Proceedings of the Eighth International Workshop on Machine Learning *Chicago, Illinois, June 27-29 1991*

Learning Polynomial Functions by Feature Construction

Richard S. Sutton
GTE Laboratories Incorporated
40 Sylvan Rd., Waltham MA 02254
sutton@gte.com

Christopher J. Matheus
GTE Laboratories Incorporated
40 Sylvan Rd., Waltham MA 02254
matheus@gte.com

Abstract

We present a method for learning higher-order polynomial functions from examples using linear regression and feature construction. Regression is used on a set of training instances to produce a weight vector for a linear function over the feature set. If this hypothesis is imperfect, a new feature is constructed by forming the product of the two features that most effectively predict the squared error of the current hypothesis. The algorithm is then repeated. In an extension to this method, the specific pair of features to combine is selected by measuring their joint ability to predict the hypothesis' error.

1 INTRODUCTION

We present a method for learning higher-order polynomial functions from examples using linear regression and feature construction. The linear regression algorithm takes a training set and returns a hypothesis represented as the coefficients for a linear function over the feature set. If this hypothesis is less than perfect when applied to the training set, a new feature is constructed by forming the product of the two features that most effectively predict the squared error of the current hypothesis; and then the process is repeated. In an extension to this method, the specific pair of features to combine is selected by measuring their joint ability to predict the hypothesis' error. Preliminary experiments with this approach are promising and suggest some extensions.

We begin this paper with a description of the learning problem followed by a general explanation of our method. We then describe the algorithm and illustrate its use on two simple examples. We conclude with a short discussion of anticipated extensions of this research.

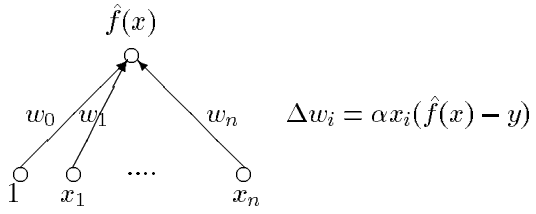
2 THE PROBLEM

The problem we are interested in is learning polynomial functions from examples: Given a set of real-valued vectors $\{y, x_1, ...x_n\}$, learn the function $y = f(x)$. This task represents a subset of the general problem of multivariate function approximation (see (Breiman *et al.*, 1984; Friedman, 1988)).

When $f(x)$ is a first-order polynomial it can be represented as a linear function over the independent variables x:

$$f(x) = w_0 + \sum_{i=0}^{n} w_i x_i$$

We can derive the coefficient vector w in this equation by doing a linear regression on the training instances. Equivalently, we could induce the coefficients by training a simple linear network using the LMS rule, $\Delta w_i = \alpha x_i(\hat{f}(x) - y)$, where α is the learning rate and $\hat{f}(x)$ is the network's estimate of $f(x)$:



Either of these methods will find the set of coefficients that form the optimal linear approximation $\hat{f}(x)$ in terms of the minimum mean squared error on the training instances.

Unfortunately, linear functions are inadequate for representing higher-order polynomials. To learn these we must either use a learning method that searches a more complex hypothesis space (e.g., a multi-layered connectionist network) or change the set of features used as the basis for the linear function. The method described in this paper implements the latter approach through a new form of feature construction.

3 THE METHOD

As a simple example, consider the function $y = 2x_1 + 5x_2x_3$. Clearly, y is not linear over the set of primitive features $\{x_1, x_2, x_3\}$. If, however, we add a new feature $x' = x_2x_3$, then y can be represented as a linear function over this new basis: $y = 2x_1 + 5x'$. The problem with this sort of feature construction is knowing which new features to construct. For example, for a polynomial of order δ with training instances described by n independent features and with c constructive operators, the cardinality of the space of potentially useful new features is of order $c(n+1)^\delta$.

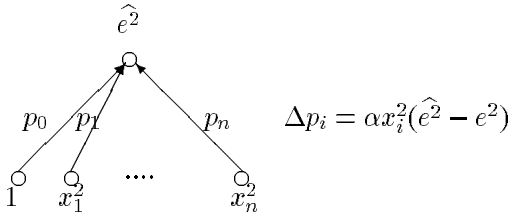
In our method we control the complexity of feature construction with two major constraints. First, we use multiplication as our only constructive operator. For polynomial functions this single operator is sufficient; its use for this problem domain can be viewed as a piece of domain knowledge. Second, the multiplication operator we use is binary. This means we can combine only two features at a time, and must apply the operator iteratively to construct higher-order features. For example, to construct the new feature $x_1x_2x_3$ would require two iterations of feature construction, e.g. $x' = product(x_1, x_2)$ followed by $x'' = product(x', x3)$.

