A Genetic Algorithm for Classification by Feature Partitioning

H. Altay Güvenir
Computer Engr. and Info. Sci. Dept.
Bilkent University,
Ankara 06533 TURKEY
guvenir@trbilun

İzzet Şirin
Computer Engr. and Info. Sci. Dept.
Bilkent University,
Ankara 06533 TURKEY
sirin@trbilun

Abstract

In this paper we describe a method for hybridizing a genetic algorithm and a feature partitioning (FP) classification algorithm. Learning in FP is accomplished by storing the objects separately in each feature dimension as partitions of the set of values. A partition is expanded through generalization, which is limited by a generalization limit. Prediction in FP is done through a voting among the class predictions of each feature. The effect of the prediction of a feature in the voting is proportional with the weight of that feature. Feature partitioning method is implemented in the CFP (Classification by Feature Partitioning) algorithm. We use the genetic algorithm and a training data set to learn real-valued weights and generalization limits associated with each feature in that domain. The experimental results indicate that the genetic algorithm can be used to learn the domain dependent parameters of a feature partitioning classifier, and the resulting hybrid system can be used to create compact representations with very good predictive accuracy.

1 INTRODUCTION

In recent years, there has been a great deal of progress in the development of automated classification techniques. Several different representation techniques have been used to describe concepts for supervised learning tasks. Among others, these include decision trees in Quinlan (1986), instance-based representation in Aha and Kibler (1991) and hyperrectangles in Salzberg (1991). The representation of the concepts learned by the exemplar-based learning techniques stores only specific examples that are representatives of other several similar instances.

Previous implementations of this approach usually extend the nearest neighbor algorithm, in which some kind of similarity (or distance) metric is used for prediction. Hence, prediction complexity of such algorithms is proportional to the number of instances (or objects) stored. Kelly and Davis have applied a hybrid genetic algorithm for the K Nearest Neighbors (KNN) classification algorithm (Kelly & Davis 1991). Their algorithm, called the GA-WKNN (for Genetic Algorithm with Weighted K Nearest Neighbor), combines the optimization capabilities of a genetic algorithm with the classification capabilities of the weighted KNN algorithm (WKNN). The goal of the GA-WKNN algorithm is to learn a feature weight vector which improves the common k nearest neighbor algorithm.

Feature partitioning (FP) is another form of exemplar-based learning. The FP technique makes several significant improvements over other exemplar-based learning algorithms, where the examples are stored in memory without any change in the representation. For example, IBL (Instance Based Learning) algorithms learn a set of instances (a representative subset of all training examples), EACH (Exemplar-Aided Constructor of Hyperrectangles) learns a set of hyperrectangles of the examples. On the other hand, the FP method stores the instances as factored out by their feature values. The program that implements the feature partitioning technique in this paper is called CFP, for Classification by Feature Partitioning. In CFP an example is described in terms of a set of *features* (essentially property/value pair). The CFP partitions the set of possible values for a feature into a disjoint sets corresponding to concepts. Therefore, the concept description learned by the CFP is a collection of feature partitions. In other words, the CFP learns a projection of the concept on each feature dimension.

Since the CFP learns projections of the concepts, it does not use any similarity (or distance) metric for prediction. Each feature contributes to the prediction process by its local knowledge. Final prediction is based on a voting among the predictions of the features. The strength of the contribution of a feature

in the voting process is determined by the weight of that feature. Assigning variable weights to the features modifies the importance of each feature to reflect its relevance for classification. The feature weights are domain dependent parameters of the CFP algorithm, and their selection is an optimization problem.

Because the feature partitions are disjoint sets of values, the CFP algorithm pays attention not to over generalize the partitions. In the CFP algorithm this is done by limiting the generalization of partitions. When generalizing a partition to cover a point, the distance between the new point and the existing partition must be less than a given generalization limit set for that feature. The generalization limits are domain dependent, and their selection is also an optimization problem.

