Ensembles

Computer Science 760

Patricia J Riddle

Ensembles of Classifiers

An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way (typically weighted or unweighted voting) to classify new examples

Ensembles are often much more accurate than the individual classifiers that make them up

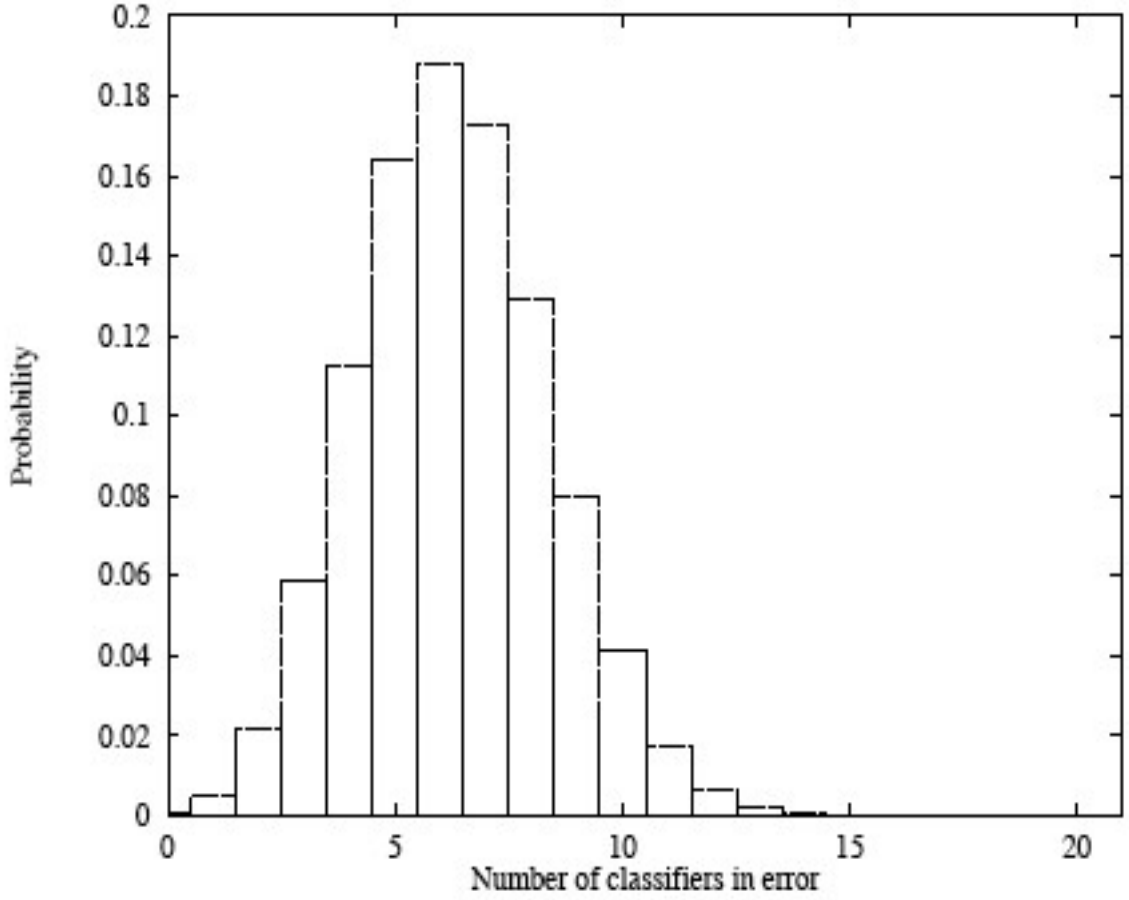
Key to Ensembles

An ensemble can only be more accurate than its component classifiers if the individual classifiers disagree with one another

If individual hypotheses make uncorrelated errors at rates exceeding 0.5, then the error rate of the voted ensemble increases.

