STA 221: LECTURE 11

KRISHNA BALASUBRAMANIAN

(UNIVERSITY OF CALIFORNIA, DAVIS)

Examples of Classifiers

Logistic Regression ✓

Support vector Machine

K-NN

Decision Tree

Random Forest

Gradient Boosted Decision Tree (GBDT)

Support Vector Machine

Support vector machines (SVM) were the best performing general purpose classifier before the recent deep learning revolution.

We assume binary classification with $y \in \{+1, -1\}$ and represent the classifier using the inner product notation $\widehat{y} = \text{sign}\langle \boldsymbol{w}, \boldsymbol{x} \rangle$ where $\langle x, z \rangle = x^\top z$ in vector notation.

Note that a bias term may be included i.e. $\widehat{y} = \text{sign}(\boldsymbol{w}_0 + \langle \boldsymbol{w}, \boldsymbol{x} \rangle)$ using the notation $\langle \boldsymbol{w}, \boldsymbol{x} \rangle$ if $\boldsymbol{x}$ is augmented with an always one component.

Similarly, note that $\boldsymbol{x}$ may be a vector of non-linear features of the actual data $\boldsymbol{x}'$ i.e. $\boldsymbol{x}_1 = \boldsymbol{x}'_1 \boldsymbol{x}'^2_2$ so the linear classifier in the space of $\boldsymbol{x}$ is really non-linear in the original data space of $\boldsymbol{x}'$.

The linear classifier $f(\boldsymbol{x}) = \text{sign}\langle \boldsymbol{w}, \boldsymbol{x} \rangle$ is parameterized by a weight vector which is normal (i.e. perpendicular) to the decision boundary which is a subspace or a linear hyperplane passing through the origin (note that as described above this does not preclude having a bias term).

Any $\boldsymbol{x}$ can be represented as a sum of its projection onto the subspace and its perpendicular component $\boldsymbol{x} = \boldsymbol{x}_\perp + \boldsymbol{x}_\parallel = \boldsymbol{x}_\parallel + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$. Since

$$\langle \boldsymbol{w}, \boldsymbol{x} \rangle = \langle \boldsymbol{w}, \boldsymbol{x}_\parallel \rangle + \langle \boldsymbol{w}, r\boldsymbol{w}/\|\boldsymbol{w}\| \rangle = 0 + r\|\boldsymbol{w}\| \quad \Rightarrow$$
$$r = \langle \boldsymbol{w}, \boldsymbol{w} \rangle / \|\boldsymbol{w}\|,$$

we have that for correctly classified points $\boldsymbol{x}_i, y_i$ the distance to the hyperplane is $|r_i| = y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle / \|\boldsymbol{w}\|$.

The idea of support vector machines in the context of linearly separable data is to choose w that leads to the largest margin - defined as the distance of the closest data point to the hyperplane

$$\boldsymbol{w} = \arg\max_{\boldsymbol{w}\in\mathbb{R}^d} \left(\|\boldsymbol{w}\|^{-1} \min_{1\leq i\leq n} y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \right). \tag{1}$$

The direct solution of (1) is difficult as the objective function is non-differentiable. We proceed by solving an equivalent optimization problem that is easier to solve.

We start by observing that rescaling the weight vector $\boldsymbol{w}' = c\boldsymbol{w}, c \in \mathbb{R}_+$ leaves the classifier $f(\boldsymbol{x})$ unchanged and does not change the distance r of points to the subspace. More importantly, it also leaves the objective function (1) unchanged. By rescaling $\boldsymbol{w}$ so that the distance of the closest point to the hyperplane is 1 we get that $\min_{1 \leq i \leq n} y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq 1$ with the minimum achieved for one or more training points

$$\boldsymbol{w} = \arg \min_{\boldsymbol{w} \in \mathbb{R}^d} \frac{1}{2} \|\boldsymbol{w}\|^2 \quad \text{subject to} \quad y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq 1 \quad i = 1, \ldots, n. \tag{2}$$

A different way to see the equivalence between (1) and (2) is to note that as $\boldsymbol{w}$ gets closer to the origin (as we try to do by minimizing $\|\boldsymbol{w}\|$) one or more of the constraints will be satisfied with equality rather than inequality in which case (1) and (2) are equivalent (i.e., at the solution one or more of the constraints must be active with equality).

Problem (2) is a quadratic program (minimization of a quadratic function subject to linear constraints) and is easier to solve than (1).

However, it involves a large number of linear inequality constraints. The dual problem which is yet another equivalent SVM formulation is the easiest to solve computationally.

It is obtained by optimizing the Lagrangian

$$\mathcal{L}(\boldsymbol{w}, \lambda) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \lambda_i (y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - 1) \tag{3}$$

with respect to w first, substituting the solution into $\mathcal{L}$ and then optimizing with respect to λ (recall for a point to be a solution of the constrained optimization problem both $\nabla_{\boldsymbol{w}}\mathcal{L}$ and $\nabla_{\lambda}\mathcal{L}$ need to be zero).

