L25: Ensemble learning

- Introduction
- Methods for constructing ensembles
- Combination strategies
- Stacked generalization
- Mixtures of experts
- Bagging
- Boosting

Introduction

What is ensemble learning?

- Ensemble learning refers to a collection of methods that learn a target function by training a number of individual learners and combining their predictions

Why ensemble learning?

- **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 and evade the different limitations of each of these ways of comparing things." [Minsky, 1991]

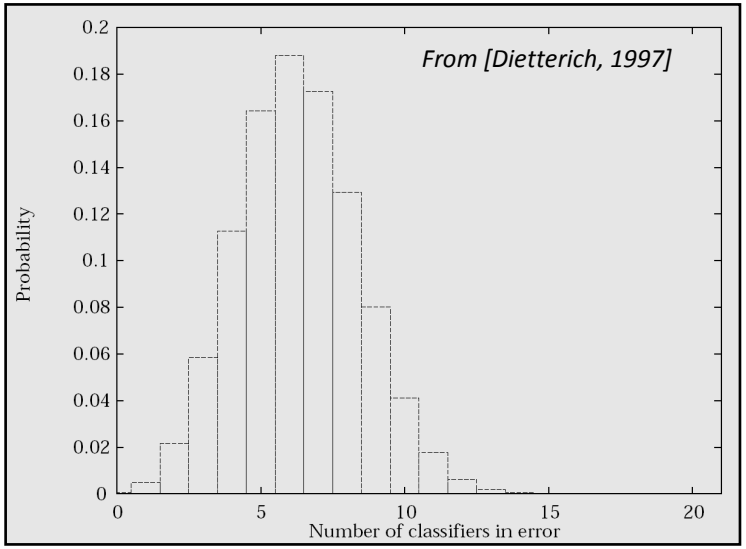
When to use ensemble learning?

- 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

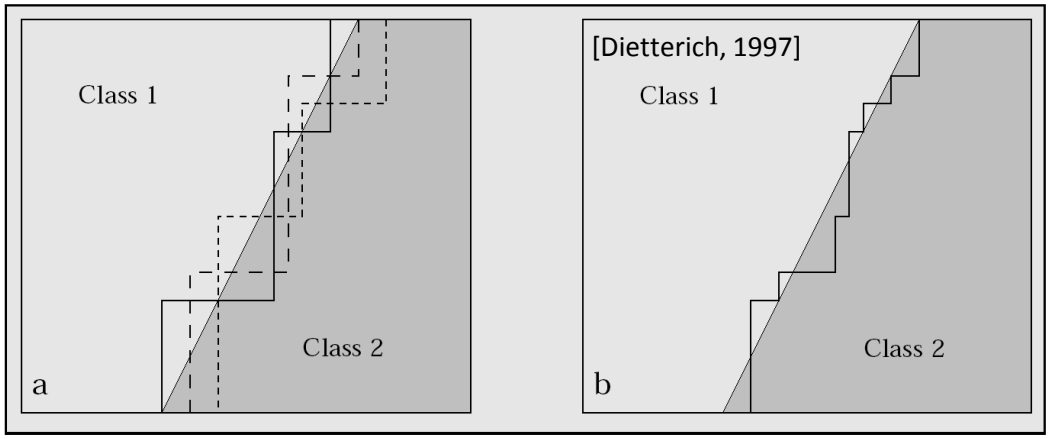
- Assume a binary classification problem for which you can train individual classifiers with an error rate of 0.3
- Assume that you build an ensemble by combining the prediction of 21 such classifiers with a majority vote
- What is the probability of error for the ensemble?
  - In order for the ensemble to misclassify an example, 11 or more classifiers have to be in error, or a probability of 0.026
  - The histogram below shows the distribution of the number of classifiers that are in error in the ensemble machine



[Dietterich, 1998]

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
- The decision boundaries constructed by these machines are hyperplanes parallel to the 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 (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

Manipulating the output targets

- The output targets for C classes are encoded with an L-bit codeword, and an individual classifier is built to predict each one of the bits in the codeword
- Additional “auxiliary” targets may be used to differentiate classifiers

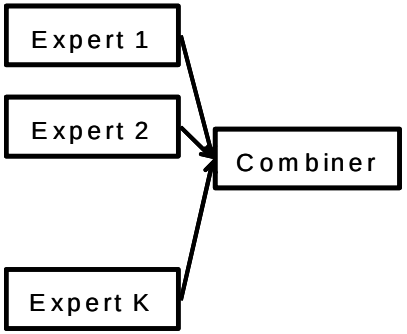
Modifying the learning parameters of the classifier

