based [5]. Model-based approaches consist of building generative models then applying pattern recognition techniques to the model parameters or predictive errors. These approaches are found infrequently in the literature and are not appropriate for problems where the relevant information is not well represented in the global sequence behavior captured by a generative model. Distance- (or similarity-) based approaches discussed in section 1.3.3 typically employ an instance-based classifier such as nearest neighbors directly to the time series data in conjunction with an appropriate distance measure. Feature-based methods that apply standard classifiers to feature vectors computed from the time series are described in section 1.3.4.

1.3.1. Applications

Potential applications for TSC methods are numerous including the following:

- Diagnosing indications of heart failure from electrocardiogram signals
- Monitoring automotive sensors for early failure detection
- Detecting fraud from credit card activity
- Analyzing network traffic for intrusion detection systems
- Developing financial models based on econometric data
- Identifying manufacturing defects in industrial processes

- Determining bearing condition in rotating machinery from vibration measurement

1.3.2. Objectives

The goals of TSC research include improved accuracy, solution interpretability, and reduced computational expense. Accuracy is estimated as the expected generalization error when applying a given solution to unseen data. Standard evaluation methods as discussed in section 1.2.2 are directly applicable to TSC. The interpretability of solutions is particularly important for data mining applications and knowledge discovery. Easily understood algorithms and models provide the possibility to develop a physical interpretation of solutions leading to improved understanding of the data-generating system. Lastly, minimal computational expense is critical for many applications such as embedded systems. The computational cost of a method should be considered both in the learning or solution design stage as well as in the application stage where new instances are classified. For a specific application, the computational expense in one or both stages may be relevant to the method selection process.

1.3.3. Distance-Based Methods

Distance-based TSC employs traditional instance-based classifiers with a variety of specialized time series distance or similarity measures. The 1-nearest neighbor classifier with Euclidean distance between time series is the simplest approach. A variety of specialized distance measures have been proposed to correct for global and local shifts and scaling of the samples in the time domain. Wang et al. review and experimentally compare a variety of distance measures both as representation methods and for their accuracy as classifiers [9]. The study showed the very popular dynamic-time warping (DTW) measure to be at least as accurate as other measures. Dynamic-time warping corrects local time shifts by finding the optimal

many-to-many monotonic mapping of samples between two time series and has long been employed as the benchmark method for TSC [10].

More recent developments have proposed alternatives to the approach of elastic distance measures. Class-specific, Mahalanobis distances were shown to outperform simple Euclidean distance [11]. Additionally, Bagnall, Davis, Hills, and Lines proposed an ensemble of distance-based classifiers utilizing various data transformations including the original input data, auto-power spectrum, auto-correlation function, and principal components [12]. This study concluded that the correct data representation may provide competitive accuracy and improved interpretability when compared to complex classifiers and distance measures. In general, distance-based TSC tends to be well-suited for certain classes of problems such as recognition of one-dimensional shape representations; however, as in standard instance-based learning, the computational cost to classify new time series is relatively large, especially due to the high dimensionality of the data. Also, distance-based TSC provides limited insight for data mining tasks, as there is no reduction of dimensionality.

1.3.4. Feature-Based Methods

Feature-based TSC methods infer features from training data then apply standard machine learning methods to classify the resulting feature vectors. Here, we define a feature algorithm as a sequence of operations to transform one or more time series inputs to a single, scalar feature. Possible operations include, but are not limited to, digital filtering; thresholding; transformations among time, frequency, phase space, or hybrid domains; correlation analysis; principal component analysis; and many more. The algorithm must include a dimension reduction step to produce a scalar feature from the input signal. Feature algorithms can be considered as complex dimension reduction processes with the goal of optimally representing

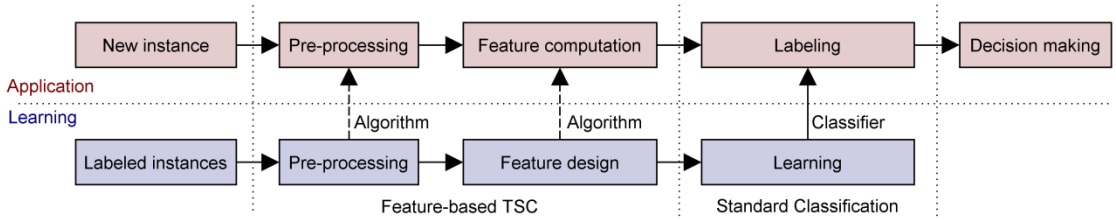


Figure 1 – Feature-based TSC workflow

information relevant to a decision process as opposed to the traditional dimension reduction goal of faithfully representing data in a lower dimensional space. In other words, feature algorithm design is a dimension reduction process with a performance objective other than self-representation. Figure 1 depicts the workflow for feature-based TSC in both the application and learning stages to show how these methods extend standard classification to time series data.

Traditionally, feature design has been performed by domain experts on an ad-hoc basis. While this approach generates very interpretable solutions, accuracy is rarely optimal except in cases of near complete domain knowledge. A generalized approach to optimal feature design that is able to produce low-dimensional, interpretable feature vectors would be highly valuable both for TSC applications and data mining and knowledge discovery from time series databases. Compared to distance-based TSC, feature-based methods are usually more interpretable depending on the complexity of the feature algorithms. Additionally, the computational cost is generally lower for classifying new time series although the learning stage is often computationally intensive. Another benefit of the feature-based approach is that the classifier can be replaced with a regression algorithm to perform time series regression and forecasting tasks using the same basic methods. A number of recent developments in feature-based TSC show promise as methods with competitive accuracy to distance-based techniques and various levels of interpretability.

Ye and Keogh proposed the concept of shapelets, which have been used to develop a variety of TSC methods [13]. A shapelet is a time series subsequence that is maximally representative of a single class or of the difference between classes. A shapelet feature is computed as the minimal distance between any subsequence in a time series and the shapelet. The originally proposed shapelet algorithm learned a decision tree by searching for the most informative shapelet at each node [14]. Later, shapelets were shown be more accurate when used to construct feature vectors for use with standard classifiers [15]. Shapelets are more interpretable than distance-based TSC, as the shapelet identifies critical time series behavior for the classification task.

Two recent works demonstrated approaches for generating large feature sets requiring advanced classifiers or an intermediate dimension reduction step using standard feature selection and construction methods. The time series forest (TSF) method randomly generates simple statistical features over thousands of random subsequences of the time series [16]. The features were then used to build a random forest resulting in one of the most accurate methods to date. Although the ensemble classifier and large feature set is difficult to interpret, Deng, Runger, Tuv, and Vladimir also produced temporal importance curves from the classifiers providing indication of the most informative regions of the time series.

A very different, pragmatic approach was taken by Fulcher, Little, and Jones to generate interpretable features. An extensive review of scientific literature was carried out to catalog thousands of features used in many disparate application areas [17]. To perform classification, the authors demonstrated a greedy, forward feature selection approach to reduce the feature database then application of a simple, linear classifier [18]. As all of the features included in the database are well understood by the scientific community, this approach produces perhaps the most interpretable solutions available. Interestingly, the accuracy achieved

utilizing thousands of expert designed features appears to be outperformed by the random feature set generated by the TSF approach, although the feature sets were utilized differently in the two methods making direct comparison of the features difficult.

Another approach is to search in the program space for feature algorithms from their constituent operations. In this space, solutions are composed of functions and arguments where the functions consist of any operations that may be part of a feature algorithm. Whereas treatment of feature design as a program search has certain advantages, the program space is infinite, making a brute force approach impossible. Some deterministic program search strategies exist such as Levin search [19], Hutter search [20], and the optimal ordered problem solver [21], which provide performance guarantees under certain conditions but can be computationally impractical for even moderately complex problems. Additionally, a random search over an infinite space is unlikely to be efficient. However, evolutionary computation provides a practical, proven search heuristic in the form of genetic programming (GP) formally introduced in [22]. GP methods for feature-based TSC are discussed in section 1.4.3.

1.4. Genetic Programming

GP is a search heuristic originally developed to automate computer programming [22]. Since its introduction, the idea has been adapted to a wide-range of problems including controller design, robotic programming, analog circuit design, and many more [23]. The search process evolves a population of candidate solutions where each individual represents a computer program built from a pre-defined set of inputs and operations. GP is unique as a search technique because the final size of the solution is not pre-determined. Interested readers are directed to [23] for a thorough introduction on the field.

Standard implementations are not directly applicable to the feature design problem, but the concept has been successfully adapted to a wide range of problems including feature design for image processing tasks such as edge detection and object recognition [24, 25]. In particular, GP-based methods have demonstrated success on problems with the following characteristics all of which apply to the field of TSC [23]:

- Minimal knowledge and understanding exists for problem domain.
- Size and structure of desired solution are unknown.
- Large amounts of digital data are available.
- Good solutions are easily tested but not directly obtained.
- Analytical solutions are unavailable.
- Optimal solutions are desired but approximations are acceptable.
- Incremental improvements are considered worthwhile.

1.4.1. Solution Space Design

The solution space for GP consists of a solution structure and the terminal and function sets that serve as building blocks for the programs. The terminal set consists of input data for the problem and often a set of pre-defined constants. Standard GP methods rely on the search process to evolve optimal constants within the solutions from this base set of constant values. The function set consists of all operations available within a solution. Typical function sets are on the order of 5-15 functions and can be as simple as basic mathematical operations of add, subtract, multiply, and divide.

The most common solution structure uses a tree-based program representation with functions serving as nodes connected by branches to the input data at the terminal leaves. The root of the tree generates the final output of the program. Other possible structures include stacks, graphs, recursive structures, and multi-program solutions. Careful design of the program

structure, function set, and terminal set is critical to create an effective solution space resulting in an efficient search process and high accuracy solutions.

1.4.2. Search Method

The search objective is to find the best combination of terminals and functions within the selected solution structure and size constraints to maximize, or minimize, a fitness measure. Fitness measures are highly problem-specific and often defined within a GP interpreter, which takes a candidate solution as input, executes the program, and returns the solution fitness. The search process uses standard evolutionary computation techniques with genetic operations specific to GP. First, an initial population is randomly generated and evaluated for fitness. Parent solutions are chosen from the population through standard selection methods such as tournament or roulette wheel selection. Next, the breeding process applies genetic operators to pairs of parents to produce offspring solutions. The offspring replace solutions in the population with poor fitness then the search continues until a desired fitness value or count of individuals is reached.

Genetic programming methods rely on the persistence and recombination of beneficial code segments in the population to find optimal-fitness solutions. The mechanism for this process is the preferential selection of solutions with better fitness as parents and utilization of appropriate genetic operators. Common GP operators include mutation and crossover. Mutation acts on a single parent to replace a section of code with a new, randomly generated section. Crossover transfers a section of code between parents. For tree-based structures, crossover selects a random subtree in one parent to replace the subtree at a randomly selected node in the second parent. Many varieties of crossover and mutation exist to improve search effectiveness or tailor the operators for a specific solution structure. GP operators are unique as they allow the

size of the offspring to vary from that of the parents removing the requirement in most evolutionary computation methods for a fixed, user-specified solution size.

1.4.3. Classifier Induction by GP

Genetic programming has been employed as a learner to infer classification algorithms for standard machine learning problems from supervised training data. Espejo, Ventura, and Herrera provide a thorough review on the use of GP for classifier induction [26]. Such methods can evolve decision trees, rule-based systems, discriminant functions, neural networks, and other structures with some advantages over standard learning algorithms. Classification accuracy of evolved systems is often equivalent or slightly better than traditional algorithms. However, the computational cost to evolve a problem-specific algorithm is significantly higher than using standard methods from machine learning, and the evolved classifiers are often very difficult to interpret or are "black box" in nature [26].

1.4.4. Feature Design by GP

As with traditional classification algorithms, standard GP methods accept multivariate scalar input but cannot take advantage of additional information within sampled signals. The literature includes a handful of adaptations for TSC and related problems. The scanning approach is the simplest adaptation wherein standard trees operate on a time series in a recursive manner to breed non-linear digital filters [27]. The Zeus system evolves feature vectors as a forest of strongly-typed trees with each tree producing a single feature [28, 29]. Two systems, genetic programming environment for FIFTH (GPE5) [30, 31] and parallel algorithm discovery and orchestration (PADO) [32, 33], circumvent the need for a strongly-typed system by using a single data stack and "smart" functions designed to handle all possible

Table 1 – Comparison of genetic programming methods for feature-based TSC

Method, Year	Solution structure	Search method	Classification	Comments
Scanning/ recursive trees, 1995 [27]	Single tree evaluated recursively while scanning along time series. Solutions act as non-linear digital filters.	Standard tree-based genetic programming with simulated annealing to optimize numeric parameters	Single feature output requires classification to be performed simultaneously with feature design during algorithm search.	Solution structure does not naturally lend itself to many common features, but use of numerical optimization within GP search proved beneficial.
PADO, 1995 [32, 33]	Separate program computes confidence for each class. Programs consist of nested graph structures acting on single data stack	Custom genetic operators evolve graphs with function parameters evolved as values on the data stack.	Feature selection performed through adjustment of weights in voting procedure to determine program output. Classification simplified to selecting class based on confidence levels.	Structure and functions designed such that many input types can be handled including sequences, images/matrices, and video data.
Zeus, 2002 [28, 29]	Forest of strongly-typed trees where each tree computes one feature in feature vector	Standard tree-based genetic programming and evolved constants	Feature design and selection performed through custom genetic operators with classification of feature vectors by SVM classifier.	Proposes and advocates use of wrapper approach wherein evolved front-end feature algorithms are combined with standard back-end classifiers.
FIFTH/ GPE5, 2007 [30, 31]	Linear series of "smart" functions with single data stack written in custom signal processing language	Standard genetic operators with crossover constrained to compatible sequences and evolved constants	Single feature output requires classification to be performed simultaneously with feature design during algorithm search.	Uses high-level functions such as the FFT which are efficiently calculated from standard optimized libraries and are unlikely to be evolved from basic mathematical operations

input data types. Table 1 compares the solution structure and search method for these systems and discusses how each system accomplishes the TSC task. Standard evaluation methods are usually employed to compute metrics such as classification accuracy to serve as the fitness measure for the GP search.

1.5. Contributions of the Dissertation Work

The contributions of this dissertation work are motivated by the need for improved time series tools in structural engineering applications such as structural health monitoring and non-destructive evaluation. Development of these tools necessitated integration of techniques from signal processing, machine learning, numerical optimization, and evolutionary computation as well as novel advancements in each of these areas. The significant contributions are summarized as follows:

1. Autofead method performs automated feature design for time series classification including numerous novel developments in the areas of time series analysis, evolutionary computation, and numerical optimization

    a. Autofead yields a compact, interpretable solution space design for discovery of TSC solutions in form of expert designed systems

    b. A comprehensive, flexible library of functions for digital signal processing and time series analysis is generated and exploited in the design.

    c. A hybrid numerical optimization and genetic programming search method with novel genetic operators is developed for implementation.

    d. A sequential global optimization scheme with a weighted evaluation framework for decoupling of parameter sets in complex optimization scenarios is presented.

    e. A fast, non-parametric, binary classification procedure to estimate univariate decision boundaries.

2. Experimental validation of Autofead method is performed and a comprehensive benchmark study with comparison to state-of-the-art time series classification methods is presented.

3. The data mining potential of Autofead method is demonstrated through numerous case studies including a variety of structural health monitoring applications.

4. A fast implementation of non-probabilistic uncertainty analysis for feature-based time series classification algorithms to evaluate solution robustness is presented.

1.6. Outline of the Dissertation

The remainder of this dissertation details, demonstrates, and evaluates the Autofead method for TSC and expounds on various related topics. Chapter 2 presents Autofead in-depth including the development process and history of the method. Chapter 3 experimentally validates the approach on two categories of synthetic TSC problems referred to as the dynamics, stationarity, and distributions (DSD) problem and signal detection problems. These problems are valuable for validation purposes as optimal analytic solutions are known and require a wide range of evolved solution behaviors. Chapter 4 presents results on three laboratory experiments related to the field of structural health monitoring. The selected experiments include binary classification, multi-class classification, and multivariate time series regression. A comprehensive benchmarking study on 49 openly-available, real-world datasets is described in Chapter 5 with direct comparison of Autofead to state-of-the-art TSC methods. Further evaluation of the Autofead method is discussed in Chapter 6 with targeted analysis carried out on experiments from the previous chapters. Chapter 7 proposes a non-probabilistic uncertainty analysis technique for evaluating the robustness of TSC solutions. The robustness analysis is intended to alleviate some of the issues of the Autofead method described in Chapter 6 but is applicable to other TSC methods as well. Lastly, conclusions and recommendations for future work are presented in Chapter 8.

Portions of this chapter have been published in IEEE Transactions on Evolutionary Computation, Dustin Harvey and Michael Todd, 2014. The title of this paper is “Automated Feature Design for Numeric Sequence Classification by Genetic Programming”. Additional portions of this chapter have been submitted for publication in Data Mining and Knowledge Discovery, Dustin Harvey and Michael Todd, 2014. The title of this paper is “Automated

Feature Design for Time Series Classification". The dissertation author was the primary investigator and author of these papers.

Chapter 2
Autofead Method

2.1. Overview

This dissertation presents a novel TSC method for automated feature design called Autofead. The overall goal of this work is to develop a general learner that can infer from time series examples an optimal, minimum-basis feature set for a given supervised training data set. Additionally, a method that produces interpretable features algorithms in a compact representation is desirable for time series data mining. Therefore, Autofead is designed according to the eight following principles:

1. Support multivariate, one-dimensional signal input
2. Perform mutli-class classification and regression
3. Generate feature algorithms substantially similar in form to human-designed features
4. Minimize required user input and assumptions on input data
5. Use standard pattern recognition methods to evaluate candidate feature sets
6. Employ numerical optimization methods appropriately and efficiently

7. Avoid black box solutions and strive for easy interpretation of features

8. Promote independence of features to minimize information redundancy

This work represents the first automated feature design system for numeric sequences to leverage the power and efficiency of both numerical optimization and standard pattern recognition algorithms. Autofead uses a GP variant to evolve features for input to standard pattern recognition algorithms through a wrapper approach as depicted in Figure 2. The unique solution space is specifically designed to generate compact, interpretable solutions while also allowing flexibility of the method to produce a wide array of feature behaviors. Unlike traditional GP, the solution structure has no constants in the terminal set; instead, a hybrid search is employed where parameters within feature algorithms are numerically optimized prior to fitness evaluation. Feature selection is performed concurrently with feature design by intelligent genetic operators. Fitness of candidate solutions, i.e. feature vectors, is measured by the classification accuracy of the selected classifier applied to the measured features through standard sampling procedures.

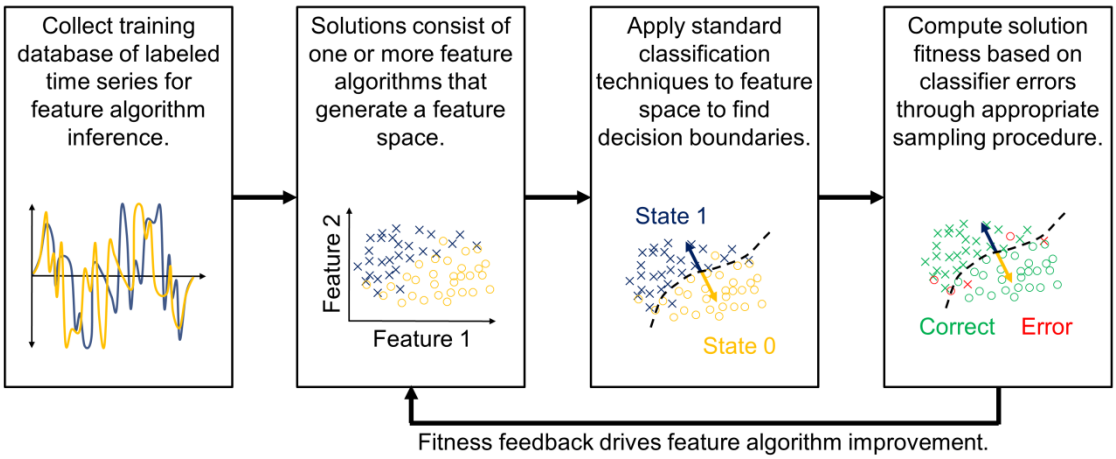


Figure 2 – Wrapper approach workflow for TSC

2.2. Solution Space

The Autofead solution structure is highly-constrained and simplified compared to other genetic programming systems developed for feature-based TSC such as Zeus [28, 29], FIFTH [30, 31], and PADO [32, 33], see section 1.4.4. A restricted structure was chosen primarily for two reasons. First, most conventional, human-designed algorithms are linear in structure and operate on one or two signals, i.e., not tree or graph structures; second, the restricted solution structure allows inclusion of high-level operations within a large function library without creating an unfeasibly large search space. The Autofead solution space includes the program structure, function library, function parameters, and classifier selections.

2.2.1. Program Structure

Autofead solutions consist of a set of one or more features each computed through a series of sequence-handling functions operating on one or two time series inputs. For example, a feature that computes energy would use the two-function algorithm [*square*, *sum*]. Each feature algorithm ends with a dimension-reduction function such as *sum* to ensure scalar feature output. Relatively short algorithms of 2-5 functions are typical facilitating easy analysis and interpretation of solutions. Figure 3 diagrams a single solution individual. The example shown

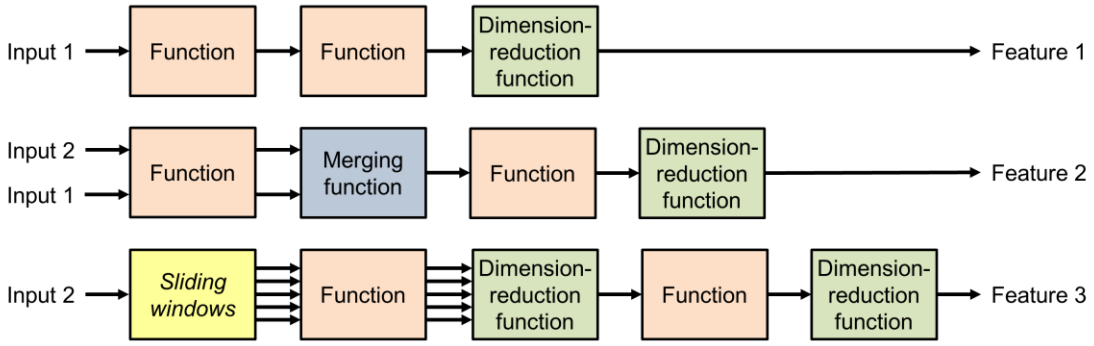


Figure 3 – Example Autofead solution structure

includes three features of increasing complexity through the inclusion of a merging function in feature 2 and the *sliding windows* function in feature 3.

Merging functions allow a feature to operate on two input time series of the same length. *Cross-correlate* is an example of a merging function. Each input to the feature algorithm is processed identically through functions prior to the merging function.

The *sliding windows* function transforms the input sequence to a matrix of subsequences. Subsequent functions operate on each subsequence individually until a dimension-reduction function returns the data flow to a single sequence. A second dimension-reduction function is then required to compute a scalar feature. This function is inspired by the segmentation process used in power spectral estimation via Welch's method [34].

Terminal sets in Autofead are greatly simplified, as there is no notion of constants. The terminal set simply consists of one or more time series inputs in the training data instances. For multivariate problems, each feature has an associated input index to select which time series the algorithm uses as input, or two inputs in the case of merging functions. While the method is intended for direct input of raw measurement data such as time series, domain knowledge may indicate that alternative representations are beneficial for a specific application. In this case, transformed input data can be used instead of or in addition to the raw time series.

Class mean signals allow for additional interesting feature behaviors. A class mean signal is generated by merging functions as the mean of all signals in the training data from a single class. The merging function then utilizes the computed class mean signal as the second input sequence. For example, the feature [*element difference* (class 0), *square*, *sum*] computes the square of the Euclidean distance between the input time series and the mean signal for class 0. The first input index must always be non-negative to select an input time series. When a

merging function is present and class mean signals are enabled, the implementation adds a selection for each class mean signal as a possible second input index.

2.2.2. Function Library

The function library is designed to allow creation of a wide range of feature behaviors from relatively short feature algorithms with minimal function redundancy. The functions are selected to admit features derived from the distributions, dynamics, and transient behaviors of time series. In many cases, functions were formed by decomposing common signal processing algorithms into their individual operations and then reducing redundancy within the overall function set [34]. Although the current library has demonstrated success on a wide range of problems, design of the optimal, minimal-basis function set for time series analysis remains an open research topic. The current set of 47 functions and their parameters is described in Table 2. Each function, with the exception of *sliding windows* and merging functions, takes a single sequence as input and outputs a modified sequence. Dimension-reduction functions reduce the output sequence length to a single sample.

The *sliding windows* function is of particular interest as it provides the primary mechanism for analysis of transients or non-stationarity in a sequence. The input sequence is expanded to a matrix of subsequences by extracting consecutive overlapped windows. To constrain the function to a single parameter and limit growth of data sizes within algorithms, the number of windows constructed and number of overlapping samples in sequential windows is internally determined based on the ratio of the window length parameter to the input sequence length. The total data size of the output subsequence matrix is limited to between 1.5 and 2 times the size of the input sequence.

Table 2 – Autofead function library

Function	Output	Parameter (*integer type)
	Element operations	
Absolute value	Absolute value of input values	-
Cube	Cube of input values	-
Exponential	Exponential function of input values	-
Inverse	Reciprocal of input values	-
Log10	Base-10 logarithm of absolute value of input values	-
Sigmoid	$x_i/(1+\lvert x_i\rvert)$ for input values, x_i, bounded on [-1,1]	-
Sign	Sign of input values, -1 or 1	-
Square root	Square root of absolute value of input values	-
Square	Square of input values	-
	Distribution-altering functions (sample order independent)	
Center	Input values with mean of all training data removed	-
Center and scale	Standard deviations from mean of all training data for input values	-
Control chart	Input with central values of data range set to center of range	Percentage of range kept (0-1)
Demean	Input values with mean of individual sequence removed	-
Normalize	Standard deviations from mean of individual sequence for input values	-
Set minimum value	Input with values below threshold raised to threshold	Data range threshold (0-1)
Set maximum value	Input with values above threshold lowered to threshold	Data range threshold (0-1)
	Order-dependent functions	
Auto-correlation function	Biased estimate of input sequence auto-correlation function for positive lags	-
Convolve	Convolution of input sequence with pre-specified sequence	-
Cumulative summation	Cumulative summation of input sequence	-
Difference	First differences of input sequence	-
FFT magnitude	Magnitude of FFT of input sequence, [0,π] rad/s bin range	-
FFT phase	Phase of FFT of input sequence, [0,π] rad/s bin range	-
FFT real	Real part of FFT of input sequence, [0,π] rad/s bin range	-
FFT imaginary	Imaginary part of FFT of input sequence, [0,π] rad/s bin range	-
Hanning window	Input sequence with Hanning window applied	-
Hilbert magnitude	Magnitude of Hilbert transform of input sequence	-
Hilbert phase	Phase of Hilbert transform of input sequence	-
Hilbert imaginary	Imaginary part of Hilbert transform of input sequence	-
High-pass filter	High-pass filtered input sequence, zero-phase digital filtering by 3rd order Butterworth filter	Normalized cutoff frequency (0-1, upper bound = π rad/s)
Low-pass filter	Low-pass filtered input sequence, zero-phase digital filtering by 3rd order Butterworth filter	Normalized cutoff frequency (0-1, upper bound = π rad/s)
Sort order	Indices of values in sorted input sequence, ascending order	-
Wavelet	Detail coefficients of discrete-wavelet transform using pre-specified wavelet family, parameter value of 0 produces approximation coefficients at the maximum useful level of decomposition	*Wavelet detail level

Table 2 – Autofead function library (continued)

Function	Output	Parameter (*integer type)
Index-altering functions		
Keep beginning	Input sequence with samples removed from end	*Samples to remove
Keep end	Input sequence with samples removed from beginning	*Samples to remove
Sliding windows	Subsequences extracted from sliding a window along input sequence, number of windows and overlapping samples between adjacent windows determined internally, no change to already windowed input	*Window length
Transpose windows	Swaps window indices and sample indices	-
Merging functions		
Cross-correlate	Cross-correlation of two input sequences, output is same length as inputs	-
Element sum	Element-wise sum of two input sequences	-
Element difference	Element-wise difference of two input sequences	-
Element product	Element-wise product of two input sequences	-
Element quotient	Element-wise quotient of two input sequences	-
Dimension-reduction functions		
Select	Sample at index of input sequence; internal brute force index search to optimize performance of output as single feature solution; for windowed input, each subsequence is treated as a separate fitness case for index selection	*Internal index selection
Bisection select	Same as *select* function but using bisection index search	*Internal index selection
Sorted select	Same as *select* function but with input sequence sorted	*Internal index selection
Sorted bisection select	Same as *sorted* select function but using bisection index search	*Internal index selection
Slope fit	Slope of best linear fit to each input sequence	-
Sum	Sum of all input values, occurs implicitly at end of every feature	-

2.2.3. Function Parameters

In the feature algorithm design process, optimal features may be overlooked because of poor parameter choices. For example, in a filtering operation, proper filter design parameters maximize signal-to-noise whereas poor parameter choices can completely obscure the useful information. Numerical optimization techniques are well understood and designed specifically to handle this task; however, standard GP uses pre-defined or random constants that are evolved within the program. This approach does not provide for an efficient numerical search. Some previous work has adopted a hybrid approach including a numerical optimization step prior to fitness evaluation resulting in improved solutions with faster convergence [35].

The hybrid approach is adopted by Autofead to perform optimization of the parameters listed for the functions in Table 2. The function parameters are divided into continuous type (normalized between 0 and 1) and integer type, which are handled separately within the parameter optimization process. Continuous parameters include filter cutoff frequency for filtering functions and threshold values for thresholding functions. *Keep beginning* and *keep end* require an integer type parameter that specifies the number of samples to remove from the input. Additionally, *wavelet*, *sliding windows*, and the four *select* function variants all use integer type parameters.