In this paper, a hybrid system, called GA-CFP, will be described which combines a genetic algorithm with the CFP algorithm. The genetic algorithm is used to determine the domain dependent feature weights and generalization limits. It is a difficult problem to find an optimum setting of these parameters. In this paper we show how to use a real-valued genetic algorithm to find setting of feature weights and generalization limits that are good in the sense that they out perform the cases where the feature weights are identical and the generalization limits are set to two extremes.

The precise details of the CFP algorithm are described in the next section, and the process of partitioning of a feature dimension is illustrated through an example. Section 3 describes the use of genetic algorithm in the GA-CFP algorithm. Section 4 presents and empirical evaluation of the GA-CFP algorithm, and a comparison with other techniques on six different data sets. The final section will summarize the conclusions.

2 THE CFP ALGORITHM

The CFP program learns the projection of the concepts over each feature dimension. In other words, the CFP learns partitions of the set of possible values for each feature. An example is defined as a vector of features values plus a label that represents the class of the example. Partition is the basic unit of representation in the CFP algorithm. For each partition, lower and upper bounds of the feature values, the associated class and the number of instances it represents are maintained.

Initially, a partition is a point (lower and upper limits are equal) on the line representing the feature dimension. For instance, suppose that the first example e_1 of class C_1 is given during the training phase (Fig. 1.a). If the value of e_1 for feature f is x_1, then the set of possible values for feature f will be partitioned into three partitions: $< [-\infty, x_1], undetermined, 0 >$, $< [x_1, x_1], C_1, 1 >$, $< [x_1, \infty], undetermined, 0 >$; where

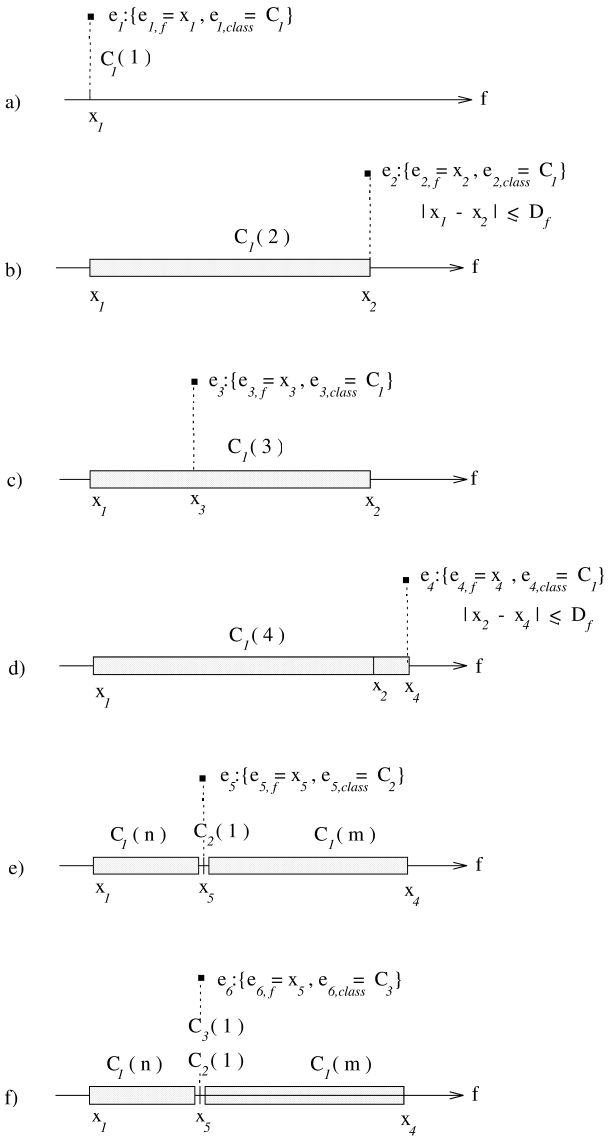

Figure 1: Partitioning of a feature dimension f.

the first element indicates the range of the partition, the second its class, and the third, called the representativeness, the number of examples represented by the partition.

