DSC 2003 Working Papers
(Draft Versions)
http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

An Ensemble Method for Clustering

Andreas Weingessel, Evgenia Dimitriadou and Kurt Hornik
Institut für Statistik und Wahrscheinlichkeitstheorie, TU Wien, Austria

Abstract

Combination strategies in classification are a popular way of overcoming instabilities in classification algorithms. A direct application of ideas such as "voting" to cluster analysis problems is not possible, as no a priori class information for the patterns is available. We present a methodology for combining ensembles of partitions obtained by clustering, discuss the properties of such combination strategies and relate them to the task of assessing partition "aggrement".

Keywords: ensemble methods, unsupervised learning, voting

1 Introduction

In the area of classification, combination strategies are a popular way of improving the recognition rate (Dietterich, 2000; Lam, 2000; Parhami, 1994; Bauer & Kohavi, 1999; Kuncheva, 2000). The combination can be implemented using a variety of strategies, such as combination of multiple classifiers (Woods et al., 1997; Lam & Suen, 1995; Schapire et al., 1998), classifier fusion (Cho & Wang, 1995; Keller et al., 1994), committees of neural networks (Bishop, 1995; Breiman, 1994) and several combinations of these schemes (Xu et al., 1992; Lee & Srihari, 1993; Lorczak et al., 1989).

A direct application of such ideas to cluster analysis problems is not possible, as no a priori class information for the patterns is available. Thus, it is not immediately clear which cluster from one particular clustering result corresponds to which in an other result. An obvious idea to identify the data points which have been repeatedly assigned to the same cluster is the construction of a pairwise concordance matrix (Fred, 2001). This matrix can then be used as a distance matrix for a hierarchical clustering algorithm. The main problem of such an approach is the size of such

a matrix which scales quadratically with the size of the data set, thus becoming infeasible for large size problems in data mining.

In this paper we present a strategy where different crisp or fuzzy partitions of a data set can be combined to a new fuzzy partition which optimally represents these partitions. It is not important whether these different partitions are the results of the application of different clustering algorithms or the results of the repeated application of one clustering algorithm with different random initializations. Our algorithm combines partitions on a sequential basis, thus overcoming the computationally infeasible simultaneous combination of all partitions. It scales linearly in the number of data points and the number of repetitions, making it feasible to be applied to large data sets. The algorithm improves also the ability of a clustering algorithm to find structures in a data set as it can find any cluster shapes in the data set and is not confined to certain cluster structures.

This paper is based on Weingessel et al. (1999) where the basic ideas are described and Dimitriadou et al. (2002b) where theoretical considerations can be found.

The paper is organized as follows. In Section 2 we formalize our ideas and present our combination scheme. Section 3 demonstrates the implementation in R and some comments on them. The conclusion is given in Section 4.

2 Voting Scheme

2.1 A distance measure between partitions with known labels

We are looking for a partition P of a given data set $\{x_1, \ldots, x_N\}$ into k classes which optimally represents a given set of M partitions of the same set. Each of these M partitions is represented by an $N \times k$ membership matrix $U^{(m)}, m = 1, \ldots, M$. The element $u_{ij}^{(m)}$ of $U^{(m)}$ is the degree of membership of x_i to the jth class of the mth partition. We denote the ith row of $U^{(m)}$ as $u_i^{(m)}$, that is $u_i^{(m)}$ is the membership vector (or membership function) of the data point x_i for the partition $U^{(m)}$. The final partition P is encoded as an $N \times k$ matrix with elements p_{ij} and rows p_i.

For the moment, we assume that the cluster labels are known, i.e., we are in a case corresponding to classification. As dissimilarity function $h(U^{(m)}, P)$ between $U^{(m)}$ and P we use the average square distance between the membership functions $u_i^{(m)}$ and p_i for all x_i (Dubois & Prade, 1980; Kacprzyk, 1976). That is,

$$h(U^{(m)}, P) := \frac{1}{N} \sum_{i=1}^{N} ||u_i^{(m)} - p_i||^2 \tag{1}$$

If $U^{(m)}$ and P are both crisp partitions, that is, one element of $u_i^{(m)}$ and p_i respectively equals 1, all others are 0, then Equation (1) measures the relative number of data points x_i which belong to different classes in $U^{(m)}$ and P. That is, in the sense of classification, Equation (1) measures exactly the misclassification

rate, if we consider P as the correct classification and $U^{(m)}$ as the result of one classifier.

