Ensemble Learning
Lecture Notes

Contents

Ensemble Learning
Lecture Notes

7 Ensemble Methods

7.1 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.”

7.2 Majority Voting

- We will use the term “majority” throughout this lecture in the context of voting to refer to both majority and plurality voting.[1]
- Plurality: mode, the class that receives the most votes; for binary classification, majority and plurality are the same

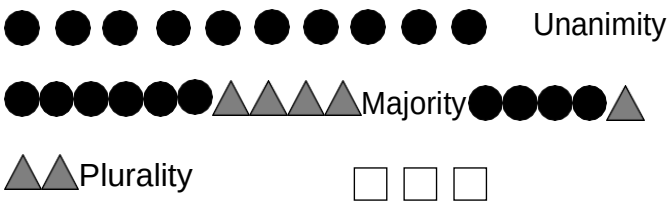

Figure 1: Illustration of unanimity, majority, and plurality voting

[1] In the UK, people distinguish between majority and plurality voting via the terms ”absolute” and ”relative” majority, respectively, https://en.wikipedia.org/wiki/Plurality_(voting).

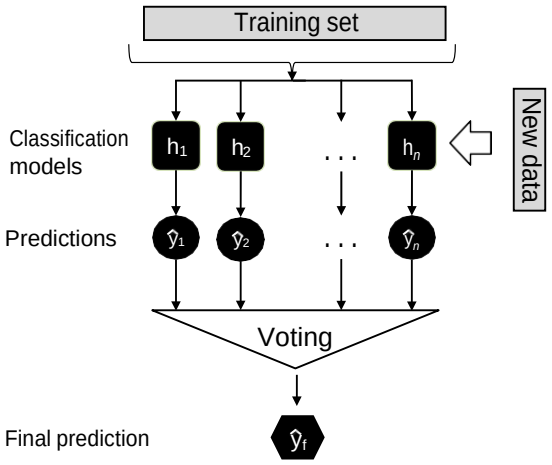

Figure 2: Illustration of the majority voting concept. Here, assume n different classifiers, $\{h_1, h_2, ..., h_m\}$ where $h_i(\mathbf{x}) = \hat{y}_i$.

In lecture 2, (*Nearest Neighbor Methods*) we learned that the majority (or plurality) voting can simply be expressed as the *mode*:

$$\hat{y}_f = mode\{h_1(\mathbf{x}), h_2(\mathbf{x}), ...h_n(\mathbf{x})\}. \quad (1)$$

The following illustration demonstrates why majority voting can be effective (under certain assumptions).

- Given are n independent classifiers ($h_1, ..., h_n$) 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),

$$\forall \epsilon_i \in \{\epsilon_1, \epsilon_2, ..., \epsilon_n\}, \epsilon_i < 0.5, \quad (2)$$

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

$$P(k) = \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k}, \quad (3)$$

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

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}. \quad (4)$$

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

$$\epsilon_{ens} = \sum_{k}^{n} \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k}. \tag{5}$$

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:

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

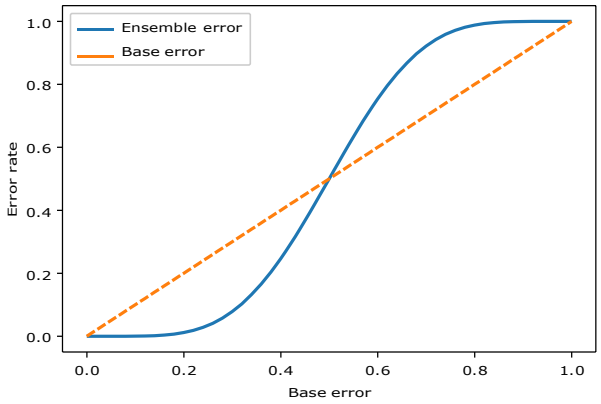

Figure 3: error-rate

7.3 Soft Majority Voting

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

$$\hat{y} = \arg\max_{j} \sum_{i=1}^{n} w_i p_{i,j}, \tag{7}$$

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

$w_i = 1/n, \forall w_i \in \{w_1, \ldots, w_n\}$

then all probabilities are weighted uniformly.