2.2.4. Classifiers

Bagnall et al. take the position that the largest performance gains in the area of sequence classification can be achieved through optimal transformation and representation of input data as opposed to development and use of complex classifiers [12]. Therefore, Autofead uses the wrapper approach with the idea that a simple classification algorithm is sufficient to determine relative performance of features during the feature algorithm design process. Classifiers for Autofead solutions are selected through the evolution process from a set of pre-selected standard algorithms. Once the final feature set is formed, more complex classifiers could be applied to further improve solution accuracy. Because the Autofead solution space has many degrees-of-freedom with multiple features, many functions, and selection of parameter values, the method is susceptible to overfitting with small data sets. For this reason, it is important to select simple, high-bias classifiers to reduce the possibility of overfitting. Recommended classifiers include Gaussian Naïve-Bayes, Linear Discriminant Analysis, and Logistic Regression. These three classifiers were implemented using the machine learning package scikit-learn from Pedregosa et al. [36].

2.3. Search Method

Designing an Autofead solution consists of selecting a classifier and the number of features then specifying for each feature the input indices, functions in the algorithm, and the functions' parameter values. The solution space is far too large for a brute force approach, and too complex (including potentially highly discontinuous) for standard optimization methods; however, the hybrid GP and numerical optimization process depicted in Figure 4 has shown to be very effective in navigating the space efficiently and consistently. To configure an Autofead search run, the user defines the training instances, or fitness cases, and sets any problem-

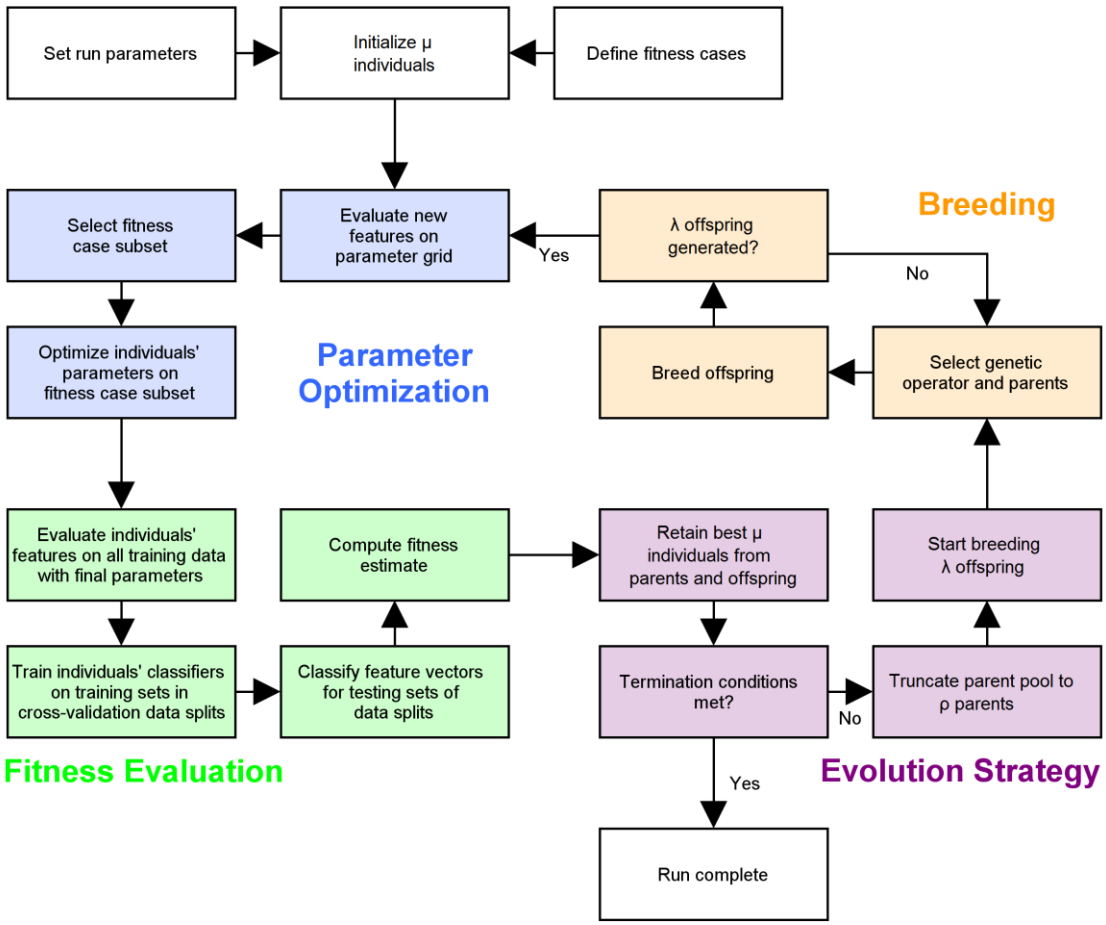


Figure 4 – Autofead search process flowchart

specific run parameters. The search begins with a randomly-generated initial population. Then the search proceeds through four modules of parameter optimization, fitness evaluation, evolution strategy, and breeding. The run continues until termination conditions are met such as a desired fitness level or a maximum number of individuals.

2.3.1. Population Initialization

The initial population of an Autofead run is generated through a ramped initialization to create solutions of varying size. The number of features, the function count within the feature algorithms, and the number of allowed parameters within the feature algorithms are uniformly distributed throughout the initial population up to the selected maximum initial solution size constraints. The number of allowed parameters is constrained in addition to function count within a feature algorithm to manage computational cost as most of the computational effort of a run is directed toward parameter optimization. Additionally, the initialization process enforces uniform usage of classifiers and functions within the population.

For multivariate problems, each feature algorithm requires the additional specification of an input index to select which time series the algorithm will use as input(s). The first input index for each feature in the initial population is distributed uniformly between the available inputs. For features containing a merging function, the second input index is chosen randomly from the set of remaining inputs and class mean signals, if enabled.

2.3.2. Parameter Optimization

Before solutions are evaluated for fitness, the parameter values within the feature algorithms must be selected. A single individual can contain multiple features each containing multiple parameters. Due to the nature of the parameter surfaces, a global optimization strategy is required; however, global optimization of all parameters within an individual during each

iteration of the search process is prohibitively expensive computationally. Additionally, the goal of the parameter optimization scheme within the Autofead search process is not necessarily to find the true optimal parameters with high accuracy but rather to assist the overall search by ensuring that good feature algorithms are not overlooked due to poor parameter choices. The parameters of the final solution's features can be re-optimized post-search by standard global optimization methods before implementation in real systems.

Accounting for these considerations, a sequential global optimization strategy is employed including a number of parameter independence assumptions. Assumption of independence both between features in an individual and between parameters in a feature algorithm greatly reduces the computational requirements for optimization. First, a parameter grid is built and evaluated for each new feature in the population. Grid evaluations are stored for subsequent optimizations to reduce repeated computations. Then, parameters are locally optimized starting from the best grid point on a randomly selected subset of the fitness cases.

Within a single individual, each feature is locally optimized independently to avoid the computational cost of training and evaluating multivariate classifiers to compute the objective function. Independent optimizations, however, may lead to high correlation between features if no method is used to promote orthogonality in the optimization process. For example, in an individual containing two identical algorithms that estimate location of a spectral peak in a parameterized frequency band, performance will improve if the two features operate in separate frequency bands to reduce redundant information. Therefore, a fitness case-weighting scheme is employed within the local optimization stage. Error-based weights for the fitness cases are adjusted after local optimization of each feature. For classification, the solution is reduced to only the features with finalized parameter values then predictions are made for the fitness case subset. The error for each fitness case is computed by the overall search fitness function. Next,

the errors are normalized to the relative error ranks across the fitness cases before updating the weights. New weights are computed as the average of previous weights and the error ranks. Therefore, subsequent features' parameters are optimized with emphasis on the fitness cases which produce largest error using previously optimized features in the individual.

The objective function for parameter optimization can be computed as classification error for any high-bias classifier or a standard divergence or information measure. For binary classification, Autofead uses a direct threshold estimation routine to determine decision boundaries for classification from a single feature. First, a low-order polynomial is fit to the difference between the normalized, empirical cumulative density functions for the two classes. Thresholds are computed as the extrema of the polynomial fit, effectively estimating zero-crossings of the difference in class probability density functions. Use of a polynomial of 3rd or 4th order ensures a small number of decision boundaries. This procedure produces a very fast, reasonably-accurate, univariate, binary classifier. For multi-class problems, Autofead uses linear discriminant analysis as the classifier in the objective function.

To further reduce computational expense, the optimization of the set of integer parameters is performed independently from each continuous parameter in a given feature. The continuous parameters in the filtering and thresholding operations tend to have strong effects on feature performance without altering the overall feature behavior. Additionally, all of the continuous parameters have a natural default value that results in minimal or no change to the input. For instance, a high cutoff frequency parameter in the *low-pass filter* function causes minimal filtering effects. In contrast, the integer parameters tend to have coupled relationships leading to a wide range of possible behaviors. For example, the *sliding windows* function followed by the *select* function creates a decimation behavior where the window length parameter controls the new data rate and the index selection parameter varies the first index

retained in decimation. For a feature containing p_I integer parameters and p_C continuous parameters, this decoupling leads to a single, p_I-dimension optimization followed by p_C, single parameter optimizations instead of a single, $(p_I + p_C)$-dimension optimization. Figure 5 diagrams the parameterization scheme for a single solution individual containing both integer and continuous type parameters.

For the integer parameter set, the initial grid search is performed using an adaptive grid density determined by variance-based total effect sensitivity indices [37]. Default values for the continuous parameters are used while building the integer parameter grid. The grid search for subsequent continuous parameters assumes the best parameter values from previous grid searches. Local optimization of the integer parameter set is performed through a bisection

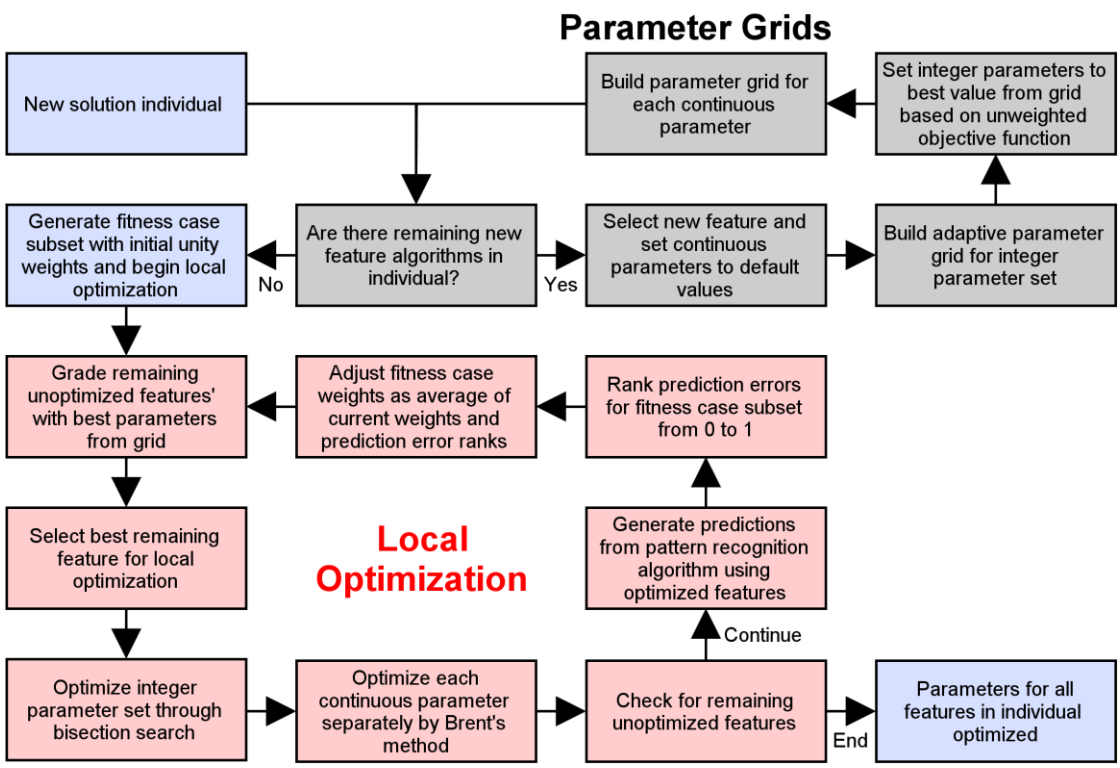


Figure 5 – Parameter optimization scheme flowchart

search with initial step sizes calculated from the adaptive grid density in each dimension. Then, the continuous parameters are optimized via Brent's method [38].

As noted in Table 2, the family of four *select* functions has internally-optimized index parameters. Because these functions occur at the end of a feature, the feature output can be evaluated for different index selections without reevaluating the entire feature from the beginning. Two unsorted and sorted variants of the *select* function are included in the library employing either a brute force or bisection search. The index parameter for the unsorted *select* function tends to have high sensitivity and a sparse optimization surface. These characteristics suggest a brute force search, which would be prohibitively computationally expensive if reevaluating long features for each index but is tractable through the internal index selection. For the *sorted select* function, the index parameter surface tends to be much smoother, thus admitting a bisection search. Hence, it is recommended the *select* and *sorted bisection select* functions be used in the function set with *bisection select* and *sorted select* omitted.

2.3.3. Fitness Evaluation

Fitness of candidate individuals is based on performance of the selected pattern recognition algorithm applied to the individuals' feature vector. In the case of regression, linear regression algorithms and RMS error are standard tools and work well for Autofead's wrapper approach. For classification problems, the fitness function, F, is calculated as

$$F = E + \delta Q, \tag{1}$$

where E and Q are classification error and quadratic loss, respectively. The value of δ is selected to be small such that the only role of Q is to break ties in E. The inclusion of quadratic loss as a tiebreaker is especially important in cases of small data sets where many candidate solutions may have zero classification error on the training data. It is important to note that

fitness here is defined as a metric to be minimized contrary to convention in most evolutionary optimization processes.

Fitness is estimated by an adaptive, repeated, stratified k-fold cross-validation procedure. In each data split, k-1 folds are used for training with a single fold held out for testing. The number of folds and repetitions is determined adaptively and designed to exhaustively sample small classes, provide accurate estimates for medium-sized classes, and reduce computations for very large classes. Number of repetitions, R, and number of folds, k, are computed based on the number of instances in the smallest and largest classes, N_{min} and N_{max}, respectively. Fold count k is restricted to the interval [5, 10] and varies linearly with N_{min} such that when $N_{min} = 10$, $k = 10$ creating a leave-one-out procedure for the smallest class. At $N_{min} = 250$, k is reduced to the minimum value of 5 folds. Furthermore, folds are restricted to a maximum of 50 instances per class to reduce computations. This means classes with more than 250 instances do not use all available data within a single cross-validation procedure. Therefore, the number of repetitions varies with the largest class as $R = N_{max}/10$ with a maximum value of 5 to make better use of very large classes. At the beginning of a single search, or run, a fixed set of training and testing splits is generated for use throughout the run. Individual runs on the same problem create a new set of splits with the same number of total repetitions and folds.

2.3.4. Evolution Strategy

The evolution strategy determines how the population is controlled and allowed to evolve. A generational approach is the simplest strategy where the entire population is replaced with an equal number of offspring during each iteration of the search process. An issue with the generational approach arises when the search moves in a detrimental random direction such that the offspring generation has lower fitness than the parents. Introducing elitism is one method to

alleviate this problem. Autofead adopts a more advanced $(\mu/\rho + \lambda)$ strategy that maintains a large diverse population of μ individuals based on methods in Bäck [39]. In each iteration of the search, the population is truncated to a parent pool of the best ρ individuals from which λ offspring are generated. Finally, an updated population is selected from the best μ offspring and parents such that the search only moves in the direction of the offspring if they outperform the parents. Figure 4 depicts the $(\mu/\rho + \lambda)$ strategy in the context of the overall Autofead search process.

2.3.5. Selection and Breeding

The GP portion of the search involves selection of parent pairs from the truncated parent pool and breeding of offspring solutions through genetic operators. This part of the search selects the most useful functions in the function set, designs the feature algorithms, and performs feature selection to combine features into an individual forming full solutions. Classifier selection is also performed through the GP component of the hybrid search.

Parent selection within Autofead can be performed by any standard selection mechanism such as roulette wheel, a.k.a. fitness proportional, selection or reward-based selection. Tournament selection provides certain benefits in terms of search efficiency including allowing back chaining [40]. In back chaining, fitness of individuals is only evaluated on an as-needed basis, often allowing many evaluations to be skipped without impacting the search. Additionally, hypothesis testing can be implemented within a tournament to reduce evaluations to the minimal amount required to select the preferred parent with statistical significance. In Autofead, all individuals are initially evaluated on a small number of cross-validation data splits to determine the ranking required for parent pool truncation. Within a tournament, further evaluations are carried out on remaining data splits in order, as needed, until the tournament is won by an individual according to the corrected, resampled *t*-test with 90% confidence [41]. All

evaluations performed within a tournament are stored to improve the individuals' fitness estimates and to avoid repeated computation in future tournaments. Individuals that reach the final population in an Autofead run are fully evaluated on remaining data splits.

The Autofead genetic operators are simplifications of the standard "crossover", "mutate", and "reproduce" operators in tree-based GP supplemented with operators that intelligently perform feature selection. The usage frequency of each operator is controlled through assigned genetic operator probabilities. "Crossover" and "mutate" modify a single feature algorithm within the first parent. "Crossover", the most often used operator, replaces a random segment of functions within one feature algorithm with a random algorithm segment from one of the second parent's features. The mutation operator randomly removes, inserts, or swaps out a single function in a feature algorithm or changes the feature's associated input index for multivariate problems. "Mutate" can also modify the solution's pattern recognition algorithm section. Mutation helps maintain function diversity within the population as it converges and enacts small changes to locally search the solution space. "Reproduce" clones a single parent with no modification of the features. Although the feature algorithms are identical for the clone, parameters may change as optimization is performed on a new fitness case subset in the subsequent search iteration.

Feature selection is performed through the "remove feature" and "add feature" operators inspired by the architecture-altering operations by Koza [42]. The objective function from the parameter optimization process is used to identify the relative value of adding features to or removing features from an individual. The "remove feature" operator deletes the worst feature from a single parent according to the unweighted objective function. "Add feature" begins by generating fitness case weights using all features from the first parent by the same method used in parameter optimization. Then, the features from the second parent are evaluated

using the weighted objective function to determine which feature is most beneficial to transfer to the first parent.

2.4. Problem Configuration

As in any evolutionary search process, Autofead requires specification of the many parameters that control the size of the population, population initialization, frequency of genetic operators, and other components of the method. For each experiment performed as part of this dissertation, the run configuration is summarized in a Koza tableau as shown in the example of Table 3. The example table describes the primary run parameters that may be varied to best suit specific problems. Choices related to the function set, parameter optimization process, population size, and termination conditions have significant impact on the computation expense of the run and are therefore often necessitated by the available computing environment and available time. Run parameters used in the following studies are based on common practice with similar GP systems and the authors' engineering judgment.

Table 3 – Autofead Koza tableau example

Parameter	Parameter description
Objective	Problem specific objective related to the desirable features for Autofead to design
Solution structure	One or more features conforming to Autofead structure and selection of pattern recognition algorithm
Function set	Count of functions from function library in Table 2 and any omissions
Pattern recognition	Specification of classifier, regressor, or list of available selections
Fitness	Fitness measure, bounds, and direction of optimality
Fitness case sampling	Sampling procedure selected
Evolution strategy	Population and breeding configuration
Solution size	Individual solution size constraints
Population initialization	Initialization approach and initial individual solution size constraints
Selection	Parent selection method and related parameters
Genetic operators	List of genetic operators and corresponding genetic operator selection possibilities
Termination	Termination conditions such as fitness level, search iterations, or total individual count
Run repetitions	Number of runs performed on problem

2.5. Development History

The three-year development of Autofead began in the application area of feature extraction for structural health monitoring systems described further in Chapter 4. The authors soon broadened the research to address the more general area of time series classification. The versions of Autofead presented during early development were significantly similar to the current method but lacked certain minor components and some major implementation improvements [43, 44]. Important modifications present in later studies include improvement of the evolution strategy, fitness estimation, and function library as well as the addition of merging functions, class mean signals, and evolved pattern recognition algorithm selection [45]. The authors' believe the Autofead implementation presented in this dissertation represents a substantially mature method; however, the potential for further refinement is acknowledged under the recommendations for future work in section 8.2.

Portions of this chapter have been published in IEEE Transactions on Evolutionary Computation, Dustin Harvey and Michael Todd, 2014. The title of this paper is "Automated Feature Design for Numeric Sequence Classification by Genetic Programming". Additional portions of this chapter have been submitted for publication in Data Mining and Knowledge Discovery, Dustin Harvey and Michael Todd, 2014. The title of this paper is "Automated Feature Design for Time Series Classification". The dissertation author was the primary investigator and author of these papers.

Chapter 3
Experimental Validation

3.1. Detecting Dynamics, Stationarity, and Distributions

The first validation experiment requires Autofead to generate three simple but very different features and perform feature selection to combine the three features in a single individual. The three required features measure information related to signal dynamics, stationarity, and distributions, here referred to as the DSD problem. It is important to note that derivation of the optimal, analytic solutions requires complete knowledge of the signal generation process. Autofead is designed to discover optimal and near-optimal solutions with minimal knowledge represented by training data alone. No application-specific information is provided to the system in any of the validation experiments in this chapter that would benefit the method.

The DSD problem involves separating a zero-mean, unit-variance, white, Gaussian noise (WGN) process in class 0 with one of three equally-likely subclasses within class 1. Subclass 1a is generated by passing WGN through a weighted, 2-sample moving average filter.

Table 4 – Koza tableau for DSD run configuration

Parameter	Setting
Objective	Derive features to separate class 0 from each subclass of class 1
Solution structure	Set of features consisting of a sequence of functions, omitted class mean signals
Function set	35 functions from function library plus separate function, *sum windows*; omitted *center, center and scale, convolve, cross-correlate, element difference, element product, element quotient, element sum, inverse, sigmoid, slope fit,* and *wavelet*
Pattern recognition	Kernel-based Naïve-Bayes classifier
Fitness	Minimize classification error bounded on [0,1]
Fitness case sampling	96% testing percentage split for each generation
Evolution strategy	Generational, population size of 500, no elitism
Solution size	Maximum size of 5 features each limited to 15 functions and 8 parameters
Population initialization	Ramped to initial maximum size of 3 features each limited to 5 functions and 3 parameters
Selection	Tournament, tournament size 4
Genetic operators	Crossover, 80%; mutate, 5%; reproduce, 5%; add feature, 5%; remove feature, 5%
Termination	20 generations; 10,000 individuals
Run repetitions	30; 300,000 total individuals

Subclass 1b has a slight variance increase halfway through the signal, and a white, uniform-distribution noise process produces subclass 1c. Each subclass involves a slight modification to the WGN process of class 0, but the expected mean values and signal energies are unaltered. All signals consist of 100 samples with index 0 to 99. Table 4 summarizes the Autofead configuration for the DSD problem. The best Autofead solution from a single run contained five features. Two features were redundant and have been removed with no loss of fitness. The remaining three features each target detection of a specific subclass of class 1.

Subclass 1a has altered dynamics from a moving average filter with weights $\sqrt{0.8}$ and $\sqrt{0.2}$ for the 0 and 1 sample lags, respectively. The optimal feature to detect this change is the value of the auto-correlation function at 1 sample lag. The Autofead feature took a different approach by calculating an energy measure for the high-frequency band of the signal. The feature algorithm is [*high-pass filter* $(p = 0.70)$, *cube*, *absolute value*, *sum*]. Because the moving average filter is a low-pass filter, class 0 contains higher energy than subclass 1a in the high-frequency band. The sequence [*cube*, *absolute value*, *sum*] computes an energy measure using the third power instead of the second power.

Subclass 1b is non-stationary with a variance change from $\sqrt{0.5}$ to $\sqrt{1.5}$ at sample 50. The global dynamics and distribution of subclass 1b are identical to a WGN process requiring the optimal feature to detect a non-stationary change in the signal. The difference in the variance of the two signal halves provides the optimal discriminator between class 0 and subclass 1b. Autofead accomplishes the task through the feature algorithm [*cube*, *absolute value*, *cumulative summation*, *select* ($p = 46$)]. Again, the sequence [*cube*, *absolute value*, *sum*] from the *cumulative summation* computes an energy measure. By taking a cumulative summation and selecting index 46 as the feature value, only samples from the low variance section of the signal are included in the feature. Here, the optimal parameter choice is index 49 to capture the entire first half of the signal, but index 46 was selected as optimal for the particular subset of training fitness cases used during optimization with minimal performance loss.

Lastly, a white, uniformly-distributed noise process is used to generate subclass 1c. Bounds of -$\sqrt{3}$ to $\sqrt{3}$ for the uniform distribution are selected to produce zero-mean, unit-variance signals. For this subclass, the optimal feature simply detects the presence of any values outside the bounds of the uniform distribution. Consequently, the associated feature in the Autofead solution finds the largest magnitude sample using the algorithm [*absolute value*, *sorted select* ($p = 99$)]. The *sorted select* function, using index 99, selects the maximum value.

Figure 6 illustrates how class 0 and each subclass are separated by the associated 3 features in the Autofead solution. Each column provides a series of images corresponding to the processing steps in a single feature algorithm. For each image, the horizontal axis indicates samples of the sequences at the given processing step and is supplemented with the sum of the current sequences on the right end. The vertical axis represents the entire data range for individual sequence indices. At each sequence index, the values from all fitness cases are sorted

and replaced by their known class labels. Then, the background images are generated by assigning a color or gray-level value to each class. In the foreground, median values and quartiles at each sample across all sequences are shown by black and white lines, respectively.

This visualization technique provides a number of advantages for feature algorithm analysis and design in conjunction with Autofead. Although absolute values and distribution shapes are admittedly obscured, class separability is only dependent on the relative distribution of the classes over the value range, which is explicitly presented by the visualization. For example, Figure 6(a) shows a completely heterogeneous image indicating that the two class distributions are highly overlapped across the entire sequence. In terms of feature design, a fully heterogeneous image dictates application of an order-dependent function as any element operations or distribution-altering, index-altering, or dimension-reduction functions would act equally on both classes.

In contrast, the final image for each feature shows clear regions of homogeneity indicating a difference between classes that can be exploited as a feature. After the *absolute value* operation in feature 1, Figure 6 (d) shows the largest values across the sequence belong to class 0 and the smallest values fall in subclass 1a. The sum column on the right clearly shows that summing the sequences at this point generates a feature with excellent class separability.

Features 2 and 3 end with different dimension-reduction functions than feature 1 corresponding to specific characteristics apparent in the processing images. Figure 6(h) contains columns in the middle of the sequence with large class separability even though the sum column only shows small class differences. Consequently, feature 2 ends by selecting index 46 as the feature output. Lastly, Figure 6(j) shows that the largest values across all indices belong unanimously to class 0. The presence of a percentile level (row in the image) composed of

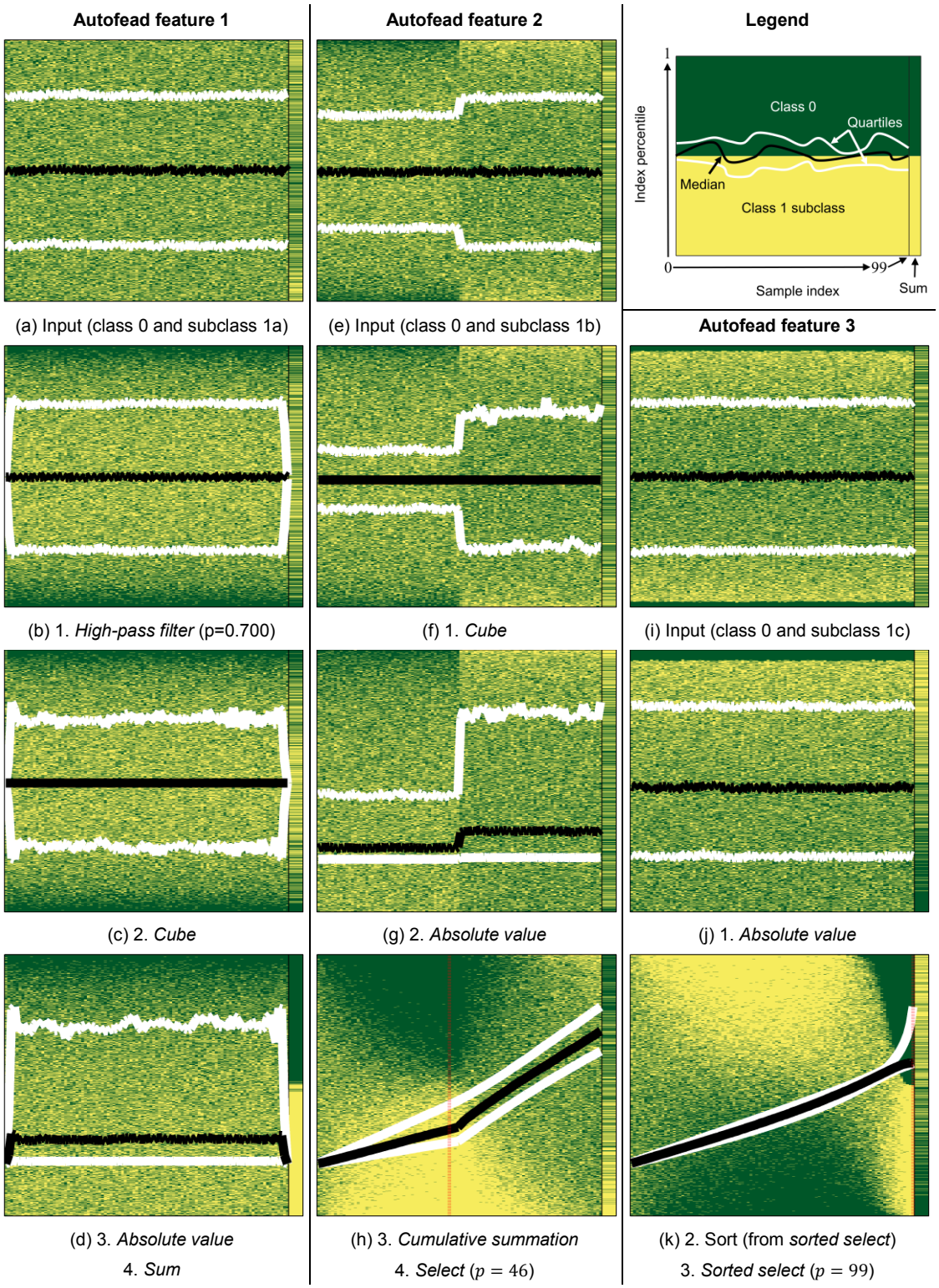


Figure 6 – DSD solution processing steps

largely a single class suggests terminating the feature through the *sorted select* function. The image in Figure 6(k) after the sort operation alone clearly shows excellent class separation for the largest indices corresponding to the maximum values in the unsorted sequences.

