Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008)

# Constraint Projections for Ensemble Learning

**Daoqiang Zhang**[1] **Songcan Chen**[1] **Zhi-Hua Zhou**[2*] **Qiang Yang**[3]
[1]Department of Computer Science & Engineering, Nanjing University of Aeronautics & Astronautics, China
[2]National Key Laboratory for Novel Software Technology, Nanjing University, China
[3]Department of Computer Science & Engineering, Hong Kong University of Science & Technology, China
{dqzhang, s.chen}@nuaa.edu.cn zhouzh@nju.edu.cn qyang@cse.ust.hk

## Abstract

It is well-known that *diversity* among base classifiers is crucial for constructing a strong ensemble. Most existing ensemble methods obtain diverse individual learners through resampling the instances or features. In this paper, we propose an alternative way for ensemble construction by resampling pairwise constraints that specify whether a pair of instances belongs to the same class or not. Using pairwise constraints for ensemble construction is challenging because it remains unknown how to influence the base classifiers with the sampled pairwise constraints. We solve this problem with a two-step process. First, we transform the original instances into a new data representation using projections learnt from pairwise constraints. Then, we build the base classifiers with the new data representation. We propose two methods for resampling pairwise constraints following the standard Bagging and Boosting algorithms, respectively. Extensive experiments validate the effectiveness of our method.

## Introduction

Ensemble learning is a learning paradigm where multiple learners are combined to solve a problem. Since it can significantly improve the generalization ability of a single classifier, ensemble learning has attracted a lot of attentions during the past decade (Kuncheva 2004). Generally, the design of a classifier ensemble contains two subsequent steps, i.e. constructing multiple base classifiers and then combining their predictions. In this paper, we focus on the first problem and adopts a simple *majority voting* scheme to combine predictions of multiple base classifiers.

Many methods have been developed for constructing ensembles. Among them, *Bagging* (Breiman 1996), *Boosting* (Freund & Schapire 1996), and *Random Subspace* (Ho 1998) are three general techniques widely used in many tasks. Both Bagging and Boosting train base classifiers by resampling training instances, while Random Subspace trains classifiers using different random subsets of input features. One difference between Bagging and Boosting lies in that the former obtains a *bootstrap replicate* by uniformly sampling with replacement from the original training set, while the latter resamples or reweights the training data by emphasizing more on instances that are misclassified by previous classifiers.

It is known that in order to get a strong ensemble, the component classifiers should be with high accuracy as well as high diversity (Kuncheva 2004). However, achieving such goals is not easy. In this paper, we propose a novel way for constructing ensembles through resampling *pairwise constraints*. Here the pairwise constraints specify whether a pair of instances belong to the same class (*must-link* constraints) or not (*cannot-link* constraints). Such kinds of constraints have been widely used in several fields of machine learning, such as semi-supervised clustering (Davidson & Basu 2007). Pairwise constraints can be given in advance, or generated from class labels. Given $n$ labeled instances, we can derive approximately $n^2$ pairwise constraints. Sampling pairwise constraints may help the base learners to have higher diversity because for $n$ instances, there are at most $2^n$ different results for sampling instances, but at most $2^{n^2}$ different results for sampling constraints. To the best of our knowledge, no previous ensemble learning research has tried to build ensembles by exploiting pairwise constraints.

In this paper, we will address the following issues regarding using pairwise constraints to build strong ensembles:

- How to use pairwise constraints to build component classifiers?
- How to resample pairwise constraints to obtain diverse classifiers?
- Is the performance of resampling pairwise constraints comparable with those of resampling instances (such as Bagging and Boosting) and resampling input features (such as Random Subspace)?

To answer the first question, we develop a pairwise constraints preserving projection and use it to project original instances into a new data representation, through which we transfer the information in pairwise constraints into the new data representation. Then we build base classifiers based on the new representation. To answer the second question, we propose two methods for resampling pairwise constraints following the styles of Bagging and Boosting, respectively.

---

[*]The authors were partially supported by NSFC (60505004, 60635030,60721002,60773061), JiangsuSF (BK2006521) and Hong Kong CERG (621307).

Finally, we answer the third question by carrying out extensive experiments on a broad range of benchmark data sets from the UCI repository (Blake, Keogh, & Merz 1998) to evaluate the proposed methods.

The rest of this paper is organized as follows: in next section, we briefly review some related work. Then we propose the **COPEN** (pairwise **CO**nstraints **P**rojection based **EN**semble) method and report the experimental results, and finally we conclude this paper and point out some future work.

## Related Work

Kuncheva (2004) summarized four fundamental approaches for building ensembles of diverse classifiers: 1) using different combination schemes; 2) using different base classifiers; 3) using different feature subsets; 4) using different data subsets. We are more interested in the last two approaches, i.e. constructing classifier ensembles by manipulating the data (including features and data samples). In fact, most existing methods fall into these two categories. For example, Bagging and Boosting belong to the 4th category, and Random Subspace belongs to the 3rd category. Another ensemble method of the 3rd category is *ensemble feature selection* (Opitz 1999), which uses a genetic algorithm to generate feature subsets instead of random sampling in Random Subspace.

