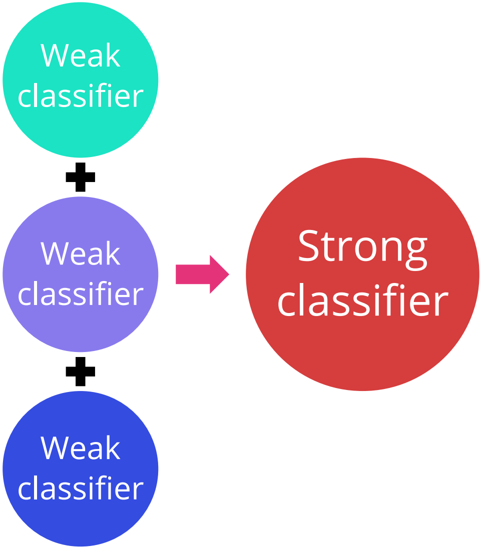
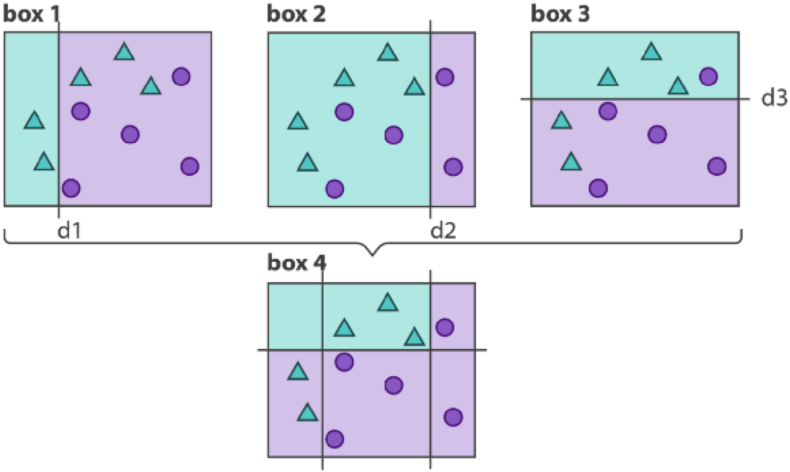
Boosting

Main idea of boosting



Boosting (Schapire, 1990)



Boosting: general algorithm

Boosting builds an **additive model**

$$f(x) = \sum_{m=1}^{M} \alpha_m T_m(x)$$ Regression

$$f(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m T_m(x)\right]$$ Classification

Here $T_m(x)$ is a weak classifier, usually a decision tree

Boosting: general algorithm

Data: $(x_1, y_1), \ldots, (x_n, y_n)$, label $y_i \in \{-1, 1\}$
Loss function: $L(y, f(x))$

- measures the “closeness” between true label y and prediction $f(x)$
- f is an accurate model when $L(y_i, f(x_i))$ is small for $i = 1, \ldots, n$

Boosting: general algorithm

Data: $(x_1, y_1), \ldots, (x_n, y_n)$, label $y_i \in \{-1, 1\}$
Loss function: $L(y, f(x))$

- measures the "closeness" between true label y and prediction $f(x)$
- f is an accurate model when $L(y_i, f(x_i))$ is small for $i = 1, \ldots, n$

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

1. Let $f_{m-1}(x) = \sum_{j=1}^{m-1} \alpha_j T_j(x)$
2. Find α_m and T_m that minimize the **total** loss:

$$\sum_{i=1}^{n} L\left(y_i, f_{m-1}(x_i) + \alpha_m T_m(x_i)\right)$$

Final model: $f(x) = \operatorname{sign}\left[\sum_{m=1}^{M} \alpha_m T_m(x)\right]$