Key: individual classifiers with error rates below 0.5 whose errors are at least somewhat uncorrelated

Probability that Majority Vote is Wrong



Probability that exactly l (of 21) hypotheses will make an error, assuming each hypothesis has an error rate of 0.3 and makes its errors independently of the other hypotheses.

Constructing Ensembles

Subsampling works especially well for unstable learning algorithms

- Bagging - bootstrap replicate - 63.2 percent
- Cross-validated committees
- Adaboost - adjusts probability distribution over training instances

Why I hate Adaboost

Manipulating the Input Features - feature selection

Volcanoes on Venus - 8 subsets of 119 input features and 4 network sizes

Failure on sonar data - only works when input features are highly redundant

Manipulating the Output Target

Error Correcting Output Coding

randomly partition K classes into two subsets A and B, learn a classifier, repeat the process L times

Each member of each class receives a vote and the class with the most votes is the prediction of the ensemble

Methods for designing good error-correcting codes can be applied

Has been combined with Adaboost

ECOC has also been combined with feature-selection

Injecting Randomness

Different initial weights in ANN - didn’t perform as well as bagging and cross-validated committees

Decision tree split criteria which chooses randomly among the best 20 tests at each node

- Others used weighted random choice

In ANN bootstrap sampling of training data and adding Gaussian noise to the input features

Markov chain Monte Carlo method - injecting randomness - with vote proportional to posterior probability

Algorithm Specific Methods

Backpropagation - train several networks simultaneously and use a correlation penalty in the error function

Genetic operators to generate new network topologies - multiplicative term that incorporates the diversity of the classifiers - prune to N best networks

Algorithm Specific Methods II

Training on auxiliary task as well as the main task

- diverse classifiers can be learned with one primary task but with different auxiliary tasks such as predicting one of its input features

Network whose secondary prediction is best is the winner - encourage different networks to become experts at predicting the auxiliary task in different local regions

