교 육 일 지

교육 제목	로지스틱회귀, 경사하강법, 트리
교육 일시	2021.10.19
교육 장소	YGL-C6
교육 내용	

로지스틱 회귀

Why logistic regression

분류 (KNN) → 회귀 분석 적용 (Logistic 이라함)

회귀 (Linear Regression) e.g) 부동산 가격

$y = \begin{cases} 1 \\ 0 \end{cases}$

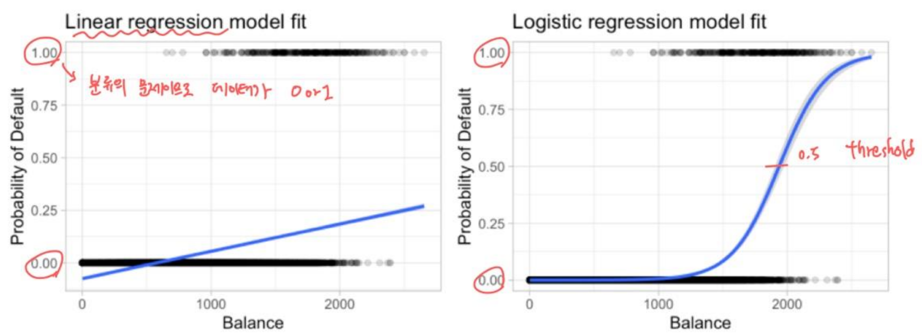


Figure 5.1: Comparing the predicted probabilities of linear regression (left) to logistic regression (right). Predicted probabilities using linear regression results in flawed logic whereas predicted values from logistic regression will always lie between 0 and 1.

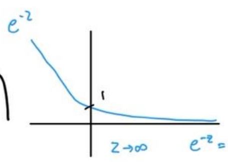
https://bradleyboehmke.github.io/HOML

Multiple logistic regression

- Formula extend

$$p(X) = \frac{e^{\beta_0+\beta_1X_1+\ldots+\beta_pX_p}}{1+e^{\beta_0+\beta_1X_1+\ldots+\beta_pX_p}} = \frac{1}{1+e^{-\underbrace{(\beta_0+\beta_1X_1+\ldots+\beta_pX_p)}_{z}}} = \frac{1}{1+e^{-z}}$$

$\rightsquigarrow y = \beta_0 + \beta_1 x$



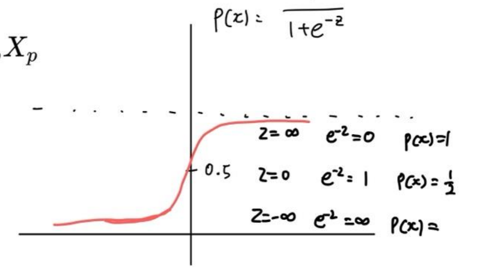
$z \to \infty \quad e^{-z} = 0$

$z \to 0 \quad e^{-z} = 1$

$z \to -\infty \quad e^{-z} = \infty$

- Logit transformation

$$g(X) = ln[\frac{p(x)}{1-p(x)}] = \beta_0 + \beta_1X + \ldots + \beta_pX_p$$



$p(x) = \frac{1}{1+e^{-z}}$

$1 - p(x) = \frac{e^{-z}}{1+e^{-z}}$

$\frac{P(x)}{1-P(x)} = \frac{1}{e^{-z}}$

$ln\left(\frac{P(x)}{1-P(x)}\right) = ln(e^{z}) = z = (\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k)$

로지스틱 회귀

- 분류의 문제에 회귀분석을 적용한 것을 로지스틱(Logistic)이라 한다.

