Classification Using Φ-Machines And Constructive Function Approximation

Doina Precup
Paul E. Utgoff

Technical Report 97-05
January 21, 1997

Department of Computer Science
University of Massachusetts
Amherst, MA 01003

Telephone: (413) 545-4843
Net: utgoff@cs.umass.edu

Abstract

The new classification algorithm CLEF combines a version of a linear machine known as a Φ-machine with a non-linear function approximator that constructs its own features. The algorithm finds non-linear decision boundaries by constructing features that are needed to learn the necessary discriminant functions. The CLEF algorithm is proven to separate all consistently labelled training instances, even when they are not linearly separable in the input variables. The algorithm is illustrated on a variety of tasks.

Contents

1 Introduction

The task of classification is to find an approximate definition for an unknown function $f : \mathbf{X} \rightarrow \{c_1, ..c_R\},\ R \geq 2$ based on a set of training examples of the form $\langle \mathbf{x}_i, f(\mathbf{x}_i) \rangle$. The components of an instance vector $\mathbf{x}_i$ can take values from discrete or continuous domains. It is also possible that the values of one or more components are missing or imprecisely recorded for certain training instances.

A popular approach to classification is to construct a decision tree (Quinlan, 1985). Typical decision tree induction algorithms operate by recursively partitioning the training data and picking the best split according to some metric, until some stopping criterion is satisfied. This repeated partitioning is problematic because the number of instances present in each block is reduced after each partition. Moreover, the way in which the tests are picked also leads to a problem related to the statistical issue of multiple comparisons (Cohen & Jensen, 1997).

Feed forward neural networks have also been used for classification. However, networks are usually difficult to tune, since their behavior depends on several parameters (Mooney, Shavlik, Towell & Gove, 1989). Backpropagation, (Rumelhart, Hinton & Williams, 1986) which is the best known algorithm for training feed forward multilayered neural networks, is not guaranteed to converge to a global minimum of the error surface.

This paper presents a new approach to classification, that aims to eliminate the disadvantages of the methods mentioned above. CLEF constructs a machine that is linear in the parameters, but non-linear in the input variables. Unlike decision trees, the method proposed keeps the whole training set together during the classifier's construction. The algorithm does not need multiple runs to achieve good results, and is proven to find a perfect separation of the training instances into classes, if one exists. The features it extracts from the data are inspectable, and thus interpretable.

2 Linear Machines

One approach that allows classification without splitting the training data is to use linear machines (Nilsson, 1965; Duda & Hart, 1973). A linear machine is a set of R linear discriminant functions g_i used collectively to assign an instance to one of R classes (Nilsson, 1965). Let $\mathbf{x} = (1, x_1, ..x_n)$ be an instance description. Each discriminant function $g_i(\mathbf{x})$ has the form $\mathbf{w}_i^{\mathbf{T}}\mathbf{x}$, where $\mathbf{w}$ is an $(n+1)$-dimensional vector of coefficients (weights). An instance is assigned class i if and only if $g_i(\mathbf{x}) > g_j(\mathbf{x})\ \forall j \neq i$. If a tie occurs, the instance is attributed randomly to one of the classes.

The training algorithm of a linear machine adjusts its weights based on a set of training instances. The machine starts with arbitrary initial weights, and sweeps through the set of training instances repeatedly. If an instance having class i is erroneously placed into class j, the weight vectors corresponding to the two classes are adjusted as follows: $\mathbf{w}_i \leftarrow \mathbf{w}_i + c\mathbf{x}$ and $\mathbf{w}_j \leftarrow \mathbf{w}_j - c\mathbf{x}$. The amount of correction c can be computed using the fractional error correction rule (Nilsson, 1965):

$$c = \alpha \frac{(\mathbf{w}_i - \mathbf{w}_j)^{\mathbf{T}}\mathbf{x}}{2\mathbf{x}^{\mathbf{T}}\mathbf{x}} + \epsilon$$

If the training instances are linearly separable, this update rule guarantees that the linear machine will converge to a boundary that classifies them correctly (Nilsson, 1965).