Some methods jointly manipulate feature (or subspace) and data samples. For example, Breiman's (2001)*Random Forest* integrated both merits of Bagging and Random Subspace, using decision trees as the base classifiers. Zhou and Yu (2005) proposed a multimodal perturbation on features, data samples and base classifier parameters to ensemble nearest neighbor classifiers. Rodríguez *et al.* (2006) proposed the *Rotation Forest* method which first randomly partitions feature sets into $K$ subsets, performs principal component analysis (PCA) on each subsets of a bootstrap replicate, and then reassembles those PCA projective vectors into a rotation matrix to form new features for a base classifier. Wang and Tang (2006) proposed a new random subspace method by randomly sampling PCA projective vectors and integrated it with Bagging for subspace face recognition. More recently, García-Pedrajas *et al.* (2007) proposed a nonlinear boosting projections method for ensemble construction, where neural networks were used to learn a projection with more emphasis on previously misclassified instances similarly as in Boosting.

Pairwise constraints (also called as *side information*) have been popularly used in areas such as semi-supervised clustering (Davidson & Basu 2007). Recently, pairwise constraints are also used for semi-supervised dimensionality reduction. Bar-Hillel *et al.* (2005) proposed the relevant component analysis (RCA) method using only equivalent (must-link) constraints. Tang *et al.* (2007) proposed a feature projection method using cannot-link constraints. Zhang *et al.* (2007) proposed the semi-supervised dimensionality reduction using both must-link and cannot-link constraints as well as unlabeled data. More recently, Liu *et al.* (2007) proposed a boosting framework for semi-supervised clustering. All above mentioned methods aimed at using pairwise constraints for semi-supervised learning, which is apparently different from our goal in this paper.

## The COPEN Method

In this section, we describe our pairwise constraints based ensemble learning algorithm, called `COPEN`. Before that, we first introduce the *Constraint Projection* algorithm, which is one of the key ingredients of COPEN. We derive two versions of the algorithm, `COPEN.bag` and `COPEN.boost`, following the standard Bagging and Boosting methods, respectively.

### Constraint Projection

Given a set of $p$-dimensional data $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$, and the corresponding pairwise must-link constraint set $\mathcal{M}$=$\{(\boldsymbol{x}_i, \boldsymbol{x}_j)|\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to the same class$\}$ and pairwise cannot-link constraint set $\mathcal{C}$=$\{(\boldsymbol{x}_i, \boldsymbol{x}_j)|\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to different classes$\}$, Constraint Projection seeks a set of projective vectors $\boldsymbol{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2, ..., \boldsymbol{w}_d]$, such that the pairwise constraints in $\mathcal{C}$ and $\mathcal{M}$ are most faithfully preserved in the transformed lower-dimensional representations $\boldsymbol{z}_i = \boldsymbol{W}^T\boldsymbol{x}_i$. That is, examples involved by $\mathcal{M}$ should be close while examples involved by $\mathcal{C}$ should be far in the lower-dimensional space.

Define the objective function as maximizing $J(\boldsymbol{W})$ w.r.t. $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}$,

$$J(\boldsymbol{W}) = \frac{1}{2n_{\mathcal{C}}}\sum_{(\boldsymbol{x}_i,\boldsymbol{x}_j)\in\mathcal{C}} \|\boldsymbol{W}^T\boldsymbol{x}_i - \boldsymbol{W}^T\boldsymbol{x}_j\|^2 - \frac{\gamma}{2n_{\mathcal{M}}}\sum_{(\boldsymbol{x}_i,\boldsymbol{x}_j)\in\mathcal{M}} \|\boldsymbol{W}^T\boldsymbol{x}_i - \boldsymbol{W}^T\boldsymbol{x}_j\|^2 \quad (1)$$

where $n_{\mathcal{C}}$ and $n_{\mathcal{M}}$ are the cardinality of the cannot-link constraint set $\mathcal{C}$ and the must-link constraint set $\mathcal{M}$, respectively, and $\gamma$ is a scaling coefficient.

