Machine Learning

Ensemble Methods: Boosting

FAST

DISCOVERING THE FUTURE

Topics of previous lectures

- ✓ Ingredients of Machine Learning
- ✓ Classification Basics, Basic Linear Classifier
- ✓ K-Nearest Neighbours and Naive Bayes Classifier
- ✓ Linear and Quadratic Discriminant Analysis
- ✓ Support Vector Machines (SVM)
- ✓ Decision Trees
- ✓ Ensemble Methods (Bagging, Weighted Voting, Stacking)
- ✓ Regression Methods
- ✓ Evaluation and Scoring of Classifiers

Topics of today's lecture

- Boosting
- AdaBoost
- Gradient Boosting

Motivation for Boosting

- Suppose we build M models, each with the same error ϵ, we then use the majority vote

Motivation for Boosting

- Suppose we build M models, each with the same error ϵ, we then use the majority vote
- Is it necessarily the best case if we choose the models to be independent?

Motivation for Boosting

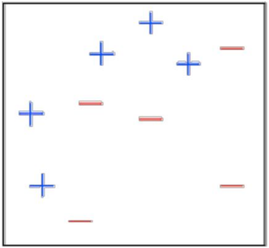
- Suppose we build M models, each with the same error ϵ, we then use the majority vote
- Is it necessarily the best case if we choose the models to be independent?
- No!

Motivation for Boosting

- Suppose we build M models, each with the same error ϵ, we then use the majority vote
- Is it necessarily the best case if we choose the models to be independent?
- No!
- The idea of boosting is that each model corrects the mistakes of the previous models in the ensemble.

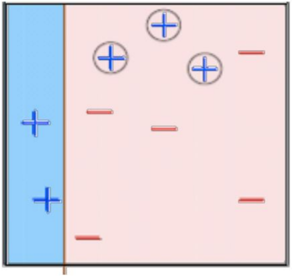
Intuition behind AdaBoost

The idea of **Boosting** is that each model corrects the weaknesses (mistakes) of the previous models in the ensemble.



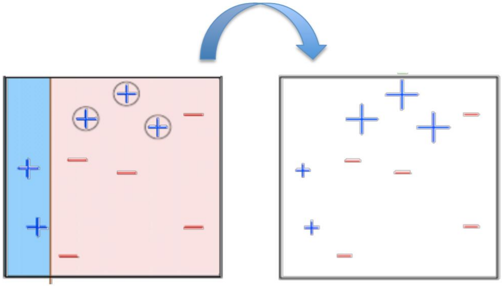
Intuition behind AdaBoost

The idea of **Boosting** is that each model corrects the weaknesses (mistakes) of the previous models in the ensemble.



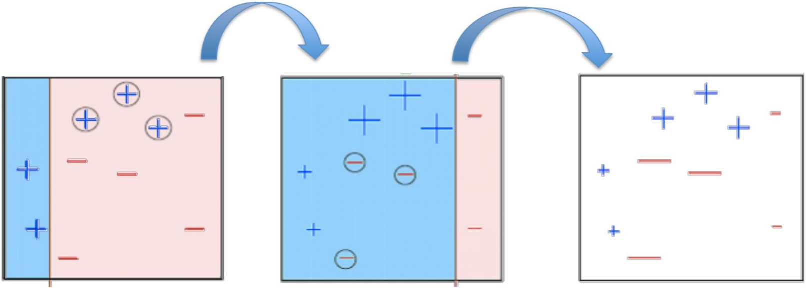
Intuition behind AdaBoost

The idea of **Boosting** is that each model corrects the weaknesses (mistakes) of the previous models in the ensemble.



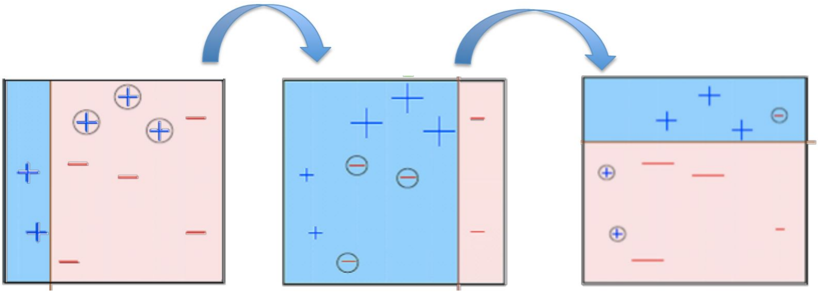
Intuition behind AdaBoost

The idea of **Boosting** is that each model corrects the weaknesses (mistakes) of the previous models in the ensemble.

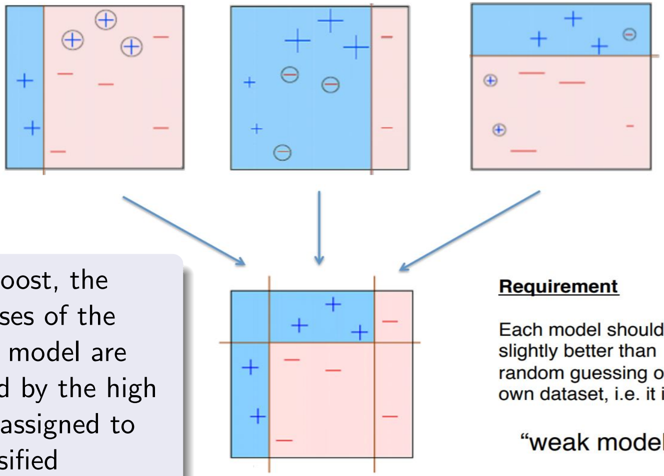


Intuition behind AdaBoost

The idea of **Boosting** is that each model corrects the weaknesses (mistakes) of the previous models in the ensemble.



Intuition behind AdaBoost



In AdaBoost, the weaknesses of the previous model are identified by the high weights assigned to missclassified instances.

Requirement

Each model should be slightly better than random guessing on its own dataset, i.e. it is a...

"weak model"

AdaBoost (**Ada**ptive **Boost**ing)

- Looks for the best model in the form of a weighted sum of weak models

AdaBoost (**Ada**ptive **Boost**ing)

- Looks for the best model in the form of a weighted sum of weak models
- Learn the first model

AdaBoost (**Ada**ptive **Boost**ing)

- Looks for the best model in the form of a weighted sum of weak models
- Learn the first model
- How to learn the second such that it would correct the errors of the first one?

AdaBoost (**Ada**ptive **Boost**ing)

- Looks for the best model in the form of a weighted sum of weak models
- Learn the first model
- How to learn the second such that it would correct the errors of the first one?
- Create a weighted dataset, where the first model would have 50% accuracy

AdaBoost (**Ada**ptive **Boost**ing)

- Looks for the best model in the form of a weighted sum of weak models
- Learn the first model
- How to learn the second such that it would correct the errors of the first one?
- Create a weighted dataset, where the first model would have 50% accuracy
- Then the second model is forced to do better than the first one

AdaBoost (**Ada**ptive **Boost**ing)

- Looks for the best model in the form of a weighted sum of weak models
- Learn the first model
- How to learn the second such that it would correct the errors of the first one?
- Create a weighted dataset, where the first model would have 50% accuracy
- Then the second model is forced to do better than the first one
- Repeat the same with successive models

AdaBoost (Freund and Schapire 1995)

Suppose that the labels are encoded as ± 1.