For many tasks, linear combinations of the input values are not enough to discriminate the groups of instances belonging to each class. When a non-linear discriminant is needed, a possible solution is to use a Φ-machine (Nilsson, 1965), which is much like a linear machine. A Φ-machine uses discriminant functions of the form $g_i(\mathbf{x}) = \mathbf{w}_i^{\mathbf{T}} F_i(\mathbf{x})$, where $F_i = \langle f_1, ..., f_M \rangle$ is a vector of linearly independent, real, single-valued functions $f_j : \mathbf{X} \rightarrow \Re$, independent of the weights. This means that f_j are not varying with the weight adjustments. Multilayered neural networks, for instance, do not satisfy this requirement.

Φ-machines preserve the theoretical advantages of linear machines, while allowing for non-linear combinations of the inputs. Thus, they offer higher representational power. The training procedures used for linear machines can be applied to adjust the weights of Φ-machines. All the convergence theorems for linear machines apply to Φ-machines as well.

Due to the great variety of clasification tasks, one cannot know a priori what mappings f_j would be useful as components of discriminants. One would like to construct such functions f_j automatically, based on the training instances.

3 Constructing a Φ-machine for classification

An automatic method for constructing a Φ-machine adequate for the task at hand is needed. To this end, we employ the ELF function approximation algorithm (Utgoff & Precup, 1997), which produces an approximation in the form of a Φ-machine. Furthermore, ELF constructs new features as needed, by identifying subsets of instances that share intrinsic properties. Our classification algorithm uses ELF to produce a sequence of Φ-machines, each of which is progressively better suited to the task we are trying to solve.

ELF assumes that the instances are represented using Boolean input variables. Its goal is to find set covers over the instance space, grouping the instances in subsets that can be associated a common value. To this end, ELF constructs features, defined by a pattern vector with as many components as the dimensionality of an instance vector $\mathbf{x}$. Each component of a pattern has either the value '#' or the value '0'. A '#' matches any (either) of the possible values of the corresponding input vector, while a '0' in the pattern matches only a '0' value. A table depicting whether a component matches is shown in Table 1. The pattern of all '#'s covers every instance because the pattern matches any instance at every component. The pattern of all '0's covers the one instance in which all the components have value '0'. One pattern is strictly more general than another if and only if it covers all the instances covered by the other.

Table 1. Matching of Pattern and Instance

Pattern	Instance	Match
#	1	true
#	0	true
0	1	false
0	0	true

Let $\mathbf{X}$ be the space of all input instances. A feature can be expressed as a membership function for a subset of instances $\mathbf{X}_j \subseteq \mathbf{X}$. Each feature can, thus, be expressed as a function

$$f_j(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathbf{X}_j \\ 0 & \text{otherwise} \end{cases}$$

Note that every feature except the one covering the whole instance space is nonlinear in the input variables because it has value 0 for the states it does not cover, and the value of 1 for the states that it does cover. When multiplied by its corresponding weight, one term $\mathbf{w}_j\mathbf{f}_j$ has value $\mathbf{w}_j$ for the instances that it covers, and 0 elsewhere, thus associating a particular value with a particular set of instances.

Initially, each discriminant function consists of the one most general feature, which covers the whole instance space, with a weight of 0. To evaluate an instance using a discriminant function, one computes the linear combination of the feature values and feature weights. Thus, the initial approximation returns 0 for every instance. To update the approximation, the training procedure revisits the training instances and adjusts the weights of the discriminant functions using the fractional error correction rule (Nilsson, 1965). Only features that matched the instance have their weights adjusted, because features that did not match have value 0.

