A Multimodal Approach to Feature Extraction for Image and Signal Learning Problems

Damian R. Eads[a,c], Steven J. Williams[a], James Theiler[a], Reid Porter[b], Neal R. Harvey[a], Simon J. Perkins[a], Steven P. Brumby[a], and Nancy A. David[a]

[a]Space and Remote Sensing Sciences Group
[b]Space Data Systems Group
Los Alamos National Laboratory
Los Alamos, NM, USA

[c]Department of Computer Science
Rochester Institute of Technology
102 Lomb Memorial Drive
Rochester, NY, USA

ABSTRACT

We present ZEUS, an algorithm for extracting features from images and time series signals. ZEUS is designed to solve a variety of machine learning problems including time series forecasting, signal classification, image and pixel classification of multispectral and panchromatic imagery.

An evolutionary approach is used to extract features from a near-infinite space of possible combinations of nonlinear operators. Each problem type (i.e. signal or image, regression or classification, multiclass or binary) has its own set of primitive operators. We employ fairly generic operators, but note that the choice of which operators to use provides an opportunity to consult with a domain expert. Each feature is produced from a composition of some subset of these primitive operators. The fitness for an evolved set of features is given by the performance of a back-end classifier (or regressor) on training data.

We demonstrate our multimodal approach to feature extraction on a variety of problems in remote sensing. The performance of this algorithm will be compared to standard approaches, and the relative benefit of various aspects of the algorithm will be investigated.

Keywords: machine learning, support vector machines, genetic programming, remote sensing, image processing, time series analysis, lightning, classification, regression, automated feature extraction

1. INTRODUCTION

In a previous publication,[1] we introduced ZEUS as a pattern recognition tool for time-series signals. Our initial interest was with the classification of lightning strikes (as measured with a high-speed radio-frequency receiver on the FORTÉ satellite[2]), but we have since extended the ZEUS software in two directions. One extension permits ZEUS to be used for images as well as time-series. Also, where the earlier ZEUS was based on a model in which the entire time-series was a single (high-dimensional) sample point, the new ZEUS can be used in a mode that identifies each time point (or, for the images, each pixel) as a separate sample. This allows ZEUS to be used for segmenting different epochs within a long time-series signal, or for producing a pixel-by-pixel classifications within an image. Thus, ZEUS is designed for use in four separate modes, as described in Table 1: time series forecasting, time series classification, image classification, pixel-by-pixel classification within an image. Although the problems that characterize these modes are quite different in character, many of the same tools are used in their solution, and ZEUS provides a framework for incorporating those tools in a way the permits them to be used for a wide range of applications.

ZEUS is part of the Intelligent Searching of Images and Signals[3] project at Los Alamos, and follows other pattern recognition software that has been developed as part of that project, including GENIE[4–6] and AFREET.[7]

The previous generation of ZEUS was implemented in C++, but the more recent implementation of ZEUS has been rewritten with the kernel and interface in Java, and most of the mathematical processing in MATLAB . We found that the higher-level language provides the ability to more rapidly prototype new modules and capabilities. ZEUS uses the MFITSIO[8] package for reading and writing data, and the OSU SVM package[9] for the support vector machine back-end. For more information on the Zeus project, see `http://www.zeus.lanl.gov`.[10]

Send correspondence to Damian Ryan Eads – E-mail: eads@lanl.gov, Telephone: (505) 667-9575, Address: Space and Remote Sensing Sciences Group, Los Alamos National Laboratory, MS D436, Los Alamos, NM, USA

	Signals	Images
sample-wise	Time series forecasting*	Pixel-by-pixel classification
signal-wise	Time series classification	Image classification*

Table 1. Four modes of operation designed for the ZEUS software. An asterisk indicates that the mode has not been fully implemented or tested.

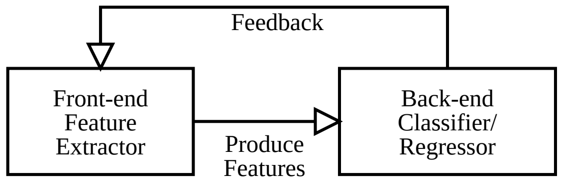

Figure 1. The basic wrapper architecture of Zeus consists of two phases: front-end feature extraction and back-end pattern recognition.

2. BACKGROUND INFORMATION

The overall approach of ZEUS follows the wrapper pattern shown in the diagram in Figure 1. The basic wrapper architecture consists of two main processes: front-end feature extraction and back-end pattern recognition. The wrapper cycle behaves differently for training and exploitation.