A partition can be extended through generalization with other neighboring points in the same feature dimension. Assume that the second example e_2 is close to e_1 in feature f and also of the same class. In that case the CFP algorithm will generalize the partition for x_1 into an extended partition: $< [x_1, x_2], C_1, 2 >$, which now represents two examples (see Fig. 1.b). Generalization of a range partition is illustrated in Fig. 1.d.

Since partitions are disjoint the CFP algorithm pays attention to avoid over generalization. In order to gen-

```
update-feature-partitioning(f, e_f):
begin
  if class of partition(f, e_f) = e_class
    increment representativeness of partition(f, e_f)
  else {different class}
    if partition(f, e_f) is a point partition
      insert-new-partition(f, e_f)
    else {partition(f, e_f) is a range partition}
      if class of partition(f, e_f) is not undetermined
        subdivide-partition(partition(f, e_f), e_f)
      else to generalize}
        if the nearest partition to left or right in D_f
          distance has the class e_class
          generalize(partition,e_f)
        else {there are no partitions in D_f distance
              with the same class as e}
          insert-new-partition(f, e_f)
end
```

Figure 2: Updating a feature partition.

eralize a partition in feature f to cover a point, the distance between them must be less than a given generalization limit (D_f). Otherwise, the new example is stored as another point partition in the feature dimension f.

If the feature value of a training example falls in a partition with the same class, then simply the representativeness value is incremented by one (see Fig. 1.c).

If the new training example falls in a partition with a different class than that of the example, the CFP algorithm specializes the existing partition by dividing it into two range partitions and inserting a point partition (corresponding to the new example) in between them (see Fig. 1.e, f). When a partition is divided into two partitions, it is important to distribute the representativeness of the old partition to the newly formed partitions The CFP distributes the representativeness of the old partition among the new ones in proportion to their sizes.

The training process in CFP algorithm consists of updating the feature partitionings for each example and for each feature of these examples. Fig. 2 summarizes the process of updating the partitions of a feature f by the e_f value of an example for that feature.

The prediction process of the CFP is based on a voting taken among the predictions made by each feature separately (Fig. 3). For a given instance e, the prediction based on a feature f is determined by using the value of e_f as a index on the partitions of f. If e_f falls properly within a partition with a known class then the prediction is the class of that partition. If e_f falls in a point partition then among all the partitions at this point the one with the highest representativeness value is chosen. If e_f falls in a partition with no known class value, then no prediction is made for that feature. The predicted class of a given instance is the one which receives the highest amount of votes among all predictions. In GA-CFP a genetic algorithm is used to learn the feature weights for a given domain.

```
prediction(e):
begin
    foreach feature f
        c = class of partition(f, e_f)
        vote_c = vote_c + w_f
    return class c with highest vote_c.
end
```

Figure 3: Prediction in the CFP.

The prediction process in exemplar-based learning algorithms which use some form of the nearest neighbor algorithm (such as EACH and IBL) involves computing the Euclidean distance (or similarity) of the instance to each stored exemplar in each dimension. Therefore, if there are n exemplars stored in the memory, and k features are used, then the complexity of the prediction is $O(kn)$. On the other hand, since the partitions are naturally sorted for each feature dimension, the prediction process in the CFP algorithm is only $O(k \log n)$. This feature of the CFP algorithm significantly reduces the prediction complexity.

Another important feature of the CFP algorithm is its low memory requirement. Since, the CFP learns each feature partitions independently, number of the partitions for each feature may be different. If appropriate D_f generalization limits are chosen, the CFP may significantly reduce memory requirement. The selection of D_f values is domain dependent, and is an optimization problem. The details of the CFP algorithm are given in Sirin (1993). In GA-CFP a genetic algorithm is used to learn the values of the generalization limits for a given domain.

The issue of unknown attribute values is an unfortunate fact of real-world data sets, that data often contain missing attribute values. Most common approaches are compared in Quinlan (1993), leading to a general conclusion that some approaches are clearly inferior but no one approach is uniformly superior to others. Since CFP treats each attribute value separately, in the case of an unknown attribute value, it simply leaves the partitioning of that feature intact.