Of course, the approximation with the one most general feature is rarely adequate. One can repeatedly adjust its single weight, attempting in effect to fit a constant function to the points. The algorithm keeps track of which feature is having the greatest difficulty in fitting, because it is getting different errors. Furthermore, ELF associates this difficulty with each of the input components. When an adjustment of the weights has ceased to be productive, the algorithm adds a new feature that is a specialized copy of the feature that has been having the greatest difficulty in fitting. The copy is specialized by changing a '#' in its pattern to a '0', thereby customizing the set of states covered by the feature.

The features that are created by this procedure are linearly independent. The proof of this statement can be done by induction on the number n of bits that are present in an input instance. Consider the base case, in which $n = 1$. The instance space contains two instances: '0' and '1'. There are two features that can be defined over this instance space: the most specialized feature, which is associated with the pattern '0' and only covers the first instance, and the most general feature, which corresponds to the pattern '#' and covers both instances. The values of the features for each instance can be tabulated in the following determinant:

$$\begin{array}{c|cc|} & 0 & \# \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{array}$$

which can be reduced to a unit determinant, by subtracting the last line from the first one.

Now comes the induction step. Consider the space of the instances that can be generated by n input bits. These instances can be viewed as being generated from the $(n-1)$-bit instances, by adding a '0' or a '1' upfront. Similarly, the features that can be defined over these instances are generated from $(n-1)$-bit features by adding a '0' or a '#' up front. Let d_{n-1} define the determinant of the $(n-1)$-bit space input features. The determinant d_n on the n-bit space can be written as:

$$\begin{array}{c|cc|} & 0F_{n-1} & \#F_{n-1} \\ 0X_{n-1} & d_{n-1} & d_{n-1} \\ 1X_{n-1} & 0 & d_{n-1} \end{array}$$

The induction hypothesis is that d_{n-1} can be reduced to a unit determinant. This can be done by adding and subtracting lines from each other, as we did in the base case. If there is a sequence of transformations that achieves this goal, we can apply it in the upper and lower part of d_n. The resulting determinant will have the form:

$$\left|\begin{array}{cccc|cccc} 1 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 1 \\ \hline 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 1 \end{array}\right|$$

By subtracting the bottom half of the determinant from the upper half, d_n can also be reduced to a unit determinant. Thus, the set of all possible features is linearly independent. This means that any subset of features will be linearly independent as well. ■

The previous proof ensures that at any point between two feature additions, the classifier that is built is a Φ-machine. One can view the process of training CLEF's classifier as constructing a sequence of Φ-machines. A machine will converge to a set of weights that separates the training instances, if a separation is possible given the current set of features. If no linear separation can be found given the current feature set, by gradually reducing the size of the corrections, the weights will still settle in a particular range (Frean, 1990).

In this case, a new feature will be added, and training will resume with a new machine. In the worst case, the process will continue until all the 2^n features that are possible have been generated. If the instances are separable when mapped through a subset of the features, they will also be separable when the whole subset is used. Thus, if a linear separation of the training instances is possible, the algorithm is guaranteed to find one. In practice, CLEF also proved to be quite efficient with respect to the number of features it generates for a particular instance space.

4 Input representation

The non-linear machine described so far requires boolean input values. Such an encoding can be generated automatically for classification tasks. Symbolic variables are mapped into a 1-of-m encoding, where m is the number of possible values for each variable. Bit j corresponding to a variable v with possible values $v_1, ...v_m$ will have the value 1 in an instance representation if and only if the test $v(\mathbf{x}) = v_j$ is true.

For numeric variables, some form of discretization is needed. The current solution adopted is to determine cutpoints in the same manner as C4.5 (Quinlan, 1993). Each cutpoint is associated with a bit in the input representation. The corresponding bit has the value 1 if the test $v(\mathbf{x}) < \mathit{cutpoint}$ is true, and false otherwise. This solution yields a dimensionality problem. The number of cutpoints can be quite big for each continuous variable. This increases the number of bits used in the input encoding. The number of training points that are needed to get a reliable separation also increases accordingly. Most of the classification tasks that are available do not provide enough data to train in this case.