The intuition behind Eq. 1 is to let the average distance in the lower-dimensional space between examples involved by the cannot-link set $\mathcal{C}$ as large as possible, while distances between examples involved by the must-link set $\mathcal{M}$ as small as possible. Since the distance between examples in the same class is typically smaller than that in different classes, a scaling parameter $\gamma$ is added to balance the contributions of the two terms in Eq. 1 and its value can be estimated by

$$\gamma = \frac{\frac{1}{n_{\mathcal{C}}}\sum_{(\boldsymbol{x}_i,\boldsymbol{x}_j)\in\mathcal{C}} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{\frac{1}{n_{\mathcal{M}}}\sum_{(\boldsymbol{x}_i,\boldsymbol{x}_j)\in\mathcal{M}} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2} \quad (2)$$

Note that in Eq. 2, we do not necessarily compute all pairwise distances because usually only a sample of constraints will be contained in the constraint sets $\mathcal{C}$ and $\mathcal{M}$. With simple algebra, the objective function in Eq. 1 can be reformulated in a more convenient way as

$$J(\boldsymbol{W}) = trace(\boldsymbol{W}^T(\boldsymbol{S}_{\mathcal{C}} - \gamma\boldsymbol{S}_{\mathcal{M}})\boldsymbol{W}), \quad (3)$$

where $\boldsymbol{S}_{\mathcal{C}}$ and $\boldsymbol{S}_{\mathcal{M}}$ are respectively defined as

$$\boldsymbol{S}_{\mathcal{C}} = \frac{1}{2n_C}\sum_{(\boldsymbol{x}_i,\boldsymbol{x}_j)\in\mathcal{C}} (\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \quad (4)$$

**Algorithm 1: COPEN.bag**

```
Input: training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ with labels $y_i \in Y = \{1, ..., c\}$, base learning algorithm Learner, ensemble size $L$, number of must-link constraints $n_M$, number of cannot-link constraints $n_C$
Initialize: $ind = \{1, ..., n\}$.
For $l = 1, ..., L$
    1. Let $\mathcal{C} = \varnothing$, $\mathcal{M} = \varnothing$.
    2. Randomly draw a pair of numbers $i \neq j$ from $ind$ with replacement.
    3. If ($y_i = y_j$ and $|\mathcal{M}| < n_M$), add $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ into $\mathcal{M}$;
       elseif ($y_i \neq y_j$ and $|\mathcal{C}| < n_C$), add $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ into $\mathcal{C}$.
    4. Repeat steps 2-3, until $|\mathcal{M}| = n_M$ and $|\mathcal{C}| = n_C$.
    5. Compute the projective matrix $\boldsymbol{W}_l$ using Constraint Projection algorithm with $\mathcal{M}$ and $\mathcal{C}$.
    6. Call Learner, providing it with the training data set $\{\boldsymbol{W}_l^T \boldsymbol{x}_i, y_i\}_{i=1}^n$.
    7. Get a hypothesis $h_l : \boldsymbol{W}_l^T \boldsymbol{X} \rightarrow Y$.
Output: the final hypothesis
    $h_f(\boldsymbol{x}) = \arg\max_{y \in Y} \sum_{l:h_l(\boldsymbol{W}_l^T \boldsymbol{x})=y} 1$
```

**Algorithm 2: COPEN.boost**

```
Input: training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$ with labels $y_i \in Y = \{1, ..., c\}$, base learning algorithm Learner, ensemble size $L$, number of must-link constraints $n_M$, number of cannot-link constraints $n_C$, sampling threshold $r (0 < r < 1)$
Initialize: $ind = \{1, ..., n\}$.
For $l = 1, ..., L$
    1-7. The same as in COPEN.bag.
    8. Update the index set
       $ind = \{i | h_l(\boldsymbol{W}_l^T \boldsymbol{x}_i) \neq y_i, 1 \leq i \leq n\}$ .
    9. If $|ind| < \lfloor rn \rfloor$, randomly sample $\lfloor rn \rfloor - |ind|$ numbers from $\{1, ..., n\}$, add them into $ind$.
Output: the final hypothesis
    $h_f(\boldsymbol{x}) = \arg\max_{y \in Y} \sum_{l:h_l(\boldsymbol{W}_l^T \boldsymbol{x})=y} 1$
```

and

$$\boldsymbol{S}_{\mathcal{M}} = \frac{1}{2n_M} \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{M}} (\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \qquad (5)$$

In this paper, we call $\boldsymbol{S}_{\mathcal{C}}$ and $\boldsymbol{S}_{\mathcal{M}}$ as *cannot-link scatter matrix* and *must-link scatter matrix*, respectively, which reassembles the concepts of *between-class scatter matrix* and *within-class scatter matrix* respectively in linear discriminant analysis (LDA). The difference lies in that the latter uses class labels to generate scatter matrices, while the former uses pairwise constraints to generate scatter matrices.

