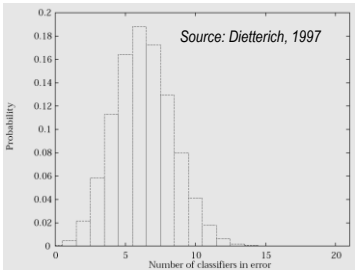
ENSEMBLES

- *Ensemble learning* refers to a collection of methods that learn a target function by training a number of individual learners (of similar or different types) and combining their predictions
- Why Ensembles?: The result from an ensemble model is usually better than the result from one of the individual models
  - **Accuracy**: a more reliable mapping can be obtained by combining the output of multiple "experts"
  - **Efficiency**: a complex problem can be decomposed into multiple sub-problems that are easier to understand and solve (divide-and-conquer approach)
  - There is not a single model that works for all PR problems!

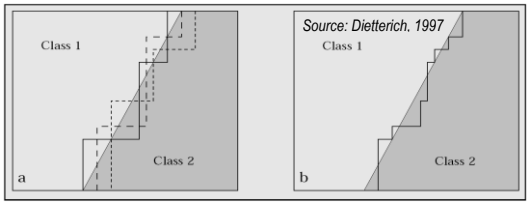
    "To solve really hard problems, we'll have to use several different representations..... It is time to stop arguing over which type of pattern-classification technique is best..... Instead we should work at a higher level of organization and discover how to build managerial systems to exploit the different virtues." [Minsky, 1991]
- When to Use Ensemble Learning?: When you can build component classifiers that are more accurate than chance and, more importantly, that are independent from each other

WHY DO ENSEMBLES WORK?

- Because uncorrelated errors of individual classifiers can be eliminated through averaging
  - Assume a binary classification problem for which you can train (independent) individual classifiers with error rate of 0.3 (30%)
  - Assume that you build ensemble by combining the prediction of 21 such classifiers with a majority vote
  - What is the probability of error for the ensemble?
    - In order for the ensemble to misclassify an example, 11 or more classifiers have to be in error, or a probability of 0.026
    - Histogram shows the distribution of number of classifiers that are in error in the ensemble.



- The target function may not be implementable with single classifiers, but may be approximated by ensemble averaging
  - Assume that you want to build a diagonal decision boundary with decision trees
  - Decision boundaries constructed by these machines are hyperplanes parallel to coordinate axes, or "staircases" in the example below
  - By averaging a large number of such "staircases", the diagonal decision boundary can be approximated with arbitrarily small accuracy

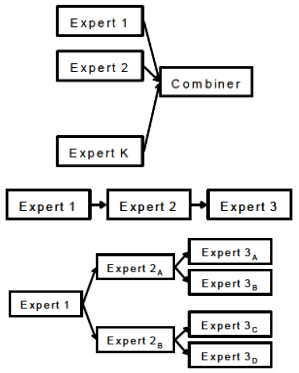


METHODS FOR CONSTRUCTING ENSEMBLES

- *Subsampling the Training Examples*: Multiple hypotheses are generated by training individual classifiers on different datasets obtained by resampling a common training set (e.g., Bagging, Boosting)
- *Manipulating the Input Features*: Multiple hypotheses are generated by training individual classifiers on different representations, or different subsets of a common feature vector (e.g., Random Forests)
- *Modifying the Learning Parameters of the Classifier*: A number of classifiers are built with different learning parameters, such as number of neighbors in a k Nearest Neighbor rule, initial weights in an MLP, etc.

STRUCTURE OF ENSEMBLE CLASSIFIERS

- *Parallel*:
  - All the individual classifiers are invoked independently, and their results are fused with a combination rule (e.g., average, weighted voting) or a metaclassifier (e.g., stacked generalization)
  - The majority of ensemble approaches in the literature fall under this category