Feed forward neural networks can be used to construct features based on continuous input values. However, using this kind of representation would give away any guarantee of convergence. The issue of finding a more efficient representation of continuous variables for CLEF will be addressed in the near future.

If the value of a variable is missing in the input representation, then all the bits corresponding to that variable will be set to 0. This prevents the missing value from having any role in the classification process, since it will not interfere with the matching process.

5 Illustration

The Boolean encoding of the features allows an interpretation of the units that form a non-linear classification machine. Feature interpretation can be generated automatically, by printing the negation of each test for which there is a '0' in the feature's pattern.

Table 2 illustrates the features that have been constructed for one of the units (discriminant functions) in the hepatitis task. This is a two-class problem, thus the corresponding linear machine will have two discriminant functions, one for each class. However, due to the training procedure, these discriminant functions are always trained with equal amounts of error having opposite signs. Thus, in this two class case, the functions end up having the same features, with opposite sign weights.

Table 2. Unit corresponding to the "die" class in the hepatitis task

Hepatitis

Weight	Feature
-0.019	age $\not<$ 37.50
0.013	ascites $\neq$ no
-0.012	age $\not<$ 37.50, liver-firm $\neq$ yes, spiders $\neq$ no, varices $\neq$ no
0.008	
0.008	age $\not<$ 37.50, protime $\not<$ 44.50
0.008	age $\not<$ 37.50, varices $\neq$ no
0.007	age $\not<$ 37.50, spiders $\neq$ no, varices $\neq$ no
-0.006	sgot $\not<$ 80.50, protime $\not<$ 87.50
-0.005	steroid $\neq$ yes
-0.004	bilirubin $\not<$ 1.35
-0.004	protime $\not<$ 87.50
-0.004	sex $\neq$ female
0.003	sex $\neq$ female, anorexia $\neq$ yes
-0.002	sex $\neq$ female, liver-firm $\neq$ no
0.001	spiders $\neq$ no, histology $\neq$ yes
-0.000	spiders $\neq$ no

One can interpret this table as a "health test", which tells how to compute a score for an instance. For each line in the table, one would check if the instance satisfies the test in the right column. If so, the corresponding weight would be added to the total score. If the total score is positive, the instance would be considered as belonging to the "die" class.

6 Analysis

One would predict that CLEF should perform at least as well as a top-down decision tree inducer. The encoding of the instances is in terms of the same

Boolean tests that a decision tree algorithm might use. Decision trees can be represented in the form of Φ-machines. Instead of building subtrees recursively on subsets of the training data, CLEF constructs non-linear features over all the data, from which discriminants linear in the features are formed. The salient difference is that tree inducers partition the data into smaller subproblems, while CLEF does not. It should be advantageous to CLEF that it solves one classification problem using all the data, instead of many subproblems, each using only some of the data.

CLEF does a considerable amount of computation. Will it find a separating Φ-machine in a reasonable amount of time? Will it construct a large number of features, perhaps producing an incomprehensible classifier? In order to answer some of these questions empirically, CLEF and C4.5 were run on twenty classification tasks, mostly from the UCI data repository (Murphy & Aha, 1994). This allows for a comparison in terms of classification accuracy. In addition, several measurements of interrest were taken on CLEF during these runs. All values are computed from a ten-fold stratified cross-validation, with CLEF and C4.5 using identical folds for each task.

CLEF was trained by repeatedly sampling at random N times from the training set, where $N = 100|\mathbf{X}|$. Training stops either when the instances in the training set are perfectly separated, or after a maximum number of iterations. For C4.5, the default settings were used (Quinlan, 1993), and pruning was turned on.

Table 3 shows the accuracy results of the two algorithms, in terms of the mean and standard deviation for each task. The results are mixed, which is supported by lack of significant difference in the algorithm means, as measured by a one-way analysis of variance. This is a surprise, given the prediction, but further scrutiny suggests a cause that has a remedy. CLEF has decisive wins on tictactoe, mplex-6, balance-scale and promoter. What distinguishes these tasks from the others?