Examples of loss functions

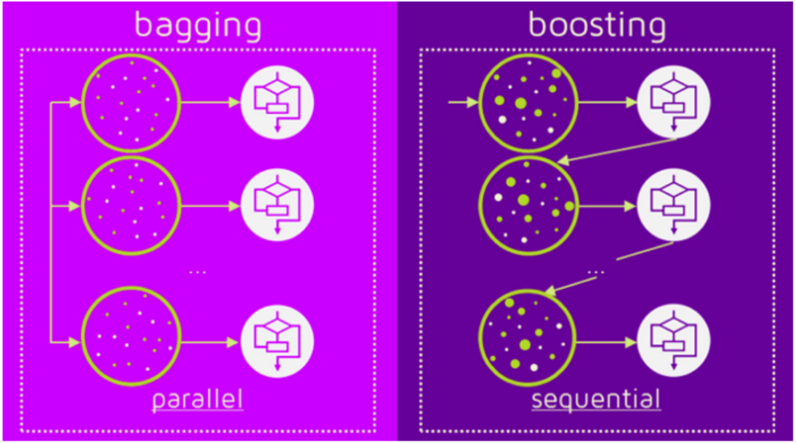
- Regression: L^2-loss/Squared loss

  $$L(y, f(x)) = (y - f(x))^2$$

- Classification: Binary cross-entropy

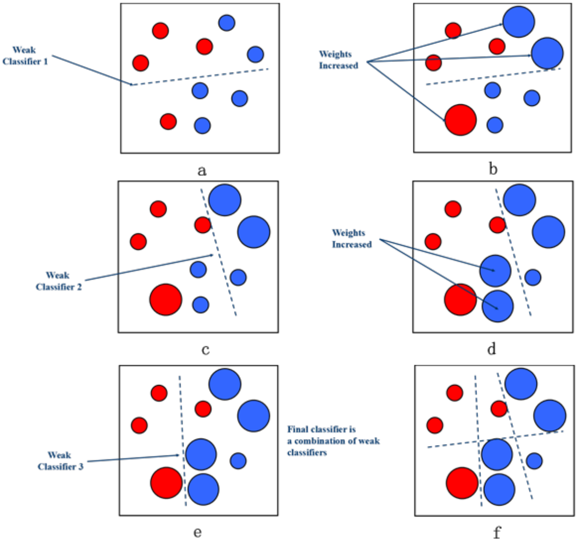
  - $$L(y, f(x)) = -y \log(f(x)) - (1 - y) \log(1 - f(x))$$

Bagging vs Boosting



AdaBoost

AdaBoost (Shapire & Freund, 1995)



AdaBoost algorithm

Data: $(x_1, y_1), \ldots, (x_n, y_n)$, label $y_i \in \{-1, 1\}$
Loss function: $L(y, f(x)) = \exp(-yf(x))$

AdaBoost algorithm

Data: $(x_1, y_1), \ldots, (x_n, y_n)$, label $y_i \in \{-1, 1\}$
Loss function: $L(y, f(x)) = \exp(-yf(x))$

1. Initialize the data weights
   $w_i = 1/n, i = 1, 2, \ldots, n$
2. or m = 1 to M repeat steps (a)-(d):

AdaBoost algorithm

Data: $(x_1, y_1), \ldots, (x_n, y_n)$, label $y_i \in \{-1, 1\}$
Loss function: $L(y, f(x)) = \exp(-yf(x))$

1. Initialize the data weights
   $w_i = 1/n, i = 1, 2, \ldots, n$
2. or m = 1 to M repeat steps (a)-(d):
   (a) Fit a classifier $T_m(x)$ to the training data with weights w_i
   that is, replace x_i with $w_i x_i$ in the training algo

AdaBoost algorithm

Data: $(x_1, y_1), \ldots, (x_n, y_n)$, label $y_i \in \{-1, 1\}$
Loss function: $L(y, f(x)) = \exp(-yf(x))$

1. Initialize the data weights
   $w_i = 1/n, i = 1, 2, \ldots, n$
2. or m = 1 to M repeat steps (a)-(d):
   (a) Fit a classifier $T_m(x)$ to the training data with weights w_i that is, replace x_i with $w_i x_i$ in the training algo
   (b) Compute

$$\text{err}_m = \sum_{i=1}^{n} w_i I\left(y \neq T_m(x_i)\right)$$

AdaBoost algorithm

Data: $(x_1, y_1), \ldots, (x_n, y_n)$, label $y_i \in \{-1, 1\}$
Loss function: $L(y, f(x)) = \exp(-yf(x))$

1. Initialize the data weights
   $w_i = 1/n, i = 1, 2, \ldots, n$
2. or m = 1 to M repeat steps (a)-(d):
   (a) Fit a classifier $T_m(x)$ to the training data with weights w_i that is, replace x_i with $w_i x_i$ in the training algo
   (b) Compute
   $$\mathrm{err}_m = \sum_{i=1}^{n} w_i I\left(y \neq T_m(x_i)\right)$$
   (c) Compute $\alpha_m = \frac{1}{2} \log\left((1 - \mathrm{err}_m)/\,\mathrm{err}_m\right)$

