Tree and Random Forest

Statistical Machine Learning (ENGG*6600*08)

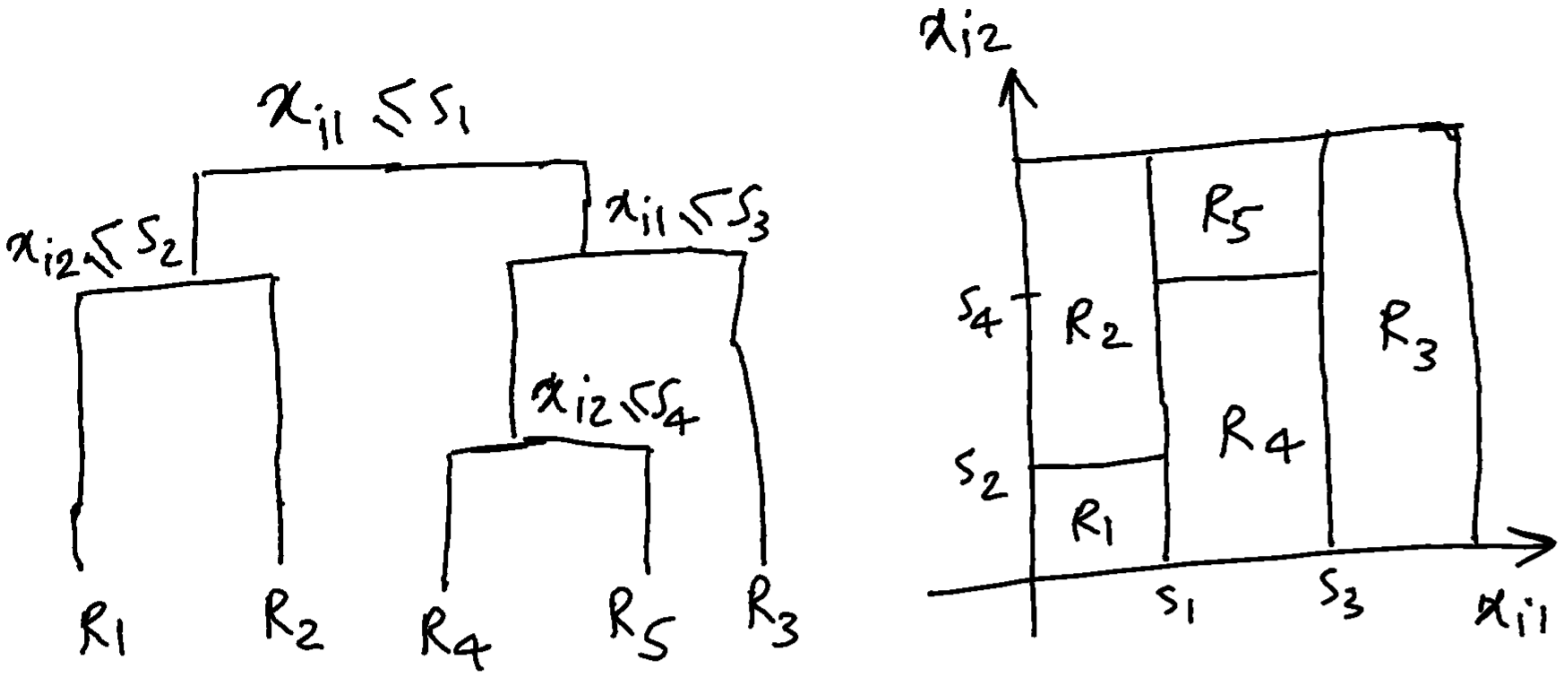
School of Engineering,
University of Guelph, ON, Canada

Course Instructor: Benyamin Ghojogh
Fall 2023

Tree

Tree

- Tree is used for classification and regression.
- It is easy to implement and is intuitive and how it works is explainable.
- It is common to use in bio-statistics and health sciences.
- Assume we have a dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ is the data point and $y_i \in \mathbb{R}$ is the scalar label.
- Let $\boldsymbol{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}]^\top$ where x_{ij} denotes the j-th dimension (feature) of $\boldsymbol{x}_i$.
- A tree partitions the space into regions. It splits over one of the features at every split. An example binary tree in 2D is as follows:



Split in Tree

- The initial region is the whole space.
- A binary tree splits a region into two disjointed sub-regions whose union is the parent region:

$$R_1(j, s) = \{\boldsymbol{x}_i \mid x_{ij} \leq s\}, \tag{1}$$
$$R_2(j, s) = \{\boldsymbol{x}_i \mid x_{ij} > s\}, \tag{2}$$
$$R = R_1(j, s) \cup R_2(j, s), \tag{3}$$
$$\varnothing = R_1(j, s) \cap R_2(j, s), \tag{4}$$

  where s is the threshold of split on the feature $j \in \{1, \ldots, d\}$.
- There are some questions:
  - Over which feature j do we split?
  - What is the value of the threshold s?
  - What constant value (estimated label) should we assign to every region?

Local Cost Function

- There are some questions:
  - Over which feature j do we split?
  - What is the value of the threshold s?
  - What constant value (estimated label) should we assign to every region?
- The **local cost function** of the tree is defined as:

$$\min_{j,s} \quad \Big(\sum_{\boldsymbol{x}_i \in R_1} (y_i - \bar{c}_1)^2 + \sum_{\boldsymbol{x}_i \in R_2} (y_i - \bar{c}_2)^2 \Big), \tag{5}$$

where R_1 and R_2 are the two sub-regions that the parent region is split to, j is the feature to split over, s is the value of the threshold, and $\bar{c}_1$ and $\bar{c}_2$ are the mean labels of the sub-regions:

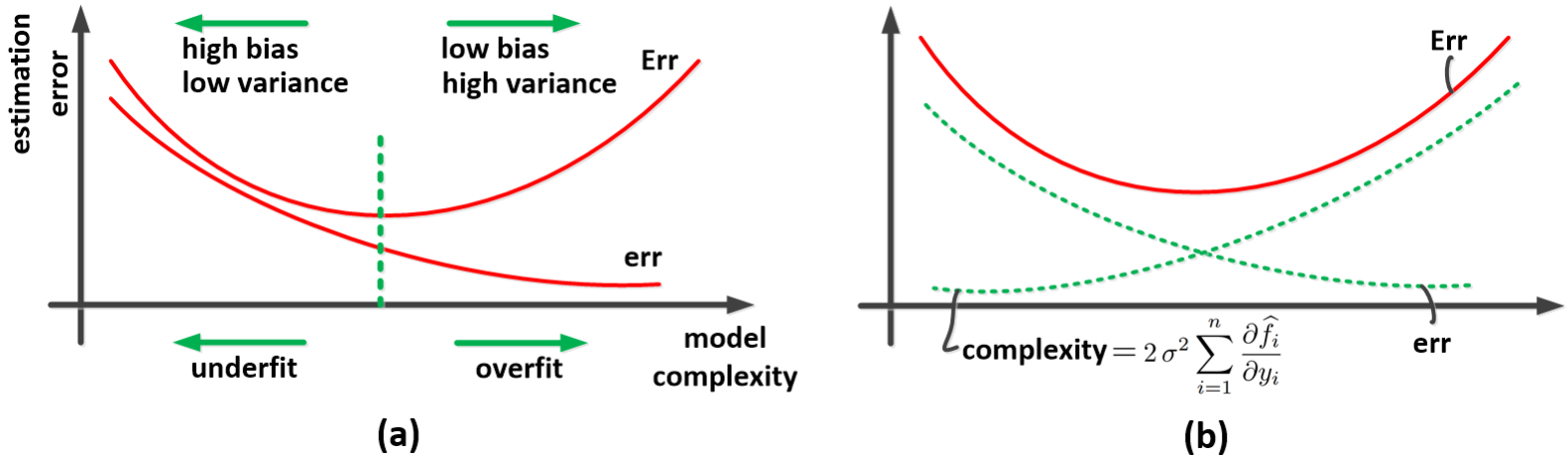
$$\bar{c}_1 := \text{average}(\{y_i | \boldsymbol{x}_i \in R_1\}), \tag{6}$$

$$\bar{c}_2 := \text{average}(\{y_i | \boldsymbol{x}_i \in R_2\}). \tag{7}$$

- Therefore, we find the best j and s so that we have the smallest mean squared error of the estimation of the labels in the regions.