Algorithm AdaBoost($D, T, \mathscr{A}$) – train an ensemble of binary classifiers from reweighted training sets.

Input : training data set D; ensemble size T; learning algorithm $\mathscr{A}$.
Output : weighted ensemble of models.

```
w_{1i} ← 1/|D| for all x_i ∈ D ;                      // start with uniform weights
for t = 1 to T do
    run 𝒜 on D with weights w_{ti} to produce a model h_t;
    calculate weighted error ε_t = Σ_{i=1}^{|D|} w_{ti} I[h_t(x_i) ≠ y_i];
    if ε_t ≥ 1/2 then
        set T ← t − 1 and break
    end
    α_t ← (1/2) ln((1−ε_t)/ε_t) ;                      // confidence for this model
    w_{(t+1)i} ← w_{ti} e^{−α_t y_i h_t(x_i)} for i = 1,…,|D| ;          // update weights
    w_{(t+1)i} ← w_{(t+1)i} / Σ_{j=1}^{|D|} w_{(t+1)j} for i = 1,…,|D| ;   // renormalize weights
end
return H(x) = Σ_{t=1}^{T} α_t h_t(x)
```

AdaBoost (Freund and Schapire 1995)

Suppose that the labels are encoded as ± 1.

Algorithm AdaBoost($D, T, \mathscr{A}$) – train an ensemble of binary classifiers from reweighted training sets.

Input : training data set D; ensemble size T; learning algorithm $\mathscr{A}$.
Output : weighted ensemble of models.

$w_{1i} \leftarrow 1/|D|$ for all $\mathbf{x}_i \in D$; // start with uniform weights
for $t = 1$ to T **do**
  run $\mathscr{A}$ on D with weights w_{ti} to produce a model h_t;
  calculate weighted error $\epsilon_t = \sum_{i=1}^{|D|} w_{ti} I[h_t(\mathbf{x}_i) \neq y_i]$;
  if $\epsilon_t \geq 1/2$ **then**
    set $T \leftarrow t - 1$ and break
  end
  $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$; // confidence for this model
  $w_{(t+1)i} \leftarrow w_{ti} e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$ for $i = 1, \ldots, |D|$; // update weights
  $w_{(t+1)i} \leftarrow w_{(t+1)i} / \sum_{j=1}^{|D|} w_{(t+1)j}$ for $i = 1, \ldots, |D|$; // renormalize weights
end
return $H(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$

On a test instance AdaBoost predicts: $\hat{y} = sign(H(\mathbf{x})) = sign(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$

AdaBoost (Freund and Schapire 1995)

Suppose that the labels are encoded as ± 1.

Algorithm AdaBoost($D, T, \mathscr{A}$) – train an ensemble of binary classifiers from reweighted training sets.

Input : training data set D; ensemble size T; learning algorithm $\mathscr{A}$.
Output : weighted ensemble of models.

$w_{1i} \leftarrow 1/|D|$ for all $\mathbf{x}_i \in D$; // start with uniform weights
for $t = 1$ to T **do**
  run $\mathscr{A}$ on D with weights w_{ti} to produce a model h_t;
  calculate weighted error $\epsilon_t = \sum_{i=1}^{|D|} w_{ti} I[h_t(\mathbf{x}_i) \neq y_i]$;
  if $\epsilon_t \geq 1/2$ **then**
    set $T \leftarrow t - 1$ and break
  end
  $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$; // confidence for this model
  $w_{(t+1)i} \leftarrow \frac{w_{ti} e^{\alpha_t}}{Z_t}$ for misclassified instances $\mathbf{x}_i \in D$; // increase
  $w_{(t+1)j} \leftarrow \frac{w_{tj} e^{-\alpha_t}}{Z_t}$ for correctly classified instances $\mathbf{x}_j \in D$; // decrease
end
return $H(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$

On a test instance AdaBoost predicts: $\hat{y} = sign(H(\mathbf{x})) = sign(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}))$

Derivation of the key formulas in AdaBoost

- AdaBoost can be interpreted as greedily minimizing exponential loss function at each step $L(y, \hat{f}(\mathbf{x})) = \exp(-y\hat{f}(\mathbf{x}))$

Derivation of the key formulas in AdaBoost

- AdaBoost can be interpreted as greedily minimizing exponential loss function at each step $L(y, \hat{f}(\mathbf{x})) = \exp(-y\hat{f}(\mathbf{x}))$
- After the $(t-1)$-th iteration of the boosted classifier is a linear combination of the weak classifiers

$$H_{t-1}(\mathbf{x}_i) = \sum_{j=1}^{t-1} \alpha_j h_j(\mathbf{x}_i)$$

Derivation of the key formulas in AdaBoost

- AdaBoost can be interpreted as greedily minimizing exponential loss function at each step $L(y, \hat{f}(\mathbf{x})) = \exp(-y\hat{f}(\mathbf{x}))$
- After the $(t-1)$-th iteration of the boosted classifier is a linear combination of the weak classifiers

$$H_{t-1}(\mathbf{x}_i) = \sum_{j=1}^{t-1} \alpha_j h_j(\mathbf{x}_i)$$

- At the t-th iteration we extend it to a better classifier by adding another weak classifier h_t with weight α_t:

$$H_t(\mathbf{x}_i) = H_{t-1}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i)$$

Derivation of the key formulas in AdaBoost

- AdaBoost can be interpreted as greedily minimizing exponential loss function at each step $L(y, \hat{f}(\mathbf{x})) = \exp(-y\hat{f}(\mathbf{x}))$
- After the $(t-1)$-th iteration of the boosted classifier is a linear combination of the weak classifiers

$$H_{t-1}(\mathbf{x}_i) = \sum_{j=1}^{t-1} \alpha_j h_j(\mathbf{x}_i)$$

- At the t-th iteration we extend it to a better classifier by adding another weak classifier h_t with weight α_t:

$$H_t(\mathbf{x}_i) = H_{t-1}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i)$$

- We need to find h_t (the classifier) and α_t (its weight) that minimizes the loss

$$L = \sum_{i=1}^{N} e^{-y_i H_t(\mathbf{x}_i)}$$

Derivation of the key formulas in AdaBoost

We need to find h_t (the classifier) and α_t (its weight) that minimizes the loss

$$L = \sum_{i=1}^{N} e^{-y_i H_t(\mathbf{x}_i)} =$$

Derivation of the key formulas in AdaBoost

We need to find h_t (the classifier) and α_t (its weight) that minimizes the loss

$$L = \sum_{i=1}^{N} e^{-y_i H_t(\mathbf{x}_i)} = \sum_{i=1}^{N} \exp\{-y_i(H_{t-1}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i))\} =$$