In these tasks, the encoding of the original input variables shows a common property. The number of possible values for each of the variables is identical. For example, for tictactoe there are three possible values for each variable. Now recall how the ELF approximator constructs a new feature. It chooses a poorly fitting feature and specializes the bit with the greatest accumulated error. In any instance in which the input variables are coded in the same number of bits, the probability of any input bit having the value TRUE is equal, assuming that all the input instances are equiprobable. However, in a

Table 3. Accuracy Results

Task	Accuracy C4.5	Accuracy CLEF
audio-no-id	77.8 ± 6.6	79.1 ± 9.1
balance-scale	77.5 ± 3.2	92.5 ± 4.0
breast-cancer	75.5 ± 3.9	63.8 ± 6.4
bupa	64.6 ± 5.6	66.3 ± 7.6
cleveland	46.8 ± 5.4	46.8 ± 6.2
hepatitis	77.5 ± 5.7	81.9 ± 5.2
hungarian	78.3 ± 4.0	78.0 ± 5.8
led24	62.4 ± 9.4	61.9 ± 11.1
lenses	83.3 ± 22.4	76.7 ± 21.3
lymphography	78.0 ± 11.9	76.7 ± 8.0
monks-2	65.9 ± 0.0	92.3 ± 4.8
mplex-6	57.1 ± 19.2	91.4 ± 14.6
primary-tumor	40.9 ± 6.4	36.2 ± 7.4
promoter	77.3 ± 14.2	87.3 ± 6.0
soybean	92.2 ± 2.4	91.0 ± 3.2
switzerland	33.1 ± 7.7	34.6 ± 14.3
tictactoe	68.1 ± 2.3	78.4 ± 2.8
va	26.7 ± 7.7	32.4 ± 6.7
votes	96.6 ± 3.3	94.3 ± 3.4
zoo	91.8 ± 6.4	95.5 ± 6.1
	68.6	72.8

different task, for variables coded with different numbers of bits, this is not so. The probability of a bit corresponding to a low arity variable being on is higher than the probability of a bit being on for a high arity variable. Hence, ELF's method for accumulating bitwise errors has a bias towards bits coming from low arity variables. A simple adjustment will remove this bias: the error attributed to each bit has to be normalized with respect to the probability of that bit being on in an instance. This will be done in the near future.

CPU and memory costs are indicated in Table 4. The CLEF algorithm is much more costly computationally than C4.5, whose times are typically on the order of a few seconds. One can see that CLEF's cpu times are typically on the order of minutes, and sometimes hours. This does not present a particular concern. Spending extra time to achieve better accuracy is generally an obvious trade-off. CLEF does not require so much time as to be impractical.

Table 4. CLEF Parameters

Task	CPU	Size	Match train	Match test
audio-no-id	218.2 ± 42.2	88.0 ± 2.8	77.3 ± 0.8	77.3 ± 1.1
balance-scale	59.9 ± 37.8	39.0 ± 2.3	66.6 ± 1.9	67.0 ± 2.5
breast-cancer	206.8 ± 16.6	46.5 ± 3.2	40.8 ± 2.2	41.0 ± 2.5
bupa	1051.2 ± 126.7	44.3 ± 1.7	45.1 ± 1.7	44.9 ± 4.1
cleveland	1183.9 ± 263.2	92.2 ± 4.1	59.4 ± 2.6	59.6 ± 3.4
hepatitis	58.8 ± 18.2	17.4 ± 1.0	50.4 ± 4.9	51.7 ± 5.5
hungarian	258.9 ± 122.6	53.0 ± 3.3	65.5 ± 2.4	66.7 ± 3.4
led24	36.9 ± 9.4	76.8 ± 4.9	54.2 ± 1.3	53.6 ± 2.1
lenses	0.1 ± 0.0	11.4 ± 1.7	61.8 ± 4.4	61.8 ± 8.4
lymphography	22.9 ± 14.0	27.6 ± 2.2	53.6 ± 2.2	53.9 ± 4.1
monks-2	926.2 ± 515.1	59.3 ± 8.3	27.6 ± 2.2	27.8 ± 2.1
mplex-6	1.0 ± 0.8	11.5 ± 1.8	37.6 ± 2.2	37.4 ± 5.2
primary-tumor	188.0 ± 3.0	246.1 ± 5.1	31.8 ± 0.9	31.9 ± 1.3
promoter	26.9 ± 6.2	7.8 ± 0.4	64.8 ± 1.6	66.0 ± 5.1
soybean	1598.7 ± 147.8	88.6 ± 3.7	66.0 ± 2.2	66.1 ± 2.1
switzerland	128.3 ± 26.1	57.7 ± 3.2	67.9 ± 2.8	68.5 ± 2.7
tictactoe	5792.5 ± 236.0	241.6 ± 14.0	29.6 ± 1.1	29.6 ± 1.0
va	360.1 ± 51.6	80.2 ± 3.1	65.9 ± 2.5	66.1 ± 2.8
votes	22.3 ± 1.3	14.3 ± 1.4	46.3 ± 3.4	46.4 ± 3.9
zoo	1.5 ± 0.3	17.3 ± 0.9	73.9 ± 1.1	73.6 ± 2.0