The feature distributions for the optimal solution and Autofead solution are given in Figure 7 conditioned on class 0 and the three subclasses of class 1. The arrows in Figure 7(c) and Figure 7(f) indicate impulse or near-impulse distributions. For each feature, one, and only one, subclass shows large separation from class 0 verifying that the problem requires at least three features to construct a near-optimal solution. In this case, the accuracy of the discovered solution is identical to the optimal solution to within 0.1%. The results for the DSD problem demonstrate that Autofead can design simple features and perform feature selection within its population.

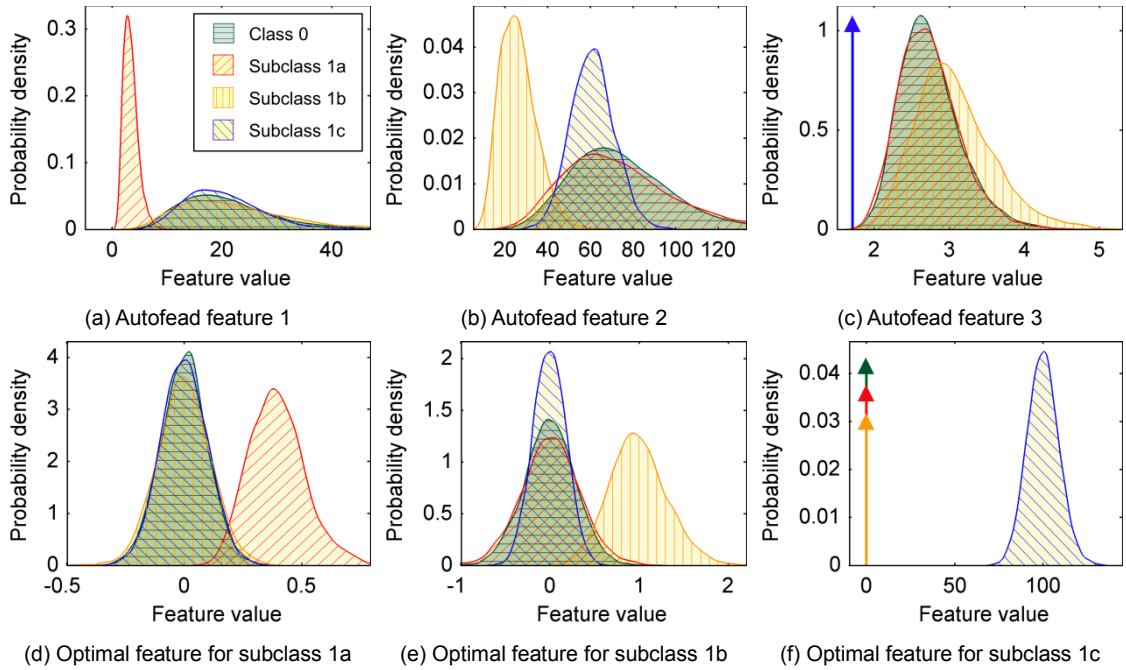


Figure 7 – DSD solution features' probability density function estimates

3.2. Signal Detection Problems

The second set of validation experiments employs two classic problems from detection theory, involving determining the presence of a sinusoidal signal obscured by WGN. The run configuration for these problems is provided by Table 5. Solutions were limited to a single feature in these experiments to reduce computational expense as the optimal solutions were known to only require a single feature.

For signal detection problem 1, class 0 contains noise alone, and class 1 includes a -10 dB signal-to-noise ratio sinusoid with random phase and frequency. The phase is uniformly distributed over $-\pi$ to π radians, and the frequency is uniformly distributed between $\pi/5$ and $2\pi/5$ rad/s. With 100 sample input signals, the frequency range endpoints correspond to the bin centers of FFT bins 10 and 20.

The optimal feature for signal detection problem 1 is the maximum magnitude of the FFT from bins 10 through 20. Four separate behaviors are necessary to compute the optimal feature. Low- and high-frequency content outside the signal frequency range must be removed.

Table 5 – Koza tableau for signal detection run configuration

Parameter	Setting
Objective	Derive optimal single feature for signal detection
Solution structure	Single feature consisting of a sequence of functions, omitted class mean signals
Function set	35 functions from function library plus separate function, *sum windows*; omitted *center, center and scale, convolve, cross-correlate, element difference, element product, element quotient, element sum, inverse, sigmoid, slope fit,* and *wavelet*
Pattern recognition	Kernel-based Naïve-Bayes classifier
Fitness	Minimize classification error bounded on [0,1]
Fitness case sampling	96% testing percentage split for each generation
Evolution strategy	Generational, population size of 500, no elitism
Solution size	Single feature limited to 15 functions and 8 parameters
Population initialization	Single feature ramped to initial maximum size of 5 functions and 3 parameters
Selection	Tournament, tournament size 4
Genetic operators	Crossover, 80%; mutate, 5%; reproduce, 5%; add feature, 5%; remove feature, 5%
Termination	20 generations; 10,000 individuals
Run repetitions	30; 300,000 total individuals

Next, the amount of sinusoidal energy present at each frequency is measured. Finally, the maximum energy level is selected as the feature. Figure 8 depicts four solutions found by Autofead that perform these behaviors in different ways. Feature A is the true optimal feature in minimal form. Feature B uses simple filtering operations to control the frequency content with a slight reduction in fitness due to the filter roll-off characteristics in the filtering functions. Function C utilizes an interesting pattern to select the maximum over the first 20 FFT bins with the low-frequency content removed by high-pass filtering. Lastly, feature D extracts FFT bins 10 to 20 using the *sliding windows* function resulting in optimal fitness. Here, the construction of the third of 9, 11-sample subsequence windows corresponding exactly with indices 10 through 20 is purely coincidental. With a different range of signal frequencies, the same feature would have suffered a small performance loss by missing useful, or retaining unneeded, FFT bins.

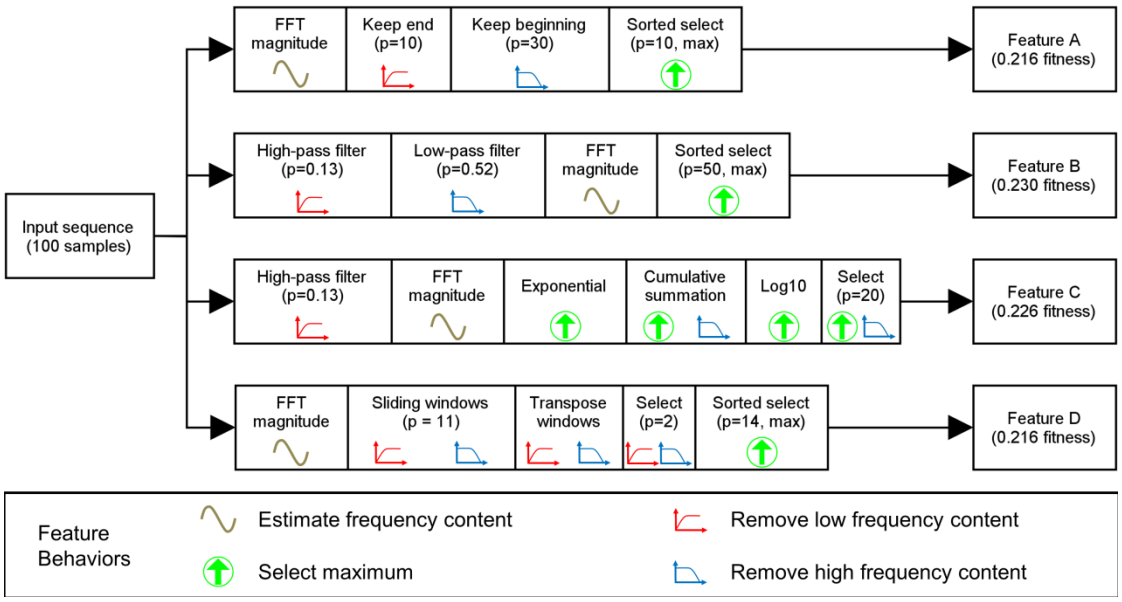


Figure 8 – Signal detection problem 1 solution feature algorithms

Signal detection problem 1 represents a significantly more challenging feature algorithm design problem than the DSD problem. Here, the optimal solution requires at least four functions including three parameters. Autofead again performed excellently finding a variety of optimal and near-optimal features. Furthermore, the search discovered unconventional function patterns to effectively perform needed behaviors such as in feature C. For all of these features, the optimized parameter values selected in the optimization scheme represented the true, global optimal values.

Signal detection problem 2 adds an additional level of complexity to the previous experiment by obscuring the sinusoidal signal in a longer WGN sequence with a random signal arrival time. Each signal contains 500 samples with class 1 including a 100-sample long sinusoid with starting index uniformly distributed over sample range 0 to 399. The optimal solution requires calculation of the optimal feature from signal detection problem 1 for every 100-sample subsequence in the time series then selecting the maximum value. The *sliding windows* function is designed to permit this type of behavior; however, the restrictions on number of windows and overlap between sequential windows make the optimal solution unrealizable in the solution space. The best Autofead solution as-found is shown in Figure 9 along with the optimal solution utilizing an *all sliding windows* function not present in the function library. The classification accuracy of the as-found solution is 85.8% compared to the optimal level of 90.8%, and the solution is accomplished with only three parameters as opposed to five in the optimal feature.

Although Autofead did not find an optimal-fitness solution, the as-found feature is sufficient as a starting point for manual refinements to reach the optimal feature. Figure 9 shows three refinement steps based on understanding of the behaviors within the as-found feature.

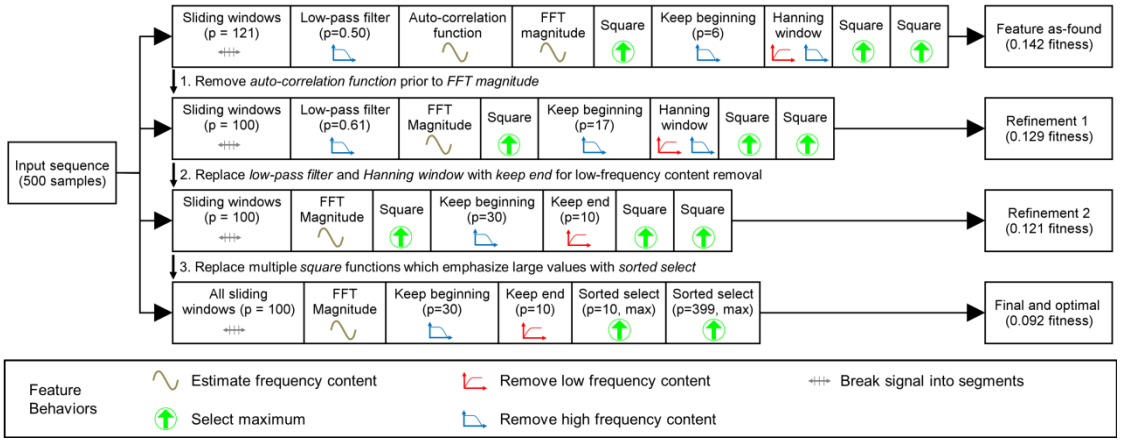


Figure 9 – Signal detection problem 2 solution feature algorithms

First, the *auto-correlation function* prior to an FFT operation is removed as it only acts as a triangular windowing process due to the biased estimation process. For refinement 2, it is recognized that the *Hanning window* and *keep beginning* functions act together to weight a range of sequence indices in the summation. Replacing *Hanning window* with *keep end* creates a rectangular window sized by the parameter optimization process. Lastly, repeated power functions and exponentials before a summation tend to emphasize the largest values, so we replace the three *square* functions with *sorted select* resulting in the final and optimal feature. Therefore, the Autofead as-found solution, while useful on its own, is best treated as a starting point for a post-search refinement and improvement process. Currently, such refinement must be performed manually but could be automated.

Through the DSD and signal detection experiments, Autofead is shown to produce optimal and near-optimal solutions in substantially similar form to the known analytical solutions. Only training instances are provided to the method to achieve these results whereas derivation of the optimal solution requires complete system knowledge. These experiments validate the basic solution space design and hybrid search method utilized by Autofead.

A portion of this chapter has been published in IEEE Transactions on Evolutionary Computation, Dustin Harvey and Michael Todd, 2014. The title of this paper is “Automated Feature Design for Numeric Sequence Classification by Genetic Programming”. The dissertation author was the primary investigator and author of this paper.

Chapter 4
Application to Structural Health Monitoring

4.1. Introduction to Structural Health Monitoring

Structural health monitoring (SHM) systems provide in-situ damage and performance information for civil, aerospace, and other high-capital value or safety-critical structures. The conventional processing flow for data-driven SHM systems begins with acquisition of time series structural response measurements. After pre-processing, a set of one or more damage-sensitive features is calculated in a feature extraction process that is then input to a pattern recognition algorithm to perform the desired task. The classification or regression analysis forms the basis for decision-making under the uncertainty and noise that typically affect the SHM process. Summary reviews on the field are provided in Doebling, Farrar, and Prime [46] and Farrar, Doebling, and Nix [47]. For the interested reader, Farrar and Worden provide a more comprehensive treatment of the field of SHM in their recent work [48].

Typical structural response measurements used by SHM systems consist of time series or spectral measurements of acceleration, velocity, force, or strain. The desired output includes binary detection of damage, classification of damage type, and estimation of damage location

and extent. As such, the field can be viewed as a specialized application of time series classification. In conventional model-based SHM, knowledge of the structure, loading, and operating conditions drives the development and verification of a model, which can be data-based or physics-based [48]. Parameters of the model or error measures within the modeling and prediction processes are used as features. Feature-based approaches are also common for SHM with feature algorithm design based on engineering judgment and domain knowledge. Little work has applied general time series classification methods that rely on training data alone to SHM problems.

4.2. SHM Feature Extraction

Development of effective features for a specific SHM system presents two key difficulties. First, the features must be sensitive to the damage states of interest in the system such that statistically detectable changes occur when the system undergoes damage. Second, the features must be insensitive to the varying operational or environmental conditions the system may experience in use. SHM axiom IVb states that "without intelligent feature extraction, the more sensitive a measurement is to damage, the more sensitive it is to changing operational and environmental conditions" [49]. The design of robust, damage-sensitive features for a specific application can be an expensive and time-consuming task requiring extensive system knowledge and domain expertise or, absent that, substantial trial-and-error.

The objectives of a fully data-driven approach to SHM feature design are thus two-fold. First, the question is posed "How should we process response measurements to extract damage information when minimal system and domain knowledge is available?" An automated, data-driven approach to feature design directly infers from a training database of response measurements a feature set specific to the problem at hand. Such an automated approach

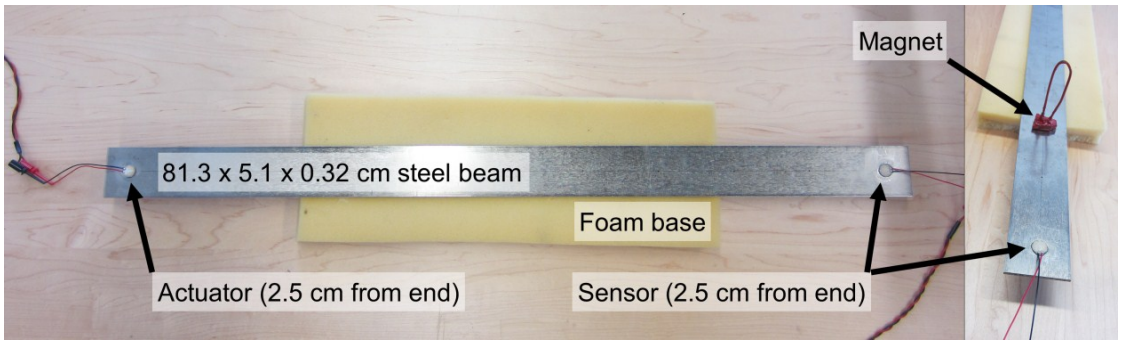
provides the additional benefit of significantly reduced time and effort for the overall SHM system design process. Second, the additional goal of designing robust features may be met directly by including examples of the expected operational and environmental variability within the training data. Thus, the fundamental assumption in this approach is that data are available that are known to span the desired classification or regression spaces of the SHM system.

Autofead provides a feature-based time series classification solution with no required knowledge of the data-generating system. This approach is widely applicable to SHM problems involving complex structures or failure modes that are difficult to model or to reduce the required development time and effort for an SHM system. The remainder of this chapter demonstrates direct application of Autofead to three SHM-related laboratory experiments.

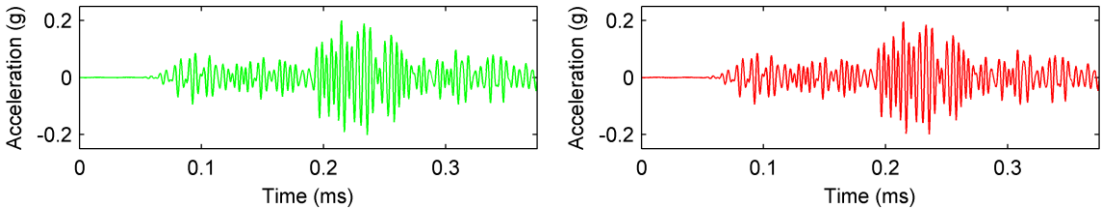
4.3. Ultrasonic Damage Detection

For the first experiment, the goal is to determine the presence of damage in a steel beam simulated by a magnet using a single pitch-catch measurement from a pair of piezoelectric elements. The experimental structure is shown in Figure 10. The training data contains 500 undamaged instances with no magnet and 500 damaged instances with the magnet located at 40 different locations on the same side of the beam as the actuator and sensor. Recorded signals include 750 samples collected at 2 MHz as seen in the examples in Figure 10. The actuation signal is a 200 kHz Gaussian-modulated sine wave with 0.6 normalized bandwidth.

Conventional pre-processing operations for this task include matched filtering, envelope analysis, and baseline subtraction. The *convolve* function is configured here to perform convolution with the actuation signal for matched filtering. An envelope analysis can be carried



(a) Experiment photos



(b) Undamaged (without magnet) example time series (c) Damaged (with magnet) example time series

Figure 10 – Ultrasonic damage detection experiment

out by the *Hilbert magnitude* function. Additionally, the undamaged measurements are averaged to provide a healthy baseline input. By including a baseline, it is possible for Autofead to construct features including a baseline subtraction component through the *element difference* merging function. The use of the actuation signal for convolution and inclusion of an average baseline channel represent examples of how domain knowledge can be incorporated into Autofead to tailor the method for a specific problem.

For comparison solutions, waveforms are pre-processed by matched filtering and envelope analysis; then total energy and peak amplitude features with and without baseline subtraction are computed. Interested readers are directed to Farrar and Worden [48], Raghavan and Cesnik [50], and Flynn, Todd, Wilcox, Drinkwater, and Croxford [51] for an introduction to the use of ultrasonics and guided-waves in SHM. The Autofead solution was found using the configuration in Table 6.

Table 6 – Koza tableau for ultrasonic damage detection run configuration

Parameter	Setting
Objective	Design optimal feature set to detect presence of simulate damage by magnet
Solution structure	Set of features consisting of a sequence of functions, omitted class mean signals
Function set	44 functions from function library; omitted *bisection select, convolve,* and *sorted select*
Pattern recognition	Kernel-based Naïve-Bayes classifier
Fitness	Minimize classification error bounded on [0,1]
Fitness case sampling	k-fold cross-validation
Evolution strategy	$(\mu/\rho+\lambda)$ with $\mu = 1{,}000$, $\rho = 1{,}000$, and $\lambda = 50$
Solution size	Maximum size of 5 features each limited to 15 functions and 8 parameters
Population initialization	Ramped to initial maximum size of 3 features each limited to 5 functions and 3 parameters
Selection	Tournament, tournament size 2
Genetic operators	Crossover, 80%; mutate, 5%; reproduce, 5%; add feature, 5%; remove feature, 5%
Termination	980 iterations; 50,000 individuals
Run repetitions	50; 2.5 million total individuals

The Autofead solution contains a single feature based on a cross-correlation between the measured waveform and average baseline. Figure 11 characterizes the original input data and how the data progresses through each processing step in the Autofead feature. See section 3.1 for a detailed interpretation of the solution processing images.

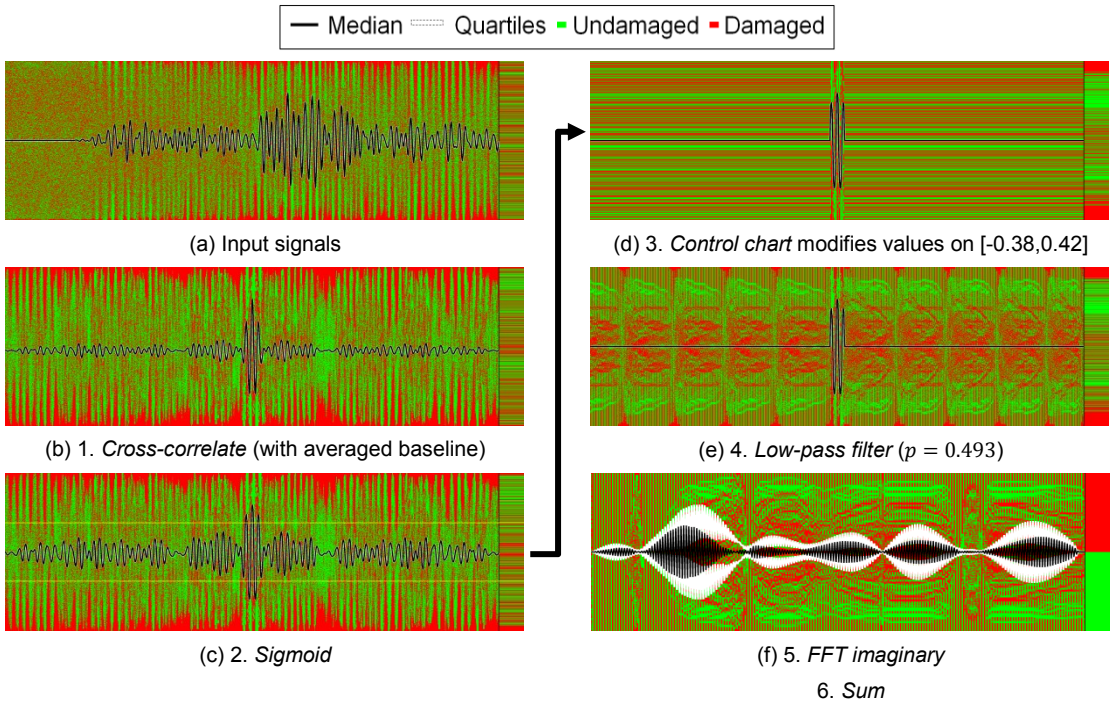


Figure 11 – Ultrasonic damage detection solution processing steps

Figure 12 shows class-conditioned histograms and probability density function (PDF) estimates for the Autofead feature and four conventional features. From the distributions, the Autofead feature clearly provides greater separation of the undamaged and damaged classes. The unusual distributions for the damaged class are primarily due to the inclusion of 40 different magnet locations with varying levels of detection difficulty. For example, if only a single damage location was included, one would expect peak amplitude without baseline subtraction to either be consistently smaller, larger, or identical with damage to the undamaged class. The receiver operating characteristics (ROC) curves in Figure 13 show the trade-off between false positives and correct detections using each feature from Figure 12. The ROC curves confirm that the Autofead feature outperforms the conventional solutions with nearly

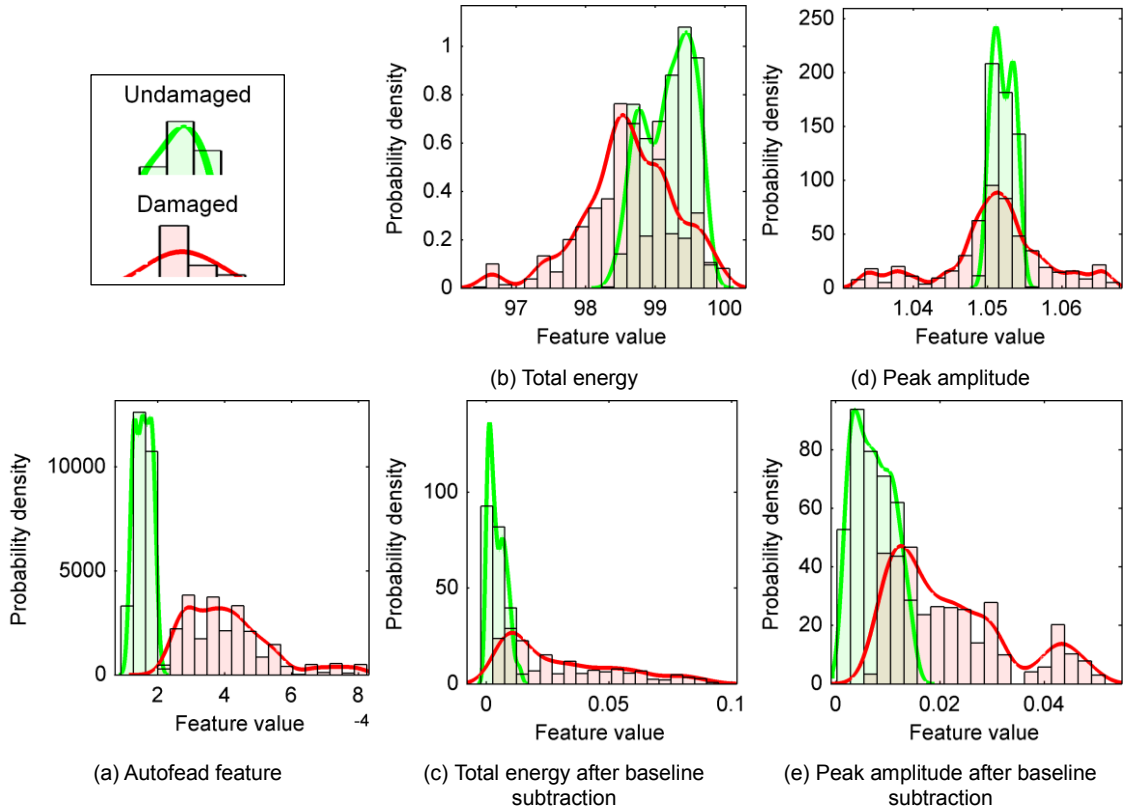


Figure 12 – Ultrasonic damage detection solutions' feature probability density function estimates

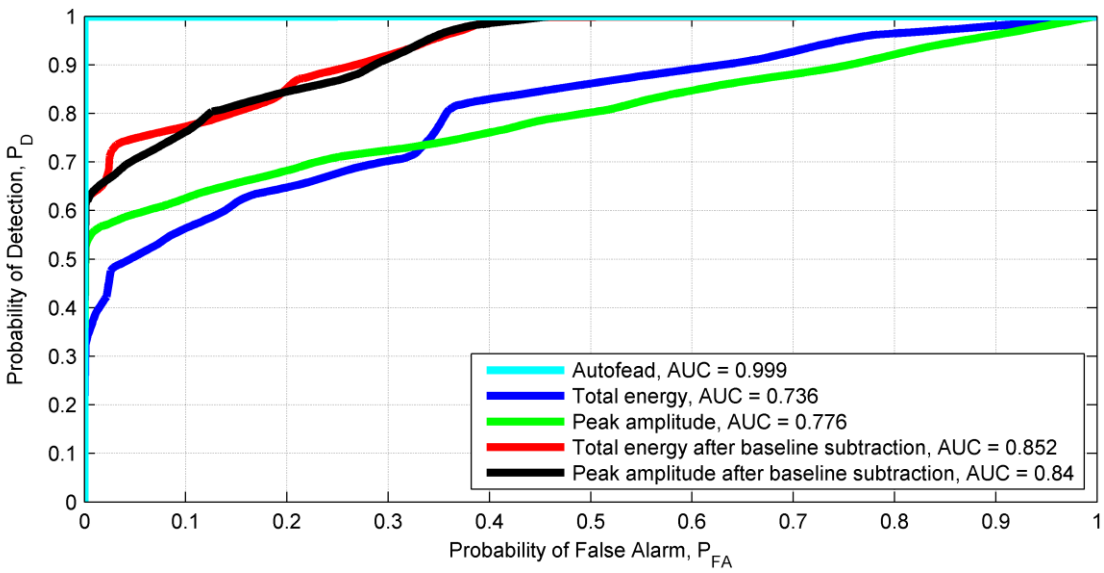


Figure 13 – Ultrasonic damage detection receiver operating characteristic (ROC) curves

perfect classification accuracy at 99.9%. However, it is important to note that the Autofead solution is designed to be specific to the training data. In this case, the experiment only includes a single beam, sensor pair, and damage type in a controlled laboratory environment. The comparison solutions may generalize better to variation of the conditions of the experiment as well as other ultrasonic structural interrogation applications. To design a general feature for these applications, any expected structural, operational, and environmental variability would need to be included in the training data input to Autofead.

