Chapter 4: ML Models for Tabular Datasets

ML Instruction Team, Fall 2022

CE Department
Sharif University of Technology

Ensemble methods

- **Ensemble methods** is a machine learning technique that combines several base models in order to produce one optimal predictive model.

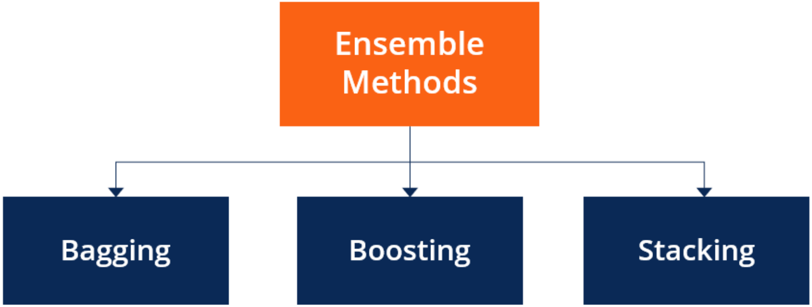


Figure: Three main approaches, (Source)

When to use Ensemble learning

- Ensemble learning works best when the base models are not correlated. Models with high correlation tend to deviate towards same direction, thus, promediating a greater and more correlated error.
- It also helps with reducing variance and making more robust models. More samples lead to more unbiased predictor. (Due to Central limit theorem)
- You will often use Ensemble methods **near the end of a project** , once you have already built a few good predictors, to combine them into an even better predictor.

Hard Voting

- **Hard Voting** → Majority Vote

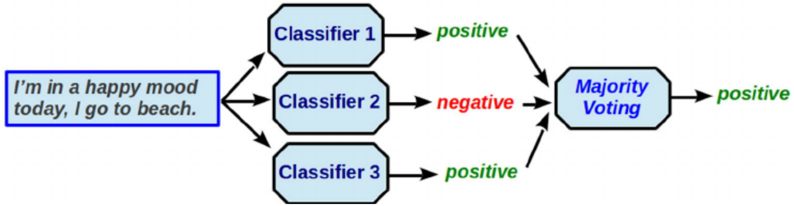


Figure: Example of Majority Voting, (Source)

$$\hat{y} = arg\max_{j} \sum_{i=1}^{n} h_i^j(x)$$

$$H(x) = \{h_1, h_2, ..., h_n\}, h_i^j(x) \in [0, 1]$$

$h_i^j(x)$: The predicted class membership probability of the ith classifier for class label j.

Why Majority Voting?

- Assume n independent (errors are uncorrelated) binary classifiers with a base error rate ϵ. (better than random guessing)
- Probability of making a wrong prediction via the ensemble if k classifiers predict same class label:

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

- **Ensemble error:**

$$\epsilon_{ens} = \sum_{k}^{n} \epsilon^k (1-\epsilon)^{n-k}, k > \lceil n/2 \rceil$$

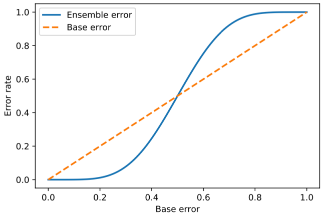


Figure: Ensemble error plot[2]

Soft Voting

- **Soft Voting** $\rightarrow$ Highest class probability (When **all** classifiers are able to estimate class probabilities)

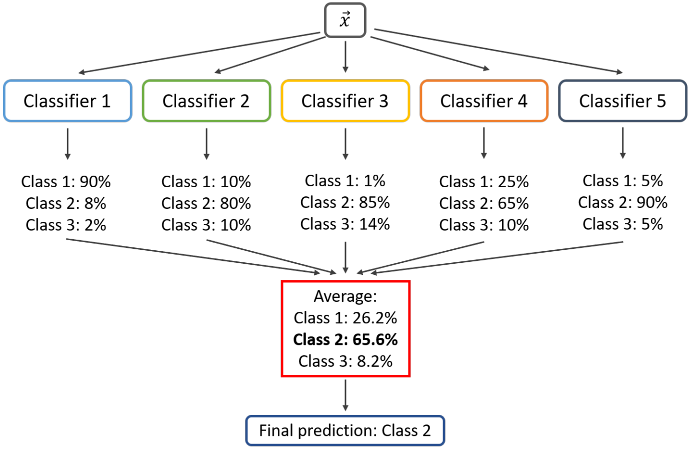


Figure: Example of Soft Voting, (Source)