$$0 = \frac{\partial \mathcal{L}(\boldsymbol{w}, \lambda)}{\partial \boldsymbol{w}_i} \quad i = 1, \ldots, n \quad \Rightarrow \quad \boldsymbol{w}^* = \sum_{j=1}^{n} \lambda_j^{(j)} \boldsymbol{x}_i \boldsymbol{x}_j \tag{4}$$

$$\mathcal{L}(\lambda, w^*) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_i \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \sum_{i=1}^{n} \lambda_i$$

$$= \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle. \tag{5}$$

We have thus shown, using convex duality that the equivalent dual formulation of SVM is

$$f(\boldsymbol{x}) = \text{sign}\langle \boldsymbol{w}, \boldsymbol{x} \rangle = \text{sign}\Big\langle \sum_{j=1}^{n} \lambda_j y_j \boldsymbol{x}_j, \boldsymbol{x} \Big\rangle \tag{6}$$

$$= \text{sign} \sum_{j=1}^{n} \lambda_j y_j \langle \boldsymbol{x}_j, \boldsymbol{x} \rangle \tag{7}$$

where

$$\lambda = \arg\max_{\lambda \in \mathbb{R}^d} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \tag{8}$$

$$\text{subject to} \quad \lambda_i \geq 0. \tag{9}$$

Notice that (6) is also a quadratic program, but in contrast to (2) the constraints are much simpler. Also the number of variables and constrains are n rather than d (data dimensionality) which favors (6) further in high dimensional cases. Furthermore, the solution of (6) will have non-zero λ_j only for constraints that hold with equality at the optimum (the corresponding training points are called support vectors). Thus, many of the λ_j will converge early on to zero effectively reducing the dimensionality of (6).

Non-Separable Case: Thus far, we have assumed that the training data is linearly separable (can be correctly classified using a linear decision surface).

We proceed as before in the separable case, only that this time some examples may be on the wrong side of the hyperplane. In these cases we introduce ξ_i that measure the amount of violation in the constraints $y_i\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle \geq 1$:

$$(\boldsymbol{w}, \xi) = \arg \min_{\boldsymbol{w}\in\mathbb{R}^d, \xi\in\mathbb{R}^n} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m}\xi_i \tag{10}$$

$$\text{subject to} \quad y_i\langle \boldsymbol{w}, \boldsymbol{x}_i\rangle \geq 1-\xi_i, \quad \xi_i \geq 0 \quad i = 1, \ldots, n. \tag{11}$$

The parameter $C \geq 0$ is a regularization parameter controlling the relative importance of the two terms: maximizing margin for correctly classified examples and minimizing the misclassification penalty $\sum \xi_i$. Since we want small ξ_i (in order to minimize the objective function) we can easily determine $\xi_i = \max(0, 1 - y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle)$ and remove ξ_i as optimization variables

$$\boldsymbol{w} = \arg \min_{\boldsymbol{w} \in \mathbb{R}^d} \frac{1}{2} \|\boldsymbol{w}\|^2 + C \sum_{i=1}^{m} \max(0, 1 - y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle). \quad (12)$$

Proceeding as before the dual formulation leads to

$$\lambda = \arg\max_{\lambda \in \mathbb{R}^d} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \tag{13}$$

$$\text{subject to} \quad 0 \leq \lambda_i \leq C \quad i = 1, \ldots, n \tag{14}$$

(with the classifier $f(\boldsymbol{x})$ given in terms of the dual variables λ as before).

An interesting observation is that (12) can be interpreted as L_2 regularized margin based minimizier

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} l_{\text{hinge}}(y_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle) + c\|\boldsymbol{w}\|^2$$

where $l_{\text{hinge}}(z) = (1 - z)_+$. Viewed in this way, we see a striking similarity between SVM and other penalized likelihood methods such as regularized logistic regression.

The function $l_{\text{hinge}}(z)$, called the hinge loss, represents an empirical loss and replaces the logistic regression negative loglikelihood.

The term $\|w\|^2$ represents regularization penalty.

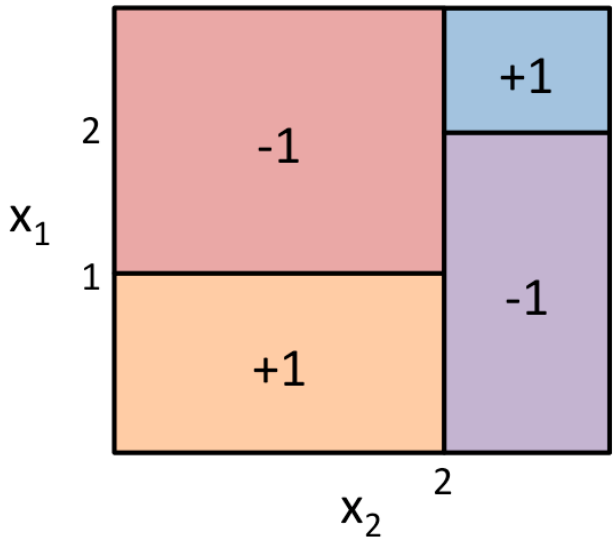
Decision Tree

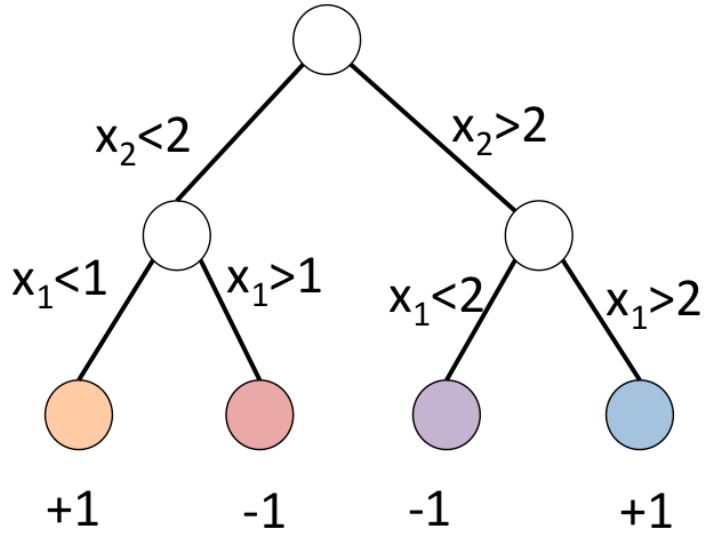
Decision Tree

Each node checks one feature x_i:

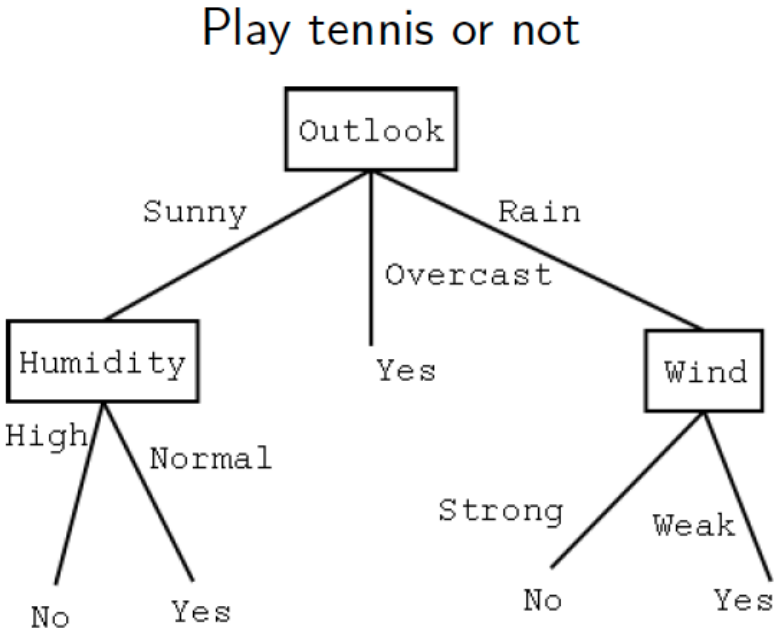
Go left if $x_i <$ threshold

Go right if $x_i \geq$ threshold





A real example



Decision Tree

Strength:

- It's a nonlinear classifier
- Better interpretability
- Can naturally handle categorical features

Computation:

- Training: slow
- Prediction: fast
  - h operations (h: depth of the tree, usually ≤ 15)

Decision Tree

Strength:

- It's a nonlinear classifier
- Better interpretability
- Can naturally handle categorical features

Computation:

- Training: slow
- Prediction: fast
  - h operations (h: depth of the tree, usually ≤ 15)

Splitting the node

Classification tree: Split the node to maximize entropy

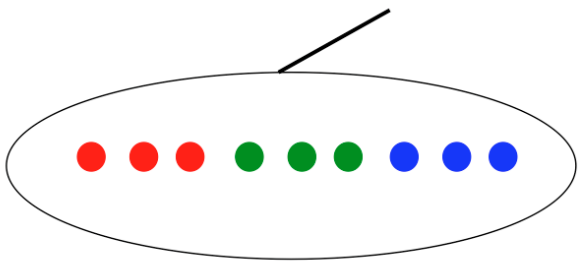
Let S be set of data points in a node, $c = 1, \cdots, C$ are labels:

$$\text{Entroy} : H(S) = -\sum_{c=1}^{C} p(c) \log p(c),$$

where $p(c)$ is the proportion of the data belong to class c.

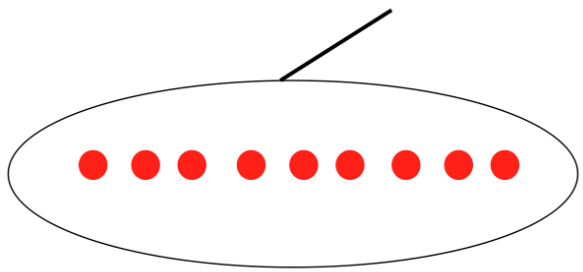
Entropy=0 if all samples are in the same class