Source: Jain, 2000

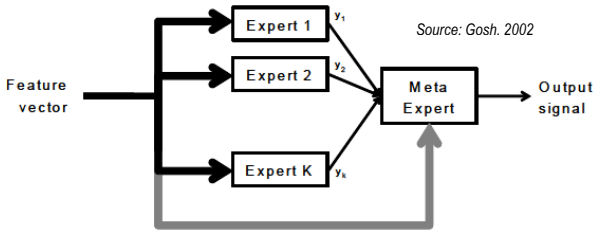
- *Cascading or Hierarchical*:
  - Classifiers are invoked in sequential or tree-structured fashion
  - For the purpose of efficiency, inaccurate but fast methods are invoked first (maybe using a small subset of the features), and computationally more intensive but accurate methods are left for the latter stages

COMBINATION STRATEGIES

- *Static Combiners*:
  - Combiner decision rule is independent of the feature vector. Static approaches can be broadly divided into *non-trainable and trainable.*
  - Non-trainable: Voting is performed independent of performance of each individual classifier
    - Various combiners may be used, depending on the type of output produced by the classifier, including
      - *Voting*: used when each classifier produces a single class label. In this case, each classifier "votes" for a particular class, and the class with the majority vote on the ensemble wins
      - *Averaging*: used when each classifier produces a confidence estimate (e.g., a posterior). In this case, the winner is the class with the highest average posterior across the ensemble
  - Trainable: The combiner undergoes a separate training phase to improve the performance of the ensemble. Two noteworthy approaches are:
    - *Weighted averaging*: the output of each classifier is weighted by a measure of its own performance, e.g., prediction accuracy on a separate validation set
    - *Stacked generalization*: the output of the ensemble serves as a feature vector to a meta-classifier
- *Adaptive Combiners*:
  - The combiner is a function that depends on the input feature vector
    - Thus, the ensemble implements a function that is local to each region in feature space

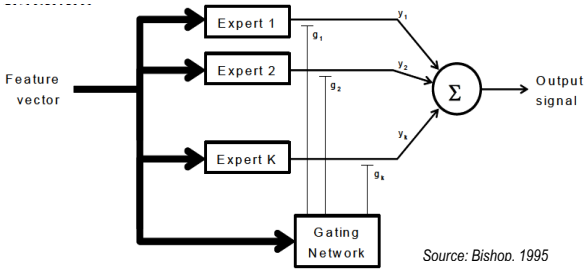
- This divide-and-conquer approach leads to modular ensembles where relatively simple classifiers specialize in different parts of I/O space
- In contrast with static-combiner ensembles, the individual experts here do not need to perform well for all inputs, only in their region of expertise

- Representative examples of this approach are Mixture of Experts (ME) and Hierarchical ME [Jacobs et al., 1991; Jordan and Jacobs, 1994]



MIXTURE OF EXPERTS

- ME is the classical adaptive ensemble method
  - A gating network is used to partition feature space into different regions, with one expert in the ensemble being responsible for generating the correct output within that region [Jacobs et al., 1991]
  - The experts in the ensemble and the gating network are trained simultaneously, which can be efficiently performed with EM
  - ME can be extended to a multi-level hierarchical structure, where each component is itself a ME. In this case, a linear network can be used for the terminal classifiers without compromising the modeling capabilities of the machine



SUBSAMPLING THE TRAINING SET: *BAGGING* [Brieman, 1996]

- Bagging (for bootstrap aggregation) creates an ensemble by training individual classifiers on bootstrap samples of the training set
- As a result of the sampling-with-replacement procedure, each classifier is trained on the average of 63.2% of the training examples
- For a dataset with N examples, each example has a probability of $1-(1-1/N)^N$ of being selected at least once in the N samples.
- Bagging traditionally uses component classifiers of the same type (e.g., decision trees), and a simple combiner consisting of a majority vote across the ensemble
- Bagging can be expected to improve accuracy if the induced classifiers are uncorrelated
  - In some cases, such as k Nearest Neighbors, bagging has been shown to degrade performance as compared to individual classifiers as a result of effectively smaller training set

SUBSAMPLING THE TRAINING SET: *BOOSTING* [Schapire, 1990; Freund and Schapire, 1996]

