Ensemble Learning

Ziping Zhao

School of Information Science and Technology
ShanghaiTech University, Shanghai, China

CS182: Introduction to Machine Learning (Fall 2021)
http://cs182.sist.shanghaitech.edu.cn

Outline

Introduction

Voting

Boosting

Outline

Rationale

- No Free Lunch Theorem: There is no single learning algorithm that in any domain always induces the most accurate learner.
- The usual approach is to try many and choose the one that performs the best on a separate validation set.
- Each learning algorithm dictates a certain model with a set of assumptions, i.e., the inductive bias, leading to error if the assumptions do not hold for the data.
- Ensemble learning:
  - We construct a group of base-learners which, when combined, has higher accuracy than the individual learners.
  - The base-learners are usually not chosen for their accuracy, but for their simplicity.
  - The base-learners should be diverse, that is, accurate on different instances, specializing in subdomains of the problem, so that they can complement each other.

Diverse Base-Learners

- Different learning algorithms: different algorithms make different assumptions about the data and lead to different classifiers.
- Different hyperparameters of the same learning algorithm: e.g., number of hidden units in a multilayer perceptron, k in k-nearest neighbor classifier, error threshold in a decision tree, initial state of an iterative procedure, etc.
- Different representations of the same input object or event: multiple sources of information are combined, e.g., both acoustic input and video sequence of lip movements for speech recognition.
- Different training sets: multiple base-learners are trained either in parallel or serially using different training sets.
- Different subtasks: the main task is defined in terms of a number of subtasks solved by different base-learners.

Combining Base-Learners

- There are different ways the multiple base-learners are combined to generate the final output.
- Multiexpert combination methods:
  - The base-learners work in parallel.
  - Given an instance, they all give their decisions which are then combined to give the final decision.
  - E.g., voting, mixture of experts, stacked generalization.
- Multistage combination methods:
  - The base-learners work in serial.
  - The base-learners are sorted in increasing complexity: a complex base learner is not used unless the preceding simpler base-learners are not confident.
  - E.g., boosting, cascading.
- Given L base-learners, we denote by $d_j(x)$ the prediction of the jth base-learner given the input x. The final prediction is calculated from

$$y = f(d_1, d_2, ..., d_L \mid \mathbf{\Phi})$$

where $f(\cdot)$ is the combining/fusion function with $\mathbf{\Phi}$ denoting its parameters.

Outline

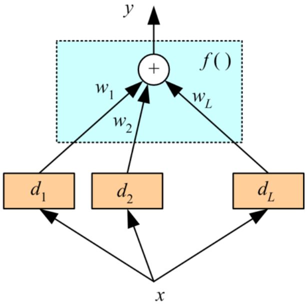
Voting

- Voting takes a convex combination of the base-learners:

$$y = f(d_1, \ldots, d_L \mid \boldsymbol{\Phi}) = \sum_{j=1}^{L} w_j d_j$$

  where y is the final prediction and $\boldsymbol{\Phi} = (w_1, \ldots, w_L)^T$ with w_j the weight satisfying

$$w_j \geq 0 \text{ and } \sum_{j=1}^{L} w_j = 1.$$



- When there are K outputs, for each learner there are $d_{ji}(x)$, $j = 1, \ldots, L$, $i = 1, \ldots, K$, and, combining them, we also generate K values, y_i, $i = 1, \ldots, K$.
- For example in classification, we choose the class with the maximum y_i value:

$$\text{Choose } C_i \text{ if } y_i = \max_k y_k$$

Combination Rules

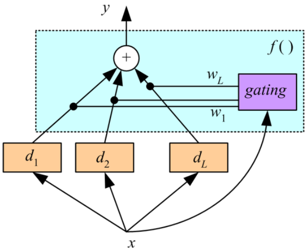
Rule	Fusion function $f(\cdot)$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$

- Sum rule is the most intuitive and is the most widely used in practice.
- Median rule is more robust to outliers.
- Minimum and maximum rules are pessimistic and optimistic, respectively.
- With the product rule, each learner has veto power; regardless of the other ones, if one learner has an output of 0, the overall output goes to 0.