AdaBoost algorithm

Data: $(x_1, y_1), \ldots, (x_n, y_n)$, label $y_i \in \{-1, 1\}$
Loss function: $L(y, f(x)) = \exp(-yf(x))$

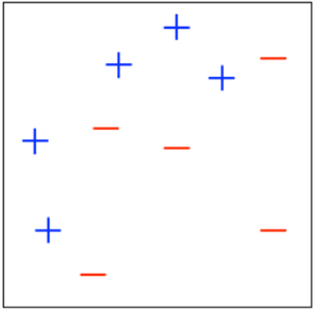
1. Initialize the data weights
   $w_i = 1/n, i = 1, 2, \ldots, n$
2. or m = 1 to M repeat steps (a)-(d):
   (a) Fit a classifier $T_m(x)$ to the training data with weights w_i
   that is, replace x_i with $w_i x_i$ in the training algo
   (b) Compute
   $$\mathrm{err}_m = \sum_{i=1}^{n} w_i I\left(y \neq T_m(x_i)\right)$$
   (c) Compute $\alpha_m = \frac{1}{2} \log\left((1 - \mathrm{err}_m) / \mathrm{err}_m\right)$
   (d) Update weights for $i = 1, \ldots, n$:
   $$w_i \leftarrow w_i \cdot \exp\left[-\alpha_m \cdot I(y_i \neq T_m(x_i))\right]$$
   and renormalize to w_i to sum to 1: $w_i \leftarrow w_i / \sum_{i=1}^{n} w_i$

AdaBoost algorithm

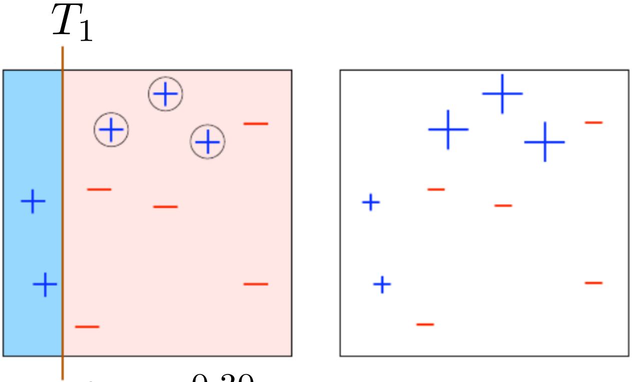
Data: $(x_1, y_1), \ldots, (x_n, y_n)$, label $y_i \in \{-1, 1\}$
Loss function: $L(y, f(x)) = \exp(-yf(x))$

1. Initialize the data weights
   $w_i = 1/n, i = 1, 2, \ldots, n$
2. or m = 1 to M repeat steps (a)-(d):
   (a) Fit a classifier $T_m(x)$ to the training data with weights w_i that is, replace x_i with $w_i x_i$ in the training algo
   (b) Compute
   $$\mathrm{err}_m = \sum_{i=1}^{n} w_i I\left(y \neq T_m(x_i)\right)$$
   (c) Compute $\alpha_m = \frac{1}{2} \log\left((1 - \mathrm{err}_m)/\,\mathrm{err}_m\right)$
   (d) Update weights for $i = 1, \ldots, n$:
   $$w_i \leftarrow w_i \cdot \exp\left[-\alpha_m \cdot I(y_i \neq T_m(x_i))\right]$$
   and renormalize to w_i to sum to 1: $w_i \leftarrow w_i / \sum_{i=1}^{n} w_i$
3. Final model: $f(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m T_m(x)\right]$