Soft Voting

$$\hat{y} = arg\max_j \sum_{i=1}^{n} w_i h_i^j(x), w_i > 0, \sum_{i=1}^{n} w_i = 1$$

$$H(x) = \{h_1, h_2, ..., h_n\}, h_i^j(x) \in [0, 1]$$

- $h_i^j(x)$: The predicted class membership probability of the ith classifier for class label j, w_i is the optional weighting parameter. **Example:**

$H(x)$	$j = 0$	$j = 1$	w_i
h_1	0.9	0.1	0.2
h_2	0.8	0.2	0.3
h_3	0.3	0.7	0.5

Figure: Probability and Weights of each predictor[1]

$$p(j = 0|x) = 0.2 * 0.9 + 0.3 * 0.8 + 0.3 * 0.5 = 0.57$$

$$p(j = 1|x) = 0.2 * 0.1 + 0.3 * 0.2 + 0.3 * 0.7 = 0.28$$

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

Plurality Voting

- **Plurality Voting (Unstable)** $\rightarrow$ Most Voted (When none of the predictions get more than half of the votes)

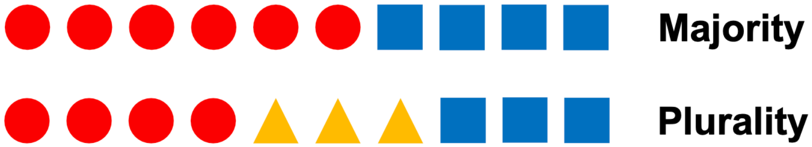


Figure: Majority vs Plurality[1]

Bootstrapping

- A resample method that consists of repeatedly drawn, with replacement, samples from data to form other smaller datasets. (The observations **can** appear more than one time)

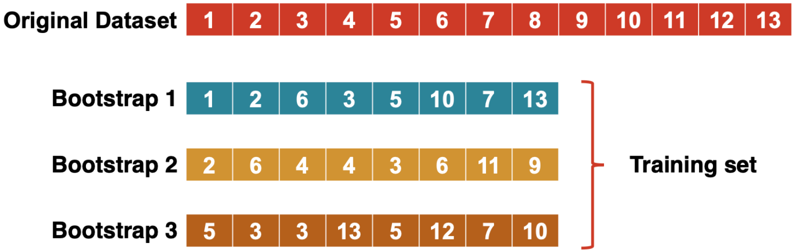


Figure: Example of Bootstrapping[1]

Bagging

- Use the same algorithm for every predictor and train them on different random subsets.
  - Sampling with replacement: **Bagging** (Bootstrap Aggregating)
  - Sampling without replacement: **Pasting**
- In this approach we generate B different bootstrapped training data sets.

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

- The aggregation function is typically:
  - Mode (Most frequent) for Classification
  - Average for Regression
- Predictors can all be trained in parallel, via different CPU cores or even different servers. Similarly, predictions can be made in parallel.

Bagging

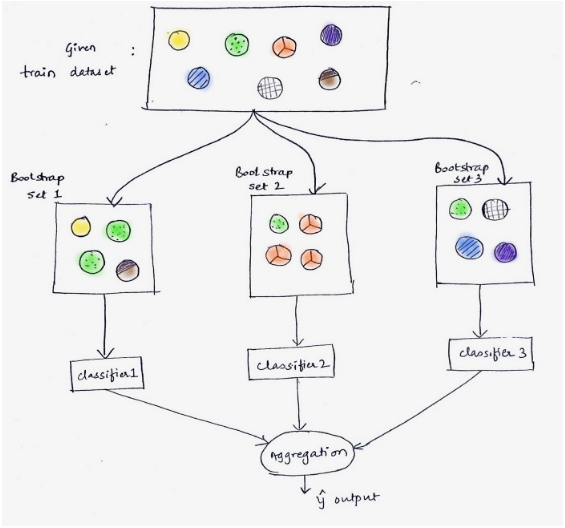


Figure: Example of Bagging, (Source)

Out-of-Bag Evaluation

- With bagging, some instances may be sampled several times for any given predictor, while others may not be sampled at all.
- The remaining of the training instances that are not sampled are called **out-of-bag (OOB)** instances.

$$P(NotChosen) = (1 - \frac{1}{n})^n, n \to \infty$$

$$\frac{1}{e} \approx 0.368$$

$$P(Chosen) = 1 - (1 - \frac{1}{n})^n \approx 0.632$$

- This means that when the dataset is big enough, **37%** of its samples are never selected and we could use it to test our model.

