CS1138

Machine Learning

Lecture : Ensemble Methods

(Sebastian Raschka and Kilian Weinberger)

Arpan Gupta

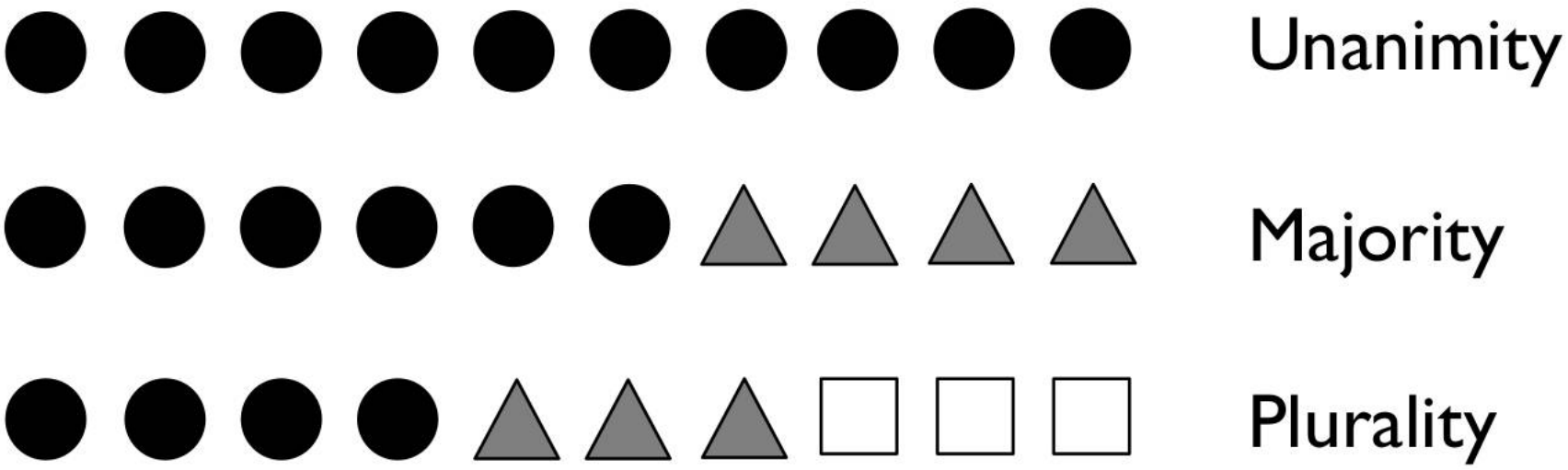
Overview

- Introduction
- Majority Voting
- Bagging – Bootstrap Aggregating
- Random Forest
- Boosting

Ensemble Methods: Introduction

- Most widely used among non-DL models.
- 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.
- Methods to improve the performance of weak learners.
- Weak learners (eg. Classification trees) don’t perform that well. They are slightly better than random classification.