Having just this single binary operator reduces the number of candidate new features at any given moment to $\frac{n^2}{2}$. This number is still rather large to simply add all candidates, most of which would be useless anyway. Instead, we selectively construct new features one at a time by rating all features individually and then forming the product of the pair of features with the highest ratings. In our method, features are rated according to their ability to predict the squared error of the hypothesis (cf., Sanger, 1991).

The rational behind this way of rating features can be explained by viewing the model in terms of a single unit network. After the network is trained on a set of examples it will accurately predict y for some training instances but not for others. On the instances where its prediction is poor, the squared error (the square of the difference between the predicted and actual values) will be high. When the squared error is high, the largest adjustments are made by the learning rule, and the largest of these are made in the coefficients whose features have the largest magnitudes at the time (recall $\Delta w_i = \alpha x_i(\hat{f}(x) - y)$). If, after the network coefficients have stabilized, a given feature x_j still tends to be high when the error is high, this implies that this unit is relevant to the target function but by itself it is unable to accurately contribute to the prediction of the function's value.

One way to measure the ability of a feature to predict the squared error is to measure the correlation between the squared error and the square of the feature's value: $correlation_i = \mathrm{E}[e^2x_i^2]$ where E stands for the expected value. This is the approach proposed by Sanger (1991), who notes that this correlation is proportional to the variance of the coefficients in the limit as they approach stability.[1] We have used this feature-rating method successfully, but we have also had a problem with it: if a new feature such as x_i^2 is constructed, its correlation is typically very similar to that of its single parent x_i, and therefore it is rated highly for inclusion in a further new features. Consequently, the system proceeds to construct more and more complex features in the same terms, contributing little new to the feature set.

We have modified Sanger's idea to give it a competitive quality that discourages the formation of features that do not improve the prediction of the squared error. Instead of using the correlations between squared error and squared value we use the coefficients from a regression of the squared error over the squared values: $Regress(e^2|x^2)$. The equivalent network is as shown below:



We refer to these coefficients p as "potentials" because we use them to determine a feature's relative potential for being useful as part of a new feature. To construct a new feature we take the product of the two features with the highest potentials.[2]

4 THE ALGORITHM

The algorithm to implement our method is written as pseudo-code in Figure 1. After normalizing the training instances such that all features have zero mean and variance of one, a regression is performed (step 1) on the training set to learn the best set of coefficients for predicting y given x. If the error (the residual) is approximately zero, then the function has been learned and the algorithm halts; otherwise feature construction is initiated (step 2). A second regression is performed to derive the potentials p for predicting the squared error from the (normalized) values of x_i^2. The product of the two features with the highest potentials becomes the definition of a new feature. This feature

[1] Note that this differs from "cascade correlation" (Fahlman and Lebiere, 1990) in that we correlate with the *square* of the error, rather that with the error itself.

[2] If the two features with the highest potentials have already been used to define a feature then the next best pair is used.

step 0: *initialize variables*
I = number of instances
n = number of features
$X = norm(\{\{x_1^1 ... x_n^1\} ... \{x_1^I, ... x_n^I\}\})$
$Y = norm(\{y^1 ... y^I\})$
$FeatureSet = \{X_1 ... X_n\}$
step 1: *do the regression*
$w = Regress(Y|X)$
$SquaredError = \alpha \sum_{i=1}^{I} ((w_0 + \sum_{j=1}^{I} w_j x_j^i) - y^i)^2$
if $SquaredError \approx 0$ return results and STOP.
step 2: *construct new feature*
$X^2 = norm(\{\{(x_1^1)^2 ... (x_n^1)^2\} ... \{(x_1^I)^2, ... (x_n^I)^2\}\})$
$p = Regress(SquaredError|X^2)$
$Index_1 = index(heighest(p))$
$Index_2 = index(second - heighest(p))$
$NewFeature = product(x_{index_1}, x_{index_2})$
$FeatureSet = FeatureSet + NewFeature$
$n = n + 1$
$X = norm(\{\{x_1^1, ... x_{n-1}^1, product(x_{index_1}^1, x_{index_2}^1)\}$
...
$\{x_1^I, ... x_{n-1}^I, product(x_{index_1}^I, x_{index_2}^I)\}\})$
GOTO step 1.