Example



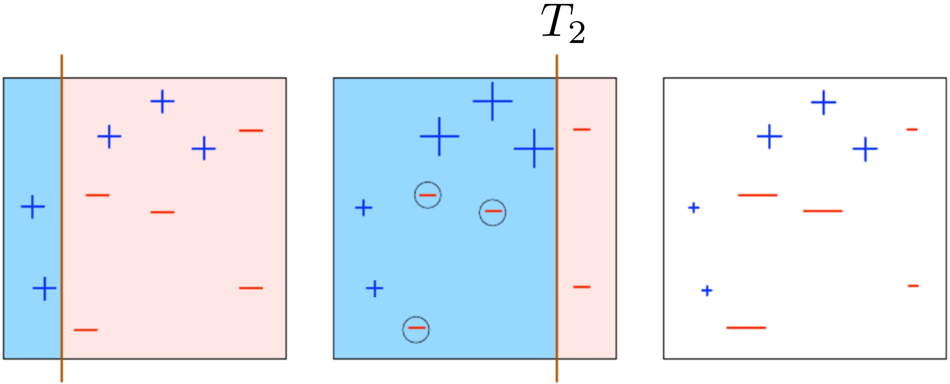
Adaboost round 1



$$\text{err}_1 = 0.30$$

$$\alpha_1 = 0.42$$

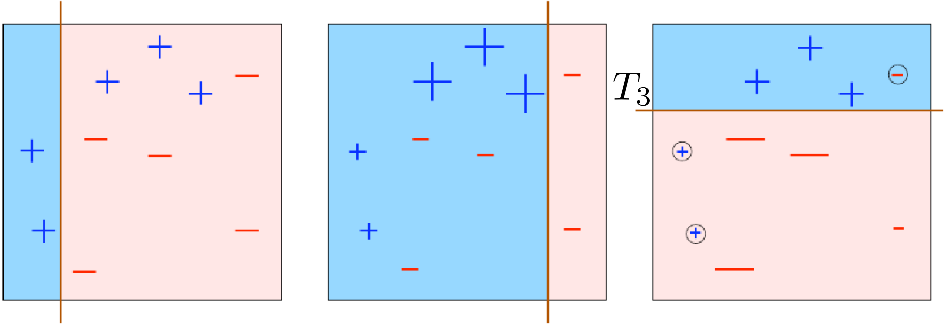
Adaboost round 2



$$\text{err}_2 = 0.21$$

$$\alpha_2 = 0.65$$

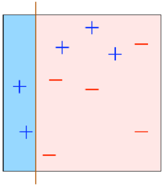
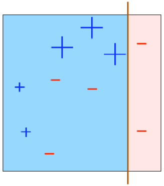
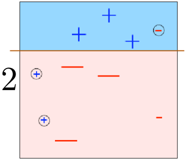
Adaboost round 3

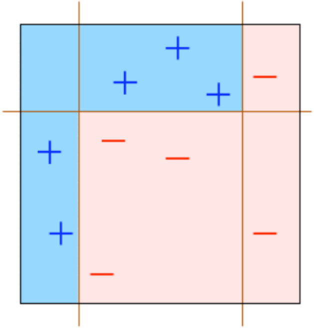


$$\mathrm{err}_3 = 0.14$$

$$\alpha_3 = 0.92$$

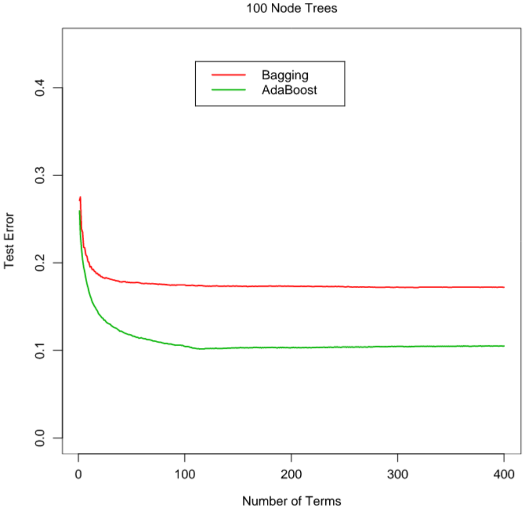
Adaboost final model

$f = \text{sign}\Big(0.42$  $+0.65$  $+0.92$  $\Big)$

$=$ 

AdaBoost vs Bagging

2000 simulated data points with 10 features



Gradient boosting

Motivation

Recall the general boosting algorithm:
For $m = 1, 2, \ldots, M$:

1. Let $f_{m-1}(x) = \sum_{j=1}^{m-1} \alpha_j T_j(x)$
2. Find α_m and T_m that minimize the **total** loss:

$$\sum_{i=1}^{n} L\left(y_i, f_{m-1}(x_i) + \alpha_m T_m(x_i)\right)$$

Final model: $f(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m T_m(x)\right]$