Mixture of Experts as A Voting

- In voting, the weights w_j are constant over the input space x.
- The mixture of experts architecture may be seen as a voting method based on the weighted sum in which the weights depend on the input and are in general different for different inputs.
- The weights of the experts are determined by a gating network:

$$y = \sum_{j=1}^{L} w_j(x) d_j$$



- The competitive learning algorithm used by the mixture of experts localizes the base-learners such that each of them becomes an expert in a different part of the input space and have its weight, $w_j(x)$, close to 1 in its region of expertise.

Voting for Classification

- For class C_i:

$$y_i = \sum_{j=1}^{L} w_j d_{ji}$$

  where d_{ji} is the vote of learner j for C_i and w_j is the weight of its vote.
- Simple voting (a.k.a. plurality voting, or majority voting for 2-class problem):

$$w_j = \frac{1}{L}$$

- If the voters can also supply the additional information of how much they vote for each class (e.g., by the posterior probability).
- Another possible way to find w_j is to assess the accuracies of the learners (regressor or classifier) on a separate validation set and use that information to compute the weights, so that we give more weights to more accurate learners.

Voting Under A Bayesian Framework

- Voting schemes can be seen as approximations under a Bayesian framework.
  - weights w_j approximating prior model probabilities $P(\mathcal{M}_j)$, and
  - model decisions $d_{ji}(x)$ approximating model-conditional likelihoods $P(C_i \mid x, \mathcal{M}_j)$.
- For example, in classification Bayesian model combination:

$$P(C_i \mid x) = \sum_{\text{all models } \mathcal{M}_j} P(\mathcal{M}_j) P(C_i \mid x, \mathcal{M}_j)$$

- Simple voting corresponds to a uniform prior.
- If we have a prior distribution preferring simpler models, this would give larger weights to them.
- We cannot integrate over all models; we only choose a subset for which we believe $P(\mathcal{M}_j)$ is high, or we can have another Bayesian step and calculate $P(\mathcal{M}_j \mid \mathcal{X})$, the probability of a model given the sample, and sample high probable models from this density.

Analysis

- Let there be L independent two-class classifiers, where $\mathrm{E}[d_j]$ and $\mathrm{Var}(d_j)$ are the expected value and variance of d_j for classifier j.
- Expected value and variance of output for independent classifiers:

$$\mathrm{E}[y] = \mathrm{E}\Big[\sum_j \frac{1}{L} d_j\Big] \geq \frac{1}{L} L \min_j \{\mathrm{E}[d_j]\} = \min_j \{\mathrm{E}[d_j]\}$$

$$\mathrm{Var}(y) = \mathrm{Var}\Big(\sum_j \frac{1}{L} d_j\Big) = \frac{1}{L^2}\mathrm{Var}\Big(\sum_j d_j\Big) \leq \frac{1}{L^2} L \max_j \{\mathrm{Var}(d_j)\} = \frac{1}{L} \max_j \{\mathrm{Var}(d_j)\}$$

  As L increases, the expected value (and hence the bias) does not change but the variance decreases, and hence the mean squared error of the estimator y decreases, leading to an increase in accuracy.
- General case (non-independent classifiers):

$$\mathrm{Var}(y) = \frac{1}{L^2}\mathrm{Var}\Big(\sum_j d_j\Big) = \frac{1}{L^2}\Big[\sum_j \mathrm{Var}(d_j) + 2\sum_j \sum_{i<j} \mathrm{Cov}(d_j, d_i)\Big]$$

Bagging

- Bagging, short for bootstrap aggregating, is a voting method whereby the base-learners are made different by training on slightly different training sets.
- The L different training sets are generated by bootstrap, which draws N instances randomly from a training set $\mathcal{X}$ of size N with replacement.
- Bagging can be seen as a special case of model averaging which helps to reduce variance and hence improve accuracy.
- Unstable algorithms (e.g., decision trees and multilayer perceptrons) that cause large changes in the generated learner (i.e., high variance) with small changes in the training set can particularly benefit from bagging.