Figure 1: The Potentials Algorithm

is added to the feature set and all the instances are updated and renormalized. The algorithm then returns to step 1 and the process is repeated.

5 AN EXAMPLE

Figure 2 shows a simple example illustrating the algorithm and some of its problems. For this example 200 training instances were created by randomly generating five real-value variables (range = [0,1]) and calculating their values for the function $y = 2 + 3x_1x_2 + 4x_3x_4x_5$. In addition to y and these five relevant variables x_1, x_2, x_3, x_4, and x_5, four distracter variables x_6, x_7, x_8, and x_9 were added to each instance (randomly generated in the same way).

In this example, the algorithm derived the correct function in the seventh cycle, after creating six new terms: x_5x_5, x_4x_5, $x_4x_4x_5x_5$, $x_3x_4x_5$, x_2x_2, and x_1x_2. The output from each cycle reports the following information:

Coeffs the coefficients for each feature resulting from the linear regression

Potentials the potentials used to select the two features to combine into a new term

Residual the total squared error of the best linear model of the target function as derived by the regression algorithm

New Terms the new term that was constructed

One thing to notice in this example is that the algorithm completely disregards the distracter variables (a characteristic shared by the correlation method). This has been a consistent result in all our tests in which the training set was sufficiently large for the complexity of the terms in the function. As the number of primitive features in a term increases, each individual feature's contribution to the function's value decreases; as this contribution becomes small it becomes difficult to separate from the random noise of the distracters, unless the size of the training set is appropriately increased.

There are two problems with this example, one obvious and the other more subtle. First, four unnecessary features were constructed: $x5x5$, $x3x3$, $x4x4x5x5$, and $x1x1$. These did not prevent solution of the problem, but they did slow down the process; for more complex functions these additional terms could become problematic. Second, the target function is such that the potentials of the relevant features are significantly different depending upon which of the two terms of the function they participated in. As a result, the algorithm is able to determine which relevant features go together. So for example, x_1 is in a term of two features having a coefficient of 3 whereas x_3 is in a term of three features weighted by 4; consequently their initial potentials are quite different. If we had used the function $y = x_1x_2 + x_3x_4$ the algorithm would not have been able to tell that x_1 should be paired with x_2 rather than with x_3 or x_4, other than by chance.

6 JOINT POTENTIALS

The example discussed above shows that the method of potentials is good at identifying which terms are relevant to the function, but not at determining which *combinations* of these terms are relevant. To remedy this shortcoming we developed the notion of "joint potentials," that is, the potentials of pairs of features. To compute the potentials of all pairs of n features is not feasible as it would require a regression on n^2 variables. Instead, we restrict attention to the pairs of features that appear most promising according to the product of the potentials of their constituent features. For the m most promising pairs, for $m << n^2$, we perform another regression onto the squared error. The coefficients of this regression are called the *joint potentials*. The pair with the highest joint potential is then selected as the next new feature. This is basically a traditional generate-and-test approach to feature construction,[3] but guided by the potential ratings of the individual original features.

The example run in Figure 3 shows the results of using this method on the training set from the first example (using $n = m = 9$). This time the algorithm constructs the minimum number of new features required to learn