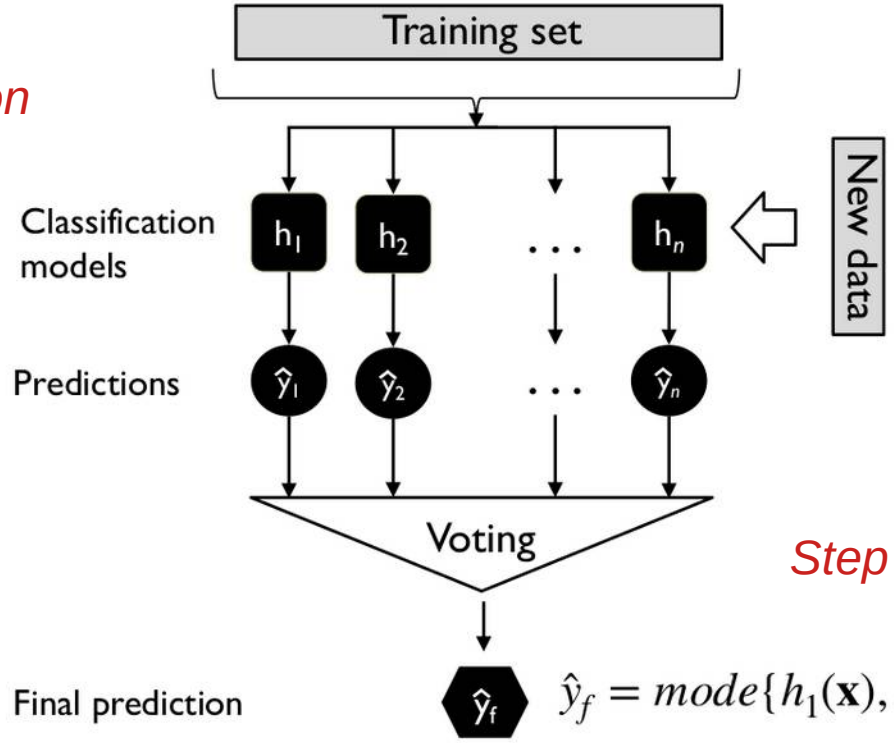
Majority Voting



Majority is >= 50%
Relative Majority is same as Plurality (most frequent label)

Majority Voting Classifier

Step 1: Given n classifiers, trained on the training set.



Step 2: Using n classifiers, generate predictions on new data.

Step 3: Voting

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

where $h_i(\mathbf{x}) = \hat{y}_i$

Why Majority Voting?

- assume n independent classifiers with a base error rate ϵ
- here, independent means that the errors are uncorrelated
- assume a binary classification task
- assume the error rate is better than random guessing (i.e., lower than 0.5 for binary classification)