4.4. Damage Type Identification for Rotating Machinery

The second experiment involves identifying three bearing health states in the Machinery Fault Simulator from Spectraquest, Inc. depicted in Figure 14. The motor drives a gearbox through a set of drive belts and a 91.4 cm shaft supported by three Rexnord ER12K ball bearings. Damage is introduced to the bearing farthest from the motor by replacing the healthy

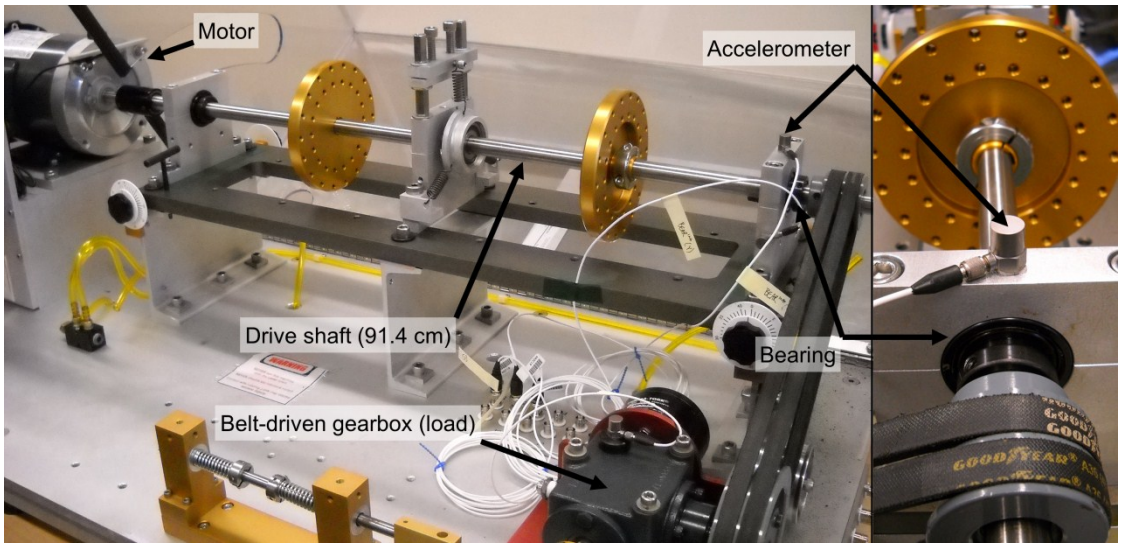


Figure 14 – Rotating machinery experiment photos

bearing with a pre-damaged specimen. The damage cases include a bearing with ball spalling and a bearing with an outer race fault.

Time series response measurements are collected from the accelerometer located on top of the housing of the bearing of interest. Each response measurement consists of 540 samples collected at a sampling rate of 2.56 kHz. All responses are collected at a steady-state shaft speed of 1,000 rpm (16.7 Hz). Figure 15 shows example measurements for each class.

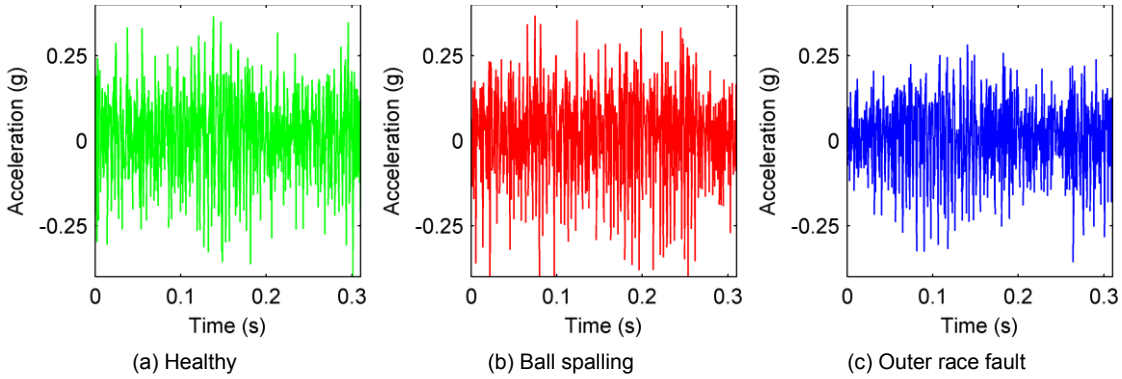


Figure 15 – Rotating machinery example time series

Table 7 – Koza tableau for rotating machinery run configuration

Parameter	Setting
Objective	Design optimal feature set to detect presence of damaged bearings
Solution structure	Set of features consisting of a sequence of functions, omitted class mean signals
Function set	45 functions from function library; omitted *bisection select* and *sorted select*
Pattern recognition	Kernel-based Naïve-Bayes classifier
Fitness	Minimize classification error bounded on [0,1]
Fitness case sampling	k-fold cross-validation
Evolution strategy	$(\mu/\rho + \lambda)$ with $\mu = 1{,}000$, $\rho = 1{,}000$, and $\lambda = 50$
Solution size	Maximum size of 5 features each limited to 15 functions and 8 parameters
Population initialization	Ramped to initial maximum size of 3 features each limited to 5 functions and 3 parameters
Selection	Tournament, tournament size 2
Genetic operators	Crossover, 80%; mutate, 5%; reproduce, 5%; add feature, 5%; remove feature, 5%
Termination	980 iterations; 50,000 individuals
Run repetitions	50; 2.5 million total individuals

The loading of the gearbox through the drive belts and the presence of an outer race fault on one bearing specimen produce an asymmetry to the system along the shaft axis. Therefore, the system is disassembled and reassembled 8 times per bearing specimen to mitigate experimental errors. 5,120 instances are collected for each of the three damage conditions. Autofead's training data includes twenty-five percent of the instances with the rest held out for independent solution testing. The selected run parameters are listed in Table 7.

Conventional solutions for comparison are selected from ten metrics presented in the review of vibration analysis methods for rotating machinery by Lebold, McClintic, Campbell, Byington, and Maynard [52]. Calculation of many of the metrics requires knowledge of the internal geometry of components such as bearings and gearboxes to identify fundamental frequencies for various filtering operations. Of the ten metrics, FM4 and RMS provide the best performing pair of features for use with the classifier used by Autofead solutions.

Figure 16 shows the signal processing flow for the two features in the Autofead solution. The first feature is approximately the median value after pre-processing by a non-linear scaling operation, high-pass filtering, and lastly an envelope analysis. This feature is difficult to interpret due to the untraditional pre-processing sequence. Feature 2 is simply the

energy within a narrow frequency band after the *sigmoid* operation, which primarily reduces the effect of outliers. Both features contain parameters that are optimized to remove lower-frequency components of the response.

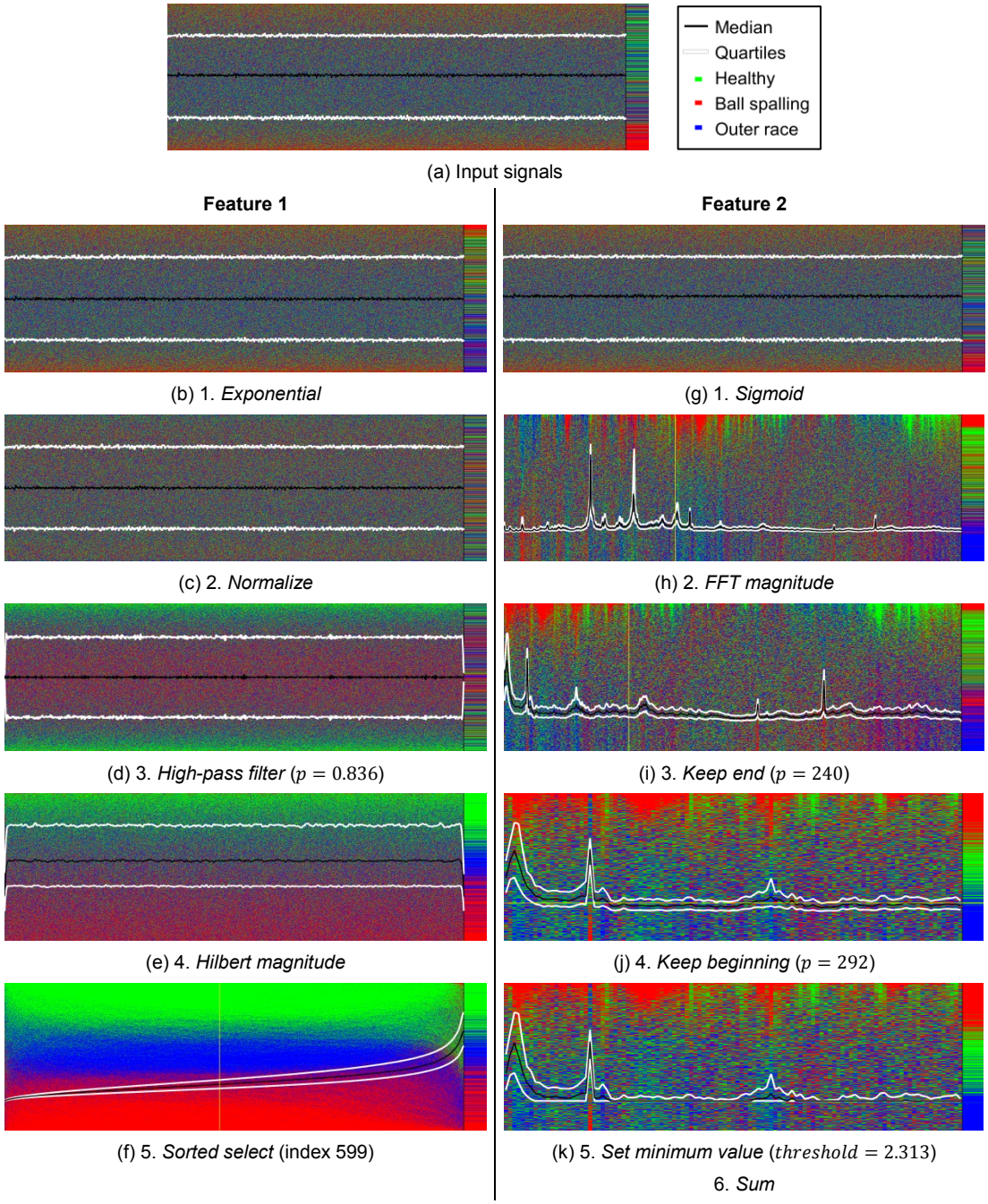


Figure 16 – Rotating machinery solution processing steps

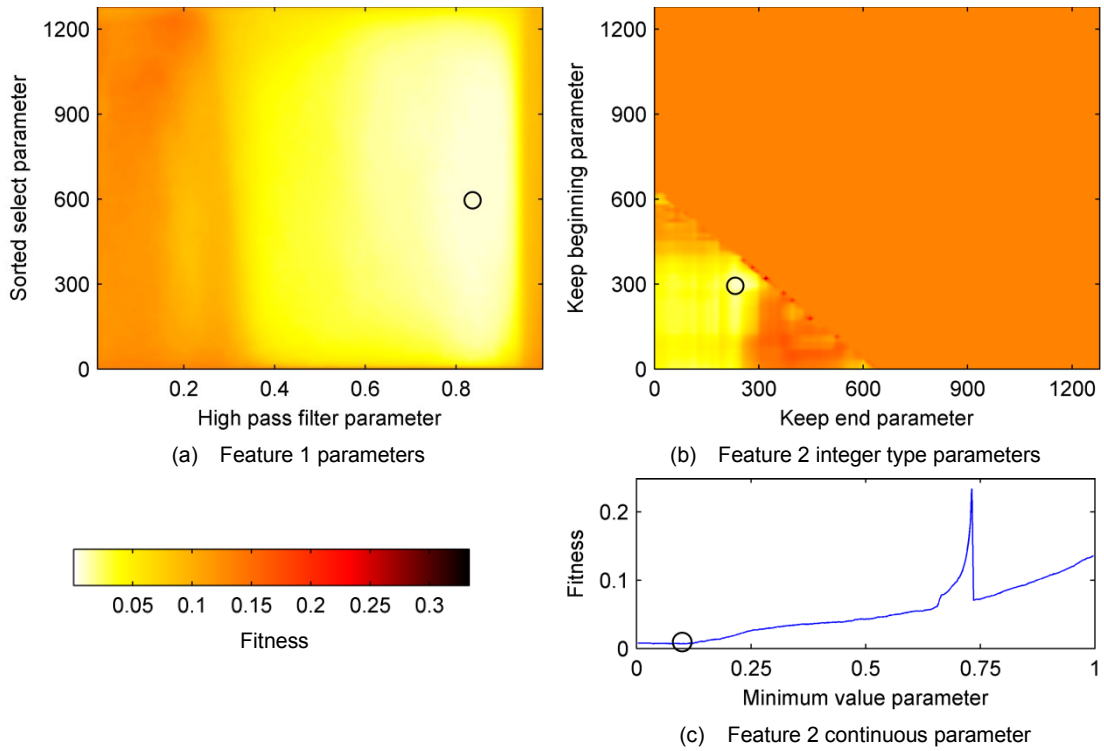


Figure 17 – Rotating machinery solution parameter spaces

Figure 17 presents the parameter spaces for the two parameters in feature 1 and three parameters in feature 2. In each image, fitness levels are computed with all other parameters in the individual held to their final optimized values indicated by the black circles. These surfaces show some of the complexity and variety of parameter optimization problems encountered within the Autofead solution space. The parameter space for feature 1 is fairly smooth and unimodal, while the space for the integer parameters in feature 2 is dominated by a large low-fitness region. The small region of higher fitness is fairly rough containing many local extrema. Finally, the surface for the *set minimum value* parameter in feature 2 includes a significant discontinuity around $p = 0.75$. In each case, Autofead's parameter optimization scheme provides results near the global optimum.

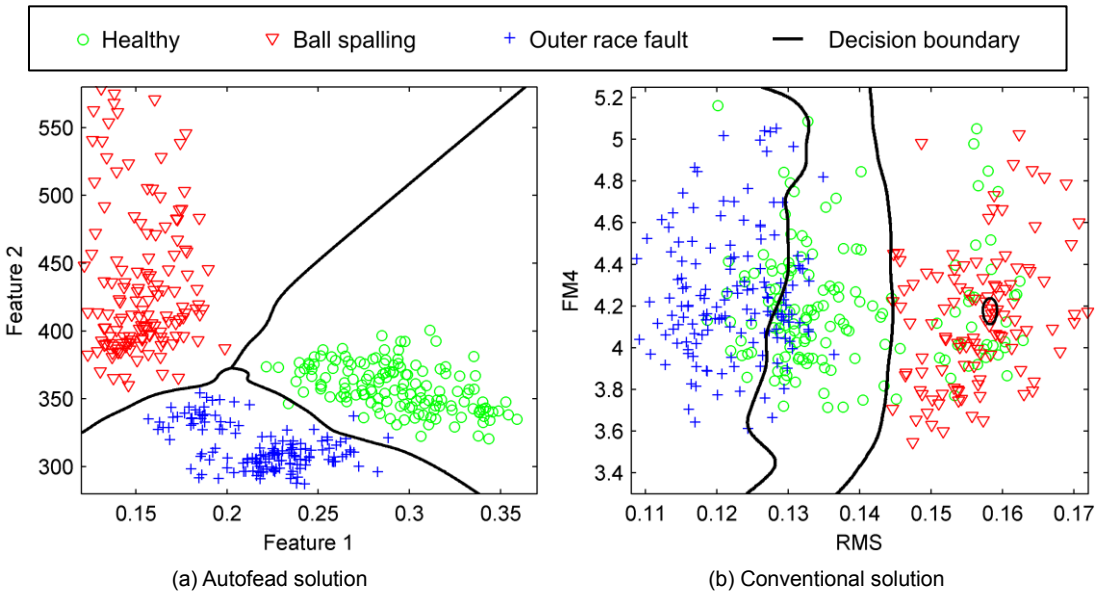


Figure 18 – Rotating machinery solution feature space

The two-dimensional feature spaces for the Autofead solution and conventional metrics are given in Figure 18 including decision boundaries from the Naïve-Bayes classifier. The conventional solution relies primarily on the RMS metric to separate the classes and achieve classification accuracy of 78%. In comparison, the two features in the Autofead solution complement each other to provide excellent separation between all three classes resulting in 99% classification accuracy.

4.5. Vibration-Based Damage Extent Estimation

The final experiment uses the bolted, aluminum bookshelf structure in Figure 19 with the goal of estimating damage extent from vibration-based response measurements. The structure is composed of four, 2.5 cm thick aluminum plates measuring 30.5 cm wide by 30.5 cm deep. The plates are supported by rectangular columns at each corner for a total structure height of 53.1 cm. The entire structure is mounted on a rail system to constrain the motion to a

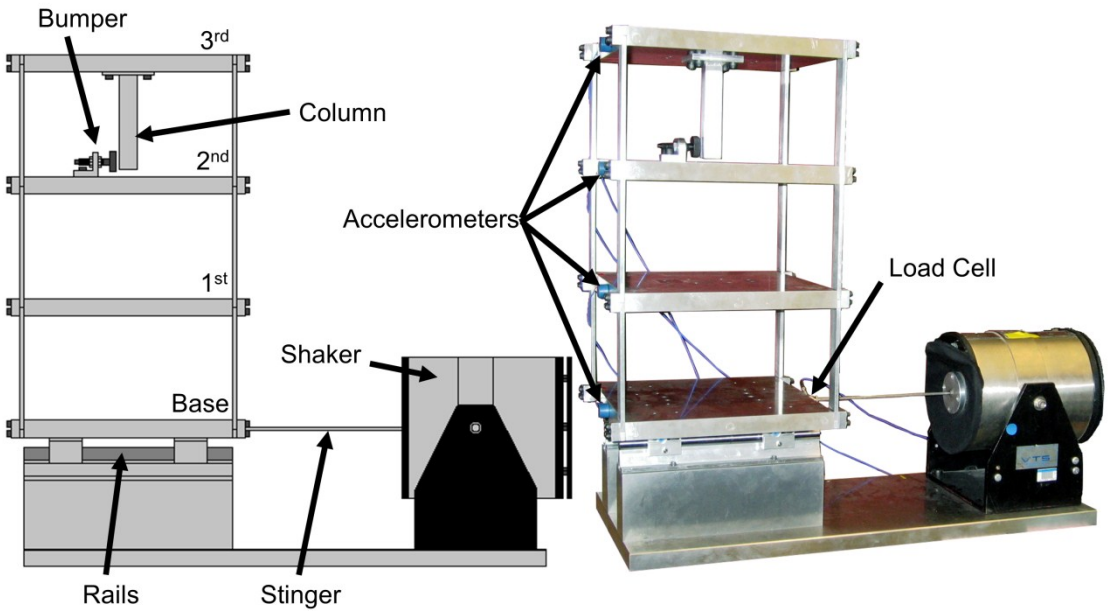


Figure 19 – Damage extent estimation experiment photos

single primary direction. An electrodynamic shaker provides excitation along the midline of the bottom floor though a stinger and load cell. Responses are measured by accelerometers mounted to the midlines of each floor in the primary direction of motion.

Damage is introduced by changing the width of the gap in a bumper and column system mounted on the second and third floors, respectively. Relative motion of the two floors causes impacts between the bumper and column. When the width of the gap is small, more impacts occur for a given excitation representing a higher level of damage. Six damage levels are included in the training data with the task of estimating the damage level using linear regression from the four response measurements and the input measurement from the load cell. Damage level 0 represents the healthy state with a wide enough gap such that no impacts occur. Damage levels 1, 1.33, 1.54, 2, and 4 are proportional to their respective gap widths of 0.20, 0.15, 0.13, 0.10, and 0.05 mm.

Table 8 – Koza tableau for damage extent estimation run configuration

Parameter	Setting
Objective	Design optimal feature set to estimate extent of simulated damage from bumper impacts with column
Solution structure	Set of features consisting of a sequence of functions
Function set	44 functions from function library; omitted *bisection select, convole,* and *sorted select*
Pattern recognition	Linear regression
Fitness	Minimize RMS error [0,1]
Fitness case sampling	k-fold cross-validation
Evolution strategy	$(\mu/\rho + \lambda)$ with $\mu = 1{,}000$, $\rho = 1{,}000$, and $\lambda = 50$
Solution size	Maximum size of 5 features each limited to 15 functions and 8 parameters
Population initialization	Ramped to initial maximum size of 3 features each limited to 5 functions and 3 parameters
Selection	Tournament, tournament size 2
Genetic operators	Crossover, 80%; mutate, 5%; reproduce, 5%; add feature, 5%; remove feature, 5%
Termination	980 iterations; 50,000 individuals
Run repetitions	50; 2.5 million total individuals

The structure is excited by band-limited white noise from 20-150 Hz to include the first three natural frequencies and avoid low-frequency, rigid-body modes. Response measurements are collected at 320 Hz for 3.2 seconds (1,024 samples). Two hundred fitness cases are included in the training data from each of the six damage levels. Full details of the structure, data acquisition system, and test plan are found in Figueiredo, Park, Figueiras, Farrar, and Worden [53]. The Autofead configuration for this multivariate regression task is given by Table 8.

A multitude of possible solutions exist in the literature for vibration-based damage detection and localization such as those in Farrar and Worden [48]. For this example, conventional solutions were selected from the methods compared in Figueiredo et al. [53]. The statistical moments solution includes the mean, variance, skewness, and kurtosis for each of the four response channels for a total of 16 features. Skewness is a particularly useful feature for this problem as the simulated damage case adds asymmetry to an initially symmetric system. The natural frequencies solution includes estimates of the first three natural frequencies of the structure through the frequency response and complex mode indicator functions. Two solutions are based on autoregressive (AR) time series modeling. The AR(5) parameters solution uses the parameters from a fifth-order model as features. A separate model is fit for each response

channel to generate 20 features. Lastly, AR(20) RMS error includes the root mean square error level for each response channel using a 20-order model fit to responses from damage level 0. For this solution, the RMS error is expected to increase as the damage level increases creating a system progressively less similar to the healthy state.

The Autofead solution uses three features as depicted in Figure 20. For visualization purposes, the processing images are displayed with only damage levels 0, 1, and 4. Features 1 and 2 use the second floor response while feature 3 uses a cross-correlation between the top two floors. While these features are difficult to fully interpret due to highly non-linear behavior of some of the element operations, certain characteristics of the data are clear. For instance, the higher damage levels contain more high-frequency content. Unsurprisingly, all three features utilize the response measurements from the floors where the bumper and column are mounted.

Investigation of the three-dimensional feature space formed by the Autofead solution reveals the complementary nature of the three features. Figure 21 depicts a three-dimensional scatter plot with PDF estimates along each feature axis conditioned to the six damage levels. Features 1 and 3 both increase monotonically with the damage level but cannot separate damage levels 0 and 1; however, feature 2 provides the additional information needed to separate the lowest two damage levels. Although the classes are fairly well-separated in the feature space, it is clear that a linear regression model is insufficient to fully utilize the information in the Autofead solution. This is an example where the simple pattern recognition algorithm is sufficient within Autofead to perform feature design, but the final solution would benefit from selection of a more advanced regressor.

To evaluate the relative performance of the Autofead solution and four comparison solutions, distributions of the damage level estimates using linear regression are shown in

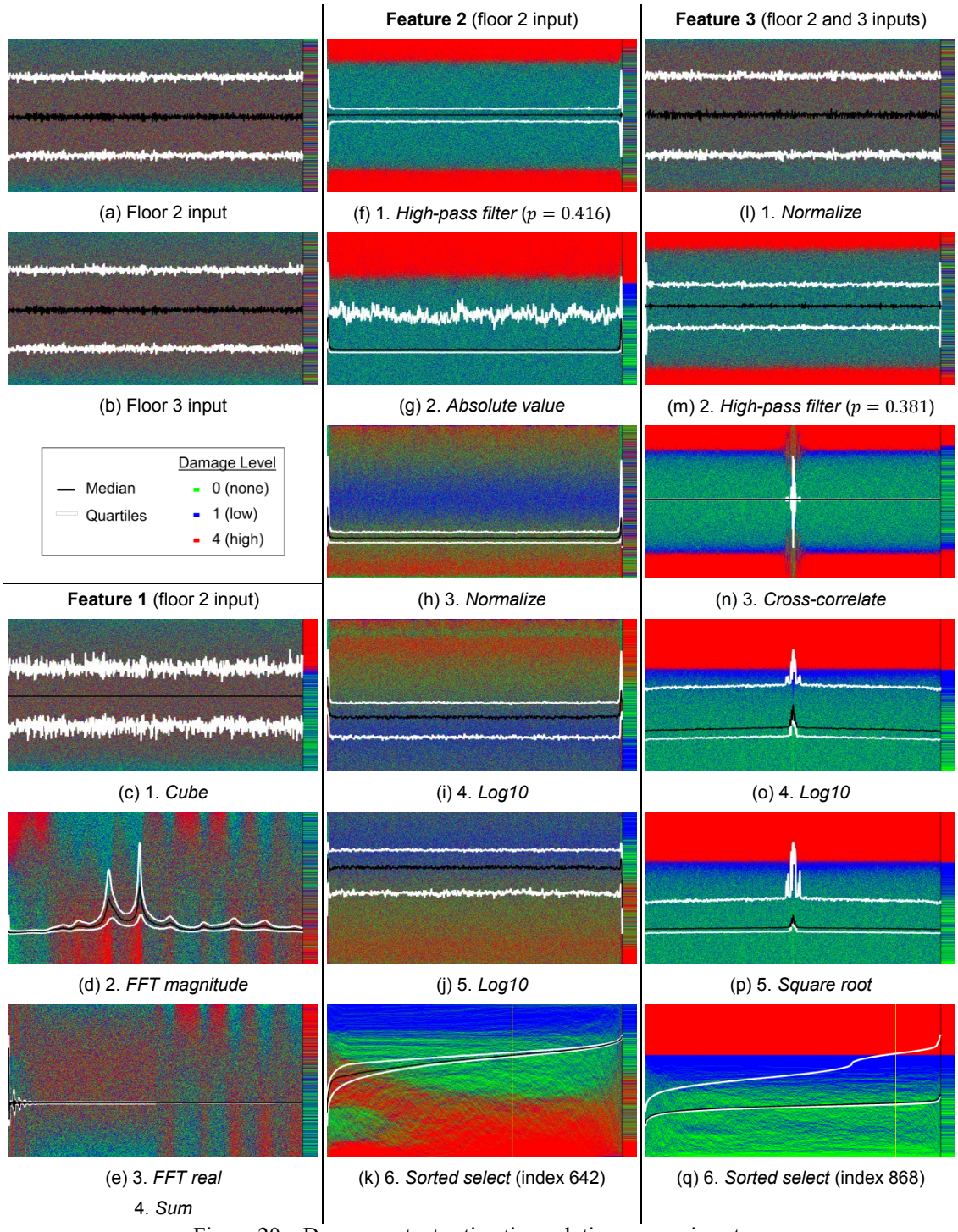


Figure 20 – Damage extent estimation solution processing steps

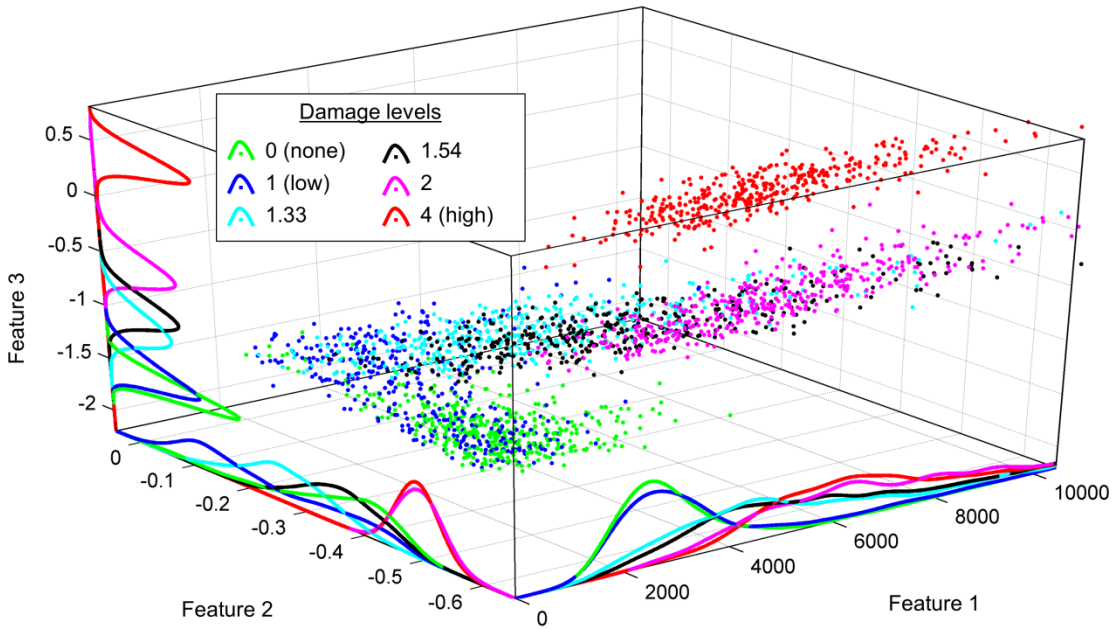


Figure 21 – Damage extent estimation solution features' probability density function estimates and feature space

Figure 22 along with the true damage levels indicated by vertical, dashed lines. Clearly, the Autofead solution and AR(5) parameters provide the most accurate and consistent estimates across the damage levels. Damage level 1 is an interesting case where bimodal behavior is observed for some of the solutions with one mode overlapping damage level 0. This result is most likely due to a mislabeling in the training data in cases where the gap width was set correctly to 0.20 mm but no actual impacts occurred within the measurement time. However, without an independent measure of impact counts, this hypothesis cannot be confirmed. If true, the Autofead solution significantly outperforms the other solutions in separating the instances from damage level 1 where impacts do and do not occur.