- If $L(y, f(x)) = \exp(-yf(x))$, from which we get AdaBoost
- For general loss functions this is a very difficult optimization problem
- **Gradient boosting** optimizes $L(y, f(x))$ using its **gradient**

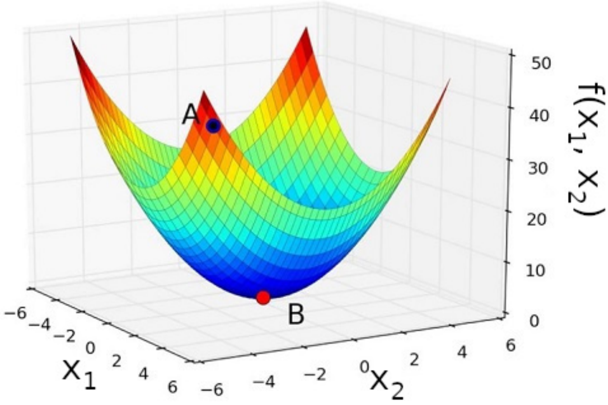
Gradient

Let $f(x_1, x_2) = x_1^2 + x_2^2$. Compute:

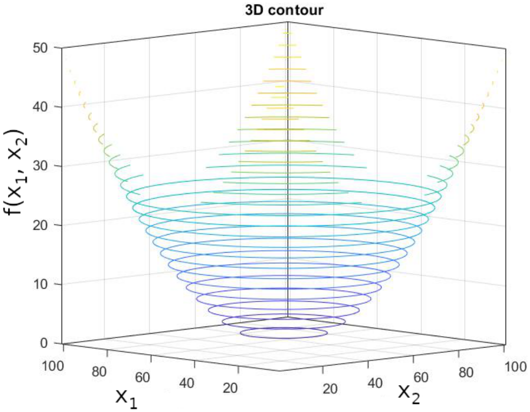
1. $\frac{\partial f}{\partial x_1}$
2. $\frac{\partial f}{\partial x_2}$
3. $\nabla f(x_1, x_2)$

Why gradient?

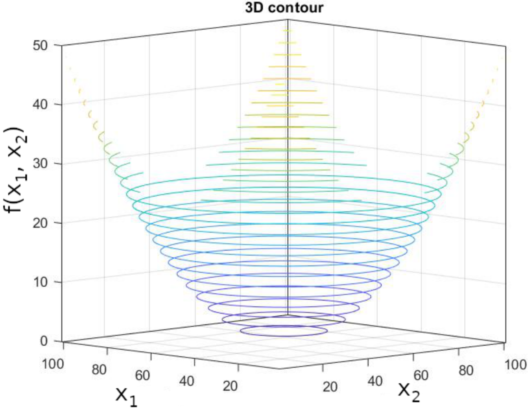
Suppose that, starting from A, we want to reach the minimum B

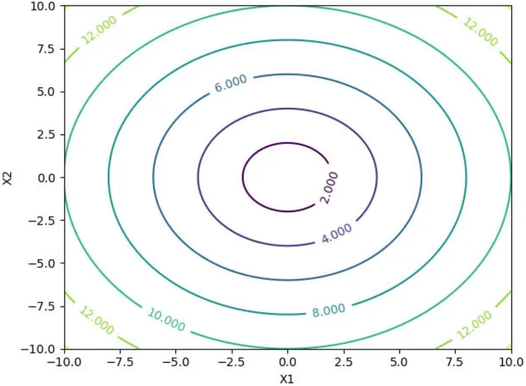


Why gradient?



Why gradient?



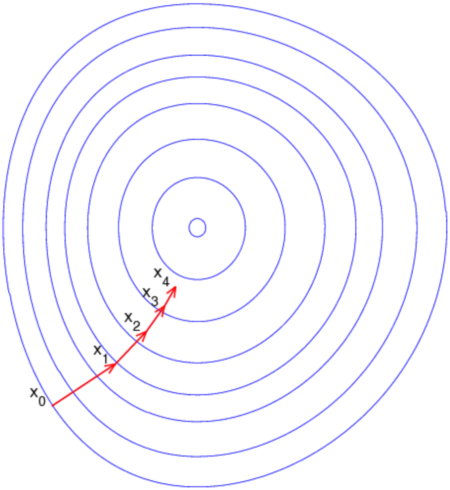


Gradient descent algorithm

Goal: find a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$ that minimizes a function $f(\boldsymbol{x})$
η: learning rate
T: optimization steps