In [1]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

fish = pd.read_csv('https://bit.ly/fish_csv_data')
fish.head()
```

Out [1]:

	Species	Weight	Length	Diagonal	Height	Width
0	Bream	242.0	25.4	30.0	11.5200	4.0200
1	Bream	290.0	26.3	31.2	12.4800	4.3056
2	Bream	340.0	26.5	31.1	12.3778	4.6961
3	Bream	363.0	29.0	33.5	12.7300	4.4555
4	Bream	430.0	29.0	34.0	12.4440	5.1340

In [2]:

```python
print(fish.shape)
fish[fish.columns[0]].value_counts()
```

```
(159, 6)
```

Out [2]:

```
Perch        56
Bream        35
Roach        20
Pike         17
Smelt        14
Parkki       11
Whitefish     6
Name: Species, dtype: int64
```

- Input data 만들기

In [3]:

```python
fish_input = fish[fish.columns[1:]].to_numpy()
fish_input[:5]
```

Out [3]:

```
array([[242.    ,  25.4   ,  30.    ,  11.52  ,   4.02  ],
       [290.    ,  26.3   ,  31.2   ,  12.48  ,   4.3056],
       [340.    ,  26.5   ,  31.1   ,  12.3778,   4.6961],
       [363.    ,  29.    ,  33.5   ,  12.73  ,   4.4555],
       [430.    ,  29.    ,  34.    ,  12.444 ,   5.134 ]])
```

- Target Data 만들기

- Target Data 만들기

In [4]:
```
fish_target = fish[fish.columns[0]].to_numpy()
fish_target[:5]
```

Out [4]:
```
array(['Bream', 'Bream', 'Bream', 'Bream', 'Bream'], dtype=object)
```

- Data Split

In [5]:
```
from sklearn.model_selection import train_test_split

train_input, test_input, train_target, test_target = train_test_split(
    fish_input,
    fish_target,
    random_state = 42,
    stratify = fish_target    # fish_target의 Class 비율에 맞게 split
)

print('train_shape: ',train_input.shape,'\ntest_shape : ',test_input.shape)
```

```
train_shape:  (119, 5)
test_shape :  (40, 5)
```

- Feature Rescaling

In [6]:
```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
ss.fit(train_input)

#data Transform
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)  # 테스트 scale도 train data로 fit한 객체로 변환

train_scaled[:5]
```

Out [6]:
```
array([[-0.75628803, -0.66065677, -0.62357446, -0.78015159, -0.45043644],
       [-0.45991057, -0.1248453 , -0.24414603, -0.4293487 ,  0.03516919],
       [ 0.07356886,  0.0212851 ,  0.2165885 ,  0.79541208,  0.37481797],
       [ 1.54063728,  1.0441979 ,  1.23743166,  2.29283234,  1.34130358],
       [-0.87483902, -0.75807703, -0.82232269, -0.80672937, -0.5697143 ]])
```

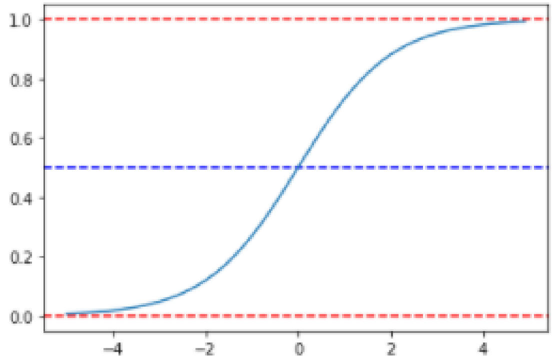
로지스틱 회귀 (Logistic Regression)

시그모이드 함수(Sigmoid Function)

$$Sigmoid(z) = \frac{1}{1+e^{-z}}$$

In [7]:

```
z = np.arange(-5, 5, 0.1)
prob_y = 1 / (1+np.exp(-z)) # sigmoid Function

plt.axhline(1, linestyle = '--',color = 'r')
plt.axhline(0.5, linestyle = '--',color = 'b')
plt.axhline(0, linestyle = '--',color = 'r')
plt.plot(z,prob_y)
plt.show()
```



smelt 와 bream의 데이터만 추출

In [8]:

```
bream_smelt_index = (train_target == "Bream") | (train_target == 'Smelt')
bream_smelt_index
```

Out [8]:

