Department of Computer Science

Classification of EEG Signals Using a Sparse Polynomial Builder

Edward S. Orosz and Charles W. Anderson

Technical Report CS-94-111

April 27, 1994

Colorado State University

Classification of EEG Signals Using a Sparse Polynomial Builder

Edward S. Orosz *, Charles W. Anderson †

Abstract

Sutton and Matheus' algorithm for sparse polynomial construction is used to construct a classifier to recognize a mental task from recorded EEG signals. Data used is from Keirn [Keirn 88]. Since the data set is very limited, the classification accuracy was calculated by the *jack knife* or *loom* method of cross validation. Results indicate that the classifier does not generalize well on untrained data; accuracy averaged only 65%. This was believed to be the result of having too many features, initially causing over fitting. Since the algorithm starts with the features and only add news ones, initial over fitting remains. To overcome this limitation, the polynomial was modified to have zero features initially. On average the classification accuracy was unimproved, although most subjects' accuracy was $\approx$ 10 to 20% higher. The low accuracy is speculated to be the result of bad feature selection and the algorithm's susceptibility to noisy output and incomplete sampling of patterns. One result of this project is a reusable and modular set of C++ classes for training classifiers based on Sutton and Matheus' method.

*Colorado State University, email: orosze@cs.colostate.edu
†Colorado State University, email: anderson@cs.colostate.edu

Contents

1 INTRODUCTION

Humanity has always dreamt of controlling its world with the mind. Popular literature abounds with heros and villains who are able to hurl objects just by concentrating with their minds. While the domain of telekinesis is likely to be in the realm of fiction for a long time to come, research into new methods of controlling computers has been very active — ranging from a simple mouse to virtual reality gloves with tactile sensors.

In pursuit of a new machine interface, Keirn [Keirn 88] and Aunon [Keirn and Aunon 90] attempted to determine the mental task a subject was performing by examining and processing the recorded electroencephalogram (EEG) signals. Classifiers were built to output the corresponding task using only the inputed signal. The performance of a classifier was measured using classification accuracy — the percentage of correct outputs using testing data. The classifier has no previous experience with the testing data. This is in direct contrast to training data, which is utilized in the creation of the classifier. Properties of the data which are believe to signify and identify the task are called *features*. Classifiers typically operate on features instead of the whole data set. Keirn achieved results of up to 100 percent classification accuracy using two tasks and one subject. The study had low accuracy (80%) on some subject/tasks pairs.

There were some shortcomings in Keirn's study. The classification was attempted with only simple Bayes Quadratic classifiers. The amount of data used was small, with just seven subjects and 85 total examples per task. The classifier required a lot of preprocessing, making real time response difficult. Finally, features had to be picked by "hand", a laborious and time consuming task. Features are propertied of the data which are believe to signified and identified the item

The next section summarizes Keirn's work. Presented next is the data taken from Keirn's experiment, the feature selection algorithm [Sutton and Matheus 91] used, and the *loom* method employed to determine classification accuracy. The accuracy achieved for each subject is summarized and compared to Keirn's results. Following that summary is a description of the modification made to the initial algorithm in order to improve accuracy on the EEG data and and a discussion of the results obtained with the new algorithm. The paper concludes with comments and observations.

2 PREVIOUS WORK

In Keirn's, [Keirn 88], study subjects' EEG were recorded while they performed the following mental tasks:

Baseline: The subjects were told to think of "nothing in particular".

Letter Composing: The subject composed a letter to a friend or relative without vocalizing. When the task was repeated, the subject was instructed to continue with the previous letter.

Geometric Figure Rotation: The subject was given a three dimensional block figure and thirty seconds to study it. The subject was then to mentally rotate the figure about any axis.

Mental Arithmetic: The subject was told to multiply two non-trivial numbers and not to expect to finish.

Visual Counting: The subject visualized numbers being written in sequential order on a blackboard. When the task was repeated, the subject was to continue from the previous stopping point.

Standard signal processing techniques were applied to the signal in order to extract features. These features were used in a Bayes Quadratic classifier to distinguish between a pair of tasks from a single subject.

There has been a fair amount of research done on processing EEG signals, but most of that has involved the use of "invoked potentials". An external stimulus, such as a flashing light or an audible noise, etc., triggers the signal. Attempts like Keirn's to classify a signal that has been internally created solely from the subject's mental processing are much rarer. One such effort used Kohonen nets to classify the signals [Shiao-Lin, et al. 93]. They recorded new data, but the tasks and features used were based on Keirn's work.

2.1 Data

Measurements were taken from six sensors located on the scalp. The standard "10-20 system" [Jasper 58] of electrode placement was used (see figure 1). Locations selected were C3, C4, P3, P4, O1 and O2. Data was collected for ten seconds. The sampling rate was 250 samples/sec resulting in 2500 floating point numbers in the approximate range -50.0 to $+50.0$ microvolts per task. The subjects were instructed to keep their eyes open (using only normal concentration, no extra effort) for one trial. For the second trial, they were told to keep their eyes closed. Thus the data was divided into "eyes closed" and "eyes open" sets.

The spectral density curve for each sensor was calculated by the Burg and Wiener-Khinchine methods [Keirn 88]. The curve shows the energy at a particular frequency (see figure 3). For each of the six channels, the area under the curve was computed at four common frequency bands of the brain - delta (0-3 Hz), theta (4-7 Hz), alpha (8-13 Hz), and beta (14-20 Hz). This resulted in 24 power values.

In addition, asymmetry ratios were calculated for each combination of right and left hemisphere leads, given by

$$(R-L)/(R+L)$$

where R is energy in a frequency band of a right hemisphere lead (O1, P4, or C4) and L the energy for a left hemisphere lead (O2, P3, or C3). This was done for all four frequency bands, giving $9*4=36$ ratios. Combining the 24 power values and the 36 asymmetry ratios yields 60 total features for a task.

These features were calculated for both two second and quarter second data segments, chosen as close to the "middle" of the task as possible. Keirn reasoned that at the beginning the subject had not "settled" into the task; while towards the end the subject might become bored and tired. The preferred starting point was just after the first 1.5 to 2 seconds. Only eye blink free data segments were used. Keirn felt that the spikes and noise in the six electrodes caused by the eye blinks would be too much of a hindrance to classification.