Overfitting and Underfitting

- If the depth of the tree is small, the number of children regions is small. So, it is **underfitting**.
- If the depth of the tree is too large, the number of children regions is too large. In the worst case, every region can contain one single data point. So, it is **overfitting**.
- Therefore, the number of children nodes in a tree T is a good measure of overfitting and complexity. Let this number be denoted by $|T|$.
- Question: how large should a tree be?
- Solution 1: Keep splitting the tree until the validation error **Err** starts going up while the training error **err** is going down. This idea is similar to the idea of early stopping in neural networks.



Global Error

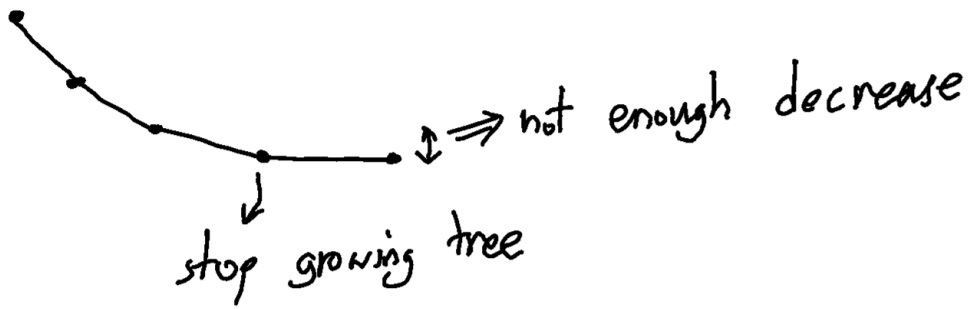
- Solution 2: consider the **global error** of the tree:

$$\sum_{m=1}^{|T|} n_m Q_m(T), \tag{8}$$

where summation is over the children nodes, $n_m := \sum_{i=1}^{n} \mathbb{I}(\boldsymbol{x}_i \in R_m)$ is the number of data points in child region R_m (note that $\mathbb{I}(.)$ is the indicator function), and Q_m is the average error of the child region R_m defined as:

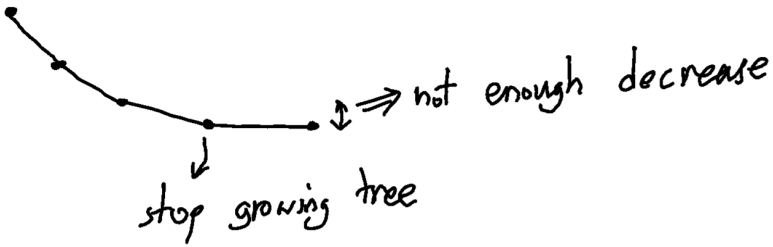
$$Q_m := \frac{1}{n_m} \sum_{\boldsymbol{x}_i \in R_m} (y_i - \bar{c}_m)^2. \tag{9}$$

- We can calculate the global error of the tree after every split and if there is not a sufficient decrease in the global error, we can stop.