Obviously, the problem expressed by Eq. 3 is a typical eigen-problem, and can be efficiently solved by computing the eigenvectors of $\boldsymbol{S}_{\mathcal{C}} - \gamma \boldsymbol{S}_{\mathcal{M}}$ corresponding to the $d$ largest eigenvalues. Suppose $\boldsymbol{W} = [\boldsymbol{W}_1, \boldsymbol{W}_2, ..., \boldsymbol{W}_d]$ are solutions to Eq. 3, and the corresponding eigenvalues are $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_d$. Denote $\boldsymbol{\Lambda} = diag(\lambda_1, \lambda_2, ..., \lambda_d)$, then

$$trace(\boldsymbol{W}^T(\boldsymbol{S}_{\mathcal{C}} - \gamma \boldsymbol{S}_{\mathcal{M}})\boldsymbol{W}) = trace(\boldsymbol{\Lambda}) = \sum_i \lambda_i \qquad (6)$$

Eq. 6 implies that Eq. 3 achieves the optimal value when the number of projective vectors $d$ is set as the number of non-negative eigenvalues of Eq.3, and thus the value of $d$ is determined. Note that since both $\boldsymbol{S}_{\mathcal{C}}$ and $\boldsymbol{S}_{\mathcal{M}}$ are positive semi-definite, $\boldsymbol{S}_{\mathcal{C}} - \gamma \boldsymbol{S}_{\mathcal{M}}$ is unlikely to be negative definite.

## Our Methods

Using Constraint Projection, we can build the base classifiers for ensemble as follows. Given a set of labeled data $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$, and assume that we have obtained a set of pairwise must-link constraints $\mathcal{M}$ and a set of pairwise cannot-link constraints $\mathcal{C}$ (we will later discuss how to obtain $\mathcal{M}$ and $\mathcal{C}$). We first use $\mathcal{M}$ and $\mathcal{C}$ to learn a projective matrix $\boldsymbol{W}$ by Constraint Projection, and project original data $\boldsymbol{x}_i$ into a new data representation $\boldsymbol{z}_i = \boldsymbol{W}^T \boldsymbol{x}_i$, through which we transfer the information in pairwise constraints sets $\mathcal{M}$ and $\mathcal{C}$ into the new data set $\{\boldsymbol{z}_i, y_i\}_{i=1}^n$. Then we train base classifiers using the training data set $\{\boldsymbol{z}_i, y_i\}_{i=1}^n$. Thus we have solved the first problem raised in section 1, i.e., how to build base classifiers using pairwise constraints.

It is noteworthy that the quality and diversity of a base classifier in one round is directly determined by the training data set $\{\boldsymbol{z}_i, y_i\}_{i=1}^n$ in that round. Since $z_i$ is determined by $\boldsymbol{W}$, which is further determined by the pairwise constraint sets $\mathcal{M}$ and $\mathcal{C}$ used in that round, we can say that the pairwise constraint control the property of base classifiers. Thus we come to the second question raised in Section 1, i.e., how to resample pairwise constraints to obtain diverse base classifiers?

In this paper, we propose two methods for resampling pairwise constraints, inspired by previous methods for resampling instances, i.e., Bagging and Boosting, respectively. For convenience, we denote COPEN based on those two resampling methods as COPEN.bag and COPEN.boost, respectively.

**COPEN.bag** We begin with COPEN.bag, the simpler version. The detailed procedure of this algorithm is summarized in `Algorithm 1`.

First, we generate randomly the must-link constraint set $\mathcal{M}$ and cannot-link constraint set $\mathcal{C}$ for each turn of ensemble. It can be implemented as follows: We draw randomly with replacement a pair of data $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ from $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$. If $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ have the same label ($y_i = y_j$), we add a must-link constraint $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ into $\mathcal{M}$; on the other hand, if $y_i \neq y_j$, we add a cannot-link constraint $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ into $\mathcal{C}$. We repeat this process until $n_M$ must-link constraints and $n_C$ cannot-link constraints are sampled for $\mathcal{M}$ and $\mathcal{C}$, respectively. Then, we use Constraint Projection with $\mathcal{M}$ and $\mathcal{C}$ to learn the projective matrix $\boldsymbol{W}$, and build base classifiers on the data set $\{\boldsymbol{W}^T \boldsymbol{x}_i, y_i\}_{i=1}^n$.

**COPEN.boost** Unlike in COPEN.bag where a simple random resampling scheme is adopted to generate pairwise constraints sets for base classifiers, in COPEN.boost, we resample pairwise constraints by putting more emphasis on previously misclassified instances as in Boosting. `Algorithm 2` gives the detailed procedure of COPEN.boost.

We first randomly generate an initial pairwise constraints set, on which we obtain a base classifier using the same procedure as in COPEN.bag. Then we classify the training data with that classifier and obtain a set of misclassified data. In next round, we resample pairwise constraints from only those pairwise constraints whose involved instances are both misclassified by previous base classifier. Finally, in case that the number of previously misclassified data is very small, we randomly sample a certain percentage of data from the whole training data to enlarge the resampling pool such that in each round the desired number of pairwise constraints can be correctly generated. Here, we do not use weighting scheme for the ease of implementation, since maintaining weights on about $n^2$ pairwise constraints will be more expensive than maintaining weights on $n$ instances in classical Boosting procedure.

It is noteworthy that the principles of Bagging and Boosting are only used to resample pairwise constraints sets, which are further used by Constraint Projection algorithm to learn a projection. Once the projection is learned, we use it to project original training data into a new representation and then use *all* the data to build base classifiers, which is apparently different from both Bagging and Boosting. This would be helpful in improving base classifiers' accuracy as well as robustness to noises.