- causes the errors in the primary output to become decorrelated

Decision Trees - option trees - equivalent and more understandable than bagging

Combining Different Algorithms

Some perform much worse than others

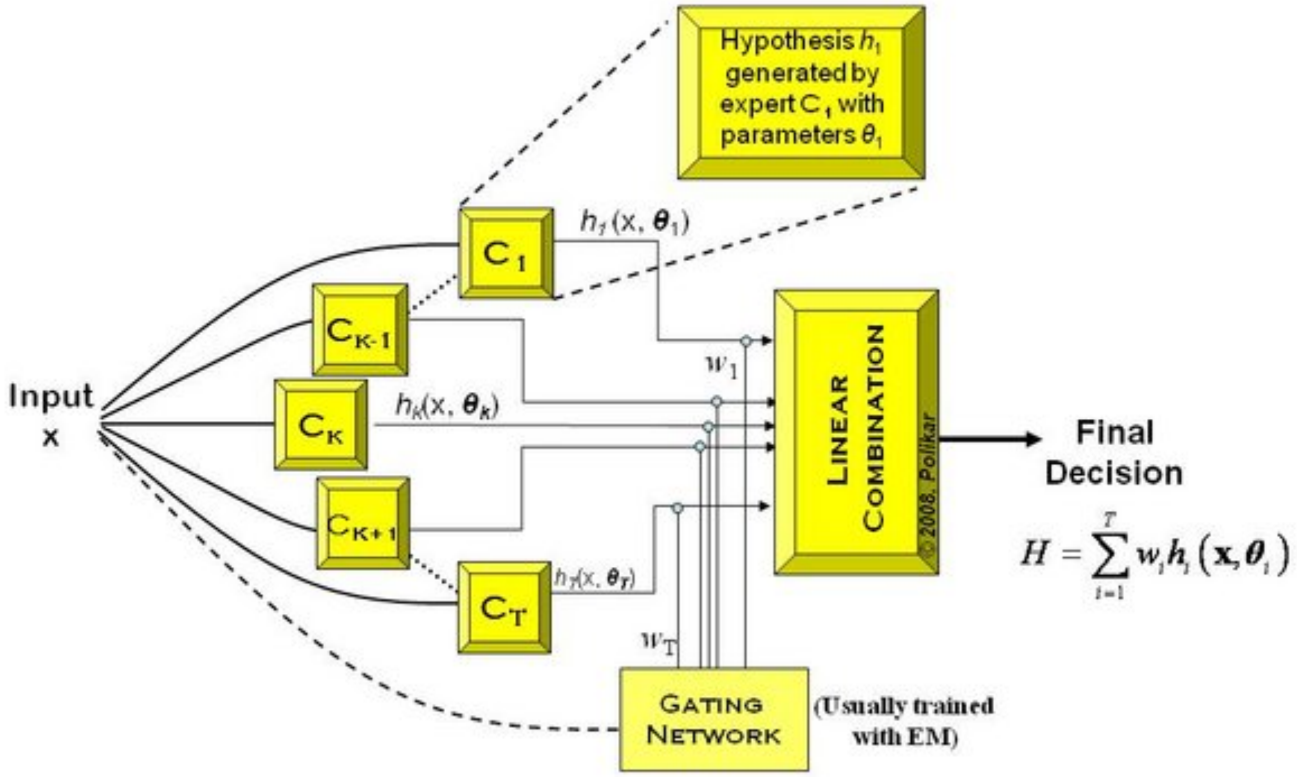
No guarantee of diversity

Weighted combination

Combining Classifiers

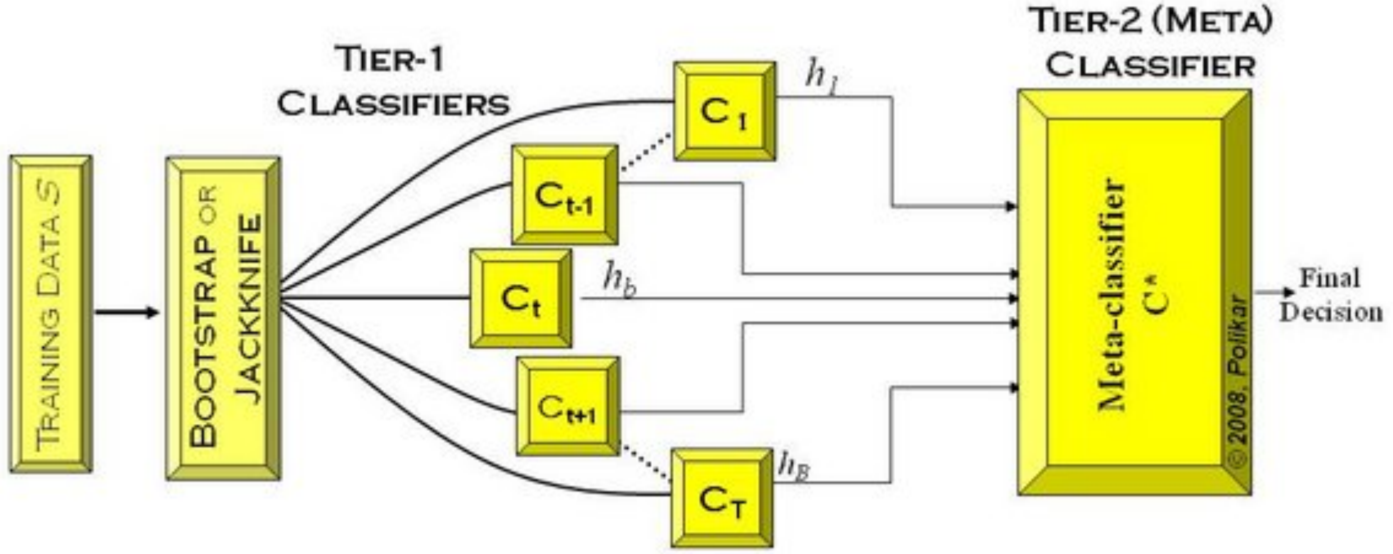
1. Unweighted vote
   - bagging, ECOC - shown to be robust
   - probability estimate if classifier can produce class probability estimates
2. Many weighted voting methods:
   - Regression - weight should be inversely proportional to the variance of the estimates of h
   - Classification - weights proportional to accuracies
3. Learn good weights - gating function or gating network - overfitting problem
4. Stacking - use outputs of L classifiers as attributes for target in leave one out - good results in combining different forms of linear regression

