

Foundation of Machine Learning (CSE4032)

Lecture 11: Ensemble Methods

Dr. Kundan Kumar

Associate Professor

Department of ECE



Faculty of Engineering (ITER)

S'O'A Deemed to be University, Bhubaneswar, India-751030

© 2021 Kundan Kumar, All Rights Reserved

Outline

- 1 Introduction
- 2 Majority Voting
- 3 Soft Majority Voting
- 4 Bagging
- 5 Boosting
- 6 Random Forests
- 7 References

Ensemble Methods

Introduction

- In broad terms, using **ensemble methods** is about **combining models** to an ensemble such that the ensemble has a better performance than an individual model on average.
- The main categories of ensemble methods
 - involve voting schemes among high-variance models to prevent “outlier” predictions and overfitting, and
 - the other involves boosting “weak learners” to become “strong learners.”

Majority Voting

- We will use the term “majority” throughout this lecture in the context of voting to refer to both majority and plurality voting.
- Plurality: mode, the class that receives the most votes; for binary classification, majority and plurality are the same.

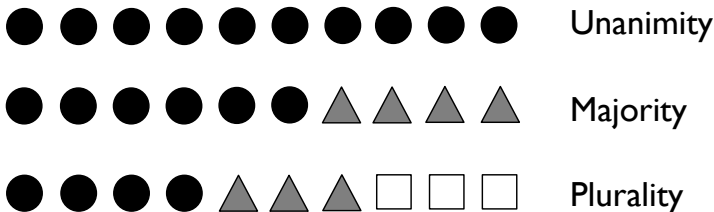
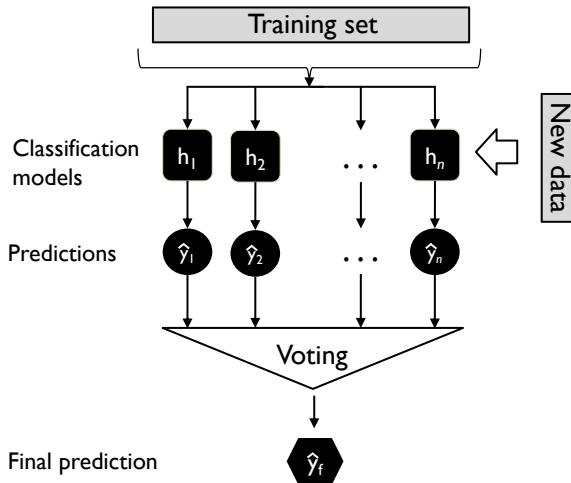


Figure: Illustration of unanimity, majority, and plurality voting

Majority Voting



Majority Voting

- In Nearest Neighbor Methods, we learned that the majority (or plurality) voting can simply be expressed as the mode:

$$\hat{y}_f = \text{mode} \{h_1(\mathbf{x}), h_2(\mathbf{x}), \dots, h_n(\mathbf{x})\}$$

- The following illustration demonstrates why majority voting can be effective (under certain assumptions).
 - Given are n independent classifiers $(h_1; \dots; h_n)$ with a base error rate ϵ .
 - Here, independent means that the errors are uncorrelated
 - Assume a binary classification task
- Assuming the error rate is better than random guessing (i.e., lower than 0.5 for binary classification),

$$\forall \epsilon_i \in \{\epsilon_1, \epsilon_2, \dots, \epsilon_n\}, \epsilon_i < 0.5$$

Majority Voting

- The error of the ensemble can be computed using a **binomial probability distribution** since the ensemble makes a wrong prediction if more than 50% of the n classifiers make a wrong prediction.
- The probability that we make a wrong prediction via the ensemble if k classifiers predict the same class label (where $k > \lceil n/2 \rceil$ because of majority voting) is then