[3] As in, e.g., (Ivakhnenko, 1971).

```
Cycle 1:          1     2     3     4     5     6     7     8     9
  Coeffs:      -0.19  0.14  0.24  0.31  0.17  0.48  0.03  0.05  0.58
  Potentials:   0.22  0.24  0.25  0.32  0.33  0.01  0.08  0.01  0.05
  Residual: 193  New Terms:  X5X5
Cycle 2:          1     2     3     4     5     6     7     8     9     55
  Coeffs:      -0.20  0.14  0.23  0.31  0.17  0.48  0.02  0.06  0.58  0.05
  Potentials:   0.22  0.24  0.25  0.33  0.38  0.01  0.07  0.01  0.04 -0.08
  Residual: 193  New Terms:  X4X5
Cycle 3:          1     2     3     4     5     6     7     8     9     55    45
  Coeffs:      -0.20  0.14  0.24  0.31  0.18  0.48  0.02  0.05  0.57  0.05  0.05
  Potentials:   0.25  0.23  0.25  0.05  0.07  0.03  0.07  0.02  0.03 -0.08  0.47
  Residual: 193  New Terms:  X4X4X5X5
Cycle 4:          1     2     3     4     5     6     7     8     9     55    45  4455
  Coeffs:      -0.22  0.16  0.25  0.25  0.24  0.46 -0.05  0.06  0.55  0.61  0.00 -0.56
  Potentials:   0.19  0.22  0.24 -0.00 -0.01 -0.01  0.09  0.02  0.07 -0.02  0.65 -0.12
  Residual: 189  New Terms:  X3X4X5
Cycle 5:          1     2     3     4     5     6     7     8     9     55    45  4455  345
  Coeffs:       0.03 -0.25  0.09  0.34 -0.01  0.27  0.07  0.22  0.41  0.24 -0.28 -0.22  4.04
  Potentials:   0.51  0.59  0.02  0.02 -0.09  0.01 -0.04 -0.04  0.05  0.02 -0.05  0.00  0.04
  Residual:  69  New Terms:  X2X2
Cycle 6:          1     2     3     4     5     6     7     8     9     55    45  4455  345   22
  Coeffs:       0.02 -0.22  0.09  0.33 -0.01  0.28  0.06  0.21  0.42  0.22 -0.27 -0.21  4.04 -0.18
  Potentials:   0.51  0.58  0.03  0.03 -0.08 -0.00 -0.04 -0.04  0.06  0.02 -0.08  0.02  0.03  0.01
  Residual:  68  New Terms:  X1X2
Cycle 7:          1     2     3     4     5     6     7     8     9     55    45  4455  345   22    12
  Coeffs:      -0.00  0.00 -0.00  0.00  0.00  0.00 -0.00  0.00  0.00  0.00 -0.00 -0.00  4.00  0.00  3.00
  Residual:   0
  Solution:  (+ 2.0 (* 4.0 X3 X4 X5) (* 3.0 X1 X2) )
```

Figure 2: Application of the algorithm to the target function $y = 2+3x_1x_2+4x_3x_4x_5$. The training set included four distracter variables and 200 examples.

the target function. The information displayed on each cycle now includes the top four joint potentials (limited to four for display purposes).

The example run in Figure 4 shows how the joint potentials method also solves the problem of determining which pairs of relevant features belong together. Recall our statement that functions of the form $y = x_1x_2 + x_3x_4$ (where each term is of equal weight and length) are difficult for the straight potential method because it has no way of deciding how to pair features. This third example demonstrates that the joint potential selection method can correctly decide which of the relevant features to combine. Notice that if the algorithm had used the (non-joint) potentials to pick the new feature pair on cycle 1, it would have created the useless new feature x_1x_4 since x_1 and x_4 have the two highest potentials. Using joint potentials, the relevant feature x_1x_2 is constructed instead.

7 EXTENSIONS

This research is still preliminary. We are in the process of designing more thorough tests of this approach and we are developing a better theoretical explanation for when and why this approach works. In addition we are exploring several new ideas, among them:

An incremental algorithm. The current algorithm assumes that all the instances are available at the outset. We intend to make it able to construct features and learn functions incrementally, i.e., by processing each example only once.

Comparisons. We are currently working with Sanger to obtain direct comparisons with his original method on the basis of learning rate and computational requirements.

Multiple new features per cycle. Rather than adding just a single feature per cycle we are working on a method that adds a set of features based on their potentials or joint potentials.

Application to more complex functions. Because we have constrained our method to the use of a multiplication operator the domain of solvable problems is quite limited. We are exploring the use of additional constructive operators, such as division, roots, and transcendental functions.