Entropy is large if $p(1) = \cdots = p(C)$



Entropy:
-(1/3)*log(1/3) – (1/3)*log(1/3) – (1/3) * log(1/3)
= 1.58

Bad split



Entropy:
-1*log*(1) = 0

Good split

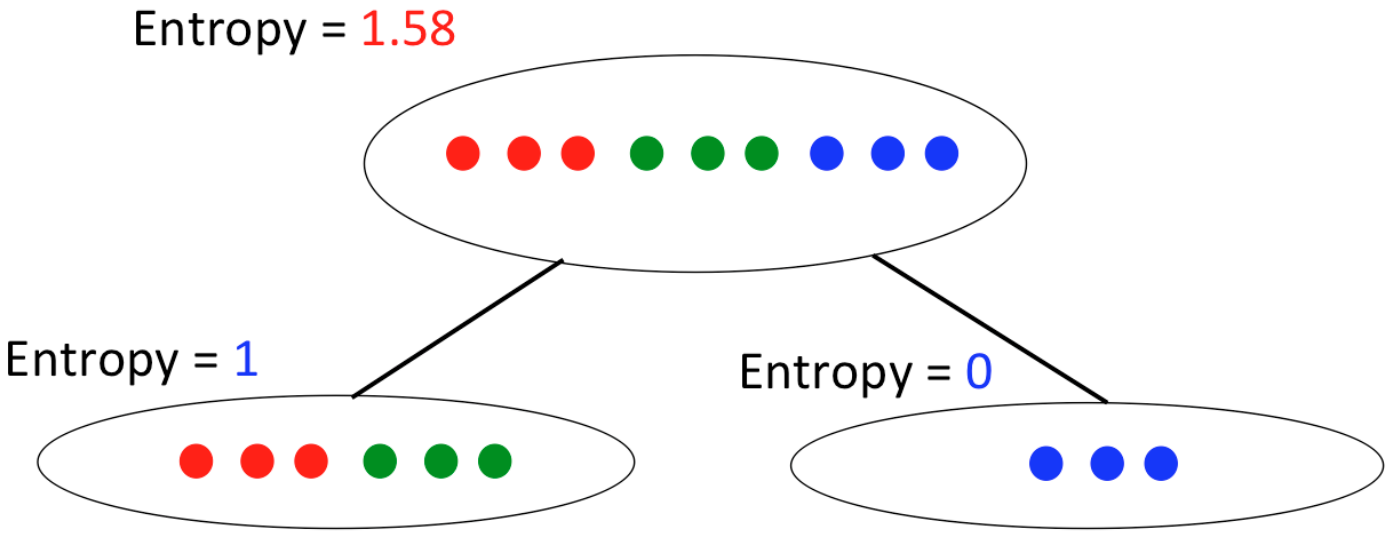
The averaged entropy of a split $S \to S_1, S_2$

$$\frac{|S_1|}{|S|} H(S_1) + \frac{|S_2|}{|S|} H(S_2)$$

Information gain: measure how good is the split

$$H(S) - \Big((|S_1|/|S|) H(S_1) + (|S_2|/|S|) H(S_2) \Big)$$

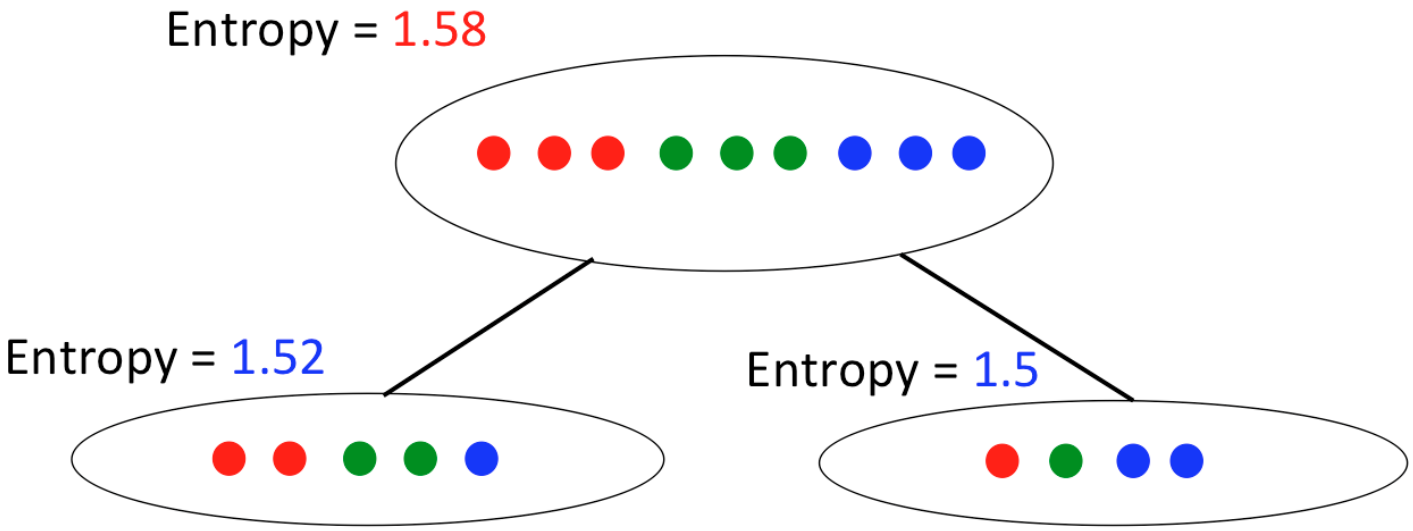
Information Gain



Averaged entropy: 2/3*1 + 1/3*0 = 0.67

Information gain: 1.58 – 0.67 = 0.91

Information Gain



Averaged entropy: 1.51

Information gain: 1.58 – 1.51 = 0.07

Splitting the node

Given the current note, how to find the best split?

For all the features and all the threshold

Compute the information gain after the split

Choose the best one (maximal information gain)

For n samples and d features: need $O(nd)$ time

Splitting the node

Given the current note, how to find the best split?

For all the features and all the threshold

- Compute the information gain after the split
- Choose the best one (maximal information gain)

For n samples and d features: need $O(nd)$ time

Splitting the node

Given the current note, how to find the best split?