3 THE GA-CFP ALGORITHM

The GA-CFP algorithm combines the optimization capabilities of a genetic algorithm with the classification capabilities of CFP. The goal of the algorithm is to

```
CFP(training-set, test-set):
begin
      /* train */
      foreach e in training-set
         foreach feature f
            update-feature-partitioning(f, e_f)
      /* test */
      correct-count = 0
      foreach e in test-set
         P=prediction(e)
         if P = e_class then
            increment correct-count
      return correct-count / | test-set |
end
```

Figure 4: The CFP fitness function of the Genetic Algorithm.

learn the weights and the generalization limits for each feature, both of which are real-valued. The genetic algorithm we used is GAucsd 1.4. We used the standard operators of genetic algorithms, namely, reproduction, crossover and mutation. The chromosomes are treated as rings and crossover is done by exchanging the sections between two crossover points.

Chromosomes are vectors of real-valued weights and generalization limits for each feature. Each chromosome is a vector of decimal numbers between 0 and 1 inclusive. A vector value is associated with the weight and the generalization limit for each feature. Thus the length of the vector is twice the number of the features. The initial population of chromosomes in each run of the GA-CFP algorithm was randomly generated. The fitness function used to evaluate the chromosomes is the accuracy of the CFP algorithm with the weights and generalization limits encoded in the chromosome (see Fig. 4). In order to compute the fitness value of a chromosome, the CFP algorithm is trained with the examples in the training set using the feature weights and generalization limits, and then tested with the examples in the test set. The fitness value computed as the ratio of the correctly predicted test examples to the size of the test set.

4 EMPIRICAL STUDIES WITH GA-CFP

The GA-CFP algorithm has been tested using real and artificial data from six different problem domains: (1) *classifying iris flowers*, (2) *predicting survival for heart attack victims*, (3) *Hungarian database* from Hungarian Institute of Cardiology, (4) *Cleveland database* from Cleveland Clinic Foundation, (5) *Classifying waveforms*, and (6) *Classifying glass types*. The use of real data in these tests provide a measure of the system's performance on noisy and incomplete data sets, and most importantly, allowed comparisons between GA-CFP and other systems.

Table 1: Comparison of GA-CFP with regular CFP in terms of accuracy (%)

Database	**CFP** $D_f = 0$	**CFP** $D_f = max$	**GA-CFP**
Iris	88.0	92.7	98.0
Echocardiogram	63.5	71.6	83.8
Glass	44.4	86.0	92.1
Hungarian	68.7	75.5	91.5
Cleveland	77.6	70.3	87.5
Waveform	53.4	62.0	88.0

Each run of the genetic algorithm maintained a population of size 100. Crossover probability is 0.6 and mutation probability is 0.008. The maximum number of fitness function evaluations in each GA run changed between 2000 and 8000 depending on the size on the chromosomes. Each data set is described briefly and experimental results are presented below.

Iris Flowers: Iris flowers data set from Fisher (1936) consists of four integer-valued features and a particular species of iris. There are three different species (classes): *virginica*, *setosa*, and *versicolor*. The data set contains 150 instances.

Echocardiogram: Echocardiogram data set contains 74 records from people who had recently suffered acute myocardial infarctions (heart attacks). This data set contains small number of examples, but it provides an opportunity to compare CFP to other systems. In addition, this data is real, noisy and incomplete. The goal of physicians using these measurements is to predict a patient's chances of survival. This data set contains six measurements and the outcome (patient survived or not at the end of the one year period after the heart attack).

Medical Diagnosis: The CFP was tested on widely used two medical databases, namely the Cleveland and Hungarian databases. The Cleveland and Hungarian data sets contain heart disease diagnoses collected from the Cleveland Clinic Foundation and Hungarian Institute of Cardiology, respectively. Diagnosis are described by 13 numeric-valued features (e.g. age, fasting blood sugar level etc.). The objective here is to determine whether a patient has a heart disease. The Cleveland data set consists of 303 instances and the Hungarian data set consists of 294 instances.