Gating Network



Hypothesis h_1 generated by expert C_1 with parameters θ_1
C_1
$h_1(x, \boldsymbol{\theta}_1)$
C_{K-1}
Input x
C_K
$h_k(x, \boldsymbol{\theta}_k)$
C_{K+1}
C_T
$h_T(x, \boldsymbol{\theta}_T)$
w_1
w_T
Linear Combination
© 2008, Polikar
Final Decision
$H = \sum_{t=1}^{T} w_t h_t(\mathbf{x}, \boldsymbol{\theta}_t)$
Gating Network
(Usually trained with EM)

Stacking


Training Data S
Bootstrap or Jackknife
Tier-1 Classifiers
C1
Ct-1
Ct
Ct+1
CT
h1
hb
hB
Tier-2 (Meta) Classifier
Meta-classifier C*
© 2008, Polikar
Final Decision

Why Ensembles Work

Why should it be possible to find ensembles of classifiers that make uncorrelated errors?

Why shouldn’t we be able to find a single classifier that performs as well as an ensemble?

1. Statistical - Training data might not be sufficient - in 2 class problem need O(log(H)) examples minimum - many equally good hypothesis on the amount of data we have seen

Why Ensembles Work Visual

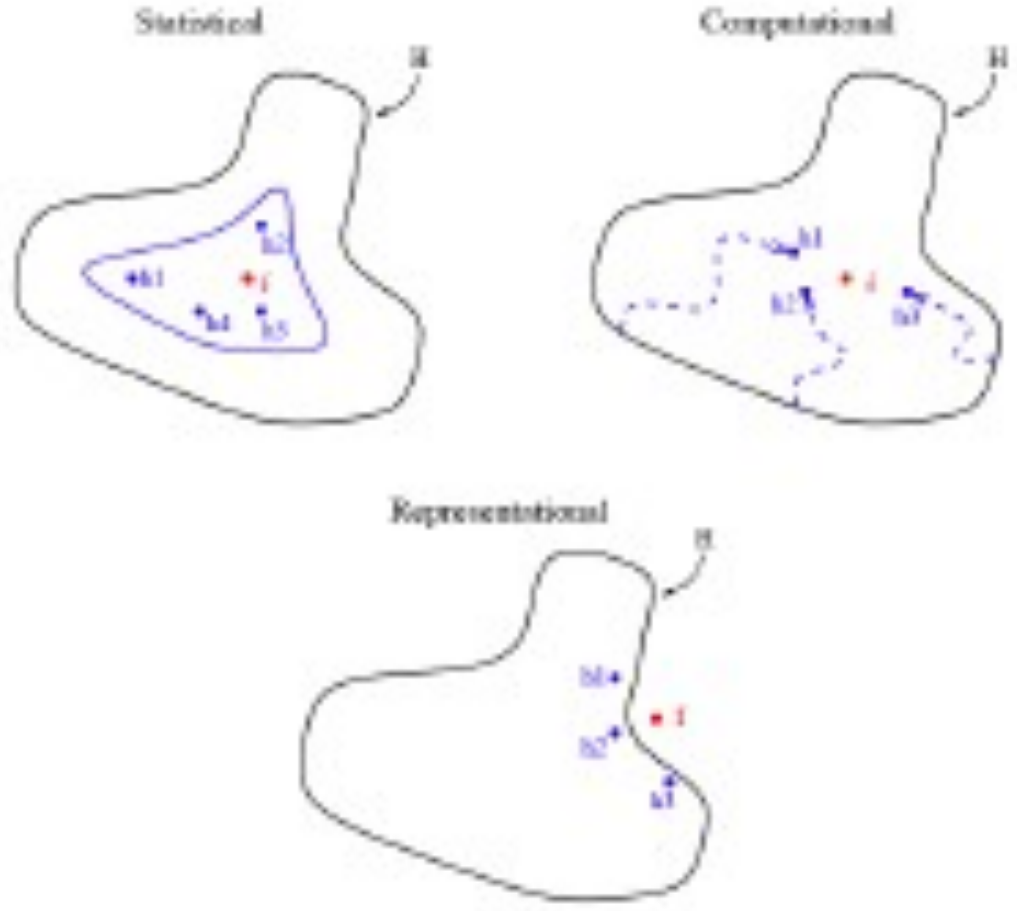