- 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

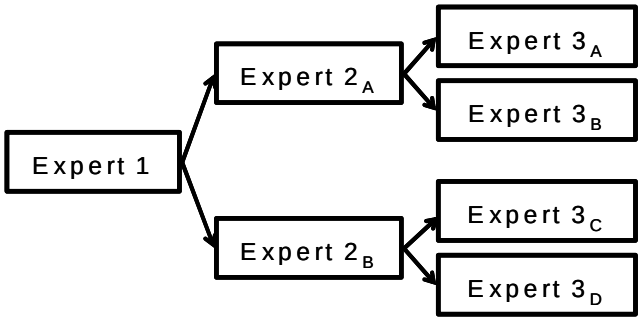
Parallel

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



Cascading or Hierarchical

- Classifiers are invoked in a 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



[Jain, 2000]

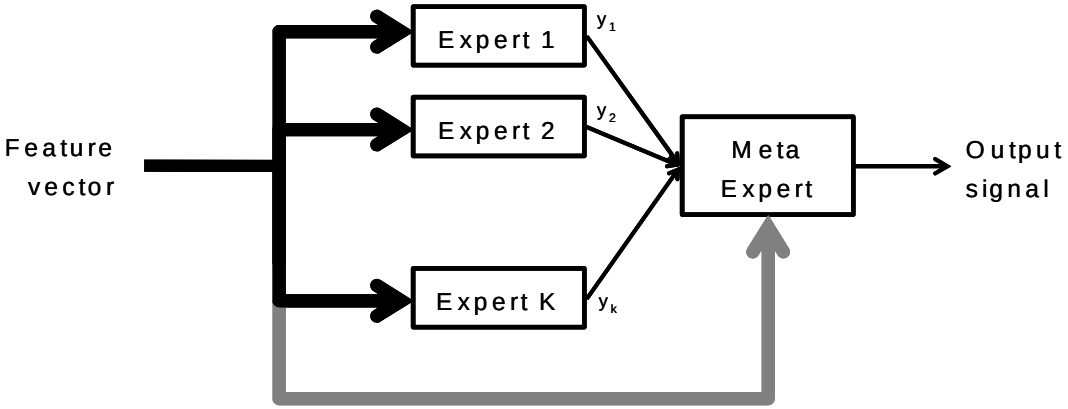
Combination strategies

Static combiners

- The combiner decision rule is independent of the feature vector. Static approaches can be broadly divided into non-trainable and trainable
- Non-trainable: The voting is performed independently of the 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
    - **Borda counts**: used when each classifier produces a rank. The Borda count of a class is the number of classes ranked below it [Ho et al., 1994]
- 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]

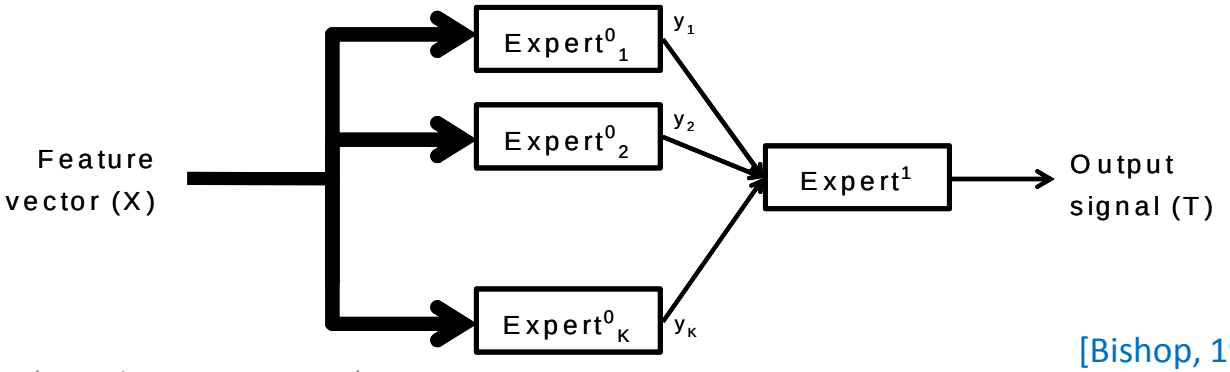


[Gosh, 2002]

Stacked generalization

In stacked generalization, the output of the ensemble serves as an input to a second-level expert [Wolpert, 1992]

- Training of this modular ensemble can be performed as follows
  - From a dataset X with N examples, leave out one test example, and train each of the level-0 experts on the remaining $N-1$ examples
  - Generate a prediction for the test example. The output pattern $y = [y_1, y_2, \ldots, y_k]$ across the level-0 experts, along with the target t for the test example, becomes a training example for the level-1 expert
  - Repeat this process in a leave-one-out fashion. This yields a training set Y with N examples, which is used to train the level-1 expert separately
  - To make full use of the training data, re-train all the level-0 experts one more time using all N examples in X