$$P(k) = \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k}$$

where $\binom{n}{k}$ is the binomial coefficient

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}.$$

Majority Voting

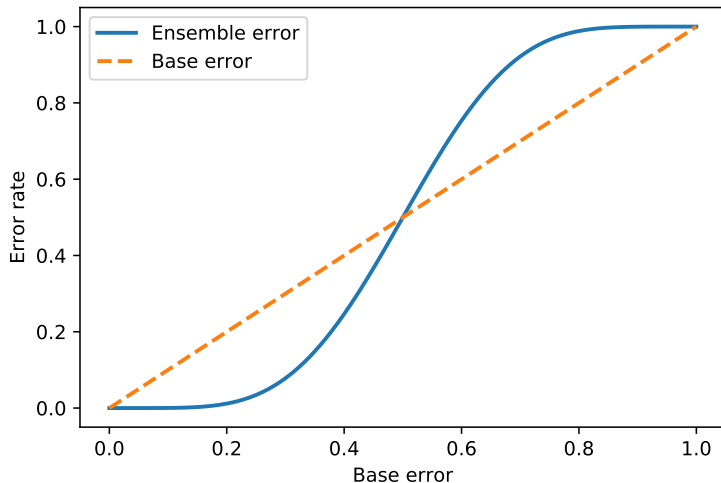
- However, we need to consider all cases $k \in \{\lceil n/2 \rceil, \dots, n\}$ (cumulative prob. distribution) to compute the ensemble error

$$\epsilon_{ens} = \sum_k^n \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k}.$$

- Consider the following example with $n = 11$ and $\epsilon = 0.25$, where the ensemble error decreases substantially compared to the error rate of the individual models:

$$\epsilon_{ens} = \sum_{k=6}^{11} \binom{11}{k} 0.25^k (1 - 0.25)^{11-k} = 0.034 .$$

Majority Voting



Soft Majority Voting

- For well calibrated classifiers we can also use the predicted class membership probabilities to infer the class label,

$$\hat{y} = \arg \max_j \sum_{i=1}^n w_i p_{i,j}$$

where $p_{i,j}$ is the predicted class membership probability for class label j by the i th classifier. Here w_i is an optional weighting parameter.

- If we set $w_i = 1/n, \forall w_i \in \{w_1, \dots, w_n\}$ then all probabilities are weighted uniformly.

Soft Majority Voting

- To illustrate this, let us assume we have a binary classification problem with class labels $j \in \{0, 1\}$ and an ensemble of three classifiers h_i ($i \in \{1, 2, 3\}$):

$$h_1(\mathbf{x}) \rightarrow [0.9, 0.1], \quad h_2(\mathbf{x}) \rightarrow [0.8, 0.2], \quad h_3(\mathbf{x}) \rightarrow [0.4, 0.6]$$

- We can then calculate the individual class membership probabilities as follows:

$$p(j = 0 \mid \mathbf{x}) = 0.2 \cdot 0.9 + 0.2 \cdot 0.8 + 0.6 \cdot 0.4 = 0.58, \quad (1)$$

$$p(j = 1 \mid \mathbf{x}) = 0.2 \cdot 0.1 + 0.2 \cdot 0.2 + 0.6 \cdot 0.6 = 0.42 \quad (2)$$

The predicted class label is then

$$\hat{y} = \arg \max_j \{p(j = 0 \mid \mathbf{x}), p(j = 1 \mid \mathbf{x})\} = 0$$

Bagging

- Bagging relies on a concept similar to majority voting but uses the same learning algorithm (typically a decision tree algorithm) to fit models on different subsets of the training data (bootstrap samples).
- Bagging can improve the accuracy of unstable models that tend to overfit.

Algorithm 1 Bagging

- 1: Let n be the number of bootstrap samples
 - 2:
 - 3: **for** $i=1$ to n **do**
 - 4: Draw bootstrap sample of size m , \mathcal{D}_i
 - 5: Train base classifier h_i on \mathcal{D}_i
 - 6: $\hat{y} = \text{mode}\{h_1(\mathbf{x}), \dots, h_n(\mathbf{x})\}$
-

