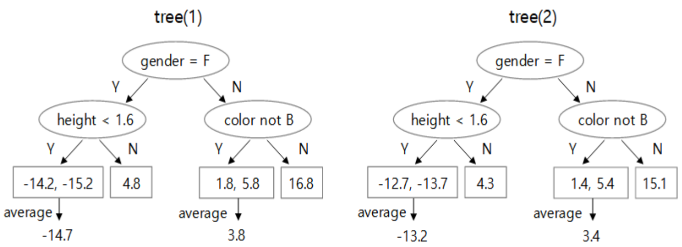
MX-AI

feature (x)			target (y)	residual (m=1)	residual (m=2)	residual (m=3)
height	favorite color	gender	weight	$r_{i,1}$	$r_{i,2}$	$r_{i,3}$
1.6	B	M	88	16.8	15.1	13.6
1.6	G	F	76	4.8	4.3	3.9
1.5	B	F	56	-15.2	-13.7	-12.4
1.8	R	M	73	1.8	1.4	1.1
1.5	G	M	77	5.8	5.4	5.1
1.4	B	F	57	-14.2	-12.7	-11.4



10. Gradient Boosting Method (GBM)

Part 1: Training & Prediction process

- This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

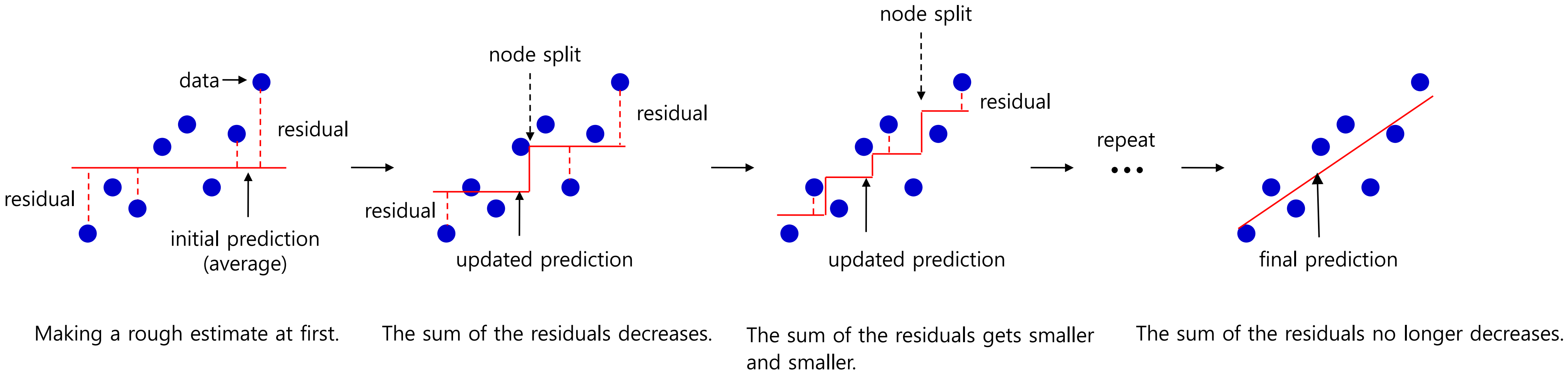
Regression

Classification

MX-AI

▪ Gradient Boosting Method (GBM): Overview

- Gradient Boosting Method (GBM) is a type of boosting ensemble learning algorithm that trains data with multiple weak learners and synthesizes the results. Decision trees are used as weak learners.
- GBM was proposed by Leo Breiman in 1997, updated by Jeremy H. Friedman in 1999 (Stochastic Gradient Boosting), and later developed into XGBoost and LightGBM, etc.
- GBM is applicable to both regression and classification and is a method of learning and reducing the residuals of target (label) data.



▪ Regression: Training process

▪ Let's see how GBM works through a simple example. The example is from the YouTube "StatQuest with Josh Starmer", Gradient Boost Part 1.

feature (x)			target (y)	residual (m=1)	residual (m=2)
height	favorite color	gender	weight	$r_{i,1}$	$r_{i,2}$
1.6	B	M	88	16.8	15.1
1.6	G	F	76	4.8	4.3
1.5	B	F	56	-15.2	-13.7
1.8	R	M	73	1.8	1.4
1.5	G	M	77	5.8	5.4
1.4	B	F	57	-14.2	-12.7

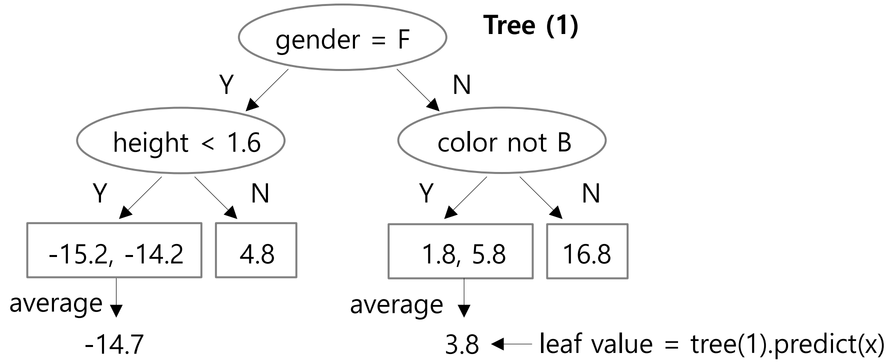
Training data

The residuals are smaller.

1) Calculate the overall average of the weight, which is the target y. $F_0 = 71.2$
Initially, this value is used as a rough estimate.

2) Calculate the residuals of the first iteration (m=1).
$r_{i,1} = y_i - F_0$
Residual of the first data point: $r_{1,1} = 88 – 71.2 = 16.8$

3) Create a shallow Decision Tree using the feature x and the residual $r_{i,1}$.
tree(1).fit(x, $r_{i,1}$)



4) Use the tree to predict a new target,y, F1, and compute a new residual, $r_{i,2}$, for each data point.
$F_{i,1} = F_0 + \alpha$ * tree(1).predict(x) ($\alpha = 0.1$: learning rate)
$r_{i,2} = y_i – F_{i,1}$

88 – (71.2 + 0.1 * 16.8) = 15.1	73 – (71.2 + 0.1 * 3.8) = 1.4
76 – (71.2 + 0.1 * 4.8) = 4.3	77 – (71.2 + 0.1 * 3.8) = 5.4
56 – (71.2 - 0.1 * 14.7) = -13.7	57 – (71.2 - 0.1 * 14.7) = -12.7

MX-AI

Regression: Training process

- This process is repeated until the residuals no longer decrease, and when training is complete, the trained trees are saved.
- For Stochastic Gradient Boosting (SGB: 1999, Friedman), feature x and residuals r are sampled without replacement to generate the tree at every iteration. SGB is a widely used approach to regularization of boosting models based on Decision Trees.

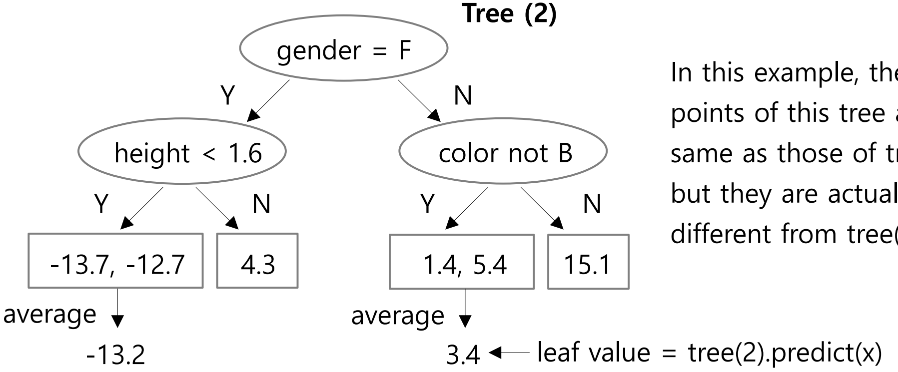
feature (x)			target (y)	residual (m=1)	residual (m=2)	residual (m=3)
height	favorite color	gender	weight	$r_{i,1}$	$r_{i,2}$	$r_{i,3}$
1.6	B	M	88	16.8	15.1	13.6
1.6	G	F	76	4.8	4.3	3.9
1.5	B	F	56	-15.2	-13.7	-12.4
1.8	R	M	73	1.8	1.4	1.1
1.5	G	M	77	5.8	5.4	5.1
1.4	B	F	57	-14.2	-12.7	-11.4

Training data

The residuals get smaller and smaller.

5) Create a shallow Decision Tree using the feature x and the residual $r_{i,2}$.
tree(1).fit(x, $r_{i,2}$)

Tree (2)



In this example, the split points of this tree are the same as those of tree(1), but they are actually different from tree(1).

[$F_1 = F_0 + \alpha$ * tree(1).predict(x)]

6) Calculate the new residuals, $r_{i,3}$, using tree(1) and tree(2). $F_{i,2} = F_{i,1} + \alpha$ * tree(2).predict(x), residual $r_{i,3} = y_i - F_{i,2}$

88 – (71.2 + 0.1 * 16.8 + 0.1 * 15.1) = 13.6	73 – (71.2 + 0.1 * 3.8 + 0.1 * 3.4) = 1.1
76 – (71.2 + 0.1 * 4.8 + 0.1 * 4.3) = 3.9	77 – (71.2 + 0.1 * 3.8 + 0.1 * 3.4) = 5.1
56 – (71.2 - 0.1 * 14.7 - 0.1 * 13.2) = -12.4	57 – (71.2 - 0.1 * 14.7 – 0.1 * 13.2) = -11.4

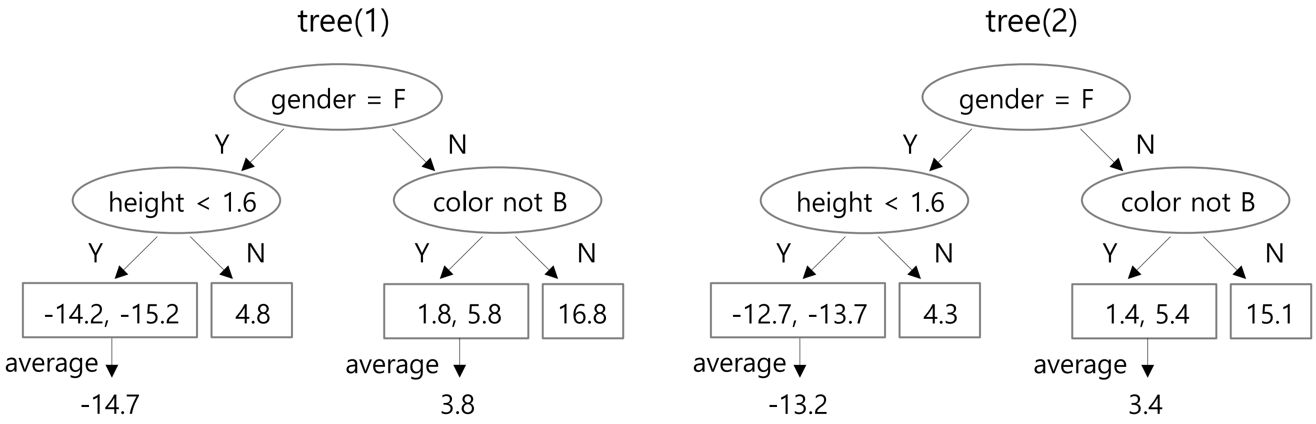
Regression: Prediction process

- The target values of the test data can be predicted using the trees saved during the training process.

feature (x_test) | target (y_pred)

height	favorite color	gender	weight
1.5	G	F	?

test data



6) Predict the target value of the test data using tree (1) and tree (2).

$$y\_pred\ (F^*) = F_0 + \alpha * \text{tree(1).predict(x\_test)} + \alpha * \text{tree(2).predict(x\_test)}$$

$$= 71.2 + 0.1 * (-14.7) + 0.1 * (-13.2) = 68.41$$

- To simplify the example, only two trees are used.
- In the actual training process, multiple trees are generated through iteration until the residuals do not decrease.
- The actual splitting points of tree(1) and tree(2) will be different.