We want to find a partition P which optimally represents a set of given partitions $U^{(m)}$. We chose as an optimality criterion the average dissimilarity function $h(U^{(m)}, P)$, which should be minimal. Thus, the task is to find P in such a way that

$$\min_P l(U^{(1)}, \ldots, U^{(M)}; P) := \min_P \left(\frac{1}{M} \sum_{m=1}^{M} h(U^{(m)}, P) \right) \tag{2}$$

is minimized over all P.

2.2 Derivation of the Algorithm

The above considerations are valid for the classification case where the partitions given by $U^{(m)}$ are fixed in the sense that for all m the first column of $U^{(m)}$ corresponds to the first class, the second column to the second class and so on.

Things are different in clustering where it is not clear which class number (label) is assigned to which class in each run. That is, any relabeling of the classes, which can be written as a column permutation $\Pi_m(U^{(m)})$, is to be considered the same partition. Thus, partitions $U^{(m)}$ and $\Pi_m(U^{(m)})$ which only differ by a permutation of the class labels, are to be considered the same and thus we demand that the distances stay the same

$$h(U^{(m)}, P) = h(\Pi_m(U^{(m)}), P), \quad \forall P \tag{3}$$

where $\Pi_m(U^{(m)})$ is any permutation of the columns of $U^{(m)}$. To ensure the validity of (3) we have to extend the dissimilarity function (1). As we are interested in finding similar partitions it is natural to use the following definition.

Definition The dissimilarity function $h(U^{(m)}, P)$ between two clustering partitions $U^{(m)}$ and P is

$$h(U^{(m)}, P) := \min_{\Pi_m} \left(\frac{1}{N} \sum_{i=1}^{N} ||\Pi_m(u_i^{(m)}) - p_i||^2 \right) \tag{4}$$

where the minimum is taken over all possible column permutations Π_m.

Inserting into the loss functional (2), we finally get

$$l(U^{(1)}, \ldots, U^{(M)}; P) = \min_{\Pi_1, \ldots, \Pi_M} \left(\frac{1}{M} \sum_{m=1}^{M} h(U^{(m)}, P) \right) \tag{5}$$

The task of finding an optimal partition P is given by the minimization problem

$$\min_P l(U^{(1)}, \ldots, U^{(M)}; P) = \min_{p_1, \ldots, p_N} \min_{\Pi_1, \ldots, \Pi_M} \left(\frac{1}{M} \sum_{m=1}^{M} \frac{1}{N} \sum_{i=1}^{N} ||\Pi_m(u_i^{(m)}) - p_i||^2 \right) \tag{6}$$

which we are aiming to solve with our voting algorithm. That is, we are looking for an optimal relabeling of the clusters (minimization over Π_m) to identify corresponding clusters in the different partitions and for an optimal combination of these relabeled clusters (minimization over p_n).

Note that, for finding an optimal P, we have to minimize p_i and Π_m simultaneously, because the choice of the permutations Π_m depends on the values of p_i. Therefore, a direct solution of the minimization problem (6) is infeasible, since it requires an enumeration of all possible column permutations.[1]

For establishing an approximation algorithm we first transform the minimization problem into another optimization problem.

Theorem 2.1 *The solution of (5) is equivalent to the solution of*

$$\max_{\Pi_1,\ldots,\Pi_M} \frac{1}{N}\sum_{i=1}^{N} ||p_i||^2 \tag{7}$$

under the constraint

$$p_i = \frac{1}{M}\sum_{m=1}^{M} \Pi_m(u_i^{(m)})$$

2.3 Description of the Voting Algorithm

We will now describe the voting algorithm which is based upon the mathematical considerations in the previous section.

Suppose we have a set of M partitions $U^{(m)}$ of the same data set. In the following $P^{(m)}$ will denote the voting result after the first m steps, $P = P^{(M)}$ the final result.

Algorithm Voting()

1. Set $P^{(1)} := U^{(1)}$ and $\hat{\Pi}_1 = \mathrm{id}$.
2. For $m = 2$ to M
   (a) Compute the solution $\hat{\Pi}_m$ of

$$\max_{\Pi_m} \mathrm{tr}\left((\sum_{l=1}^{m-1} \hat{\Pi}_l(U^{(l)}))'\Pi_m(U^{(m)})\right) \hat{=} \max_{\Pi_m} \mathrm{tr}\left((P^{(m-1)})'\Pi_m(U^{(m)})\right)$$

   That means permute the labels of the mth partition $U^{(m)}$ such that $\mathrm{tr}\left((P^{(m-1)})'\Pi_m(U^{(m)})\right)$ is maximized. That is maximize the diagonal of the confusion matrix of $P^{(m-1)}$ and $U^{(m)}$.