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 $h_i (i \in \{1, 2, 3\})$:

$$h_1(\mathbf{x}) \rightarrow [0.9, 0.1], h_2(\mathbf{x}) \rightarrow [0.8, 0.2], h_3(\mathbf{x}) \rightarrow [0.4, 0.6]. \tag{8}$$

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

$$p(j = 0|\mathbf{x}) = 0.2 \cdot 0.9 + 0.2 \cdot 0.8 + 0.6 \cdot 0.4 = 0.58, \; p(j = 1|\mathbf{x}) = 0.2 \cdot 0.1 + 0.2 \cdot 0.2 + 0.6 \cdot 0.6 = 0.42. \tag{9}$$

The predicted class label is then

$$\hat{y} = \arg\max_{j} \; p(j = 0|\mathbf{x}), \; p(j = 1|\mathbf{x}) \quad = 0. \tag{10}$$

7.4 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[2].

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, D_i
5: Train base classifier h_i on D_i
6: $\hat{y} = mode\{h_1(\mathbf{x}), ..., h_n(\mathbf{x})\}$

Figure 4: The bagging algorithm.

Figure 5: Illustration of bootstrap sampling

- 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

$$P(\text{not chosen}) = \left(1 - \frac{1}{n}\right)^{n}, \tag{11}$$

which is asymptotically equivalent to

$$\frac{1}{e} \approx 0.368 \quad \text{as} \quad n \to \infty. \tag{12}$$

[2]Leo Breiman. "Bagging predictors". In: *Machine learning* 24.2 (1996), pp. 123–140.

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

$$P(\text{chosen}) = 1 - \left(1 - \frac{1}{n}\right)^n \approx 0.632. \quad (13)$$

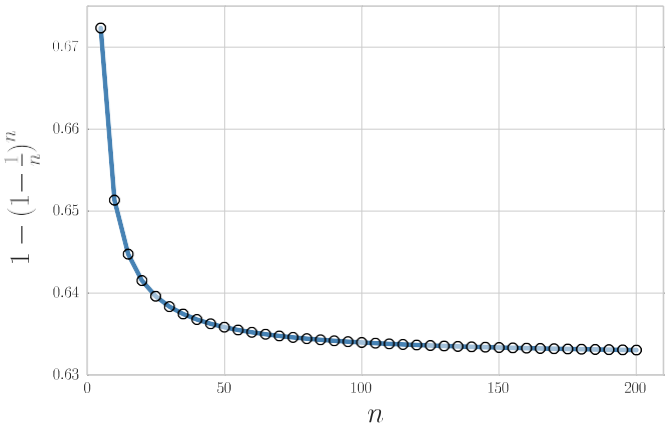

Figure 6: Proportion of unique training examples in a bootstrap sample.

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 7: Illustration of bootstrapping in the context of bagging.

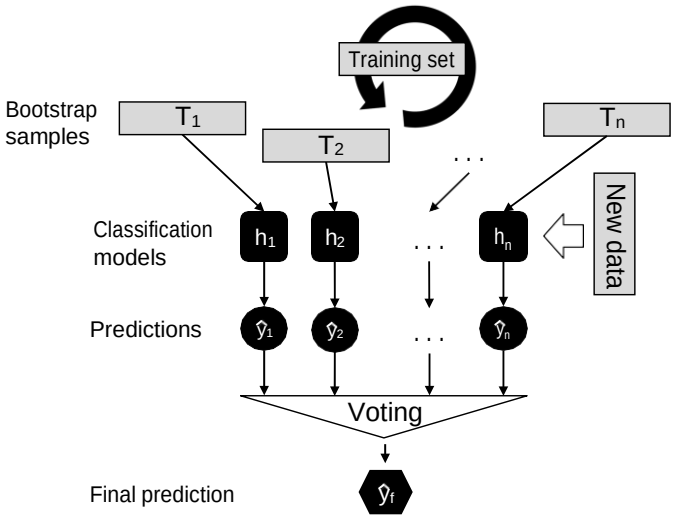

Figure 8: The concept of bagging. Here, assume n different classifiers, $\{h_1, h_2, ..., h_m\}$ where $h_i(\mathbf{x}) = \hat{y}_i$.