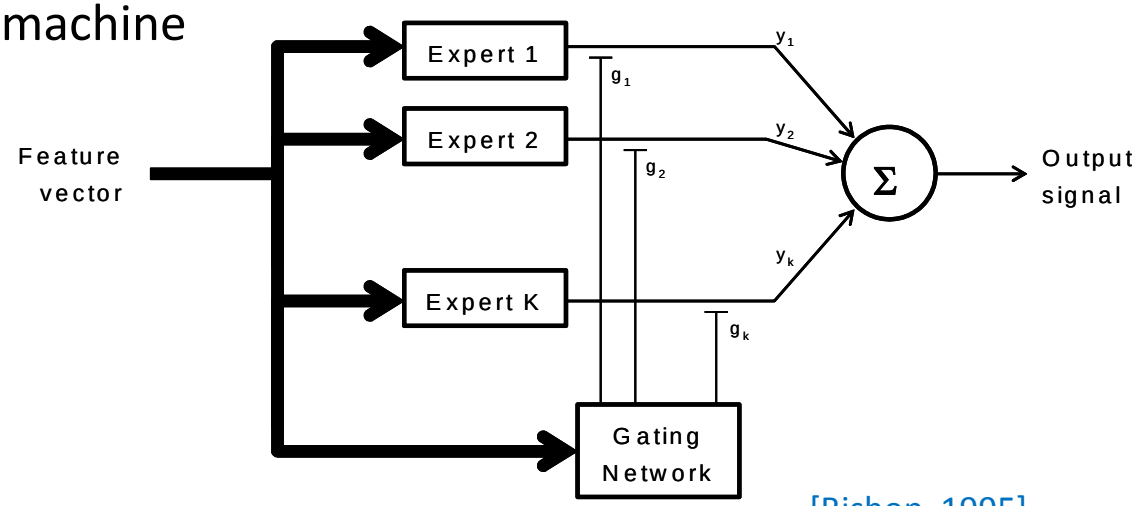


[Bishop, 1995]

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



[Bishop, 1995]

Subsampling the training set

Bagging [Breiman, 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.
  - For $N \to \infty$, this number converges to $(1 - 1/e)$ or 0.632 [Bauer and Kohavi, 1999]
- 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

[Bauer and Kohavi, 1999l; Duda et al., 2001]

Bagging (continued)

- The perturbation in the training set due to the bootstrap resampling causes different hypotheses to be built, particularly if the classifier is unstable
  - A classifier is said to be unstable if a small change in the training data (e.g., order of presentation of examples) can lead to a radically different hypothesis. This is the case of decision trees and, arguably, neural networks
- 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 an effectively smaller training set
- A related approach to bagging is “cross-validated committees”, in which the component classifiers are built on different partitions of the training set obtained through k-fold cross-validation

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
- Boosting is based on the concept of a "weak learner", an algorithm that performs slightly better than chance (e.g., 50% classification rate on binary tasks)
  - Schapire has shown that a weak learner can be converted into a strong learner by changing the distribution of training examples
  - Boosting can also be used with classifiers that are highly accurate, but the benefits in this case will be very small
- 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
  - NOTE: boosting is also known as arcing (for adaptive resampling and combining), a term that was coined by Breiman

AdaBoost

AdaBoost operates as follows

- At iteration n, boosting provides the weak learner with a distribution D_n over the training set, where $D_n(i)$ represents the probability of selecting the i-th example
  - The initial distribution is uniform: $D_1(i) = 1/N$. Thus, all examples are equally likely to be selected for the first component
- The weak learner subsamples the training set according to D_n and generates a trained model or hypothesis H_n
- The error rate of H_n is measured with respect to the distribution D_n
- A new distribution D_{n+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 $\{h_1, h_2, \dots, h_T\}$ according to their performance

Note

- The strength of AdaBoost derives from the adaptive re-sampling of examples, not from the final weighted combination
  - To prove this point Breiman developed a variant of AdaBoost, known as 'arc-x4', in which the ensemble voting is unweighted [Breiman, 1996]: his results show that AdaBoost (referred to as 'arc-fs') and 'arc-x4' have similar performance [Bauer and Kohavi, 1999]

The bias and variance decomposition

The effectiveness of Bagging and Boosting has been explained in terms of the bias-variance decomposition of classification error

- The expected error of a learning algorithm can be decomposed into
  - A **bias** term that measures how closely the average classifier produced by the learning algorithm matches the target function
  - A **variance** term that measures how much the learning algorithm's predictions fluctuate for different training sets (of the same size)
  - An **intrinsic target noise**, which is the minimum error that can be achieved: that of the Bayes optimal classifier
- Following this line of reasoning, Breiman has suggested that both Bagging and Boosting reduce errors by reducing the variance term
- Along the same lines, Freund and Schapire have argued that Boosting also attempts to reduce the bias term since it focuses on misclassified samples
  - Work by Bauer and Kohavi, however, seems to indicate that Bagging can also reduce the bias term

[Opitz and Maclin, 1999]