- In training, a population of feature extractors is maintained. These feature extractors are programs composed of image and signal processing operations. Their purpose is to produce features from the input data, which is used to build models for classification or regression. These models are then applied to the feature patterns to obtain a result and compute a fitness. This fitness measure is then fed back to the front-end to help the optimizer refine the feature extractors. This cycle continues until some stopping condition is satisfied—either the desired performance is attained or the maximum number of iterations is exceeded.
- In exploitation, a single feature extractor is applied to the input data and a set of feature patterns is produced. The back-end model is then applied to these feature patterns to produce a result. No feedback occurs.

This technique of combining front-end stochastic optimization of feature extractors with back-end pattern recognition techniques helps automate the tool-building process, which is typically very expensive to achieve manually. The wrapper technique was chosen for ZEUS because of its success when employed with the AFREET and GENIE tools.[4–7]

All of the ZEUS classifiers have an "intuit" mode and a "study" mode. The intuit mode applies the back-end (usually a support vector machine classifier) directly to the raw data; the study mode uses an iterative method (usually a genetic algorithm) to produce a set of features that are then applied to the back-end. The intuit mode, in general, is much faster to train than the study mode. And while the study mode can achieve higher in-sample scores, the no-free-lunch theorem[11] assures us that we cannot say with certainty which will produce higher out-of-sample scores.

2.1. Front-end Feature Extraction

Feature extraction also occurs in the back-end because the pattern recognizer (such as a neural network or support vector machine) attempts to capture complex relationships in the data by transforming it in a way that can be discriminated to recognize the patterns of interest.

However, previous techniques found it useful to employ additional feature extraction in front of the back-end to better define the patterns of interest, and ultimately improve performance.[4,5,7] Thus, to prevent confusion, the term *feature extraction* shall refer to the front-end feature extraction.

2.1.1. Feature Extractor Representation

These feature extractors are programs represented by graph or tree structures. The steps of feature extraction are operations specific to a particular modality; images or signals. The following example of a feature extractor: outputs three feature planes; uses morphological, arithmetic, laws texture energy, and moment spatial operators; is represented by tree structures given the S-expression structure of the program; and since intermediate values are permitted, is also represented by a graph structure.

```
(setplane 0 (gslaws (getplane 0 1) '0.98407 '0.77694))
(setplane 1 (gslincomb (getplane 1 0) (getplane 1 0) '0.84629 '0.56066))
(setplane 2 (gsdivide (getplane 0 0) (getplane 0 0)))
(setplane 0 (gserode (getplane 0 0) (gssel 0.96021 0.89993 0.02082 0.78384)))
(setplane 2 (gsdilate (getplane 0 2) (gssel 0.21138 0.86163 0.50033 0.99005)))
(setplane 2 (gsmean (getplane 0 3) '0.34806))
```

The program above transforms the input space into a feature space that will hopefully improve classification performance. The selection of operators, their parameters, and the order they appear in the program must be carefully chosen. Since the feature extractors are initially random, they must be refined using some form of stochastic optimization.

2.1.2. Genetic Programming

In ZEUS, we used genetic programming (GP) to refine feature extractors. Genetic programming was first formally introduced and explored by John Koza.[12] We briefly describe GP here; however refer the reader to Banzhaf[13] for a more in-depth discussion. GP is a form of evolutionary algorithm, where the individuals being evolved are computer programs, such as the one shown above. Most computer programs such as those written in LISP can be represented by tree structures. When intermediate values are permitted, the inherent representation of the program is a graph. Special graph and tree-based mutation and crossover operations are sometimes used.[7, 12, 13] The program representation in ZEUS consists mainly of one edge in the graph per line of code. For ease of implementation, we use the same element-wise crossover and mutation operators as used previously ZEUS.[1] We will briefly describe the genetic operators that were used to conduct the experiments for this study:

- **Steady State GA** A steady state GA is a genetic algorithm which replaces a few chromosomes (typically only one or two) in a population during each iteration.[14] It differs from a generational GA in that it does not replace the entire population during each generation.

- **Selection** Before crossover, two parent chromosomes A and B must be chosen from a population of chromosomes. *Tournament Selection* involves selecting four chromosomes: two of which are the losers of two different tournaments and the other two are the winners. The winners are recombined to produce two offspring which replace the losers in the population. A tournament involves evaluating N chromosomes where N is the tournament size. The individual with best fitness is the winner and the one of the worst fitness is the loser.

- **Crossover** A crossover operator combines two parent chromosomes A and B together to produce a single offspring C. *Uniform Crossover* involves replacing the genes at each position in the chromosome C with the corresponding genes in either A or B, chosen with equal probability. The genes themselves are simply copied and are not modified in any way.

- **Mutation** Each new offspring has a probability p_m of mutation. *Gene randomization* involves choosing a gene (line-of-code) at random, discarding it, and replacing it with an entirely new line-of-code.

2.1.3. Data Types

There are eleven data types available in the ZEUS system, each having their own unique identifier (UID): `scalar` (1), `series` (2), `image` (3), `msi` (4), `void` (5), `bounds` (6), `time_series` (7), `spectral_series` (8), `index` (9), `pyramid` (10), and `struct_element` (11).

