Machine Learning

Random Forests and Bagging

Edgar F. Roman-Rangel.
edgar.roman@itam.mx

Digital Systems Department.
Instituto Tecnológico Autónomo de México, ITAM.

May 14th, 2021.

Outline

Decision trees

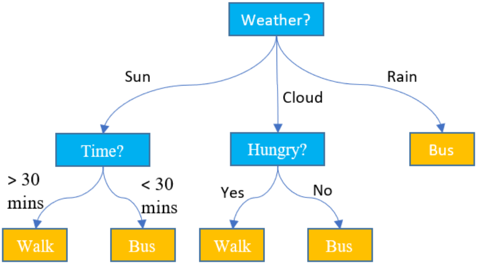
Random forest

Bagging

Boosting

Reminder

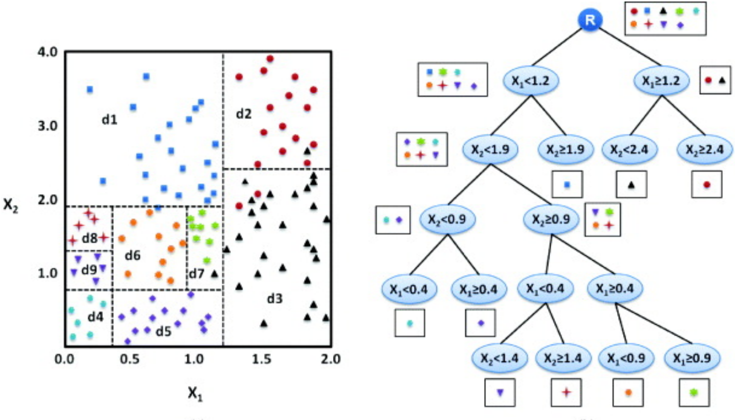
- Break data down, by a series of decisions.
- Intuitive (good interpretability).
- Work with numeric, rank, and categorical features.



- Can be turn into regression trees.
- Prone to overfitting.

Feature space

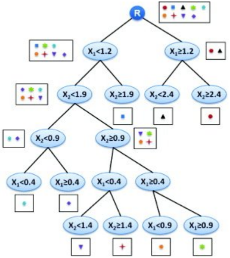
Good interpretability: partitioning of the feature space.



(a) (b)

Training

1. Start with the **root**. Split data using the feature with the largest **information gain**.
2. Repeat the process iteratively, creating new **nodes** until **leaves** are pure.



- Might produce very deep trees.
- Might result in overfitting.

Regularize by pruning:

- Limiting depth, or
- Number of points in a leave.

Information gain

Difference between the impurity of a parent node (N_P) and the sum of its children (N_j) impurities.

$$G(f|N_p) = I(N_p) - \sum_{j=1}^{J} \frac{|N_j|}{|N_p|} I(N_j),$$

where,

- $G(\cdot)$: information gain.
- f: feature being evaluated.
- $I(\cdot)$: impurity function.
- j: index of the j-th children (often, $J = 2$ for simplicity).
- $|\cdot|$: cardinality function.

Impurity functions

Entropy (I_H)

$$I_H = -\sum_{c=1}^{C} p(c|n) \log p(c|n),$$

where, $p(c|n)$: proportion of samples from class c at node n.

Gini index (I_G)

$$I_G = \sum_{c=1}^{C} p(c|n) \left(1 - p(c|n)\right)$$

$$= 1 - \sum_{c=1}^{C} p(c|n)^2.$$

Regression trees

For regression problems.

- Nodes are intervals of the independent variables.
- Leaves are the average of dependent variables.
- Minimize residual error metrics, e.g., mse.

Outline

Decision trees

Random forest

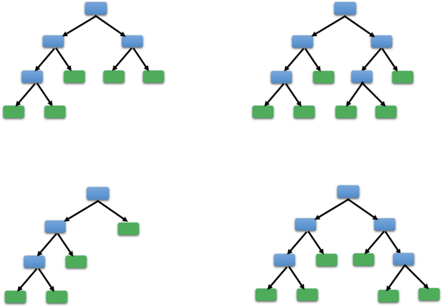
Bagging

Boosting

Intuition

Limitation of a single decision tree, might be overcome by an assemble of trees, a.k.a., forest.

"Combine weak learners to build a strong learner".



Training

1. Draw a random bootstrap sample set from the training set (with replacement).
2. Grow a decision tree from the bootstrap sample set.
   2.1 Randomly select d features at each step (without replacement).
   2.2 Split data using information gain.
3. Repeat 1., and 2., k times (create k random trees).
4. Aggregate results by majority voting.

Forest

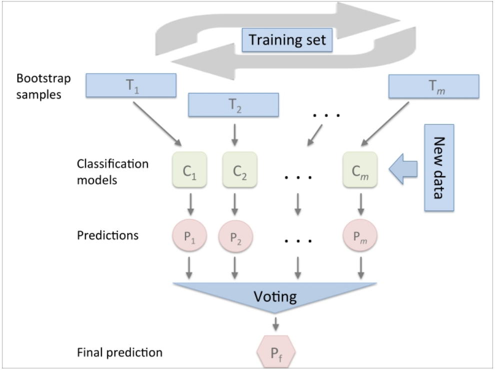
- Training results in a wide variety of trees.
- Often leading to better performance.
- Limited interpretability.
- Hyperparameters: k, size of bootstrap set (N), number of features ($d = \sqrt{D}$).
- Often used with shallow trees (depth ≈ 2).