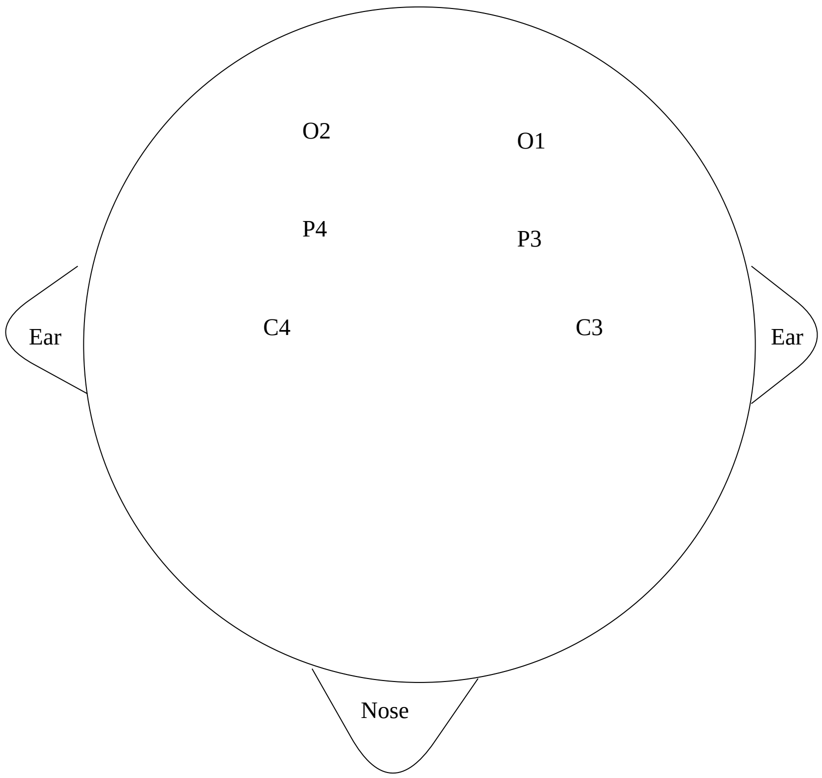

Figure 1: Top view of head showing the locations of the 6 electrodes used

2.1.1 Raw Data

Figure 2 plots 512 data points (2 seconds) of data from electrode C3, recorded while the subject was performing the "math" task. The number of points has been rounded to the nearest power of two — to make the calculation of the fast Fourier transform (fft) easier. Data from other channels/tasks is very similar.

2.1.2 Spectral Density Curve

Figure 3 shows the spectral density curve of the data shown in figure 2. Note that little energy exists above ≈ 25 Hz. Most research, including Keirn's, only examines frequencies from 0 to 20 Hz.

2.1.3 Features

Figure 4 shows the 60 Wiener-Khinchine features for the data segment shown in figure 2. The first 36 numbers are the ratios. Features 1 — 9 are from the delta frequency band, 10 — 18 are from the theta band, 19 — 27 from alpha and 28 — 36 from beta. The next four sets of six numbers (37 — 42, 43 — 48, etc.) are the energies in the frequency bands, starting with delta, followed by theta and alpha and ending with beta. The order of the electrodes in each set is the same: O1, O2, P3, P4, C3 and C4.