$P(Chosen)$

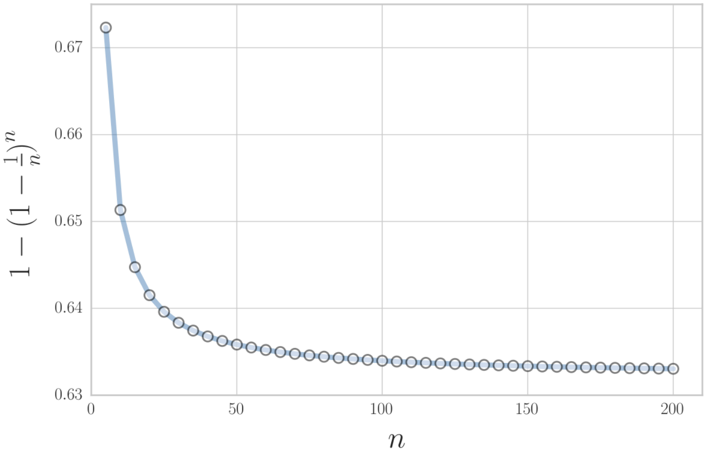


Figure: Plot of $P(Chosen)$[3]

Random Patches and Random Subspaces

- You can sample **features** instead of instances and train each predictor on them.
  - Sampling both training instances and features is called the **Random Patches** method.
  - But keeping all training instances, but sampling features is called the **Random Subspaces** method.
- They are useful when you are dealing with high-dimensional inputs. (such as images)
- Sampling features leads to trading a bit more bias for a lower variance.

Random Forests

- **Ensemble of Decision Trees**. Generally uses bagging and feature randomness to try to create an uncorrelated forest of trees.
- Instead of searching for the very best feature when splitting a node, it searches for the best feature among a **random subset of features**.

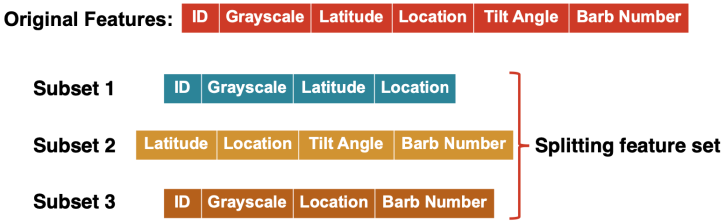


Figure: Split based on features[1]

- It is possible to make trees even more random by also using random thresholds for each feature rather than searching for the best possible thresholds.
- It is called an **Extremely Randomized Trees ensemble** or **Extra-Trees** for short.

Steps of Random Forest

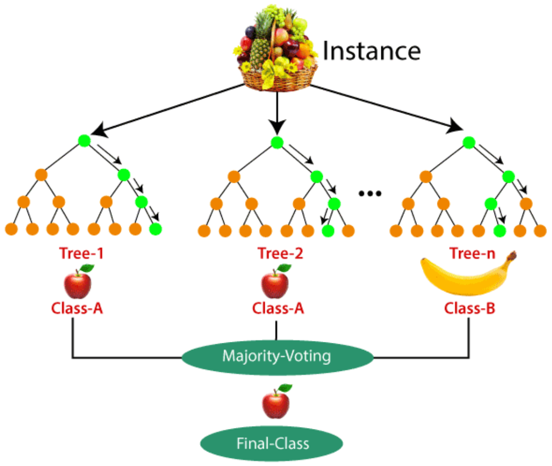


Figure: Example of Random forest, (Source)

Features of Random Forest

- Pros:
  - It can be used for both regression and classification tasks
  - Higher accuracy
  - Reduce risk of overfitting
  - Measures feature's importance
- Cons:
  - Time consuming (ineffective for real-time predictions)
  - Many parameters: depth of tree, number of trees, type of node tests, random sampling
  - Requires a lot of training data and large memory footprint
  - Sacrifice the interpretability of model