For all the features and all the threshold

- Compute the information gain after the split
- Choose the best one (maximal information gain)

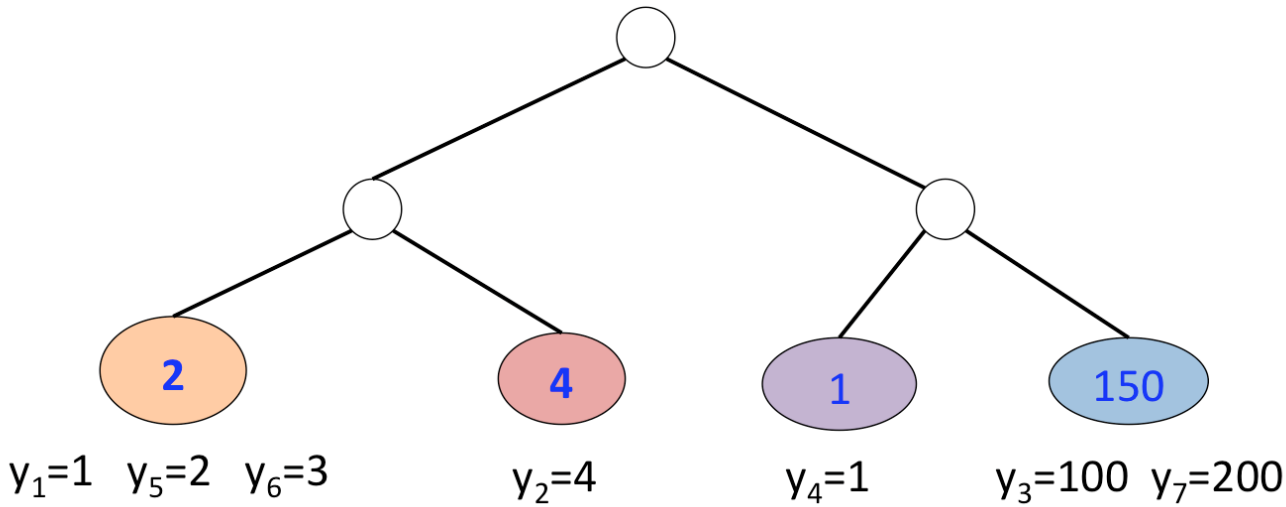
For n samples and d features: need $O(nd)$ time

Regression Tree

Assign a real number for each leaf

Usually averaged y values for each leaf

(minimize square error)



Objective function:

$$\min_F \frac{1}{n}\sum_{i=1}^{n}(y_i - F(\boldsymbol{x}_i))^2 + (\text{Regularization})$$

The quality of partition $S = S_1 \cup S_2$ can be computed by the objective function:

$$\sum_{i\in S_1}(y_i - y^{(1)})^2 + \sum_{i\in S_2}(y_i - y^{(2)})^2,$$

where $y^{(1)} = \frac{1}{|S_1|}\sum_{i\in S_1} y_i$, $y^{(2)} = \frac{1}{|S_2|}\sum_{i\in S_2} y_i$

Find the best split:

Try all the features & thresholds and find the one with minimal objective function

Objective function:

$$\min_F \frac{1}{n}\sum_{i=1}^{n}(y_i - F(\boldsymbol{x}_i))^2 + (\text{Regularization})$$

The quality of partition $S = S_1 \cup S_2$ can be computed by the objective function:

$$\sum_{i\in S_1}(y_i - y^{(1)})^2 + \sum_{i\in S_2}(y_i - y^{(2)})^2,$$

where $y^{(1)} = \frac{1}{|S_1|}\sum_{i\in S_1} y_i$, $y^{(2)} = \frac{1}{|S_2|}\sum_{i\in S_2} y_i$

Find the best split:

Try all the features & thresholds and find the one with minimal objective function

Parameters

Maximum depth: (usually ~ 10)

Minimum number of nodes in each node: (10, 50, 100)

Single decision tree is not very powerful$\cdots$

Can we build multiple decision trees and ensemble them together?

Parameters

Maximum depth: (usually ~ 10)

Minimum number of nodes in each node: (10, 50, 100)

Single decision tree is not very powerful$\cdots$

Can we build multiple decision trees and ensemble them together?

Random Forest

Random Forest

Random Forest (Bootstrap ensemble for decision trees):

- Create T trees
- Learn each tree using a subsampled dataset S_i and subsampled feature set D_i
- Prediction: Average the results from all the T trees