2.1.4. Primitive Operators

Each primitive operator: is defined in a separate M-file, has a *return* type, and has a type for each of its input parameters. Primitive operators that do not return any values (mutators, for example) have a return type of `void`. In building each line of code, the parameter types are matched with the types of input expressions to ensure invalid programs are not produced. Table 2 below lists all the operators that are available in the ZEUS system.

Signature	**Description**
`image absdiff(image im1, image im2)`	Absolute value of two images subtracted.
`image add(image im1, image im2)`	Image addition.
`image addc(image im, scalar c)`	Adds a scalar constant to each pixel.
`image close(image im, struct_elem e)`	Morphological closing.
`image dilate(image im, struct_elem e)`	Morphological dilation.
`image divide(image im, image im)`	Divides two images pixel-by-pixel.
`image erode(image im, struct_elem e)`	Morphological erosion.
`image gauss(image im, scalar khs, scalar kvs)`	Smooths by convolution with a Gaussian.
`image getplane(scalar num, scalar type)`	Retrieves a data plane or scratch plane.
`image laplacian(image im, scalar type)`	Laplacian edge detect.
`image laws(image im, scalar type1, scalar type2)`	Laws texture energy measure.
`image lincomb(scalar a, image b, scalar c, image d)`	Linear combination of planes.
`image mean(image im, scalar nsize)`	Neighborhood average.
`image median(image im, scalar nsize)`	Neighborhood median.
`image mexhat17(image im, scalar type)`	Mexican hat filter of size 17.
`image mexhat5(image im, scalar type)`	Mexican hat filter of size 5.
`image mexhat9(image im, scalar type)`	Mexican hat filter of size 9.
`image open(image im, scalar type)`	Morphological opening.
`image prewitt(image im, scalar type)`	Prewitt compass operators.
`image prewittgrad(image im, scalar type)`	Prewitt with absolute gradient magnitude.
`struct_elem sel(scalar a, ..., scalar d)`	Constructs a structuring element.
`void setplane(scalar num, image img)`	Sets a scratch plane.
`image sobel(image im, scalar type)`	Sobel edge detection.
`image sobeldir(image im, scalar type)`	Directional sobel edge detection.
`image subtract(image a, image b)`	Image subtraction.
`image subtractc(image im, scalar c)`	Image subtraction with scalar constant.
`image unsharp(image im, scalar sigma)`	Image unsharpening.

Table 2. Primitive Image Domain Operators in ZEUS

Programs for both signal and image domains are built in ZEUS line-by-line. In the case of images, the user gives a multispectral image as input and specifies the number of scratch planes to output. The program builder then writes a program which takes input from both the data planes and scratch planes, and outputs to scratch planes. This approach is very similar to the one taken in GENIE.[4, 5] The operator pool and the implementation of operators differs than that of GENIE and AFREET however.[7]

2.2. Back-end Pattern Recognizer

The back-end is a traditional machine learning classifier. It finds the function, from the function class of interest, that optimizes the fit of the features to the marked up truth.

Currently, the main back-end is a support vector machine classifier. However, it can also employ a Fisher discriminant back-end, which is generally faster to compute than the support vector machine solution, but is optimal only if the two classes exhibit a common Gaussian distribution.

The back-end is also the part of the classifier that would be used for multi-class classification and/or regression which was not investigated in this study.

2.2.1. Support Vector Machines

The support vector machine is a linear classifier which optimizes a sum of two terms:[15] one is a quadratic term that discourages large coefficients in the linear classifier, and one is a piecewise linear term that penalizes misclassifications.

$$\underbrace{\frac{1}{2}\mathbf{w}^T\mathbf{w}}_{\text{complexity penalty}} \quad \underbrace{+C\sum_i \max\left[0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i - b)\right]}_{\text{misclassification penalty}} \tag{1}$$

Taken together the two terms are convex in the linear classifier coefficients; the optimization is a quadratic programming problem, and polynomial-time algorithms can be proven to exist.[16] Since their introduction in the mid-1990's by Vapnik *et al.*,[17, 18] a number of algorithms especially designed for the support vector machine have been developed. The choice of the piecewise linear penalty function for the misclassification terms represents a compromise between quadratic penalties (whose derivatives are linear, and whose solutions involve only linear algebra, but which are more sensitive to outliers) and piecewise constant penalty functions (which is even more robust to outliers and matches more directly to the desired outcome which is to minimize the number of errors, not their magnitude, but which are not convex, and therefore exhibit many local minima).

As well as the computational advantages of this penalty function, there is a mathematical advantage as well, in the form of guaranteed bounds on the generalization error in terms of the in-sample error.