Boosting

- **Boosting** (originally called hypothesis boosting) refers to any Ensemble method that can combine several weak learners into a strong learner.
- The model is weak if it has a substantial error rate, but the performance is not random.
- The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor.
- Most popular boosting methods (Differ in terms of how weights are updated & classifiers are combined):
  - Adaptive Boosting (AdaBoost)
  - Gradient Boosting (LightGBM, XGBoost)

Boosting

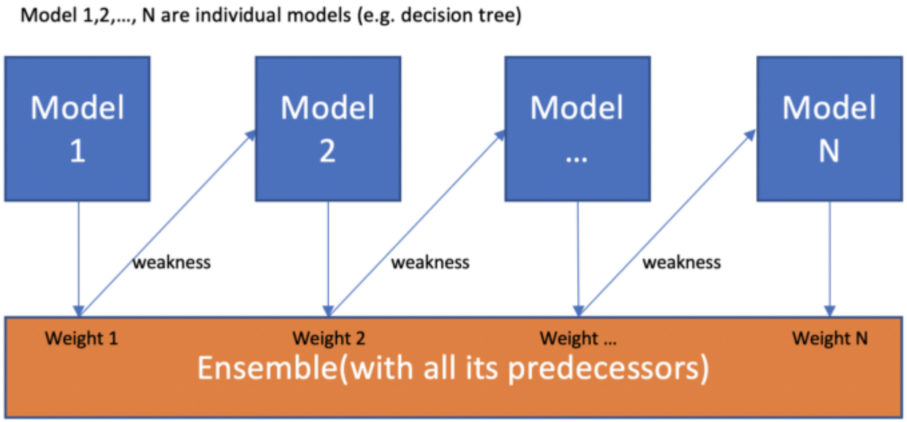

Model 1,2,…, N are individual models (e.g. decision tree)
Model 1
Model 2
Model …
Model N
weakness
weakness
weakness
Weight 1
Weight 2
Weight …
Weight N
Ensemble(with all its predecessors)

Figure: Boosting Process, (Source)

AdaBoost

- One way for a new predictor to correct its predecessor is to pay a bit more attention to the training instances that the predecessor underfitted.
- This results in new predictors focusing more and more on the hard cases.
- **Drawback of AdaBoost**: It doesn't scale as well as bagging (or pasting).

AdaBoost Steps

- First trains a base classifier and uses it to make predictions
- Then increases the relative weight of misclassified instances
- Then it trains a second classifier, using the updated weights, and again makes predictions and so on...

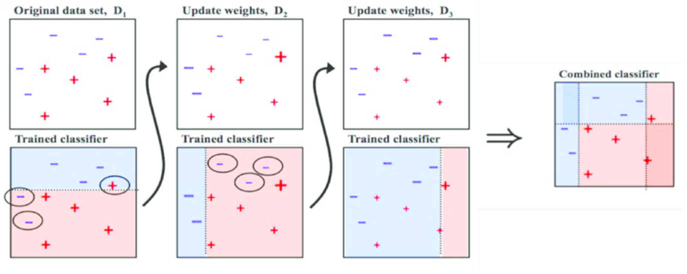


Figure: AdaBoost Steps, (Source)

- Once all predictors are trained, the ensemble makes predictions very much like bagging, except that predictors have different weights.

AdaBoost Algorithm

- Each instance weight $w^{(i)}$ is initially set to $1/m$ and first predictor's weighted error rate is r_1.
- Weighted error rate of the j^{th} predictor:

$$r_j = \frac{\sum\limits_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$$

  where $\hat{y}_j^{(i)}$ is the j^{th} predictor's prediction for the i^{th} instance.
- The predictor's weight:

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

  where η is the learning rate.
  - The more accurate the predictor is, the higher its weight will be.
  - If it is just guessing randomly, then its weight will be close to zero.
  - However, if it is most often wrong, then its weight will be negative.