Waveform: The waveform data set is artificial and consists of 21 numeric-valued features, which have values between 0 and 6. There are three different types of waveforms and they are equally distributed. The objective here is to determine the type of waveform. Each

feature includes noise (with mean 0 and variance 1).

Glass: This data set consists of attributes of glass samples taken from the scan of an accident.[1] Each of the 214 examples is a member of one of six classes. There are nine features.

In order to see the effect of a genetic algorithm in learning the feature weights and generalization parameters of the CFP, we compared the GA-CFP with the regular CFP algorithm. The comparison of GA-CFP algorithm with regular CFP algorithm, where the feature weights are identical and the generalization limits are set to two extremes is shown in Table 1. In the first extreme, for all feature D_f's are set to zero disabling any generalization. In the second extreme, for all feature D_f's are set to maximum value ($max_f - min_f$), resulting in the maximum possible generalization.

The methodology used in the testing of GA-CFP algorithm was the *leaving-one-out cross-validation* technique. A robust estimate of accuracy on unseen examples can be obtained by this technique. Cross-validation involves removing mutually exclusive test sets of examples from the data set. For each test set, the remaining examples serve as a training set, and classification accuracy is measured as the average accuracy on all the test sets. The leaving-one-out method involves removing exactly one example from the data and training on the remaining examples. The technique is repeated for every example in the data set and accuracy is measured across all examples.

It is clear from Table 1 that the genetic algorithm has learned weight and generalization limits for which the accuracy obtained is much better than the two extreme cases. It is also interesting to note that allowing maximum generalization results in a better accuracy than the case where no generalization is allowed, except for the Cleveland data set.

We have also compared the GA-CFP algorithm with instance-based learning algorithms, decision tree learning system C4.5, and GA-WKNN. In these comparisons the GA-CFP system is run and tested in the same way as the reported methods of the compared system. The classification accuracy of the GA-CFP algorithm is the fitness value of the best chromosome obtained in the final population.

The results of the comparison of the GA-CFP with the reported results of the instance based learning algorithms IB1, IB2, and IB3 (Aha, Kibler & Albert, 1991) are given in Table 2. All results reported were averaged over 50 trials. The training and test sets were always disjoint. The instances were drawn randomly from the data sets.

We have compared GA-CFP with decision tree learning algorithm C4.5 (Quinlan 1993), as well (Table 3). In these comparisons we applied the leaving-one-out cross-validation accuracy estimation technique to both algorithms. We used the decision trees that are generated after pruning, since these trees performed better than the trees before pruning in general.

In comparing GA-CFP with the reported results of the GA-WKNN, we used cross-validation accuracy estimation technique. Each data set was divided into five disjoint partitions. The only constraint on otherwise random partitioning was that classes be represented equally in each partition. We generated five training/test sets for each data set. Four-fifth of the data were used for training and the remaining fifth was used for testing. The results are shown in Table 4.

Although the results of the GA-CFP algorithm are better than other classification systems, the use of genetic algorithm is costly. This is because the computation of the fitness function requires the execution of the CFP algorithm several times due to the cross-validation methods we used. However, an important characteristic of the feature weight and generalization limits parameters of the CFP algorithm is that these parameters are domain dependent. Therefore, the genetic algorithm can be used only with a portion of all the data available. We have trained the GA-CFP system with only 1/10, 1/5, 1/4, 1/3, 1/2, 2/3, and 3/4

Table 2: Comparison of GA-CFP with IBL in terms of accuracy (%)

Database	**IB1**	**IB2**	**IB3**	**GA-CFP**
Hungarian	58.7	55.9	80.5	91.4
Cleveland	75.7	71.4	78.0	94.3
Waveform	75.2	69.6	73.8	86.5

Table 3: Comparison of GA-CFP with C4.5 in terms of accuracy (%)

Database	**C4.5**	**GA-CFP**
Iris	95.3	98.0
Echocardiogram	77.0	83.8
Glass	70.1	92.1
Hungarian	83.3	91.5
Cleveland	80.5	87.5
Waveform	88.0	88.0

Table 4: Comparison of GA-WKNN and GA-CFP in terms of accuracy (%)

Database	**GA-WKNN**	**GA-CFP**
Iris	94.0	97.3
Glass	62.2	71.7

[1] Collected by B. German of the Home Office Forensic Service at Aldermaston, Reading, Berkshire in the UK.

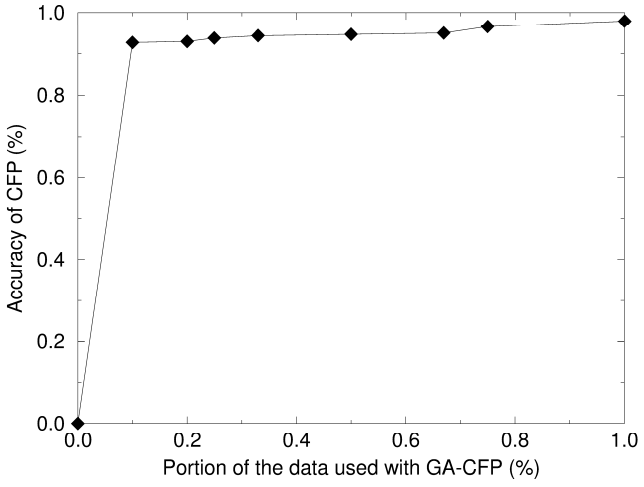

Figure 5: Learning curve of domain dependent parameters.

of the Iris data set. Then, we measured the accuracy of the CFP algorithm on the complete set of data with the parameters learned by the genetic algorithm (see Fig. 5). We used the cross-validation method in these experiments; that is, 0.92 is the average of 10 runs in each of which a disjoint set of 1/10 of the data is used in training and the remaining 9/10 in the test. In determining the fitness value of a chromosome, to avoid the dependency on the order of the examples in the data set, leaving-one-out cross-validation method is used. For example, in the run with 1/10 of data, the fitness value of a chromosome is the average of 15 runs of the CFP with the parameters it encodes; in each run 14 examples were used in training and the remaining one in the test.

It is clear from these experiments that the genetic algorithm can determine a very good set of domain dependent parameters of CFP, even when trained with a small portion of the data set. Obviously, the larger of the portion of the data is used, the better parameters are found.

5 CONCLUSION

We have presented an algorithm that hybridizes the classification power of the feature partitioning CFP algorithm with the search and optimization power of the genetic algorithm. The resulting algorithm GA-CFP requires more computational capabilities than the CFP algorithm, but achieves improved classification performance in reasonable time. Experimental results indicate that the GA-CFP algorithm outperforms other classification techniques such as IBL, C4.5 and GA-WKNN.

We have also noticed that the genetic algorithm can be trained with only a small portion of the data to learn the domain dependent parameters of the CFP algorithm with satisfactory prediction accuracy. We anticipate that extension to the research will improve the algorithm's performance. In further work we plan to incorporate other genetic algorithm techniques such as eliticism, and other techniques for learning real-valued weight vectors.

References

D. W. Aha, D. Kibler and M. K. Albert (1991). Instance–Based Learning Algorithms. *Machine Learning* **6**:37–66.

R. A. Fisher (1936). The use of multiple in taxonomic problems. *Annals of Eugenics* **7**:179–188.

I. Sirin and H. A. Guvenir (1993). *An Algorithm for Classification by Feature Partitioning.* Technical Report CIS-9301, Bilkent University, Dept. of Computer Engr. and Info. Sci., Ankara, Turkey.

J. D. Kelly and L. Davis (1991). A Hybrid Genetic Algorithm for Classification. In *Proceedings IJCAI-91*, 645-650, Sydney, Australia.

J. R. Quinlan (1986). Inductions of Decision Trees. *Machine Learning* **1**:81-106.

J. R. Quinlan (1993). *C4.5: Programs for Machine Learning.* California: Morgan Kaufmann.

S. Salzberg (1991). A Nearest Hyperrectangle Learning Method. *Machine Learning*, **6**:251-276.