Fig. 2. Three fundamental reasons why an ensemble may work better than a single classifier

OR….

2. Computational - Difficult search problems - smallest decision tree consistent with the data, finding the weights for the smallest possible Neural Network consistent with the training data NP-hard
   – Use search heuristics - so even if there is a unique best hypothesis we might not find it - so find suboptimal approximations
   – So ensembles combine different suboptimal approximations

OR…

3. Representational - Hypothesis space may not combine the true function - weighted combinations of approximations might be able to represent classifiers outside of H
   - Just complex decision trees but way too large for the available data

Decision Boundary

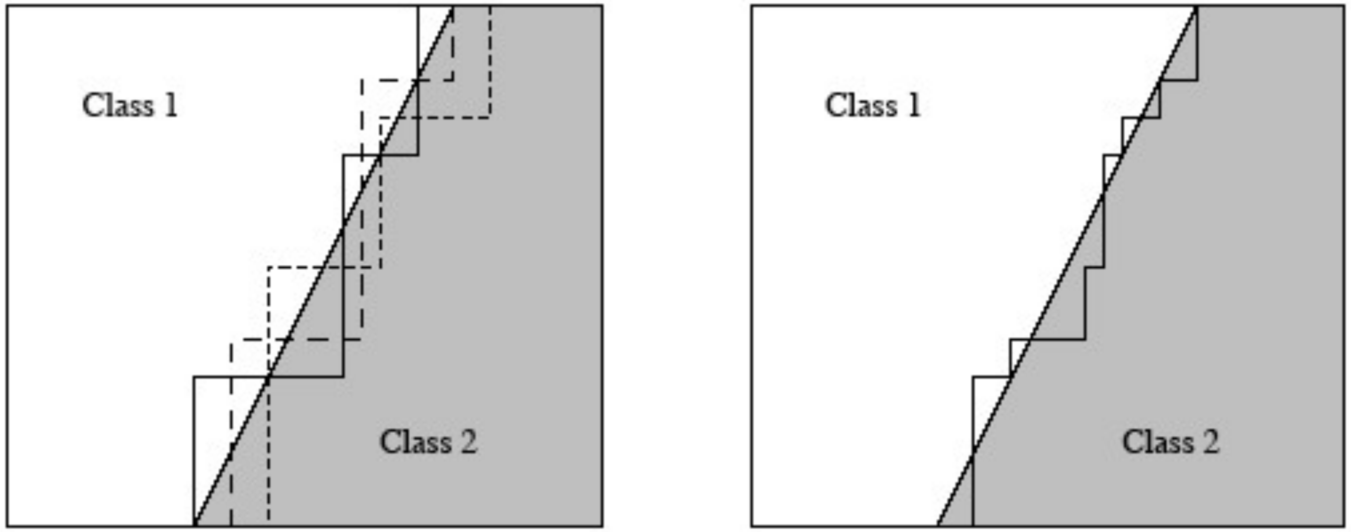


Fig. 4. The left figure shows the true diagonal decision boundary and three staircase approximations to it (of the kind that are created by decision tree algorithms). The right figure shows the voted decision boundary, which is a much better approximation to the diagonal boundary.