In addition to generalization error bounds, Support Vector Machines are able to control overfitting by maximizing the margin, characterized by the region from the nearest training point to the discriminant surface.

Kernels Finally, it is important to remark that support vector machines also permit the use of kernels. With kernels, nonlinearity is attained by implicitly projecting the input space into a higher dimension. A kernel function K is defined as $K(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x})\phi(\mathbf{y})^T$ where $\phi(\mathbf{u}) : \mathbb{R}^p \to \mathbb{R}^{p'}$ and $p' \gg p$.[15] The dimension p' is typically much larger than p and may theoretically be infinite as is the case with the radial basis function $K(\mathbf{x},\mathbf{y}) = e^{\frac{-\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}}$. Finding $\phi(\mathbf{u})$ is unnecessary—a function $K(\mathbf{x},\mathbf{y})$ is a kernel if it obeys the Mercer condition.[19]

For more information on kernels, see Schölkopf and Smola's book[19] for an excellent treatment, but note that they enable support vector machines to produce nonlinear classifiers with the same mathematical advantages as the linear classifiers.

The AFREET[7] package was developed as part of the ISIS project for applying support vector machines to pixel-by-pixel classification in multispectral images. The AFREET software also provides both "intuit" and "study" modes.

2.2.2. Scoring

The purpose of the back-end is to find the function which optimally classifies the training data. This optimization is expressed in terms of a score, which is a problem-dependent quantity. For binary classification, it is usually expressed in terms of a detection rate and a false alarm rate – following the scheme used by GENIE, we typically use a linear combination that assigns 500 points to each. That is:

$$\text{score} = 500\,(\text{detectionRate} + (1 - \text{falseAlarmRate}))\,. \tag{2}$$

This equation has to be modified for multi-class classification and for regression problems. It is sometimes useful to include some kind of penalty in the score that inhibits more complex solutions, and therefore produces solutions that are more likely to generalize well. In previous work, we investigated variants that use cross-validation score for fitness and multi-class classification.[1, 20]

3. SAMPLING AND COMPUTATIONAL ISSUES

One issue with using remotely sensed imagery is that the number of samples is typically large. It is not uncommon for an image to have several hundred thousand marked-up pixels. The computational burden of quadratic programming may increase quadratically as the number of training points increases. An alternative is to train on a small sample of the training points but this also is problematic because there is a danger of choosing an unrepresentative sample. AFREET resamples every N iterations to ensure that eventually most, if not all, of the data is used in training. This may require that all candidate solutions in a population be re-evaluated. Resampling may also adversely effect front-end optimization performance. We found that bagging was a good way to make effective use of as much training data as possible and reduce significantly the computation needed by the quadratic program.

3.1. Bagging

Although "bagging" was first introduced[21] as a variance-reduction technique for improving generalization ability of classifiers, we use it here for two reasons. One is variance-reduction, but the other is that it enables us to make separate runs with small subsets of the data; this reduces the computational burden, both in terms of computational effort (which often scales superlinearly with data set size) and computer memory (which is often fixed on any given machine).

In traditional bagging, a data set of size N is resampled with replacement to produce a new data set of size N. For each resampled data set, a separate classifier is trained. These classifiers are combined by majority vote. In our variant, the original data set of size N is resampled to produce a much smaller data set, with $n \ll N$ data points. As before, separate classifiers are trained on the smaller data sets, and again the classifiers are combined by majority vote.

In experimenting with ZEUS with bagging turned on, it was not uncommon to choose a bagging arrangement with j bags of size k points where $jk \geq N$. The advantage is that most of the training data is used while the computation time is considerably less than when training a single Support Vector Machine (SVM) with a sample size of jk. The flip-side is the technique may result in a solution which overall is less accurate than a solution obtained by training a SVM once on a large training set. In adjusting the bagging parameters, a trade-off between computational efficiency and accuracy should be considered. While undoubtedly there are computational benefits in bagged SVMs, more research is needed to determine how the bagging parameters effect generalization performance.

One advantage of using linear kernels $K(\mathbf{x}, \mathbf{y}) = \mathbf{x}\mathbf{y}^T$ when bagging is that the decision function $f(\mathbf{x}) : \mathbb{R}^p \rightarrow \{-1, 1\}$ can be expressed in terms of a weight vector $\mathbf{u} = \sum_{m=1}^{M} \alpha_i y_i \mathbf{h}_m$

$$f(\mathbf{x}) = sgn(\mathbf{x}\mathbf{u}^T - b) \tag{3}$$

where $\mathbf{h}_m$ are the support vectors, α_m are the Lagrange coefficients, $y_i \in \{-1, 1\}$ are the labels of the support vectors, and b is the bias. The model may be applied to each point with just a single dot product or a set of points by doing just matrix multiplication. Thus, our approach uses bagging to generate piecewise linear models to lessen the computational burden of training and exploitation.