7.5 Bias and Variance Intuition

- "Bias and variance" will be discussed in more detail in the next lecture, where we will decompose loss functions into their variance and bias components and see how it relates to overfitting and underfitting.

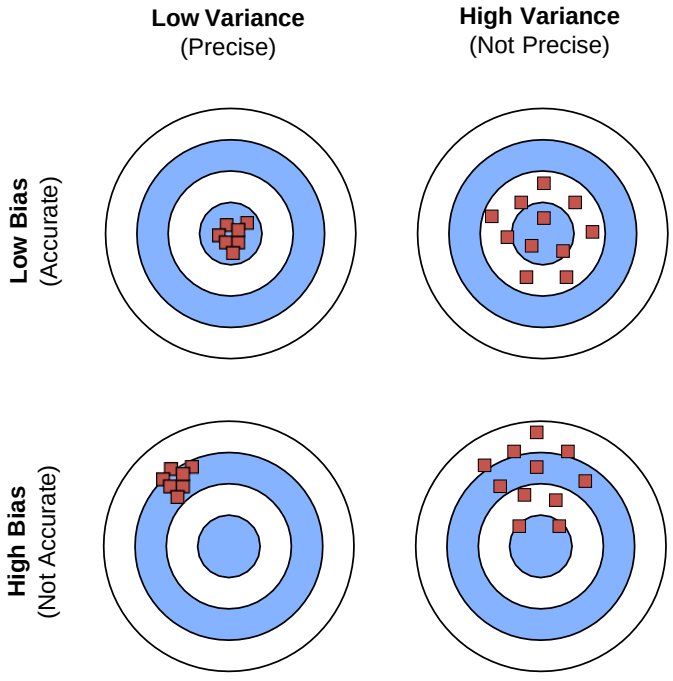

Figure 9: Bias and variance intuition.

- One can say that individual, unpruned decision tree "have high variance" (in this context, the individual decision trees "tend to overfit"); a bagging model has a lower

variance than the individual trees and is less prone to overfitting – again, bias and variance decomposition will be discussed in more detail next lecture.

7.6 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 leaner 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[3].
- If you are familiar with gradient-based optimization and interested in gradient boosting, I recommend reading Friedman's work[4] and the more recent paper on XGBoost[5], which is essentially a computationally efficient implementation of the original gradient boost algorithm.

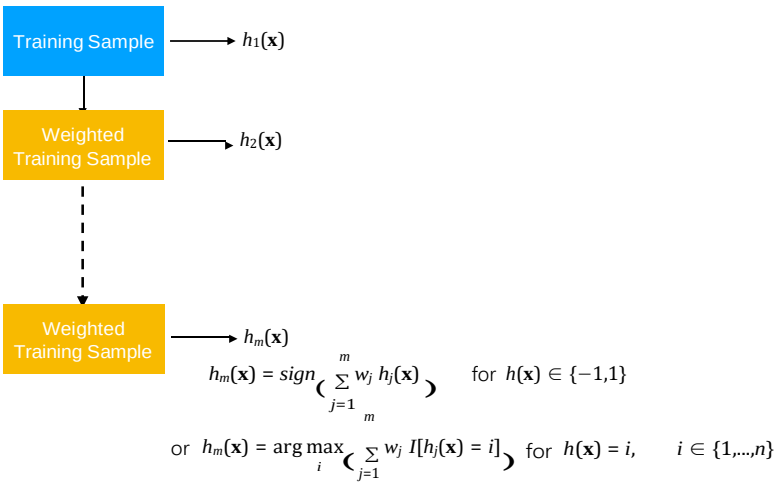

Figure 10: A general outline of the boosting procedure for *n* iterations.

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

- Initialize a weight vector with uniform weights

[3]Yoav Freund and Robert E Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[4]Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232.

[5]Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM. 2016, pp. 785–794.

- 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

7.6.1 AdaBoost (Adaptive Boosting)

Algorithm 1 AdaBoost

Initialize k: the number of AdaBoost rounds
Initialize $\mathcal{D}$: the training dataset, $\mathcal{D} = \{\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, ..., \mathbf{x}^{[n]}, y^{[n]} \rangle\}$
Initialize $w_1(i) = 1/n, \quad i = 1, ..., n, \ \mathbf{w}_1 \in \mathbb{R}^n$

for r=1 to k **do**
  For all $i : \mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$ [normalize weights]
  $h_r := FitWeakLearner(\mathcal{D}, \mathbf{w}_r)$
  $\epsilon_r := \sum_i w_r(i) \ \mathbf{1}(h_r(i) \neq y_i)$ [compute error]
  if $\epsilon_r > 1/2$ then stop
  $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r)/\epsilon_r]$ [small if error is large and vice versa]
  $w_{r+1}(i) := w_r(i) \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$

Predict: $h_k(\mathbf{x}) = \arg\max_j \sum_r \alpha_r \mathbf{1}[h_r(\mathbf{x}) = j]$

Figure 11: AdaBoost algorithm.

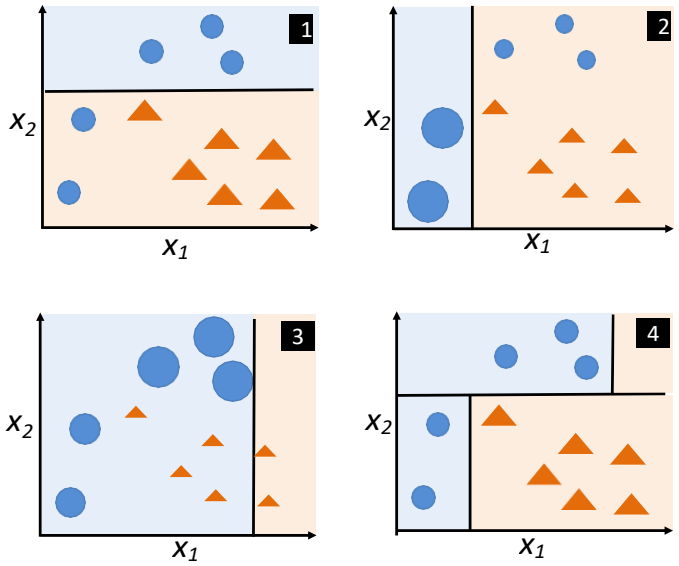

Figure 12: Illustration of the AdaBoost algorithms for three iterations on a toy dataset. The size of the symbols (circles shall represent training examples from one class, and triangles shall represent training examples from another class) is proportional to the weighting of the training examples at each round. The 4th subpanel shows a combination/ensemble of the hypotheses from subpanels 1-3.

7.7 Random Forests

7.7.1 Overview

- 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 = $\log_2 m + 1$.

7.7.2 Does random forest select a subset of features for every tree or every node?

Earlier random decision forests by Tin Kam Ho[6] 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[7]:"

> "... random forest with random features is formed by selecting at random, at each node, a small group of input variables to split on."

7.7.3 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

$$\text{PE} \leq \frac{\bar{\rho} \cdot (1 - s^2)}{s^2}, \tag{14}$$

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. We

[6] Tin Kam Ho. "Random decision forests". In: *Document analysis and recognition, 1995., proceedings of the third international conference on*. Vol. 1. IEEE. 1995, pp. 278–282.

[7] Leo Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

do not need to get into the details of how $\bar{p}$ and s are calculated to get an intuition for their relationship. I.e., 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.

7.7.4 Feature Importance via Random Forests

While random forests are naturally less interpretable than individual decision trees, where we can trace a decision via a rule sets, it is possible (and common) to compute the so-called "feature importance" of the inputs – that means, we can infer how important a feature is for the overall prediction. However, this is a topic that will be discussed later in the "Feature Selection" lecture.

7.7.5 Extremely Randomized Trees (ExtraTrees)

- A few years after random forests were developed, an even "more random" procedure was developed called *Extremely Randomized Trees*[8].
- Compared to regular random forests, the ExtraTrees algorithm selects a random feature at each decision tree nodes for splitting; hence, it is very fast because there is no information gain computation and feature comparison step.
- Intuitively, one might say that ExtraTrees have another "random component" (compared to random forests) to further reduce the correlation among trees – however, it might decrease the strength of the individual trees (if you think back of the generalization error bound discussed in the previous section on random forests).

7.8 Stacking

7.8.1 Overview

- Stacking[9] is a special case of ensembling where we combine an ensemble of models through a so-called meta-classifier.
- In general, in stacking, we have "base learners" that learn from the initial training set, and the resulting models then make predictions that serve as input features to a "meta-learner."

[8] **geurts2006extremel**.
[9] **wolpert1992stacked**.

7.8.2 Naive Stacking

Algorithm 1 "Naive" Stacking

Input: Training set $D = \{ \langle x^{[1]}, y^{[1]} \rangle , ..., \langle x^{[n]}, y^{[n]} \rangle \}$
Output: Ensemble classifier h_E

Step 1: Learn base-classifiers
for $t \leftarrow 1$ to T do
    Fit base model h_t on D
Step 2: construct new dataset D' from D
for $i \leftarrow 1$ to n do
    add $\langle x'^{[i]}, y^{[i]} \rangle$ to new dataset, where $x'^{[i]} = \{h_1(x^{[i]}), ..., h_T(x^{[i]})\}$
Step 3: learn meta-classifier h_E
return $h_E(D')$

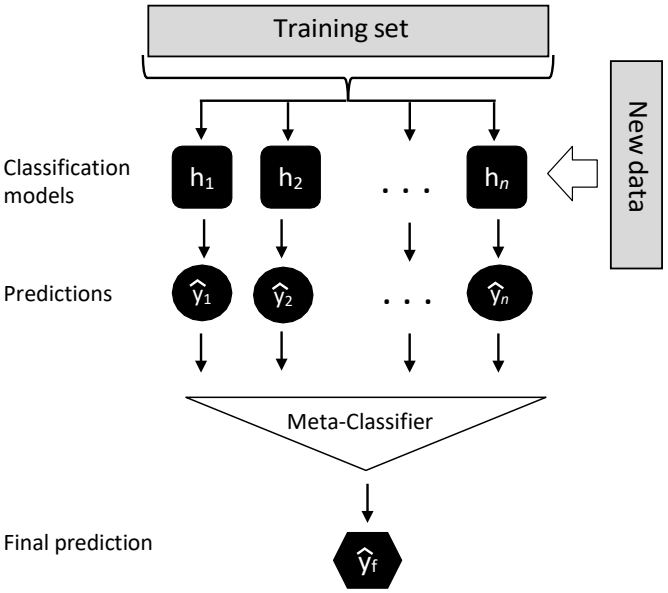

Figure 13: The basic concept of stacking is illustrated below, analogous to the voting classifier at the beginning of this lecture. Note that here, in contrast to majority voting, we have a meta-classifier that takes the predictions of the models produced by the base learners ($h_1 ... h_n$) as inputs.

The problem with the naive stacking algorithm outlined above is that it has a high tendency to suffer from extensive overfitting. The reason for a potentially high degree of overfitting is that if the base learners overfit, then the meta-classifier heavily relies on these predictions made by the base-classifiers. A better alternative would be to use stacking with k -fold cross-validation or leave-one-out cross-validation.

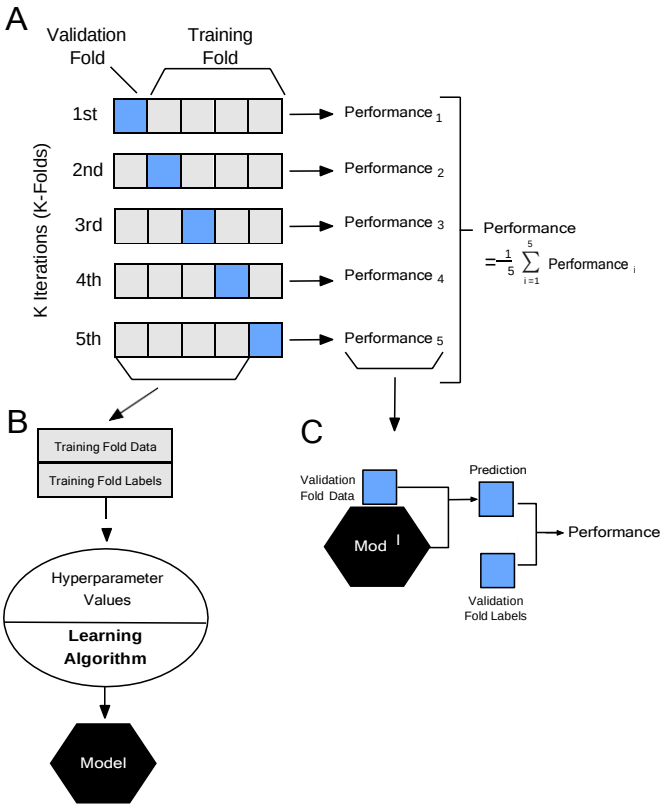

Figure 14: Illustratio of k-fold cross-validation

Figure 15: Illustration of leave-one-out cross-validation, which is a special case of k-fold cross-validation, where $k = n$ (where n is the number of examples in the training set).

7.8.3 Stacking with Cross-Validation

- The use of cross-validation (or leave-one-out cross-validation) is highly recommended for performing stacking, to avoid overfitting.

Algorithm 1 Stacking with cross-validation

Input: Training set $D = \{ \langle x^{[1]}, y^{[1]} \rangle , ..., \langle x^{[n]}, y^{[n]} \rangle \}$
Output: Ensemble classifier h_E

Step 1: Learn base-classifiers
  Construct new dataset $D' = \{\}$
  Randomly split D into k equal-size subsets: $D = \{D_1, ..., D_k\}$
  for $j \leftarrow 1$ to k do
    for $t \leftarrow 1$ to T do
      Fit base model h_t on $D \setminus D_k$
    for $i \leftarrow 1$ to $n \in |D \setminus D_k|$ do
      Add $\langle x'^{[i]}, y^{[i]} \rangle$ to new dataset D', where $x'^{[i]} = \{h_1(x^{[i]}), ..., h_T(x^{[i]})\}$
Step 3: learn meta-classifier h_E
return $h_E(D')$

Figure 16: stacking-algo-cv

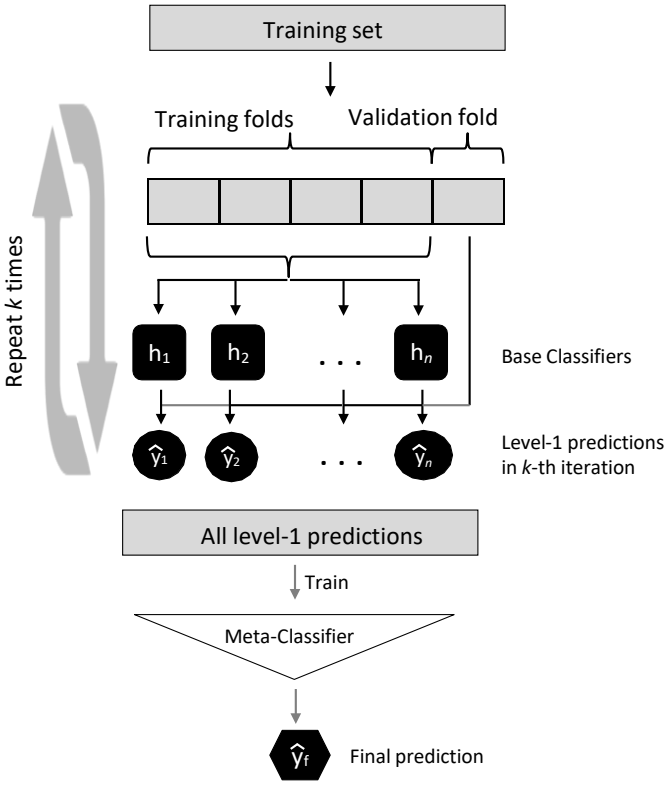

Figure 17: Illustration of stacking with cross-validation.

7.9 Resources

7.9.1 Assigned Reading

- Python Machine Learning, 2nd Ed., Chapter 7

7.9.2 Further Reading

Listed below are optional reading materials for students interested in more in-depth coverage of the ensemble methods we discussed (not required for homework or the exam).

- Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123-140.
- Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, *5*(2), 241-259.
- Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5-32.
- Freund, Y., Schapire, R., & Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, *14*(771-780), 1612.
- Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189-1232.
- Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM sigkdd international conference on knowledge discovery and data mining* (pp. 785-794). ACM.