## Experiments

We compare our methods with existing ensemble methods on a broad range of data sets. In our experiments, we compare seven ensemble methods, i.e., COPEN.bag, COPEN.boost, bsLDA, Bagging, Boosting, Random Subspace (RS) and Random Forest (RF). Here, bsLDA denotes the method which applies the popular supervised dimensionality reduction approach LDA to bootstrap samples, then train a base classifier from each sample, and finally combine all the base classifiers by majority voting. Among them, bsLDA, Bagging and Boosting work by resampling instances, Random Subspace works by resampling features, and Random Forest works by resampling both instances and features. In contrast, our methods, both COPEN.bag and COPEN.boost, work by resampling pairwise constraints.

### Experimental Setup

In this paper, we carry out experiments on a PC with 2.7GHz CPU and 1GB RAM. We use J48 in WEKA library (Witten & Frank 2005), a reimplementation of C4.5, as the base classifier for all ensemble methods, except for the Random Forest method, which constructs the tree in a different way by randomly choosing a feature at each node. The implementations of Bagging, Boosting (we use the multi-class version AdaBoost.M1 (Freund & Schapire 1996)), Random Subspace, and Random Forest are all from WEKA. The parameters of J48, Bagging, AdaBoost.M1, Random Subspace and Random Forest were kept at their default values in WEKA. In the Random Subspace method, half ($\lceil p/2 \rceil$) of the features were chosen each time, while for Random Forest, the number of features to select from at each node is set to $\lfloor \log_2 p + 1 \rfloor$. For both COPEN.bag and COPEN.boost, the number of pairwise constraints $n_M$ and $n_C$ were both set to $n$, the number of instances. For COPEN.boost, the sampling threshold $r$ is set to 0.2. Finally, the ensemble size $L$ is set to 50 for all compared methods. It is noteworthy that those parameters are rather usual and have not specially tuned to improve the performances.

Twenty data sets from the UCI Machine Learning Repository (Blake, Keogh, & Merz 1998) were used in our experiments. Those data sets have been widely used to evaluate existing ensemble methods. As our methods are defined for numeric features, discrete features were first converted to numeric ones using WEKA's *NorminalToBinary* filter (Witten & Frank 2005). For fair comparison, other methods are also executed on the filtered data. A summary of these data sets is shown in the left of Table 1(or 2), where $inst$, $attr$, and $class$ denote number of instances, attributes, and classes respectively. For each data set, a $5 \times 2$-fold cross-validation (García-Pddrajas, García-Osorio, & Fyfe 2007) was performed.

### Experimental Results

Table 1 presents the error rates of seven ensemble methods and J48. Note that the table shows the mean errors of $5 \times 2$-fold cross-validation, and the standard deviations are not listed due to space limit. From Table 1, we can see that in most cases, COPEN.bag and COPEN.boost are superior to the other five ensemble methods as well as the base classifier. Furthermore, the significance test results, as shown in bottom of Table 1, indicate that our methods are significantly better than other five ensemble methods on a lot of data sets, while are significantly worse only on a few data sets. Finally, Table 1 indicates that COPEN.boost is slightly better than COPEN.bag, but the difference is not significant.

To understand how our methods work, we use *kappa* measure to plot diversity-error diagram following the approach in (Rodríguez, Kuncheva, & Alonso 2006). Due to space limit, we only show diagrams of bsLDA, Bagging and COPEN.bag on *Ionosphere*, as shown in Figure 1. Figure 1 indicates that on this data set, COPEN.bag achieves similar accuracy but much higher diversity than bsLDA, and similar diversity but much higher accuracy than Bagging. Since COPEN.bag's overall diversity and accuracy is the highest, it is not strange that it achieves the best performance among the three methods.

We also test our methods under artificial noise in the class labels to study their robustness to noise. We choose a fraction of instances and change their class labels to other incorrect labels randomly. Table 2 shows the results under 20 percent of noise in class labels. It can be seen from Table 2 that both COPEN.bag and COPEN.boost exhibit much better robustness to noise than all other algorithms, as shown in bottom of the Table 2. Left of Figure 2 shows a representative curve of test error vs. level of noise in labels, which again validates the effectiveness of our methods.

Finally, we compare the seven ensemble methods under a range of ensemble sizes. The results on *Ionosphere* are plotted in right of Figure 2. As can be seen from the figure, as the ensemble size increases, all ensemble methods except bsLDA reduce their test errors. It is surprising to see that on

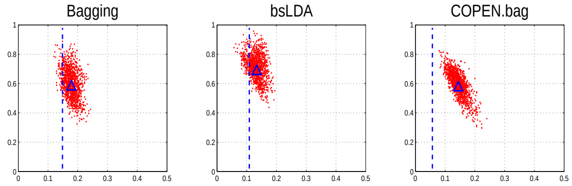

Figure 1: The diversity-error diagrams on *Ionosphere*. In each plot, $x$-axis represents average error of a pair of classifiers, and $y$-axis represents diversity evaluated by the *kappa* measure. The dashed lines show ensemble errors and the up triangles denote centroids of clouds.