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable loss function $L(y_i, F(x))$

Step-1: Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i,\ \gamma)$$

Step-2: for m = 1 to M:

(A) Compute so-called pseudo-residuals:

$$r_{i,m} = -\left[\frac{\partial L(y_i,\ F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for i = 1, ..., n}$$

(B) Fit a regression tree to the $r_{i,m}$ values and create terminal region $R_{j,m}$ for j = 1, ..., j_m

(C) for j = 1, ..., j_m compute $\gamma_{j,m} = \underset{\gamma}{argmin} \sum_{x_i \in R_{i,j}} L(y_i,\ F_{m-1}(x_i) + \gamma)$

(D) update the model: $F_m(x) = F_{m-1}(x) + \alpha \sum_{j=1}^{J_m} \gamma_{j,m} I(x \in R_{j,m})$

Step-3: Output $F_M(x)$

10. Gradient Boosting Method (GBM)

Part 2: Regression algorithm analysis

- This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

▪ GBM Algorithm analysis

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable loss function $L(y_i, F(x))$

Step-1: Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i, \gamma)$$

Step-2: for m = 1 to M:

(A) Compute so-called pseudo-residuals:

$$r_{i,m} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for i = 1, ..., n}$$

(B) Fit a regression tree to the $r_{i,m}$ values and create terminal region $R_{j,m}$ for j = 1, ..., j_m

(C) for j = 1, ..., j_m compute $\gamma_{j,m} = \underset{\gamma}{argmin} \sum_{x_i \in R_{i,j}} L(y_i, F_{m-1}(x_i) + \gamma)$

(D) update the model: $F_m(x) = F_{m-1}(x) + \alpha \sum_{j=1}^{J_m} \gamma_{j,m} I(x \in R_{j,m})$

Step-3: Output $F_M(x)$

* source: https://www.kaggle.com/code/angqx95/nuts-bolts-of-boosting-algorithms

feature (x)			target (y)	residual (m=1)
height	favorite color	gender	weight	$r_{i,1}$
1.6	B	M	88	16.8
1.6	G	F	76	4.8
1.5	B	F	56	-15.2
1.8	R	M	73	1.8
1.5	G	M	77	5.8
1.4	B	F	57	-14.2

$$L(y_i, \gamma) = \frac{1}{2} \sum_{i=1}^{n} (y - \gamma)^2$$

initial prediction

$$\frac{\partial L(y_i, \gamma)}{\partial \gamma} = -\sum_{i=1}^{n} (y - \gamma) = 0 \rightarrow \gamma = \frac{1}{n} \sum_{i=1}^{n} y_i \rightarrow F_0(x) = 71.2$$

* The average value of y is used as an initial estimate.

* if i = 1, m = 1

$$r_{1,1} = -\left[\frac{\partial L(y_1, F(x_1))}{\partial F(x_1)}\right]_{F(x) \rightarrow F_0(x)} \longrightarrow r_{1,1} = \frac{-\partial L(y_1, F_0(x_1))}{\partial F_0(x_1)}$$

$$r_{1,1} = -\frac{1}{2} \frac{\partial}{\partial F_0(x_1)} (y_1 - F_0(x_1))^2 = y_1 - F_0(x_1) = 88 - 71.2 = 16.8$$

* This is the residual from the first data point of the first iteration round.

MX-AI

▪ GBM Algorithm analysis

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable loss function $L(y_i, F(x))$

Step-1: Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i,\ \gamma)$$

height	favorite color	gender	weight	$r_{i,1}$
1.6	B	M	88	16.8
1.6	G	F	76	4.8
1.5	B	F	56	-15.2
1.8	R	M	73	1.8
1.5	G	M	77	5.8
1.4	B	F	57	-14.2

Step-2: for m = 1 to M:

(A) Compute so-called pseudo-residuals:

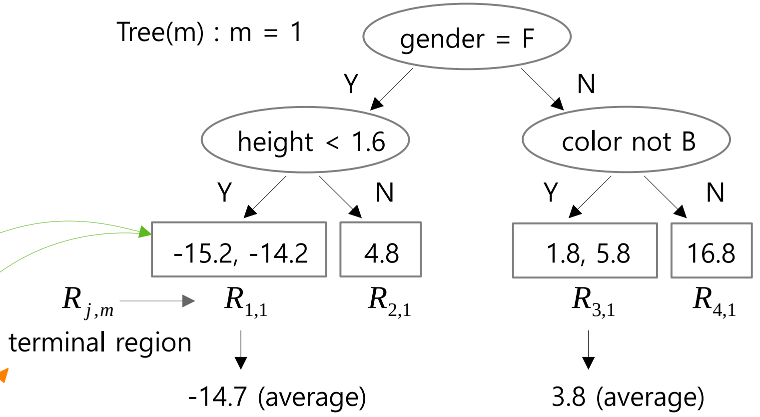
$$r_{i,m} = -\left[\frac{\partial L(y_i,\ F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for i = 1, ..., n}$$

(B) Fit a regression tree to the $r_{i,m}$ values and create terminal region $R_{j,m}$ for j = 1, ..., j_m

(C) for j = 1, ..., j_m compute $\gamma_{j,m} = \underset{\gamma}{argmin} \sum_{x_i \in R_{i,j}} L(y_i,\ F_{m-1}(x_i) + \gamma)$

(D) update the model: $F_m(x) = F_{m-1}(x) + \alpha \sum_{j=1}^{J_m} \gamma_{j,m} I(x \in R_{j,m})$

Step-3: Output $F_M(x)$



$$\gamma_{j=1,m=1} = \frac{1}{2}\frac{\partial}{\partial \gamma} \sum_{x_i \in R_{j,m}} (y_i - F_0(x_i) - \gamma)^2 = 0 \quad \leftarrow \ \in R_{1,1}$$

$$\sum_{x_i \in R_{j,m}} (y_i - F_0(x_i) - \gamma) = 0 \ \rightarrow \ 56 - 71.2 - \gamma + 57 - 71.2 - \gamma = 0$$

$\gamma_{1,1} = -14.7$ ← leaf value

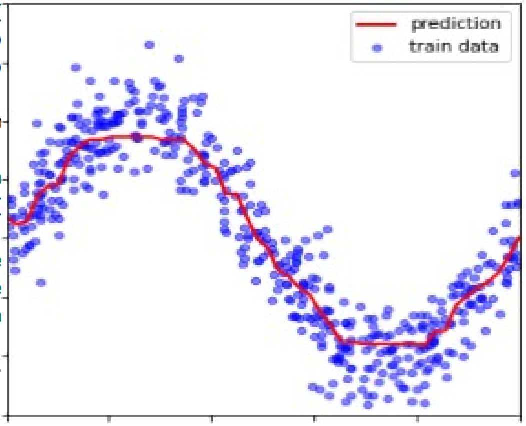
$$F_1(x_1) = F_0(x_1) + \alpha\,\gamma_{j=4,m=1} = 71.2 + 0.1 \times 16.8 = 72.88$$

- This is the first prediction (m=1) of the first training data point (i=1).
- It is closer to the actual value y2 = 88.0 than the initial estimate of 71.2.
- The more iterations we get, the closer we get to the actual value of 88.0. (m = 1, 2, 3...)

* source: https://www.kaggle.com/code/angqx95/nuts-bolts-of-boosting-algorithms

MX-AI

```
# [MXML-10-03] 1.GBM(regression).py
# Implementation of GBM algorithm using DecisionTreeRegressor.
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

# Create training data for regression
def nonlinear_data(n, s):
   rtn_x, rtn_y = [], []
   for i in range(n):
       x = np.random.random()
       y = 2.0*np.sin(
       rtn_x.append(x)
       rtn_y.append(y)

   return np.array(rtn

# Create training data
x, y = nonlinear_data(

n_depth = 3       # tre
n_tree = 300      # the
alpha = 0.01      # lea
```



10. Gradient Boosting Method (GBM)

Part 3: Implementation of GBM/SGBM

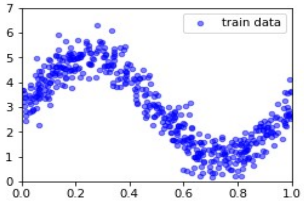
- This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

▪ Regression: Implementation of GBM

```python
# [MXML-10-03] 1.GBM(regression).py
# Implementation of GBM algorithm using DecisionTreeRegressor.
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

# Create training data for regression
def nonlinear_data(n, s):
   rtn_x, rtn_y = [], []
   for i in range(n):
       x = np.random.random()
       y = 2.0*np.sin(2.0*np.pi*x) + np.random.normal(0.0, s)+3.0
       rtn_x.append(x)
       rtn_y.append(y)

   return np.array(rtn_x).reshape(-1,1), np.array(rtn_y)

# Create training data
x, y = nonlinear_data(n=500, s=0.5)

n_depth = 3       # tree depth
n_tree = 50       # the number of trees (M)
alpha = 0.05      # learning rate
```




```python
# Visualize the training data and prediction results
def plot_prediction(x, y, x_test, y_pred, title):
    plt.figure(figsize=(6,4))
    plt.scatter(x, y, c='blue', s=20, alpha=0.5,
                label='train data')
    plt.plot(x_test, y_pred, c='red', lw=2.0,
             label='prediction')
    plt.xlim(0, 1)
    plt.ylim(0, 7)
    plt.legend()
    plt.title(title)
    plt.show()
```

MX-AI

▪ Regression: Implementation of GBM

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable loss function $L(y_i, F(x))$

Step-1: Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i,\ \gamma) \dashrightarrow \gamma = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Step-2: for m = 1 to M:

(A) Compute so-called pseudo-residuals:

$$r_{i,m} = -\left[\frac{\partial L(y_i,\ F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \text{ for i = 1, ..., n} \dashrightarrow r_{1,1} = y_1 - F_0(x_1)$$

(B) Fit a regression tree to the $r_{i,m}$ values and create terminal region $R_{j,m}$ for j = 1, ..., j_m

(C) for j = 1, ..., j_m compute $\gamma_{j,m} = \underset{\gamma}{argmin} \sum_{x_i \in R_{i,j}} L(y_i,\ F_{m-1}(x_i) + \gamma)$

(D) update the model: $F_m(x) = F_{m-1}(x) + \alpha \sum_{j=1}^{J_m} \gamma_{j,m} I(x \in R_{j,m})$

Step-3: Output $F_M(x)$

F = F_0 + α * tree(1).predict(x_test) + α * tree(2).predict(x_test) + ...