Derivation of the key formulas in AdaBoost

We need to find h_t (the classifier) and α_t (its weight) that minimizes the loss

$$L = \sum_{i=1}^{N} e^{-y_i H_t(\mathbf{x}_i)} = \sum_{i=1}^{N} \exp\{-y_i(H_{t-1}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i))\} =$$

we denote $w_i^{(t)} = \exp\{-y_i H_{t-1}(\mathbf{x}_i)\}$ for $t > 1$ and $w_i^{(1)} = 1$

Derivation of the key formulas in AdaBoost

We need to find h_t (the classifier) and α_t (its weight) that minimizes the loss

$$L = \sum_{i=1}^{N} e^{-y_i H_t(\mathbf{x}_i)} = \sum_{i=1}^{N} \exp\{-y_i(H_{t-1}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i))\} =$$

we denote $w_i^{(t)} = \exp\{-y_i H_{t-1}(\mathbf{x}_i)\}$ for $t > 1$ and $w_i^{(1)} = 1$

$$= \sum_{i=1}^{N} w_i^{(t)} \exp\{-y_i \alpha_t h_t(\mathbf{x}_i))\} =$$

Derivation of the key formulas in AdaBoost

We need to find h_t (the classifier) and α_t (its weight) that minimizes the loss

$$L = \sum_{i=1}^{N} e^{-y_i H_t(\mathbf{x}_i)} = \sum_{i=1}^{N} \exp\{-y_i(H_{t-1}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i))\} =$$

we denote $w_i^{(t)} = \exp\{-y_i H_{t-1}(\mathbf{x}_i)\}$ for $t > 1$ and $w_i^{(1)} = 1$

$$= \sum_{i=1}^{N} w_i^{(t)} \exp\{-y_i \alpha_t h_t(\mathbf{x}_i))\} = \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} e^{-\alpha_t} + \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)} e^{\alpha_t} =$$

Derivation of the key formulas in AdaBoost

We need to find h_t (the classifier) and α_t (its weight) that minimizes the loss

$$L = \sum_{i=1}^{N} e^{-y_i H_t(\mathbf{x}_i)} = \sum_{i=1}^{N} \exp\{-y_i(H_{t-1}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i))\} =$$

we denote $w_i^{(t)} = \exp\{-y_i H_{t-1}(\mathbf{x}_i)\}$ for $t > 1$ and $w_i^{(1)} = 1$

$$= \sum_{i=1}^{N} w_i^{(t)} \exp\{-y_i \alpha_t h_t(\mathbf{x}_i))\} = \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} e^{-\alpha_t} + \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)} e^{\alpha_t} =$$

$$= \sum_{i=1}^{N} w_i^{(t)} e^{-\alpha_t} + \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)} (e^{\alpha_t} - e^{-\alpha_t})$$

Derivation of the key formulas in AdaBoost

We need to find h_t (the classifier) and α_t (its weight) that minimizes the loss

$$L = \sum_{i=1}^{N} e^{-y_i H_t(\mathbf{x}_i)} = \sum_{i=1}^{N} \exp\{-y_i(H_{t-1}(\mathbf{x}_i) + \alpha_t h_t(\mathbf{x}_i))\} =$$

we denote $w_i^{(t)} = \exp\{-y_i H_{t-1}(\mathbf{x}_i)\}$ for $t > 1$ and $w_i^{(1)} = 1$

$$= \sum_{i=1}^{N} w_i^{(t)} \exp\{-y_i \alpha_t h_t(\mathbf{x}_i))\} = \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} e^{-\alpha_t} + \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)} e^{\alpha_t} =$$

$$= \sum_{i=1}^{N} w_i^{(t)} e^{-\alpha_t} + \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)} (e^{\alpha_t} - e^{-\alpha_t})$$

Here the only part of the equation depending on h_t is the second sum and L is minimized if we choose h_t s.t. $\sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}$ is minimized (assuming $\alpha_t > 0$), that is the classifier with the lowest weighted error.

Derivation of the key formulas in AdaBoost

Now let's derive the weight α_t that minimizes the loss function.

$$L = e^{-\alpha_t} \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}$$

Derivation of the key formulas in AdaBoost

Now let's derive the weight α_t that minimizes the loss function.

$$L = e^{-\alpha_t} \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}$$

$$\frac{\partial L}{\partial \alpha_t} = -e^{-\alpha_t} \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)} = 0$$

Derivation of the key formulas in AdaBoost

Now let's derive the weight α_t that minimizes the loss function.

$$L = e^{-\alpha_t} \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}$$

$$\frac{\partial L}{\partial \alpha_t} = -e^{-\alpha_t} \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)} = 0$$

taking the logarithm from both sides gives us

Derivation of the key formulas in AdaBoost

Now let's derive the weight α_t that minimizes the loss function.

$$L = e^{-\alpha_t} \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}$$

$$\frac{\partial L}{\partial \alpha_t} = -e^{-\alpha_t} \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)} = 0$$

taking the logarithm from both sides gives us

$$-2\alpha_t = \log\left(\frac{\sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}}{\sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)}}\right) \Rightarrow \alpha_t = \frac{1}{2} \log\left(\frac{\sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)}}{\sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}}\right)$$

Derivation of the key formulas in AdaBoost

Now let's derive the weight α_t that minimizes the loss function.

$$L = e^{-\alpha_t} \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}$$

$$\frac{\partial L}{\partial \alpha_t} = -e^{-\alpha_t} \sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)} + e^{\alpha_t} \sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)} = 0$$

taking the logarithm from both sides gives us

$$-2\alpha_t = \log\left(\frac{\sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}}{\sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)}}\right) \Rightarrow \alpha_t = \frac{1}{2}\log\left(\frac{\sum_{y_i = h_t(\mathbf{x}_i)} w_i^{(t)}}{\sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}}\right)$$

since the weighted error of the weak classifier is $\epsilon_t = \frac{\sum_{y_i \neq h_t(\mathbf{x}_i)} w_i^{(t)}}{\sum_{i=1}^{N} w_i^{(t)}}$,

then $\alpha_t = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

Interpretation of AdaBoost

- At each iteration of AdaBoost, the weights are updated such that:

Interpretation of AdaBoost

- At each iteration of AdaBoost, the weights are updated such that:
  - half of total weight is on misclassified instances

Interpretation of AdaBoost

- At each iteration of AdaBoost, the weights are updated such that:
  - half of total weight is on misclassified instances
  - the other half on correctly classified instances

Interpretation of AdaBoost

- At each iteration of AdaBoost, the weights are updated such that:
  - half of total weight is on misclassified instances
  - the other half on correctly classified instances
- This ensures that the current ensemble would have weighted error 50%

Interpretation of AdaBoost

- At each iteration of AdaBoost, the weights are updated such that:
  - half of total weight is on misclassified instances
  - the other half on correctly classified instances
- This ensures that the current ensemble would have weighted error 50%
- As weak learner is expected to achieve less than 50% weighted error, it is also expected to learn something new (that the ensemble does not "know" yet)

AdaBoost for regression (AdaBoost.R2 (Drucker, 1997))

Input : training data set D; ensemble size T ; learning algorithm A
Output : weighted ensemble of models

1. Start with uniform weights $w_i^{(1)} = 1/|D| \, \forall \mathbf{x}_i \in D$

AdaBoost for regression (AdaBoost.R2 (Drucker, 1997))