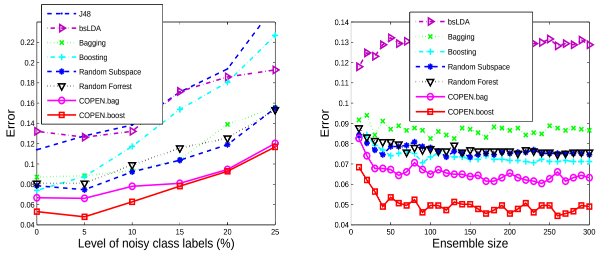

Figure 2: Results on *Ionosphere*. Left: Test error vs. level of noise in class labels; Right: Test error vs. ensemble size. In the legends of both plots, the marks from the top to bottom are single J48 decision tree (only for left plot), bsLDA, Bagging, Boosting, Random Subspace, Random Forest, COPEN.bag and COPEN.boost, respectively.

this data set the error of bsLDA increases. This suggests that sampling data and then applying feature mapping is not as effective as sampling pairwise constraints. A close observation on the figure indicates that for nearly all algorithms, a rapid change on test errors appears at the beginning, and after some values of ensemble size, e.g., 50, test errors change very slowly as ensemble size increases.

## Conclusions

In this paper, we present a new approach for ensemble construction based on pairwise constraints. To the best of our knowledge, this is the first work on using pairwise constraints for classifier ensemble. Our approach uses Constraint Projection to transfer information in pairwise constraints into new data representation and builds base classifiers under the new representation. We propose two methods inspired by Bagging and Boosting to resample pairwise constraints for obtaining diverse base classifiers. Extensive experiments on a broad range of data sets show that our COPEN approach achieves better performance than some state-of-the-art ensemble methods. An important future work is to analyze the proposed methods theoretically. In our experiments we have not finely tuned the parameters of our methods, by using some parameter selection methods such as cross validation, a better performance is expected, which will be studied in the future. We also want to study that whether there is an optimal number of pairwise constraints to be used. Moreover, we will try to apply our methods to other base classifiers.

**Acknowledgments** We thank the the anonymous reviewers for their helpful comments and suggestions.

## References

Bar-Hillel, A.; Hertz, T.; Shental, N.; and Weinshall, D. 2005. Learning a mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research* 6:937–965.

Blake, C.; Keogh, E.; and Merz, C. J. 1998. UCI repository of machine learning databases. [http://www.ics.uci.edu/∼mlearn/MLRepository.html], University of California, Irvine, CA.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

Breiman, L. 2001. Random forests. *Machine Learning* 45(1):5–32.

Davidson, I., and Basu, S. 2007. A survey of clustering with instance level constraints. [http://www.cs.ucdavis.edu/∼davidson/constrained-clustering/CAREER/Survey.pdf], constrained-clustering.org.

Freund, Y., and Schapire, R. 1996. Experiments with a new boosting algorithm. In *ICML*, 148–156.

García-Pddrajas, N.; García-Osorio, C.; and Fyfe, C. 2007. Nonlinear boosting projections for ensemble construction. *Journal of Machine Learning Research* 8:1–33.

Ho, T. 1998. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Analysis and Machine Intelligence* 20(8):832–844.

Kuncheva, L. 2004. *Combining Pattern Classifiers: Methods and Algorithms.* John Wiley and Sons.

Liu, Y.; Jin, R.; and Jain, A. 2007. Boostcluster: Boosting clustering by pairwise constraints. In *KDD*, 450–459.

Opitz, D. 1999. Feature selection for ensembles. In *AAAI*, 379–384.

Rodríguez, J.; Kuncheva, L.; and Alonso, C. 2006. Rotation forest: A new classifier ensemble method. *IEEE Trans. Pattern Analysis and Machine Intelligence* 28(10):1619–1630.

Tang, W.; Xiong, H.; Zhong, S.; and Wu, J. 2007. Enhancing semi-supervised clustering: A feature projection perspective. In *KDD*, 707–716.

Wang, X., and Tang, X. 2006. Random sampling for subspace face recognition. *International Journal Computer Vision* 70(1):91–104.

Witten, I., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd edition*. Morgan Kaufmann.

Zhang, D.; Zhou, Z.-H.; and Chen, S. 2007. Semi-supervised dimensionality reduction. In *SDM*, 629–634.

Zhou, Z.-H., and Yu, Y. 2005. Ensembling local learners through multimodal perturbation. *IEEE Trans. System, Man and Cybernetics-Part B* 35(4):725–735.