Table 9 provides the RMS error for the five solutions with the two best solutions at each damage level shown in bold. The results for damage level 1 do not follow the trends of the other levels due to the previously discussed bimodal behavior issue. For all other levels, as well as on

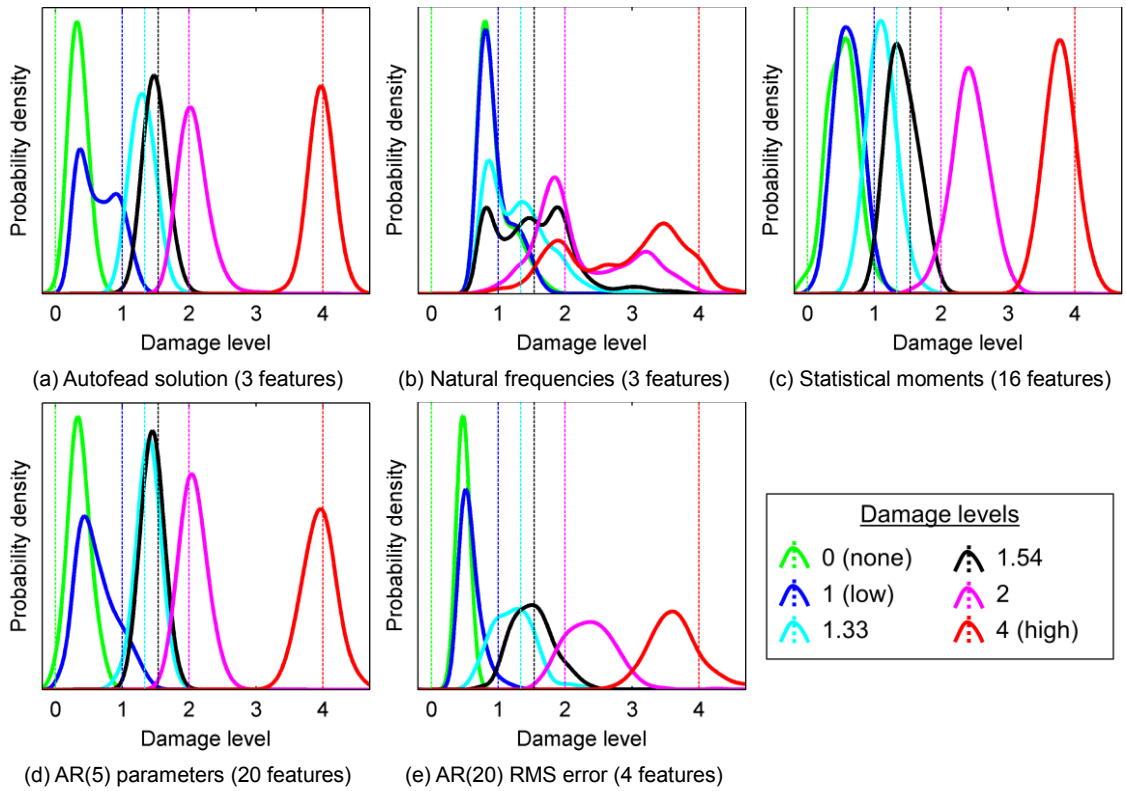


Figure 22 – Damage extent estimation solution accuracies

average, the Autofead solution provides the lowest or second-lowest error level. While the error is similar for the next best solution of AR(5) parameters, the dimension reduction provided by Autofead is significant using only 3 features compared to 20.

Table 9 – Damage extent estimation solution RMS errors

Damage level	Autofead	Natural frequencies	Statistical moments	AR(5) parameters	AR(20) RMS error
0	**0.36**	0.98	0.56	**0.39**	0.47
1	0.45	**0.24**	**0.44**	0.48	0.45
1.33	**0.16**	0.44	0.30	**0.16**	0.34
1.54	**0.15**	0.58	0.24	**0.16**	0.30
2	**0.21**	0.78	0.50	**0.19**	0.53
4	**0.14**	1.41	0.30	**0.23**	0.52
Average	**0.25**	0.74	0.39	**0.27**	0.44

This chapter experimentally demonstrates the Autofead method on a variety of SHM problems including binary damage detection, classification of damage type for rotating machinery, and multivariate regression of damage extent. Significant performance improvements over conventional features are realized utilizing only supervised examples. The results show Autofead as a promising method for SHM feature design in applications where sufficient training data can be acquired.

A portion of this chapter has been published in Smart Materials and Structures, Dustin Harvey and Michael Todd, 2014. The title of this paper is “Structural Health Monitoring Feature Design by Genetic Programming”. The dissertation author was the primary investigator and author of this paper.

Chapter 5
Benchmark Study

5.1. Problem Set

This chapter provides a direct comparison between Autofead and state-of-the-art TSC methods. Openly available data sets allow for evaluation and comparison of methods over a wide range of application areas. The largest repository is hosted at the *UCR Time Series Classification/Clustering Page* [54]. This database is used by many TSC researchers providing direct, unbiased comparison of results using the predefined training and testing splits in the data. For this study, 43 problems in the UCR database were supplemented with an additional 6 problems provided by Bagnall et al. [12].

5.1.1. Characterization

The 49 problems selected for this study represent a wide range of data types and problem dimensions. Table 10 provides characteristics for each of the 49 problems studied including data type, data source, number of classes, instance counts, and the number of samples in the input data. 25 problems consist of time series measurements from a variety of sensors.

Table 10 – Benchmark study problem characteristics

Data type	Problem	Source	Number of classes	Training instances	Testing instances	Time series length
Sensor	ChlorineConcentration	[54]	3	467	3840	166
	CinC_ECG_torso	[54]	4	40	1380	1639
	Cricket_X	[54]	12	390	390	300
	Cricket_Y	[54]	12	390	390	300
	Cricket_Z	[54]	12	390	390	300
	Earthquakes	[12]	2	322	139	512
	ECG200	[54]	2	100	100	96
	ECGFiveDays	[54]	2	23	861	136
	ElectricDevices	[12]	7	8953	7745	96
	FordA	[12]	2	3571	1320	500
	FordB	[12]	2	3601	810	500
	Gun-Point	[54]	2	50	150	150
	InlineSkate	[54]	7	100	550	1882
	ItalyPowerDemand	[54]	2	67	1029	24
	Lightning2	[54]	2	60	61	637
	Lightning7	[54]	7	70	73	319
	Motes	[54]	2	20	1252	84
	SonyAIBORobotSurface	[54]	2	20	601	70
	SonyAIBORobotSurfaceII	[54]	2	27	953	65
	StarLightCurves	[54]	3	1000	8236	1024
	TwoLeadECG	[54]	2	23	1139	82
	uWaveGestureLibrary_X	[54]	8	896	3582	315
	uWaveGestureLibrary_Y	[54]	8	896	3582	315
	uWaveGestureLibrary_Z	[54]	8	896	3582	315
	Wafer	[54]	2	1000	6174	152
Shape representation	50Words	[54]	50	450	455	270
	Adiac	[54]	37	390	391	176
	DiatomSizeReduction	[54]	4	16	306	345
	FaceAll	[54]	14	560	1690	131
	FaceFour	[54]	4	24	88	350
	FacesUCR	[54]	14	200	2050	131
	Fish	[54]	7	175	175	463
	HandOutlines	[12]	2	1000	300	2709
	Haptics	[54]	5	155	308	1092
	OSULeaf	[54]	6	200	242	427
	SwedishLeaf	[54]	15	500	625	128
	Symbols	[54]	6	25	995	398
	WordsSynonyms	[54]	25	267	638	270
	Yoga	[54]	2	300	3000	426
Synthetic	ARSim	[12]	2	2000	2000	500
	CBF	[54]	3	30	900	128
	MALLAT	[54]	8	55	2345	1024
	Synthetic_Control	[54]	6	300	300	60
	Trace	[54]	4	100	100	275
	Two_Patterns	[54]	4	1000	4000	128
Spectral	Beef	[54]	5	30	30	470
	Coffee	[54]	2	28	28	286
	OliveOil	[54]	4	30	30	570
Histogram	MedicalImages	[54]	10	381	760	99

One-dimensional shape representations account for 14 problems. Six problems are synthetically generated to test specific aspects of TSC methods, and the last four contain spectral and histogram data. The authors note that the Motes data set is sometimes referred to as MoteStrain in other works.

5.1.2. Problem Descriptions

This section provides a brief explanation of the type of data and classes in each problem. In some cases, insufficient information was available to clearly state the nature of the data and classes. Problems are listed alphabetically with problem names emboldened.

50Words: 50 handwritten words from shape profiles

Adiac: 37 taxa of single-celled alga from curvatures extracted from binary images

ARSim: 2 distinct AutoRegressive processes designed specifically to be difficult to separate in the time domain

Beef: 5 degrees of contamination in beef from spectrographs

CBF: 3 synthetic waveform shapes scaled, stretched, and shifted with random additive noise

ChlorineConcentration: 3 classes related to the concentration of chlorine in a simulated water distribution system from time series of chlorine levels at pipe junctions

CinC_ECG_torso: 4 patients from electrocardiogram measurements at various torso-surface sites

Coffee: 2 classes of coffee (Arabica or Robusta) from spectrographs

Cricket_X: 12 cricket umpire gestures from x-axis accelerometer measurements in a wristband

Cricket_Y: 12 cricket umpire gestures from y-axis accelerometer measurements in a wristband

Cricket_Z: 12 cricket umpire gestures from z-axis accelerometer measurements in a wristband

DiatomSizeReduction: 4 taxa of single-celled alga from shape representations

Earthquakes: Binary prediction of impending major earthquake event in California from segments of one-hour averages of seismic activity

ECG200: Binary abnormality detection in a single heartbeat of an electrocardiogram

ECGFiveDays: 2 dates five days apart from single heartbeats of an electrocardiogram of a 67 year old male

ElectricDevices: 7 groups of household devices from electrical demand profiles

FaceAll: Similar to FaceFour and FacesUCR

FaceFour: 4 people from shape representations of head outlines from side

FacesUCR: 14 people from shape representations of head outlines from side

Fish: 7 species of fish from shape representations

FordA: Binary detection of a symptom in an automotive subsystem from acoustic measurements of engine noise collected in typical operating conditions

FordB: Binary detection of a symptom in an automotive subsystem from acoustic measurements of engine noise with training data collected in typical operating conditions and test data collected under noisy conditions

Gun-Point: 2 distinct actions performed by actors from x-axis record of hand motion from video surveillance camera

HandOutlines: Binary detection of hand segmentation accuracy in an image from shape representation

Haptics: 5 shapes drawn on a touchscreen from trace of x-axis position alone

InlineSkate: 7 speeds of professional inline speed skaters on a treadmill from angular measurement of ankle movement

ItalyPowerDemand: 2 ranges of dates from power demand records in Italy

Lightning2: 2 categories of lightning strikes from power density profiles measured from satellite instruments

Lightning7: 7 categories of lightning strikes from power density profiles measured from satellite instruments

MALLAT: 8 synthetic waveforms with simulated faults

MedicalImages: 10 different region of human body from histograms of pixel intensity in a medical image

Motes: 2 sensors in a laboratory network of Berkeley Mote sensors

OliveOil: 4 countries of origin from spectrographs of olive oil

OSULeaf: 6 tree species from shape representations of leaves

SonyAIBORobotSurface: 2 surfaces from x-axis accelerometer measurements of Sony AIBO robot motion

SonyAIBORobotSurfaceII: 2 surfaces from y-axis accelerometer measurements of Sony AIBO robot motion

StarLightCurves: 3 categories of stars from light intensity measurements

SwedishLeaf: 15 tree species from shape representations of leaves

Symbols: 6 classes representing 3 symbols drawn on a touchscreen using either x- or y-axis position traces

Synthetic_Control: 6 common control chart patterns from synthetic waveforms

Trace: 4 transient signal behaviors from synthetic waveforms

Two_Patterns: 4 combinations of upward and downward steps in synthetic waveforms

TwoLeadECG: 2 leads of electrocardiogram record

uWaveGestureLibrary_X: 8 gestures from x-axis of handheld device acceleration measurement

uWaveGestureLibrary_Y: 8 gestures from y-axis of handheld device acceleration measurement

uWaveGestureLibrary_Z: 8 gestures from z-axis of handheld device acceleration measurement

Wafer: Binary abnormality detection from in-line process control measurement for silicon wafer processing

WordsSynonyms: 50 handwritten words remapped to 25 classes from shape profiles

Yoga: Gender identification from shape representations of images of yoga poses

5.1.3. Autofead configuration

The run configuration is identical for all problems in the benchmark study as shown in Table 11. 750,000 candidate solutions were generated for each problem over 25 runs from which the minimum fitness candidate is reported as the Autofead solution. The entire study required roughly six months of computing time on a dozen standard desktop workstations utilizing Intel Core i5-3470 quad-core, 3.2 GHz processors.

5.2. Comparing TSC Methods

For benchmarking, Autofead is evaluated in terms of accuracy, interpretability and computational cost on 49 data sets across a wide range of problem types and characteristics. To assess accuracy, Autofead solutions are directly compared with published state-of-the-art TSC methods. The interpretability of solutions and data mining potential is demonstrated through four examples in section 5.2.2. For computational cost, it was not considered worthwhile to perform direct comparisons at this stage of Autofead's implementation and development, but general conclusions are still possible in this regard.

Table 11 – Koza tableau for benchmark study run configuration

Parameter	Setting
Objective	Design optimal feature set for time series classification problems
Solution structure	Classifer selection and set of features consisting of a sequence of functions
Function set	44 functions from function library; omitted *convolve, select,* and *sorted select*
Pattern recognition	Selection of Gaussian Naïve-Bayes, linear disciminant analysis, and logistic regression classifiers
Fitness	Minimize classification error bounded on [0,1] and break ties with quadratic loss
Fitness case sampling	Rxk-fold repeated, stratified cross-validation
Evolution strategy	$(\mu/\rho + \lambda)$ with $\mu = 2{,}000$, $\rho = 2{,}000$, and $\lambda = 100$
Solution size	Maximum size of 5 features each limited to 15 functions and 8 parameters
Population initialization	Ramped to initial maximum size of 3 features each limited to 5 functions and 3 parameters
Selection	Tournament, tournament size 2
Genetic operators	Crossover, 70%; mutate, 7%; reproduce, 3%; add feature, 10%; remove feature, 10%
Termination	280 iterations; 30,000 individuals
Run repetitions	25; 750,000 total individuals

5.2.1. Accuracy

Table 12 and Table 13 provide classification error on independent test data across all problems for the 11 methods included for comparison. The authors believe the methods presented represent the state-of-the-art for TSC accuracy at the time of writing. Table 12 includes Autofead, time series forests, two variants of dynamic time warping, and a shapelet method using a support vector machine (SVM) classifier. The missing values indicate publications that did not provide results for all problems used in this study. Table 13 includes the feature database from Fulcher and Jones [18], a transformation-based ensemble [12], and 4 standard classifiers implemented in the open-source machine learning package Weka applied directly to the time series [55]. The latter methods represent a simple approach to TSC of treating each sample in the time series as a feature and applying standard classification techniques.

The result for the best method on each problem is given in bold text. Autofead and TSF account for over half of the best results across all problems with 13 each. The two DTW variants and the shapelet method each perform best on an additional 7 or 8 problems. The final

Table 12 – Benchmark study classification error for Autofead, TSF, DTW, and Shapelets

Problem	Autofead	TSF entrance [16]	1-NN DTW, best warping window [54]	1-NN DTW, no warping window [54]	SVM (linear) with shapelets [15]
ChlorineConcentration	0.423	0.254	0.350	0.352	0.439
CinC_ECG_torso	**0.016**	0.039	0.070	0.349	-
Cricket_X	0.487	0.290	0.236	**0.223**	-
Cricket_Y	0.390	0.200	**0.197**	0.208	-
Cricket_Z	0.464	0.244	**0.180**	0.208	-
Earthquakes	0.266	-	-	-	-
ECG200	**0.000**	0.080	0.120	0.230	-
ECGFiveDays	**0.002**	0.056	0.203	0.232	0.011
ElectricDevices	**0.340**	-	-	-	-
FordA	**0.002**	-	-	-	-
FordB	**0.186**	-	-	-	-
Gun-Point	0.040	0.047	0.087	0.093	**0.000**
InlineSkate	0.729	0.682	**0.613**	0.616	-
ItalyPowerDemand	0.133	**0.030**	0.045	0.050	0.079
Lightning2	0.443	0.180	**0.131**	**0.131**	-
Lightning7	0.288	**0.260**	0.288	0.274	0.301
Motes	0.133	0.119	0.134	0.165	**0.113**
SonyAIBORobotSurface	0.205	0.233	0.305	0.275	**0.133**
SonyAIBORobotSurfaceII	0.282	0.187	0.141	0.169	-
StarLightCurves	**0.022**	0.036	0.095	0.093	-
TwoLeadECG	0.046	0.118	0.132	0.096	**0.007**
uWaveGestureLibrary_X	0.264	**0.210**	0.227	0.273	-
uWaveGestureLibrary_Y	0.351	**0.288**	0.301	0.366	-
uWaveGestureLibrary_Z	0.298	**0.262**	0.322	0.342	-
Wafer	**0.000**	0.005	0.005	0.020	-
50Words	0.448	0.266	**0.242**	0.310	-
Adiac	0.276	**0.230**	0.391	0.396	0.762
DiatomSizeReduction	0.173	0.049	0.065	**0.033**	0.078
FaceAll	0.366	0.233	0.192	0.192	-
FaceFour	0.330	**0.023**	0.114	0.170	**0.023**
FacesUCR	0.260	0.101	**0.088**	0.095	-
Fish	0.251	**0.154**	0.160	0.167	-
HandOutlines	**0.097**	-	-	-	-
Haptics	0.705	**0.552**	0.588	0.623	-
OSULeaf	0.219	0.434	0.384	0.409	-
SwedishLeaf	0.138	**0.106**	0.157	0.210	-
Symbols	0.068	0.112	0.062	**0.050**	0.154
WordsSynonyms	0.575	0.379	**0.252**	0.351	-
Yoga	**0.064**	0.151	0.155	0.164	-
ARSim	**0.002**	-	-	-	-
CBF	0.004	0.026	0.004	**0.003**	-
MALLAT	0.208	**0.045**	0.086	0.066	-
Synthetic_Control	0.010	0.027	0.017	**0.007**	0.127
Trace	**0.000**	0.020	0.010	**0.000**	0.020
Two_Patterns	**0.000**	0.054	0.002	**0.000**	-
Beef	0.300	0.233	0.467	0.500	**0.133**
Coffee	0.143	0.036	0.179	0.179	**0.000**
OliveOil	0.267	**0.067**	0.167	0.133	-
MedicalImages	0.387	**0.224**	0.253	0.263	0.475
Win/Loss/Tie vs Autofead	-	28/15/0	22/19/2	23/18/2	9/8/0

Table 13 – Benchmark study classification error for feature database, transformation ensemble, and Weka

Problem	Feature database [18]	Ensemble with weighted votes [12]	1-NN Euclidean [55]	MLP [55]	Random forest [55]	SVM (LibSVM) [55]
ChlorineConcentration	-	-	0.315	**0.107**	0.333	0.445
CinC_ECG_torso	-	-	0.099	0.574	0.387	0.219
Cricket_X	-	-	0.436	0.454	0.510	0.459
Cricket_Y	-	-	0.351	0.374	0.441	0.372
Cricket_Z	-	-	0.405	0.436	0.474	0.415
Earthquakes	-	**0.124**	0.288	0.281	0.230	0.252
ECG200	0.010	0.110	0.110	0.160	0.180	0.140
ECGFiveDays	-	-	0.194	0.030	0.333	0.390
ElectricDevices	-	0.378	0.401	0.720	0.359	0.673
FordA	-	0.152	0.320	0.154	0.232	0.484
FordB	-	0.265	0.410	0.285	0.388	0.505
Gun-Point	0.073	0.047	0.080	0.067	0.127	0.233
InlineSkate	-	-	0.669	0.711	0.682	0.784
ItalyPowerDemand	-	-	0.043	0.047	0.061	0.044
Lightning2	0.197	0.230	0.197	0.262	0.295	0.295
Lightning7	0.438	0.301	0.370	0.356	0.370	0.438
Motes	-	-	0.142	0.139	0.166	0.131
SonyAIBORobotSurface	-	-	0.323	0.286	0.336	0.329
SonyAIBORobotSurfaceII	-	-	**0.136**	0.185	0.239	0.185
StarLightCurves	-	-	0.138	0.144	0.063	0.143
TwoLeadECG	-	-	0.275	0.053	0.228	0.429
uWaveGestureLibrary_X	-	-	0.262	0.255	0.276	0.240
uWaveGestureLibrary_Y	-	-	0.338	0.327	0.343	0.327
uWaveGestureLibrary_Z	-	-	0.346	0.318	0.333	0.308
Wafer	**0.000**	0.004	0.006	0.037	0.007	0.007
50Words	0.453	0.352	0.356	0.336	0.444	0.396
Adiac	0.355	0.356	0.407	0.246	0.458	0.916
DiatomSizeReduction	-	-	0.065	**0.033**	0.131	0.699
FaceAll	0.292	0.294	0.314	**0.165**	0.405	0.196
FaceFour	0.261	0.136	0.125	0.125	0.295	0.489
FacesUCR	-	-	0.250	0.242	0.383	0.309
Fish	0.171	0.217	0.217	0.160	0.280	0.549
HandOutlines	-	0.400	0.141	0.641	0.108	0.141
Haptics	-	-	0.620	0.562	0.601	0.575
OSULeaf	**0.165**	0.422	0.455	0.554	0.512	0.508
SwedishLeaf	0.227	0.152	0.203	0.126	0.211	0.269
Symbols	-	-	0.096	0.145	0.212	0.216
WordsSynonyms	-	-	0.378	0.478	0.486	0.483
Yoga	0.226	0.163	0.167	0.260	0.191	0.329
ARSim		0.322	0.486	0.415	0.452	0.500
CBF	0.289	0.172	0.150	0.147	0.182	0.114
MALLAT	-	-	0.178	0.143	0.197	0.429
Synthetic_Control	0.037	0.090	0.120	0.087	0.160	0.023
Trace	0.010	0.190	0.180	0.230	0.160	0.490
Two_Patterns	0.074	0.100	0.094	0.104	0.297	0.107
Beef	0.433	0.179	0.400	0.267	0.433	0.500
Coffee	**0.000**	0.200	0.250	0.036	0.286	0.107
OliveOil	0.100	0.266	0.233	0.133	0.200	0.600
MedicalImages	-	-	0.321	0.309	0.338	0.413
Win/Loss/Tie vs Autofead	7/12/1	8/18/0	21/28/0	25/24/0	15/34/0	15/34/0

row in each table gives pairwise comparisons between Autofead and each of the other 10 methods. The missing values make statistical comparisons difficult; however, these pairwise comparisons clearly show that Autofead is competitive with other state-of-the-art TSC methods.

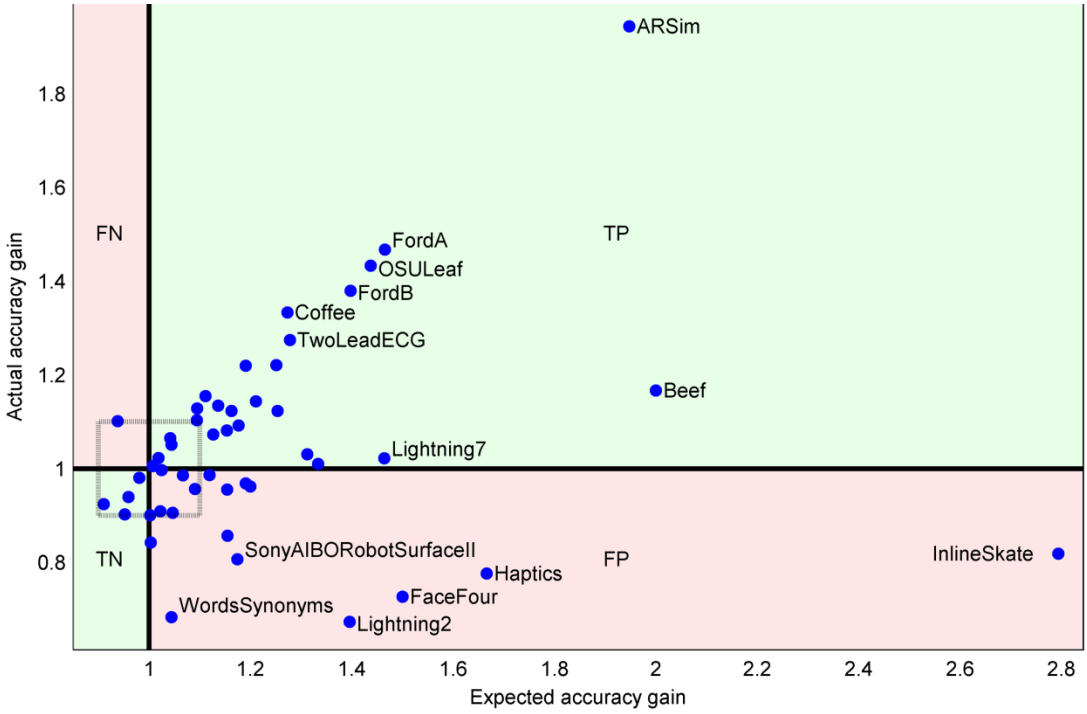
A closer look at Autofead's accuracy for each problem reveals indications of the problem characteristics that lead to poor solutions. Table 14 gives Autofead's ranking and percentile rank for each problem with a percentile rank of 100 indicating Autofead provided the best solution. Problems are divided by the type of data involved and key characteristics of the problem are given in the final two columns. Overall, Autofead performs very well on the sensor data and synthetic problems, but other methods appear better suited for the shape problems. This result is not surprising, as Autofead's strength is in finding transformations that improve the data representation which is unlikely to be the best approach for shape data.

Additionally, a clear correlation is shown between large numbers of classes and poor Autofead performance. The authors hypothesize that the current configuration with a maximum of 5 features per solution and relatively short runs is insufficient to develop the complexity of features required for problems with 7 or more classes. Lastly, problems with very small training data sets pose an additional difficulty for Autofead to avoid overfitting.

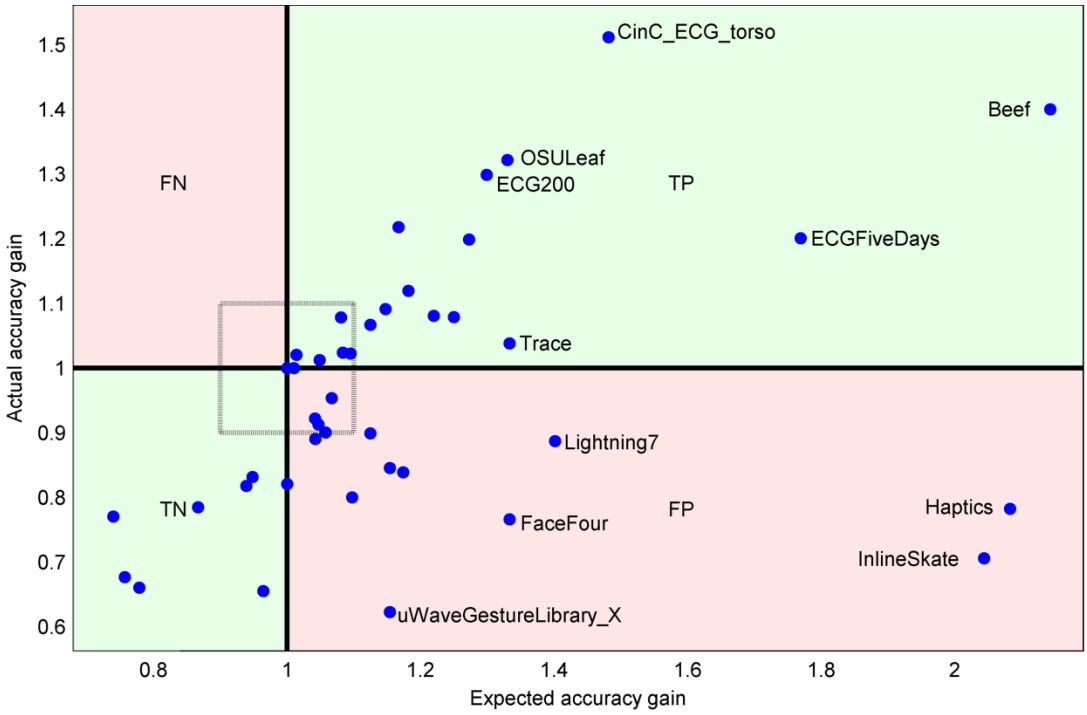
Comparison of accuracy based on testing data alone is insufficient to select between methods for application to new problems. A critical, often overlooked, factor is the ability to correctly determine which method will generalize best to test data based solely on training data. The plots shown in Figure 23 provide such analysis to compare Autofead with 1-NN Euclidean and 1-NN DTW. The expected and actual accuracy gains normalize Autofead's accuracy on the training and testing data, respectively, by accuracy of the compared method. In the shaded regions labeled TP for true-positive and TN for true-negative, Autofead's fitness measure

Table 14 – Autofead performance summary for benchmark study

Data type	Problem	Rank / out of	Percentile rank	Number of classes	Average instances per class
Sensor	ChlorineConcentration	7/9	25	3	156
	CinC_ECG_torso	1/8	100	4	10
	Cricket_X	7/8	14	12	33
	Cricket_Y	7/8	14	12	33
	Cricket_Z	7/8	14	12	33
	Earthquakes	4/6	40	2	161
	ECG200	1/10	100	2	50
	ECGFiveDays	1/9	100	2	12
	ElectricDevices	1/6	100	7	1279
	FordA	1/6	100	2	1786
	FordB	1/6	100	2	1801
	Gun-Point	2/11	90	2	25
	InlineSkate	7/8	14	7	14
	ItalyPowerDemand	9/9	0	2	34
	Lightning2	10/10	0	2	30
	Lightning7	3.5/11	80	7	10
	Motes	4/9	63	2	10
	SonyAIBORobotSurface	2/9	88	2	10
	SonyAIBORobotSurfaceII	8/8	0	2	14
	StarLightCurves	1/8	100	3	333
	TwoLeadECG	2/9	88	2	12
	uWaveGestureLibrary_X	6/8	29	8	112
	uWaveGestureLibrary_Y	7/8	14	8	112
	uWaveGestureLibrary_Z	2/8	86	8	112
	Wafer	1.5/10	100	2	500
Shape representation	50Words	9/10	11	50	9
	Adiac	3/11	80	37	11
	DiatomSizeReduction	8/9	13	4	4
	FaceAll	9/10	11	14	40
	FaceFour	10/11	10	4	6
	FacesUCR	6/8	29	14	14
	Fish	8/10	22	7	25
	HandOutlines	1/6	100	2	500
	Haptics	8/8	0	5	31
	OSULeaf	2/10	89	6	33
	SwedishLeaf	3/10	78	15	33
	Symbols	3/9	75	6	4
	WordsSynonyms	8/8	0	25	11
	Yoga	1/10	100	2	150
Synthetic	ARSim	1/6	100	2	1000
	CBF	2.5/10	89	3	10
	MALLAT	7/8	14	8	7
	Synthetic_Control	2/11	90	6	50
	Trace	1.5/11	100	4	25
	Two_Patterns	1.5/10	100	4	250
Spectral	Beef	5/11	60	5	6
	Coffee	6/11	50	2	14
	OliveOil	9/10	11	4	8
Histogram	MedicalImages	7/9	25	10	38



(a) Autofead versus 1-NN Euclidean



(b) Autofead versus 1-NN DTW, no warping window

Figure 23 – Comparison of expected (training data) and actual (testing data) accuracy gains from selecting Autofead over competing methods

correctly predicts whether or not Autofead will outperform the competing method. The false-positive, FP, region in the lower-right of the figures indicates data sets where Autofead is incorrectly expected to be more accurate, possibly due to overfitting. The analysis shows sufficient evidence for considering Autofead versus 1-NN DTW and a strong preference for Autofead over 1-NN Euclidean. Based solely on accuracy, Autofead is clearly a method worthy of consideration for TSC problems, especially with a small number of classes and large training data sets.