In addition, ensuring there are an equal number of patterns from each class in each bag may help accommodate an unbalanced data set while maximizing the use of as much training data as possible.

Currently, bagging is the only ensemble-based method that is implemented. Future plans include a boosting mode,[22, 23] and a weighted order statistic classifier mode.[24]

3.2. Fast Polynomial Kernel Evaluation Using the Support Scalar Technique

The function $K(\mathbf{x},\mathbf{y}) = (\gamma\mathbf{x}\mathbf{y} + c)^d$ is known as a polynomial kernel, and when used with an SVM, it permits more complex models to be produced than a linear SVM. Evaluating models with polynomial kernels can be very expensive in both training and testing if there are a large number of support vectors. It is not uncommon to exploit a model on a remotely sensed image with millions of pixels. Exploitation, can therefore, be extremely slow when using non-linear SVM models with a large number of support vectors.

We found that explicitly expanding polynomial kernels may be more efficient, in some scenarios, than using the kernel trick. It has been suggested that such cases are only useful for "toy" examples.[19] However, we found explicit expansions useful in select cases.

Many satellite imagery data sets are represented by a few bands. Landsat-7 imagery, for example, is represented by seven spectral channels, and the images explored in this study consisted of only ten bands. Thus, the dimensionality of the pixel inputs for many satellite data sets is modest. If a model could be produced that achieves adequate performance with a low degree then explicit expansions might reduce the computational burden of exploiting complex SVM models (i.e. models with large numbers of support vectors.)

We describe an efficient algorithm for exploiting a Support Vector Machine model with a polynomial kernel and suggest conditions for its efficiency. It involves explicitly performing binomial expansions to evaluate the kernel.

The decision function of a support vector machine with a polynomial kernel is

$$f_p(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m y_m (\gamma \mathbf{x}\mathbf{h}_m^T + c)^d \tag{4}$$

where c is the "coefficient" of the kernel and γ is a constant multiplier, α_m are the non-zero Lagrange multipliers, y_m are the labels of the support vectors, $\mathbf{h}_m$ are the support vectors, the signum of $f_p(\mathbf{x})$ is the predicted label of the point to exploit $\mathbf{x}$, and M is the total number of support vectors.

Our algorithm is inspired from the binomial expansion of the polynomial kernel. We thus define the decision function with its kernel expressed in terms of such an expansion:

$$g(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m y_m (\gamma \mathbf{x}\mathbf{h}_i^T + c)^d = \sum_{m=1}^{M} \alpha_m y_m \left(\sum_{n=0}^{d} \binom{d}{n} (\gamma \mathbf{x}\mathbf{h}_m^T)^n c^{d-n} \right). \tag{5}$$

We augment the point-to-exploit $\mathbf{x} \leftarrow [\mathbf{x}|1]$ and each support vector $\mathbf{h}_m \leftarrow [\mathbf{h}_m|c]$ to account for the additive coefficient which may be ignored when it is zero. We define a $\binom{p+d}{d-1} \times d$ matrix $B = [\mathbf{b}_1^T, \mathbf{b}_2^T, \ldots, \mathbf{b}_n^T]^T$ of exponent specifications $\mathbf{b}_i$,

$$\mathbf{b}_1 = [0, 1, \ldots, d] \text{ and } \mathbf{b}_i \in \{0, 1, \ldots, d\}^{p+1} \tag{6}$$

subject to the constraints that there are no duplicate row vectors in B and the sum of the exponent specifications for each row vector equals the degree of the polynomial,

$$\forall j, 1 < j < i, \mathbf{b}_i \neq \mathbf{b}_j \text{ and } \sum_{j=1}^{p+1} b_{ij} = d. \tag{7}$$

The binomial expansion of the decision function is rearranged and reduced,

$$g(\mathbf{x}) = \sum_{i=1}^{\binom{p+d}{d-1}} \left(\prod_{j=1}^{p+1} x_j^{b_{ij}} \right) \left(\binom{d}{\mathbf{b}_i} \sum_{m=1}^{M} \alpha_m y_m \prod_{j=1}^{p+1} h_{mj}^{b_{ij}} \right) = \sum_{i=1}^{\binom{p+d}{d-1}} s_i \prod_{j=1}^{p+1} x_j^{b_{ij}} \tag{8}$$

where

$$s_i = \binom{d}{\mathbf{b}_i} \sum_{m=1}^{M} \alpha_m y_m \prod_{j=1}^{p+1} h_{mj}^{b_{ij}} \tag{9}$$

and is known as a *support scalar* and is calculated independently of $\mathbf{x}$. The polynomial is exploited using the support scalar method in approximately $O(\binom{p+d}{d-1}(p+1))$ time. The model of the classifier consists of the exponent matrix B, the support scalars $\mathbf{s} = \left\{s_1, s_2, \ldots, s_{\binom{p+d}{d-1}}\right\}$, and the bias b. The number of support scalars $N_s = \binom{p+d}{d-1}$ is independent of the number of data points or support vectors. Thus, given sufficiently small input dimension, small degree, and model complexity; this approach may be more efficient than the kernel trick.