Bagging

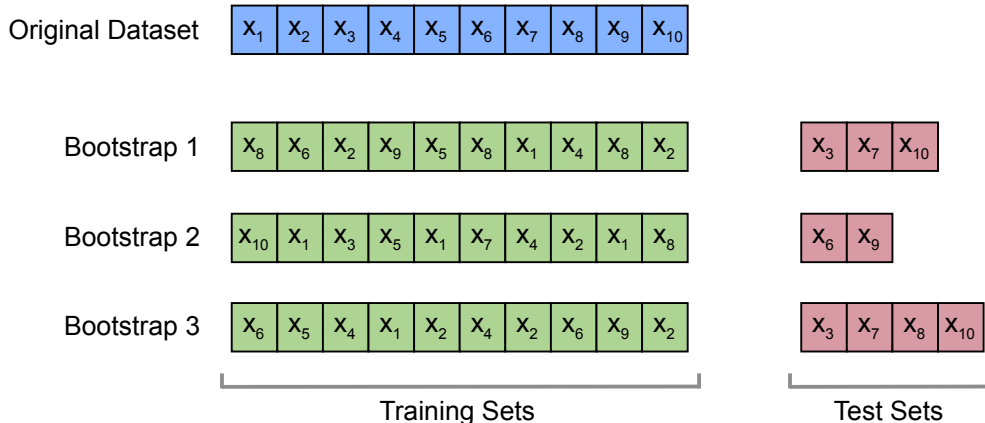


Figure: Illustration of bootstrap sampling

Bagging

- If we sample from a uniform distribution, we can compute the probability that a given example from a dataset of size n is not drawn as a bootstrap sample as

$$P(\text{not chosen}) = \left(1 - \frac{1}{n}\right)^n$$

which is asymptotically equivalent to

$$\frac{1}{e} \approx 0.368 \quad \text{as} \quad n \rightarrow \infty.$$

- Vice versa, we can then compute the probability that a sample is chosen as

$$P(\text{chosen}) = 1 - \left(1 - \frac{1}{n}\right)^n \approx 0.632.$$

Bagging

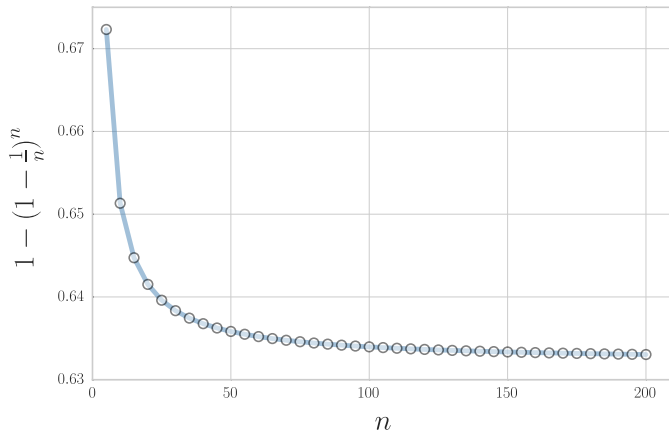
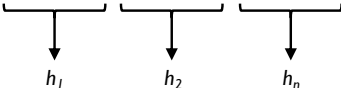


Figure: Proportion of unique training examples in a bootstrap sample.

Bagging

Training example indices	Bagging round 1	Bagging round 2	...
1	2	7	...
2	2	3	...
3	1	2	...
4	3	1	...
5	7	1	...
6	2	7	...
7	4	7	...



h_1 h_2 h_n

Figure: Illustration of bootstrapping in the context of bagging.

Bagging

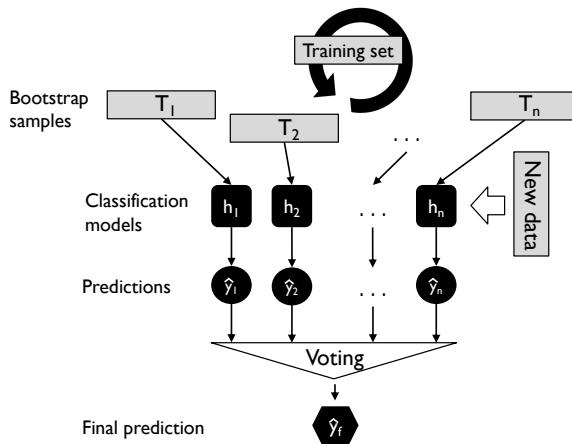


Figure: The concept of bagging. Here, assume n different classifiers, $\{h_1, h_2, \dots, h_m\}$ where $h_i(\mathbf{x}) = \hat{y}_i$