   (b) Compute the voting result $P^{(m)}$ after m runs as

$$P^{(m)} = \frac{1}{m}\sum_{l=1}^{m} \hat{\Pi}_l(U^{(l)}) = \frac{m-1}{m}P^{(m-1)} + \frac{1}{m}\hat{\Pi}_m(U^{(m)})$$

In order to compute step (2a) we must maximize over all $k!$ permutations of the cluster labels. Whereas this is fast for a small number of clusters k, this might need its time for higher values of k. For such cases we use the following approximation algorithm to relabel a partition U to fit to a partition P.

[1] The solution of our minimization problem is not unique, since the numbers of the classes (labels) in the final result are arbitrary. We can set for example $\Pi_1 \equiv id$, thus avoiding a global maximum which is trivially non-unique.

1. Build up the confusion matrix between P and U.
2. Find the maximum element in this confusion matrix.
3. Associate the two clusters corresponding to the maximum element.
4. Remove these two clusters.
5. With the reduced confusion matrix goto 2.

Another possibility to solve the above problem in cubical time is the usage of the Hungarian algorithm (Papadimitriou & Steiglitz, 1982).

Theorem 2.2 *The Voting algorithm yields the stepwise optimal solution to the reduced version of (6)*

$$\min_{p_1,\ldots,p_N} \min_{\hat{\Pi}_m} \left(\frac{1}{m} \sum_{l=1}^{m} \frac{1}{N} \sum_{i=1}^{N} ||\hat{\Pi}_l(u_i^{(l)}) - p_i||^2 \right) \tag{8}$$

After voting of M runs we get for every data point x_i and every cluster k a value p_{ik} which gives the fraction of times this data point has been assigned to this cluster. For interpreting the final result we can either accept this fuzzy decision or assign every data point x_i to that cluster $k = \text{argmax}_j(p_{ij})$ where it has been assigned most often.

The voting result gives also structural information of the data set. We define the *sureness* of a data point as the percentage of times it has been assigned to its "winning" cluster k, that is $sureness(x) = \max_j(p_{ij})$. The sureness of a data point shows how strong a certain point belongs to its cluster. We can also see how clearly a cluster is defined by computing the average sureness of a cluster (*Avesure*) as the average sureness of all the points of a cluster that belong to it.

3 Implementation in R

All our experiments have been performed in R (Ihaka & Gentleman, 1996), a system for statistical computation and graphics, which implements the well-known S-language for statistics. R runs under a variety of Unix platforms (such as Linux) and under Windows95/98/ME/2000/NT. It is available freely via CRAN, the Comprehensive R Archive Network, whose master site is at `http://www.R-project.org`.

As the voting algorithm can be applied to the result of any cluster procedure, the implementation should allow a wide variety of cluster algorithms to use. As the results of cluster algorithms are random and we want to make the results repeatable, the algorithm should also allow the use ready cluster results.

The crisp and fuzzy clustering algorithms used for our experiments can be found in the `cclust` (Dimitriadou, 2002), `e1071` and (Dimitriadou et al., 2002a) packages. *k*means (Linde et al., 1980; Fritzke, 1997), hard competitive learning Fritzke (1997), and unsupervised fuzzy competitive learning clustering runs are used for the ensemble partition.