AdaBoost Algorithm

- Next, the AdaBoost algorithm update the instance weights, which boosts the weights of the misclassified instance:

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} \exp(\alpha_j) & \text{if } \hat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$

  Then all the instance weights gets normalized by dividing by $\sum_{i=1}^{m} w^{(i)}$.
- Finally, a new predictor is trained using the updated weights, and the whole process is repeated.
- The algorithm stops when the desired number of predictors is reached, or when a perfect predictor is found.
- AdaBoost predictions:

$$\hat{y}(x) = argmax \sum_{\substack{j=1 \\ \hat{y}_j(x)=k}}^{N} \alpha_j$$

Gradient Boosting

- In this approach, instead of tweaking the instance weights at every iteration like AdaBoost does, this method tries to fit the new predictor to the residual errors made by the previous predictor.

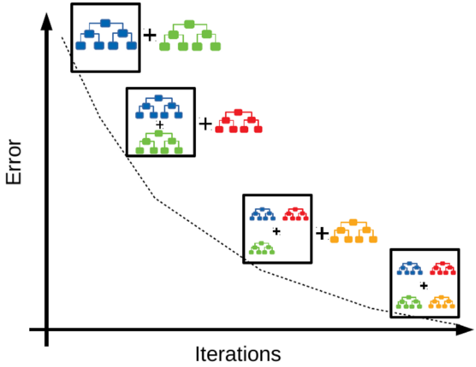
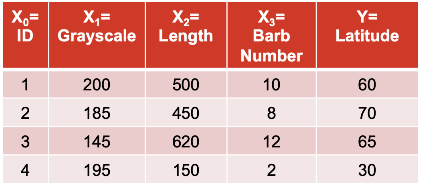


Figure: Gradient Boosting Process, (Source)

Gradient Boosting Steps

Example:



X_0= ID	X_1= Grayscale	X_2= Length	X_3= Barb Number	Y= Latitude
1	200	500	10	60
2	185	450	8	70
3	145	620	12	65
4	195	150	2	30

Figure: Dataset[1]

- Construct base tree(just the root node):

$$y_1^* = \frac{1}{n}\sum_{i=1}^{n} y^i = 56.25$$

Gradient Boosting Steps

- Build next tree based on errors of the previous tree:

$$r_1 = y_1 - y_1^*$$

X_0= ID	X_1= Grayscale	X_2= Length	X_3= Barb Number	Y= Latitude	r_1= residual
1	200	500	10	60	60-56.25=3.75
2	185	450	8	70	70-56.25=13.75
3	145	620	12	65	65-56.25=8.75
4	195	150	2	30	30-56.75=-26.75

Figure: Dataset after calculating residuals[1]

Gradient Boosting Steps

- Then, create a tree based on $x_1, x_2, \ldots, x_m$ to fit the residuals:

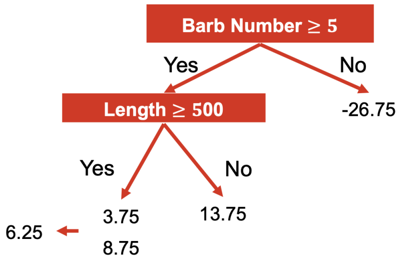


Figure: Create a tree[1]

- Combine tree from step 1 with tree:

$$\text{Predict (ID=3)}: 56.25 + \alpha * 6.25$$

Where α learning rate between 0 and 1(if $\alpha = 1$, low bias but high variance)

XGBoost

- **XGBoost**, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library.
- It provides parallel tree boosting and is the leading machine learning library for regression, classification, and ranking problems.
- The main difference is that XGBoost uses a more regularized model, which helps to prevent overfitting.

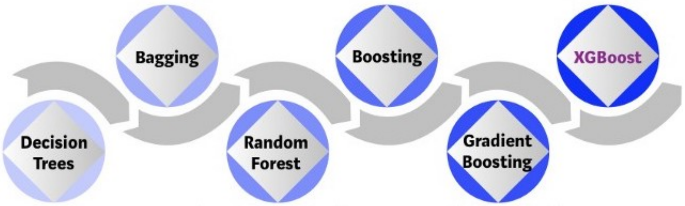


Figure: Evolution of tree-based algorithms, (Source)

XGBoost

- XGBoost uses pre-sorted algorithm & Histogram-based algorithm for computing the best split.
- **Pre-Sorting splitting**
  - For each node, enumerate over all features
  - For each feature, sort the instances by feature value
  - Use a linear scan to decide the best split along that feature basis information gain
  - Take the best split solution along all the features
- In simple terms, Histogram-based algorithm splits all the data points for a feature into discrete bins and uses these bins to find the split value of histogram.