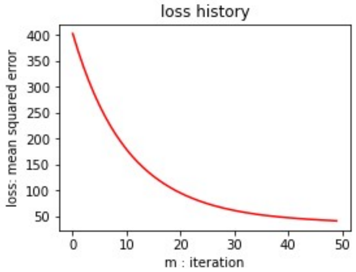
```
# step-1: Initialize model with a constant value.
F0 = y.mean()
```

```
# Training
Fm = F0
models, loss = [], []
for m in range(n_tree):
    # step-2 (A): Compute so-called pseudo-residuals
    residual = y - Fm

    # step-2 (B): Fit a regression tree to the residual
    gb_model = DecisionTreeRegressor(max_depth=n_depth)
    gb_model.fit(x, residual)

    # step-2 (C): compute gamma (prediction)
    gamma = gb_model.predict(x)

    # step-2 (D): Update the model
    Fm = Fm + alpha * gamma

    # Store trained tree models
    models.append(gb_model)

    # Calculate loss. loss = mean squared error.
    loss.append(((y - Fm) ** 2).sum())

# step-3: Output Fm(x) – Prediction of test data
y_pred = F0
x_test = np.linspace(0, 1, 50).reshape(-1, 1)

for model in models:
    y_pred += alpha * model.predict(x_test)
```

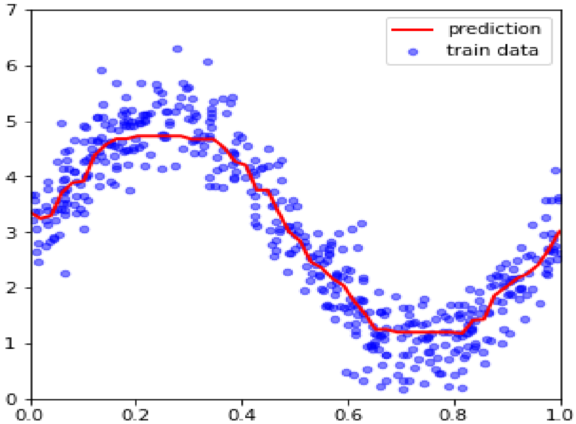

MX-AI

▪ Regression: Implementation of GBM

```
# Check the loss history
plt.figure(figsize=(5,4))
plt.plot(loss, c='red')
plt.xlabel('m : iteration')
plt.ylabel('loss: mean squared error')
plt.title('loss history')
plt.show()

# Visualize the training data and predicti
plot_prediction(x, y, x_test, y_pred, 'From scratch')
```



From scratch using DecisionTreeRegressor

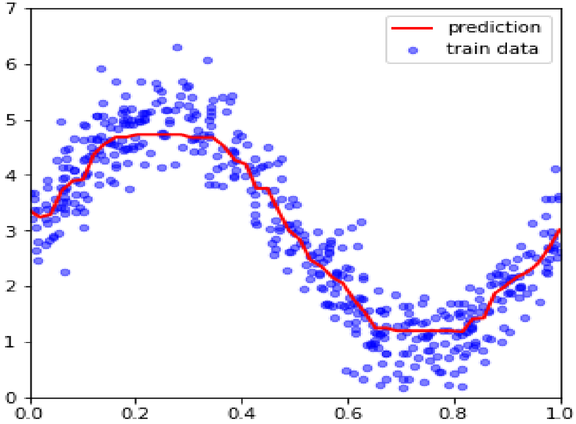


Result of GBM regression

▪ Implementation of this algorithm using scikit-klearn

```
# Compare with the results of sklearn's GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingRegressor

sk_model = GradientBoostingRegressor(n_estimators=n_tree,
                                     learning_rate=alpha,
                                     max_depth=n_depth)

sk_model.fit(x, y)                  # Training
y_pred = sk_model.predict(x_test)   # Preediction

# Visualize the training data and prediction results
plot_prediction(x, y, x_test, y_pred, 'GradientBoostingRegressor')
```

GradientBoostingRegressor

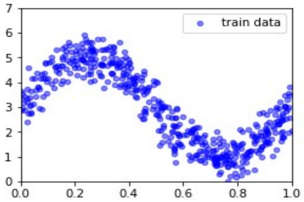


The two results agree well.

MX-AI

- Regression: Implementation of Stochastic Gradient Boosting Method (SGBM)

```
# [MXML-10-03] 2.SGBM(regression).py
# Stochastic Gradient Boosting Method (1999, Friedman)
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

# Create training data for regression
def nonlinear_data(n, s):
   rtn_x, rtn_y = [], []
   for i in range(n):
       x = np.random.random()
       y = 2.0*np.sin(2.0*np.pi*x) + np.random.normal(0.0,s) + 3.0
       rtn_x.append(x)
       rtn_y.append(y)

   return np.array(rtn_x).reshape(-1,1), np.array(rtn_y)

# Visualize the training data and prediction results
def plot_prediction(x, y, x_test, y_pred):
    plt.figure(figsize=(5,5))
    plt.scatter(x, y, c='blue',s=20,alpha=0.5,label='train data')
    plt.plot(x_test, y_pred, c='red', lw=2.0, label='prediction')
    plt.xlim(0, 1)
    plt.ylim(0, 7)
    plt.legend()
    plt.show()

# Create training data
x, y = nonlinear_data(n=500, s=0.5)
n_data = x.shape[0]
```



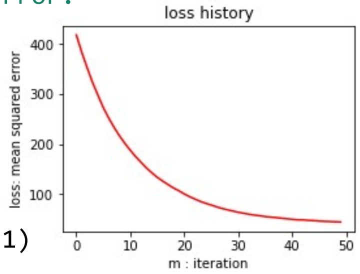

```
n_depth = 3          # tree depth (weak learner)
n_tree = 50          # the number of trees (M)
f_rate = 0.5         # rate of sampling
lr = 0.05            # learning rate

# step-1: Initialize model with a constant value.
F0 = y.mean()

# Training
Fm = np.repeat(F0, n_data)
models, loss = [], []
for m in range(n_tree):
    # data sampling without replacement
    si = np.random.choice(range(n_data), int(n_data * f_rate),\
                          replace=False)

    # step-2 (A): Compute so-called pseudo-residuals
    residual = y[si] - Fm[si]

    # step-2 (B): Fit a regression tree to the residual
    gb_model = DecisionTreeRegressor(max_depth=n_depth)
    gb_model.fit(x[si], residual)

    # step-2 (C): compute gamma (prediction)
    gamma = gb_model.predict(x)

    # step-2 (D): Update the model
    Fm = Fm + lr * gamma
    # Store trained tree models
    models.append(gb_model)
```

at each iteration a subsample of the training data is drawn at random (without replacement) from the full training data set. A random subsample of size $\tilde{N} < N$ is given by … (1999, Friedman, page 3)

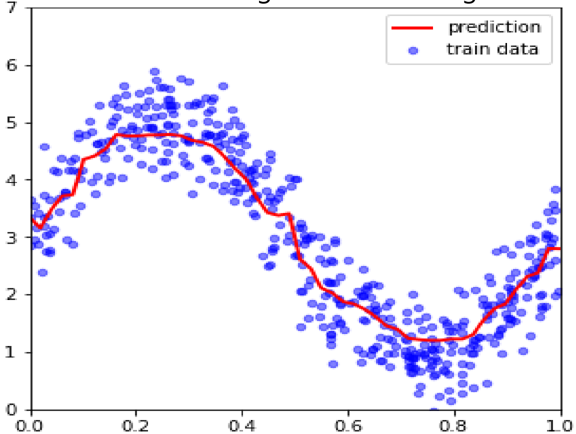
MX-AI

▪ Regression: Implementation of Stochastic Gradient Boosting Method (SGBM)

```
    # Calculate loss. loss = mean squared error.
    loss.append(((y - Fm) ** 2).sum())

# Check the loss history
plt.plot(loss, c='red')

# step-3: Output Fm(x) - Prediction
y_pred = F0
x_test = np.linspace(0, 1, 50).reshape(-1, 1)
for model in models:
    y_pred += lr * model.predict(x_test)

# Visualize the training data and prediction results
plot_prediction(x, y, x_test, y_pred)
```



From scratch using DecisionTreeRegressor

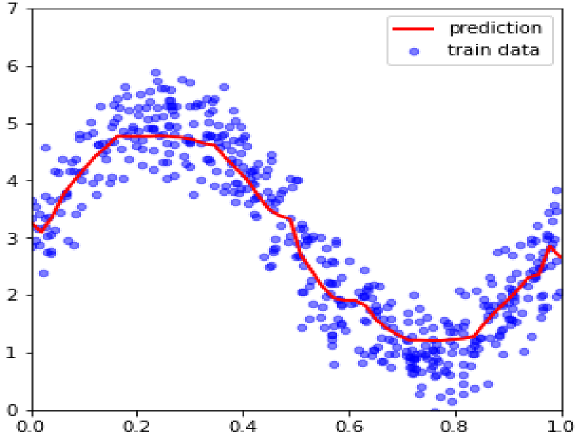


The result of SGBM regression. The prediction curve have become slightly smoother. This is the effect of regularization.

▪ Implementation of this algorithm using scikit-klearn

```
# Compare with the results of sklearn's GradientBoostingRegressor
from sklearn.ensemble import GradientBoostingRegressor
sk_model = GradientBoostingRegressor(n_estimators=n_tree,
                                     learning_rate=lr,
                                     max_depth=n_depth,
                                     subsample=f_rate)

sk_model.fit(x, y)                 # Training
y_pred = sk_model.predict(x_test)  # Prediction

# Visualize the training data and prediction results
plot_prediction(x, y, x_test, y_pred)
```

GradientBoostingRegressor

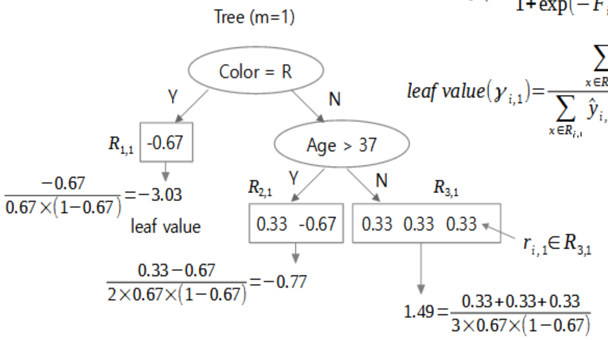


The two results agree well.

	feature (x)			target y (Yes=1, No=0)	initial prediction		residual	leaf value	Log odds	new prediction
i	likes popcorn	age	favorite color	loves Troll 2	$\hat{y}_{i,0}$	$F_{i,0}$	$r_{i,1}$	$\gamma_{i,1}$	$F_{i,1}$	$\hat{y}_{i,1}$
1	Yes	12	B	Yes	0.67	0.69	0.33	1.49	1.88	0.87
2	Yes	87	G	Yes	0.67	0.69	0.33	-0.77	0.07	0.52
3	No	44	B	No	0.67	0.69	-0.67	-0.77	0.07	0.52
4	Yes	19	R	No	0.67	0.69	-0.67	-3.03	-1.73	0.15
5	No	32	G	Yes	0.67	0.69	0.33	1.49	1.88	0.87
6	No	14	B	Yes	0.67	0.69	0.33	1.49	1.88	0.87

$$\hat{y}_{i,0}=\frac{1}{1+\exp(-F_{i,0})}=0.67$$

$$leaf\ value(\gamma_{i,1})=\frac{\sum_{x\in R_{i,j}} r_{i,1}}{\sum_{x\in R_{i,j}} \hat{y}_{i,0}(1-\hat{y}_{i,0})}$$



10. Gradient Boosting Method

GBM: Classification

Part 4: Training & Prediction process

- This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

▪ GBM Classification: Training process

▪ The GBM classification is similar to the regression process. The example is from the YouTube "StatQuest with Josh Starmer", Gradient Boost Part 3.

feature (x) | target y (Yes=1, No=0) | initial prediction | Log odds | residual | Leaf value

i	likes popcorn	age	favorite color	loves Troll 2	$\hat{y}_{i,0}$	$F_{i,0}$	$r_{i,1}$	$\gamma_{i,1}$
1	Yes	12	B	Yes	0.67	0.69	0.33	1.49
2	Yes	87	G	Yes	0.67	0.69	0.33	-0.77
3	No	44	B	No	0.67	0.69	-0.67	-0.77
4	Yes	19	R	No	0.67	0.69	-0.67	-3.03
5	No	32	G	Yes	0.67	0.69	0.33	1.49
6	No	14	B	Yes	0.67	0.69	0.33	1.49

training data | initial values | iterations (m=1)

1) Set the initial prediction of target y. Initially, we use a rough prediction of p(Yes). Then calculate the logodds prediction (F0) value of p(Yes).

$$\hat{y}_{i,0} = p(Yes) = \frac{4}{6} = 0.67, \forall i, \quad F_{i,0} = \log\left(\frac{\hat{y}_{i,0}}{1-\hat{y}_{i,0}}\right) = \log\left(\frac{4}{2}\right) = 0.69$$

* Since 0.67 is close to 1, we initially assume that everyone likes Troll-2.

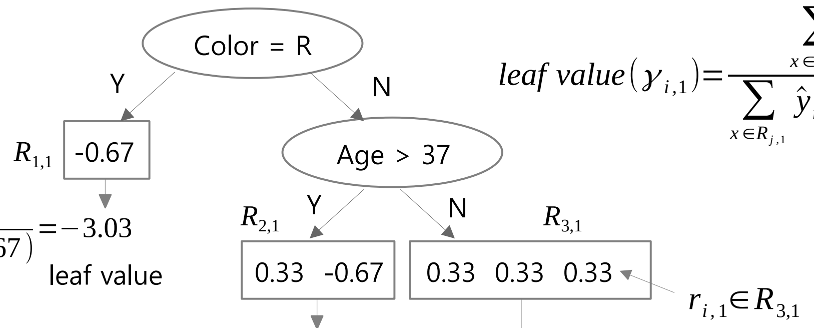
2) Calculate the residuals. $r_{i,1} = y_i - \hat{y}_{i,0}, \in (0,1)$

The first data point in the first iteration (i=1, m=1): $r_{1,1}$ = 1 – 0.67 = 0.33

3) Create a regression tree using the feature x and the residuals $r_{i,1}$. Since the residuals are real values, a regression tree is used for GBM classification. Leaf value (γ) can be calculated using the formula below. We'll look at this formula in more detail later.

$$\hat{y}_{i,0} = \frac{1}{1+\exp(-F_{i,0})} = 0.67$$

Tree (m=1)



$$leaf\ value(\gamma_{i,1}) = \frac{\sum_{x \in R_{j,1}} r_{i,1}}{\sum_{x \in R_{j,1}} \hat{y}_{i,0}(1-\hat{y}_{i,0})}$$

$$\frac{-0.67}{0.67 \times (1-0.67)} = -3.03$$

leaf value

$$\frac{0.33-0.67}{2 \times 0.67 \times (1-0.67)} = -0.77$$

$$1.49 = \frac{0.33+0.33+0.33}{3 \times 0.67 \times (1-0.67)}$$

MX-AI

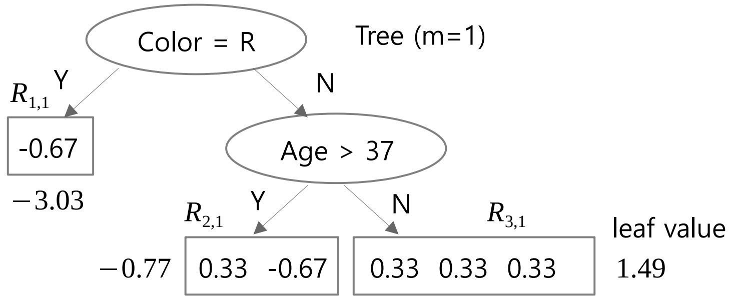
▪ GBM Classification: Training process

4) Update previous predictions with new predictions. The new prediction is closer to the target y than the initial prediction. The second data point is further away, but each iteration brings it closer.

feature (x) | target y (Yes=1, No=0) | initial prediction | residual | leaf value | Log odds | new prediction

i	likes popcorn	age	favorite color	loves Troll 2	$\hat{y}_{i,0}$	$F_{i,0}$	$r_{i,1}$	$\gamma_{i,1}$	$F_{i,1}$	$\hat{y}_{i,1}$
1	Yes	12	B	Yes	0.67	0.69	0.33	1.49	1.88	0.87
2	Yes	87	G	Yes	0.67	0.69	0.33	-0.77	0.07	0.52
3	No	44	B	No	0.67	0.69	-0.67	-0.77	0.07	0.52
4	Yes	19	R	No	0.67	0.69	-0.67	-3.03	-1.73	0.15
5	No	32	G	Yes	0.67	0.69	0.33	1.49	1.88	0.87
6	No	14	B	Yes	0.67	0.69	0.33	1.49	1.88	0.87

training data | initial value | iterations (m=1)



$$F_m(x) = F_{m-1}(x) + \alpha \sum_{j=1}^{J_m} \gamma_{j,m} I(x \in R_{j,m})$$

$$F_{1,1} = F_{1,0} + \alpha\gamma_{1,1} = 0.69 + 0.8 \times 1.49 = 1.88$$

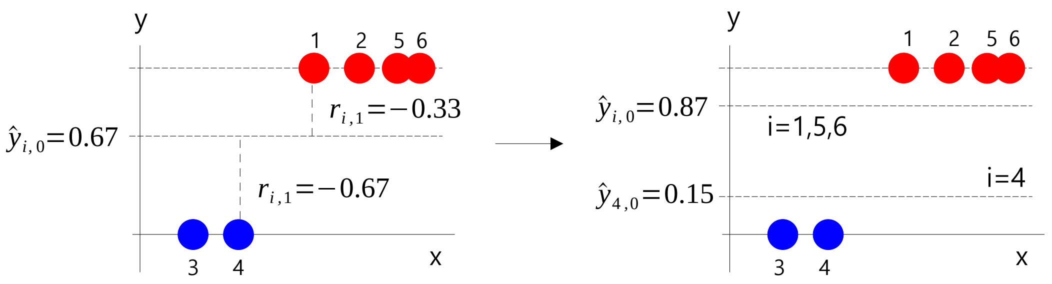
$$F_{2,1} = F_{2,0} + \alpha\gamma_{2,1} = 0.69 + 0.8 \times (-0.77) = 0.074$$

$\vdots$

$$\hat{y}_{1,1} = \frac{1}{1+\exp(-F_{1,1})} = \frac{1}{1+\exp(-1.88)} = 0.87$$

$$\hat{y}_{2,1} = \frac{1}{1+\exp(-F_{2,1})} = \frac{1}{1+\exp(0.07)} = 0.52$$

$\vdots$



MX-AI

▪ GBM Classification: Training process

5) Calculate the residuals for the next round (m=2) using the updated predictions.

$$r_{1,2} = y - \hat{y}_{1,1} = 1 - 0.87 = 0.13, \quad r_{2,2} = y - \hat{y}_{2,1} = 1 - 0.52 = 0.48, \quad \ldots$$

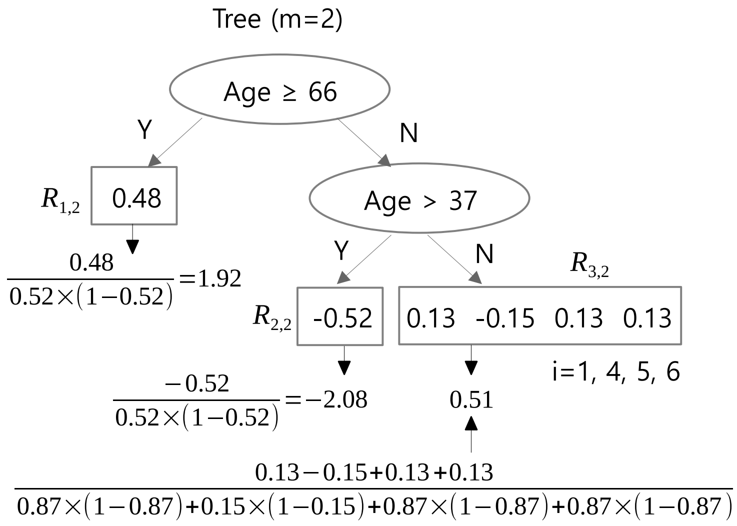
6) Use the features x and the residuals $r_{i,2}$ to create a regression tree and predict the leaf values (γ).

Use the tree to learn the residuals and predict the leaf values (γ) and logodds (F). And convert this into probability.

feature (x) | target y (Yes=1, No=0) | initial prediction

i	likes popcorn	age	favorite color	loves Troll 2	$\hat{y}_{i,0}$	$F_{i,0}$	$r_{i,1}$	$\gamma_{i,1}$	$F_{i,1}$	$\hat{y}_{i,1}$	$r_{i,2}$	$\gamma_{i,2}$	$F_{i,2}$	$\hat{y}_{i,2}$
1	Yes	12	B	Yes	0.67	0.69	0.33	1.49	1.88	0.87	0.13	0.51	2.29	0.91
2	Yes	87	G	Yes	0.67	0.69	0.33	-0.77	0.07	0.52	0.48	1.92	1.61	0.83
3	No	44	B	No	0.67	0.69	-0.67	-0.77	0.07	0.52	-0.52	-0.52	-0.34	0.41
4	Yes	19	R	No	0.67	0.69	-0.67	-3.03	-1.73	0.15	-0.15	0.51	-1.32	0.21
5	No	32	G	Yes	0.67	0.69	0.33	1.49	1.88	0.87	0.13	0.51	2.29	0.91
6	No	14	B	Yes	0.67	0.69	0.33	1.49	1.88	0.87	0.13	0.51	2.29	0.91

training data | initial value | iterations (m=1) | iterations (m=2)



7) Update previous predictions with new predictions.

$$F_{1,2} = F_{1,1} + \alpha\gamma_{1,2} = 1.88 + 0.8\times0.51 = 2.29$$
$$F_{2,2} = F_{2,1} + \alpha\gamma_{2,2} = 0.07 + 0.8\times1.92 = 1.61$$
$$\vdots$$

$$\hat{y}_{1,2} = \frac{1}{1+\exp(-F_{1,2})} = \frac{1}{1+\exp(-2.29)} = 0.91$$
$$\hat{y}_{2,2} = \frac{1}{1+\exp(-F_{2,2})} = \frac{1}{1+\exp(-1.61)} = 0.83$$

* As iterations progress, predictions become more accurate. Repeat until the residuals are sufficiently small.

$$\hat{y}_{i,0} \rightarrow \hat{y}_{i,1} \rightarrow \hat{y}_{i,2} \rightarrow \ldots$$

MX-AI

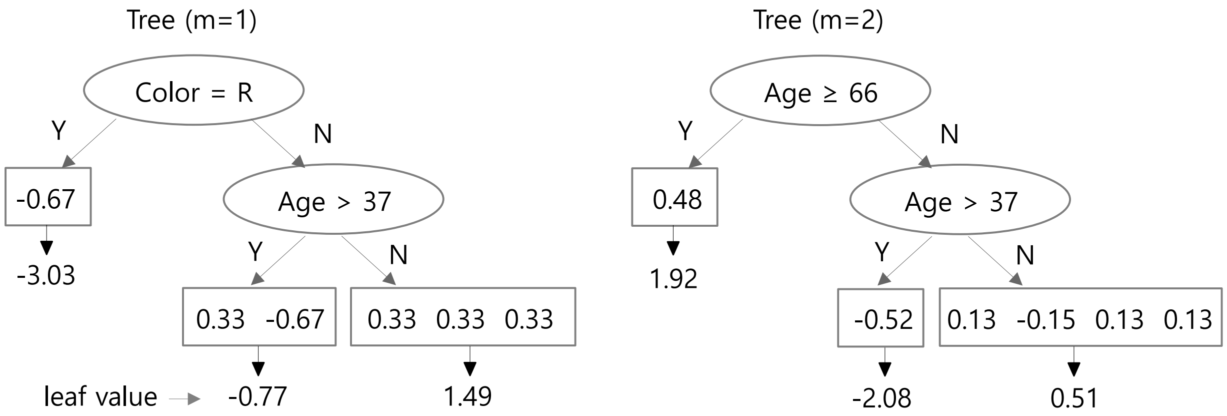
GBM Classification: Prediction process

- We can predict the target class of test data using the tree models saved during the training process.

feature (x_test) target (y_pred)

likes popcorn	age	favorite color	loves Troll 2
No	19	B	?

test data



8) Predict the target class of the test data using the trees.

$$F^* = F_0 + \alpha * \text{tree(1).predict(x\_test)} + \alpha * \text{tree(2).predict(x\_test)}$$

$$= 0.69 + 0.8 * 1.49 + 0.8 * 0.51 = 2.29$$

$$\text{y\_prob} = 1 / (1 + \exp(-2.29)) = 0.91$$ ← Since it is greater than 0.5, it is predicted as y_pred = Yes.

- As a simple example, here we only calculated the residuals two times.
- In practice, the training process is repeated until the residuals no longer decrease.

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable loss function $L(y_i, F(x))$

Step-1: Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i, \gamma)$$

Step-2: for m = 1 to M:

(A) Compute so-called pseudo-residuals:

$$r_{i,m} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n$$

(B) Fit a regression tree to the $r_{i,m}$ values and create terminal region $R_{j,m}$ for j = 1, ..., j_m

(C) for j = 1, ..., j_m compute $\gamma_{j,m} = \underset{\gamma}{argmin} \sum_{x_i \in R_{i,j}} L(y_i, F_{m-1}(x_i) + \gamma)$

(D) update the model: $F_m(x) = F_{m-1}(x) + \alpha \sum_{j=1}^{J_m} \gamma_{j,m} I(x \in R_{j,m})$

Step-3: Output $F_M(x)$

10. Gradient Boosting Method

GBM: Classification

Part 5: Classification algorithm analysis

- This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

- Classification Algorithm Analysis: Odds, Log(odds), Binary cross entropy

  - Target y ∈ {0, 1}

$$Odds = \frac{p(y=1|x)}{p(y=0|x)} = \frac{p(y=1|x)}{1-p(y=1|x)} \equiv \frac{p}{1-p} \quad \text{--- (1)}$$

$$= \frac{\hat{y}}{1-\hat{y}}$$

$$\log(odds) = \log\left(\frac{p}{1-p}\right) \equiv F \quad \text{--------- (2)}$$

$$\frac{p}{1-p} = \exp(F)$$

$$p-(1-p)\exp(F) = 0$$

$$p = \frac{\exp(F)}{1+\exp(F)} = \frac{1}{1+\exp(-F)} \quad \text{--------- (3)}$$

* Probability p and log(odds) F are interchangeable each other.

* p ∈ {0, 1}, F ∈ (-∞, +∞)

- Binary cross entropy

$$L = -y\cdot\log(p) - (1-y)\cdot\log(1-p) \quad \text{--------- (4)}$$

$p = \hat{y}$ ← predicted probability

$$\frac{\partial L}{\partial p} = \frac{-y}{p} + \frac{1-y}{1-p} = \frac{-y+p}{p(1-p)} \quad \text{--------- (5)}$$

- The binary cross entropy using Log(odds)

$$L = -y\cdot\log\left(\frac{\exp(F)}{1+\exp(F)}\right) - (1-y)\cdot\log\left(\frac{1}{1+\exp(F)}\right)$$

$$= -yF + y\cdot\log(1+\exp(F)) + \log(1+\exp(F)) - y\cdot\log(1+\exp(F))$$

$$= -yF + \log(1+\exp(F)) \quad \text{--------- (6)}$$

$$\frac{\partial L}{\partial F} = -y + \frac{\exp(F)}{1+\exp(F)} = -y+p \quad \text{--------- (7)}$$

(8)

$$\frac{d^2 L}{dF^2} = \frac{d}{dF}\left[-y + \frac{\exp(F)}{1+\exp(F)}\right] = \frac{\exp(F)(1+\exp(F))-\exp(F)\exp(F)}{(1+\exp(F))^2} = \hat{y}(1-\hat{y})$$

▪ Classification Algorithm Analysis

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable loss function $L(y_i, F(x))$

Step-1: Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i,\ \gamma)$$

Step-2: for m = 1 to M:

(A) Compute so-called pseudo-residuals:

$$r_{i,m} = -\left[\frac{\partial L(y_i,\ F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for i = 1, ..., n}$$

(B) Fit a regression tree to the $r_{i,m}$ values and create terminal region $R_{j,m}$ for j = 1, ..., j_m

(C) for j = 1, ..., j_m compute $\gamma_{j,m} = \underset{\gamma}{argmin} \sum_{x_i \in R_{i,j}} L(y_i,\ F_{m-1}(x_i) + \gamma)$

(D) update the model: $F_m(x) = F_{m-1}(x) + \alpha \sum_{j=1}^{J_m} \gamma_{j,m} I(x \in R_{j,m})$

Step-3: Output $F_M(x)$

$$L(y_i, \gamma) = -\sum_{i=1}^{n} [y_i \gamma - \log(1+\exp(\gamma))]$$ ← formula (6) on previous page

$$\frac{\partial L}{\partial \gamma} = -\sum_{i=1}^{n} \left(y_i - \frac{\exp(\gamma)}{1+\exp(\gamma)}\right)$$

$p(y=1|x) = \hat{y}$

formula (3) on the previous page

$$= -\sum_{i=1}^{n} (y_i - \hat{y}) = 0 \rightarrow \hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i = \frac{4}{6} = 0.67$$

$$F_0(x) = \gamma = \log\left(\frac{\hat{y}}{1-\hat{y}}\right) = \log\left(\frac{4}{2}\right) = 0.69$$

loves Troll 2
Yes
Yes
No
No
Yes
Yes

Initial prediction:

$$\hat{y}_{i,0} = p(Yes) = \frac{4}{6} = 0.67, \quad F_{i,0} = \log\left(\frac{\hat{y}_{i,0}}{1-\hat{y}_{i,0}}\right) = \log\left(\frac{4}{2}\right) = 0.69$$

* if i = 1, m = 1

$$r_{1,1} = -\left[\frac{\partial L(y_1,\ F(x_1))}{\partial F(x_1)}\right]_{F(x) \rightarrow F_0(x)} \rightarrow r_{1,1} = \frac{-\partial L(y_1,\ F_0(x_1))}{\partial F_0(x_1)}$$

$$r_{1,1} = -\frac{\partial}{\partial F_0(x_1)}[-y_1 F_0(x_1) + \log(1+\exp(F_0(x_1)))]$$

$$r_{1,1} = y_1 - \frac{\exp(F_0(x_1))}{1+\exp(F_0(x_1))} = y_1 - \hat{y}_1 = 1 - 0.67 = 0.33$$

formula (7) on the previous page

MX-AI

▪ Classification Algorithm Analysis

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable loss function $L(y_i, F(x))$

Step-1: Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i, \gamma)$$

Step-2: for m = 1 to M:

(A) Compute so-called pseudo-residuals:

$$r_{i,m} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for i = 1, ..., n}$$

(B) Fit a regression tree to the $r_{i,m}$ values and create terminal region $R_{j,m}$ for j = 1, ..., j_m

(C) for j = 1, ..., j_m compute $\gamma_{j,m} = \underset{\gamma}{argmin} \sum_{x_i \in R_{i,j}} L(y_i, F_{m-1}(x_i) + \gamma)$

(D) update the model: $F_m(x) = F_{m-1}(x) + \alpha \sum_{j=1}^{J_m} \gamma_{j,m} I(x \in R_{j,m})$

Step-3: Output $F_M(x)$

$$L(y_i, \gamma) = -\sum_{i=1}^{n} [y_i \gamma - \log(1+\exp(\gamma))] \leftarrow \text{formula (6) on previous page}$$

$$\frac{\partial L}{\partial \gamma} = -\sum_{i=1}^{n} \left(y_i - \frac{\exp(\gamma)}{1+\exp(\gamma)}\right)$$

$p(y=1|x) = \hat{y}$

formula (3) on the previous page

$$= -\sum_{i=1}^{n} (y_i - \hat{y}) = 0 \rightarrow \hat{y} = \frac{1}{n}\sum_{i=1}^{n} y_i = \frac{4}{6} = 0.67$$

$$F_0(x) = \gamma = \log\left(\frac{\hat{y}}{1-\hat{y}}\right) = \log\left(\frac{4}{2}\right) = 0.69$$

loves Troll 2
Yes
Yes
No
No
Yes
Yes

Initial prediction:

$$\hat{y}_{i,0} = p(Yes) = \frac{4}{6} = 0.67, \quad F_{i,0} = \log\left(\frac{\hat{y}_{i,0}}{1-\hat{y}_{i,0}}\right) = \log\left(\frac{4}{2}\right) = 0.69$$

* if i = 1, m = 1

$$r_{1,1} = -\left[\frac{\partial L(y_1, F(x_1))}{\partial F(x_1)}\right]_{F(x)\rightarrow F_0(x)} \rightarrow r_{1,1} = \frac{-\partial L(y_1, F_0(x_1))}{\partial F_0(x_1)}$$

$$r_{1,1} = -\frac{\partial}{\partial F_0(x_1)}[-y_1 F_0(x_1) + \log(1+\exp(F_0(x_1)))]$$

$$r_{1,1} = y_1 - \frac{\exp(F_0(x_1))}{1+\exp(F_0(x_1))} = y_1 - \hat{y}_1 = 1 - 0.67 = 0.33$$

formula (7) on the previous page

▪ Classification Algorithm Analysis

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable loss function $L(y_i, F(x))$

Step-1: Initialize model with a constant value:

$$F_0(x) = \underset{\gamma}{argmin} \sum_{i=1}^{n} L(y_i,\ \gamma)$$

Step-2: for m = 1 to M:

(A) Compute so-called pseudo-residuals:

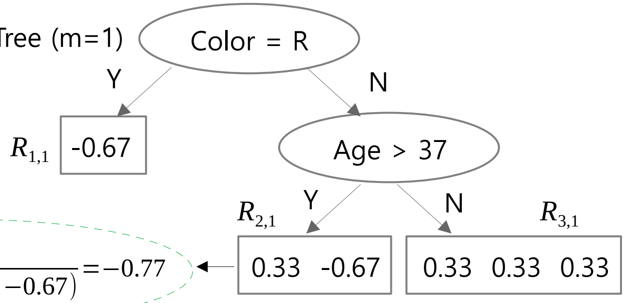
$$r_{i,m} = -\left[\frac{\partial L(y_i,\ F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \qquad \text{for i = 1, ..., n}$$

(B) Fit a regression tree to the $r_{i,m}$ values and create terminal region $R_{j,m}$ for j = 1, ..., j_m

(C) for j = 1, ..., j_m compute $\gamma_{j,m} = \underset{\gamma}{argmin} \sum_{x_i \in R_{i,j}} L(y_i,\ F_{m-1}(x_i) + \gamma)$

(D) update the model: $F_m(x) = F_{m-1}(x) + \alpha \sum_{j=1}^{J_m} \gamma_{j,m} I(x \in R_{j,m})$

Step-3: Output $F_M(x)$



$$\frac{0.33 - 0.67}{0.67(1-0.67) + 0.67(1-0.67)} = -0.77$$

Taylor series

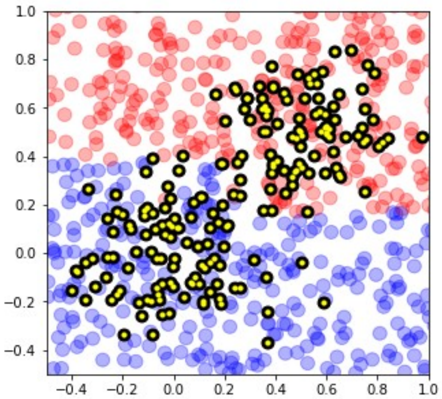
$$f(x+a, y+b) = f(x,y) + \frac{\partial f}{\partial x}a + \frac{\partial f}{\partial y}b + \frac{1}{2}\left(\frac{\partial^2 f}{\partial x^2}a^2 + \frac{\partial^2 f}{\partial y^2}b^2 + 2\frac{\partial^2 f}{\partial x \partial y}ab\right) + \ldots$$

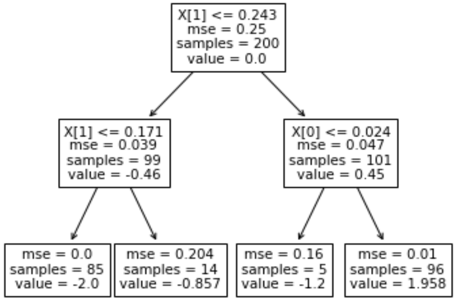
$$\sum L(y_i,\ F_{m-1}(x_i) + \gamma) \simeq \sum L(y_i,\ F_{m-1}(x_i)) + \gamma \sum \frac{d}{dF} L(y_i,\ F_{m-1}(x_i)) + \frac{1}{2}\gamma^2 \sum \frac{d^2}{dF^2} L(y_i,\ F_{m-1}(x_i))$$

$$\frac{d \sum L}{d\gamma} \simeq 0 + \sum \frac{dL}{dF} + \gamma \sum \frac{d^2 L}{dF^2} = 0 \ \rightarrow\ \gamma = -\sum \frac{dL}{dF} \Big/ \sum \frac{d^2 L}{dF^2}$$

$$-\frac{dL}{dF} = y - \hat{y} \leftarrow \text{formula (7)} \qquad \frac{d^2 L}{dF^2} = \hat{y}(1-\hat{y}) \leftarrow \text{formula (8)}$$

$$\gamma = \frac{\sum residual}{\sum \hat{y}(1-\hat{y})} \leftarrow \text{the leaf value}$$





10. Gradient Boosting Method

GBM: Classification

Part 6: Implementation of GBM Classification

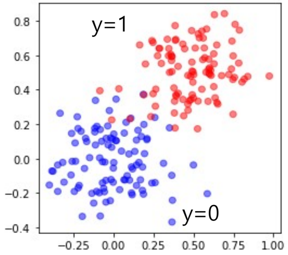
- This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

MX-AI

- Implementation of GBM classification using DecisionTreeRegressor and GradientBoostingClassifier

```
# [MXML-10-06] 3.GBM(classification).py
# Source: www.youtube.com/@meanxai
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree

# log(odds) --> probability
def F2P(f): return 1. / (1. + np.exp(-f))

# Creating training data
x, y = make_blobs(n_samples=200, n_features=2,
                  centers=[[0., 0.], [0.5, 0.5]],
                  cluster_std=0.18, center_box=(-1., 1.))
```




```
n_data = x.shape[0]
n_depth = 2    # tree depth
n_tree = 50    # the number of trees (M)
alpha = 0.1    # learning rate

# step-1: Initialize model with a constant value.
F0 = np.log(y.mean() / (1. - y.mean()))
Fm = np.repeat(F0, n_data)
```

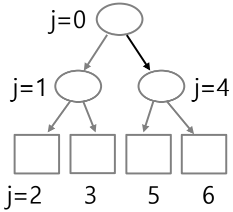
$$F_0(x) = \log\left(\frac{\hat{y}_0}{1-\hat{y}_0}\right)$$

```
models, loss = [], []
for m in range(n_tree):
    # step-2 (A): Compute so-called pseudo-residuals
    y_hat = F2P(Fm)
    residual = y - y_hat

    # step-2 (B): Fit a regression tree to the residuals
    gb_model = DecisionTreeRegressor(max_depth=n_depth)
    gb_model.fit(x, residual)

    # The leaf nodes of this tree contain the average of the
    # residuals. The predict() function returns this average value.
    # We replace these values with the leaf values gamma. Then the
    # predict() function will return the gamma.

    # step-2 (C): compute gamma
    # leaf_id = The leaf node number to which x belongs.
    leaf_id = gb_model.tree_.apply(x.astype(np.float32))

    # Replace the leaf values of all leaf nodes with their gamma
    # values, and update Fm.
    for j in np.unique(leaf_id):
        # i=Index of data points belonging to leaf node j.
        i = np.where(leaf_id == j)[0]
        gamma = residual[i].sum()/(y_hat[i] * (1.-y_hat[i])).sum()
```

$$\gamma = \frac{\sum residual}{\sum \hat{y}(1-\hat{y})}$$

$$\gamma_{j,m} = \underset{\gamma}{argmin} \sum_{x_i \in R_{i,j}} L(y_i,\ F_{m-1}(x_i)+\gamma)$$

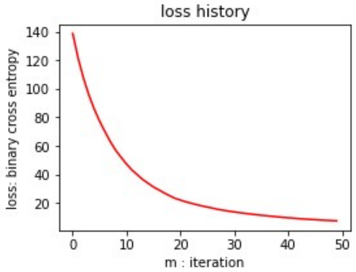
```
        # step-2 (D): Update the model
        Fm[i] += alpha * gamma

        # Replace the leaf values with their gamma
        gb_model.tree_.value[j, 0, 0] = gamma

    # save the trained model
    models.append(gb_model)
```



[preorder traversal]

- Implementation of GBM classification using DecisionTreeRegressor and GradientBoostingClassifier

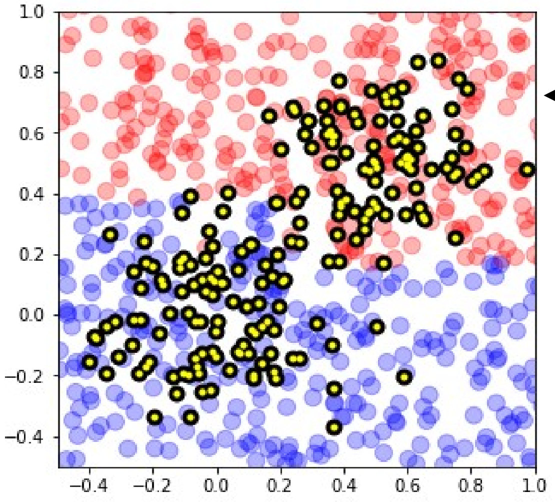
```
    # Calculating loss. loss = binary cross entropy.
    loss.append(-(y * np.log(y_hat + 1e-8) + \
                 (1.- y) * np.log(1.- y_hat + 1e-8)).sum())

# step-3: Output Fm(x) – Prediction of test data
Fm = F0
x_test = np.random.uniform(-0.5, 1.5, (1000, 2))

for model in models:
    Fm += alpha * model.predict(x_test)

y_prob = F2P(Fm)
y_pred = (y_prob > 0.5).astype('uint8')

# Check the loss history visually.
plt.figure(figsize=(5,4))
plt.plot(loss, c='red')
plt.xlabel('m : iteration')
plt.ylabel('loss: binary cross entropy')
plt.title('loss history')
plt.show()
```




```
# Visualize training and prediction results.
def plot_prediction(x, y, x_test, y_pred):
    plt.figure(figsize=(5,5))
    color = ['red' if a == 1 else 'blue' for a in y_pred]
    plt.scatter(x_test[:, 0], x_test[:, 1], s=100, c=color,
                alpha=0.3)
    plt.scatter(x[:, 0], x[:, 1], s=80, c='black')
    plt.scatter(x[:, 0], x[:, 1], s=10, c='yellow')
    plt.xlim(-0.5, 1.0)
    plt.ylim(-0.5, 1.0)
    plt.show()
```

```
# Visualize test data and y_pred.
plot_prediction(x, y, x_test, y_pred)

# Compare with Sklearn's GradientBoostingClassifier result.
from sklearn.ensemble import GradientBoostingClassifier

sk_model = GradientBoostingClassifier(n_estimators=n_tree,
                                      learning_rate=alpha,
                                      max_depth=n_depth)
sk_model.fit(x, y)

# Predict the target class of test data.
y_pred1 = sk_model.predict(x_test)

# Visualize test data and y_pred1.
plot_prediction(x, y, x_test, y_pred1)

# Check the first tree with the naked eye.
# Check that the trees generated by the two methods match well.
plt.figure(figsize=(6,5))
plot_tree(models[0])
plt.title('tree (m=1) by DecisionTreeRegressor')
plt.show()

plt.figure(figsize=(6,5))
plot_tree(sk_model.estimators_[0 ,0])
plt.title('tree (m=1) by GradientBoostingClassifier')
plt.show()
```

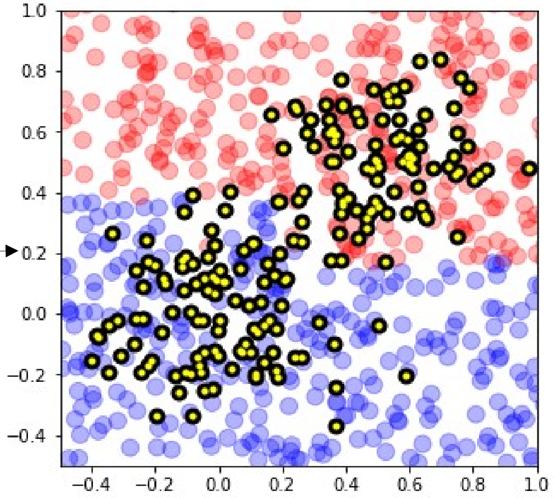

- Implementation of GBM classification using DecisionTreeRegressor and GradientBoostingClassifier

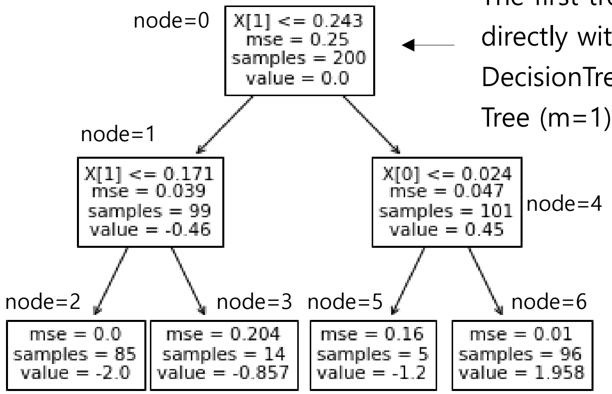


GBM result implemented directly with Sklearn's DecisionTreeRegressor.

- Training data
- Test data (y=1)
- Test data (y=0)

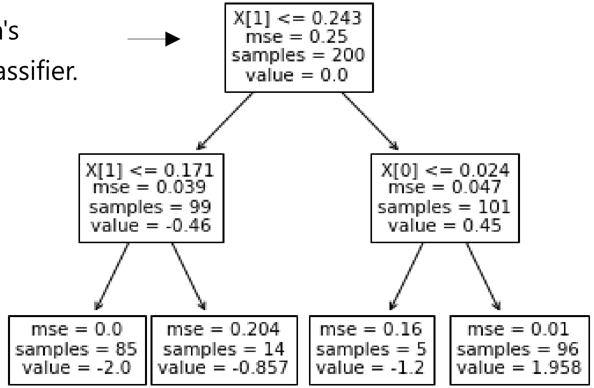
GBM result using Sklearn's GradientBoostingClassifier.





The first tree implemented directly with Sklearn's DecisionTreeRegressor. Tree (m=1)

The first tree implemented directly with Sklearn's GradientBoostingClassifier. Tree (m=1)

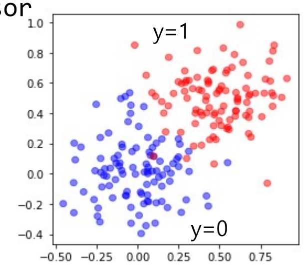


* The two results agree well.

MX-AI

▪ Implementation of SGBM classification

```
# [MXML-10-06] 4.SGBM(classification).py
# Stochastic Gradient Boosting Method (1999, Friedman)
# Source: www.youtube.com/@meanxai
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree


# log(odds) --> probability
def F2P(f):
    return 1. / (1. + np.exp(-f))

# Creating training data
x, y = make_blobs(n_samples=200, n_features=2,
                  centers=[[0., 0.], [0.5, 0.5]],
                  cluster_std=0.2, center_box=(-1., 1.))

n_data = x.shape[0]
n_depth = 2         # tree depth
n_tree = 200        # the number of trees (M)
f_rate = 0.5        # sampling ratio
alpha = 0.05        # learning rate

# step-1: Initialize model with a constant value.
F0 = np.log(y.mean() / (1. - y.mean()))
Fm = np.repeat(F0, n_data)

# Repeat
models, loss = [], []
```




```
for m in range(n_tree):
    # data sampling without replacement
    si = np.random.choice(range(n_data), int(n_data * f_rate),
                          replace=False)

    # step-2 (A): Compute so-called pseudo-residuals
    y_hat = F2P(Fm)
    residual = y[si] - y_hat[si]

    # step-2 (B): Fit a regression tree to the residual
    gb_model = DecisionTreeRegressor(max_depth=n_depth)
    gb_model.fit(x[si], residual)

    # The leaf nodes of this tree contain the average of the
    # residuals. The predict() function returns this average value.
    # We replace these values with the leaf values gamma. Then the
    # predict() function will return the gamma.

    # step-2 (C): compute gamma
    leaf_id = gb_model.tree_.apply(x[si].astype(np.float32))

    for j in np.unique(leaf_id):
        i = np.where(leaf_id == j)[0]
        xi = si[i]
        gamma = residual[i].sum()/(y_hat[xi] * (1.-y_hat[xi])).sum()

        # step-2 (D): Update the model
        Fm[xi] += alpha * gamma

        # Replace the leaf values with their gamma
        gb_model.tree_.value[j, 0, 0] = gamma
```


MX-AI

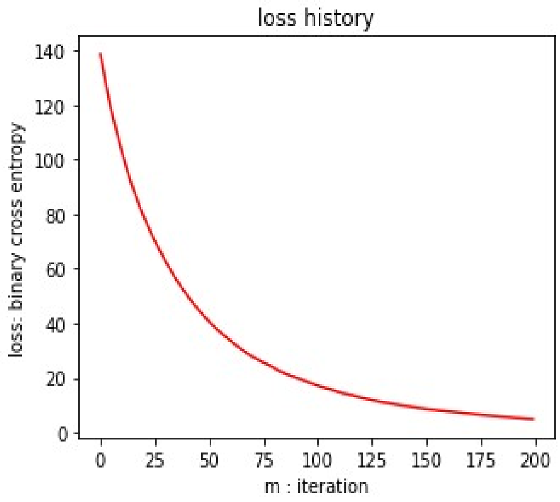
- Implementation of SGBM classification

```python
    # save the trained model
    models.append(gb_model)

    # Calculating loss. loss = binary cross entropy.
    loss.append(-(y * np.log(y_hat + 1e-8) + \
                 (1.- y) * np.log(1.- y_hat + 1e-8)).sum())

# Check the loss history visually.
plt.figure(figsize=(5,4))
plt.plot(loss, c='red')
plt.xlabel('m : iteration')
plt.ylabel('loss: binary cross entropy')
plt.title('loss history')
plt.show()

# step-3: Output Fm(x) - Prediction of test data
Fm = F0
x_test = np.random.uniform(-0.5, 1.5, (1000, 2))

for model in models:
    Fm += alpha * model.predict(x_test)

y_prob = F2P(Fm)
y_pred = (y_prob > 0.5).astype('uint8')

# Visualize training and prediction results.
def plot_prediction(x, y, x_test, y_pred):
    plt.figure(figsize=(5,5))
    color = ['red' if a == 1 else 'blue' for a in y_pred]
```

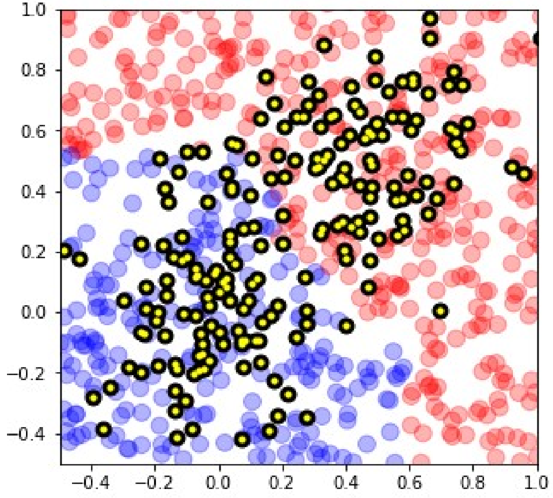
```python
    plt.scatter(x_test[:, 0], x_test[:, 1], s=100, c=color,
                alpha=0.3)
    plt.scatter(x[:, 0], x[:, 1], s=80, c='black')
    plt.scatter(x[:, 0], x[:, 1], s=10, c='yellow')
    plt.xlim(-0.5, 1.0)
    plt.ylim(-0.5, 1.0)
    plt.show()

# Visualize test data and y_pred.
plot_prediction(x, y, x_test, y_pred)

# Compare with Sklearn's GradientBoostingClassifier result.
from sklearn.ensemble import GradientBoostingClassifier
sk_model = GradientBoostingClassifier(n_estimators=n_tree,
                                      learning_rate=alpha,
                                      max_depth=n_depth,
                                      subsample=f_rate)
sk_model.fit(x, y)

# Predict the target class of test data.
y_pred1 = sk_model.predict(x_test)

# Visualize test data and y_pred1.
plot_prediction(x, y, x_test, y_pred1)
```


MX-AI

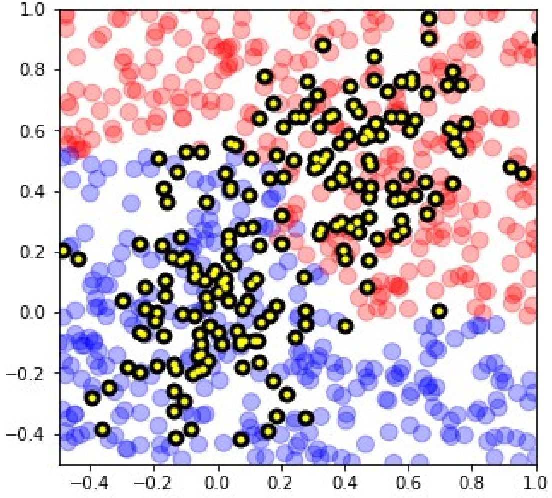
- Implementation of SGBM classification



SGBM result implemented directly with Sklearn's DecisionTreeRegressor.

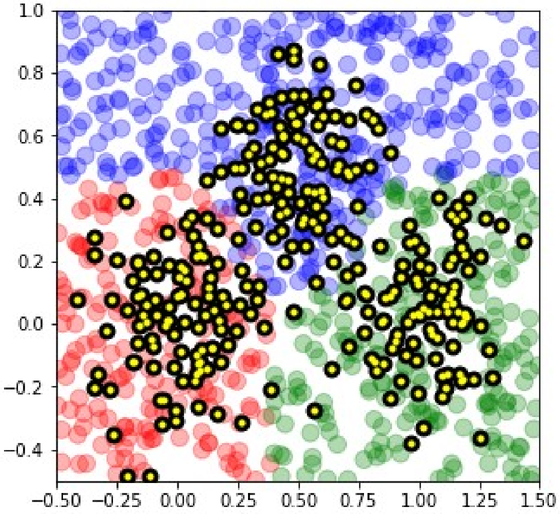


SGBM result using Sklearn's GradientBoostingClassifier.



Training data
Test data (y=1)
Test data (y=0)

* Due to sampling, the two results are not identical, but similar.



10. Gradient Boosting Method

GBM: Classification

Part 7: Multiclass Classification

- This video was produced in Korean and translated into English, and the audio was generated by AI (TTS).

www.youtube.com/@meanxai

▪ Multiclass Classification

- Convert multiclass y to one-hot encoding, and perform binary classification for each encoding.
- For example, if you have three types of classes (y = [0, 1, 2]), you can train them using three separate binary classification models.
- At each iteration (m=1, 2, 3…M), as many regression trees are created as there are classes. the number of models=(M * n_classes)
- The prediction step uses the largest softmax probability to predict the class.
- For detailed process, please refer to the code on the next page.

multiclass data | one-hot encoding

no	x1	x2	y	p(y=0)	p(y=1)	p(y=2)
1	0.8	1.2	0	1	0	0
2	9.8	2.0	2	0	0	1
3	3.2	3.5	0	1	0	0
4	4.8	5.2	1	0	1	0
…	…	…	…	…	…	…
N	9.2	2.5	2	0	0	1

x1	x2	p(y=0)
0.8	1.2	1
9.8	2.0	0
3.2	3.5	1
4.8	5.2	0
…	…	…
9.2	2.5	0

GBM model-0

$F_m^{(0)}, \ \hat{y}_m^{(0)}$

x1	x2	p(y=1)
0.8	1.2	0
9.8	2.0	0
3.2	3.5	0
4.8	5.2	1
…	…	…
9.2	2.5	0

GBM model-1

$F_m^{(1)}, \ \hat{y}_m^{(1)}$

x1	x2	p(y=2)
0.8	1.2	0
9.8	2.0	1
3.2	3.5	0
4.8	5.2	0
…	…	…
9.2	2.5	1

GBM model-2

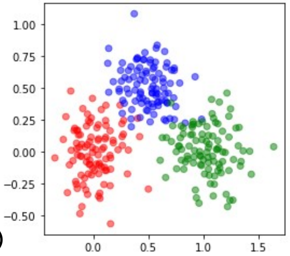
$F_m^{(2)}, \ \hat{y}_m^{(2)}$

At each iteration, the predicted probability of each model is converted to softmax and the residuals are calculated.

MX-AI

- Implementation of Multiclass Classification using DecisionTreeRegressor and GradientBoostingClassifier

```python
# [MXML-10-07] 5.GBM(multi-classification).py
import numpy as np
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
from sklearn.tree import DecisionTreeRegressor

# Create training data
x, y = make_blobs(n_samples=300, n_features=2,
                  centers=[[0., 0.], [0.5, 0.5], [1.0, 0.]],
                  cluster_std=0.18, center_box=(-1., 1.))

# One-hot encoding of y
y_ohe = OneHotEncoder().fit_transform(y.reshape(-1,1)).toarray()

# log(odds) --> probability
def F2P(f):
    return 1. / (1. + np.exp(-f))

def softmax(x):
    x = x - x.max(axis=1, keepdims=True)
    ex = np.exp(x)
    return ex / ex.sum(axis=1, keepdims=True)

n_data = x.shape[0]
n_class = y_ohe.shape[1]
n_depth = 2    # tree depth
n_tree = 30    # the number of trees (M)
alpha = 0.1    # learning rate
u_class = np.unique(y)
```




```python
# GradientBoostingClassifier: multi-class
# https://scikit-learn.org/stable/modules/ensemble.html
# Note: Classification with more than 2 classes requires the
# induction of n_classes regression trees at each iteration, thus,
# the total number of induced trees equals n_classes * n_estimators.

# step-1: Initialize model with a constant value.
# F0, Fm.shape = (n_data, n_class)
F0 = [np.log(y_ohe[:,c].mean()/(1. - y_ohe[:,c].mean())) \
      for c in u_class]
Fm = np.tile(np.array(F0), [n_data, 1])
```

$$F_0(x) = \log(\frac{\hat{y}_0}{1-\hat{y}_0})$$

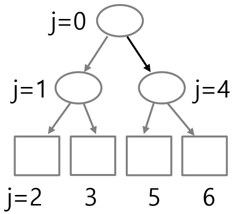
```python
# Repeat
gb_models = []
loss = []
for m in range(n_tree):
    # step-2 (A): Compute so-called pseudo-residuals
    # y_hat, residual.shape = (n_data, n_class)
    prob = F2P(Fm)
    y_hat = softmax(prob)
    residual = y_ohe - y_hat

    # step-2 (B): Fit a regression tree to the residual
    gb_model = []
    for ci in u_class:
        gb_model.append(DecisionTreeRegressor(max_depth=n_depth))
        gb_model[-1].fit(x, residual[:, ci])

        # step-2 (C): compute gamma
        # leaf_id = The leaf node number to which x belongs.
        leaf_id = gb_model[-1].tree_.apply(x.astype(np.float32))
```

▪ Implementation of Multiclass Classification using DecisionTreeRegressor and GradientBoostingClassifier

```
        # Replace the leaf values of all leaf nodes with their gamma
        # values, and update Fm.
        for j in np.unique(leaf_id):
            # xi=index of data points belonging to leaf node j.
            xi = np.where(leaf_id == j)[0]
            gamma = residual[:, ci][xi].sum() / \
                  (y_hat[:, ci][xi] * (1. - y_hat[:, ci][xi])).sum()

            # step-2 (D): Update the model
            Fm[:, ci][xi] += alpha * gamma

            # Replace the leaf values with their gamma
            gb_model[-1].tree_.value[j, 0, 0] = gamma

    gb_models.append(gb_model)

    # Calculating loss. loss = cross entropy.
    loss.append(-np.sum((y_ohe * np.log(y_hat + 1e-8)).sum(axis=1)))

# Check the loss history visually.
plt.figure(figsize=(5,4))
plt.plot(loss, c='red')
plt.xlabel('m : iteration')
plt.ylabel('loss: cross entropy')
plt.title('loss history')
plt.show()

# step-3: Output Fm(x) - Prediction of test data
x_test = np.random.uniform(-0.5, 1.5, (1000, 2))
```

$$\gamma = \frac{\sum residual}{\sum \hat{y}(1-\hat{y})}$$

```
Fm = np.tile(np.array(F0), [x_test.shape[0], 1])
for model in gb_models:
    for ci in u_class:
        Fm[:, ci] += alpha * model[ci].predict(x_test)

y_prob = F2P(Fm)
y_soft = softmax(y_prob)
y_pred = np.argmax(y_soft, axis=1)

# Visualize training and prediction results.
def plot_prediction(x, y, x_test, y_pred):
    plt.figure(figsize=(5,5))
    color = [['red', 'blue', 'green'][a] for a in y_pred]
    plt.scatter(x_test[:, 0], x_test[:, 1], s=100, c=color,
                alpha=0.3)
    plt.scatter(x[:, 0], x[:, 1], s=80, c='black')
    plt.scatter(x[:, 0], x[:, 1], s=10, c='yellow')
    plt.xlim(-0.5, 1.0)
    plt.ylim(-0.5, 1.0)
    plt.show()

# Visualize test data and y_pred.
plot_prediction(x, y, x_test, y_pred)
```

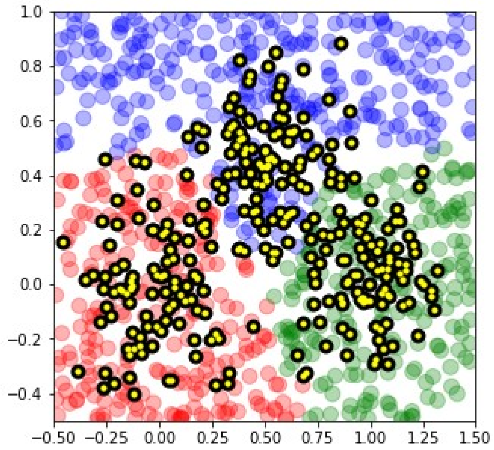


[preorder traversal]

MX-AI

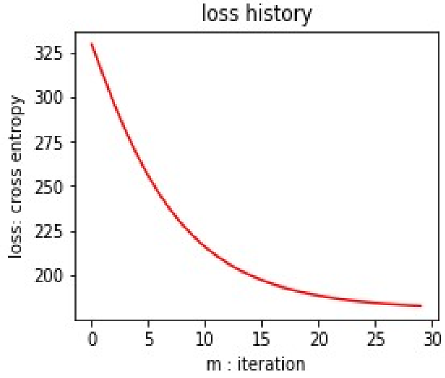
- Implementation of Multiclass Classification using DecisionTreeRegressor and GradientBoostingClassifier

```
# Compare with Sklearn's GradientBoostingClassifier result.
from sklearn.ensemble import GradientBoostingClassifier

sk_model = GradientBoostingClassifier(n_estimators=n_tree,
                                      learning_rate=alpha,
                                      max_depth=n_depth,
                                   criterion='squared_error')

sk_model.fit(x, y)

# Predict the target class of test data.
y_pred1 = sk_model.predict(x_test)

# Visualize test data and y_pred1.
plot_prediction(x, y, x_test, y_pred1)


sk_model.estimators_.shape --> [30, 3]

sk_model.estimators_[0]

array([DecisionTreeRegressor(max_depth=2),
       DecisionTreeRegressor(max_depth=2),
       DecisionTreeRegressor(max_depth=2),
      dtype=object)
```



← GBM result implemented using Sklearn's DecisionTreeRegressor.

Training data
Test data (y=0)
Test data (y=1)
Test data (y=2)



GBM result using Sklearn's GradientBoostingClassifier.