Input : training data set D; ensemble size T ; learning algorithm A

Output : weighted ensemble of models

1. Start with uniform weights $w_i^{(1)} = 1/|D| \, \forall \mathbf{x}_i \in D$
2. For $t = 1, \ldots, T$ do

AdaBoost for regression (AdaBoost.R2 (Drucker, 1997))

Input : training data set D; ensemble size T ; learning algorithm A
Output : weighted ensemble of models

1 Start with uniform weights $w_i^{(1)} = 1/|D| \, \forall \mathbf{x}_i \in D$
2 For $t = 1, \ldots, T$ do
    3 Run A on D with weights $w_i^{(t)}$ to produce a model h_t ;

AdaBoost for regression (AdaBoost.R2 (Drucker, 1997))

Input : training data set D; ensemble size T ; learning algorithm A

Output : weighted ensemble of models

1. Start with uniform weights $w_i^{(1)} = 1/|D| \, \forall \mathbf{x}_i \in D$
2. For $t = 1, \ldots, T$ do
   3. Run A on D with weights $w_i^{(t)}$ to produce a model h_t ;
   4. Calculate weighted error

      let $Z_t = \max_{j \in \{1, \ldots, |D|\}} |y_j - h_t(x_j)|$

      $e_i^{(t)} = \frac{|y_i - h_t(x_i)|}{Z_t}$

      $\epsilon_t = \sum_{i=1}^{|D|} w_i^{(t)} e_i^{(t)}$

AdaBoost for regression (AdaBoost.R2 (Drucker, 1997))

Input : training data set D; ensemble size T ; learning algorithm A

Output : weighted ensemble of models

1. Start with uniform weights $w_i^{(1)} = 1/|D| \, \forall \mathbf{x}_i \in D$
2. For $t = 1, \ldots, T$ do
   3. Run A on D with weights $w_i^{(t)}$ to produce a model h_t ;
   4. Calculate weighted error
      let $Z_t = \max_{j \in \{1,\ldots,|D|\}} |y_j - h_t(x_j)|$
      $e_i^{(t)} = \frac{|y_i - h_t(x_i)|}{Z_t}$
      $\epsilon_t = \sum_{i=1}^{|D|} w_i^{(t)} e_i^{(t)}$
   5. If $\epsilon_t \geq \frac{1}{2}$ then set $t = T - 1$ and break

AdaBoost for regression (AdaBoost.R2 (Drucker, 1997))

Input : training data set D; ensemble size T ; learning algorithm A

Output : weighted ensemble of models

1. Start with uniform weights $w_i^{(1)} = 1/|D| \, \forall \mathbf{x}_i \in D$
2. For $t = 1, \ldots, T$ do
    3. Run A on D with weights $w_i^{(t)}$ to produce a model h_t ;
    4. Calculate weighted error

       let $Z_t = \max_{j \in \{1,\ldots,|D|\}} |y_j - h_t(x_j)|$

       $e_i^{(t)} = \frac{|y_i - h_t(x_i)|}{Z_t}$

       $\epsilon_t = \sum_{i=1}^{|D|} w_i^{(t)} e_i^{(t)}$
    5. If $\epsilon_t \geq \frac{1}{2}$ then set $t = T - 1$ and break
    6. Let $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

AdaBoost for regression (AdaBoost.R2 (Drucker, 1997))

Input : training data set D; ensemble size T ; learning algorithm A
Output : weighted ensemble of models

1. Start with uniform weights $w_i^{(1)} = 1/|D| \, \forall \mathbf{x}_i \in D$
2. For $t = 1, \ldots, T$ do
   3. Run A on D with weights $w_i^{(t)}$ to produce a model h_t ;
   4. Calculate weighted error
      let $Z_t = \max_{j \in \{1,\ldots,|D|\}} |y_j - h_t(x_j)|$
      $$e_i^{(t)} = \frac{|y_i - h_t(x_i)|}{Z_t}$$
      $\epsilon_t = \sum_{i=1}^{|D|} w_i^{(t)} e_i^{(t)}$
   5. If $\epsilon_t \geq \frac{1}{2}$ then set $t = T - 1$ and break
   6. Let $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
   7. Update the weights $w_i^{(t+1)} = w_i^{(t)} \beta_t^{1-e_i^{(t)}}$

AdaBoost for regression (AdaBoost.R2 (Drucker, 1997))

Input : training data set D; ensemble size T ; learning algorithm A

Output : weighted ensemble of models

1. Start with uniform weights $w_i^{(1)} = 1/|D| \, \forall \mathbf{x}_i \in D$
2. For $t = 1, \ldots, T$ do
   3. Run A on D with weights $w_i^{(t)}$ to produce a model h_t ;
   4. Calculate weighted error

      let $Z_t = \max_{j \in \{1,\ldots,|D|\}} |y_j - h_t(x_j)|$

      $e_i^{(t)} = \frac{|y_i - h_t(x_i)|}{Z_t}$

      $\epsilon_t = \sum_{i=1}^{|D|} w_i^{(t)} e_i^{(t)}$
   5. If $\epsilon_t \geq \frac{1}{2}$ then set $t = T - 1$ and break
   6. Let $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
   7. Update the weights $w_i^{(t+1)} = w_i^{(t)} \beta_t^{1-e_i^{(t)}}$
   8. Normalize the weights $w_i^{(t+1)} = \frac{w_i^{(t+1)}}{\sum_{i=1}^{|D|} w_i^{(t+1)}}$

AdaBoost for regression (AdaBoost.R2 (Drucker, 1997))

Input : training data set D; ensemble size T ; learning algorithm A

Output : weighted ensemble of models

1 Start with uniform weights $w_i^{(1)} = 1/|D| \, \forall \mathbf{x}_i \in D$

2 For $t = 1, \ldots, T$ do

 3 Run A on D with weights $w_i^{(t)}$ to produce a model h_t ;

 4 Calculate weighted error

 let $Z_t = \max_{j \in \{1,\ldots,|D|\}} |y_j - h_t(x_j)|$

 $e_i^{(t)} = \frac{|y_i - h_t(x_i)|}{Z_t}$

 $\epsilon_t = \sum_{i=1}^{|D|} w_i^{(t)} e_i^{(t)}$

 5 If $\epsilon_t \geq \frac{1}{2}$ then set $t = T - 1$ and break

 6 Let $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

 7 Update the weights $w_i^{(t+1)} = w_i^{(t)} \beta_t^{1-e_i^{(t)}}$

 8 Normalize the weights $w_i^{(t+1)} = \frac{w_i^{(t+1)}}{\sum_{i=1}^{|D|} w_i^{(t+1)}}$

9 Return the weighted median of $h_t(x)$ for $t = 1, \ldots, T$, using $\ln(\frac{1}{\beta_t})$ as weights

AdaBoost.R2

The errors in step 4 are usually calculated using the following loss functions

- $e_i^{(t)} = \dfrac{|y_i - h_t(x_i)|}{Z_t}$ (linear)