CatBoost

- An open-source machine learning algorithm which allows users to quickly handle categorical features for a large dataset.
- CatBoost uses oblivious decision trees, where the same splitting criterion is used across an entire level of the tree.
- Advantages:
  - No need to preprocess categorical features and also Support missing value
  - No overfitting unlike Gradient Boosting
  - It has all the good features of others boosting

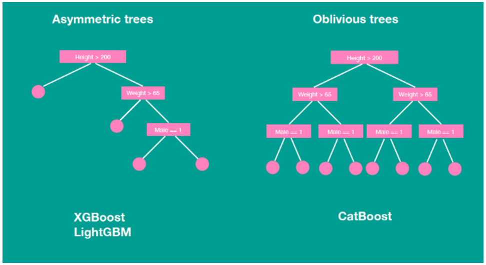


Figure: Difference between CatBoost and XGBoost, (Source)

Stacking

- **Blender** or **Meta-learner** or **Meta-classifier** takes predictions of each predictor as inputs and makes the final prediction. Basically we train a model to perform aggregation.

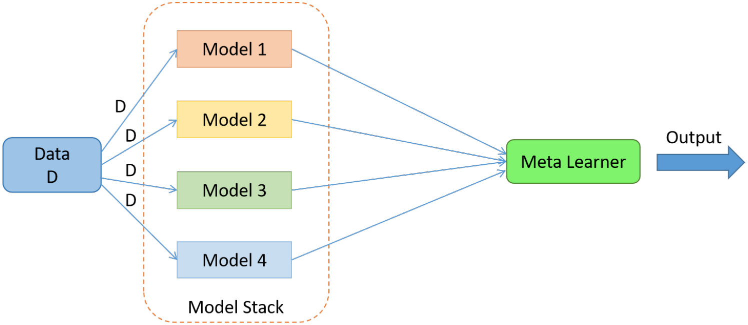


Figure: Example of Stacking, (Source)

Features of Stacking

- Usually has weak-learners of different types.
- Meta-learner tries to learn which weak-learner is more important.
- The weak-learners are trained in **parallel**, but the meta learner is trained **sequentially**.
- Once the weak-learners are trained, their weights are kept static to train the meta-learner.
- Usually, the meta-learner is trained on a different subset than what was used to train the weak-learners.
- It is common to use a linear model for aggregation to avoid complexity.

Training Blender

- To train the blender, a common approach is to use a **hold-out set**.
- First, the training set is split into two subsets. The first subset is used to train the predictors in the first layer.
- Next, the first layer's predictors are used to make predictions on the second (held-out) set.
- Then we train the blender on this new training set, so it learns to predict the target value, given the first layer's predictions.

Training Blender

Stacking with Cross-Validation

- The problem with this method is that it has a high tendency to suffer from extensive overfitting.
- A better alternative would be to use stacking with k-fold cross-validation or leave-one-out cross-validation.

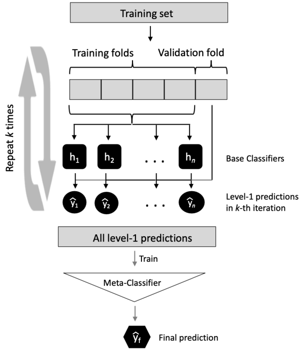


Figure: Example of Stacking with Cross-validation[3]

Multilayer Stacking

Key Elements

- Bagging
  - Bootstrap samples of the training dataset
  - Unpruned decision trees fit on each sample
  - Simple voting or averaging of predictions
- Stacking
  - Unchanged training dataset
  - Different machine learning algorithms for each ensemble member
  - Machine learning model to learn how to best combine predictions
- Boosting
  - Bias training data toward those examples that are hard to predict
  - Iteratively add ensemble members to correct predictions of prior models
  - Combine predictions using a weighted average of models

Resources

[1] Qi Hao. *Decision Tree and Ensemble learning*.

[2] Sebastian Raschka. *STAT 451: Machine Learning*. http://stat.wisc.edu/~sraschka/teaching/stat451-fs2020/. 2020.

[3] Sebastian Raschka. *STAT 479: Machine Learning Lecture Notes*. http://stat.wisc.edu/ sraschka/teaching/stat479-fs2018/. 2018.

Thank You!

Any Question?