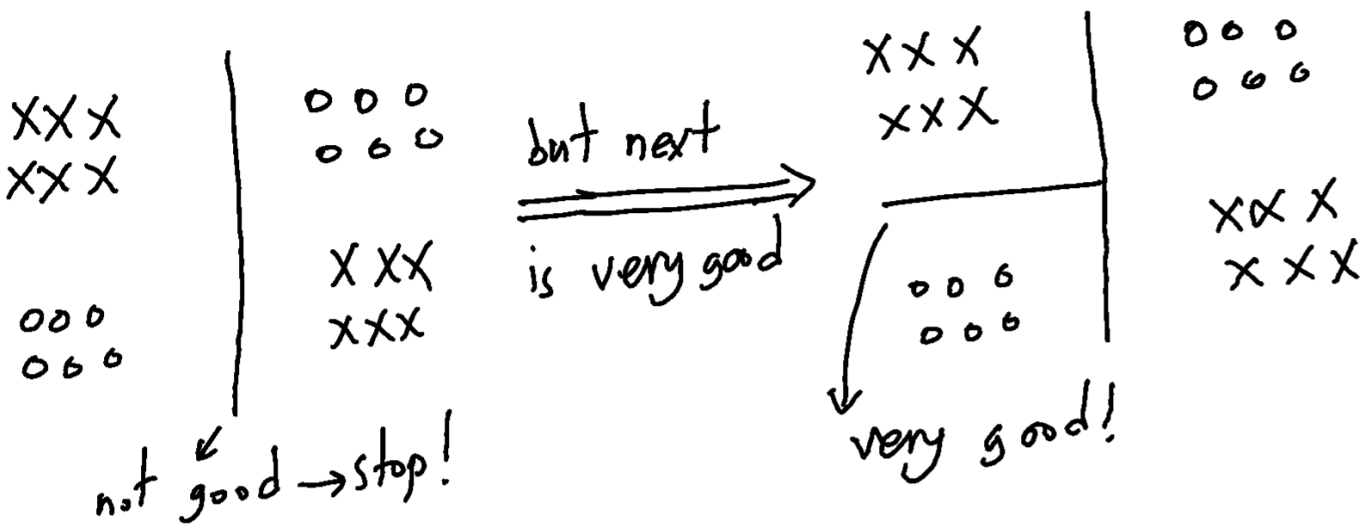


Global Error

- We can calculate the global error of the tree after every split and if there is not a sufficient decrease in the global error, we can stop.



- However, this solution is not good because what if the global error in the next split suddenly decreases drastically! This is like a chess game where we should consider the long-term reward/cost and not only the short-term reward/cost.



Regularization and Pruning

- Solution 3: Another solution is to use **regularization**.
- We **grow** the tree by the local error optimization for every split, as done in Eq. (5):

$$\min_{j,s} \quad \Big(\sum_{\boldsymbol{x}_i \in R_1} (y_i - \bar{c}_1)^2 + \sum_{\boldsymbol{x}_i \in R_2} (y_i - \bar{c}_2)^2 \Big),$$

- and then we **prune** it by the **regularized global error** of the tree:

$$\sum_{m=1}^{|T|} n_m Q_m(T) + \alpha |T|, \tag{10}$$

where $\alpha > 0$ is the regularization parameter and Q_m is the average error of the child region R_m defined in Eq. (9):