AdaBoost.R2

The errors in step 4 are usually calculated using the following loss functions

- $e_i^{(t)} = \dfrac{|y_i - h_t(x_i)|}{Z_t}$ (linear)
- $e_i^{(t)} = \dfrac{(y_i - h_t(x_i))^2}{Z_t^2}$ (square)

AdaBoost.R2

The errors in step 4 are usually calculated using the following loss functions

- $e_i^{(t)} = \dfrac{|y_i - h_t(x_i)|}{Z_t}$ (linear)
- $e_i^{(t)} = \dfrac{(y_i - h_t(x_i))^2}{Z_t^2}$ (square)
- $e_i^{(t)} = 1 - \exp\left\{ -\dfrac{|y_i - h_t(x_i)|}{Z_t} \right\}$ (exponential)

AdaBoost.R2

The errors in step 4 are usually calculated using the following loss functions

- $e_i^{(t)} = \dfrac{|y_i - h_t(x_i)|}{Z_t}$ (linear)
- $e_i^{(t)} = \dfrac{(y_i - h_t(x_i))^2}{Z_t^2}$ (square)
- $e_i^{(t)} = 1 - \exp\left\{ - \dfrac{|y_i - h_t(x_i)|}{Z_t} \right\}$ (exponential)

where $Z_t = \max\limits_{j \in \{1,\ldots,|D|\}} |y_j - h_t(x_j)|$

Motivation for Gradient Boosting

- Suppose in some competition, you are given a set of data points for a regression task $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where $(x_i, y_i) \in \mathbb{R}^2$

Motivation for Gradient Boosting

- Suppose in some competition, you are given a set of data points for a regression task $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where $(x_i, y_i) \in \mathbb{R}^2$
- You need to fit a model $\hat{f}(x)$ in order to minimize the squared loss

Motivation for Gradient Boosting

- Suppose in some competition, you are given a set of data points for a regression task $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where $(x_i, y_i) \in \mathbb{R}^2$
- You need to fit a model $\hat{f}(x)$ in order to minimize the squared loss
- Someone gives you a candidate model for $\hat{f}(x)$, which is good, but not perfect

Motivation for Gradient Boosting

- Suppose in some competition, you are given a set of data points for a regression task $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where $(x_i, y_i) \in \mathbb{R}^2$
- You need to fit a model $\hat{f}(x)$ in order to minimize the squared loss
- Someone gives you a candidate model for $\hat{f}(x)$, which is good, but not perfect
- For example, the suggested model gives the following results

$$\hat{f}(x_1) = 0.3, \text{ when } y_1 = 0.4$$

$$\hat{f}(x_2) = 1.6, \text{ when } y_2 = 1.5$$

and so on

Motivation for Gradient Boosting

- Suppose in some competition, you are given a set of data points for a regression task $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where $(x_i, y_i) \in \mathbb{R}^2$
- You need to fit a model $\hat{f}(x)$ in order to minimize the squared loss
- Someone gives you a candidate model for $\hat{f}(x)$, which is good, but not perfect
- For example, the suggested model gives the following results

$$\hat{f}(x_1) = 0.3, \text{ when } y_1 = 0.4$$

$$\hat{f}(x_2) = 1.6, \text{ when } y_2 = 1.5$$

  and so on
- Your task is to improve the performance of this model, but

Motivation for Gradient Boosting

- Suppose in some competition, you are given a set of data points for a regression task $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where $(x_i, y_i) \in \mathbb{R}^2$
- You need to fit a model $\hat{f}(x)$ in order to minimize the squared loss
- Someone gives you a candidate model for $\hat{f}(x)$, which is good, but not perfect
- For example, the suggested model gives the following results

$$\hat{f}(x_1) = 0.3, \text{ when } y_1 = 0.4$$

$$\hat{f}(x_2) = 1.6, \text{ when } y_2 = 1.5$$

  and so on
- Your task is to improve the performance of this model, but
  - you are not allowed to change anything in the given $\hat{f}(x)$

Motivation for Gradient Boosting

- Suppose in some competition, you are given a set of data points for a regression task $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where $(x_i, y_i) \in \mathbb{R}^2$
- You need to fit a model $\hat{f}(x)$ in order to minimize the squared loss
- Someone gives you a candidate model for $\hat{f}(x)$, which is good, but not perfect
- For example, the suggested model gives the following results

$$\hat{f}(x_1) = 0.3, \text{ when } y_1 = 0.4$$

$$\hat{f}(x_2) = 1.6, \text{ when } y_2 = 1.5$$

  and so on
- Your task is to improve the performance of this model, but
  - you are not allowed to change anything in the given $\hat{f}(x)$
  - you can only add an additional model h to $\hat{f}(x)$, so that the new prediction is $\hat{f}(x) + h(x)$

Motivation for Gradient Boosting

- Suppose in some competition, you are given a set of data points for a regression task $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, where $(x_i, y_i) \in \mathbb{R}^2$
- You need to fit a model $\hat{f}(x)$ in order to minimize the squared loss
- Someone gives you a candidate model for $\hat{f}(x)$, which is good, but not perfect
- For example, the suggested model gives the following results

$$\hat{f}(x_1) = 0.3, \text{ when } y_1 = 0.4$$

$$\hat{f}(x_2) = 1.6, \text{ when } y_2 = 1.5$$

  and so on
- Your task is to improve the performance of this model, but
  - you are not allowed to change anything in the given $\hat{f}(x)$
  - you can only add an additional model h to $\hat{f}(x)$, so that the new prediction is $\hat{f}(x) + h(x)$
- How would you improve the performance of the initial model?

Motivation for Gradient Boosting

You want to improve the model such that

$$\hat{f}(x_1) + h(x_1) = y_1$$
$$\hat{f}(x_2) + h(x_2) = y_2$$
$$\dots$$
$$\hat{f}(x_n) + h(x_n) = y_n$$

Motivation for Gradient Boosting

You want to improve the model such that

$$\hat{f}(x_1) + h(x_1) = y_1$$
$$\hat{f}(x_2) + h(x_2) = y_2$$
$$\dots$$
$$\hat{f}(x_n) + h(x_n) = y_n$$

Or equivalently

$$h(x_1) = y_1 - \hat{f}(x_1)$$
$$h(x_2) = y_2 - \hat{f}(x_2)$$
$$\dots$$
$$h(x_n) = y_n - \hat{f}(x_n)$$

Motivation for Gradient Boosting

You want to improve the model such that

$$\hat{f}(x_1) + h(x_1) = y_1$$
$$\hat{f}(x_2) + h(x_2) = y_2$$
$$\dots$$
$$\hat{f}(x_n) + h(x_n) = y_n$$

Or equivalently

$$h(x_1) = y_1 - \hat{f}(x_1)$$
$$h(x_2) = y_2 - \hat{f}(x_2)$$
$$\dots$$
$$h(x_n) = y_n - \hat{f}(x_n)$$

Can any regression tree h help to achieve this goal approximately?