The computation of B and s_i is greatly optimized by using MATLAB's vectorization constructs. We recommend looking at our source code to better visualize such optimization. The source code for this approach to kernel evaluation is available for download, `http://nis-www.lanl.gov/∼eads/fpoly`.

4. EXPERIMENTAL DESIGN

4.1. Experimental Design

We tested ZEUS on a pixel classification task that was first addressed by Harvey *et al.*[6] This consisted of twelve images – three images for each of four classes. The classes were: roads, golf courses, urban areas, and clouds. The task was to identify the parts of the images in which these features were present. For the in-sample investigation, single images were used, and the reported in-sample scores are the averages over the three images considered. The overall average was the average over each of the four tasks. For out-of-sample score, these were obtained by training on single images, and testing on the other two. Again, the reported score was the average over the three ways this could be done.

To do this comparison, we used ZEUS in two modes: intuit, and intuit with bagging. We also used AFREET in intuit mode, GENIE, and a suite of classifiers that are available as part of the ENVI package..[25] We also considered a straight Fisher discriminant, and a Fisher discriminant with ridge regularization.

Each of the experiments below was conducted using each of the twelve input images. A model is produced for every image and classification method.

Experiment 1: ZEUS Intuit Mode The following parameters were used:

- error penalty (C values) set: $R = \{10^i | i \in \{-2, -1, 0, 1, \ldots, 5\}\}$,
- kernel set: $S = (\text{linear}, \text{RBF})$, and
- parameter set: $P = R \times S$.

A sample of 5000 marked pixels from each class was chosen without replacement from the training image. If the number of pixels for a particular class was less than 5000 pixels, all of the pixels were chosen. This approach was taken to avoid class weighting to account for unbalanced datasets. It also reduces the computational burden of cross validation. Three-fold cross validation was performed on a SVM using each element in the parameter set P. The final kernel and error penalty is chosen according to the best three-fold cross validation score found. Finally, a SVM is trained using the final parameters, and a final model is produced for further testing.

Experiment 2: ZEUS Intuit Bagging Mode The following parameters were used:

- error penalty (C values) set: $R = \{10^i | i \in \{-2, -1, 0, 1, \ldots, 5\}\}$,
- kernel set: $S = (\text{linear})$,
- bag size set: $B_s = \{2^i | i \in \{2, 3, \ldots, 14\}\}$,
- bag count set: $B_n = \{2^i | i \in \{2, 3, \ldots, 14\}\}$,
- bagging parameter set: $B = B_s \times B_n$, and

- SVM parameter set: $P = S \times R$

In this experiment, we used bagging with an equal number of samples from each class. As mentioned in section 3.1, bagging was employed to: reduce the computational burden of training a single model on a large number of points, make effective use of as much training data as possible, and to account for unbalanced data sets.

For each bagging parameter in the set B, the C value is adjusted based on the three-fold cross validation score of the bagged SVM. The C values are global to all bags although adjusting the C values local to each bag deserves further investigation. The final model is chosen according to the best out-of-sample score.

Experiment 3: ZEUS Study Mode For ZEUS study, we prepared several configuration files, choosing parameters based on problem difficulty. Problems with images of varying difficulty had two to three parameter sets. It is important to note that the number of parameters tried for Experiment 3 is significantly less than in Experiment 2. Table 3 lists the parameters used.

Problem	C	Bag Sizes	Bag Count	Depth	Pop. Size	Iterations
Roads	1000	512, 5000	16 bags	3 planes	20 chr.'s	200
Golf	100, 1000	512	16, 32 bags	3 planes	20 chr.'s	200
Urban	100, 1000	2000, 5000	8, 32 bags	3 planes	20 chr.'s	500
Clouds	1000	512	16 bags	3 planes	20 chr.'s	200

Table 3. ZEUS Study Parameters: The parameters used for each of the problems, chosen arbitrarily.

Experiment 4: AFREET Intuit Mode AFREET Intuit was trained using the default parameters: linear kernel and a C value of 100). Unlike experiment 1, no adjustment of parameters is made and all labeled input pixels are used for training.

Experiment 5: GENIE Refer to Harvey *et al* for a comprehensive explanation of the GENIE software and the experiments performed on this data set.[6]

Experiment 6: Fisher Discriminant A Fisher discriminant was computed for each of the training sets using a straight Fisher discriminant and a Fisher discriminant with ridge regularization. The effect of the ridge regularization is to replace the covariance matrix $\mathbf{K}$ with a version $\mathbf{K}' = \mathbf{K} + \epsilon\mathbf{I}$ where ϵ is a small scalar regularization parameter, and $\mathbf{I}$ is the identity matrix. This effectively penalizes complexity of the solution.