5.2.2. Data Mining

Autofead is designed not only to be competitively accurate as a general TSC method but also to provide easily interpretable solutions making the approach valuable for data mining of time series databases. Although interpretability is difficult to quantify, the highlighted example solutions in this section provide a strong case for Autofead in this regard. The full solutions for all 49 problems can be found in section 5.3.

The ARSim problem is composed of 2 classes of synthetic AutoRegressive (AR) processes with different AR coefficients. This data set is specifically designed to be very difficult to separate in the time domain but much easier through a frequency or auto-correlation representation [12]. Autofead's solution primarily relies on the single feature [*difference*, *FFT magnitude*, *Hilbert magnitude*, *square root*, *normalize*, *slope fit*]. The key steps in this feature are conversion to the frequency domain through *FFT magnitude* and the final function, *slope fit*. *Hilbert magnitude*, *square root*, and *normalize* have little effect on the feature's effectiveness. The feature can be interpreted as a measure of the relative distribution of energy between low- and high-frequency portions of the spectrum after computing first differences. This solution makes sense to separate the two classes, as the power spectral densities of the two processes are distinct despite the similarity in the time domain.

In FordB, the TSC task is to identify whether or not an automotive engine symptom is present from acoustics measurements. The critical feature in the Autofead solution is an averaged measure of energy for the frequency 0.273 rad/s. The feature is represented as [*sliding windows* ($p = 46$), *FFT magnitude*, *square root*, *bisection select* ($p = 3$), *sum*]. First, *sliding windows* divides the 500 sample time series into 21 overlapping subsequences of 46 samples each. Then, *FFT magnitude* is again used to convert to the frequency domain. The last two functions sum the fourth FFT bin from each subsequence. This algorithm is very similar to Welch's method for power spectral analysis. Without further knowledge of the data-generating system, it is impossible to speculate on the importance of the 0.273 rad/s frequency, but the feature clearly separates the two classes very well.

Acceleration data from a robot is used in SonyAIBORobotSurface to determine whether the robot is walking on a concrete or carpeted surface. Autofead uses a very simple feature for its solution computed as [*difference*, *square*, *sorted bisection select* (68)]. The last function sorts the sequence and keeps the last index making the feature the maximum squared magnitude of first differences. The feature is larger for the robot walking on the harder concrete surface due to the higher energy and frequency of impacts during motion.

Lastly, the CBF problem provides an example of a shape problem where Autofead performs very well. The synthetic data set includes three different shapes embedded at a random position in noise. The cylinder shape is a single square wave pulse with discontinuities at each end. The bell shape is a linear ramp up followed by a discontinuous step down, and the funnel is opposite with a step up then linear ramp back down. Autofead's solution utilizes the presence of strong discontinuities in each shape through [*Hilbert imaginary*, *cube*, *sum*]. The Hilbert transform generates a large impulse in the imaginary component inversely related to the discontinuities. Cubing the imaginary component further emphasizes these impulses prior to

summing. The bell and funnel shapes each produce a single impulse, positive and negative, respectively. The cylinder produces opposite but roughly equal impulses at either end of the square wave pulse producing a feature near zero.

In each of the highlighted examples, Autofead's solution relies on a single, easily understood feature. In comparison to other feature-based TSC methods, the solution representation produced by Autofead is far more similar to human expert designed features. Other GP based approaches utilize large, complex tree or graph program structures that require extensive simplification before any attempt at interpretation. Shapelets and the temporal importance curves produced by TSF can provide useful information for certain classes of TSC problems but do not generalize well for data mining of time series which require a data transformation for optimal representation.

5.2.3. Computational Expense

The computational expense of traditional GP is immense for even moderately large problems. The addition of numerical optimization procedures and training of thousands of pattern recognition algorithms was unrealistic on desktop workstations as recently as a decade ago. On a single, modern desktop workstation utilizing four processor cores, a typical Autofead run time is measured on the order of hours. As with many GP methods, the search process can be heavily influenced by the random initialization of the first generation. Therefore, 10’s to 100’s of runs are necessary to provide a measure of confidence that the best possible solution is found requiring days to a few weeks for multiple runs on large problems.

Consequently, Autofead has a relatively large upfront computational expense. The majority of the computational time is spent on numerical optimization of function parameters. Relaxing parameter tolerances or removing features which require parameters greatly reduces run time. Although the computational requirements are high for Autofead runs, the final

solutions are trivial to compute relative to other TSC methods, especially distance-based methods, which use instance-based classifiers. For many applications, the high upfront expense is worthwhile to produce very fast TSC solutions suitable for embedded, real-time, and other resource-constrained systems.

5.3. Detailed Autofead Solutions

The final solution evolved by Autofead for each problem in the benchmark study is listed in this section. Each feature algorithm is listed in square brackets followed by the selected classifier. Parameter values and class mean sequences are given in parentheses where applicable. Index parameters are given using a 0-based index. Problems are listed alphabetically with problem names emboldened.

50Words: [*inverse*, *inverse*, *difference*, *element sum* (class index 30), *sorted bisection select* ($p = 115$)], [*sliding windows* ($p = 178$), *absolute value*, *center*, *slope fit*, *sum*], [*FFT magnitude*, *sum*], [*keep end* ($p = 154$), *normalize*, *slope fit*], [*element product* (class index 12), *slope fit*], Gaussian Naïve-Bayes

Adiac: [*absolute value*, *sum*], [*demean*, *cube*, *sigmoid*, *sum*], [*auto-correlation function*, *sum*], [*control chart* ($thresholds = -0.553, 1.190$), *set minimum value* ($thresholld = 1.512$), *Hilbert phase*, *demean*, *sorted bisection select* ($p = 38$)], [*difference*, *FFT imaginary*, *absolute value*, *slope fit*], Gaussian Naïve-Bayes

ARSim: [*FFT magnitude*, w*avelet* ($p = 0$), *bisection select* ($p = 12$)], [*difference*, *FFT magnitude*, *Hilbert magnitude*, *square root*, *normalize*, *slope fit*], [*center*, *difference*, *auto-correlation function*, *Hilbert phase*, *bisection select* ($p = 1$)], Gaussian Naïve-Bayes

Beef: [w*avelet* ($p = 1$), *slope fit*], [*bisection select* ($p = 52$)], [*FFT imaginary*, *sum*], [*sort order*, *element quotient* (class index 3), *center*, *absolute value*, *sum*], linear discriminant analysis

CBF: [*sorted bisection select* ($p = 43$)], [*Hilbert imaginary*, *cube*, *sum*], Gaussian Naïve-Bayes

ChlorineConcentration: [*difference*, *sign*, *FFT magnitude*, *sum*], [*difference*, *log10*, *FFT real*, *slope fit*], [*Hilbert magnitude*, *FFT real*, *slope fit*], Gaussian Naïve-Bayes

CinC_ECG_torso: [*difference*, *bisection select* ($p = 909$)], [*FFT magnitude*, *sum*], [*FFT magnitude*, *difference*, *sigmoid*, *log10*, *slope fit*], [*element product* (class index 1), *FFT magnitude*, *sum*], [*element product* (class index 3), *absolute value*, *sum*], Gaussian Naïve-Bayes

Coffee: [*square*, *cube*, *FFT magnitude*, *Hilbert phase*, *bisection select* ($p = 16$)], linear discriminant analysis

Cricket_X: [*difference*, *sorted bisection select* ($p = 298$)], [*demean*, *element difference* (class index 2), *log10*, *sorted bisection select* ($p = 299$)], [*difference*, *slope fit*], [*element product* (class index 7), *sum*], [*cube*, *sigmoid*, *difference*, *Hilbert magnitude*, *sum*], linear discriminant analysis

Cricket_Y: [*keep beginning* ($p = 124$), *sum*], [*element product* (class index 7), *difference*, *FFT magnitude*, *sum*], [*slope fit*], [*element product* (class index 4), *sum*], [*difference*, *low-pass filter* ($p = 0.172$), *center and scale*, *absolute value*, *sum*], Gaussian Naïve-Bayes

Cricket_Z: [*element product* (class index 7), *sliding windows* ($p = 100$), *sorted bisection select* ($p = 9$), *sum*], [*sorted bisection select* ($p = 146$)], [*FFT imaginary*, *cube*, *log10*, *sum*], [*element product* (class index 8), *slope fit*], [*element product* (class index 2), *keep beginning* ($p = 180$), *difference*, *log10*, *sorted bisection select* ($p = 99$)], linear discriminant analysis

DiatomSizeReduction: [*FFT real*, *center*, *FFT phase*, *slope fit*], Gaussian Naïve-Bayes

Earthquakes: [*set minimum value* ($threshold = 3.085$), *element product* (class index 1), *sorted bisection select* ($p = 135$)], [*auto-correlation function*, *element sum* (class index 0), *set minimum value* ($threshold = 601$), *difference*, *slope fit*], [*element product* (class index 1), *sorted bisection select* ($p = 496$)], [*FFT phase*, *element product* (class index 1), *sum*], [*set minimum value* ($threshold = 3.79$), *sorted bisection select* ($p = 510$)], linear discriminant analysis

ECG200: [*FFT real*, *log10*, *sorted bisection select* ($p = 0$)], linear discriminant analysis

ECGFiveDays: [*FFT magnitude*, *normalize*, *bisection select* ($p = 5$)], Gaussian Naïve-Bayes

ElectricDevices: [*Hilbert phase*, *keep end* ($p = 45$), *sum*], [*slope fit*], [*sigmoid*, *high-pass filter* ($p = 0.358$), *FFT real*, *sorted bisection select* ($p = 0$)], [*log10*, *difference*, *sorted bisection select* ($p = 0$)], [*keep end* ($p = 0$), *low-pass filter* ($p = 0.123$), *square root*, *demean*, *sorted bisection select* ($p = 92$)], Gaussian Naïve-Bayes

FaceAll: [*square*, *slope fit*], [*square root*, *keep beginning* ($p = 77$), *sigmoid*, *slope fit*], [*bisection select* ($p = 0$)], [*keep end* ($p = 103$), *Hanning window*, *sum*], [*Hilbert magnitude*, *low-pass filter* ($p = 0.622$), *cross-correlate* (class index 4), *set maximum value* ($threshold = 174$), *exponential*, *sum*], Gaussian Naïve-Bayes

FaceFour: [*cross-correlate* (class index 1), *slope fit*], [*element product* (class index 2), *sum*], [*inverse*, *sigmoid*, *element product* (class index 3), *sum*], [*Hilbert phase*, *Hilbert phase*, *element product* (class index 1), *auto-correlation function*, *sum*], linear discriminant analysis

FacesUCR: [*slope fit*], [*Hilbert magnitude*, *slope fit*], [*cross-correlate* (class index 12), *bisection select* ($p = 30$)], [*Hilbert magnitude*, *auto-correlation function*, *sum*], [*auto-*

correlation function, *normalize*, *element difference* (class index 9), *Hanning window*, *demean*, *slope fit*], Gaussian Naïve-Bayes

Fish: [*set minimum value* ($threshold = 1.06$), *sorted bisection select* ($p = 386$)], [*Hanning window*, *sorted bisection select* ($p = 372$)], [*Hilbert phase*, *sum*], [*difference*, *sign*, *slope fit*], [*high-pass filter* ($p = 0.276$), *sliding windows* ($p = 85$), *element product* (class index 1), *sum*, *sum*], Gaussian Naïve-Bayes

FordA: [*sliding windows* ($p = 2$), *normalize*, *sigmoid*, *sorted bisection select* ($p = 0$), *sorted bisection select* ($p = 342$)], [*log10*, *difference*, *sign*, *sigmoid*, *sorted bisection select* ($p = 0$)], [*absolute value*, *cumulative summation*, *sigmoid*, *sorted bisection select* ($p = 305$)], [*demean*, *difference*, *sign*, *square*, *sum*], linear discriminant analysis

FordB: [*sorted bisection select* ($p = 495$)], [*sorted bisection select* ($p = 83$)], [*sliding windows* ($p = 46$), *FFT magnitude*, *square root*, *bisection select* ($p = 3$), *sum*], [*Hilbert magnitude*, *sliding windows* ($p = 186$), *auto-correlation function*, *normalize*, *bisection select* ($p = 8$), *sum*], logistic regression

Gun_Point: [*sigmoid*, *Hilbert imaginary*, *center and scale*, *sigmoid*, *sorted bisection select* ($p = 105$)], [*cube*, *auto-correlation function*, *center and scale*, *log10*, *difference*, *sorted bisection select* ($p = 24$)], [*sorted bisection select* ($p = 87$)], [*sigmoid*, *Hilbert magnitude*, *bisection select* ($p = 17$)], [*inverse*, *square*, *log10*, *sorted bisection select* ($p = 137$)], linear discriminant analysis

HandOutlines: [*sorted bisection select* ($p = 2371$)], [*exponential*, *FFT magnitude*, *slope fit*], [*bisection select* ($p = 912$)], [*sorted bisection select* ($p = 2082$)], [*transpose windows*, *Hilbert imaginary*, *bisection select* ($p = 593$)], logistic regression

Haptics: [*bisection select* ($p = 864$)], [*inverse*, *FFT phase*, *transpose windows*, *element product* (class index 4), *sum*], [*cumulative summation*, *FFT magnitude*, *sigmoid*, *keep*

end ($p = 169$), *sum*], [*inverse*, *FFT phase*, *transpose windows*, *element product* (class index 3), *sum*], [*inverse*, *FFT phase*, *transpose windows*, *element product* (class index 1), *sum*], logistic regression

InlineSkate: [*difference*, *sort order*, *element quotient* (class index 2), *sum*], [*cross-correlate* (class index 4), *high-pass filter* ($p = 0.822$), *absolute value*, *sum*], [*difference*, *sort order*, *element quotient* (class index 4), *sum*], [*difference*, *sort order*, *element quotient* (class index 3), *sum*], [*slope fit*], linear discriminant analysis

ItalyPowerDemand: [*bisection select* ($p = 19$)], [*bisection select* ($p = 0$)], [*set minimum value* ($p = 0.318$), *keep end* ($p = 0$), *FFT phase*, *set maximum value* ($threshold = 1.67$), *bisection select* ($p = 4$)], [*control chart* ($thresholds = -0.309, 0.742$), *keep end* ($p = 7$), *FFT phase*, *bisection select* ($p = 3$)], Gaussian Naïve-Bayes

Lightning2: [*square*, *cube*, *element quotient* (class index 1), *sorted bisection select* ($p = 636$)], [*log10*, *square*, *cube*, *element quotient* (class index 1), *sorted bisection select* ($p = 636$)], Gaussian Naïve-Bayes

Lightning7: [*sigmoid*, *demean*, *slope fit*], [*center*, *square*, *low-pass filter* ($p = 0.238$), *absolute value*, *sum*], [*keep beginning* ($p = 17$), *sorted bisection select* ($p = 104$)], [*sigmoid*, *exponential*, *Hilbert imaginary*, *sorted bisection select* ($p = 0$)], [*Hilbert imaginary*, *element difference* (class index 4), *sorted bisection select* ($p = 318$)], Gaussian Naïve-Bayes

MALLAT: [*difference*, *Hilbert magnitude*, *slope fit*], [*high-pass filter* ($p = 0.765$), *absolute value*, *Hanning window*, *element quotient* (class index 4), *Hanning window*, *sum*], [*element product* (class index 6), *transpose windows*, *exponential*, *sorted bisection select* ($p = 625$)], [*Hanning window*, *slope fit*], [*element product* (class index 3), *low-pass filter* ($p = 0.985$), *sum*], Gaussian Naïve-Bayes

MedicalImages: [*difference*, *FFT imaginary*, *Hilbert magnitude*, *sum*], [*Hilbert imaginary*, *slope fit*], [*sign*, *slope fit*], [*sigmoid*, *FFT phase*, *sum*], [*FFT magnitude*, *absolute value*, *high-pass filter* ($p = 0.619$), *sorted bisection select* ($p = 44$)], Gaussian Naïve-Bayes

Motes: [*sort order*, *difference*, *sorted bisection select* ($p = 44$)], Gaussian Naïve-Bayes

OliveOil: [*sorted bisection select* ($p = 312$)], [*keep end* ($p = 217$), *sum*)], [*sum*], [*bisection select* ($p = 161$)], [*FFT magnitude*, *Hilbert imaginary*, *sort order*, *FFT imaginary*, *bisection select* ($p = 32$)], linear discriminant analysis

OSULeaf: [*FFT magnitude*, *bisection select* ($p = 95$)], [*high-pass filter* ($p = 0.214$), *element sum* (class index 4), *sorted bisection select* ($p = 422$)], [*FFT magnitude*], [*cross-correlate* (class index 5), *square*, *sum*], [*FFT magnitude*, *sigmoid*, *auto-correlation function*, *sorted bisection select* ($p = 47$)], linear discriminant analysis

SonyAIBORobotSurface: [*bisection select* ($p = 64$)], [*difference*, *square*, *sorted bisection select* ($p = 68$)], linear discriminant analysis

SonyAIBORobotSurfaceII: [*difference*, *difference*, *sign*, *Hanning window*, *bisection select* ($p = 7$)], linear discriminant analysis

StarLightCurves: [*sigmoid*, *auto-correlation function*, *slope fit*], [*auto-correlation function*, *normalize*, *sigmoid*, *slope fit*], [*set minimum value* ($threshold = -0.064$), *FFT magnitude*, *square root*, *slope fit*], [*difference*, *square root*, *FFT magnitude*, *slope fit*], logistic regression

SwedishLeaf: [*sliding windows* ($p = 39$), *sum*, *sum*], [*FFT magnitude*, *sum*], [*cross-correlate* (class index 7), *sum*], [*difference*, *FFT magnitude*, *sigmoid*, *Hanning window*, *sum*], [*difference*, *FFT magnitude*, *center and scale*, *slope fit*], linear discriminant analysis

Symbols: [*difference*, *Hilbert magnitude*, *sum*], [*auto-correlation function*, *sum*], Gaussian Naïve-Bayes

Synthetic_control: [*difference*, *center and scale*, *FFT magnitude*, *sum*], [*FFT magnitude*, *absolute value*, *sum*], [*cube*, *slope fit*], [*square*, *auto-correlation function*, *FFT magnitude*, *cube*, *slope fit*], [*cube*, *slope fit*], Gaussian Naïve-Bayes

Trace: [*Hilbert imaginary*, *slope fit*], [*Hilbert magnitude*, *sigmoid*, *FFT magnitude*], linear discriminant analysis

Two_Patterns: [*Hilbert imaginary*, *control chart* ($thresholds = -3.26, 3.25$), *slope fit*], [*demean*, *wavelet* ($p = 1$), *Hanning window*, *bisection select* ($p = 26$)], [*Hilbert imaginary*, *normalize*, *control chart* ($thresholds = -3.18, 3.17$), *sum*], [*Hilbert phase*, *sum*], Gaussian Naïve-Bayes

TwoLeadECG: [*sorted bisection select* ($p = 9$)], [*FFT magnitude*, *FFT real*, *Hilbert phase*, *bisection select* ($p = 9$)], linear discriminant analysis

uWaveGestureLibrary_X: [*difference*, *element product* (class index 2), *log10*, *sum*], [*cumulative summation*, *element product* (class index 1), *sum*], [*demean*, *auto-correlation function*, *square*, *slope fit*], [*element product* (class index 4), *sliding windows* ($p = 218$), *sum*, *sum*], [*absolute value*, *Hanning window*, *square root*, *cube*, *slope fit*], linear discriminant analysis

uWaveGestureLibrary_Y: [*slope fit*], [*FFT imaginary*, *slope fit*], [*Hilbert imaginary*, *slope fit*], [*sliding windows* ($p = 304$), *Hanning window*, *center*, *slope fit*, *keep end* ($p = 1$), *low-pass filter* ($p = 0.985$), *FFT real*, *slope fit*], [*difference*, *element difference* (class index 3), *log10*, *sum*], linear discriminant analysis

uWaveGestureLibrary_Z: [*cross-correlate* (class index 6), *slope fit*], [*Hilbert phase*, *sum*], [*cross-correlate* (class index 2), *sum*], [*slope fit*], [*difference*, *log10*, *Hilbert magnitude*, *sum*], Gaussian Naïve-Bayes

Wafer: [*difference*, *difference*, *sorted bisection select* ($p = 66$)], Gaussian Naïve-Bayes

WordsSynonyms: [*log10*, *Hilbert imaginary*, *slope fit*], [*Hanning window*, *sum*], [*keep beginning* ($p = 234$), *absolute value*, *difference*, *exponential*, *sum*], [*Hanning window*, *sigmoid*, *absolute value*, *normalize*, *slope fit*], [*FFT magnitude*, *Hilbert phase*, *slope fit*], Gaussian Naïve-Bayes

Yoga: [*high-pass filter* ($p = 0.744$), *sorted bisection select* ($p = 47$)], [*sorted bisection select* ($p = 189$)], [*sigmoid*, *slope fit*], [*set maximum value* ($threshold = -0.540$), *sliding windows* ($p = 116$), *sum*, *sum*], [*difference*, *log10*, *sigmoid*, *FFT imaginary*, *sum*], logistic regression

5.4. Conclusions from Benchmark Study

In this study, the accuracy of Autofead solutions is directly compared to state-of-the-art TSC methods on 49 openly available data sets. The accuracy is shown to be competitive with other general, state-of-the-art TSC methods, and in 13 cases the evolved solution provided the best accuracy among all methods compared. Additionally, Autofead is shown to produce highly interpretable features in many cases making the method viable as a data mining tool for time series databases. Computational cost for the evolutionary system is relatively high to perform the search for solutions; however, the computational expense for classifying new time series is very low making Autofead suitable for embedded and real-time systems.

A portion of this chapter has been submitted for publication in Data Mining and Knowledge Discovery, Dustin Harvey and Michael Todd, 2014. The title of this paper is "Automated Feature Design for Time Series Classification". The dissertation author was the primary investigator and author of this paper.

Chapter 6
Method Evaluation

6.1. Solution Space

The Autofead solution space consists of multiple features constructed from functions in the function library and selection of a classifier. Figure 24 summarizes the usage frequency of all functions and classifiers averaged over all runs and problems in the benchmark study. Dimension-reduction functions are represented on a separate scale, as every feature algorithm must include at least one of these four functions. It is important to note that the usage is uniform across all functions and classifiers at the start of each run. Most functions and classifiers have fairly similar usage with the exceptions of *FFT magnitude* and *difference* as slight outliers. It is not surprising that these two functions would be important as they are very common operations in digital signal processing and time series analysis. Similarly, Gaussian Naïve-Bayes is a very popular method and accounts for roughly half of the classifier selections.

The error bars in Figure 24 for each feature and classifier show the minimum and maximum usage for a single problem out of the 49. These values indicate that even though

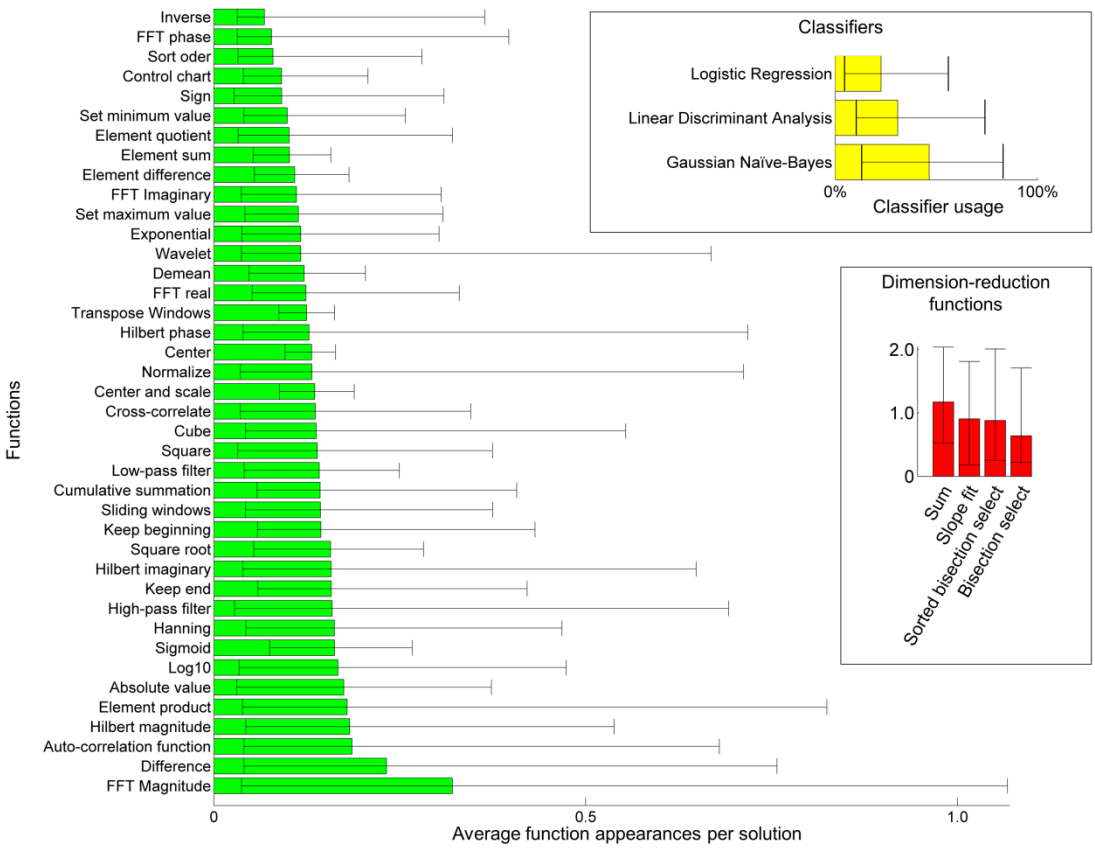


Figure 24 – Function and classifier usage analysis for benchmark study

some functions are used somewhat less on average, they are still important for certain problems. For the classifiers, although Gaussian Naïve-Bayes is used most often, linear discriminant analysis and logistic regression are the majority classifier on some problems.

Another interesting aspect of the solution space is the average size of Autofead solutions. All GP systems are subject to bloat whereby good solutions protect critical portions of code in the evolutionary process by adding sacrificial code [23]. Bloat is evident within Autofead solutions, as they tend to always contain the maximum number of features even though only a few are actually important for classification. The extra features reduce the probability of important features being modified in the breeding process. Within feature algorithms, solutions rarely use more than 5 or 6 functions even though the constraint is set

much higher. The authors hypothesize that Autofead's highly-constrained solution structure does not permit many opportunities for inconsequential code sections, thereby reducing the effects of bloat. This quality helps make Autofead solutions compact and interpretable.

6.2. Search Method

The goal of the search process described in section 2.3 is to evolve an initial randomly generated solution population in the direction of optimal fitness. Table 15 provides strong evidence that the search process is performing well. For each problem, the first two columns show a large decrease in mean fitness between the initial random population and the final population over all 25 runs. The last three columns give the minimum fitness individual for the initial population of all runs, final population of all runs, and final population in the worst performing run. Although very low fitness individuals are generated for some problems in the initial population alone, the search process is clearly further improving the fitness of the final solution. The last column indicates that performing a single run on each problem instead of 25 runs would lead to some decrease in accuracy, but overall run consistency is good.