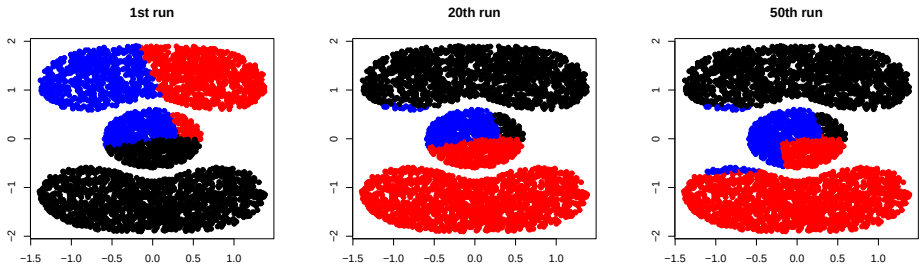

Figure 1: Stepwise $k$means ensembles

Cassini 2-Dimensional Data Set This data set is 2-dimensional with data points uniformly distributed within the structures seen in Figure 1. This data set is generated in R and can be found in a collection package of benchmark data sets (see `mlbench` (Leisch & Dimitriadou, 2001)). The middle structure consists of 900 data points whereas both the external ones consist of 1500 points. Typical results of k-means clustering and the result which yielded the minimum error in all runs show that k-means (*kmeans*), hard competitive learning (*hardcl*) and unsupervised fuzzy competitive learning (*ufcl*) divides one of the two big structures into two clusters, thus ignoring the small cluster in the middle.

To illustrate the functionality of the voting algorithm we present two figures. In Figure 1 we show a typical result of a k-means run (left), the result of combining 20 cluster results (middle) and the result of combining 50 runs (right). We can see that the results gradually improve as more cluster results get mixed.

In Figure 2 we compare the results of 50 voting runs between different base clustering algorithms, namely k-means, hard competitive learning and unsupervised fuzzy competitive learning. The final figure gives the combination of all 150 cluster runs combined. We see that the k-means results does resemble the three parts worst. This can be explained by the fact that k-means usually only converges into a few number of different results thus giving not much diversity for the combination scheme. Hard competitive learning optimizes the same error function as k-means but, as an online algorithm, does not converge in exactly the same results. Finally, the fuzzy clustering algorithm also incooperates membership functions also in the single cluster runs.

4 Conclusion

In this paper we present a method to combine the results of several independent cluster runs by voting between their results. Our sequential scheme helps us to overcome the computationally infeasible simultaneous combination of all partitions. It allows us to deal with the problem of local minima of cluster algorithms and to find a partition of the data which is supported by repeated applications of the cluster algorithm and not influenced by the randomness of initialization or the

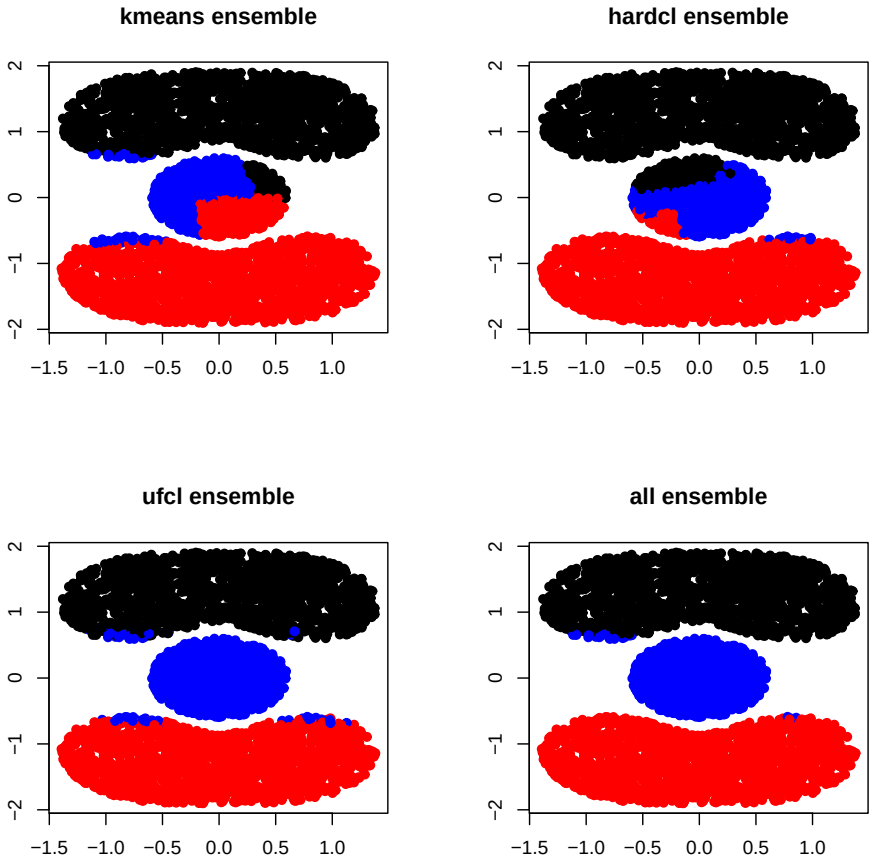

Figure 2: Ensemble Results

cluster process itself.