Experiment 7: ENVI In Harvey *et al*, several built-in ENVI classifiers were compared against GENIE using this data set.[6] They include: Spectral Angle Mapper, Minimum Distance, Fisher Discriminant, Maximum Likelihood, Binary Encoding, and Mahalanobis Distance.

5. RESULTS

5.1. In-sample Performance Comparison

We see that on average, GENIE scored the best in-sample despite being fourth in the out-of-sample rankings. This could be due to overfitting or convergence to a solution. Interestingly, despite performing the best out-of-sample, ZEUS Intuit with bagging is the second worst performer in-sample.

Learner	Roads	Golf	Urban	Clouds	Average
Zeus Study Mode	931.6 (3)	971.9 (3)	971.0 (3)	999.3 (2)	968.5 (3)
Zeus Intuit Mode	956.4 (2)	977.3 (2)	972.9 (2)	998.3 (3)	976.3 (2)
Zeus Intuit Bagging Mode	912.2 (4)	959.8 (6)	908.5 (7)	985.4 (7)	941.5 (6)
genie	963.3 (1)	998.3 (1)	998.9 (1)	999.9 (1)	990.0 (1)
afreet, Intuit Mode	886.6 (5)	960.4 (5)	951.6 (5)	997.0 (4)	948.9 (4)
Fisher Discriminant	883.3 (6)	965.4 (4)	953.7 (4)	987.9 (6)	947.6 (5)
Fisher with Ridge Regularization	876.7 (7)	951.7 (7)	942.9 (6)	992.4 (5)	940.9 (7)

Table 4. In-Sample Performance: The average in-sample performance and rank for each learner is listed for all of the four problem types. The average performance is obtained by averaging the in-sample fitnesses for each problem.

5.2. Out-of-sample Performance

We see that on average, zeus intuit with bagging performed better out-of-sample than any of the other alter1natives. One reason for this is that the parameters for the bagging technique were carefully chosen. Unexpectedly, zeus study performed worse than zeus intuit with bagging. However, it should be noted that the parameters were rarely, if at all, adjusted for the zeus study experiments.

Afreet intuit performs better than zeus intuit but worse than zeus intuit bagging. One reason is the sample it trains with includes all labeled points rather than a small subset of them.

Learner	Roads	Golf	Urban	Clouds	Average
Zeus Study Mode	746.7 (3)	888.1 (1)	685.0 (5)	993.1 (2)	828.2 (2)
Zeus Intuit Mode	638.5 (6)	783.6 (6)	623.0 (7)	808.4 (6)	713.4 (6)
Zeus Intuit Bagging Mode	835.4 (1)	882.1 (2)	744.6 (4)	976.8 (5)	859.7 (1)
genie	763.2 (2)	739.8 (7)	813.5 (1)	978.0 (4)	823.6 (4)
afreet Intuit Mode	730.8 (4)	820.6 (4)	764.1 (2)	993.6 (1)	827.3 (3)
Fisher Discriminant	541.5 (7)	800.4 (5)	641.7 (6)	704.3 (7)	672.0 (7)
Fisher with Ridge Regularization	696.0 (5)	828.0 (3)	745.4 (3)	986.7 (3)	814.0 (5)

Table 5. Out-of-Sample Performance: The out-of-sample performance and rank for each learner is listed for all of the four problem types.

6. CONCLUSIONS AND FUTURE WORK

We presented a multimodal approach to feature extraction for image and signal learning problems. Our extension of zeus to the pixel classifier domain was compared against other approaches such as genie, afreet, and envi. We considered techniques such as bagging and fast polynomial kernel evaluation to improve the computational efficiency of the pixel classifier while making effective use of as much training data as possible.

In the future, we will continue to investigate the use of ensemble techniques with Support Vector Machines to improve computational performance and accuracy. The image classification and time series forecasting modules will be implemented in the hope of further demonstrating the flexibility of our architecture.

ACKNOWLEDGMENTS

We are grateful to the ISIS team for many useful discussions. This work was supported in part by a Los Alamos Laboratory Directed Research and Development award, and is performed under the auspices of the United States Department of Energy.