Michael Goebel's PhD Thesis

- Comparison of cross-validated communities and bagging
- One was better sometimes and the other sometimes
- Why?
  - Size of the dataset with respect to the hypothesis space
- 1/2 bag and double bag
- Why is bagging always better?

Random Forests

https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

- Each tree is grown as follows:
  1. Sample N (size of training set) cases at random - with replacement, from the original data. This sample will be the training set for growing the tree.
  2. For M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
  3. Each tree is grown to the largest extent possible. There is no pruning.

Error Rates in Random Forests

- In the original paper on random forests, it was shown that the forest error rate depends on two things:
  - The correlation between any two trees in the forest. Increasing the correlation increases the forest error rate.
  - The strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

Finding a good m

- Reducing m reduces both the correlation and the strength.
- Increasing it increases both. Somewhere in between is an "optimal" range of m - usually quite wide.
- Using the oob error rate a value of m in the range can quickly be found. This is the only adjustable parameter to which random forests is somewhat sensitive.

out-of-bag (oob) error estimate

In random forests, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. It is estimated internally, during the run, as follows:

- Each tree is constructed using a different bootstrap sample from the original data. About one-third of the cases are left out of the bootstrap sample and not used in the construction of the kth tree.

- Put each case left out in the construction of the kth tree down the kth tree to get a classification. In this way, a test set classification is obtained for each case in about one-third of the trees. At the end of the run, take j to be the class that got most of the votes every time case n was oob. The proportion of times that j is not equal to the true class of n averaged over all cases is the oob error estimate. This has proven to be unbiased in many tests.

Features of Random Forests

- It is unexcelled in accuracy among current algorithms.
- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets. - http://statistics.berkeley.edu/sites/default/files/tech-reports/666.pdf

More Features of Random Forests

- Generated forests can be saved for future use on other data.
- Prototypes are computed that give information about the relation between the variables and the classification.
- It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.

Variable Importance

- In every tree grown in the forest, put down the oob cases and count the number of votes cast for the correct class. Now randomly permute the values of variable m in the oob cases and put these cases down the tree. Subtract the number of votes for the correct class in the variable-m-permuted oob data from the number of votes for the correct class in the untouched oob data. The average of this number over all trees in the forest is the raw importance score for variable m.

- If the values of this score from tree to tree are independent, then the standard error can be computed by a standard computation. The correlations of these scores between trees have been computed for a number of data sets and proved to be quite low, therefore we compute standard errors in the classical way, divide the raw score by its standard error to get a z-score, ands assign a significance level to the z-score assuming normality.

Variable Importance 2

- If the number of variables is very large, forests can be run once with all the variables, then run again using only the most important variables from the first run.

- For each case, consider all the trees for which it is oob. Subtract the percentage of votes for the correct class in the variable-m-permuted oob data from the percentage of votes for the correct class in the untouched oob data. This is the local importance score for variable m for this case, and is used in the graphics program RAFT.

Open Problems

1. When to use which ensemble methods - Adaboost best except when there is noisy data - so NEVER

2. Bagging and ECOC combined perform better than either separately - other combinations should be explored

3. Few systematic studies of ensembles on ANN and rule-learning systems

4. Ensembles take a lot of memory - can they be converted to less redundant representations?

5. Ensembles provide little insight - can we obtain explanations from ensembles?

Questions you should be able to answer

- What two things must be true for an ensemble to improve results?
- When does bagging ensembles work better then cross-validated and vice versa?
- Why does random forests perform so well?

References

- Ensemble Methods in Machine Learning, Thomas Dietterich, tgd@cs.orst.edu
- http://web.engr.oregonstate.edu/~tgd/publications/mcs-ensembles.pdf

- Random Forests, Leo Breiman, leo@stat.berkeley.edu
- https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
- https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf