Parallel Classification by Feature Partitioning

Hüseyin Simitçi H. Altay Güvenir

Computer Engineering and Information Science Department
Bilkent University, Ankara 06533 TURKEY
E-mail: {simitci,guvenir}@trbilun.BITNET

Abstract

This work presents a parallel method for learning from examples using *parallel feature partitioning* (PFP). Feature partitioning (FP) is an inductive, incremental and supervised learning method proposed by Şirin and Güvenir [1]. PFP assigns feature dimensions to separate nodes. Learning in PFP is accomplished by storing the objects separately in each feature dimension as disjoint partitions of values. Every node expands a partition, which is initially a point in the feature dimension through generalization. The CFP algorithm specializes a partition by subdividing it into sub-partitions.

PFP is implemented in the PCFP (Parallel Classification by Feature Partitioning) algorithm. PCFP is tested in six different domains, and results are compared with CFP of Şirin and Güvenir [1].

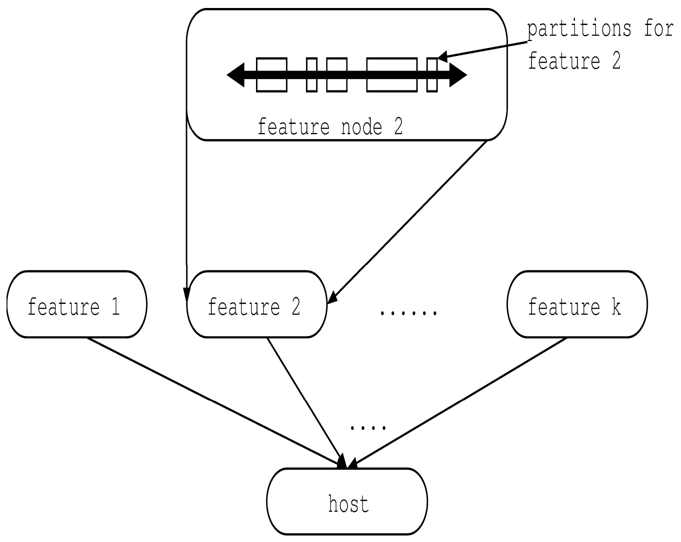

Figure 1: Topology of the feature nodes and the host node

1 Introduction

This work presents a parallel method for learning from examples using em parallel feature partitioning (PFP). Feature partitioning (FP) is an inductive, incremental and supervised learning method proposed by Şirin and Güvenir[1].

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. The FP method stores the instances as factored out by their feature values. This form of representation is very appropriate for parallel implementations. PFP assigns feature dimensions to separate parallel working nodes. The program that implements the parallel feature partitioning technique in this work is called PCFP, for Parallel Classification by Feature Partitioning. In PCFP every node partitions its feature into segments corresponding to concepts. Therefore, the concept description learned by the PCFP is a collection of feature partitions on each node. In other words, the PCFP learns a projection of the concept on each feature dimension which are also stored in separate nodes.

Each feature node contributes to the prediction process by its local knowledge. Final prediction is based on a voting among the predictions of the feature nodes, which is accomplished on a predetermined host node. Since prediction process is running in all feature nodes in parallel, prediction complexity reduces by a factor of number of 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. This scheme allows smooth performance degradation when data set contains irrelevant features. Therefore, the PCFP algorithm is tolerant to noisy data. In case of an unknown attribute value, PCFP leaves that feature's partitioning

```
train(Training Set):
begin
    foreach feature $f$ do in parallel
        foreach $e$ in Training Set
            if class of partition($f, e_f$) $= e_{class}$
                then $w_f = (1 + \Delta)w_f$
                else $w_f = (1 - \Delta)w_f$
            update-feature-partitioning($f, e_f$)
end
```

Figure 2: Training algorithm of the PCFP.

intact.

The precise details of the CFP algorithm can be found in [1]. Next section gives the details of PCFP. Section 3 gives empirical evaluation of PCFP and compares it with CFP algorithm on six different data sets.

2 The PCFP Algorithm

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

The virtual topology required by PCFP is a set of feature nodes each connected to a host node (Fig. 1). The training process in PCFP algorithm has two steps: learning of feature weights and learning of feature partitionings (Fig. 2). For each training example, the prediction of each feature is compared with the actual class of the example in parallel. If the prediction of feature is correct the weight of that feature is incremented by Δ (global feature weight adjustment rate) percent; otherwise, it is decremented by the same amount.

The prediction process of the PCFP is based on a voting in the host taken among the predictions made by each feature node in parallel (Fig. 3). The effect of the prediction of a feature in the voting is proportional with the weight of that feature. The predicted class of a given instance is the one which receives the highest amount of votes among all predictions.

Since the partitions are naturally sorted for each feature dimension, the predic-

```
prediction($e$):
**begin**
    **foreach** feature $f$ **do in parallel**
        $c$ = class of partition($f, e_f$)
    **foreach** feature $f$ **do on host**
        $vote_c = vote_c + w_f$
    **return** class $c$ with highest $vote_c$.
**end**
```

Figure 3: Prediction process of the PCFP.

tion process in the CFP algorithm is $O(k \log n)$, with n examples and k features. The PCFP algorithm reduces this prediction complexity to $O(\log n)$ by making predictions in each feature dimension in parallel using $O(k)$ processors.

3 Empirical Studies with PCFP

The PCFP algorithm is implemented on iPSC/2 parallel computer. iPSC/2 computer has a hypercube topology with nearest neighbor connectivity. For a database consisting of k features k number of node processors are allocated for PCFP.

The PCFP algorithm has been tested using real and artificial data from six different databases: (1) classifying iris flowers, (2) randomly generated artificial database with 6 features, (3) Glass database, (4) Hungarian database from Hungarian Ins. of Cardiology, (5) Cleveland database from Cleveland Clinic Foundation, (6) randomly generated artificial database with 15 features. The description of databases are given in Şirin [2].

In tests the speedup is measured against sequential CFP algorithm. In the experiments 70 percent of the database is used for training and remaining part is used for testing. The achieved speedups for the prediction phase is shown in Table 1. In training phase slightly better speedups are obtained, because load-balancing is slightly better in the beginning of the training.

On most of the databases above or near linear speedup is achieved (see Fig. 4). The deviations in the number of partitions in each feature dimension violates load balancing. Fig. 6 shows that efficiency degrades as the ratio of standard deviation of partitions over average number of partitions increases in each feature. This is the case in databases Cleveland and Hungarian. When load balancing is achieved thorough balanced number of partitions, efficiency is quite high (Fig. 5).