- Boosting takes a different resampling approach than bagging, which maintains a constant probability of $1/N$ for selecting each example
- In boosting, this probability is adapted over time based on performance
  - The component classifiers are built sequentially, and examples that are mislabeled by previous components are chosen more often than those that are correctly classified
- A number of variants of boosting are available in the literature. We focus on the most popular form, known as AdaBoost (for Adaptive Boosting), which allows the designer to continue adding components until an arbitrarily small error rate is obtained on the training set
- ***AdaBoost***:
  - At iteration n, boosting provides the weak learner with a distribution Dn over the training set, where $Dn(i)$ represents the probability of selecting the i-th example
    - The initial distribution is uniform: $D1(i)=1/N$. Thus, all examples are equally likely to be selected for the first component
  - The weak learner subsamples the training set according to Dn and generates a trained model or hypothesis Hn
  - The error rate of Hn is measured with respect to the distribution Dn
  - A new distribution $Dn+1$ is produced by decreasing the probability of those examples that were correctly classified, and increasing the probability of the misclassified examples
  - The process is repeated T times, and a final hypothesis is obtained by weighting the votes of individual hypotheses $\{h1,h2,\ldots,hT\}$ according to their performance

Notes:

- The strength of AdaBoost derives from the adaptive re-sampling of examples, not from the final weighted combination
- Both Bagging and Boosting address the bias-variance dilemma by reducing the variance term

RANDOM FORESTS [Brieman, 1996]

- *Random Forest* is an ensemble classifier using many decision tree models
- It combines Breiman's "bagging" idea and random selection of features, introduced first by Tin Kam Ho (1995)
- Can be used for classification or regression
- Accuracy and variable importance information is provided with the results
- How Random Forests Work:
  - A different subset of the training data are selected (~2/3), with replacement, to train each tree
  - Remaining training data are used to estimate error and variable importance
  - Class assignment is made by the number of votes from all of the trees and for regression the average of the results is used
- Using a Subset of Variables:
  - A randomly selected subset of variables is used to split each node
  - The number of variables used is decided by the user
  - Smaller subset produces less correlation (lower error rate) but lower predictive power (high error rate)
  - Optimum range of values is often quite wide
- Common Parameters for Random Forests:
  - Input data (predictor and response)
  - Number of trees
  - Number of variables to use at each split
  - Options to calculate error and variable significance information
  - Sampling with or without replacement
- Proximity Measure:
  - Proximity measures how frequent unique pairs of training samples (in and out of bag) end up in the same terminal node
  - Used to fill in missing data and calculating outliers

- Information from Random Forests
  - Classification accuracy
  - Variable importance
  - Outliers (classification)
  - Missing data estimation
  - Error rates for random forest objects
- Advantages: No need for pruning trees, Accuracy and variable importance generated automatically, Overfitting is not a problem, Not very sensitive to outliers in training data, and Easy to set parameters
- Limitations: Not quite a model (collection of often a large number of decision trees)

IMPLEMENTING ENSEMBLES IN MATLAB

- Matlab provides a variety of ensemble algorithms: AdaBoostM1, AdaBoostM2, Bag, GentleBoost, LogitBoost, LPBoost, LSBoost, RobustBoost, RUSBoost, Subspace, TotalBoost
- ***Classification Ensembles***: Can be built using boosting, random forest, bagging, random subspace, and ECOC ensembles for multiclass learning
  - A classification ensemble is a predictive model composed of a weighted combination of multiple classification models. In general, combining multiple classification models increases predictive performance.
  - To explore classification ensembles interactively, use the Classification Learner app.
  - For greater flexibility, use fitcensemble in the command-line interface to boost or bag classification trees, or to grow a random forest [11].
  - For details on all supported ensembles, see Ensemble Algorithms.
  - To reduce a multiclass problem into an ensemble of binary classification problems, train an error-correcting output codes (ECOC) model. For details, see fitcecoc.
- ***Regression Tree Ensembles***: Can be built using Random forests, boosted and bagged regression trees
  - A regression tree ensemble is a predictive model composed of a weighted combination of multiple regression trees. In general, combining multiple regression trees increases predictive performance.
  - To boost regression trees using LSBoost, use fitrensemble.
  - To bag regression trees or to grow a random forest [11], use fitrensemble or TreeBagger.
  - To implement quantile regression using a bag of regression trees, use TreeBagger.