

Composite Models

Many methods can be seen as:

decision tree	}	of	}	decision trees
logistic function				logistic functions
linear function				linear functions
...				kernel functions
				lower dimensional subspace
				...

E.g., neural networks, regression trees, random forest, ...

Some combinations don't help.

Consider a generalized linear model

$$\hat{Y} = f(w_0 + w_1 * F_1 * \dots * w_m * F_m)$$

Where the the features F_i come from?

- Input features.
- Boolean functions (e.g., using “and”, “or”, “equals”, “greater than”) of input features \rightarrow gradient boosted trees
- Piecewise linear functions of input features \rightarrow neural networks (with ReLU)

Boosting uses a sequence of learners where each one learns from the errors of the previous ones.

The features of a boosting algorithm are:

- There is a sequence of **base learners** e.g., small decision trees or (squashed) linear functions.
- Each learner is trained to fit the examples that the previous learners did not fit well.
- The final prediction uses a mix (e.g., sum, weighted mean, or mode) of the predictions of each learner.

The base learners can be **weak learners**.

They do not need to be very good; just better than random!

These weak learners are then boosted to be components in the ensemble that performs better than any of them.

Functional Gradient Boosting for Regression

- Hyperparameter K is the number of rounds of boosting.
- The final prediction is

$$p_0 + d_1(X) + \cdots + d_K(X)$$

where p_0 is an initial prediction e.g., mean of training data.

- The i th prediction is

$$p_i(X) = p_0 + d_1(X) + \cdots + d_i(X).$$

Then $p_i(X) = p_{i-1}(X) + d_i(X)$.

Functional Gradient Boosting for Regression (cont.)

- $p_i(X) = p_{i-1}(X) + d_i(X)$.
- Each d_i is constructed so that the error of p_i is minimal, given that p_{i-1} is fixed.
- At each stage, the base learner learns \hat{d}_i to minimize

$$\sum_e \text{loss}(p_{i-1}(e) + \hat{d}_i(e), Y(e)) = \sum_e \text{loss}(\hat{d}_i(e), Y(e) - p_{i-1}(e)).$$

for any loss based on the difference between the actual and predicted value. (Which are these?)

- The i th learner learns $d_i(e)$ to fit $Y_i(e) - p_{i-1}(e)$.
This is equivalent to learning from a modified dataset, where the previous prediction is subtracted from the actual value of the training set.
- Each learner is made to correct the errors of the previous prediction.

- 1: **procedure** *Boosting_learner*(Xs, Y, Es, L, K)
- 2: **Inputs**
- 3: Xs : set of input features; Y : target feature; Es :
training examples; L : base learner; K : number of components
in the ensemble
- 4: **Output**
- 5: function to make prediction on examples
- 6: $mean := \sum_{e \in Es} Y(e) / |Es|$
- 7: define $p_0(e) = mean$
- 8: **for each** i from 1 to K **do**
- 9: let $E_i = \{\langle Xs(e), Y(e) - p_{i-1}(e) \rangle \text{ for } e \in Es\}$
- 10: let $d_i = L(E_i)$ ▷ Learns function on examples given
 $\langle x, y \rangle$ pairs
- 11: define $p_i(e) = p_{i-1}(e) + d_i(e)$
- 12: **return** p_k

Gradient-Boosted Trees

- Gradient-boosted trees are generalized linear models. The features are binary decision trees, learned using boosting.
- For regression, the loss is regularized squared error:

$$\left(\sum_e (\hat{y}_e - y_e)^2 \right) + \sum_{k=1}^K \Omega(f_k).$$

The regularization is $\Omega(f) = \gamma * |w| + \frac{1}{2} \lambda * \sum_j w_j^2$, where w is vector of weights. γ and λ are nonnegative numbers.

- For Boolean classification, predict the sigmoid of sum of trees

$$\hat{y}_e = \text{sigmoid}\left(\sum_{k=1}^K f_k(x_e)\right)$$

Optimize sum of log loss with the same regularization:

$$\left(\sum_e \text{logloss}(\hat{y}_e, y_e) \right) + \sum_{k=1}^K \Omega(f_k).$$

Gradient-Boosted Trees

- Gradient-boosted trees, the trees are built sequentially: each tree is learned assuming the previous trees are fixed.
- Two issues:
 - ▶ Selecting leaf values
 - ▶ Selecting splits
- For regression with squared error (or any loss based on the difference between the actual and predicted value), learn a tree for the difference between data and previous prediction.

Selecting Leaf Values: Boolean Classification

- For the t th tree, optimize log loss with $L2$ regularization:

$$\hat{y}_e^{(t)} = \text{sigmoid}\left(\sum_{k=1}^t f_k(x_e)\right)$$

$$\mathcal{L}^{(t)} = \sum_e \text{logloss}(\hat{y}_e^{(t)}, y_e) + \frac{1}{2}\lambda * \sum_j w_j^2 + \text{constant}$$

- Consider j th leaf, where $I_j = \{e \mid q(x_e)=j\}$ is the set of training examples that map to it.
- Taking the derivative with respect to w_j :

$$\frac{\partial}{\partial w_j} \mathcal{L}^{(t)} = \lambda * w_j + \sum_{e \in I_j} (\hat{y}_e - y_e)$$

- A gradient descent step gives (Newton–Raphson method):

$$w_j = \frac{\sum_{e \in I_j} (y_e - \hat{y}_e^{(t-1)})}{\sum_{e \in I_j} \hat{y}_e^{(t-1)} * (1 - \hat{y}_e^{(t-1)}) + \lambda}$$