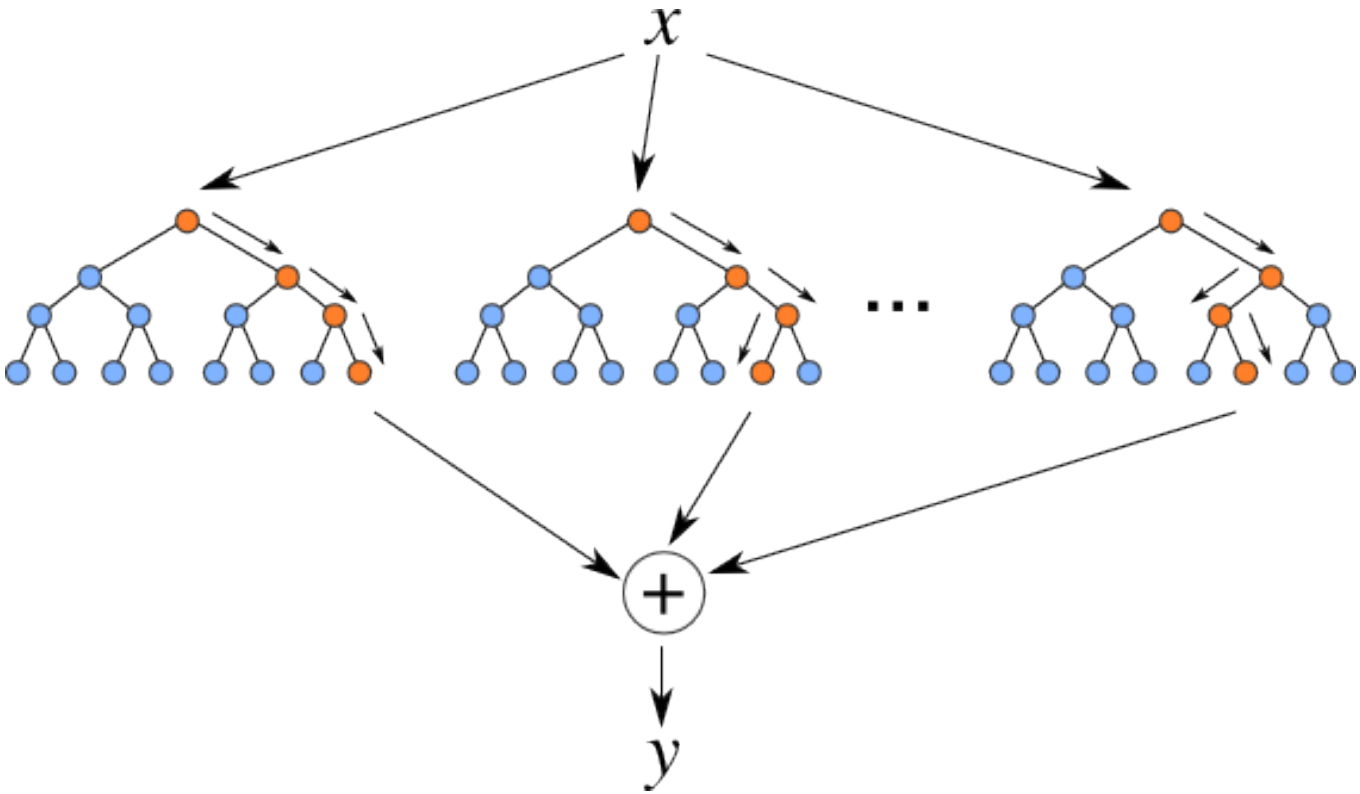
Benefit:

- Avoid over-fitting
- Improve stability and accuracy

Good software available:

- R: "randomForest" package
- Python: sklearn

Random Forest



Gradient Boosted Decision Tree

Boosted Decision Tree

Minimize loss $\ell(y, F(x))$ with $F(\cdot)$ being ensemble trees

$$F^* = \operatorname*{argmin}_F \sum_{i=1}^{n} \ell(\mathbf{y}_i, F(\boldsymbol{x}_i)) \quad \textbf{with} \quad F(\boldsymbol{x}) = \sum_{m=1}^{T} f_m(\boldsymbol{x})$$

(each f_m is a decision tree)

Direct loss minimization: at each stage m, find the best function to minimize loss

- solve $f_m = \operatorname{argmin}_{f_m} \sum_{i=1}^{N} \ell(y_i, F_{m-1}(\boldsymbol{x}_i) + f_m(\boldsymbol{x}_i))$
- update $F_m \leftarrow F_{m-1} + f_m$

$F_m(\boldsymbol{x}) = \sum_{j=1}^{m} f_j(\boldsymbol{x})$ is the prediction of $\boldsymbol{x}$ after m iterations.

Two problems:

- Hard to implement for general loss
- Tend to overfit training data

Boosted Decision Tree

Minimize loss $\ell(y, F(x))$ with $F(\cdot)$ being ensemble trees

$$F^* = \operatorname*{argmin}_F \sum_{i=1}^{n} \ell(\mathbf{y}_i, F(\boldsymbol{x}_i)) \quad \textbf{with} \quad F(\boldsymbol{x}) = \sum_{m=1}^{T} f_m(\boldsymbol{x})$$

(each f_m is a decision tree)

Direct loss minimization: at each stage m, find the best function to minimize loss

- solve $f_m = \operatorname{argmin}_{f_m} \sum_{i=1}^{N} \ell(y_i, F_{m-1}(\boldsymbol{x}_i) + f_m(\boldsymbol{x}_i))$
- update $F_m \leftarrow F_{m-1} + f_m$

$F_m(\boldsymbol{x}) = \sum_{j=1}^{m} f_j(\boldsymbol{x})$ is the prediction of $\boldsymbol{x}$ after m iterations.

Two problems:

- Hard to implement for general loss
- Tend to overfit training data

Boosted Decision Tree

Minimize loss $\ell(y, F(x))$ with $F(\cdot)$ being ensemble trees

$$F^* = \operatorname*{argmin}_F \sum_{i=1}^{n} \ell(\mathbf{y}_i, F(\boldsymbol{x}_i)) \quad \textbf{with} \quad F(\boldsymbol{x}) = \sum_{m=1}^{T} f_m(\boldsymbol{x})$$

(each f_m is a decision tree)

Direct loss minimization: at each stage m, find the best function to minimize loss

solve $f_m = \operatorname{argmin}_{f_m} \sum_{i=1}^{N} \ell(y_i, F_{m-1}(\boldsymbol{x}_i) + f_m(\boldsymbol{x}_i))$

update $F_m \leftarrow F_{m-1} + f_m$

$F_m(\boldsymbol{x}) = \sum_{j=1}^{m} f_j(\boldsymbol{x})$ is the prediction of $\boldsymbol{x}$ after m iterations.