Motivation for Gradient Boosting

You want to improve the model such that

$$\hat{f}(x_1) + h(x_1) = y_1$$
$$\hat{f}(x_2) + h(x_2) = y_2$$
$$\dots$$
$$\hat{f}(x_n) + h(x_n) = y_n$$

Or equivalently

$$h(x_1) = y_1 - \hat{f}(x_1)$$
$$h(x_2) = y_2 - \hat{f}(x_2)$$
$$\dots$$
$$h(x_n) = y_n - \hat{f}(x_n)$$

Can any regression tree h help to achieve this goal approximately?
We can fit a regression tree to data
$(x_1, y_1 - F(x_1)), (x_2, y_2 - F(x_2)), \dots, (x_n, y_n - F(x_n))$

Motivation for Gradient Boosting

- The strategy is to use a regression tree to learn the **residuals** of the initial model

Motivation for Gradient Boosting

- The strategy is to use a regression tree to learn the **residuals** of the initial model
- What if results from the new model $\hat{f}(x) + h(x)$ are still not satisfactory?

Motivation for Gradient Boosting

- The strategy is to use a regression tree to learn the **residuals** of the initial model
- What if results from the new model $\hat{f}(x) + h(x)$ are still not satisfactory?
- We can add another regression tree

Motivation for Gradient Boosting

- The strategy is to use a regression tree to learn the **residuals** of the initial model
- What if results from the new model $\hat{f}(x) + h(x)$ are still not satisfactory?
- We can add another regression tree
- In this way we can improve the predictions from individual models

Gradient Boosting (J. Friedman, 2000)

- It is also an ensemble of weak learners (typically decision trees)

Gradient Boosting (J. Friedman, 2000)

- It is also an ensemble of weak learners (typically decision trees)
- Similar to AdaBoost, in each stage a weak learner is introduced to compensate the weaknesses of existing weak learners

Gradient Boosting (J. Friedman, 2000)

- It is also an ensemble of weak learners (typically decision trees)
- Similar to AdaBoost, in each stage a weak learner is introduced to compensate the weaknesses of existing weak learners
- Here the weaknesses are identified by **gradients**

Gradient Boosting (J. Friedman, 2000)

- It is also an ensemble of weak learners (typically decision trees)
- Similar to AdaBoost, in each stage a weak learner is introduced to compensate the weaknesses of existing weak learners
- Here the weaknesses are identified by **gradients**
- It is a generalization of AdaBoost that supports various loss function

Gradient Boosting (J. Friedman, 2000)

- It is also an ensemble of weak learners (typically decision trees)
- Similar to AdaBoost, in each stage a weak learner is introduced to compensate the weaknesses of existing weak learners
- Here the weaknesses are identified by **gradients**
- It is a generalization of AdaBoost that supports various loss function
- It can be used for regression, classification and ranking purposes

Gradient Boosting for Regression

- How is the previous example related to gradient descent?

Gradient Boosting for Regression

- How is the previous example related to gradient descent?
- Let's formulate the regression problem in terms of the squared loss function

$$L(y, \hat{f}(x)) = \frac{(y - \hat{f}(x))^2}{2}$$

Gradient Boosting for Regression

- How is the previous example related to gradient descent?
- Let's formulate the regression problem in terms of the squared loss function

$$L(y, \hat{f}(x)) = \frac{(y - \hat{f}(x))^2}{2}$$

- We want to minimize $J = \sum_i^n L(y_i, \hat{f}(x_i))$ with respect to $\hat{f}(x_1), \ldots, \hat{f}(x_n)$

Gradient Boosting for Regression

- How is the previous example related to gradient descent?
- Let's formulate the regression problem in terms of the squared loss function

$$L(y, \hat{f}(x)) = \frac{(y - \hat{f}(x))^2}{2}$$

- We want to minimize $J = \sum_i^n L(y_i, \hat{f}(x_i))$ with respect to $\hat{f}(x_1), \ldots, \hat{f}(x_n)$
- Let's take derivates with respect to $\hat{f}(x_i)$

$$\frac{\partial J}{\partial \hat{f}(x_i)} = \frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}(x_i)} = \hat{f}(x_i) - y_i$$

Gradient Boosting for Regression

- How is the previous example related to gradient descent?
- Let's formulate the regression problem in terms of the squared loss function

$$L(y, \hat{f}(x)) = \frac{(y - \hat{f}(x))^2}{2}$$

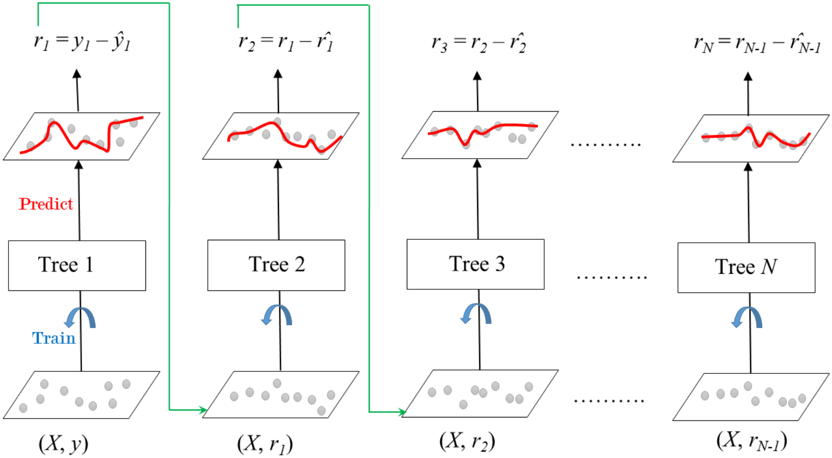
- We want to minimize $J = \sum_i^n L(y_i, \hat{f}(x_i))$ with respect to $\hat{f}(x_1), \ldots, \hat{f}(x_n)$
- Let's take derivates with respect to $\hat{f}(x_i)$

$$\frac{\partial J}{\partial \hat{f}(x_i)} = \frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}(x_i)} = \hat{f}(x_i) - y_i$$

- The residuals are the negative gradients

$$y_i - \hat{f}(x_i) = -\frac{\partial J}{\partial \hat{f}(x_i)}$$

Gradient Boosting



Gradient Boosting for Regression (with squared loss)

Input : training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$; ensemble size M, loss function $L(y, f(x)) = (y - f(x))^2$, learning rate α

Output : ensemble of models

1. Start with a model with constant value $\hat{f}_0(x) = \operatorname{argmin}_f \sum_{i=1}^n L(y_i, f) = \frac{1}{n} \sum_{i=1}^n y_i$

Gradient Boosting for Regression (with squared loss)

Input : training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$; ensemble size M, loss function $L(y, f(x)) = (y - f(x))^2$, learning rate α

Output : ensemble of models

1. Start with a model with constant value $\hat{f}_0(x) = \text{argmin}_f \sum_{i=1}^{n} L(y_i, f) = \frac{1}{n} \sum_{i=1}^{n} y_i$
2. For $m = 1, \ldots, M$ do

Gradient Boosting for Regression (with squared loss)

Input : training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$; ensemble size M, loss function $L(y, f(x)) = (y - f(x))^2$, learning rate α
Output : ensemble of models