Outline

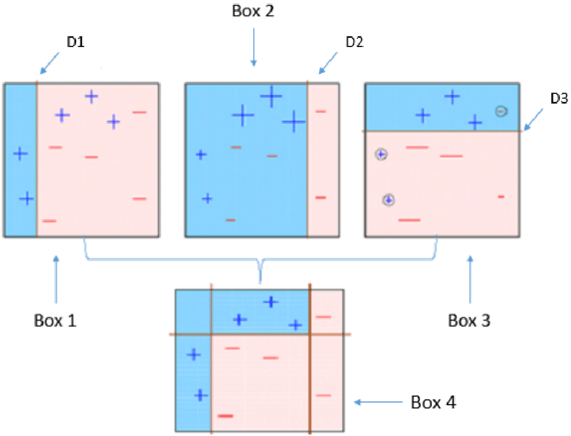
Boosting

- In bagging, generating complementary base-learners is left to chance and to the instability of the learning algorithm.
- In boosting, complementary base-learners are generated by training the next learner on the mistakes of the previous learners.
- Boosting combines weak learners (learners with accuracy just required to be better than random guessing, i.e., $> 1/K$ for K-class classification problems; weak but not too weak) to generate a strong learner (learners with arbitrarily small error probability).

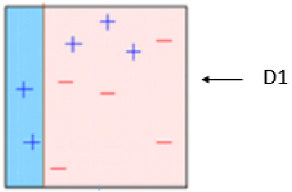
AdaBoost

- AdaBoost (a short form for adaptive boosting) is an iterative procedure that generates a sequence of base-learners each focusing on the errors of previous ones.
- The original algorithm is AdaBoost.M1, but many variants of AdaBoost have also been proposed.
- AdaBoost modifies the probabilities of drawing instances for classifier training as a function of the error of the previous base-learner.
- Initially all N instances have the same probability of being drawn.
- Moving from one iteration to the next iteration, the probability of a correctly classified instance is decreased and that of a misclassified instance is increased.
- The success of AdaBoost is due to its property of increasing the margin, making the aim of AdaBoost similar to that of SVM.
- One statistical view of boosting is an additive form of logistic regression.

A Simple Example

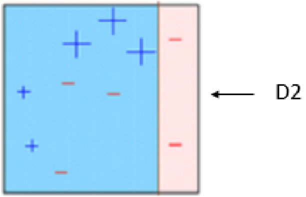


A Simple Example: Base Learner 1



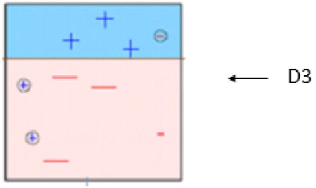
- All data points have equal weights.
- Base-learner D1, which is a simple decision tree with only one single level (a.k.a. decision stump), misclassifies three + (plus) data points as − (minus).

A Simple Example: Base Learner 2



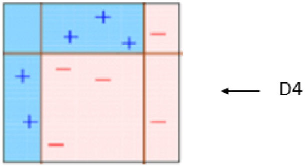
- The three data points misclassified by D1 now have higher weights, making it more likely for base learner D2 to classify them correctly.
- D2 misclassifies three $-$ (minus) data points as $+$ (plus).

A Simple Example: Base Learner 3



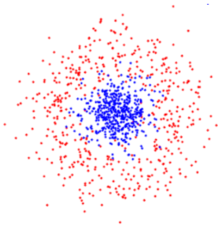
- The three data points misclassified by D2 now have higher weights, making it more likely for base learner D3 to classify them correctly.
- D3 generates a horizontal decision boundary.

A Simple Example: Combining Base-Learners



- The three weak learners (D1, D2, D3) are combined to give a strong learner (D4) which can classify all data points correctly.
- Although the weak learners are simple linear classifiers, the strong learner is highly nonlinear.

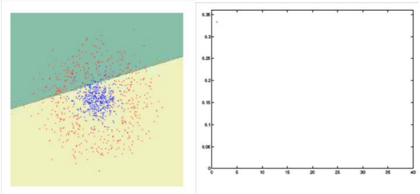
A More Realistic Example



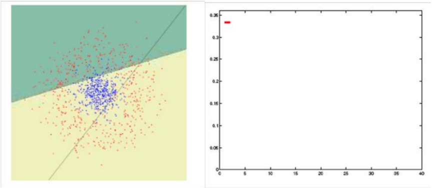
- The data distribution is radial with uniform angular distribution.
- Linear classifiers are used as weak learners.