Pruning of unnecessary features. As we saw in the above examples, the method of potentials is quite good at identifying irrelevant features. It should be a simple matter to use this information to prune the feature set and thereby reduce the computation at each cycle.

```
Cycle 1:          1     2     3     4     5     6     7     8     9
  Coeffs:      -0.19  0.14  0.24  0.31  0.17  0.48  0.03  0.05  0.58
  Potentials:   0.22  0.24  0.25  0.32  0.33  0.01  0.08  0.01  0.05
  Joints:       ((260 5 . 4) (248 5 . 3) (246 4 . 3) (200 4 . 4))
  Residual: 193  New Terms:  X4X5
Cycle 2:          1     2     3     4     5     6     7     8     9     45
  Coeffs:      -0.19  0.14  0.24  0.30  0.18  0.48  0.03  0.05  0.57  0.05
  Potentials:   0.25  0.22  0.25  0.05  0.02  0.03  0.08  0.02  0.03  0.47
  Joints:       ((306 10 . 3) (268 10 . 1) (260 10 . 2) (235 10 . 10))
  Residual: 193  New Terms:  X3X4X5
Cycle 3:          1     2     3     4     5     6     7     8     9     45    345
  Coeffs:       0.04 -0.26  0.09  0.37 -0.04  0.27  0.10  0.22  0.42 -0.26  4.07
  Potentials:   0.52  0.59  0.03  0.02 -0.08  0.03 -0.05 -0.06  0.05 -0.05  0.05
  Joints:       ((114 2 . 1) (81 2 . 9) (78 2 . 2) (74 1 . 9))
  Residual:  69  New Terms:  X1X2
Cycle 4:          1     2     3     4     5     6     7     8     9     45    345   12
  Coeffs:      -0.00 -0.00 -0.00  0.00 -0.00  0.00  0.00  0.00  0.00 -0.00  4.00  3.00
  Residual:   0
  Solution:  (+ 2.0 (* 4.0 X3 X4 X5) (* 3.0 X1 X2) )
```

Figure 3: Application of the algorithm to the target function $y = 2 + 3x_1x_2 + 4x_3x_4x_5$ using Joint Potentials. The training set included four distracter variables and 200 examples.

```
Cycle 1:          1     2     3     4     5     6     7     8     9
  Coeffs:       0.08 -0.13 -0.03  0.16  0.04 -0.04  0.10  0.06  0.16
  Potentials:   0.30  0.26  0.25  0.41 -0.03 -0.03 -0.11 -0.03  0.01
  Joints:       ((230 1 . 2) (229 4 . 1) (228 4 . 2) (189 4 . 4))
  Residual: 189  New Terms:  X1X2
Cycle 2:          1     2     3     4     5     6     7     8     9     12
  Coeffs:      -0.04 -0.02 -0.00  0.07  0.06 -0.10  0.03 -0.00  0.01  1.07
  Potentials:   0.07  0.02  0.56  0.58 -0.07  0.05 -0.07 -0.04 -0.05 -0.10
  Joints:       ((151 4 . 3) (111 4 . 6) (109 3 . 6) (100 3 . 1))
  Residual:  89  New Terms:  X3X4
Cycle 3:          1     2     3     4     5     6     7     8     9     12    34
  Coeffs:      -0.00  0.00  0.00  0.00  0.00 -0.00 -0.00 -0.00 -0.00  1.00  1.00
  Residual:   0
  Solution:  (+ (* 1.0 X1 X2) (* 1.0 X3 X4) )
```

Figure 4: Application of the algorithm to the target function $y = x_1x_2 + x_3x_4$ using Joint Potentials. The training set included four distracter variables and 200 examples.

Acknowledgments

We would like to thank Terry Sanger for his encouragement and discussion of this work. Thank you also to Rich Brandau for his helpful comments.

References

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. (1984) *Classification and Regression Trees.* Wadsworth & Brooks, Belmont, California.

S. E. Fahlman and C. Lebiere. (1990) The cascade-correlation learning architecture. In D. S. Touretzky, (ed.), *Advances in Neural Information Processing Systems 2*, pages 524–532. Morgan Kaufmann.

J. H. Friedman. (1988) Multivariate adaptive regression splines. Technical Report TR No. 102, Laboratory for Computational Statistics, Stanford University, Stanford, CA.

A. G. Ivakhnenko. (1971) Polynomial theory of complex systems. *IEEE Trans. Systems, Man, and Cybernetics*, SMC-1(4):364–378.

T. D. Sanger. (1991) A tree-structured algorithm for reducing computation in networks with separable basis functions. *Neural Computation*, 3(1):67–78.