$$Q_m := \frac{1}{n_m} \sum_{\boldsymbol{x}_i \in R_m} (y_i - \bar{c}_m)^2.$$

Regularization and Pruning

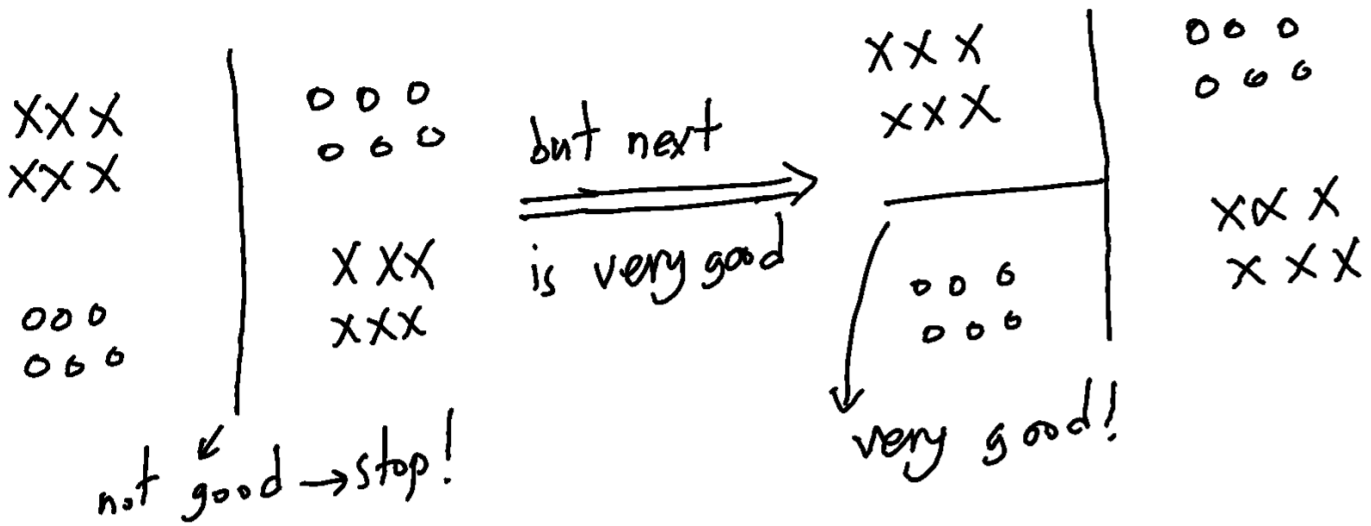
- Pruning is the opposite of splitting regions. It is collapsing or merging child regions into the parent region so the parent region becomes a new child region.
- For pruning, we merge the child tree (merge its children) which has the smallest regularized global error, defined in Eq. (10).

$$\min_{T} \quad \sum_{m=1}^{|T|} n_m Q_m(T) + \alpha |T|, \tag{11}$$

  where the optimization variable is the tree because every prune makes a new tree.
- This optimization is because that child tree plays the smallest role in decrease of the regularized global cost of the tree.

Regularization and Pruning

- **Question**: why do we grow the tree until long and then prune? Why don't we just stop growing at the desired depth?
  - This is because we might find interesting splits later in the tree and then prune from the less important children trees to keep that important split. Note that it is like playing chess and we should think to long-term analysis.



Algorithm of regression tree in summary

- Algorithm of regression tree in summary:
  1. Grow the tree iteratively (recursively) until some long enough depth using:

$$\min_{j,s} \quad \Big(\sum_{x_i \in R_1} (y_i - \bar{c}_1)^2 + \sum_{x_i \in R_2} (y_i - \bar{c}_2)^2 \Big).$$

  2. Prune the tree iteratively until its depth becomes the desired depth:

$$\min_{T} \quad \sum_{m=1}^{|T|} n_m Q_m(T) + \alpha |T|,$$

where:

$$Q_m := \frac{1}{n_m} \sum_{x_i \in R_m} (y_i - \bar{c}_m)^2.$$

Decision Tree for Classification

Decision Tree for Classification

- The tree can be used for classification. It is called **decision tree** because it decides the class label of the data points in its regions.
- The cost of the decision tree should be a loss for misclassification.
- Assume there are c classes and ℓ denotes the label of the the ℓ-th class. We define:

$$p_{m\ell} = \frac{1}{n_m} \sum_{\boldsymbol{x}_i \in R_m} \mathbb{I}(y_i = \ell) = \frac{\sum_{i=1}^{n} \mathbb{I}(\boldsymbol{x}_i \in R_m, y_i = \ell)}{\sum_{i=1}^{n} \mathbb{I}(\boldsymbol{x}_i \in R_m)}, \tag{12}$$

  where $\mathbb{I}(.)$ is the indicator function and $n_m := \sum_{i=1}^{n} \mathbb{I}(\boldsymbol{x}_i \in R_m)$ is the number of data points in child region R_m.
- This $p_{m\ell}$ shows what fraction of points in the region R_m have class label ℓ. In other words, how pure the region is for the class label ℓ.

