Composite Models

Many methods can be see as:

$$\left.\begin{array}{r}\text{decision tree}\\ \text{logistic function}\\ \text{linear function}\\ \ldots\end{array}\right\}\ \text{of}\ \left\{\begin{array}{l}\text{decision trees}\\ \text{logistic functions}\\ \text{linear functions}\\ \text{kernel functions}\\ \text{lower dimensional subspace}\\ \ldots\end{array}\right.$$

E.g., neural networks, regression trees, random forest, ...
Some combinations don't help.

Linear Models

Consider a generalized linear model

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

Linear Models

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?

Linear Models

Consider a generalized linear model

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

Where the the features F_i come from?

- Input features.

Linear Models

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

Linear Models

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 $\longrightarrow$ gradient boosted trees

Linear Models

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 $\longrightarrow$ gradient boosted trees
- Piecewise linear functions of input features

Linear Models

Consider a generalized linear model

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

Where the the features F_i come from?

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

Boosting

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

Boosting

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.

Boosting

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.

Boosting

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.

Boosting

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!

Boosting

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.

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.

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 ith prediction is

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

  Then $p_i(X) =$

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 ith 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.)

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

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.

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 $\widehat{d_i}$ to minimize

$$\sum_e loss(p_{i-1}(e)+\widehat{d_i}(e), Y(e)) = \sum_e loss(\widehat{d_i}(e), Y(e)-p_{i-1}(e)).$$

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

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 $\widehat{d_i}$ to minimize

  $$\sum_e loss(p_{i-1}(e)+\widehat{d_i}(e), Y(e)) = \sum_e loss(\widehat{d_i}(e), Y(e)-p_{i-1}(e)).$$

  for any loss based on the difference between the actual and predicated value. (Which are these?)
- The ith learner learns $d_i(e)$ to fit $Y_i(e) - p_{i-1}(e)$.

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 $\widehat{d_i}$ to minimize

$$\sum_e loss(p_{i-1}(e)+\widehat{d_i}(e), Y(e)) = \sum_e loss(\widehat{d_i}(e), Y(e)-p_{i-1}(e)).$$

  for any loss based on the difference between the actual and predicated value. (Which are these?)
- The ith 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.

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 $\widehat{d_i}$ to minimize

$$\sum_e loss(p_{i-1}(e)+\widehat{d_i}(e), Y(e)) = \sum_e loss(\widehat{d_i}(e), Y(e)-p_{i-1}(e)).$$

  for any loss based on the difference between the actual and predicated value. (Which are these?)
- The ith 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.

Boosting_learner

procedure *Boosting_learner*(Xs, Y, Es, L, K)
  Inputs
    Xs: set of input features; Y: target feature; Es: training examples; L: base learner; K: number of components in the ensemble
  Output
    function to make prediction on examples

Boosting_learner

procedure $\mathit{Boosting\_learner}(Xs, Y, Es, L, K)$
    Inputs
        Xs: set of input features; Y: target feature; Es: training examples; L: base learner; K: number of components in the ensemble
    Output
        function to make prediction on examples
    $mean := \sum_{e \in Es} Y(e)/|Es|$
    define $p_0(e) = mean$

Boosting_learner

procedure *Boosting_learner*(Xs, Y, Es, L, K)

  Inputs

    Xs: set of input features; Y: target feature; Es: training examples; L: base learner; K: number of components in the ensemble

  Output

    function to make prediction on examples

  $mean := \sum_{e \in Es} Y(e)/|Es|$

  define $p_0(e) = mean$

  for each i from 1 to K **do**

    let $E_i = \{\langle Xs(e), Y(e) - p_{i-1}(e)\rangle \text{ for } e \in Es\}$

Boosting_learner

procedure $Boosting\_learner(Xs, Y, Es, L, K)$

  Inputs

    Xs: set of input features; Y: target feature; Es: training examples; L: base learner; K: number of components in the ensemble

  Output

    function to make prediction on examples

  $mean := \sum_{e \in Es} Y(e)/|Es|$

  define $p_0(e) = mean$

  for each i from 1 to K **do**

    let $E_i = \{\langle Xs(e), Y(e) - p_{i-1}(e)\rangle$ for $e \in Es\}$

    let $d_i = L(E_i)$ ▷ Learns function on examples given $\langle x, y \rangle$ pairs

Boosting_learner

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

Boosting_learner

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

Gradient-Boosted Trees

- Gradient-boosted trees are generalized linear models. The features are binary decision trees, learned using boosting.

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:

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

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.

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:

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

  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

$$\widehat{y_e} = sigmoid(\sum_{k=1}^{K} f_k(x_e))$$

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:

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

  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

$$\widehat{y_e} = sigmoid(\sum_{k=1}^{K} f_k(x_e))$$

  Optimize sum of log loss with the same regularization:

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

Gradient-Boosted Trees

- Gradient-boosted trees, the tress are build sequentially: each tree is learned assuming the previous trees are fixed.

Gradient-Boosted Trees

- Gradient-boosted trees, the tress are build sequentially: each tree is learned assuming the previous trees are fixed.
- Two issues:
  - Selecting leaf values
  - Selecting splits

Gradient-Boosted Trees

- Gradient-boosted trees, the tress are build 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 predicated value), learn a tree for the difference between data and previous prediction.

Selecting Leaf Values: Boolean Classification

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

$$\widehat{y_e}^{(t)} = sigmoid(\sum_{k=1}^{t} f_k(x_e))$$

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

- Consider jth 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:

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

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

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