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

Why Majority Voting?

- Probability that we make a wrong prediction via the ensemble if k classifiers predict the same class label:
- More than k classifiers have to make wrong prediction

$$P(k) = \binom{n}{k} \epsilon^k (1-\epsilon)^{n-k} \qquad k > \lceil n/2 \rceil$$

(Probability mass func. of a binomial distr.)

Why Majority Voting?

The probability that we make a wrong prediction via the ensemble if k classifiers predict the same class label

$$P(k) = \binom{n}{k} \epsilon^k (1-\epsilon)^{n-k} \qquad k > \lceil n/2 \rceil$$

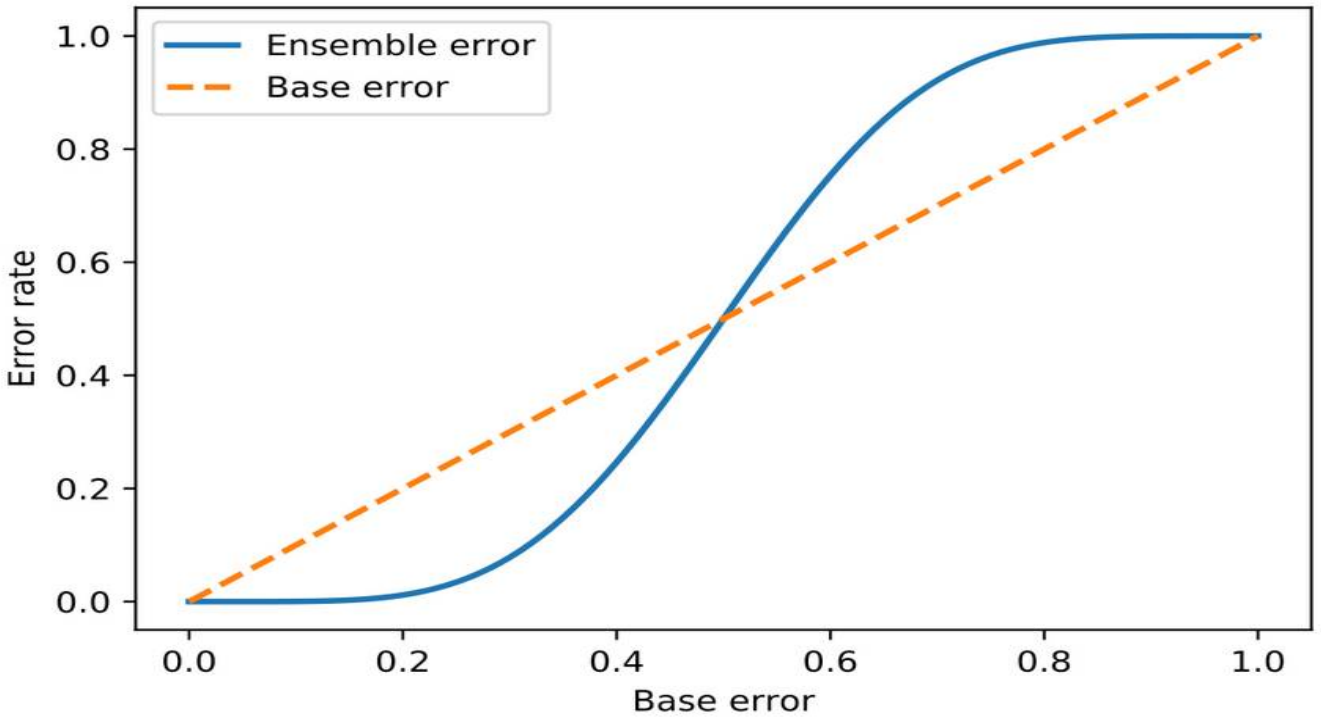
Ensemble error:

$$\epsilon_{ens} = \sum_{k}^{n} \binom{n}{k} \epsilon^k (1-\epsilon)^{n-k}$$ (cumulative prob. distribution)

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

Why Majority Voting?

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



Bias Variance Decomposition

$$\text{Loss} = \text{Bias} + \text{Variance} + \text{Noise}$$

Suppose, we have a regression task. The total loss of a model may be because of 3 components : bias, variance and noise.

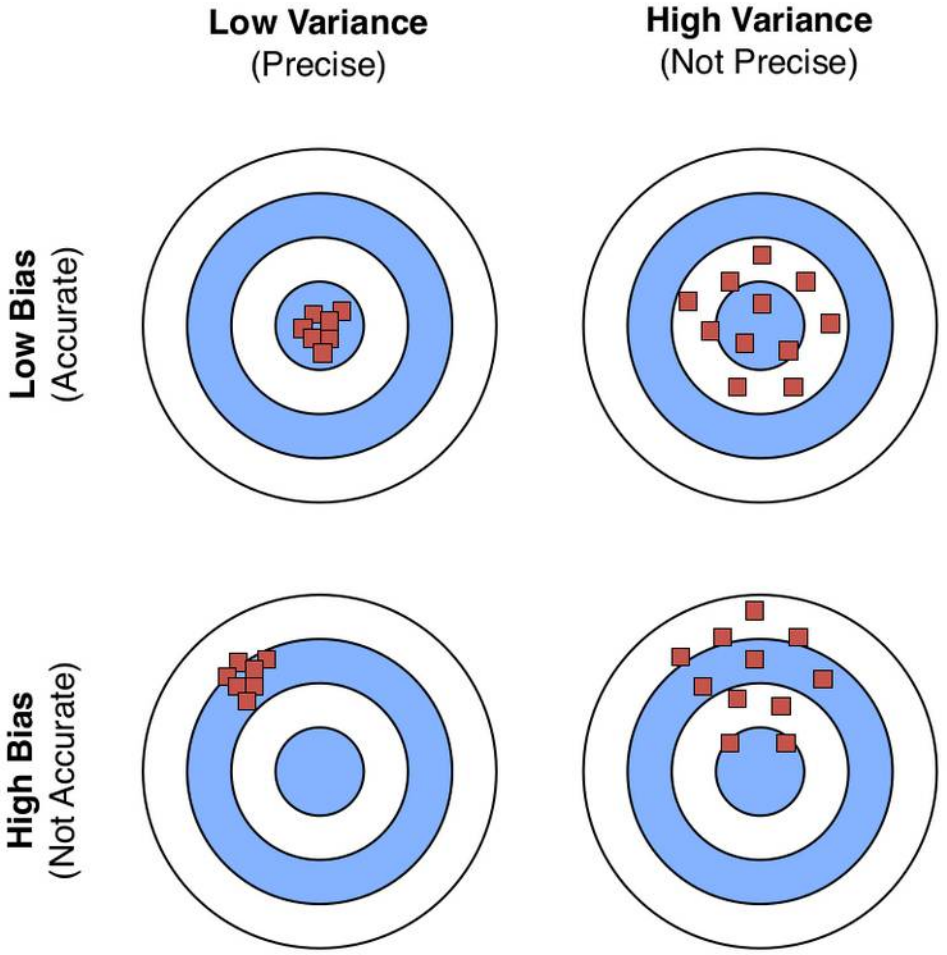
Bias Variance Decomposition

- The decomposition of the expected test error is:

$$\underbrace{E_{\mathbf{x},y,D}\left[(h_D(\mathbf{x})-y)^2\right]}_{\text{Expected Test Error}} = \underbrace{E_{\mathbf{x},D}\left[\left(h_D(\mathbf{x})-\bar{h}(\mathbf{x})\right)^2\right]}_{\text{Variance}} + \underbrace{E_{\mathbf{x},y}\left[(\bar{y}(\mathbf{x})-y)^2\right]}_{\text{Noise}} + \underbrace{E_{\mathbf{x}}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)^2\right]}_{\text{Bias}^2}$$

- **Variance:** Captures how much your classifier changes if you train on a different training set. How "over-specialized"is your classifier to a particular training set (overfitting)? If we have the best possible model for our training data, how far off are we from the average classifier?
- **Bias:** What is the inherent error that you obtain from your classifier even with infinite training data? This is due to your classifier being "biased" to a particular kind of solution (e.g. linear classifier). In other words, bias is inherent to your model.
- **Noise:** How big is the data-intrinsic noise? This error measures ambiguity due to your data distribution and feature representation. You can never beat this, it is an aspect of the data.

Bias Variance Intuition



Bagging: Bootstrap Aggregating

- Proposed by Leo Brieman (1996)
- Decision trees have bias and variance problem and finding optimal place is hard.
- A single DT, when grown fully, will result in “pure” leaf nodes or nodes with equal +ve and -ve samples with same features.
- This results in overfitting on the training set.
- But they are very fast, adaptive and robust.
- **Idea:** Grow a big tree and fix the high variance problem by learning multiple trees. This we do by bootstrapping.

Bagging

- Bootstrap sampling: get different splits / subsets of the data.
- Aggregating: majority voting or averaging.

Bagging

- Let P be a data generating process, or distribution. Also termed as the population to which we do not have access.
- We have D ~ P (D sampled from P), where D has m i.i.d. points taken from distribution P.
- Let n be the number of classifiers that we learn.
- Take D_1, ..., D_n different datasets and train classifiers on each of these n datasets.
- $$h(x)=\frac{1}{n}\sum_{i=1}^{n} h_{D_i}(x)$$