Decision Tree for Classification

- We had:

$$p_{m\ell} = \frac{1}{n_m} \sum_{\boldsymbol{x}_i \in R_m} \mathbb{I}(y_i = \ell) = \frac{\sum_{i=1}^{n} \mathbb{I}(\boldsymbol{x}_i \in R_m, y_i = \ell)}{\sum_{i=1}^{n} \mathbb{I}(\boldsymbol{x}_i \in R_m)}.$$

- We can have various cost functions such as these:
  - The misclassification error:

$$e = \sum_{m=1}^{|T|} (1 - p_{m\ell}). \tag{13}$$

  - The Gini index:

$$e = \sum_{m=1}^{|T|} \sum_{\ell=1, \ell \neq k}^{c} p_{m\ell} p_{mk}. \tag{14}$$

  - The cross entropy:

$$e = -\sum_{m=1}^{|T|} \sum_{\ell=1}^{c} p_{m\ell} \log(p_{m\ell}). \tag{15}$$

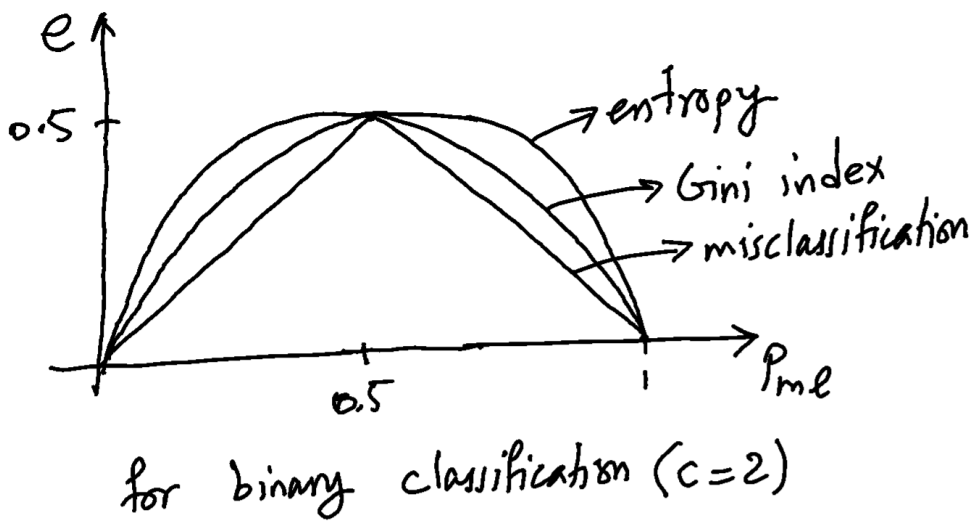
Decision Tree for Classification

- The misclassification error:

$$\text{misclassification:} \quad e = \sum_{m=1}^{|T|}(1 - p_{m\ell}).$$

$$\text{Gini index:} \quad e = \sum_{m=1}^{|T|} \sum_{\ell=1, \ell \neq k}^{c} p_{m\ell} p_{mk}.$$

$$\text{cross entropy:} \quad e = -\sum_{m=1}^{|T|} \sum_{\ell=1}^{c} p_{m\ell} \log(p_{m\ell}).$$



where the cross entropy is scaled to pass through the point $[0.5, 0.5]$.

Noise Injection to Data in Trees

Noise Injection to Data in Trees

- One regularization technique in trees is noise injection to input training data at some level. This prevents overfitting.
- Recall the lecture of overfitting:

$$\textbf{Err} = \textbf{err} - m\,\sigma^2, \quad \text{(instance not in the training set)}$$

$$\textbf{Err} = \textbf{err} - n\,\sigma^2 + 2\,\sigma^2 \sum_{i=1}^{n} \frac{\partial \widehat{f}_i}{\partial y_i}. \quad \text{(instance in the training set)}$$

- If we do not add noise to data, $\sigma = 0$, so we get $\textbf{Err} = \textbf{err} - n\,\sigma^2$ for training data and we will not understand when it starts to overfit. Therefore, if we get overfit without knowing, it will work poorly in the test phase.