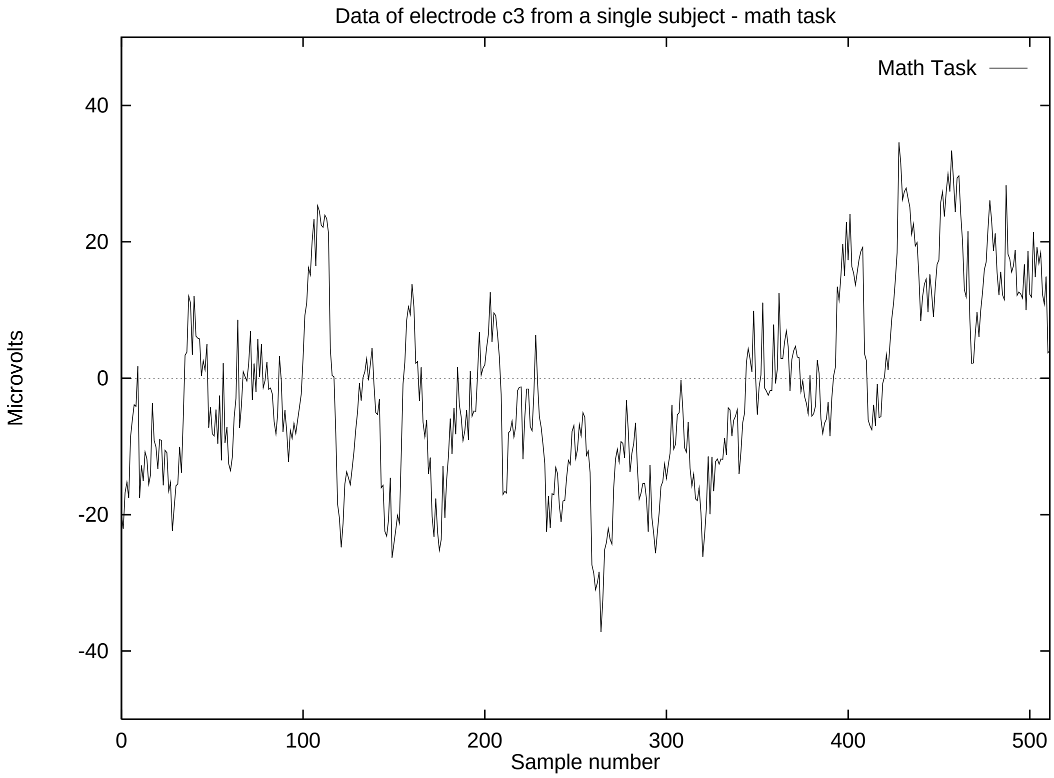

Figure 2: Raw data from electrode C3 and the math task

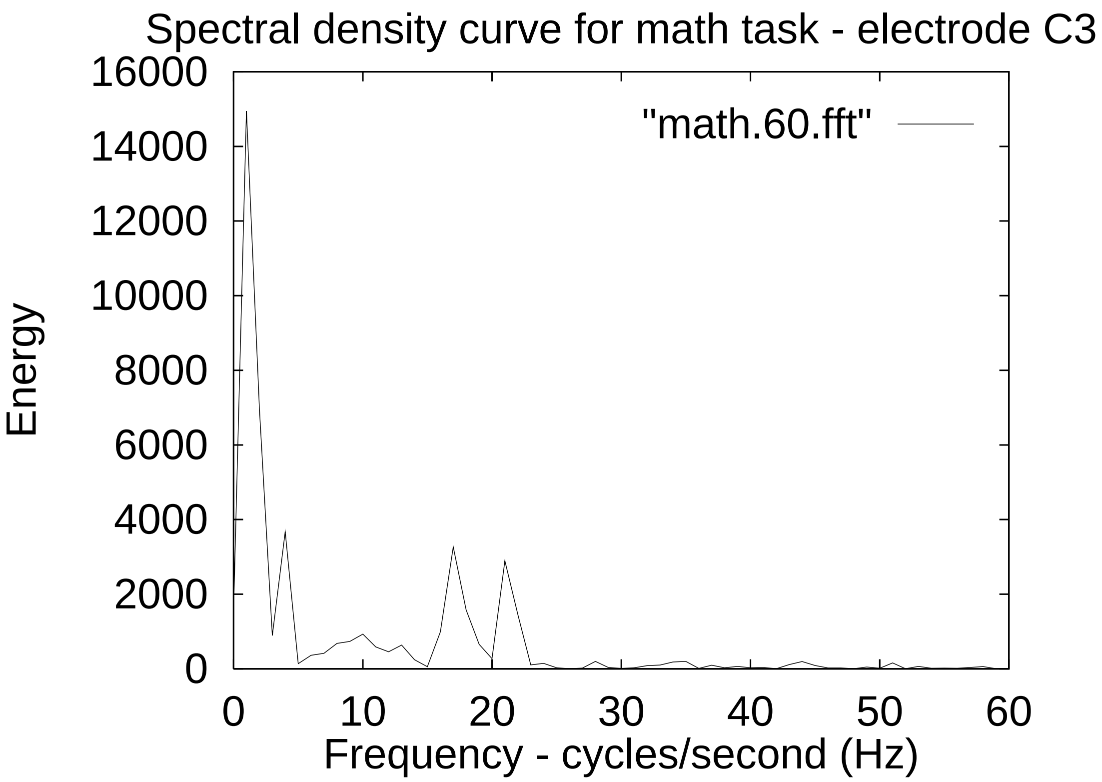

Figure 3: Fft of data from figure 2

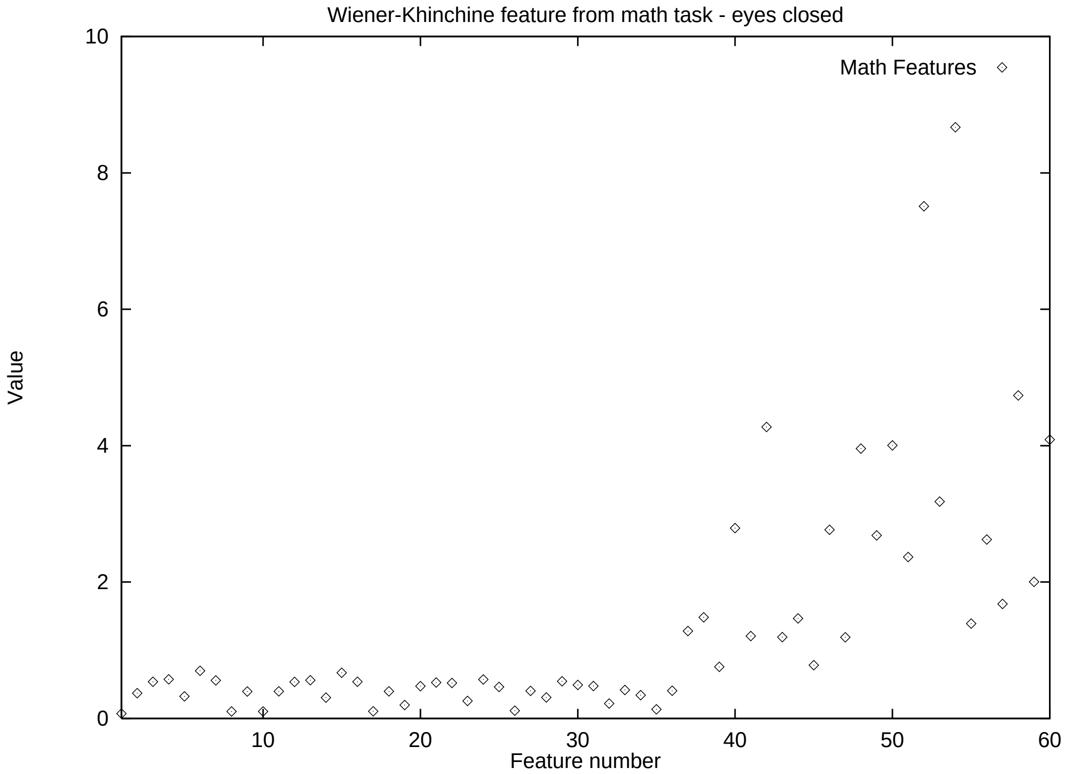

Figure 4: Wiener-Khinchine features from a subject's "math" task

Subject	1	3	4	5	6
Average	92%	86.3%	95%	84.7%	91.8%

Table 1: *Classification accuracy on task pairs*
[Keirn 88]

2.2 Keirn's Results

In general, Keirn achieved high classification rates, even 100%, for several subjects and task pairs. Table 1 gives the average classification Keirn achieved (using 20 different task pairs) with the Wiener-Khinchine method for feature creation.

For the task pair used in this report, the feature Keirn used was the ratio with p3 and p4 (alpha band) [Keirn 88]. However, Keirn stated that for most task pairs, there were "several" other feature combinations yielding the same accuracy. No particular set stood out. For subject 3 with Wiener-Khinchine features and 10 second data segments, the feature set was different for the first session, second session, and trials combined from both sessions.

3 OBJECTIVES

The problem this project addressed was feature selection. The Sutton and Matheus algorithm was chosen as a novel and untried procedure of feature selection [Sutton and Matheus 91].

Feature selection is the process of extracting from the data the information necessary to perform the classification. The information, the features, is used to determine the classification. The problem is to find a set small enough to train the classifier, without throwing away needed information. Since the search space of potential features is often huge, efficient methods of finding features is also an issue.

Since the algorithm was unfamiliar, much refinement and "tuning" was inevitable. A modular and reusable implementation was sought. The implementation language selected was C++ with the aim of using object oriented language features, while leveraging existing C libraries and expertise.

In the previous study, Keirn found features by "hand". Since the classifier from Keirn was trained and operated on a single subject, automatic features selection was deemed a useful and important component to have in the system. It was hoped that automatic feature selection would result in better classification.

Therefore, the main objectives of the project were to:

1 Develop modular, reusable C++ code for training classifiers based on Sutton and Matheus' method.

2 Apply the implementation to Keirn's EEG problem and compare results.

4 METHOD

4.1 Data

The data used was from Keirn's earlier study. The features obtained using the Wiener-Khinchine method (two second data segments) to calculate the spectral density curve were used for this project. Each classifier was trained and tested on data from all sessions of one subject. For this project, the tasks focused on were "math" and "letter" (eyes closed).

4.2 Sutton's Algorithm

Sutton and Matheus' algorithm [Sutton and Matheus 91, Sanger, et al. 92] for sparse polynomial construction was used on the EEG features in order to find the features that could be used in a linear discriminant classifier. The algorithm was designed to construct polynomials in the presence of irrelevant inputs. It proposed a solution to the "combining" problem: Given that x_1 and x_2 are relevant terms, how do you combine them (x_1x_1, x_2x_2, $x_1x_2x_2$, etc.)? Results from Keirn (1988) indicated that only one or two of the 60 features might be all that is needed to perform the classification. This suggested that the EEG classifier was a similar problem — a large set of potential features, with a small set that needs to be combined to make the terms for the pattern classifier.

4.2.1 The Algorithm

In order to better understand the algorithm, a few terms and procedures will be explained, before it is presented. $Normalize(x)$ replaces each element (x_i) of the 1-dimensional array x by $(x_i - mean)/std$, where $mean$ is the arithmetic mean of x and std is the standard deviation. The resulting array has a mean of 0 and a standard deviation of 1. $Regress(y|x)$ performs Least Mean Squared (LMS) linear regression. X, x and $Xsquared$ in the description are 2-dimensional arrays. $x[*][i]$ is the i^{th} column of x. Y, y and $SquaredError$ are 1-dimensional arrays. $Y[i]$ is the i^{th} element of y.

The following is a summary of the algorithm from [Sutton and Matheus 91]. This description corrects errors and typos made in their paper, plus adds a few comments.