Outline

Bootstrap + aggregating

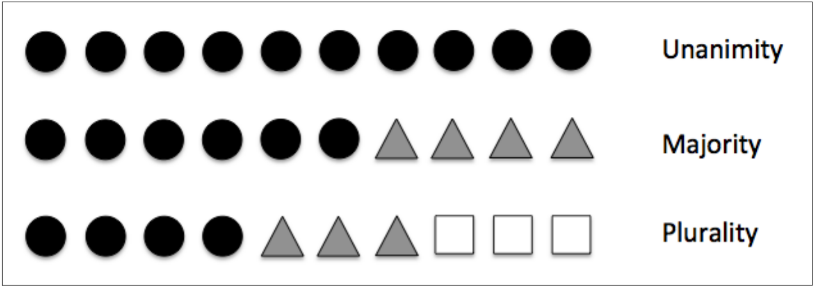
Bagging: bootstrap dataset with aggregation (majority voting).

Can be extended beyond assemble of trees (use different methods).



Voting

Consider: unanimity vs., majority vs., plurality.



Prediction

$$\hat{y} = \text{mode}\{f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x})\},$$

this is, the most voted output among all models f_m.

Alternatively, we can use a weighted vote, as

$$\hat{y} = \arg\max_{\mathcal{C}} \sum_{m=1}^{M} \omega_m \mathbb{1}\left(f_m(\mathbf{x}) = c\right),$$

where,

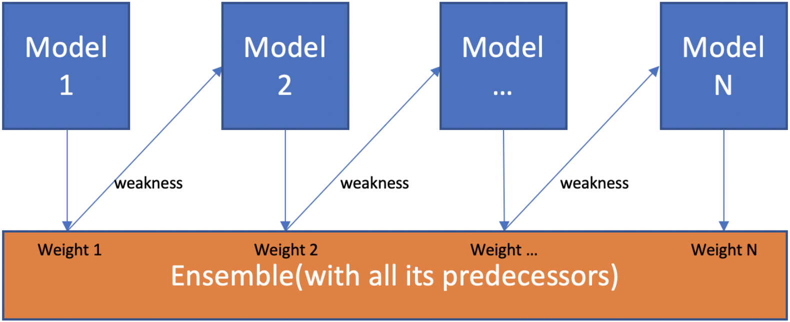
- $\mathcal{C}$: is the set of classes,
- ω_m: is the weight for the m-th model,
- $\mathbb{1}(\cdot)$: is the indicator function (1 if $f(\mathbf{x}) = c$ or 0 otherwise).

Outline

Intuition

Combine a set of weak learners. Subsequently learn from misclassified training samples to improve the performance.

Model 1,2,..., N are individual models (e.g. decision tree)

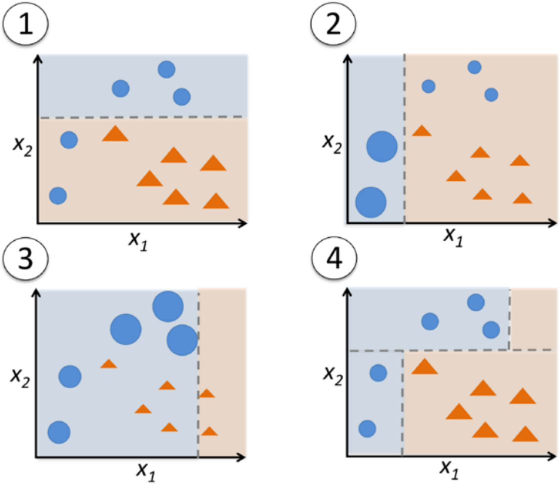


Initial idea

1. Draw a random subset $\mathbf{X}_1 \subset \mathbf{X}$, without replacement.
2. Train a weak learner $f_1(\cdot)$.
3. Draw second random subset $\mathbf{X}_2 \subset \mathbf{X}$, without replacement, and add 50% of previously misclassified samples.
4. Train a second weak learner $f_2(\cdot)$.
5. Find training set $\mathbf{X}_3 \subset \mathbf{X}$ on which $f_1(\cdot)$ and $f_2(\cdot)$ disagree.
6. Train a third weak learner $f_3(\cdot)$.
7. Combine $f_1(\cdot)$, $f_2(\cdot)$, and $f_3(\cdot)$ via majority voting.

Adaptive boosting

Adaboost: adaptive reweighting of samples.



Training Adaboost

1. Set weight vector $\mathbf{w}$ to uniform weights, where $\sum_n \omega_n = 1$.
2. For $m = 1, \ldots, M$ boosting rounds, do:
   2.1 Train a weighted weak learner $f_m(\cdot)$.
   2.2 Predict class labels: $\hat{\mathbf{y}} = f_m(\mathbf{X})$.
   2.3 Compute weighted error rate: $\varepsilon = \mathbf{w}^T(\hat{\mathbf{y}} == \mathbf{y})$.
   2.4 Compute coefficient: $\alpha_m = 0.5 \log \frac{1-\varepsilon}{\varepsilon}$.
   2.5 Update weights: $\mathbf{w} = \mathbf{w} \times \exp(-\alpha_m \times \hat{\mathbf{y}} \times \mathbf{y})$.
   2.6 Normalize weights: $\mathbf{w} = \frac{\mathbf{w}}{\sum_n \mathbf{w}_n}$.
3. Compute final prediction: $\hat{\mathbf{y}} = \sum_{m=1}^{M} \alpha_m \times f_m(\mathbf{X})$.

Q&A

Thank you!

edgar.roman@itam.mx