`voting` *Voting in Clustering*

Description

An ensemble clustering method that combines several clustering results to a final partition.

Usage

```
voting (x, centers, rep=100, baseClAlg = cmeans, ClMethod = "cmeans",
        crisp = TRUE, matchmethod= "exact", crosstab ="sum", verbose= FALSE,
        ReadyResults = NULL, weight = rep(1/rep, rep), mix = TRUE, ...)
```

Arguments

`x`	The data matrix where columns correspond to variables and rows to observations
`centers`	Number of clusters or initial values for cluster centers
`rep`	Number of clustering results to vote on
`baseClAlg`	The algorithm used for the single clustering results
`ClMethod`	The method of the algorithm used
`crisp`	If the voting takes place on the crisp membership or the fuzzy membership of the data points in the clusters
`matchmethod`	the permutation method used for finding the associated clusters
`crosstab`	the criterion used for the clusters association in the diffusion matrix
`verbose`	If `TRUE`, print the number of the clustering run
`ReadyResults`	Initialize with ready clustering results
`weight`	The weight of every clustering result on the final voted partition
`mix`	If `TRUE` the order of the ready results is mixed
`...`	Parameters for the clustering algorithms can be given as arguments

Details

The data given by `x` is clustered by the ensemble clustering method.

If `centers` is a matrix, its rows are taken as the initial cluster centers. If `centers` is an integer, `centers` rows of `x` are randomly chosen as initial values.

The number of the clustering results to be combined by voting is given by `rep`.

The name of the function used for the clustering results. If `"cclust"` then the crisp clustering methods are used. Note that `cclust` package is required. If `"cmeans"` then the fuzzy clustering methods from package `e1071` are used.

The clustering method is defined. For example if `baseClAlg=cmeans` and `ClMethod="ufcl"` then the on-line unsupervised fuzzy competitive learning method is used (from the function `"cmeans"`).

If `crisp=TRUE` then the crisp memberships of the data points in the clusters are considered for voting; otherwise the fuzzy (for the fuzzy algorithms).

If `mathmethod="exact"` all the cluster permutation are considered for the association of the clusters to each other. For more details see aliasmathClasses.

If `crosstab="sum"` then the diagonal of the confusion table is maximized. If `crosstab="rowmean"` (`crosstab="colmean"`) then the rows (colums) of the confusion table are normalized before the diagonal maximization.

If `verbose` is `TRUE`, it displays for each iteration the number the value of the objective function.

If `ReadyResults` then a list of the clustering results of the `"cclust"` or `"fclust"` class is given.

The weight of each clustering result on the final voting partition is given by `weight`. By default all the clustering results are equally weighted.

If `mix` is `TRUE`, then the order of the clustering results given in `ReadyResults` is randomly mixed.

Value

`membership`	a matrix with the membership values of the data points to the clusters.
`centers`	The centers of the ensemble, final, voting partition.
`ClCenters`	The permuted centers of the clustering results.
`size`	The number of data points in each cluster.
`cluster`	Vector containing the indices of the clusters where the data points are assigned to. The maximum membership value of a point is considered for partitioning it to a cluster.
`baseClResults=baseClResults`	A list with the single clustering results.

Author(s)

Evgenia Dimitriadou and Andreas Weingessel

References

Evgenia Dimitriadou, Andreas Weingessel and Kurt Hornik. "A Combination Scheme for Fuzzy Clustering" in *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 16, no. 7, pages: 901-912, 2002.

Examples

```
# a 2-dimensional example
x <- rbind(matrix(rnorm(100,sd=0.3),ncol=2),
         matrix(rnorm(100,mean=1,sd=0.3),ncol=2))
vot <- voting(x,2,50)
```

References

Bauer, E. & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, **36**, 105–139.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford.

Breiman, L. (1994). *Bagging predictors*. Tech. Rep. Technical Report 421, Department of Statistics, University of California at Berkeley.

Cho, K. B. & Wang, B. H. (1995). Radial basis function based adaptive systems. In *Proc. FUZZ/IEEE*, pp. 247–252, Yokohama, Japan.

Dietterich, T. (2000). Ensemble methods in machine learning. In Kittler & Roli (eds.), *Multiple Classifier Systems*, vol. 1857 of *Lecture Notes in Pattern Recognition*, pp. 1–15. Springer.