Step 0: *initialize variables*
- I = number of instances (training examples)
- n = number of features (dimension of the input)
- for $j = 1$ to n do
  - $X[*][j] =$ Normalize$(x[1][j] \ldots x[I][j])$ ($x[i][j]$ is j^{th} feature of pattern i)
  - $Xsquared[*][i] =$ Normalize$(X[i][1]^2 \ldots X[i][I]^2)$
- $Y =$ Normalize$(y[1] \ldots y[I])$ ($y[j]$ is the correct classification for pattern j)

Step 1: *do the regression*
- $w =$ Regress$(Y|X)$
- for $i = 1$ to I do
  - $SquaredError[i] = ((\sum_{j=1}^{n} w[j] * X[i][j]) - Y[i])^2$
- if $\sum_{i=1}^{I} SquaredError[i] \approx 0$ return results and stop

Step 2: *construct new feature*
- $SquaredError_n =$ Normalize$(SquaredError[1] \ldots SquaredError[I])$ [1]
- $p[*] =$ Regress$(SquaredError_n|Xsquared)$ ($p =$ potentials, used to rate features.)
- $index1, index2 =$ select_new_feature(p) (pick the 2 features with the highest ranking)
- for $i = 1$ to I do
  - $x[i][n+1] = x[index1] * x[index2]$
- $X[*][n+1] =$ Normalize$(x[1][n+1], x[2][n+1], \ldots, x[I-1][n+1], x[I][n+1])$
- $Xsquared[*][n+1] =$ Normalize$(X[1][n+1]^2, X[2][n+1]^2, \ldots, X[I-1][n+1]^2, X[I][n+1]^2)$
- $n = n + 1$
- GOTO step 1

The *select_new_feature* is shorthand for the methods Sutton and Matheus (1991) employed. They were "potentials" and "joint potentials". The potentials are the coefficients from a regression of the squared error against the squared features. The potentials approach just creates the new feature as the product of the two features with largest potentials. The product can use just a single feature (i.e. $x_1 x_1$). If the new feature is already in the polynomial, the next highest pair is used. The joint method makes m "joints" (potential new

[1] This step was omitted in the [Sutton and Matheus 91] paper.

features) as the product of two existing features. m is usually set to the number of initial features (n). The joints are selected based on the m highest products of the two constituent features' potentials. Another regression with the *normalized* SquaredError and *joints* is executed. The joint feature with the highest weight from the regression is the new feature added.

4.2.2 The How and Why

This section will briefly discuss ideas and theory used by Sutton and Matheus (1991) in the development of the technique, and "rounded out" with an example.

They start off by presenting a simple example function $y = 2x_1 + 5x_2x_3$. This function is not linear over x_1, x_2, x_3. However, it is a linear function $y = 2x_1 + 5x^{'}$ over the set $x1, x2, x3, x^{'}$ where $x^{'} = x_2x_3$, i.e., $y = 2x_1 + 5x^{'}$. The stumbling block with this approach is deciding which features to create and add. The search space grows exponentially with the number of operators, independent features and the order of the polynomial. The complexity is $c(n+1)^{\delta}$, where $c =$ number of operators, n the number of features, and δ the order of the polynomial.

In order to restrict the complexity, the method was restricted to generating polynomials. This restriction means that multiplication is the only operator needed. A second constraint is the binary use of multiplication. Only two features can be combined in a single "step". The process iterates until the error is less than a threshold value. To make $x_3x_4x_5$ for example, a new feature $x^{'} = x_4x_5$ is added, followed by $x^{''} = x^{'}x_3$. The restriction to polynomial functions was viewed as using "domain knowledge".

This reduces the complexity to $\frac{n^2}{2}$, but the practicality of trying all the possibilities is questionable. Instead, features are rated and the top two most "useful" ones are used to form the product. The rating method developed by Sutton and Matheus was a modification of the method from Sanger [Sanger 91] which is based on the ability to predict the squared error.

The reasoning behind the rating method is explained here in an intuitive rather than formal way, along the lines of [Sutton and Matheus 91]. Viewing the polynomial as a single layered network which has been trained, some of the outputs (y) are predicted with a small error, others with a large error. The biggest changes in the weights (coefficients) are made when the squared error is high, the largest of those in the coefficients of the features with the largest magnitude at that cycle ($\triangle w_i = \alpha x_i(y^f(x) - y)$, where $y^f(x)$ is the correct output for input vector x). If, after the changes in the coefficients have stabilized, a feature x_i tends to have a high magnitude when the error is high, then a correlation between the feature and the error is suggested. This correlation implies that x_i contributes to the target function, but can not accurately by itself predict the output of the function.

The approach proposed by Sanger (1991) measures the ability of a feature to predict the squared error by calculating the correlation between the squared error and the square of the feature's value. The motivation for [Sutton and Matheus 91] method is that the correlation of x_i and x_ix_j is similar, resulting in the same terms being added. The algorithm becomes "stuck".

The modification done by [Sutton and Matheus 91] was to give the rating more of a competive quality that discourages new features that do not improve the prediction squared error. The competion is done by a regression of the squared error over the squared features ($Regess(e^2|x^2)$). The regression is an approximate measure of the correlation, with the "winner" (highest coefficient) the best predictor. This has been extended to a regression on features *before* they are added ("joint potentials").

The following example will demonstrate the algorithm. The function presented is the product of two variables ($y = x_1 x_2$). A third variable x_3 serves as a distracter. The input consists of 27 patterns: all combinations of x_1, x_2 and x_3 with the values of -1, 0 and 1. This is to ensure that the input space is completed sampled. The algorithm is able to successfully pair x_1 and x_2 in second cycle. Here is the output:

```
Cycle #1, Number of features: 3
Coeffs (Weights)
0*X1 + 0*X2 + 0*X3
Residual: 27
Potentials:
0.632444 0.632444 7.3586e-09
Joint potentials:
(0 0 0) (0 0 0) (0 0 0)

Cycle #2, Number of features: 4
Coeffs (Weights)
3.21018e-10*X1 + 2.94205e-10*X2 + 0*X3 + 0.999976*X1X2
Residual: 1.53477e-08
Potentials:
0.276815 0.276815 2.55251e-09 0.605421
Joint potentials:
(-0.210818 0 0) (0.333333 0 1) (-0.210818 1 1)

After 2 cycles: Joint potential Method  - No loom.
Number of features: 4
Coeffs (Weights)
2.6211e-10*X1 + 2.40217e-10*X2 + 0*X3 + 0.999976*X1X2
Constant: 0
Residual: 1.53477e-08
```

The first thing to note is that the weights from the linear model in cycle one are all 0. Ignoring the distracter (third input), the input space is 3 dimensional ($x_1, x_2, x_1 * x_2$). The points consists of the 9 combinations of x_1 and x_2 with the values of -1,0, and 1. The regression finds the "best fit line" (plane in 3d), such that the sum of the distances from the points to the plane is minimized. Two of the points lie in the plane $z = 1$, five in the plane $z = 0$, and two in $z = -1$. The plane with the closest distance to most of the points is $z = 0$. It sits half way between $z = -1$ and $z = 1$, thereby minimizing the distance to the rest of the points. So $z = 0$ is the best fit plane. The coefficients of 0 follow from the equation of a plane $z = a * x + b * y$, for $z = 0$.

The potentials of x_1 and x_2 in cycle one and two are equal. This makes intuitive sense — both x_1 and x_2 "contribute" equally to the output (y) in $y = x_1 * x_2$. This is especially true if x_1 and x_2 are equal. In this example, the distribution of points along x_1's axis and x_2's are identical, so the distance between them and the Squared error vectors are the same. The weights of the potentials regression against the squared error, will reflect that and hence be the same. In this example, the potentials of x_1 and x_2, will reflect the proportional relationship of x_1 and x_2 — if in the input the magnitudes of x_1 are increased and decreased for x_2 , there will be a corresponding increase or decrease in the potentials of x_1 and x_2.

The values of the potentials are used to pick which "joints" are tested. In this case, m is 3 (the initial number of features). The m features with highest product of its two constituent potentials are picked. This can be done by forming all the possible products, and finding the m highest. This approach is approximately $O(n^2)$. This can be done in $O(\ln n)$, but has little practical effect on speed, since the vast majority of the time is spent in the regression routine. The joints chosen are x_1x_1, x_2x_2, and x_1x_2.

The joint regression is done at the end of cycle one to determine which feature to add. The results of the joint regression are reported in cycle two. The values are reported in each cycle right after the regression in step 1, before joints are calculated and a new feature made. This is in keeping with the convention Sutton and Matheus used. The joint potentials of x_1x_1 and x_2x_2 are the same. This is because of the values picked for x_1 and x_2 (-1,0 and 1). In most cases, x_1x_1 and x_2x_2 equal x_1 and x_2. So, as in the case of x_1 and x_2, the potentials values will be indistinguishable.

Finally, x_1x_2 is added as a feature. A linear regression of the four features against the output is done. The error is below the threshold, so the algorithm exits. The model found is $0 * x_1 + 0 * x_2 + 0 * x_3 + 1.0 * x_1x_2$.

4.2.3 Implementation

This section will briefly discuss the design used to implement the algorithm. Hindsight is "20/20 vision"; hindsight in software design is akin to the more powerful 20/25 vision. Implementing and testing a program invariably increases one's understanding of the problem and points out shortcomings in the design, which can lead to a new design and the process repeating. This project was no exception, with both good and bad characteristics. For purposes of this project, the implementation resulted in easier and quicker changes because of the modular nature. A positive and a negative aspect of the design will be highlighted.

An object oriented design followed an initial implementation in C. The C work was based on a translation from the pseudocode that utilized little knowledge of the algorithm. Based on the knowledge and experience gained, the design was broken up into the following C++ classes:

patterns: The items to be classified. Represented as a matrix of floating point numbers. Among its properties are mean and standard deviation.

training data: A combination of a pattern and its classification/output. The classification is a floating point number. The class has the mean and standard deviation of the outputs, as well as the inherited properties of patterns.

polynomial: This class is composed of the properties needed to represent a polynomial equation. The coefficients, the symbolic name of a term, the constant term, and a list of which features are in the polynomial are included. It does not inherit or use other classes.

joint potential: The potential new features considered. Properties consist of the two individual features and the product of their potentials.

training set: The is the main "driving" class. It inherits the training data, polynomial and the joint potential classes. This is the biggest class, properties include the selection method, training and testing error, the potentials and a flag for the loom testing.

The isolation of the polynomial functions into a class aided the development of the "add" method (starting with zero features; see section 6). The class was initialized with $num\_terms = 0$ rather than $num\_terms = n$ (n = number of features). There was no hard coded assumption that all of the features were in the polynomial, that was just the default. Adding a list (array) of features in the polynomial to the polynomial class and having routines use the list was straightforward. In hindsight the training set class became bloated, it took on properties that should have been isolated in other class.

A "testing class" would have allowed better extendibility to other testing methods. The patterns class implemented the loom (section 4.3), kept track of the testing pattern, The testing pattern is skipped by the pattern access routines *begin, next,* and *end* (defined in the class patterns). These routines should be in a "partition" class which would break up the data into training and testing sets. It could also implement the list access routines; this would enable use of pre-existing libraries for dynamic allocation, such as a linked list, binary tree, etc. The list (an array) was in the foundation of the class patterns. As a result, the program was dependent on knowing the implementation details of the class patterns and therefore less modular.

4.2.4 Notes on Algorithm

The primary characteristic of the algorithm observed during its implementation and execution of the examples is that it is very dependent on the distribution of the input data. The algorithm would fail to work on Sutton's examples from the paper, if the x values $[0 \ldots 1]$ were not normalized (with mean of 0 and a standard deviation of 1) *before* calculating the output (y). This follows from the algorithm depending on the regression to show which features are significant and how to combine them. If a part of the function is not sampled, the linear model will fail to "hint" at the function correctly. This problem becomes more critical as the number of dimensions increases, the fraction of space sampled is $\frac{1}{2^n}$ where n is the number of dimensions. This assumes half an axis is not sampled (i.e. only positive values).

4.3 The Loom Method

The "leave one out method" or loom was used to test the classification accuracy of the polynomial constructed. The loom procedure uses all but one pattern for training. The left out pattern is used with the resulting classifier. This is repeated for all possible patterns left out. The average certainty or number correct over all patterns left out is the "accuracy" of