1. Start with a model with constant value $\hat{f}_0(x) = \text{argmin}_f \sum_{i=1}^{n} L(y_i, f) = \frac{1}{n} \sum_{i=1}^{n} y_i$
2. For $m = 1, \ldots, M$ do
    1. Compute the **pseudo-residuals**:

$$r_{im} = -\Big(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}(x_i)}\Big)_{\hat{f}(x)=\hat{f}_{m-1}(x)}, \; i = 1, \ldots, n$$

Gradient Boosting for Regression (with squared loss)

Input : training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$; ensemble size M, loss function $L(y, f(x)) = (y - f(x))^2$, learning rate α

Output : ensemble of models

1. Start with a model with constant value $\hat{f}_0(x) = \text{argmin}_f \sum_{i=1}^n L(y_i, f) = \frac{1}{n} \sum_{i=1}^n y_i$
2. For $m = 1, \ldots, M$ do
    1. Compute the **pseudo-residuals**:

$$r_{im} = -\Big(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}(x_i)}\Big)_{\hat{f}(x) = \hat{f}_{m-1}(x)}, \; i = 1, \ldots, n$$

    2. Fit a model (typically a weak learner) $h_m(x)$ to pseudo-residuals $\{(\mathbf{x}_i, r_{im})\}_{i=1}^n$

Gradient Boosting for Regression (with squared loss)

Input : training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$; ensemble size M, loss function $L(y, f(x)) = (y - f(x))^2$, learning rate α

Output : ensemble of models

1 Start with a model with constant value $\hat{f}_0(x) = \text{argmin}_f \sum_{i=1}^n L(y_i, f) = \frac{1}{n}\sum_{i=1}^n y_i$

2 For $m = 1, \ldots, M$ do

 1 Compute the **pseudo-residuals**:

$$r_{im} = -\Big(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}(x_i)}\Big)_{\hat{f}(x)=\hat{f}_{m-1}(x)}, \, i = 1, \ldots, n$$

 2 Fit a model (typically a weak learner) $h_m(x)$ to pseudo-residuals $\{(\mathbf{x}_i, r_{im})\}_{i=1}^n$

 3 Update the model:

$$\hat{f}_m = \hat{f}_{m-1}(x) + \alpha h_m(x)$$

Gradient Boosting for Regression (with squared loss)

Input : training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$; ensemble size M, loss function $L(y, f(x)) = (y - f(x))^2$, learning rate α

Output : ensemble of models

1. Start with a model with constant value $\hat{f}_0(x) = \text{argmin}_f \sum_{i=1}^n L(y_i, f) = \frac{1}{n}\sum_{i=1}^n y_i$
2. For $m = 1, \ldots, M$ do
   1. Compute the **pseudo-residuals**:
      $$r_{im} = -\Big(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}(x_i)}\Big)_{\hat{f}(x)=\hat{f}_{m-1}(x)}, \; i = 1, \ldots, n$$
   2. Fit a model (typically a weak learner) $h_m(x)$ to pseudo-residuals $\{(\mathbf{x}_i, r_{im})\}_{i=1}^n$
   3. Update the model:
      $$\hat{f}_m = \hat{f}_{m-1}(x) + \alpha h_m(x)$$
3. Return $\hat{f}_M(x)$.

Gradient Boosting for Regression (with arbitrary loss)

Input : training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$; ensemble size M, loss function $L(y, f(x))$, learning rate α

Output : ensemble of models

1. Start with a model with constant value $\hat{f}_0(x) = \text{argmin}_f \sum_{i=1}^n L(y_i, f)$
2. For $m = 1, \ldots, M$ do
   1. Compute the **pseudo-residuals**:

      $$r_{im} = -\Big(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}(x_i)}\Big)_{\hat{f}(x)=\hat{f}_{m-1}(x)}, \; i = 1, \ldots, n$$

   2. Fit a model (typically a weak learner) $h_m(x)$ to pseudo-residuals $\{(\mathbf{x}_i, r_{im})\}_{i=1}^n$
   3. Update the model:

      $$\hat{f}_m = \hat{f}_{m-1}(x) + \alpha h_m(x)$$

3. Return $\hat{f}_M(x)$.

Gradient Boosting for Regression (with arbitrary loss)

Input : training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$; ensemble size M, loss function $L(y, f(x))$, learning rate α

> The learning rate for each update may also be optimized inside the algorithm

Output : ensemble of mode

1. Start with a model wit
   $\hat{f}_0(x) = \text{argmin}_f \sum_{i=1}^n L(y_i, f)$
2. For $m = 1, \ldots, M$ do
   1. Compute the **pseudo-residuals**:

   $$r_{im} = -\left(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}(x_i)}\right)_{\hat{f}(x)=\hat{f}_{m-1}(x)}, \; i = 1, \ldots, n$$

   2. Fit a model (typically a weak learner) $h_m(x)$ to pseudo-residuals $\{(\mathbf{x}_i, r_{im})\}_{i=1}^n$
   3. Update the model:

   $$\hat{f}_m = \hat{f}_{m-1}(x) + \alpha h_m(x)$$

3. Return $\hat{f}_M(x)$.

Gradient Boosting for Regression (with arbitrary loss)

Input : training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$; ensemble size M, loss function $L(y, f(x))$, learning rate α

Output : ensemble of models

1. Start with a model with constant value $\hat{f}_0(x) = \text{argmin}_f \sum_{i=1}^n L(y_i, f)$
2. For $m = 1, \ldots, M$ do
    1. Compute the **pseudo-residuals**:

    $$r_{im} = -\Big(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}(x_i)}\Big)_{\hat{f}(x)=\hat{f}_{m-1}(x)}, \; i = 1, \ldots, n$$

    2. Fit a model (typically a weak learner) $h_m(x)$ to pseudo-residuals $\{(\mathbf{x}_i, r_{im})\}_{i=1}^n$
    3. Compute the learning rate α_m:

    $$\alpha_m = \underset{\alpha}{\text{argmin}} \sum_{i=1}^n L(y_i, \hat{f}_{m-1}(x_i) + \alpha h_m(x_i))$$

3. Return $\hat{f}_M(x)$.

Gradient Boosting for Regression (with arbitrary loss)

Input : training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$; ensemble size M, loss function $L(y, f(x))$, learning rate α

Output : ensemble of models

1. Start with a model with constant value $\hat{f}_0(x) = \text{argmin}_f \sum_{i=1}^n L(y_i, f)$
2. For $m = 1, \ldots, M$ do
    1. Compute the **pseudo-residuals**:

    $$r_{im} = -\Big(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}(x_i)}\Big)_{\hat{f}(x)=\hat{f}_{m-1}(x)}, \; i = 1, \ldots, n$$

    2. Fit a model (typically a weak learner) $h_m(x)$ to pseudo-residuals $\{(\mathbf{x}_i, r_{im})\}_{i=1}^n$
    3. Compute the learning rate α_m:

    $$\alpha_m = \underset{\alpha}{\text{argmin}} \sum_{i=1}^n L(y_i, \hat{f}_{m-1}(x_i) + \alpha h_m(x_i))$$