Table 1: Error rates of J48 and seven ensemble methods on 20 UCI data sets. Bottom rows of the table present Win-Loss-Tie (W-L-T) comparisons between COPEN (denoted as C.bag and C.boost) against other approaches. *Abs.* and *Sig.* present the comparison on W-L-T before and after pairwise $t$-tests at 95% significance level, respectively.

| Data sets | $(inst/attr/class)$ | C.bag | C.boost | bsLDA | Bagging | Boosting | RS | RF | J48 |
|---|---|---|---|---|---|---|---|---|---|
| balance_scale | (625/4/3) | **0.0829** | 0.1024 | 0.0858 | 0.1504 | 0.1955 | 0.1850 | 0.1648 | 0.2157 |
| breast_cancer | (286/48/2) | **0.2672** | 0.2811 | 0.3141 | 0.2812 | 0.3309 | 0.2812 | 0.2847 | 0.3218 |
| breast_w | (699/9/2) | 0.0303 | **0.0283** | 0.0320 | 0.0432 | 0.0398 | 0.0358 | 0.0306 | 0.0598 |
| credit_g | (1000/61/2) | 0.2700 | 0.2668 | 0.2614 | **0.2496** | 0.2612 | 0.2502 | 0.2674 | 0.2954 |
| ecoli | (336/7/8) | 0.1457 | **0.1407** | 0.1438 | 0.1561 | 0.1529 | 0.1887 | 0.1492 | 0.1842 |
| heart_c | (303/22/2) | 0.1617 | **0.1597** | 0.1749 | 0.1821 | 0.1940 | 0.1842 | 0.1795 | 0.2211 |
| heart_h | (294/22/2) | 0.1844 | 0.1782 | **0.1680** | 0.1905 | 0.2095 | 0.1844 | 0.1789 | 0.2061 |
| heart_statlog | (270/13/2) | 0.1963 | 0.1881 | **0.1822** | 0.2030 | 0.2089 | 0.1963 | 0.1919 | 0.2378 |
| hepatitis | (155/19/2) | 0.1962 | 0.1781 | 0.1820 | 0.1937 | 0.1858 | 0.1898 | **0.1690** | 0.2195 |
| ionosphere | (351/34/2) | 0.0667 | **0.0530** | 0.1322 | 0.0871 | 0.0740 | 0.0786 | 0.0809 | 0.1140 |
| iris | (150/4/3) | 0.0747 | 0.0747 | **0.0440** | 0.0547 | 0.0667 | 0.0640 | 0.0560 | 0.0587 |
| letter | (5000/16/26) | 0.1480 | 0.1396 | 0.1701 | 0.1644 | **0.1137** | 0.1322 | 0.1503 | 0.2635 |
| lung_cancer | (32/56/3) | **0.4518** | 0.4725 | 0.5831 | 0.4878 | 0.5706 | 0.5196 | 0.4922 | 0.5757 |
| primary_tumor | (339/23/18) | **0.6892** | 0.6905 | 0.6914 | 0.7213 | 0.7491 | 0.7214 | 0.7350 | 0.7479 |
| segment | (2310/19/7) | 0.0444 | 0.0419 | 0.0526 | 0.0374 | **0.0226** | 0.0326 | 0.0374 | 0.0484 |
| sonar | (208/60/2) | 0.1866 | **0.1856** | 0.3180 | 0.2432 | 0.2355 | 0.2240 | 0.1990 | 0.3037 |
| spect_heart | (267/22/2) | **0.1850** | 0.1995 | 0.2067 | 0.2353 | 0.2513 | 0.2620 | 0.2299 | 0.2460 |
| vehicle | (846/18/4) | 0.2504 | 0.2499 | **0.2213** | 0.2683 | 0.2490 | 0.2636 | 0.2697 | 0.2981 |
| vowel | (990/27/11) | 0.0982 | 0.0820 | 0.1335 | 0.1418 | 0.0824 | 0.1103 | **0.0610** | 0.2739 |
| waveform | (5000/40/3) | 0.1346 | **0.1332** | 0.1990 | 0.1747 | 0.1626 | 0.1632 | 0.1768 | 0.2608 |
| C.boost vs. others | Abs. W-L-T | 14-5-1 | – | 14-6-0 | 17-3-0 | 15-5-0 | 16-4-0 | 16-4-0 | 19-1-0 |
| | Sig. W-L-T | 1-1-18 | – | 7-2-11 | 10-1-9 | 9-2-9 | 7-3-10 | 7-0-13 | 18-0-2 |
| C.bag vs. others | Abs. W-L-T | – | 5-14-1 | 13-7-0 | 16-4-0 | 13-7-0 | 13-5-2 | 13-7-0 | 19-1-0 |
| | Sig. W-L-T | – | 1-1-18 | 6-1-13 | 8-2-10 | 8-2-10 | 6-3-11 | 4-2-14 | 16-0-4 |