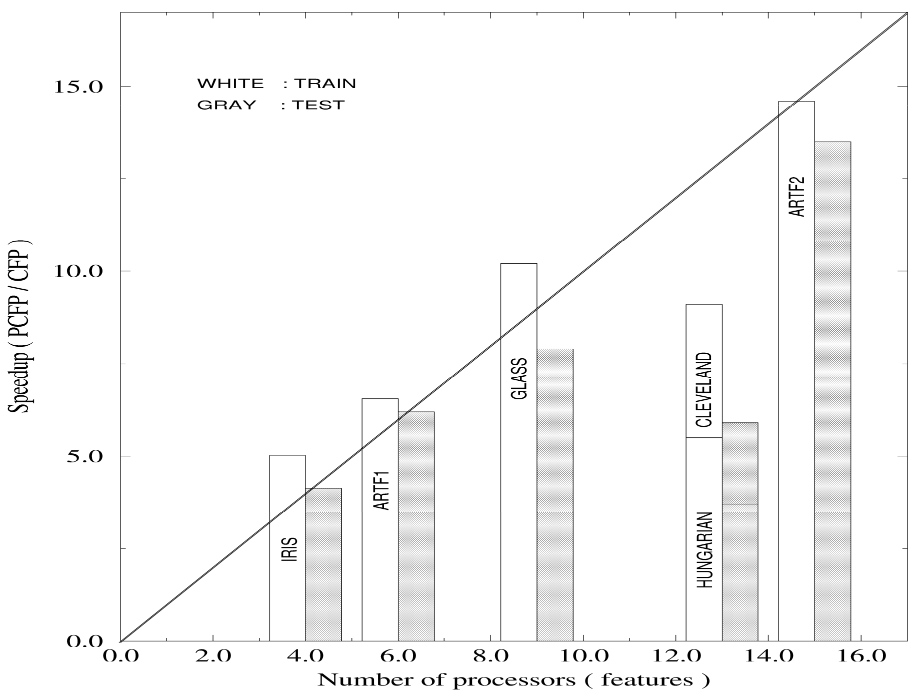

Figure 4: Achieved speedups with different number of processors

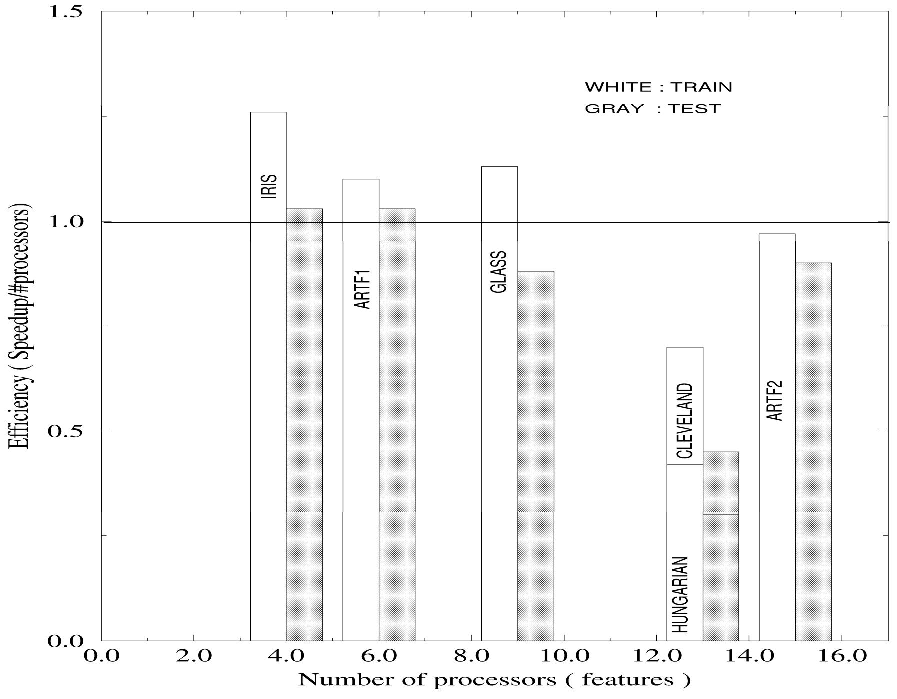

Figure 5: Achieved efficiencies with different number of processors

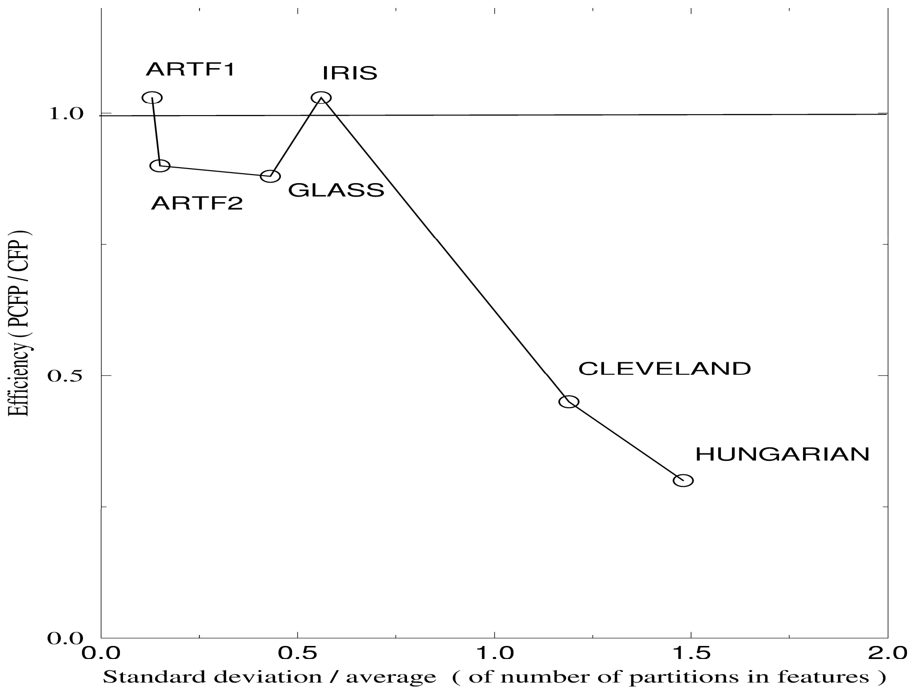

Figure 6: Efficiency (test) vs. deviations in number of partitions

4 Conclusion

This work presents a new parallel algorithm for learning from examples, called parallel feature partitioning (PFP).

PFP assigns feature dimensions to separate nodes. Learning in PFP is accomplished by storing the objects separately in each feature dimension as disjoint partitions of values. Every node expands a partition, which is initially a point in the feature dimension through generalization. The PFP algorithm specializes a partition by subdividing it into sub-partitions.

Table 1: Achieved Efficiencies in Prediction

Database	No. of Features	Speedup	Efficiency
Iris	4	4.1	1.03
Artificial1	6	6.2	1.03
Glass	9	7.9	0.88
Hungarian	13	3.7	0.30
Cleveland	13	5.9	0.45
Artificial2	15	13.5	0.90

PFP is implemented in the PCFP (Parallel Classification by Feature Partitioning) algorithm. PCFP is tested in six different domains, and results are compared with CFP of Şirin and Güvenir [1].

PCFP algorithm has a prediction complexity of $O(\log n)$ using n examples having k features with $O(k)$ processors. When the deviation in number of partitions in each feature dimension is small load balance between processors is achieved and the prediction complexity of CFP which is $O(k \log n)$ is reduced by a factor of number of features, k.

References

[1] İ. Şirin and H. A. Güvenir. *Classification by Feature Partitioning*, Technical Report CIS-9301, Computer Engineering and Information Science Dept. of Bilkent University, Ankara 06533, Turkey.

[2] İ. Şirin and H. A. Güvenir. *Empirical Evaluation of CFP*, Technical Report CIS-9310, Computer Engineering and Information Science Dept. of Bilkent University, Ankara 06533, Turkey.