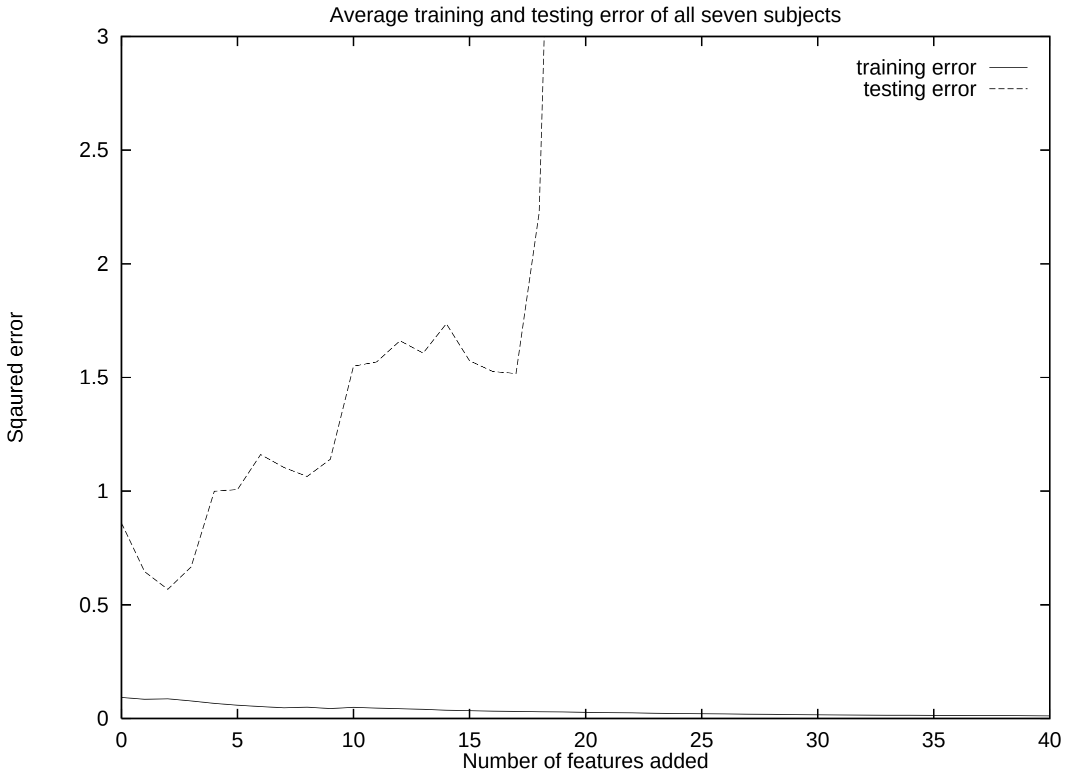

Figure 5: Average errors on polynomial classifier

the classifier. The 60 Wiener-Khinchine features from Keirn were used as the input, with a 0 or 1 for the output, based on the task. This was done for all seven subjects from Keirn's study. For the polynomial classifier, the closest point (0 or 1) to the output was used to determine the classification.

5 RESULTS of INITIAL IMPLEMENTATION

The algorithm achieved poor classification accuracy, far worse than Keirn's. Although the sum of the squared error over the training patterns was usually low, the classification on untrained data was poor. The graph of training errors for each subject exhibited different curves, but had the general feature of starting high, and had a general slope downward as features were added. The error sometimes increased when a feature was added, but usually went down again as more features were added.

Figure 5 is a plot of the average training and testing error over all seven subjects, calculated as follows. For each subject, the training error was calculated as the average of the error squared for each training pattern. The testing error was the error squared of the pattern left out. The training and testing error was calculated for all possible patterns left out. The average of these training and testing errors were the errors assigned to a subject. The training and testing error was recalculated for each new feature added. In addition, the classification accuracy was recalculated as the features were added.

The best classification achieved was 100% from subject 2, with 9 features added. Every

Subject	*# features added*	*Classification accuracy*	*Keirn's Accuracies*
1	3	75%	90%
2	9	100%	NOT GIVEN
3	2	87%	75%
4	0	55%	95%
5	1	47%	76.7%
6	16	47%	90%
7	1	50%	NOT GIVEN
Average	4.57	65.8%	85.3%

Table 2: *Best classification accuracy using joint potentials*

subjects' classification accuracy was worse than Keirn's, except for subject 3. The accuracy on average was 36% less per subject than Keirn's, while subject 3 was 11.6% higher. Table 2 lists the highest classification rates found for each subject and the number of features used.

6 ALGORITHM MODIFICATION

The results from the previous section indicate that the polynomials constructed did not generalize well, even though the squared error on training data was extremely low. This indicated that the data was over fit. Also, with 60 terms the error on the coefficients become problematic. In the examples, a coefficient of 0.01 instead of 0, did not cause a significant shift in the error vector. However, in a pattern classifier with an output of 0 or 1, 0.01 added up 60 times would cause the output to be off by 0.6. This would result in an incorrect classification. This led to the following modification: start with zero features rather than the original n. At each cycle the algorithm would either add a single feature from the *out* set (single features that are not a part of the polynomial); or make joints of the *in* features (those that make up the polynomial). A goal of this modification was to produce the smallest set of needed features for the classification. The smaller number of features would cause less over fitting in the data. This would result in better classification and fewer irrelevant terms produced. This was in contrast to the "pruning" method suggested by Sutton and Matheus [Sutton and Matheus 91].

Although not detailed, the pruning method would presumably involve using the potentials to determine unneeded features. As before, the polynomial would start with all the features. At the end of each cycle, an other ranking of the features would be made. Those that were below a certain threshold of "usefulness" would be removed. The goal is to end up with only the terms needed.

As an additional test of the add method, the "random" method was developed. In this variation of the add method, the features are picked randomly. The first feature is still picked based on a potentials regression. Each additional feature is selected as follows. A feature is randomly picked. If that feature is out, it is added as a single feature to the in set. If the feature is in, another one is selected randomly from the in set. The feature is added to the in set as the product of the two randomly selected features from the in set.

6.1 The New Algorithm

Step 0: *initialize variables*

I = number of instances (training examples)
n = dimension of the input
for $j = 1$ to n do
    $X[*][j]$ = normalize($x[1][j], \ldots, x[I][j]$) ($x[i][j]$ is j^{th} feature of pattern i)
Y = normalize($y[1] \ldots y[I]$) ($y[j]$ is the correct classification for pattern j)

w = Regress($Y|X$) do
for $i = 1$ to I do
    $SquaredError[i] = ((w[0] + \sum_{j=1}^{n} w[j] * X[i][j]) - Y[i])^2$
$SquaredError_n$ = normalize($SquaredError[1], \ldots, SquaredError[I]$)
for $j = 1$ to n do
    $Xsquared[*][j]$ = normalize($X[1][j]^2, \ldots, X[I][j]^2$)
p = Regress($SquaredError_n|X^2$)
$index$ = highest(p)
$in\_features = X[index]$
$num\_terms = 1$

Step 1: *do the regression*

w = Regress($Y|in\_features$)
for $i = 1$ to I do
    $SquaredError[i] = ((\sum_{j=1}^{n} w[j] * in_features[i][j]) - Y[i])^2$
if $\sum_{i=1}^{I} SquaredError[i] \approx 0$ return results and stop

Step 2: *construct new feature*

$SquaredError_n$ = normalize($SquaredError[1] \ldots SquaredError[I]$)
$p[*]$ = Regress($SquaredError_n|Xsquared$)
$in\_features = in\_features$ + select_new_feature(p)
$num\_terms = num\_terms + 1$
if new feature is product of two terms then
for $i = 1$ to I do
    $x[i][n+1] = x[index1] * x[index2]$
$X[*][n+1]$ = normalize($x[1][n+1], x[2][n+1], \ldots, x[I-1][n+1], x[I][n+1]$)
$Xsquared[*][n+1]$ = normalize($X[1][n+1]^2, \ldots, X[I][n+1]^2$)
$n = n + 1$
endif
GOTO step 1

The algorithm for *select_new_feature* is as follows: Do a potentials regression against the squared error, using the out features and m joints of the in features (with the highest potentials). These values are called *add potentials.* If the feature with the highest *add* potential is out, add it as a single new feature. If the feature with the highest *add* potential is a joint, the compound feature is the new one added.

6.2 Notes on Modified Algorithm

The main effect of starting with zero features is that the algorithm takes more cycles. For example, the function from Sutton and Matheus, $y = 2+3x_1x_2+4x_3x_4x_5$ using the joint method is solved with 3 features constructions: one to make x_1x_2, one for x_3x_4 and one to create $x_3x_4x_5$. The add method makes eight features: one for each of the features x_1, x_2, x_3, x_4, x_5, plus 3 for the compound features.

The one tantalizing characteristic observed of the new algorithm is that it failed about 10% of the time on normalized examples that worked 100% with the joint potentials method. Apparently having the data points normalized is not enough to guarantee the algorithm converges to the right solution. The failure is due to unneeded joints being constructed and the needed single features not being added. In the example from the previous paragraph, the features x_3 and x_4 were added to the polynomial and the joint x_3x_4 created. But x_5 is never added, therefore $x_3x_4x_5$ can never be made. The other needed single features x_1 and x_2 are also never added. The regression gives the joint features higher ratings than the out features.

7 EXPERIENCE WITH MODIFICATION

Figure 6 is the same graph as Figure 5 except the "add" method was used for the training. Since the number of features started with was initially zero, the x axis is the total number of terms in the polynomial. Subject 1 did not have the big jump in testing error as seen in figure 6. The rest of the graphs were well behaved at the beginning ($0\ldots5$ features), with small testing error. The error then fluctuated with about 6 to 9 peaks (local minimum or maximum) per subject. The testing error displayed no differentiable pattern between subjects.

The classification accuracy improved for subjects 1, 3, 5, 6 and 7. The smallest improvement was only 5 percent for subject 1, while subject 7 had a respectable 20% increase. Subject 2 had a decrease of 50%, resulting in a poor classification average. The average for the add method was 0.4% less than the joint method, indicating no statistically significant difference in performance. Table 3 lists the results for each subject.

The random method actually did better than both the add and joint potentials! The increase was only about 10%, the statistical confidence in such an increase is small. Figure 7 graphs the training and testing errors for the random method. The results for each subject are listed in table 4.

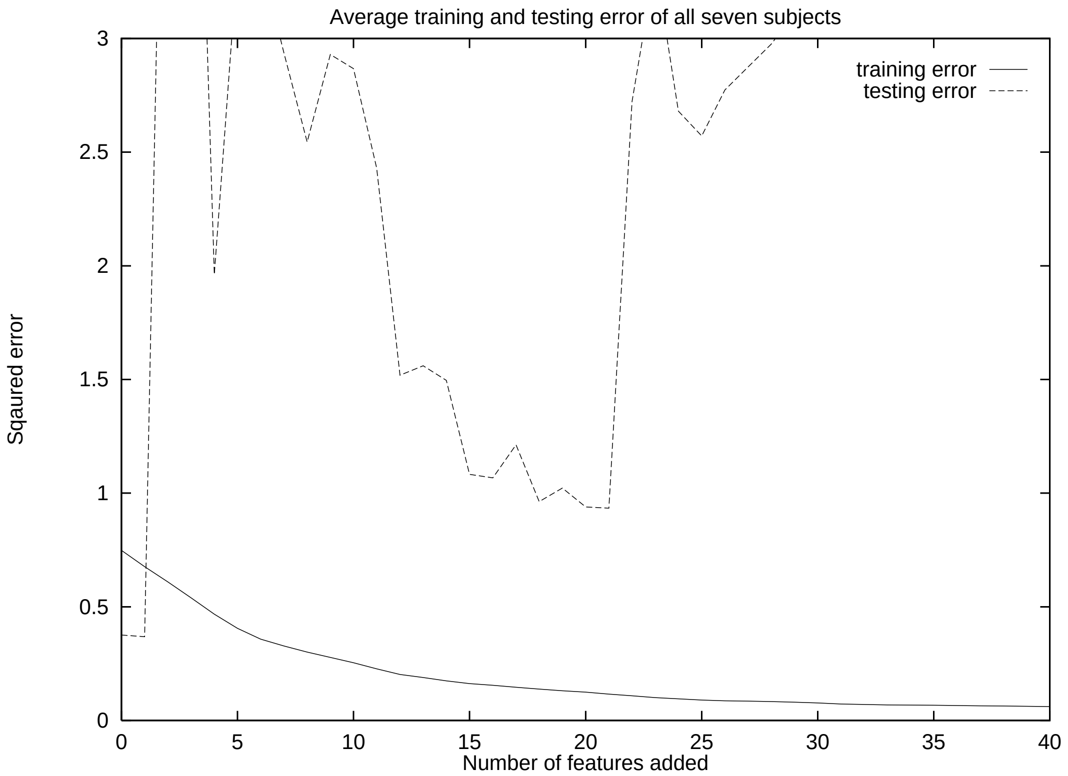

Figure 6: Average errors on polynomial classifier ("add method")

Subject	*# features*	*Classification accuracy*
1	14	80%
2	3	50%
3	38	100%
4	4	50%
5	11	56.6%
6	30	55%
7	1	70%
Average	14.4	65.4%

Table 3: *Best classification accuracy for the "add" method*

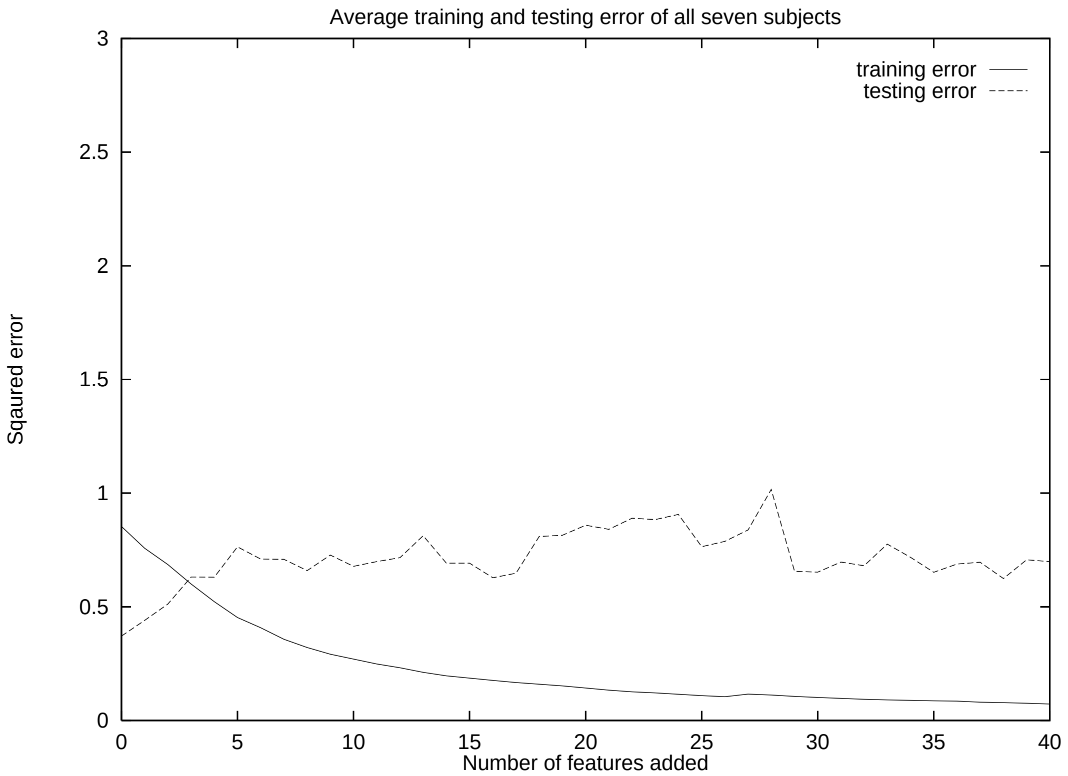

Figure 7: Average errors on polynomial classifier ("random method")