Dimitriadou, E. (2002). `cclust` – convex clustering methods and clustering indexes. R package, Version 0.6-9. Available from http://cran.R-project.org.

Dimitriadou, E., Hornik, K., Leisch, F., Meyer, D., & Weingessel, A. (2002a). `e1071` – misc functions of the Department of Statistics (e1071), TU Wien. R package, Version 1.3-5. Available from http://cran.R-project.org.

Dimitriadou, E., Weingessel, A., & Hornik, K. (2002b). A combination scheme for fuzzy clustering. *International Journal of Pattern Recognition and Artificial Intelligence*, **16**(7), 901–912.

Dubois, D. & Prade, H. (1980). *Fuzzy Sets and Systems: Theory and Applications*, chap. Fuzzy Sets, p. 24. Academic Press, Harcourt Brace Jovanovich.

Fred, A. (2001). Finding consistent clusters in data partitions. In Kittler & Roli (eds.), *Multiple Classifier Systems*, vol. 2096 of *Lecture Notes in Computer Science*, pp. 309–318. Springer.

Fritzke, B. (1997). Some competitive learning methods. http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/research/gsn/.

Ihaka, R. & Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, **5**(3), 299–314.

Kacprzyk, J. (1976). Fuzzy-set-theoretic approach to the optimal assignment of work places. In Guardabassi, G. & Locatelli, A. (eds.), *Large Scale Systems: Theory and Applications, Proceedings of the IFAC Symposium*, pp. 123–131, Udine, Italy.

Keller, J. M., Gader, P., Tahani, H., Chiang, J. H., & Mohamed, M. (1994). Advances in fuzzy integration for pattern recognition. *Fuzzy Sets and Systems*, **65**, 273–283.

Kuncheva, L. I. (2000). Clustering-and-selection model for classifier combination. In *Proc. of the 4th International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies (KES'2000.*

Lam, L. (2000). Classifier combinations: Implementations and theoretical issues. In Kittler & Roli (eds.), *Multiple Classifier Systems*, vol. 1857 of *Lecture Notes in Pattern Recognition*, pp. 78–86. Springer.

Lam, L. & Suen, C. Y. (1995). Optimal combination of pattern classifiers. *IEEE Trans. on Systems, Man, and Cybernetics*, **16**, 945–954.

Lee, D. S. & Srihari, S. N. (1993). Handprinted digit recognition: a comparison of algorithms. In *Proc. 3rd Int. Workshop Frontiers Handwriting Recognition*, pp. 153–162, Buffalo, NY.

Leisch, F. & Dimitriadou, E. (2001). `mlbench`—a collection for artificial and real-world machine learning benchmarking problems. R package, Version 0.5-6. Available from `http://cran.R-project.org`.

Linde, Y., Buzo, A., & Gray, R. M. (1980). An algorithm for vector quantizer design. *IEEE Transactions on Communications*, **COM-28**(1), 84–95.

Lorczak, P. R., Caglayan, A. K., & Eckhardt, D. E. (1989). A theoretical investigation of generalized voters for redundant systems. In *Proc. Int. Symp. Fault Tolerant Computing*, pp. 444–451, Chicago.

Papadimitriou, C. H. & Steiglitz, K. (eds.) (1982). *Combinatorial Optimiztion, Algorithms and Complexity*, chap. Weighted Matching, p. 248. Prentice-Hall.

Parhami, B. (1994). Voting algorithms. *IEEE Transactions on Reliability*, **43**(4), 617–629.

Schapire, R. E., Freund, Y., Bartlett, P., & Lee, W. S. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, **26**(5), 1651–1686.

Weingessel, A., Dimitriadou, E., & Hornik, K. (1999). A voting scheme for cluster algorithms. In Krell, G., Michaelis, B., Nauck, D., & Kruse, R. (eds.), *Neural Networks in Applications, Proceedings of the Fourth International Workshop NN'99*, pp. 31–37, Otto-von-Guericke University of Magdeburg, Germany.

Woods, K., Kegelmeyer, W. P., & Bowyer, K. (1997). Combination of multiple classifiers using local accuracy estimates. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, **19**, 405–410.

Xu, L., Krzyzak, A., & Suen, C. Y. (1992). Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Syst., Man, and Cybern.*, **22**, 418–435.