Boosting

- There are two broad categories of boosting: **Adaptive boosting** and **gradient boosting**.
- Adaptive and gradient boosting rely on the same concept of boosting “weak learners” (such as decision tree stumps) to “strong learners.”
- **Boosting is an iterative process**, where the training set is reweighted, at each iteration, based on mistakes a weak learner made (i.e., misclassifications);
- The two approaches, adaptive and gradient boosting, differ mainly regarding how the weights are updated and how the classifiers are combined.
- Since we have not discussed gradient-based optimization, in this lecture, we will focus on adaptive boosting.
- In particular, we will focus on AdaBoost as initially described by Freund and Schapire in 1997.

Boosting

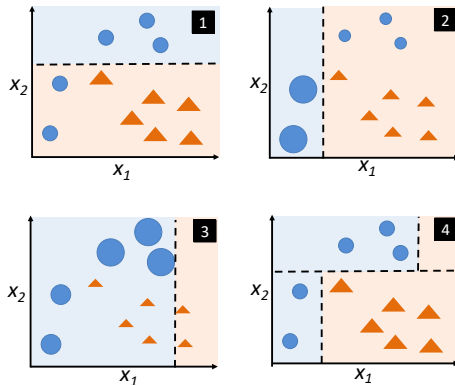
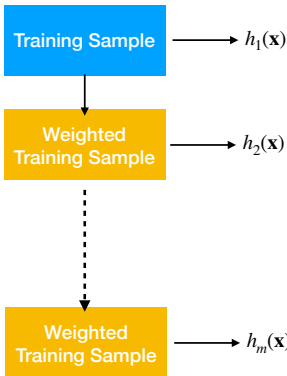


Figure: Illustration of the AdaBoost algorithms for three iterations on a toy dataset. The size of the symbols (circles shall represent training examples from one class, and triangles shall represent training examples from another class) is proportional to the weighting of the training examples at each round. The 4th subpanel shows a combination/ensemble of the hypotheses from subpanels 1-3.

Boosting



$$h_m(\mathbf{x}) = \text{sign} \left(\sum_{j=1}^m w_j h_j(\mathbf{x}) \right) \quad \text{for } h(\mathbf{x}) \in \{-1, 1\}$$

or $h_m(\mathbf{x}) = \arg \max_i \left(\sum_{j=1}^m w_j I[h_j(\mathbf{x}) = i] \right) \quad \text{for } h(\mathbf{x}) = i, \quad i \in \{1, \dots, n\}$

Boosting

Intuitively, we can outline the general boosting procedure as follows:

- Initialize a weight vector with uniform weights
- Loop:
 - Apply **weak** learner to weighted training examples (instead of orig. training set, may draw bootstrap samples with weighted probability)
 - Increase weight for misclassified examples
- (Weighted) majority voting on trained classifiers