Subject	*# features*	*Classification accuracy*
1	4	75%
2	29	100%
3	30	100%
4	37	65%
5	29	66.66%
6	2	55%
7	0	70%
Average	18.7	75.95%

Table 4: *Best classification accuracy for the "random" method*

Subject	*Feature - Ranking*		
1	$\beta P3$ - 16/20	$\beta P4O1$ - 10/20	$\beta P4P3$ - 9/20
2	$\beta O2C3$ - 4/10	$\alpha C4P3$ - 4/10	$\beta P3$ - 3/10
3	$\delta P4O1$ - 20/20	$\theta P4O1$ - 19/20	$\delta P4C3$ - 12/20
4	$\beta O2O1$ - 13/20	$\beta C4O1$ - 8/20	$\beta O1$ - 7/20
5	$\beta C4$ - 27/30	$\beta O1$ - 9/30	$\delta P4P3$ - 9/30
6	$\alpha P4C3$ - 8/20	$\beta O2C3$ - 7/20	$\beta O2P3$ - 7/20
7	$\theta P4O1$ - 4/10	$\beta O2P3$ - 3/10	$\beta P4P3$ - 3/10

Table 5: *Features created by the "add method"*

7.1 Features Selected

Table 5 shows the three most common features created by the add method. The ranking of the features was done on a per subject basis as follows: For each n training sets, the first three features created were selected. The percentage of times a feature was selected over all the training sets denotes its ranking.

The nomenclature for naming the features from [Keirn 88] is used. The power values are denoted by the *band* followed by the *lead*. For example, $\alpha O1$, is the area under electrode O1 at the alpha (α) frequency band. The other bands use the corresponding Greek symbol, β for beta, δ for delta and θ for theta.

The asymmetry ratios is specified in the same way, except both leads are given. $\alpha O2O1$ is the asymmetric ratio of the leads O2 and O1 at the alpha band. $\alpha P4P3$ was the feature Keirn used for the task pair used in this report (features given only for subject 3). The add method created it once in 20 training sessions for subject 1, and in no other.

8 DISCUSSION

The algorithms failed to perform as well as Keirn's feature selection method. The current data set is too small to definitely discern if the performance is a "failure" of the algorithm or caused by the lack of information in the Wiener-Khinchine features.

The algorithm is fundamentally dependent on regression to find and rank features. The information available depends on the correlations linear regression can find. If the regression can not come up with the right features, it is essentially taking blind "stabs" at the solution.

One main difference from the examples in [Sutton and Matheus 91] and the EEG data was that the potentials of the individual features did not decrease when it was paired to make a compound feature. In one of Sutton and Matheus' examples, a feature $x_3x_4x_5$ was needed, x_4's potential was 0.32 and x_5's was 0.33. x_4x_5 was the highest joint, so the feature x_4x_5 was added. In the next cycle, x_4's potential was 0.05 and x_5's was 0.02. Because of the low potentials, the *individual* features x_4 and x_5 were not a part of any joint considered, but the compound feature x_4x_5 was (0.47 potential). As a result of the individual potential not changing (in the EEG data), the same joints were always winning. To prevent the algorithm from "looping", the "runner up" was chosen. This indicated that the potentials were not an accurate ranking of a feature's usefulness in forming new features.

A reasonable hypothesis for the failure on the EEG data is that the data was too spare for the regression to find a correlation. As an example, consider a pattern classification problem where points on one side of $y = x$ are assigned an output of 0, points on the other side are assigned an output of 1. If the points are clustered together, the model found by the linear regression will have a low enough error to quit. In the case of widely scattered points, the model found will be a bad fit. Many of the points could be very distant from the best fit line found by the regression. The error on these points will be high, causing the classification accuracy to be low. The algorithm will then attempt to build a non linear function of the input to the output which over fits the data that is best fitted by the linear relationship $y = x1 - x2 + 0.5$.

The hypothesis that the data was too small or noisy for the regression to find meaningful correlations is supported by the results of the "random" method. Since the results were actually slightly higher than the add method, we can conclude that with the EEG data the method failed to extract and use any information from the data. However, the random method fails to find the solution for the [Sutton and Matheus 91] examples. The add method works 90% of the time on those problems.

The goal of automatic feature creation, while interesting theoretically, is of less practical importance for this problem than initially believed. Most of the work in finding the feature was already done in distilling the 3072 raw data points down to 60 features. This was a conjecture made by reviewing the literature. Knowledge of the particular problem domain heavily guided the creation of these features, something a generic "black box" can not do. When the search space is large, an order of magnitude saved in steps required can make a significant difference. But if the search space is comparatively small, such as that defined by the 60 Wiener-Khinchine features, the extra time used is a negligible part of the development effort.

9 CONCLUSION

We conclude that the Sutton and Matheus' algorithm and our modification of it are not useful for this problem. If the data contains the information, it's deeply buried. Much pondering and visual examination of the data, as well as a few standard statistical measurements, fail to reveal any obvious patterns.

Relating back to the objectives, the C++ code withstood much modification before finally developing too much "baggage". The performance of the algorithm on the EEG data is not disappointing. The method was "unexplored" territory; the motivation curiosity. Both were pursued well.

It is humbling to note that the saying "the mind is the last frontier" and what we know is merely a small part of "the great ocean of knowledge" standing in front of us, is as pertinent today as it was in Isaac Newton's time.

Acknowledgments

We would like to thank Zak Keirn for his helpful comments and suggestions in understanding and recreating his work, and Richard Sutton for use of his lisp code. This work was supported in part by the National Science Foundation under Grant No. IRI-9202100.

References

[Jasper 58] Jasper H. (1958) The Ten Twenty Electrode System of the International Federation. *Electroencephalography and Clinical Neurophysiology.*

[Keirn 88] Keirn, Z. A. (1988) Alternative Modes of Communication Between Man and Machine *Master's Thesis*, Purdue University.

[Keirn and Aunon 90] Keirn, Z. A. and Aunon, J. I. (1990) A new mode of communication between Man and his surroundings. *IEEE Trans.*, **BME-37,** 1209-1214.

[Sanger 91] Sanger T. D. (1991) Basis-function trees as a generalization of local variable selections methods for function approximation. In D. S. Touretzky, (ed.), *Advances in Neural Information Processing Systems 3.* Morgan Kaufmann, San Mateo, CA..

[Sanger, et al. 92] Sanger, T. D., Sutton, R. S. and Matheus, C. J. (1992) Iterative Construction of Sparse Polynomial Approximations. In Moody, Hanson and Lippmann (ed.s), *Advances in Neural Information Processing Systems 4.* Morgan Kaufmann.

[Shiao-Lin, et al. 93] Lin S., Tsai Y. and Liou C. (1993) Conscious mental tasks and their EEG signals. *Medical. & Biological Engineering & Computing*, **31**, 421-425.

[Sutton and Matheus 91] Sutton, R. S. and Matheus, C. J. (1991) Learning Polynomial functions by feature Construction. In *Proceedings of the Eighth International Workshop on Machine Learning.* Chicago, Illinois, June 27-29.