Random Forest

Bagging

- **Bagging** is short for **Bootstrap AGGregatING**, first proposed by [1] (1996).
- It is a meta algorithm which can be used with any model (classifier, regression, etc).
- The definition of **bootstrapping** is as follows. Suppose we have a sample $\{\boldsymbol{x}_i\}_{i=1}^n$ with size n where $f(\boldsymbol{x})$ is the unknown distribution of the sample, i.e., $\boldsymbol{x}_i \overset{iid}{\sim} f(\boldsymbol{x})$. We would like to sample from this distribution but we do not know the $f(\boldsymbol{x})$. **Approximating the sampling from the distribution by randomly sampling from the available sample** is named bootstrapping. In bootstrapping, we use **simple random sampling with replacement**. The drawn sample is named **bootstrap sample**.
- In bagging, we draw k **bootstrap** samples each with some sample size. Then, we train the model h_j using the j-th bootstrap sample, $\forall j \in \{1, \ldots, k\}$. Hence, we have k trained models rather than one model. Finally, we **aggregate** the results of estimations of the k models for an instance $\boldsymbol{x}$:

$$\widehat{f}(\boldsymbol{x}) = \frac{1}{k} \sum_{j=1}^{k} h_j(\boldsymbol{x}). \tag{16}$$

- If the model is classifier, we should probably use sign function:

$$\widehat{f}(\boldsymbol{x}) = \text{sign}\Big(\frac{1}{k} \sum_{j=1}^{k} h_j(\boldsymbol{x})\Big). \tag{17}$$

Bagging

- Let e_j denote the error of the j-th model in estimation of the observation of an instance. Suppose this error is a random variable with normal distribution having mean zero, i.e., $e_j \overset{iid}{\sim} \mathcal{N}(0, s)$ where $s := \sigma^2$.
- We denote the covariance of estimations of two trained models using two different bootstrap samples by c.
- Therefore, we have:

$$\mathbb{E}(e_j^2) = s \implies \mathbb{V}\text{ar}(e_j) = \mathbb{E}(e_j^2) - (\mathbb{E}(e_j))^2 = s - 0 = s \implies \mathbb{V}\text{ar}(h_j(\boldsymbol{x})) = s, \tag{18}$$

$$\begin{aligned}\mathbb{E}(e_j\, e_\ell) = c \implies \mathbb{C}\text{ov}(e_j, e_\ell) &= \mathbb{E}(e_j\, e_\ell) - \mathbb{E}(e_j)\mathbb{E}(e_\ell) = c - (0 \times 0) = c \\ &\implies \mathbb{C}\text{ov}(h_j(\boldsymbol{x}), h_\ell(\boldsymbol{x})) = c,\end{aligned} \tag{19}$$

  for all $j, \ell \in \{1, \ldots, k\}, j \neq \ell$.
- According to Eqs. (16), (18), and (19), we have:

$$\begin{aligned}\mathbb{V}\text{ar}\big(\widehat{f}(\boldsymbol{x})\big) &= \frac{1}{k^2}\mathbb{V}\text{ar}\Big(\sum_{j=1}^{k} h_j(\boldsymbol{x})\Big) = \frac{1}{k^2}\sum_{j=1}^{k}\mathbb{V}\text{ar}(h_j(\boldsymbol{x})) + \frac{1}{k^2}\sum_{j=1}^{k}\sum_{\ell=1,\ell\neq j}^{k}\mathbb{C}\text{ov}(h_j(\boldsymbol{x}), h_\ell(\boldsymbol{x})) \\ &= \frac{1}{k^2}ks + \frac{1}{k^2}k(k-1)c = \frac{1}{k}s + \frac{k-1}{k}c.\end{aligned} \tag{20}$$