Memory costs are not large. The table presents the memory requirements of the resulting classifier in terms of the total number of features present in the machine. CLEF typically constructs a small set of features, each of which consists of a simple bit pattern and a single weight. Finding a nonlinear separation of the instances does not require a large number of features.

In order to measure the degree of overlap of the features that form a classifier, the average percentage of features matching an instance was evaluated. The “match train” and “match test” columns show this measure respectively for the training and the test set. These values show that there is a lot of overlap in the features that are constructed. The matching factors have similar values on the training and the test set, which is a desirable property.

CLEF currently does nothing to avoid overfitting. This is needed because overfitting does occur. For some problems, when plotting test set accuracy during training, one can observe a rise and then a drop in accuracy. This will be improved soon, by checking for highly specialized features, and feature groups that have small intersections.

7 Summary

CLEF is a classification algorithm that constructs a Φ-machine to fit the multiclass data. By using the ELF function approximator, non-linear features are constructed as needed. The sequence of feature sets produced by ELF has the effect that CLEF produces a sequence of Φ-machine classifiers. This sequence will finally produce a Φ-machine that separates the instances, whether or not they are linearly separable in the input variables. By using CLEF, one obtains a separating Φ-machine that is based on all the training instances.

Acknowledgments

Carla Brodley provided helpful comments.

References

Cohen, P. R., & Jensen, D. (1997). Overfitting explained. *Proceedings of the Sixth International Workshop on Artificial Intelligence and Statistics.*

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis.* New York: Wiley & Sons.

Frean, M. (1990). *Small nets and short paths: Optimising neural computation.* Doctoral dissertation, Center for Cognitive Science, University of Edinburgh.

Mooney, R., Shavlik, J., Towell, G., & Gove, A. (1989). An experimental comparison of symbolic and connectionist learning algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 775-780). Detroit, Michigan: Morgan Kaufmann.

Murphy, P. M., & Aha, D. W. (1994). *UCI repository of machine learning databases*, Irvine, CA: University of California, Department of Information and Computer Science.

Nilsson, N. J. (1965). *Learning machines.* New York: McGraw-Hill.

Quinlan, J. R. (1985). Decision trees and multi-valued attributes. In Michie (Ed.), *Machine Intelligence.*

Quinlan, J. R. (1993). *C4.5: Programs for machine learning.* Morgan Kaufmann.

Rumelhart, D. E., Hinton, G. E., & Williams, R.J. (1986). Learning internal representations by error propagation. In Rumelhart & McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition.* Cambridge, MA: MIT Press.

Utgoff, P. E., & Precup, D. (1997). *Constructive function approximation*, (Technical Report 97-04), Amherst, MA: University of Massachusetts, Department of Computer Science.