    4. Update the model: $\hat{f}_m = \hat{f}_{m-1}(x) + \alpha_m h_m(x)$
3. Return $\hat{f}_M(x)$.

Gradient Boosting for Classification

- Suppose we have a multi-class classification problem with K labels

Gradient Boosting for Classification

- Suppose we have a multi-class classification problem with K labels
- In case of classification, we need to introduce K models in our gradient boosting machine in each iteration

Gradient Boosting for Classification

- Suppose we have a multi-class classification problem with K labels
- In case of classification, we need to introduce K models in our gradient boosting machine in each iteration
- The probability $\mathbb{P}(Y_i = k|X = \mathbf{x}_i)$ is modeled as a softmax of the final outcomes of the K functions

Gradient Boosting for Classification

Input : training data set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ (1-hot encoded $\mathbf{y}$); ensemble size M, loss function $L(y, f(x)) = -\sum_{j=1}^K y_j \log(f^{(j)}(x))$, learning rate α
Output : ensemble of models

1. Initialize the models with prior probability values $\hat{f}^{(j)}(x) = \frac{1}{n}\sum_{i=1}^n \mathbf{y}_i$ for $j = 1, \ldots, K$

Gradient Boosting for Classification

Input : training data set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ (1-hot encoded $\mathbf{y}$); ensemble size M, loss function $L(y, f(x)) = -\sum_{j=1}^K y_j \log(f^{(j)}(x))$, learning rate α
Output : ensemble of models

1. Initialize the models with prior probability values $\hat{f}^{(j)}(x) = \frac{1}{n} \sum_{i=1}^n \mathbf{y}_i$ for $j = 1, \ldots, K$
2. For $m = 1, \ldots, M$ do

Gradient Boosting for Classification

Input : training data set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ (1-hot encoded $\mathbf{y}$); ensemble size M, loss function $L(y, f(x)) = -\sum_{j=1}^K y_j \log(f^{(j)}(x))$, learning rate α

Output : ensemble of models

1. Initialize the models with prior probability values $\hat{f}^{(j)}(x) = \frac{1}{n}\sum_{i=1}^n \mathbf{y}_i$ for $j = 1, \ldots, K$
2. For $m = 1, \ldots, M$ do
    1. Compute the **pseudo-residuals**:

$$r_{im}^{(j)} = -\Big(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}^{(j)}(x_i)}\Big),\ i = 1, \ldots, n,\ j = 1, \ldots, K$$

Gradient Boosting for Classification

Input : training data set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ (1-hot encoded $\mathbf{y}$); ensemble size M, loss function $L(y, f(x)) = -\sum_{j=1}^K y_j \log(f^{(j)}(x))$, learning rate α
Output : ensemble of models

1. Initialize the models with prior probability values $\hat{f}^{(j)}(x) = \frac{1}{n}\sum_{i=1}^n \mathbf{y}_i$ for $j = 1, \dots, K$
2. For $m = 1, \dots, M$ do
    1. Compute the **pseudo-residuals**:

    $$r_{im}^{(j)} = -\Big(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}^{(j)}(x_i)}\Big),\ i = 1, \dots, n,\ j = 1, \dots, K$$

    2. Fit K **regression** models (typically a weak learner) $h_m^{(j)}(x)$ to pseudo-residuals $\{(\mathbf{x}_i, r_{im}^{(j)})\}_{i=1}^n$

Gradient Boosting for Classification

Input : training data set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ (1-hot encoded $\mathbf{y}$); ensemble size M, loss function $L(y, f(x)) = -\sum_{j=1}^K y_j \log(f^{(j)}(x))$, learning rate α

Output : ensemble of models

1. Initialize the models with prior probability values $\hat{f}^{(j)}(x) = \frac{1}{n}\sum_{i=1}^n \mathbf{y}_i$ for $j = 1, \ldots, K$
2. For $m = 1, \ldots, M$ do
    1. Compute the **pseudo-residuals**:

    $$r_{im}^{(j)} = -\Big(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}^{(j)}(x_i)}\Big), \, i = 1, \ldots, n, \; j = 1, \ldots, K$$

    2. Fit K **regression** models (typically a weak learner) $h_m^{(j)}(x)$ to pseudo-residuals $\{(\mathbf{x}_i, r_{im}^{(j)})\}_{i=1}^n$
    3. Update the models: $\hat{f}_m^{(j)} = \hat{f}_{m-1}^{(j)}(x) + \alpha h_m^{(j)}(x)$

Gradient Boosting for Classification

Input : training data set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n}$ (1-hot encoded $\mathbf{y}$); ensemble size M, loss function $L(y, f(x)) = -\sum_{j=1}^{K} y_j \log(f^{(j)}(x))$, learning rate α

Output : ensemble of models

1. Initialize the models with prior probability values $\hat{f}^{(j)}(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{y}_i$ for $j = 1, \ldots, K$
2. For $m = 1, \ldots, M$ do
    1. Compute the **pseudo-residuals**:

    $$r_{im}^{(j)} = -\Big(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}^{(j)}(x_i)}\Big),\ i = 1, \ldots, n,\ j = 1, \ldots, K$$

    2. Fit K **regression** models (typically a weak learner) $h_m^{(j)}(x)$ to pseudo-residuals $\{(\mathbf{x}_i, r_{im}^{(j)})\}_{i=1}^{n}$
    3. Update the models: $\hat{f}_m^{(j)} = \hat{f}_{m-1}^{(j)}(x) + \alpha h_m^{(j)}(x)$
    4. Apply softmax function to obtain probabilities $\hat{f}_m^{(j)} = \frac{\exp(\hat{f}_m^{(j)})}{\sum_{j=1}^{K} \exp(\hat{f}_m^{(j)})}$

Gradient Boosting for Classification

Input : training data set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ (1-hot encoded $\mathbf{y}$); ensemble size M, loss function $L(y, f(x)) = -\sum_{j=1}^K y_j \log(f^{(j)}(x))$, learning rate α
Output : ensemble of models

1. Initialize the models with prior probability values $\hat{f}^{(j)}(x) = \frac{1}{n}\sum_{i=1}^n \mathbf{y}_i$ for $j = 1, \ldots, K$
2. For $m = 1, \ldots, M$ do
   1. Compute the **pseudo-residuals**:

      $$r_{im}^{(j)} = -\Big(\frac{\partial L(y_i, \hat{f}(x_i))}{\partial \hat{f}^{(j)}(x_i)}\Big), \, i = 1, \ldots, n, \, j = 1, \ldots, K$$

   2. Fit K **regression** models (typically a weak learner) $h_m^{(j)}(x)$ to pseudo-residuals $\{(\mathbf{x}_i, r_{im}^{(j)})\}_{i=1}^n$
   3. Update the models: $\hat{f}_m^{(j)} = \hat{f}_{m-1}^{(j)}(x) + \alpha h_m^{(j)}(x)$
   4. Apply softmax function to obtain probabilities $\hat{f}_m^{(j)} = \frac{\exp(\hat{f}_m^{(j)})}{\sum_{j=1}^K \exp(\hat{f}_m^{(j)})}$
3. Return $\hat{f}_M(x)$. Predict the label with the highest probability.

What have we learned today?

- ✓ Boosting
- ✓ AdaBoost
- ✓ Gradient Boosting