A More Realistic Example: $t = 1$ and $t = 2$

- $t = 1$:

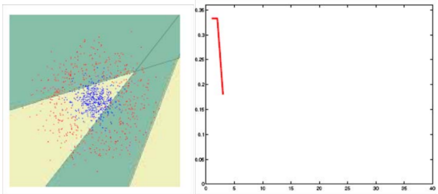


- $t = 2$:

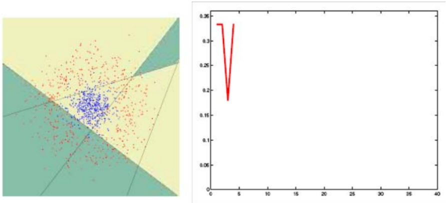


A More Realistic Example: $t = 3$ and $t = 4$

- $t = 3$:

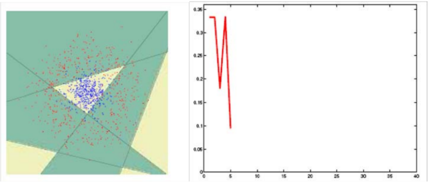


- $t = 4$:

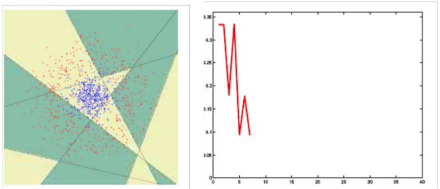


A More Realistic Example: $t = 5$ and $t = 6$

- $t = 5$:

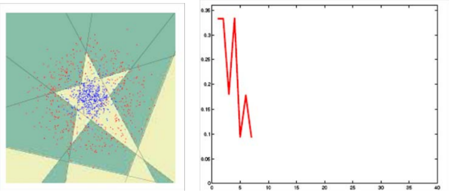


- $t = 6$:

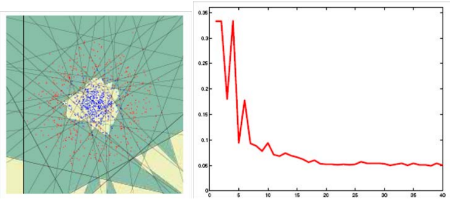


A More Realistic Example: $t = 7$ and $t = 40$

- $t = 7$:



- $t = 40$:



AdaBoost Algorithm

Training:

For all $\{x^t, r^t\}_{t=1}^N \in \mathcal{X}$, initialize $p_1^t = 1/N$

For all base-learners $j = 1, \ldots, L$

Randomly draw $\mathcal{X}_j$ from $\mathcal{X}$ with probabilities p_j^t

Train d_j using $\mathcal{X}_j$

For each (x^t, r^t), calculate $y_j^t \leftarrow d_j(x^t)$

Calculate error rate: $\epsilon_j \leftarrow \sum_t p_j^t \cdot 1(y_j^t \neq r^t)$

If $\epsilon_j > 1/2$, then $L \leftarrow j - 1$; stop

$\beta_j \leftarrow \epsilon_j / (1 - \epsilon_j)$

For each (x^t, r^t), decrease probabilities if correct:

If $y_j^t = r^t$, then $p_{j+1}^t \leftarrow \beta_j p_j^t$ Else $p_{j+1}^t \leftarrow p_j^t$

Normalize probabilities:

$Z_j \leftarrow \sum_t p_{j+1}^t; \quad p_{j+1}^t \leftarrow p_{j+1}^t / Z_j$

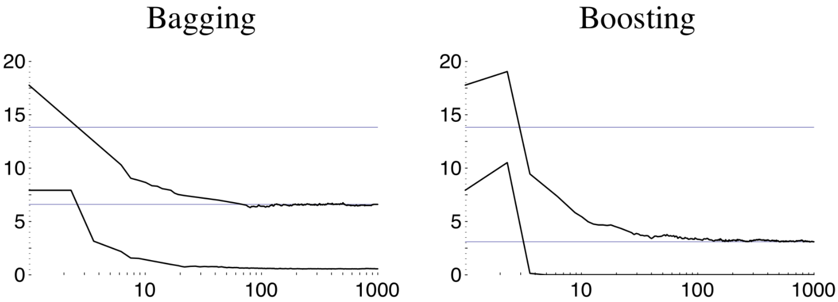
Testing:

Given x, calculate $d_j(x), j = 1, \ldots, L$

Calculate class outputs, $i = 1, \ldots, K$:

$$y_i = \sum_{j=1}^L \left(\log \frac{1}{\beta_j}\right) d_{ji}(x)$$

Bagging vs. Boosting: Training and Test Error Curves



Methods Comparisons