Two problems:

Hard to implement for general loss

Tend to overfit training data

Approximate the current loss function by a quadratic approximation:

$$\sum_{i=1}^{n} \ell_i(\widehat{y}_i + f_m(\boldsymbol{x}_i)) \approx \sum_{i=1}^{n} \left(\ell_i(\widehat{y}_i) + g_i f_m(\boldsymbol{x}_i) + \frac{1}{2} h_i f_m(\boldsymbol{x}_i)^2\right)$$

$$= \sum_{i=1}^{n} \frac{h_i}{2} \| f_m(\boldsymbol{x}_i) - g_i/h_i \|^2 + \text{constant}$$

where $g_i = \partial_{\widehat{y}_i} \ell_i(\widehat{y}_i)$ is gradient,
$h_i = \partial^2_{\widehat{y}_i} \ell_i(\widehat{y}_i)$ is second order derivative

Finding $f_m(\boldsymbol{x}, \theta_m)$ by minimizing the loss function:

$$\underset{f_m}{\operatorname{argmin}} \sum_{i=1}^{N} [f_m(\boldsymbol{x}_i, \theta) - g_i/h_i]^2 + R(f_m)$$

Reduce the training of any loss function to regression tree (just need to compute g_i for different functions)
$h_i = \alpha$ (fixed step size) for original GBDT.
XGboost shows computing second order derivative yields better performance

Algorithm:

Computing the current gradient for each $\widehat{y}_i$.
Building a base learner (decision tree) to fit the gradient.
Updating current prediction $\widehat{y}_i = F_m(\boldsymbol{x}_i)$ for all i.

Finding $f_m(\boldsymbol{x}, \theta_m)$ by minimizing the loss function:

$$\underset{f_m}{\operatorname{argmin}} \sum_{i=1}^{N} [f_m(\boldsymbol{x}_i, \theta) - g_i/h_i]^2 + R(f_m)$$

Reduce the training of any loss function to regression tree (just need to compute g_i for different functions)
$h_i = \alpha$ (fixed step size) for original GBDT.
XGboost shows computing second order derivative yields better performance

Algorithm:

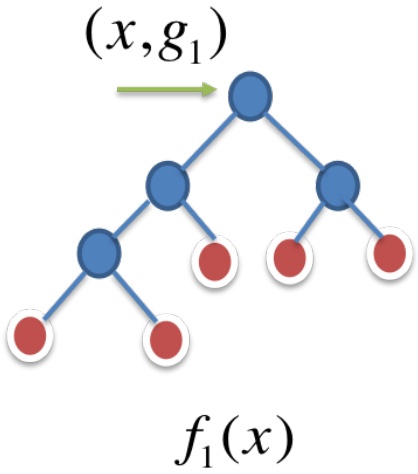
Computing the current gradient for each $\widehat{y}_i$.
Building a base learner (decision tree) to fit the gradient.
Updating current prediction $\widehat{y}_i = F_m(\boldsymbol{x}_i)$ for all i.

Gradient Boosted Decision Trees (GBDT)

Key idea:

Each base learner is a decision tree

Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$

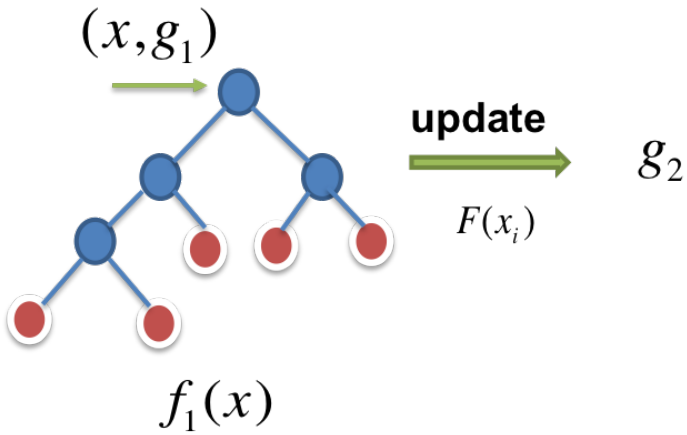


Gradient Boosted Decision Trees (GBDT)

Key idea:

Each base learner is a decision tree

Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$



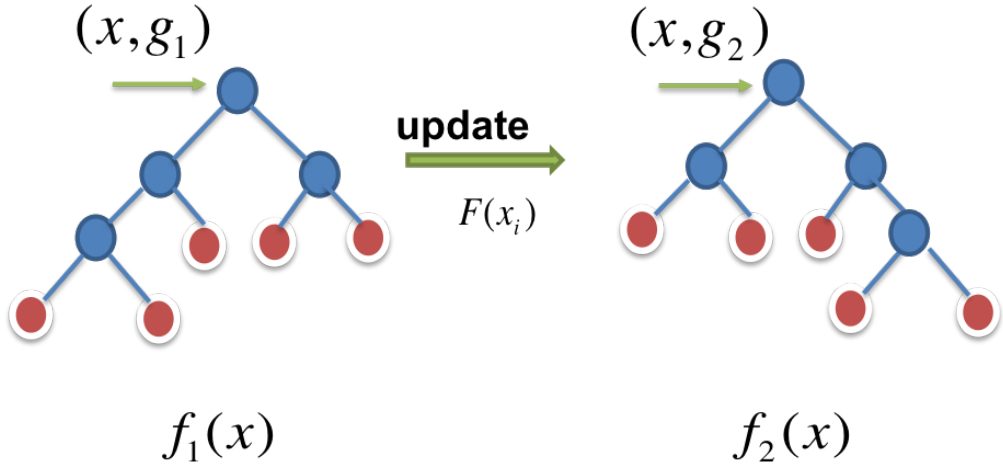
$$F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \qquad g_m(x_i) = \left. \frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} \right|_{F(x_i)=F_{m-1}(x_i)}$$