1. Initialize $\boldsymbol{x}^{\mathbf{0}}$ at random
2. For $t = 1, 2, \ldots, T$:
   $$\boldsymbol{x}^t = \boldsymbol{x}^{t-1} - \eta \nabla f(\boldsymbol{x}^{t-1})$$
3. Let $\boldsymbol{x} = \boldsymbol{x}_T$

Example



Gradient boosting algorithm

1. Initialize $f_0(x) = \gamma$ that minimizes $\sum_{i=1}^{n} L(y_i, \gamma)$

Gradient boosting algorithm

1. Initialize $f_0(x) = \gamma$ that minimizes $\sum_{i=1}^{n} L(y_i, \gamma)$
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, n$ compute

$$r_{im} = -\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$$

Gradient boosting algorithm

1. Initialize $f_0(x) = \gamma$ that minimizes $\sum_{i=1}^{n} L(y_i, \gamma)$
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, n$ compute

   $$r_{im} = -\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$$

   (b) Fit a **regression** tree to the data $(x_1, r_{1m}), \ldots, (x_n, r_{nm})$, giving terminal regions $R_{jm}, j = 1, 2, \ldots, J_m$

Gradient boosting algorithm

1. Initialize $f_0(x) = \gamma$ that minimizes $\sum_{i=1}^{n} L(y_i, \gamma)$
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, n$ compute

   $$r_{im} = -\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$$

   (b) Fit a **regression** tree to the data $(x_1, r_{1m}), \ldots, (x_n, r_{nm})$, giving terminal regions $R_{jm}, j = 1, 2, \ldots, J_m$
   (c) For $j = 1, \ldots, m$, find γ_{jm} that minimizes

   $$\sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

Gradient boosting algorithm

1. Initialize $f_0(x) = \gamma$ that minimizes $\sum_{i=1}^{n} L(y_i, \gamma)$
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, n$ compute

   $$r_{im} = -\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$$

   (b) Fit a **regression** tree to the data $(x_1, r_{1m}), \ldots, (x_n, r_{nm})$, giving terminal regions $R_{jm}, j = 1, 2, \ldots, J_m$
   (c) For $j = 1, \ldots, m$, find γ_{jm} that minimizes

   $$\sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

   (d) Update

   $$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$

Gradient boosting algorithm

1. Initialize $f_0(x) = \gamma$ that minimizes $\sum_{i=1}^{n} L(y_i, \gamma)$
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, n$ compute

   $$r_{im} = -\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$$

   (b) Fit a **regression** tree to the data $(x_1, r_{1m}), \ldots, (x_n, r_{nm})$, giving terminal regions $R_{jm}, j = 1, 2, \ldots, J_m$
   (c) For $j = 1, \ldots, m$, find γ_{jm} that minimizes

   $$\sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

   (d) Update

   $$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$

3. Output $f(x) = f_M(x)$

Python code

```
from xgboost import XGBClassifier

xgb = XGBClassifier()

parameters = {
               'n_estimators': [100, 200, 300],
               'max_depth': [4, 5, 6],
               'min_child_weight': [1, 10, 100]
              }
clf = GridSearchCV(xgb, parameters, scoring='f1', cv=5)
clf.fit(Xtrain,ytrain1)

xgb.set_params(**clf.best_params_)
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
print(classification_report(y_test, y_pred))
```

Comparison of learning models

Characteristic	Neural Nets	SVM	CART	GAM	KNN, kernels	MART
Natural handling of data of "mixed" type	🔴	🔴	🟢	🟢	🔴	🟢
Handling of missing values	🔴	🔴	🟢	🔴	🟢	🟢
Robustness to outliers in input space	🔴	🔴	🟢	🟡	🟢	🟢
Insensitive to monotone transformations of inputs	🔴	🔴	🟢	🟢	🔴	🟢
Computational scalability (large N)	🔴	🔴	🟢	🟢	🔴	🟢
Ability to deal with irrelevant inputs	🔴	🔴	🟢	🟡	🔴	🟢
Ability to extract linear combinations of features	🟢	🟢	🔴	🔴	🟡	🟡
Interpretability	🔴	🔴	🟡	🟢	🔴	🟢
Predictive power	🟢	🟢	🔴	🔴	🟢	🟢