Bagging

- We found:

$$\mathbb{V}\mathrm{ar}\big(\widehat{f}(\boldsymbol{x})\big) = \frac{1}{k}s + \frac{k-1}{k}c.$$

- The obtained expression has an interesting interpretation: If two trained models with two different bootstrap samples are very correlated, we will have $c \approx s$, thus:

$$\lim_{c \to s} \mathbb{V}\mathrm{ar}\big(\widehat{f}(\boldsymbol{x})\big) = \frac{1}{k}s + \frac{k-1}{k}s = s, \tag{21}$$

  and if the two trained models are very different (uncorrelated), we will have $c \approx 0$, hence:

$$\lim_{c \to 0} \mathbb{V}\mathrm{ar}\big(\widehat{f}(\boldsymbol{x})\big) = \frac{1}{k}s + \frac{k-1}{k}0 = \frac{1}{k}s. \tag{22}$$

- This means that if the trained models are **very correlated** in bagging, there is **not any difference from using only one model**; however, if we have **different** trained models, the **variance of estimation improves significantly by the factor of** k.
- This also implies that **bagging never is destructive**; it **either is not effective or improves** the estimation in terms of variance [2, 1].
- The more complex model usually has more variance and less bias. Therefore, the more variance corresponds to overfitting. As bagging helps **decrease the variance** of estimation, it helps **prevent overfitting**. Therefore, bagging is a meta algorithm useful to have less variance and not to get overfitted [3].
- Bagging can be seen as an **ensemble learning** method [4] which is useful because of **model averaging** [5, 6].

Random Forest

- One of the examples of using bagging in machine learning is **random forest** (2002) [7].
- In random forest, we train different models (trees) using different bootstrap samples (subsets of the training set).
- However, as the trees work similarly, they will be very correlated. For the already explained reason, this will not have a significant improvement from using one tree.
- Random forest addresses this issue by **also sampling from the features (dimensions)** of the **bootstrap sample**. This makes the trained trees very different and thus results in a noticeable improvement.

Acknowledgment

- There are variations of decision tree but they all have the same base we discussed. Some variations are C4.55 (1993) [8] and CART (1984) [9]. For more information on CART, see [10]. For more information on decision trees, see [11, 12].
- For more information on trees in machine learning, see the book: Trevor Hastie, Robert Tibshirani, Jerome H. Friedman, Jerome H. Friedman. "The elements of statistical learning: data mining, inference, and prediction". Vol. 2. New York: springer, 2009 [11].
- Some slides of this slide deck are inspired by teachings of Prof. Ali Ghodsi at University of Waterloo, Department of Statistics and Prof. Hoda Mohammadzade at Sharif University of Technology, Department of Electrical Engineering.

References

[1] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[2] P. L. Bühlmann and B. Yu, "Explaining bagging," in *Research report/Seminar für Statistik, Eidgenössische Technische Hochschule Zürich*, vol. 92, Seminar für Statistik, Eidgenössische Technische Hochschule (ETH), 2000.

[3] L. Breiman, "Arcing classifier (with discussion and a rejoinder by the author)," *The annals of statistics*, vol. 26, no. 3, pp. 801–849, 1998.

[4] R. Polikar, "Ensemble learning," in *Ensemble machine learning*, pp. 1–34, Springer, 2012.

[5] J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky, "Bayesian model averaging: a tutorial," *Statistical science*, pp. 382–401, 1999.

[6] G. Claeskens and N. L. Hjort, *Model selection and model averaging*. Cambridge Books, Cambridge University Press, 2008.

[7] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[8] J. R. Quinlan, "Program for machine learning," *C4. 5*, 1993.

[9] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and regression trees–crc press*. CRC Press, 1984.

References (cont.)

[10] J. M. Klusowski, "Analyzing cart," *arXiv preprint arXiv:1906.10086*, 2019.

[11] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*, vol. 2.
Springer, 2009.

[12] W.-Y. Loh, "Classification and regression trees," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 1, no. 1, pp. 14–23, 2011.