Gradient Boosted Decision Trees (GBDT)

Key idea:

Each base learner is a decision tree

Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$



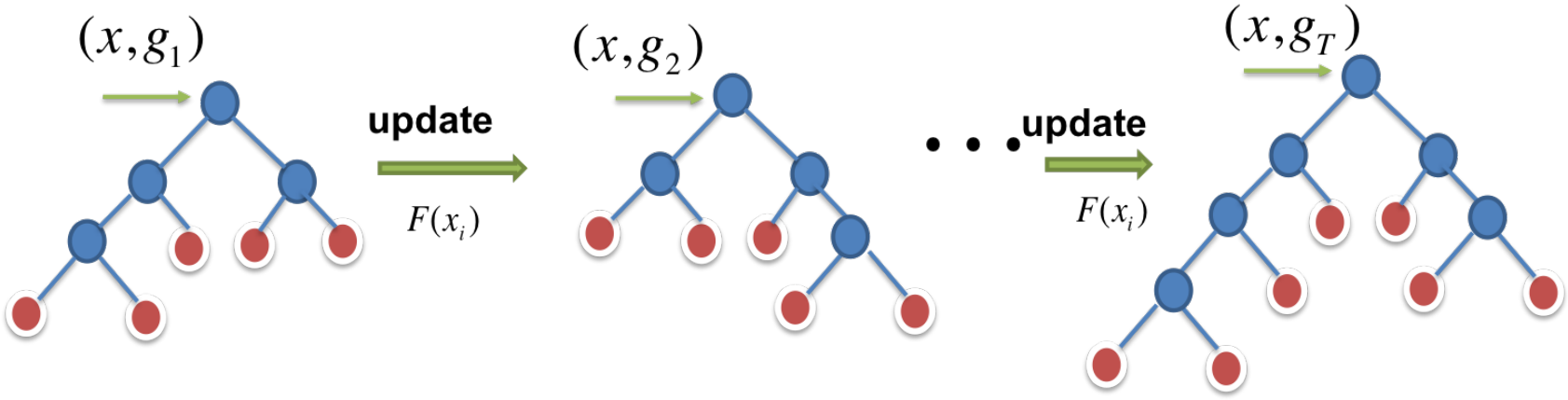
$$F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \qquad g_m(x_i) = \left. \frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} \right|_{F(x_i)=F_{m-1}(x_i)}$$

Gradient Boosted Decision Trees (GBDT)

Key idea:

Each base learner is a decision tree

Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial F}$



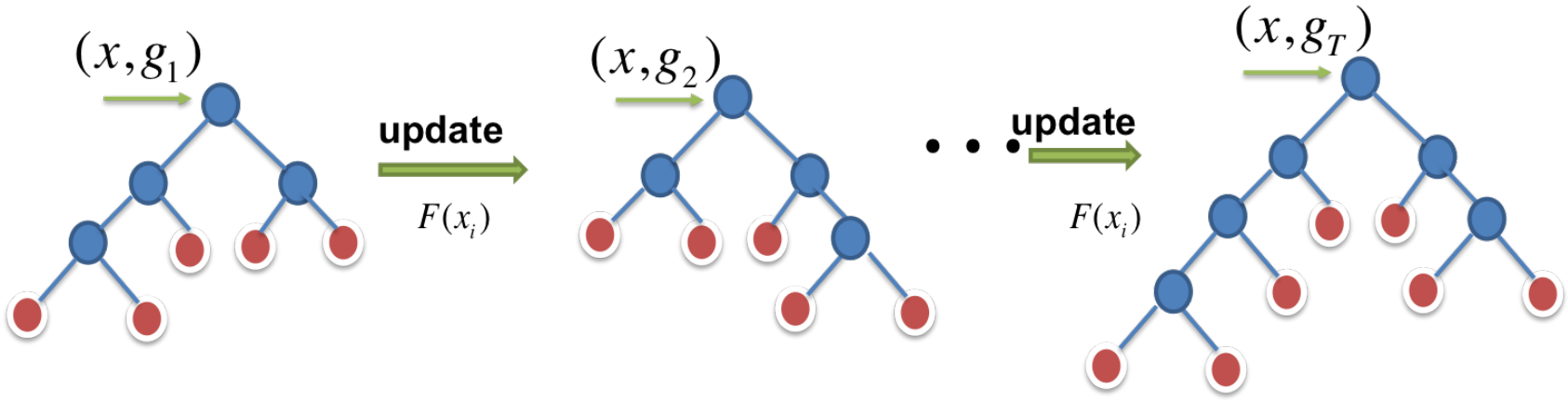
$$F_{m-1}(x_i) = \sum_{j=1}^{m-1} f_j(x_i) \quad g_m(x_i) = \left. \frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} \right|_{F(x_i)=F_{m-1}(x_i)}$$

Gradient Boosted Decision Trees (GBDT)

Key idea:

Each base learner is a decision tree

Each regression tree approximates the functional gradient $\frac{\partial \ell}{\partial f}$



Final prediction $F(x_i) = \sum_{j=1}^{T} f_j(x_i)$