Boosting

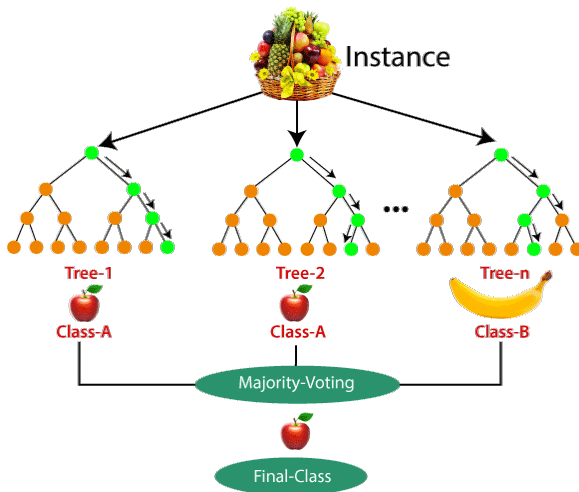
Algorithm 1 AdaBoost

- 1: Initialize k : the number of AdaBoost rounds
 - 2: Initialize \mathcal{D} : the training dataset, $\mathcal{D} = \{\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \dots, \langle \mathbf{x}^{[n]}, y^{[n]} \rangle\}$
 - 3: Initialize $w_1(i) = 1/n$, $i = 1, \dots, n$, $\mathbf{w}_1 \in \mathbb{R}^n$
 - 4:
 - 5: **for** $r=1$ to k **do**
 - 6: For all i : $\mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$ [normalize weights]
 - 7: $h_r := \text{FitWeakLearner}(\mathcal{D}, \mathbf{w}_r)$
 - 8: $\epsilon_r := \sum_i w_r(i) \mathbf{1}(h_r(i) \neq y_i)$ [compute error]
 - 9: if $\epsilon_r > 1/2$ then stop
 - 10: $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r)/\epsilon_r]$ [small if error is large and vice versa]
 - 11: $w_{r+1}(i) := w_r(i) \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$
 - 12: Predict: $h_k(\mathbf{x}) = \arg \max_j \sum_r \alpha_r \mathbf{1}[h_r(\mathbf{x}) = j]$
 - 13:
-

Random Forest

- The random forest algorithm is actually a bagging algorithm: also here, we draw random bootstrap samples from your training set.
- However, in addition to the bootstrap samples, we also draw random subsets of features for training the individual trees.
- In bagging, we provide each tree with the full set of features.
- Due to the random feature selection, the trees are more independent of each other compared to regular bagging, which often results in better predictive performance (due to better variance-bias trade-offs).
- It's also faster than bagging, because each tree learns only from a subset of features.

Random Forests



Random Forests

- Random forests are among the most widely used machine learning algorithm, probably due to their relatively good performance “out of the box” and ease of use (not much tuning required to get good results).
- In the context of bagging, random forests are relatively easy to understand conceptually: the random forest algorithm can be understood as bagging with decision trees, but instead of growing the decision trees by basing the splitting criterion on the complete feature set, we use random feature subsets.
- To summarize, in random forests, we fit decision trees on different bootstrap samples, and in addition, for each decision tree, we select a random subset of features at each node to decide upon the optimal split; while the size of the feature subset to consider at each node is a hyperparameter that we can tune, a “rule-of-thumb” suggestion is to use $NumFeatures = \log_2 m + 1$.

Random Forest

- Does random forest select a subset of features for every tree or every node?
- Earlier random decision forests by Tin Kam Ho used the “random subspace method,” where each tree got a random subset of features.
 - “The essence of the method is to build multiple trees in randomly selected subspaces of the feature space.” – Tin Kam Ho
- However, a few years later, Leo Breiman described the procedure of selecting different subsets of features for each node (while a tree was given the full set of features) — Leo Breiman’s formulation has become the “trademark” random forest algorithm that we typically refer to these days when we speak of “random forest.”
 - “... random forest with random features is formed by selecting at random, at each node, a small group of input variables to split on.”

Random Forest

Generalization Error

- The reason why random forests may work better in practice than a regular bagging model, for example, may be explained by the additional randomization that further diversifies the individual trees (i.e., decorrelates them).
- In Breiman's random forest paper, the upper bound of the generalization error is given as





$$\text{PE} \leq \frac{\bar{\rho} \cdot (1 - s^2)}{s^2}$$

- where $\bar{\rho}$ is the average correlation among trees and s measures the strength of the trees as classifiers. I.e., the average predictive performance concerning the classifiers' margin.

Random Forest

- The lower correlation, the lower the error. Similarly, the higher the strength of the ensemble or trees, the lower the error.
- So, randomization of the feature subspaces may decrease the “strength” of the individual trees, but at the same time, it reduces the correlation of the trees.
- Then, compared to bagging, random forests may be at the sweet spot where the correlation and strength decreases result in a better net result.

References

-  The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition, Hastie, Tibshirani, and Friedman, Springer.
-  In Introduction to Statistical Learning with Application in R, Second Edition, James, Witten, Hastie, and Tibshirani, Springer.
-  STAT 479: Machine Learning Lecture Notes, Sebastian Raschka, Department of Statistics University of Wisconsin–Madison https://sebastianraschka.com/pdf/lecture-notes/stat479fs18/07_ensembles_notes.pdf
-  <https://www.javatpoint.com/machine-learning-random-forest-algorithm>



Thank you!