Bagging

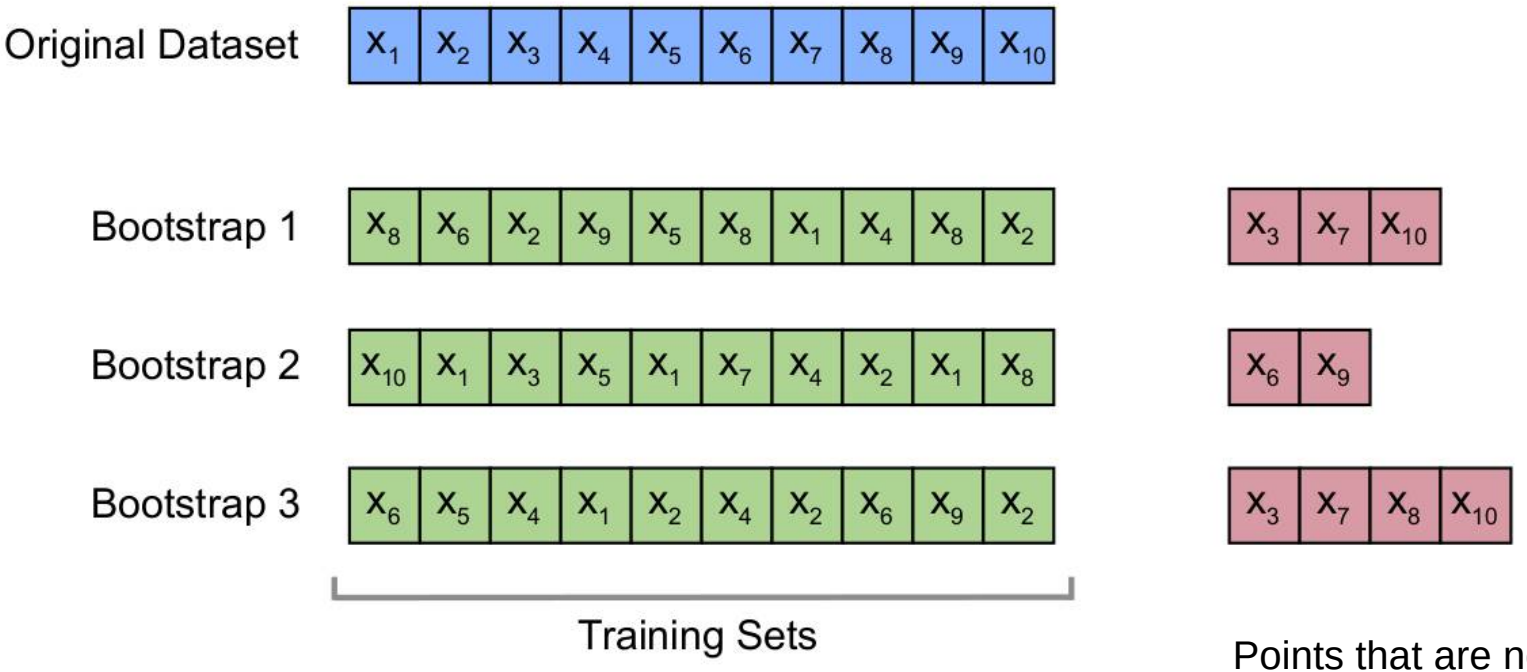
- If n gets large (n --> infinity), h(x) --> $\bar{h}(x)$. Weak law of large numbers.
- Therefore, the variance decreases as n gets large.
- But we actually don't have n different datasets from the same population. We have a single dataset D sampled from population P.
- Therefore, we sample D_1, ..., D_n from D itself, by having each D_i ~ D with replacement and having same number of samples m as that in D. This is called **Bootstrap sampling or Bootstrapping**.
- Note that D_1, ..., D_n cannot be partitions of D as each will have less number of samples and will make variance worse.
- A major advantage is that this is easy to parallelize.

Bagging – Bootstrap Aggregating

Algorithm 1 Bagging

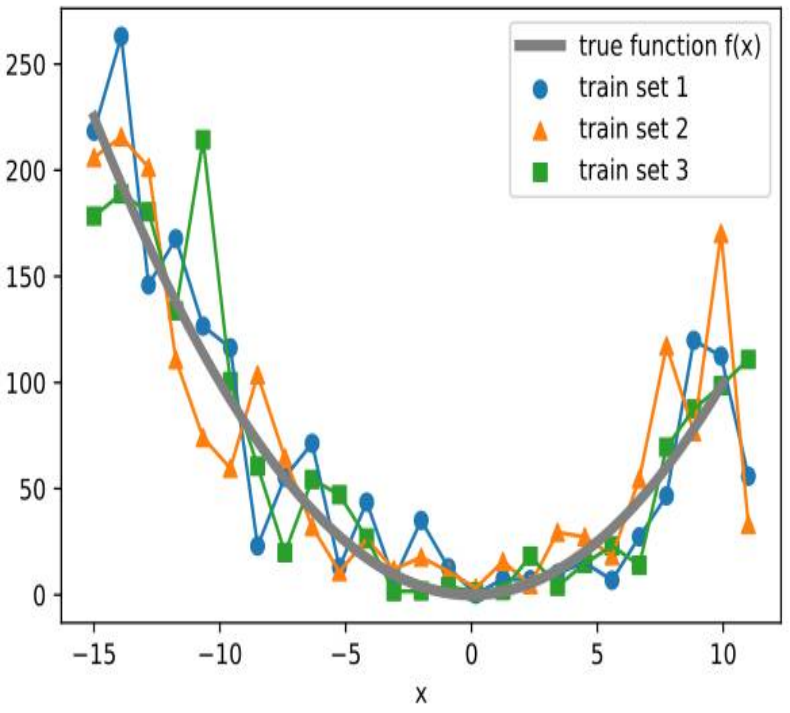
1: Let n be the number of bootstrap samples
2:
3: **for** i=1 to n **do**
4: Draw bootstrap sample of size m, $\mathcal{D}_i$
5: Train base classifier h_i on $\mathcal{D}_i$
6: $\hat{y} = mode\{h_1(\mathbf{x}), ..., h_n(\mathbf{x})\}$

Bootstrap Sampling

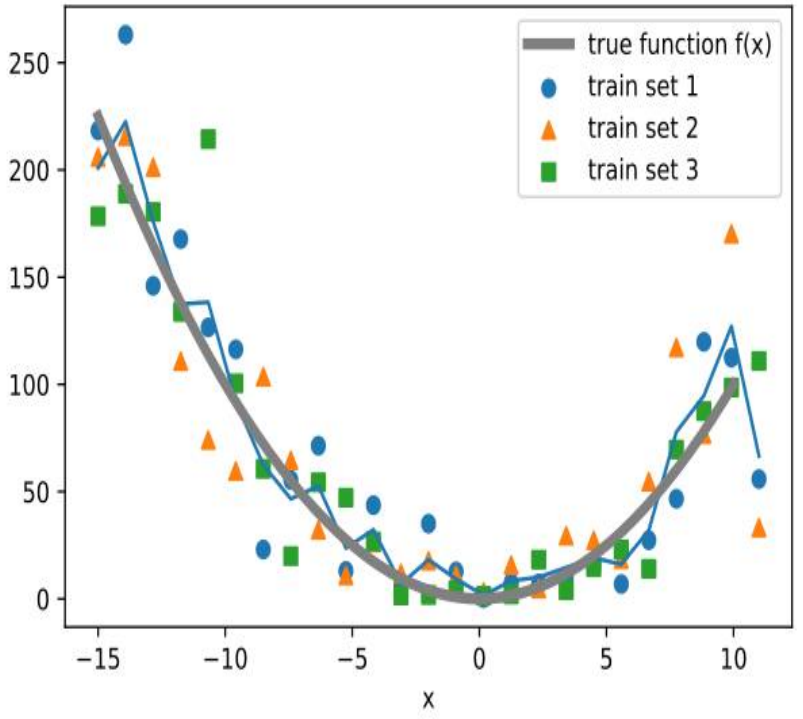


Points that are not sampled in a particular bootstrap

Bagging



3 different classifiers on bootstrap sampled data.



Comparing bagged classifier with True function.

Random Forest – An instance of Bagging

- One of the most popular instances of Bagging.
- Also invented by Leo Brieman.
- RFs are bagged decision trees.
- For n bootstrap samples of dataset D, build n classifiers (decision trees).
- In DTs, we try out every single dimension d, and for every single dimension, we try out every single split (between any 2 consecutive points).
- **Modification in RF**, instead of trying out on every single dimension, we randomly subsample **k** dimensions, where **k < d**.
- Therefore, we restrict our search in **k** dimensions only.

Random Forest

- By restricting our search for finding the best split on only k dimensions, we ensure that the individual classifiers h_1, ..., h_n are each different from one another.
- Since, each h_1, ..., h_n are very different, they will make very different errors at test time. And after averaging, the errors average out.
- Note that for every single split, we sample k features out of the d features.
- Now we have an additional hyperparameter k, and have to decide how to set k.
- Set **k = ceil (√d)** [Check proof as homework]

Random Forest

- The Random Forest is one of the best, most popular and easiest to use out-of-the-box classifier.
- There are two reasons for this:
  - The RF only has two hyper-parameters, **n** and **k**. It is extremely insensitive to both of these. A good choice for k is **k = ceil(√ d)** (where d denotes the number of features). You can set **n** as large as you can afford.
  - Decision trees do not require a lot of preprocessing. For example, the features can be of different scale, magnitude, or slope. This can be highly advantageous in scenarios with heterogeneous data, for example the medical settings where features could be things like blood pressure, age, gender, ..., each of which is recorded in completely different units.

Random Forest

- Another advantage is **Out of Bag (OOB)** Error. Means we can estimate the test error directly on the training dataset.
- We compute loss for each point in the dataset, but some points will not be present in some of the datasets D_1, ..., D_n.
- D_j contains (in expectation) around 60% of the data points from D.
- **Idea:** Compute error for a point on the models trained on D_j that do not have that point.

Random Forest

- **Out of Bag (OOB)** Error may be calculated by considering average error that x_i makes on classifiers h_j that are not trained on x_i. L is a loss function like squared loss, n are the number of classifiers, m are the number of samples, z_i are the number of classifiers that are not trained on x_i.

$$E_{OOB} = \frac{1}{n} \sum_{i=1}^{m} \frac{1}{z_i} \sum_{j\, s.t.\, (x_i, y_i) \notin D_j} L\left(h_j\left(x_i\right), y_i\right)$$

Average loss for a sample x_i on h_j's

$$z_i = \sum_{j\, s.t.\, (x_i, y_i) \notin D_j} 1$$

Boosting

- In bagging, $$h(x)=\sum_{j=1}^{n}\frac{1}{n}h_j(x)$$
- In boosting, we create an ensemble classifier (n are number of classifiers)

$$H(x)=\sum_{j=1}^{n}\alpha_j h_j(x)$$

- We can compute the loss of the classifier as average over errors for each sample x_i here m are no. Of samples in dataset. L is convex and differentiable.

$$L(H)=\frac{1}{m}\sum_{j=1}^{m}L(H(x_i),y_i)$$

Boosting

- **Scenario:** Hypothesis class H, whose set of classifiers has large bias and the training error is high (e.g. CART trees with very limited depth.)
- **Famous question: Can weak learners (h_j) be combined to generate a strong learner (H) with low bias?** (weak learners cannot bring your training errors down to zero, while strong learners can)
- **Answer: Yes**

Boosting

- This ensemble classifier is built in an iterative fashion. In iteration t we add the classifier $\alpha_j h_j$ to the ensemble
- Suppose H is the sum of the classifiers ($\alpha_j h_j$), and we have completed some iterations. We would like to add the next best classifier to H, so which classifier to add.
- The process of constructing such an ensemble in a stage-wise fashion is very similar to gradient descent.
- However, instead of updating the model parameters in each iteration, we add functions to our ensemble.

Boosting

- Now in iteration t + 1 we want to add one more weak learner h_{t+1} to the ensemble. To this end we search for the weak learner that minimizes the loss the most,

$$h_{t+1} = \mathrm{argmin}_{h \in \mathbb{H}} \ell(H_t + \alpha h_t).$$

- Once h_{t+1} has been found, we add it to our ensemble,

  i.e. $H_{t+1} = H_t + \alpha h$

Boosting

- How can we find such $h \in H$?
- **Answer:** Use gradient descent in function space. In function space, inner product can be defined as

  $$\mathbf{< h, g > = \int h(x)\, g(x)\, dx}$$

- Since we only have training set, we define

  $$\mathbf{< h, g > = \sum_{i=1}^{n} h(x_i)\, g(x_i)\, .}$$

Boosting

- ADABoost (Adaptive Boosting), and Gradient Boosted Trees (GBTs) are instances of boosting method.

References

- STAT 451: Intro to Machine Learning, Fall 2020 Sebastian Raschka
  - http://stat.wisc.edu/~sraschka/teaching/stat451-fs2020/
- Cornell CS4780 Lecture 29: Kilian Weinberger
- https://www.cs.cornell.edu/courses/cs4780/2021fa/lectures/lecturenote18.html

End of Lecture