Table 2: Error rates of J48 and seven ensemble methods with 20% noise in class labels on 20 UCI data sets. Bottom rows of the table present Win-Loss-Tie (W-L-T) comparisons between COPEN (denoted as C.bag and C.boost) against other approaches. *Abs.* and *Sig.* present the comparison on W-L-T before and after pairwise $t$-tests at 95% significance level, respectively.

| Data sets | $(inst/attr/class)$ | C.bag | C.boost | bsLDA | Bagging | Boosting | RS | RF | J48 |
|---|---|---|---|---|---|---|---|---|---|
| balance_scale | (625/4/3) | **0.1203** | 0.1277 | 0.1213 | 0.1926 | 0.2947 | 0.2102 | 0.2202 | 0.2538 |
| breast_cancer | (286/48/2) | 0.3049 | **0.2993** | 0.3531 | 0.3462 | 0.3993 | 0.3134 | 0.3596 | 0.3615 |
| breast_w | (699/9/2) | 0.0366 | **0.0335** | 0.0401 | 0.0567 | 0.1353 | 0.0541 | 0.0916 | 0.0907 |
| credit_g | (1000/61/2) | **0.2902** | 0.2962 | 0.2968 | 0.2922 | 0.3330 | 0.2930 | 0.2948 | 0.3604 |
| ecoli | (336/7/8) | 0.3678 | 0.3610 | 0.3313 | **0.2576** | 0.3457 | 0.3796 | 0.3745 | 0.4631 |
| heart_c | (303/22/2) | **0.1815** | 0.1894 | 0.2686 | 0.2475 | 0.2667 | 0.2237 | 0.2296 | 0.2931 |
| heart_h | (294/22/2) | 0.1782 | **0.1776** | 0.2116 | 0.2279 | 0.2939 | 0.1993 | 0.2381 | 0.2442 |
| heart_statlog | (270/13/2) | **0.2178** | 0.2326 | 0.2333 | 0.2459 | 0.2919 | 0.2185 | 0.2437 | 0.3000 |
| hepatitis | (155/19/2) | **0.1961** | 0.2077 | 0.2376 | 0.2361 | 0.2979 | 0.2064 | 0.2271 | 0.3044 |
| ionosphere | (351/34/2) | 0.0946 | **0.0837** | 0.1829 | 0.1350 | 0.1778 | 0.1185 | 0.1237 | 0.2068 |
| iris | (150/4/3) | 0.1053 | 0.0893 | 0.1200 | 0.0840 | 0.1933 | **0.0760** | 0.1080 | 0.1067 |
| letter | (5000/16/26) | 0.3471 | 0.3446 | **0.3386** | 0.3550 | 0.3744 | 0.3420 | 0.4245 | 0.5114 |
| lung_cancer | (32/56/3) | **0.4592** | 0.5522 | 0.5463 | 0.5357 | 0.5078 | 0.5529 | 0.5698 | 0.5690 |
| primary_tumor | (339/23/18) | 0.6980 | **0.6934** | 0.6991 | 0.7223 | 0.7337 | 0.7237 | 0.7334 | 0.7561 |
| segment | (2310/19/7) | 0.0589 | 0.0571 | 0.0692 | **0.0531** | 0.0960 | 0.0560 | 0.0762 | 0.1169 |
| sonar | (208/60/2) | 0.2902 | 0.3076 | 0.4266 | 0.2885 | 0.2921 | 0.2875 | **0.2874** | 0.3664 |
| spect_heart | (267/22/2) | 0.2023 | **0.1925** | 0.2660 | 0.2367 | 0.3042 | 0.2210 | 0.2510 | 0.2704 |
| vehicle | (846/18/4) | 0.2634 | 0.2707 | **0.2409** | 0.2806 | 0.3040 | 0.2913 | 0.3059 | 0.3818 |
| vowel | (990/27/11) | 0.1574 | **0.1404** | 0.1885 | 0.2053 | 0.2311 | 0.1952 | 0.2214 | 0.3766 |
| waveform | (5000/40/3) | 0.1425 | **0.1404** | 0.2636 | 0.1846 | 0.1908 | 0.1749 | 0.2001 | 0.3823 |
| C.boost vs. others | Abs. W-L-T | 12-8-0 | – | 16-4-0 | 14-6-0 | 17-3-0 | 13-7-0 | 18-2-0 | 20-0-0 |
| | Sig. W-L-T | 0-0-20 | – | 8-1-11 | 9-1-10 | 17-0-3 | 8-0-12 | 13-0-7 | 18-0-2 |
| C.bag vs.others | Abs. W-L-T | – | 8-12-0 | 17-3-0 | 16-4-0 | 19-1-0 | 16-4-0 | 19-1-0 | 20-0-0 |
| | Sig. W-L-T | – | 0-0-20 | 7-1-12 | 10-1-9 | 17-0-3 | 7-0-13 | 14-0-6 | 19-0-1 |