6.2.1. Genetic Programming

The benchmark study provides an excellent opportunity to perform directed analysis on specific elements of the genetic programming aspect of the Autofead search. Although function and classifier usage vary widely between problems as discussed in section 6.1, runs on the same problem are found to be much more consistent in the distributions of functions and classifiers in the final population. The function set in particular is quickly reduced within an Autofead run to a much smaller set. Although the population begins with 44 equally used

Table 15 – Population fitness analysis for benchmark study

Problem	Initial population mean	Final population mean	Best initial individual	Best run, best individual	Worst run, best individual
ChlorineConcentration	0.445	0.393	0.375	0.364	0.385
CinC_ECG_torso	0.663	0.050	0.090	0.000	0.000
Cricket_X	0.822	0.468	0.539	0.379	0.412
Cricket_Y	0.839	0.521	0.596	0.412	0.471
Cricket_Z	0.855	0.527	0.608	0.414	0.459
Earthquakes	0.190	0.053	0.056	0.022	0.037
ECG200	0.265	0.000	0.000	0.000	0.000
ECGFiveDays	0.320	0.000	0.000	0.000	0.000
ElectricDevices	0.657	0.332	0.355	0.240	0.291
FordA	0.446	0.050	0.059	0.000	0.034
FordB	0.380	0.077	0.078	0.047	0.068
Gun-Point	0.397	0.000	0.000	0.000	0.000
InlineSkate	0.811	0.490	0.517	0.162	0.416
ItalyPowerDemand	0.241	0.019	0.000	0.000	0.015
Lightning2	0.301	0.061	0.050	0.000	0.050
Lightning7	0.596	0.197	0.213	0.059	0.122
Motes	0.331	0.000	0.000	0.000	0.000
SonyAIBORobotSurface	0.275	0.000	0.000	0.000	0.000
SonyAIBORobotSurfaceII	0.265	0.000	0.000	0.000	0.000
StarLightCurves	0.413	0.055	0.062	0.023	0.035
TwoLeadECG	0.300	0.000	0.000	0.000	0.000
uWaveGestureLibrary_X	0.730	0.354	0.423	0.259	0.312
uWaveGestureLibrary_Y	0.696	0.384	0.444	0.318	0.351
uWaveGestureLibrary_Z	0.686	0.346	0.355	0.253	0.328
Wafer	0.193	0.000	0.000	0.000	0.000
50Words	0.857	0.366	0.469	0.266	0.314
Adiac	0.825	0.349	0.432	0.243	0.296
DiatomSizeReduction	0.304	0.000	0.000	0.000	0.000
FaceAll	0.807	0.306	0.429	0.192	0.235
FaceFour	0.590	0.014	0.033	0.000	0.000
FacesUCR	0.743	0.284	0.356	0.155	0.223
Fish	0.711	0.267	0.333	0.171	0.223
HandOutlines	0.340	0.152	0.139	0.105	0.128
Haptics	0.692	0.447	0.464	0.140	0.421
OSULeaf	0.713	0.223	0.337	0.109	0.146
SwedishLeaf	0.779	0.219	0.307	0.135	0.169
Symbols	0.403	0.000	0.000	0.000	0.000
WordsSynonyms	0.808	0.504	0.587	0.386	0.439
Yoga	0.442	0.104	0.090	0.035	0.087
ARSim	0.427	0.000	0.000	0.000	0.000
CBF	0.474	0.000	0.000	0.000	0.000
MALLAT	0.588	0.014	0.010	0.000	0.000
Synthetic_Control	0.588	0.002	0.007	0.000	0.000
Trace	0.463	0.000	0.000	0.000	0.000
Two_Patterns	0.639	0.021	0.129	0.000	0.010
Beef	0.542	0.142	0.160	0.000	0.050
Coffee	0.330	0.000	0.000	0.000	0.000
OliveOil	0.497	0.034	0.020	0.000	0.020
MedicalImages	0.504	0.374	0.364	0.313	0.337

functions, the distribution tends to converge to a reduced set of primarily 5-7 functions within only about five generations.

Additional analysis of the best individuals' evolution trees in the benchmark study indicates the importance of the novel genetic operators used in Autofead. Across all runs, the usage of genetic operators is fixed with the following probabilities: "crossover", 70%; "mutate", 7%; "reproduce", 3%; "add feature", 10%; "remove feature", 10%. However, if only the individuals which comprise the evolution trees of the best individual from each of the 1,225 runs are considered, the frequency of genetic operators which lead to the best solutions is 64.8% "crossover", 6.6% "mutate", 3.5% "reproduce", 19.7% "add feature", and 5.4% "remove feature". The large increase for the "add feature" operator suggests that the feature selection component of the search addressed by this operator is significant for these problems. Full run parameter studies, while computationally expensive, would provide additional opportunities to evaluate the design of the evolutionary search process.

6.2.2. Parameter Optimization

In Autofead, feature evaluations required during parameter optimizations account for 80-90% of the necessary computational effort in a run. This section provides evidence of the benefit of inclusion of parameter optimization in the search process despite the apparent computational cost. Thirty runs of signal detection problem 1 from section 3.2 are performed with and without parameter optimization. For the unoptimized runs, the optimization module was replaced with a random number generator using a uniform distribution over the parameter bounds, and the runs were carried out to 50 generations per the rule-of-thumb for traditional GP. In each generation of each run, the fitness of the best individual was saved for analysis.

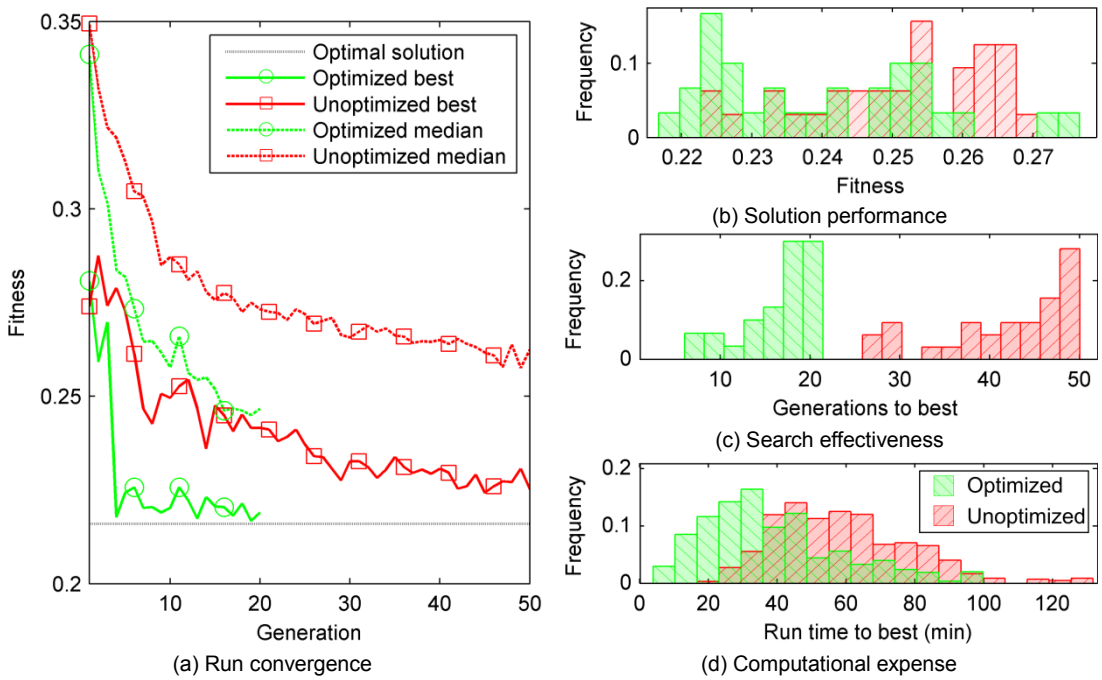


Figure 25 – Analysis of parameter optimization benefit on signal detection problem 1

Figure 25 presents results from the parameter optimization study. In Figure 25(a), the optimized runs converge in far fewer generations than the unoptimized runs and reach a higher median and best run fitness. Figure 25(b) and Figure 25(c) confirm that the inclusion of parameter optimization results in improved solutions found in fewer generations. The convergence rate in terms of generations is admittedly misleading as optimized runs require far more computational effort and run time per generation. By sampling from the distributions in Figure 25(c) and the measured run times, the distributions of run time to reach best fitness are estimated and presented in Figure 25(d). From these results, it is clear that the inclusion of parameter optimization in Autofead results in better, more consistent solutions with a slight reduction in overall run time.

Many design decisions were made in development of the Autofead parameter optimization scheme based on observation of parameter behaviors and fitness surfaces for various features, as described in section 2.3.2. Figure 26 shows example parameter surfaces for

the features from signal detection problem 1 presented in Figure 8. Three of the features end with a *sorted select* function whose parameter has been omitted for visualization purposes. The behavior of *sorted select* in these cases is always to select the maximum value. Each parameter surface is shown for the entire bounded parameter range. Features A and D in Figure 26(a) and Figure 26(b), respectively, contain an invalid parameter region shown as 0.5 fitness in black.

In general, the parameter surfaces for Autofead features tend to be smooth and unimodal or bimodal in a single region containing the optimal point. Outside this region, the surface contains a rough landscape full of local minima. These observations provide

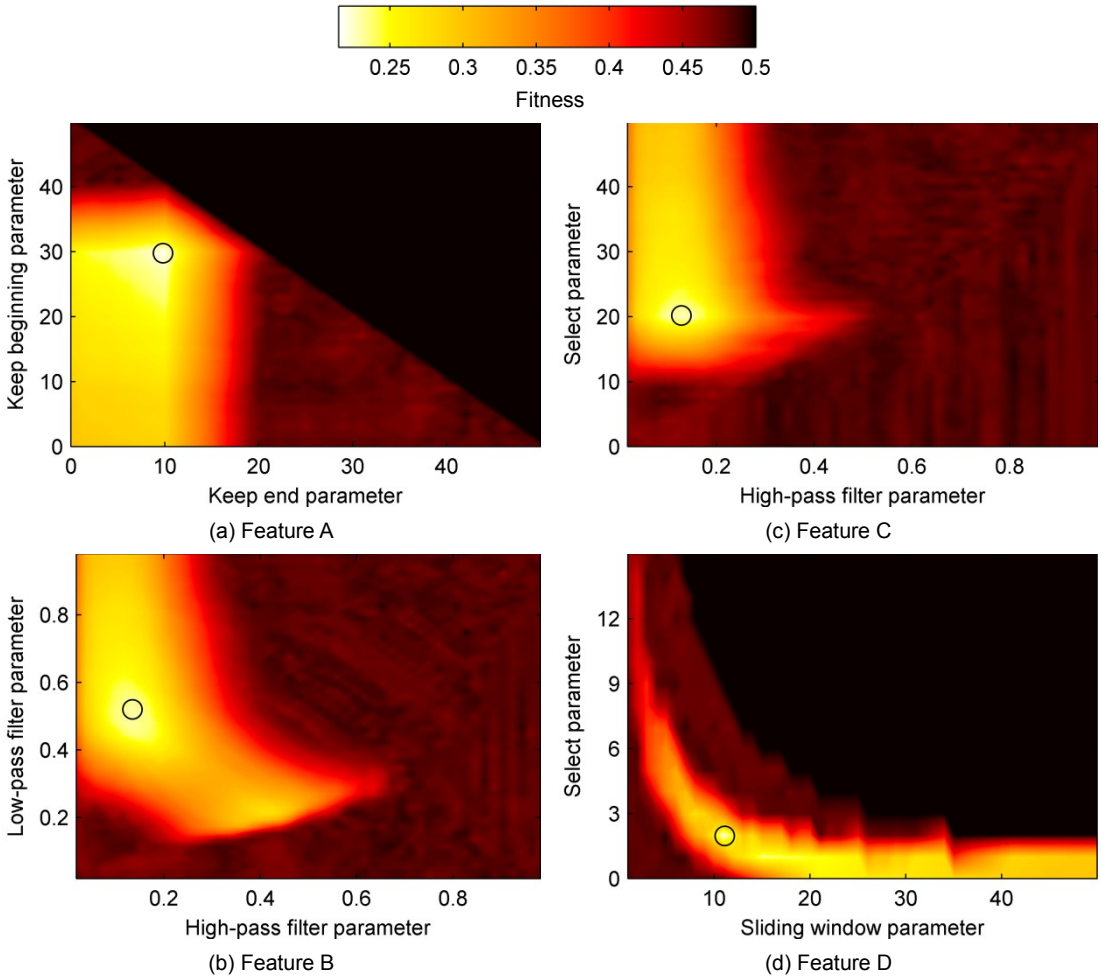


Figure 26 – Parameter surfaces for signal detection problem 1 solutions from Figure 8

justification for the choice of global optimization instead of a local method. The initial grid search locates the smooth fitness region around the optimal parameters then local optimization is initiated to reach optimal fitness. For all four features shown in Figure 26, the parameter optimization scheme found the true, global optimum in less than a hundred feature evaluations. Furthermore, the surfaces indicate a wide range of solution fitness is possible with small changes to algorithm parameters. This observation supports the assertion in section 2.2.3 that good algorithms may be easily overlooked during feature design with poor parameter choices.

6.3. Fitness Quality

The fitness measure is an estimate of the expected classification error on new time series. Autofead uses a sophisticated fitness measure and sampling procedure covered in section 2.3.3, but the large number of degrees-of-freedom makes the method very susceptible to overfitting especially for small training data sets. The importance of including quadratic loss as a tiebreaker in the fitness measure is evident in Table 15 as zero classification error solutions are produced for nearly half the problems on the training data. In these cases, the combined fitness measure uses quadratic loss to select the zero classification error solution that best separates the classes to provide high confidence classifications.

Although the search process is correctly minimizing fitness, if fitness is not representative of solution accuracy then the search will not proceed in the correct direction or select the highest accuracy individual as the final solution. Figure 27 provides an example problem where the estimated fitness is strongly correlated with classification error on the test data. The grayscale image represents the distribution of all 750,000 candidate solutions produced for the StarLightCurves problem. The red circle and green x symbols indicate the

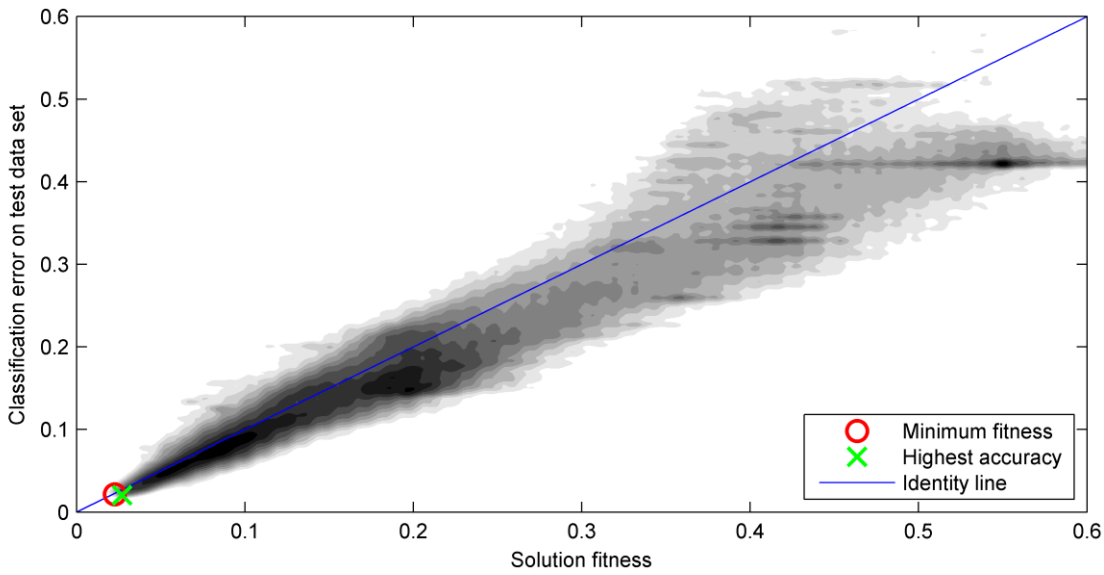


Figure 27 – Fitness quality assessment for StarLightCurves problem from benchmark study

location of the minimum fitness and highest accuracy individuals. Here, the selected minimum fitness individual is nearly the most accurate on the test data.

In Figure 28, the indications of fitness quality for the Lightning2 problem are much poorer. The Lightning2 data set only includes 60 and 61 instances, respectively, for training and testing. The vertical and horizontal banding in the figure are artifacts of the small data sets.

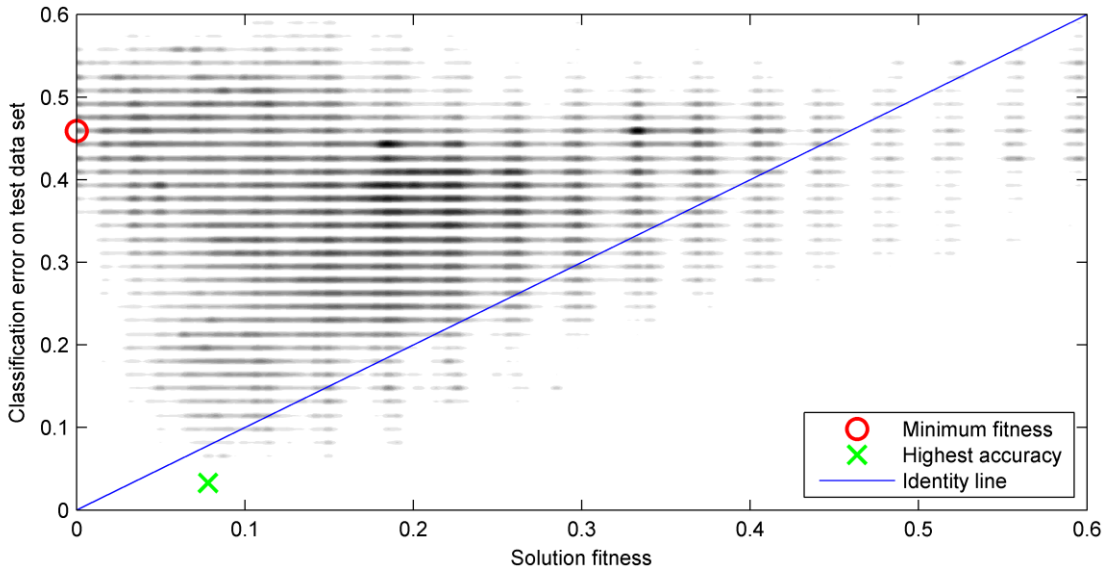


Figure 28 – Fitness quality assessment for Lightning2 problem from benchmark study

Although some highly accurate solutions are evolved with test data classification error below 10%, many more zero fitness solutions exist with poorer accuracy. This problem represents a clear case of overfitting. With 637 samples in the time series and only 30 training instances in each of the two classes, there are many opportunities to produce features with perfect separation of the training data though many are unlikely to generalize for new time series.

Table 16 compares the minimum fitness and highest accuracy individual for each problem. Only for 5 of 49 problems are these two individuals the same, and for half of the problems the minimum fitness individual has at least 5% worse classification error than the highest accuracy individual. If the highest accuracy individual had minimum fitness for each problem, Autofead would provide the best solution on 32 of the 49 problems. Figure 29 provides a visual comparison between the minimum fitness and highest accuracy individuals.

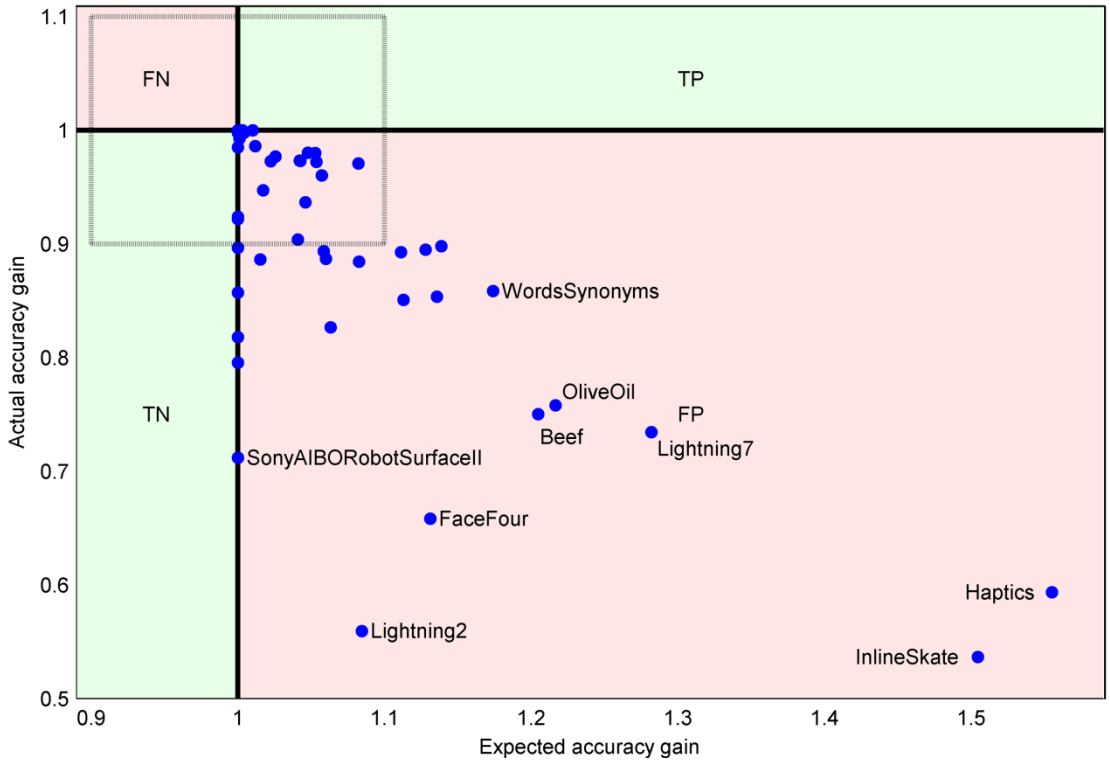


Figure 29 – Comparison of expected (training data) and actual (testing data) accuracy gains from selecting minimum fitness solutions over highest accuracy solutions in benchmark study

Table 16 – Fitness quality summary for benchmark study

Problem	Minimum fitness individual		Highest accuracy individual		Spearman rank-order correlation
	Fitness	Classification error	Fitness	Classification error	
ChlorineConcentration	0.364	0.423	0.390	0.407	0.402
CinC_ECG_torso	0.000	0.016	0.000	0.001	0.905
Cricket_X	0.379	0.487	0.442	0.397	0.922
Cricket_Y	0.412	0.390	0.422	0.356	0.950
Cricket_Z	0.414	0.464	0.484	0.372	0.958
Earthquakes	0.022	0.266	0.133	0.180	0.001
ECG200	0.000	0.000	0.010	0.000	0.594
ECGFiveDays	0.000	0.078	0.000	0.000	0.797
ElectricDevices	0.240	0.340	0.283	0.256	0.846
FordA	0.000	0.002	0.000	0.000	0.912
FordB	0.047	0.186	0.089	0.131	0.805
Gun-Point	0.000	0.020	0.050	0.000	0.748
InlineSkate	0.162	0.729	0.443	0.495	0.336
ItalyPowerDemand	0.000	0.133	0.015	0.022	0.468
Lightning2	0.000	0.459	0.078	0.033	-0.079
Lightning7	0.059	0.356	0.266	0.123	0.483
Motes	0.000	0.133	0.100	0.029	0.610
SonyAIBORobotSurface	0.000	0.218	0.000	0.017	0.486
SonyAIBORobotSurfaceII	0.000	0.303	0.000	0.021	0.612
StarLightCurves	0.023	0.022	0.027	0.020	0.883
TwoLeadECG	0.000	0.076	0.000	0.000	0.722
uWaveGestureLibrary_X	0.259	0.264	0.289	0.244	0.981
uWaveGestureLibrary_Y	0.318	0.351	0.326	0.342	0.967
uWaveGestureLibrary_Z	0.253	0.298	0.287	0.284	0.976
Wafer	0.000	0.000	0.003	0.000	0.458
50Words	0.266	0.448	0.322	0.376	0.943
Adiac	0.243	0.276	0.284	0.246	0.975
DiatomSizeReduction	0.000	0.078	0.000	0.000	0.637
FaceAll	0.192	0.366	0.240	0.233	0.887
FaceFour	0.000	0.364	0.116	0.034	0.257
FacesUCR	0.155	0.260	0.198	0.239	0.962
Fish	0.171	0.251	0.272	0.166	0.782
HandOutlines	0.105	0.097	0.173	0.070	0.518
Haptics	0.140	0.705	0.447	0.503	0.484
OSULeaf	0.109	0.219	0.144	0.136	0.685
SwedishLeaf	0.135	0.138	0.154	0.114	0.978
Symbols	0.000	0.124	0.000	0.023	0.801
WordsSynonyms	0.386	0.575	0.477	0.505	0.835
Yoga	0.035	0.064	0.059	0.042	0.843
ARSim	0.000	0.001	0.000	0.000	0.918
CBF	0.000	0.182	0.000	0.000	0.929
MALLAT	0.000	0.159	0.000	0.019	0.827
Synthetic_Control	0.000	0.007	0.001	0.000	0.925
Trace	0.000	0.000	0.000	0.000	0.540
Two_Patterns	0.000	0.000	0.000	0.000	0.980
Beef	0.000	0.300	0.170	0.067	0.229
Coffee	0.000	0.000	0.000	0.000	0.467
OliveOil	0.000	0.267	0.178	0.033	0.313
MedicalImages	0.313	0.387	0.351	0.314	0.383

By definition, the gains between these two individuals must place all data points in the false-positive region. Gains further from unity indicate data sets with poor fitness quality leading to poor solution choice. Clearly, improvement of the fitness measure or sampling procedure could greatly benefit the method.

Additionally, when the quality of the fitness measure is poor, the search is misdirected and unlikely to discover optimal solutions anywhere within the population. The Spearman rank-order correlation in the last column of Table 16 represents how well the relationship between fitness and test data classification error for the entire solution population can be modeled as a monotonic relationship [56]. Low correlation values below 0.5 indicate that the fitness measure is not driving the search toward accurate solutions for 14 problems.

This chapter discusses various aspects of Autofead's design in light of evidence from analysis of the method's behavior on the signal detection and benchmarking problems. Many positive aspects of the solution space and search method are apparent; however, the search can only be as effective as the fitness measure. Quality of the fitness estimate, likely due to overfitting, is clearly the primary factor degrading Autofead's accuracy.

Portions of this chapter have been published in IEEE Transactions on Evolutionary Computation, Dustin Harvey and Michael Todd, 2014. The title of this paper is "Automated Feature Design for Numeric Sequence Classification by Genetic Programming". Additional portions of this chapter have been submitted for publication in Data Mining and Knowledge Discovery, Dustin Harvey and Michael Todd, 2014. The title of this paper is "Automated Feature Design for Time Series Classification". The dissertation author was the primary investigator and author of these papers.

Chapter 7
TSC Solution Robustness

7.1. Robustness Measures

A quantitative measure of expected performance is required to select the best TSC method for a specific problem. Within methods such as Autofead, this metric further drives the selection of features, parameter values, classifier choices, and other aspects of the solution. While sampling methods and other best practices from machine learning are necessary to make best use of finite data, they are frequently insufficient to provide an accurate estimate of the performance of a real-world system from small training data sets. Additionally, training data is often not fully representative of the conditions experienced by the final system such as varying noise environments. Therefore, development of robust measures to evaluate relative performance of TSC solutions is of great benefit to the Autofead method, TSC, and machine learning as a whole.

Conventionally, the field of classification has relied on simple classification accuracy to measure performance or perhaps a receiver operating characteristic (ROC) analysis as suggested by Provost, Fawcett, and Kohavi [57]. Alternative fitness measures such as quadratic loss,

attempt to make use of the additional performance information contained in class posterior probabilities. Autofead uses a combined fitness measure described in section 2.3.3 where quadratic loss is used as a tiebreaker for classification error on small data sets. While this approach improves on classification error alone, it is insufficient to fully address previously discussed issues of overfitting and solution robustness.

This chapter proposes a method to robustly evaluate time series classifiers in the presence of multiple unknown sources of uncertainty that may affect real-life system deployments. Consideration of TSC solutions under uncertainty has the additional benefit of effectively increasing the size of training data, which may mitigate the issue of overfitting. Traditional methods for uncertainty analysis propagate probabilistic models applied to inputs through a given model or solution to establish confidence bounds on the outputs. Probabilistic methods are poorly suited for the task of evaluating TSC solutions as they require specification of an appropriate probabilistic uncertainty model and repeated sampling to obtain good estimates of performance probabilities. General TSC methods such as Autofead are specifically intended to address problems with limited domain knowledge that would drive the development of a probabilistic model.

7.2. Non-Probabilistic Uncertainty Analysis

Non-probabilistic uncertainty analysis methods address the question of what possible outputs can occur given a set of possible inputs. Therefore, non-probabilistic methods are suitable for determining best-case and worst-case scenarios. Info-gap decision theory (IGDT) provides a non-probabilistic framework to quantitatively evaluate the robustness and opportunity of making distinct choices in the presence of uncertainty [58]. Pierce, Ben-Haim, Worden, and Manson applied IGDT to the training of neural networks for classification and

proposed an interval arithmetic implementation of IGDT for machine learning applications to avoid sampling from the uncertainty model [59, 60]. In the area of SHM, Stull, Hemez, and Farrar recently proposed the use of IGDT to account for the uncertainties in many aspects of SHM system design [61].

7.2.1. Application to TSC Solutions

To apply IGDT analysis to TSC algorithms, decisions must be made as to what point to introduce uncertainty into the solution design process and data flow and an appropriate uncertainty model. The uncertainty model could be applied either to the original input data, measured feature spaces for feature-based methods, or to distance measures for instance-based methods. The latter two options raise scaling issues for making fair comparisons between solutions while applying uncertainty directly to the inputs is equivalent for both feature-based and distance-based TSC methods. Additionally, uncertainty can be introduced within the training data used to design solutions, or final solutions can be evaluated on a test data set with uncertainty to select the most robust solutions. Application of uncertainty to training data introduces an additional complication of training classifiers under uncertainty.

The uncertainty model proposed in this chapter is a simple instantaneous energy-bound model applied to inputs of an independent test data set [58]. The specific choice of model supports a very fast implementation scheme described in section 7.2.2. The model consists of a variable but equal-width interval applied to each time series sample in the test data. The interval radius, r, is dependent on the variable uncertainty level, α, and normalized by the standard deviation of the test dataset, σ, such that

$$r = \alpha\sigma \tag{2}$$

The uncertainty model states that each nominal sample, x_i, may occur anywhere within $(x_i - r) \leq x_i \leq (x_i + r)$.

7.2.2. Interval Arithmetic Implementation

The selection of an envelope-bound info-gap model allows for the time series intervals to be efficiently propagated through feature algorithms using interval arithmetic. Interval arithmetic represents each value as a range of possible values for bounding of errors in mathematic computations. The Autofead function library was implemented in MATLAB using the interval arithmetic package INTLAB to compute feature intervals from the envelope-bound time series info-gap model for increasing uncertainty levels [62].

Next, feature intervals must be propagated through trained classifiers to determine all possible class labels. The direct approach involves development of an interval implementation for the forward method of each classifier. An alternative approach demonstrated here is suitable for low-dimension feature spaces. In two-dimensions, and without loss of generality, feature vectors with interval uncertainty represent rectangular feature interval boxes. Boxes that reside in a single decision region must result in a single class prediction whereas boxes that intersect one or more decision boundaries produce multiple class prediction possibilities. This approach is equivalent in many cases to the threshold-based analysis of class posteriors applied by Pierce, et al. [59, 60]. However, the structure of certain types of classifiers such as decision trees is unsuitable for the latter approach while decision boundaries may be estimated for any classifier, and directly solved for in many cases. Once class prediction probabilities are found for test data set instances, a simple worst-case, best-case analysis is performed to determine the classification error interval under each level of uncertainty.

7.3. Rotating Machinery Case Study

The rotating machinery experiment described in section 4.4 is used here to demonstrate the proposed TSC solution robustness analysis. First, Autofead is applied to the training data to generate candidate solutions. Table 17 summarizes the configuration used for this study. For ease of visualization and implementation, solutions were restricted to two features and a limited function library. Some functions, such as those involving the FFT or Hilbert transform, are omitted since no interval arithmetic implementations are currently available for these functions.

The minimum fitness solution from the Autofead runs, here named solution A, uses a Gaussian Naïve-Bayes classifier. Figure 30 depicts the feature space and decision boundaries for solution A. Feature A1 uses the algorithm [*set maximum value* ($p = 0.09$), *sigmoid*, *difference*, *square*, *sum*] to separate the outer race fault class. The algorithm for feature A2 is [*set maximum value* ($threshold = 0.38$), *difference*, *difference*, *normalize*, *difference*, *absolute value*, *sum*] which tends to produce a smaller feature value for the ball spalling condition than the other classes. From Figure 30, it is evident that the two features together provide a well-

Table 17 – Koza tableau for restricted rotating machinery run configuration

Parameter	Setting
Objective	Design optimal feature set to detect presence of damaged bearings
Solution structure	Two features consisting of a sequence of functions, omitted class mean signals
Function set	26 functions from function library including *absolute value, center, center and scale, control chart, cube, cumulative summation, demean, difference, exponential, Hanning window, inverse, keep beginning, keep end, log10, normalize, select, set maximum value, set minimum value, sigmoid, sliding windows, slope fit, sort order, sorted bisection select, square, sum,* and *transpose windows*
Pattern recognition	Selection of Gaussian Naïve-Bayes, linear disciminant analysis, logistic regression and CART decision tree classifiers
Fitness	Minimize quadratic loss bounded on [0,1]
Fitness case sampling	k-fold cross-validation
Evolution strategy	$(\mu/\rho + \lambda)$ with $\mu = 2{,}000$, $\rho = 1{,}000$, and $\lambda = 100$
Solution size	Two features each limited to 15 functions and 8 parameters
Population initialization	Two features ramped to initial maximum size of 5 functions and 3 parameters
Selection	Tournament, tournament size 2
Genetic operators	Crossover, 75%; mutate, 10%; reproduce, 5%; add feature, 5%; remove feature, 5%
Termination	180 iterations; 20,000 individuals
Run repetitions	475; 9.5 million total individuals

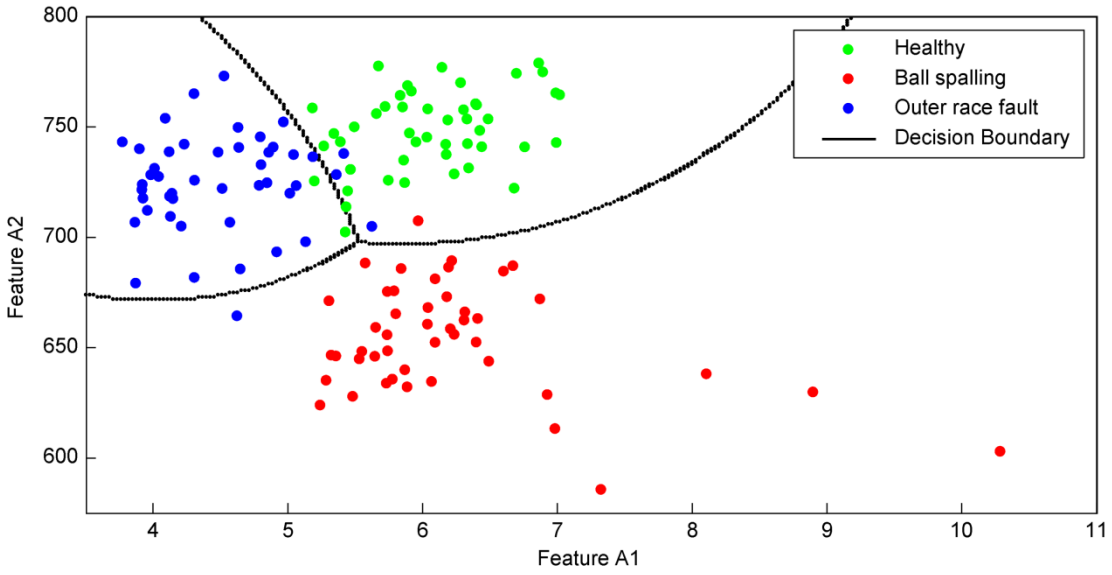


Figure 30 – Restricted rotating machinery solution A feature space

separated feature space with only a few misclassifications.

Although solution A appears to perform very well, nine and a half million candidate solutions are generated by Autofead, of which more than thirty solutions have less than 1% worse fitness than solution A. A better estimate of the generalization performance is found by computing classification error on an independent data set from the training data used within Autofead. Figure 31 compares fitness and classification error on the test dataset for the 100 minimum fitness solutions. The correlation between the two performance measures is only 0.45 within these 100 solutions; although in this case the minimum fitness solution A, highlighted in red, also has the minimum classification error on the test data set. These measures indicate selection of solution A as the final Autofead solution; however, the data is collected from a very limited set of operating conditions such as shaft speeds, examples of damage specimens, temperatures, etc. which could affect performance of a real-world system.

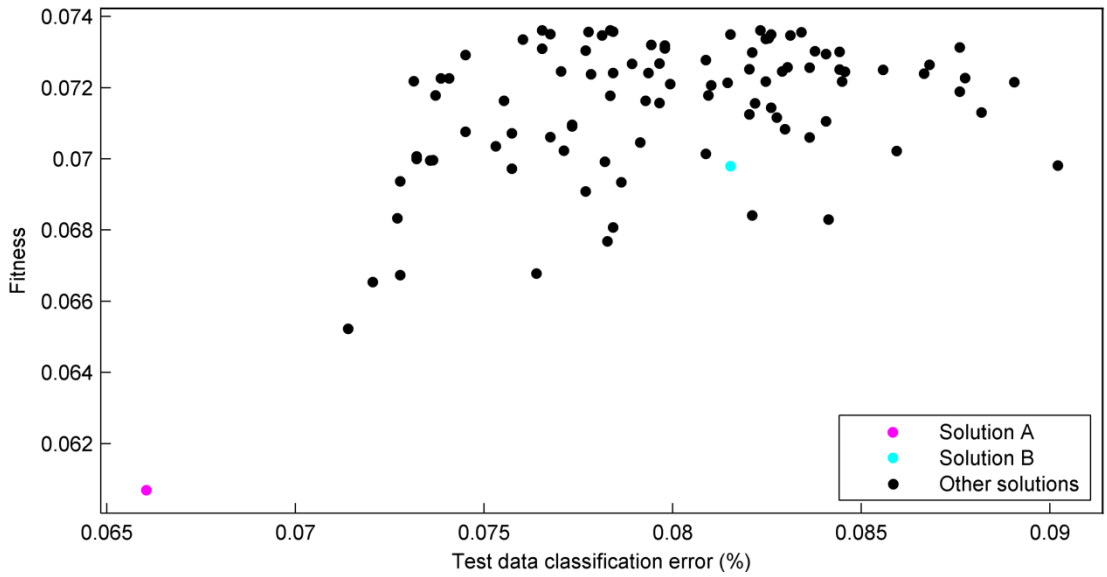
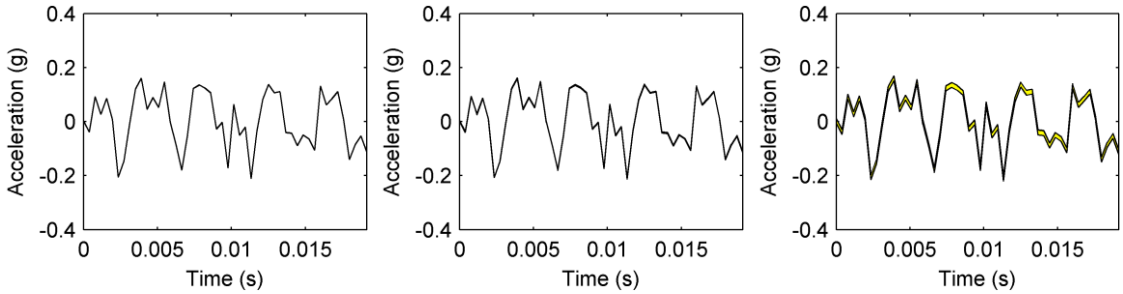


Figure 31 – Restricted rotating machinery performance comparison for 100 minimum fitness solutions

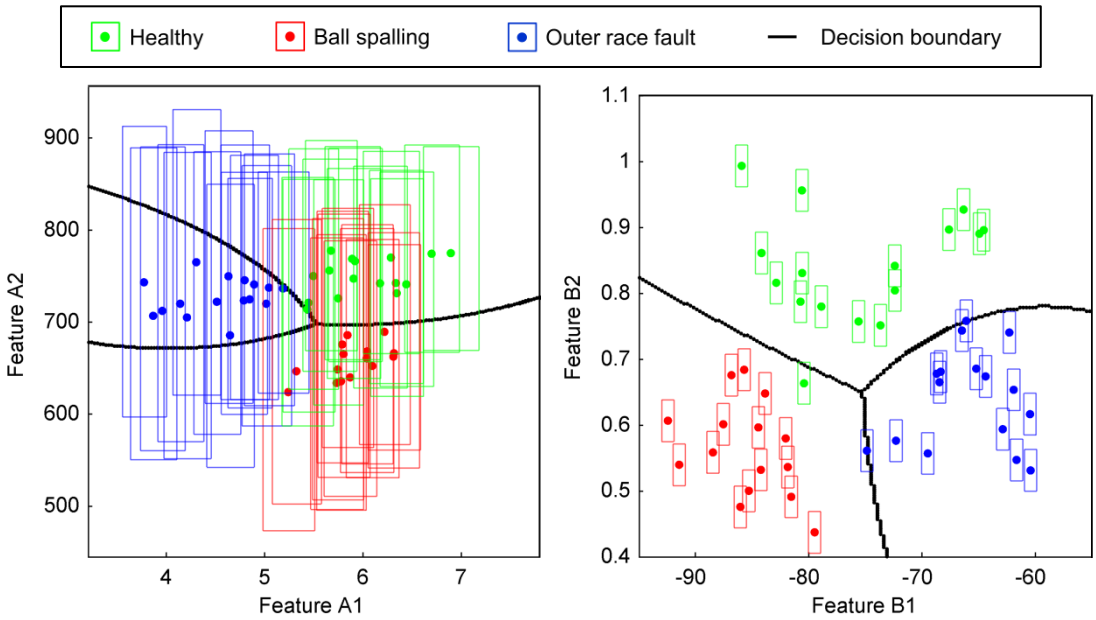
Therefore, the robustness analysis proposed in section 7.2 is carried out on the 100 minimum fitness solutions to identify worse-case performance scenarios in the presence of small changes to the test data. Figure 32 shows the first 50 samples of a single time series under no uncertainty, very small uncertainty, and a noticeable amount of uncertainty. Although the uncertainty levels appear relatively small, the dimensionality of the uncertainty model is very large containing thousands of instances and hundreds of samples in each instance.

Figure 33 depicts the resulting feature intervals for two candidate solutions with the same uncertainty level, $\alpha = 0.014$. The dot inside each feature interval box depicts the feature



(a) No uncertainty, $\alpha = 0$ (b) Low uncertainty, $\alpha = 0.014$ (c) Medium uncertainty, $\alpha = 0.065$

Figure 32 – TSC robustness envelope-bound uncertainty model



(a) Solution A, $\alpha = 0.014$ (b) Solution B, $\alpha = 0.014$

Figure 33 – Restricted rotating machinery solution A and B feature spaces under uncertainty

value under no uncertainty. For solution A, every feature interval overlaps at least one decision boundary at the level of uncertainty shown resulting in 0% worst-case classification accuracy. In contrast, the features in solution B show significantly reduced sensitivity to uncertainty relative to the separation of the classes. Consequently, the worst-case error of solution B is only increased to 14.5% from a nominal level of 8.2% with no uncertainty on the test data. Solution B's feature algorithms are [*center and scale*, *sliding windows* ($p = 21$), *sorted bisection select* ($p = 0$), *sum*] for B1 and [*difference*, *difference*, *difference*, *difference*, *sorted bisection select* ($p = 464$)] for B2. Interestingly, feature B1 divides the time series into 8.20 millisecond segments using *sliding windows*. The segment length is on the order of the characteristic vibration periods for the bearing geometry and shaft speed, and longer segment lengths degrade the performance significantly. Therefore, the algorithm may be exploiting the relative vibration frequency characteristics for the ball spalling and outer race defects to separate the two damage classes.

After propagating the time series info-gap uncertainty model through to the feature space and class prediction possibilities, robustness and opportunity curves are generated by considering the worst-case and best-case class prediction scenarios for increasing levels of uncertainty. Robustness of a candidate solution is measured as the largest uncertainty level which, in the worst case, meets a specific minimum classification accuracy requirement. Opportunity represents the smallest level of uncertainty that could produce windfall of improved classification accuracy. Windfall, here, describes the possibility of achieving better-than-nominal performance as uncertainty increases. Robustness and opportunity curves for the one-hundred minimum fitness Autofead solutions are depicted in Figure 34. Minimum fitness solution A has poor robustness but good opportunity, while solution B is the most robust but provides the worst opportunity. This analysis indicates that selection of solution B as Autofead's final solution will provide a more reliable solution if real-world operating conditions deviate from those under which training and testing data are collected.

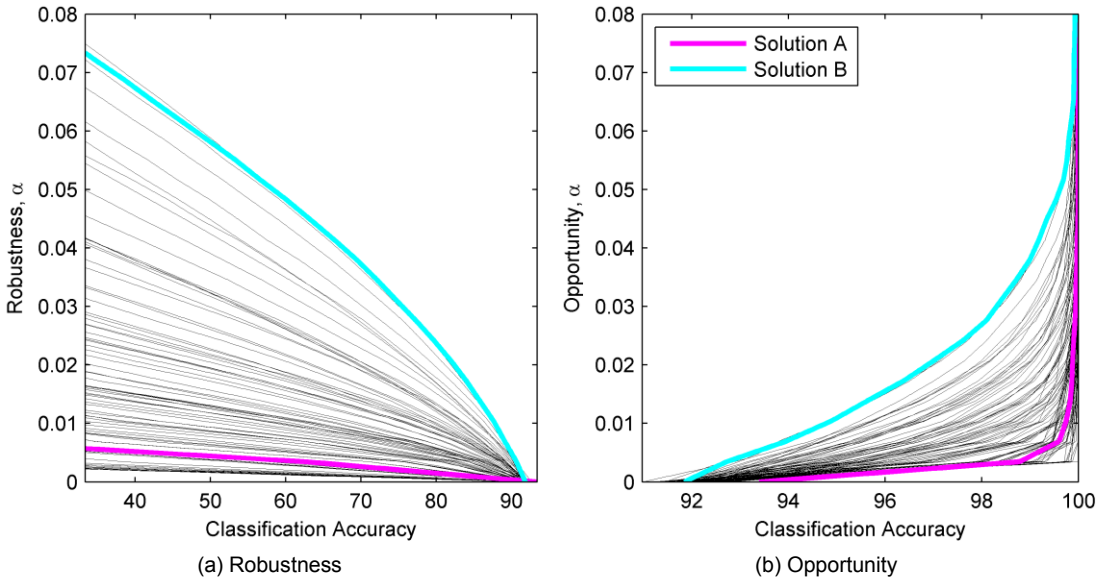


Figure 34 – Restricted rotating machinery solution robustness and opportunity

This chapter proposes and demonstrates experimental results for an info-gap decision theory based robustness analysis of time series classification algorithms. An envelope-bound info-gap uncertainty model is applied directly to the time series in the test data set then propagated through feature measurement and classification to generate class prediction possibilities under increasing uncertainty levels. The interval arithmetic implementation provides a fast, quantitative robustness measure to select among candidate solutions in the presence of uncertainty and guarantee worst-case performance. In comparison to the fitness measure currently utilized by Autofead or simple classification accuracy estimated on independent data, the proposed robustness measure should provide an improved evaluation criterion to estimate performance of real-world systems. Future studies are needed to determine the effects of employing uncertainty analysis in solution evaluation with attention to the issue of overfitting small data sets.

A portion of this chapter has been submitted for publication in Proceedings of SPIE Smart Structures/NDE conference, Dustin Harvey, Keith Worden, and Michael Todd, 2014. The title of this paper is “Robust Evaluation of Time Series Classification Algorithms for Structural Health Monitoring”. The dissertation author was the primary investigator and author of this paper.

Chapter 8
Conclusions and Future Work

8.1. Conclusions

This dissertation encompasses all research to date on the automated feature design method for time series classification called Autofead. The method uses a hybrid genetic programming and numerical optimization process with a novel solution space design to search for highly-informative features within accurate solutions. In many cases, Autofead is shown to produce far more humanly-interpretable algorithms than other general time series classification methods. For this reason, Autofead can serve as a powerful tool for time series data mining and knowledge discovery tasks. Additionally, the method transfers directly to time series regression and forecasting problems. Although the search is computationally intensive, the resulting solutions are compact and easily-computed making them well-suited for embedded systems and other resource-constrained environments.

The experimental studies performed provide extensive evidence of the method's capabilities, effectiveness, and pitfalls. Validation experiments on synthetic problems with known optimal solutions show the ability of Autofead to design optimal and near-optimal

features in a variety of situations. A suite of structural health monitoring related experiments demonstrated the flexibility of the method to address complex problems including multi-class classification and multivariate regression for dynamic system state identification. Lastly, an extensive study on real-world TSC data sets was carried out to benchmark Autofead against state-of-the-art TSC methods. The presented results verify the accuracy of Autofead solutions, at minimum, as competitive with competing methods and human expert designed features.

8.2. Future Work

Future studies related to the presented work could follow numerous beneficial research directions. The largest issue affecting solution accuracy is quality of the estimation of fitness or generalization error. In particular, Autofead is highly susceptible to overfitting for small training data sets leading to overly optimistic performance estimates. This issue is prevalent throughout the machine learning field and any improvements in related techniques such as cross-validation procedures would greatly benefit the Autofead method. Uncertainty analysis provides another possible path for improving fitness quality on small data sets. Further work is necessary to determine the benefit of robustness-based fitness measures such as the non-probabilistic robustness assessment presented in Chapter 7 of this dissertation.

Many implementation improvements are possible to reduce the computational cost of Autofead search runs. In particular, early abandon techniques could be employed during intensive calculations, for instance in the parameter optimization process. In such an approach, continuous evaluation of the value of a candidate solution within the population is performed to abort computations unlikely to produce fitness improvements.

Furthermore, a fully automated post-processing module would be beneficial to perform tasks of analyzing final populations, comparing top solutions, and further refining solutions

through automatic improvement of estimation algorithms. For example, many design decisions within the Autofead function library which constrain the solution space could be relaxed to achieve incremental improvement of final solutions. Examples of relevant solution space constraints include filter design in the filtering functions, selection of wavelet family for the *wavelet* function, and optimization of the parameters of the *sliding windows* operation.

Further investigation of the univariate, binary classifier discussed in section 2.3.2 is required to determine the methods accuracy and limitations. The classifier is employed as part of the objective function for Autofead when applied to binary classification problems due to its low computational cost. Additional validation and comparison to standard methods is necessary to determine usability as a general purpose classification method.

Lastly, directed studies in application areas should investigate methods for incorporating domain knowledge to tailor Autofead for specific tasks. Possible approaches include providing pre-transformed data representations as inputs, customization of the function library, and seeding the initial population with feature algorithms from conventional solutions to related problems. Autofead may perform best in an iterative process where knowledge gained from previous search runs as well as expert analysis is incorporated into future runs until performance goals are realized.

References

1. A.K. Jain, R.P.W. Duin, J. Mao, Statistical pattern recognition: a review, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 22, 1. 2000, 4–37.

2. C.M. Bishop, others, Pattern Recognition and Machine Learning, Vol. 1, Springer: New York. 2006.

3. T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, R. Tibshirani, The Elements of Statistical Kearning, Vol. 2, Springer. 2009.

4. N. Japkowicz, M. Shah, Evaluating Learning Algorithms, Cambridge University Press. 2011.

5. Z. Xing, J. Pei, E. Keogh, A brief survey on sequence classification, ACM SIGKDD Explorations Newsletter, 12, 1. 2010, 40–48.

6. H. Guo, L.B. Jack, A.K. Nandi, Automated feature extraction using genetic programming for bearing condition monitoring, Proc. Machine Learning for Signal Processing, 14th IEEE Signal Processing Society Workshop. 2004, 519–528,

7. T. Fu, A review on time series data mining, Engineering Applications of Artificial Intelligence, 24, 1. 2011, 164–181.

8. P. Esling, C. Agon, Time-series data mining, ACM Computing Surveys (CSUR), 45, 1. 2012, 12.

9. X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, E. Keogh, Experimental comparison of representation methods and distance measures for time series data, Data Mining and Knowledge Discovery, 26, 2. 2013, 275–309.

10. X. Xi, E. Keogh, C. Shelton, L. Wei, C.A. Ratanamahatana, Fast time series classification using numerosity reduction, Proc. 23rd International Conference on Machine learning. 2006, 1033–1040,

11. Z. Prekopcsák, D. Lemire, Time series classification by class-specific Mahalanobis distance measures, Advances in Data Analysis and Classification, 6, 3. 2012, 185–200.

12. A. Bagnall, L.M. Davis, J. Hills, J. Lines, Transformation based ensembles for time series classification, Proc. SDM. 2012, 307–318,

13. L. Ye, E. Keogh, Time series shapelets: a new primitive for data mining, Proc. 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2009, 947–956,

14. L. Ye, E. Keogh, Time series shapelets: a novel technique that allows accurate, interpretable and fast classification, Data Mining and Knowledge Discovery, 22, 1-2. 2011, 149–182.

15. J. Hills, J. Lines, E. Baranauskas, J. Mapp, A. Bagnall, Classification of time series by shapelet transformation, Data Mining and Knowledge Discovery, 28, 4. 2013, 1–31.

16. H. Deng, G. Runger, E. Tuv, M. Vladimir, A time series forest for classification and feature extraction, Information Sciences, 239. 2013, 142–153.

17. B.D. Fulcher, M.A. Little, N.S. Jones, Highly comparative time-series analysis: the empirical structure of time series and their methods, Journal of The Royal Society Interface, 10, 83. 2013.

18. B.D. Fulcher, N.S. Jones, Highly comparative, feature-based time-series classification, Knowledge and Data Engineering, IEEE Transactions on, pre-print. 2014.

19. L.A. Levin, Universal sequential search problems, Problemy Peredachi Informatsii, 9, 3. 1973, 115–116.

20. M. Hutter, The fastest and shortest algorithm for all well-defined problems, International Journal of Foundations of Computer Science, 13, 03. 2002, 431–443.

21. J. Schmidhuber, Optimal ordered problem solver, Machine Learning, 54, 3. 2004, 211–254.

22. J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, A Bradford Book. 1992.

23. R. Poli, W.B. Langdon, N.F. McPhee, A field guide to genetic programming, Lulu Enterprises Uk Limited. 2008. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk.

24. D. Andre, Automatically Defined Features: The simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them, Proc. Advances in Genetic Programming. 1994, 477–494.

25. N.R. Harvey, S.P. Brumby, S. Perkins, J.J. Szymanski, J. Theiler, J.J. Bloch, et al., Image feature extraction: GENIE vs conventional supervised classification techniques, Geoscience and Remote Sensing, IEEE Transactions on, 40, 2. 2002, 393–404.

26. P.G. Espejo, S. Ventura, F. Herrera, A survey on the application of genetic programming to classification, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 40, 2. 2010, 121–144.

27. K.C. Sharman, A.I. Alcazar, Y. Li, Evolving signal processing algorithms by genetic programming, Proc. Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA, First International Conference on. 1995, 473–480.

28. D. Eads, D. Hill, S. Davis, S. Perkins, J. Ma, R. Porter, et al., Genetic algorithms and support vector machines for time series classification, Proc. of SPIE, Vol. 4787. 2002, 75.

29. D.R. Eads, S.J. Williams, J. Theiler, R. Porter, N.R. Harvey, S.J. Perkins, et al., A multimodal approach to feature extraction for image and signal learning problems, Proc. Optical Science and Technology, SPIE's 48th Annual Meeting. 2004, 79–90,

30. K.L. Holladay, K.A. Robbins, Evolution of signal processing algorithms using vector based genetic programming, Proc. Digital Signal Processing, 15th International Conference on. 2007, 503–506.

31. K. Holladay, K. Robbins, J. Von Ronne, FIFTH™: a stack based GP language for vector processing, Genetic Programming, 4445. 2007, 102–113.

32. A. Teller, M. Veloso, Program evolution for data mining, International Journal of Expert Systems Research and Applications, 8. 1995, 213–236.

33. A. Teller, M. Veloso, PADO: A new learning architecture for object recognition, Symbolic Visual Learning. 1996, 81–116.

34. A.V. Oppenheim, R.W. Schafer, Discrete-Time Signal Processing (3rd Edition), Prentice Hall. 2009.

35. G.R. Raidl, A hybrid GP approach for numerically robust symbolic regression, Genetic Programming. 1998, 323–328.

36. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, et al., Scikit-learn: machine learning in Python, The Journal of Machine Learning Research, 12. 2011, 2825–2830.

37. T. Homma, A. Saltelli, Importance measures in global sensitivity analysis of nonlinear models, Reliability Engineering and System Safety, 52, 1. 1996, 1–17.

38. R.P. Brent, Algorithms for Minimization Without Derivatives, Courier Dover Publications. 1973.

39. T. Bäck, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford University Press. 1996.

40. R. Poli, Tournament selection, iterated coupon-collection problem, and backward-chaining evolutionary algorithms, Proc. Foundations of Genetic Algorithms. 2005, 132–155.

41. C. Nadeau, Y. Bengio, Inference for the generalization error, Machine Learning, 52, 3. 2003, 239–281.

42. J.R. Koza, Genetic Programming II: Automatic Discovery of Reusable Programs, A Bradford Book. 1994.

43. D.Y. Harvey, M.D. Todd, Automated feature design for numeric sequence classification by genetic programming, Evolutionary Computation, IEEE Transactions on, pre-print. 2014, doi:10.1109/TEVC.2014.2341451.

44. D.Y. Harvey, M.D. Todd, Structural health monitoring feature design by genetic programming, Smart Materials and Structures, 23, 9. 2014, 095002.

45. D.Y. Harvey, M.D. Todd, Automated feature design for time series classification, Data Mining and Knowledge Discovery, in review. 2014.

46. S.W. Doebling, C.R. Farrar, M.B. Prime, A summary review of vibration-based damage identification methods, Shock and Vibration Digest, 30, 2. 1998, 91–105.

47. C.R. Farrar, S.W. Doebling, D.A. Nix, Vibration–based structural damage identification, Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, 359, 1778. 2001, 131–149.

48. C.R. Farrar, K. Worden, Structural Health Monitoring: A Machine Learning Perspective, Wiley. 2012.

49. K. Worden, C.R. Farrar, G. Manson, G. Park, The fundamental axioms of structural health monitoring, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 463, 2082. 2007, 1639–1664.

50. A. Raghavan, C.E. Cesnik, Review of guided-wave structural health monitoring, Shock and Vibration Digest, 39, 2. 2007, 91–116.

51. E.B. Flynn, M.D. Todd, P.D. Wilcox, B.W. Drinkwater, A.J. Croxford, Maximum-likelihood estimation of damage location in guided-wave structural health monitoring, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 467, 2133. 2011, 2575–2596.

52. M. Lebold, K. McClintic, R. Campbell, C. Byington, K. Maynard, Review of vibration analysis methods for gearbox diagnostics and prognostics, Proc. 54th Meeting of the Society for Machinery Failure Prevention Rechnology, Virginia Beach, Virginia. 2000, 623–634.

53. E. Figueiredo, G. Park, J. Figueiras, C. Farrar, K. Worden, Structural health monitoring algorithm comparisons using standard data sets, Los Alamos National Laboratory (LANL), Los Alamos, NM, United States. 2009.

54. E. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei, et al., The UCR Time Series Classification/Clustering Homepage. 2011. http://www.cs.ucr.edu/~eamonn/time_series_data/

55. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software: an update, ACM SIGKDD Explorations Newsletter, 11, 1. 2009, 10–18.

56. C. Spearman, The proof and measurement of association between two things, American Journal of Psychology, 15, 1. 1904, 72–101.

57. F.J. Provost, T. Fawcett, R. Kohavi, The case against accuracy estimation for comparing induction algorithms, Proc. 15th International Conference on Machine Learning. 1998, 445–453.

58. Y. Ben-Haim, Info-Gap Decision Theory: Decisions Under Severe Uncertainty, Academic Press. 2006.

59. S.G. Pierce, K. Worden, G. Manson, A novel information-gap technique to assess reliability of neural network-based damage detection, Journal of Sound and Vibration, 293, 1. 2006, 96–111.

60. S.G. Pierce, Y. Ben-Haim, K. Worden, G. Manson, Evaluation of neural network robust reliability using information-gap theory, Neural Networks, IEEE Transactions on, 17, 6. 2006, 1349–1361.

61. C.J. Stull, F.M. Hemez, C.R. Farrar, On assessing the robustness of structural health monitoring technologies, Proc. Topics in Model Validation and Uncertainty Quantification, Volume 4. 2012, 1–11.

62. S.M. Rump, Developments in Reliable Computing, INTLAB-INTerval LABoratory, Kluwer Academic Publishers, Dordrecht, Netherlands. 1999.