Out-of-Sample Rank	**In-Sample Rank**
1. Zeus Intuit Bagging Mode (859)	1. genie (990)
2. Zeus Study Mode (828)	2. Zeus Intuit Mode (976)
3. afreet Intuit Mode (827)	3. Zeus Study Mode (969)
4. genie (823)	4. Maximum Likelihood* (962)
5. Fisher with Ridge Regularization (814)	5. afreet Intuit Mode (948)
6. Zeus Intuit Mode (713)	6. Fisher (947)
7. Spectral Angle Mapper* (684)	7. Zeus Intuit Bagging Mode (941)
8. Minimum Distance* (684)	8. Fisher with Ridge Regularization (940)
9. Fisher Discriminant (672)	9. Mahalanobis Distance* (905)
10. Maximum Likelihood* (608)	10. Minimum Distance* (849)
11. Binary Encoding* (602)	11. Spectral Angle Mapper* (833)
12. Mahalanobis Distance* (536)	12. Binary Encoding* (703)

Table 6. Classifier Rankings: The in-sample and out-of-sample ranks are listed in ascending order. genie performed the best in-sample while zeus Intuit Bagging mode was the best performer out-of-sample. (An asterisk indicates the result was obtained using a built-in ENVI classifier.)

REFERENCES

1. D. Eads, D. Hill, S. Davis, S. Perkins, J. Ma, R. Porter, and J. Theiler, "Genetic algorithms and support vector machines for time series classification," *Proc. SPIE* **4787**, pp. 74–85, 2002.
2. The Forté project. `http://nis-www.lanl.gov/nis-projects/forte`.
3. J. Bloch, S. Brumby, N. David, D. Esch-Mosher, M. Galassi, N. Harvey, C. Novak, S. Perkins, R. Porter, S. Szymanski, and J. Theiler, "Intelligent Searching of Images and Signals (ISIS)." `http://www.isis.lanl.gov`.
4. J. Theiler, N. R. Harvey, S. P. Brumby, J. J. Szymanski, S. Alferink, S. J. Perkins, R. B. Porter, and J. J. Bloch, "Evolving retrieval algorithms with a genetic programming scheme," *Proc. SPIE* **3753**, pp. 416–425, 1999.
5. S. P. Brumby, J. Theiler, S. Perkins, N. Harvey, J. J. Szymanski, J. J. Bloch, and M. Mitchell, "Investigation of image feature extraction by a genetic algorithm," in *Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation II*, B. Bosacchi, D. B. Fogel, and J. C. Bezdek, eds., *Proc SPIE* **3812**, 1999.
6. N. R. Harvey, J. Theiler, S. P. Brumby, S. Perkins, J. J. Szymanski, J. J. Bloch, R. B. Porter, M. Galassi, and A. C. Young, "Comparison of genie and conventional supervised classifiers for multispectral image feature extraction," *IEEE Trans. Geosci. and Remote Sens.* **40**, pp. 393–404, 2002.
7. S. Perkins, N. Harvey, S. Brumby, and K. Lacker, "Support vector machines for broad area feature extraction in remotely sensed images," *Proc. SPIE* **4381**, pp. 286–295, 2001.
8. D. Eads, "MFITSIO website." `http://nis-www.lanl.gov/∼eads/mfitsio/`.
9. J. Ma, Y. Zhao, S. Ahalt, and D. Eads, "The Ohio State University Support Vector Machine Package." `http://svm.sf.net`.
10. D. Eads, "Zeus: Multidimensional signal analysis and understanding." `http://www.zeus.lanl.gov`.
11. D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation* **1**, pp. 67–82, April 1997.
12. J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, MA, 1992.

13. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction : On the Automatic Evolution of Computer Programs and Its Applications*, Morgan Kaufmann, San Mateo, CA, 1997.
14. D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison Wesley, Reading, MA, 1989.
15. C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery* **2**(2), pp. 121–167, 1998.
16. D. Hush and C. Scovel, "Polynomial-time decomposition algorithms for support vector machines," *Machine Learning* **51**, pp. 51–71, 2003.
17. C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning* **20**(3), pp. 273–297, 1995.
18. V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, NY, 1996.
19. B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press, Cambridge, MA, 2002.
20. N. R. Harvey, J. Theiler, L. Balick, P. Pope, J. J. Szymanski, S. J. Perkins, R. B. Porter, S. P. Brumby, J. J. Bloch, N. A. David, and M. Galassi, "Automated simultaneous multiple feature classification of MTI data," *Proc. SPIE* **4725**, pp. 346–356, 2002.
21. L. Breiman, "Bagging predictors," *Machine Learning* **26**, pp. 123–140, 1996.
22. R. E. Schapire, "The strength of weak learnability," *Machine Learning* **5**, pp. 197–227, 1990.
23. Y. Freund, "Boosting a weak learning algorithm by majority," *Information and Computation* **121**, pp. 256–285, 1995.
24. R. Porter, D. Eads, D. Hush, and J. Theiler, "Weighted order statistic classifiers with large rank-order margin," in *Proceedings of the Twentieth International Conference on Machine Learning*, 2003. To appear.
25. Research Systems, Inc. `http://www.rsinc.com/envi`.