```
array([False, False,  True,  True, False, False, False,  True, False,
        True, False,  True, False,  True, False, False,  True, False,
       False,  True,  True, False,  True, False,  True, False,  True,
       False, False,  True,  True, False, False, False, False, False,
        True, False,  True, False, False, False, False, False, False,
        True, False,  True, False, False,  True,  True, False,  True,
        True, False,  True, False, False,  True, False,  True, False,
       False, False,  True, False, False, False, False, False, False,
       False,  True, False,  True, False, False, False, False, False,
       False, False,  True,  True, False, False, False,  True, False,
       False, False,  True, False, False, False,  True, False, False,
       False, False, False,  True, False, False, False,  True, False,
        True, False, False, False, False, False, False, False, False,
       False, False])
```

In [9]:

```
train_bream_smelt = train_scaled[bream_smelt_index]
train_target = train_target[bream_smelt_index]

print(train_scaled.shape)
print(train_bream_smelt.shape)
```

```
(119, 5)
(36, 5)
```

Logistic Regression 모델 fitting

In [10]:

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(train_bream_smelt, train_target)

lr.score(train_bream_smelt, train_target)
```

Out [10]: 1.0

- Test Data 필터링

In [11]:
```
test_bream_smelt_index = (test_target == 'Bream') | (test_target == 'Smelt')
test_bream_smelt_index
```

Out[11]:
```
array([False, False, False, False, False, False, False, False,  True,
       False, False, False,  True, False, False,  True, False,  True,
       False, False,  True, False, False, False, False, False, False,
        True,  True,  True,  True,  True, False,  True, False,
       False, False,  True, False])
```

In [12]:
```
print(lr.predict(train_bream_smelt[6:12]))
print(lr.predict_proba(train_bream_smelt[6:12])[:,].argmax(1))
print(lr.predict_proba(train_bream_smelt[6:12])[:,])


# train_bream_smelt[:5]
```

```
['Smelt' 'Bream' 'Bream' 'Smelt' 'Bream' 'Bream']
[1 0 0 1 0 0]
[[2.35400847e-02 9.76459915e-01]
 [9.94483928e-01 5.51607203e-03]
 [9.99387859e-01 6.12141469e-04]
 [3.36376065e-02 9.66362393e-01]
 [9.87489845e-01 1.25101548e-02]
 [9.83240385e-01 1.67596152e-02]]
```

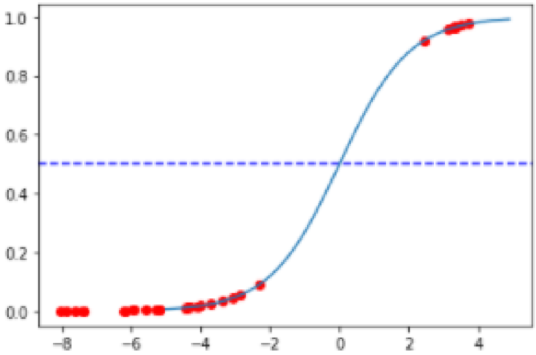
In [13]:
```
print(lr.coef_, lr.intercept_)
```

```
[[-0.4235112  -0.61604834 -0.70216369 -0.97498265 -0.7403996 ]] [-2.46732659]
```

In [14]:
```
test_bream_smelt = test_scaled[test_bream_smelt_index]
test_target = test_target[test_bream_smelt_index]

lr.score(test_bream_smelt, test_target)
```

Out[14]: 1.0

In [19]:
```

x = np.arange(-5,5,0.1)
y = 1 / (1+np.exp(-z))

point = lr.intercept_ + lr.coef_[0][0]*train_bream_smelt[:,0] +  lr.coef_[0][1]*train_bream_smelt[:
v_point = 1 / (1+np.exp(-point))
plt.axhline(0.5, linestyle = '--',color = 'b')
plt.plot(x,y)
plt.scatter(point,v_point,color = 'r')
plt.show()
```



In [20]:
```python
from sklearn.model_selection import train_test_split

train_input, test_input, train_target, test_target = train_test_split(
    fish_input,
    fish_target,
    random_state = 42,
    stratify = fish_target   # fish_target의 Class 비율에 맞게 split
)

print('train_shape: ',train_input.shape,'\ntest_shape : ',test_input.shape)

ss = StandardScaler()
ss.fit(train_input)

#data Transform
train_scaled = ss.transform(train_input)
test_scaled = ss.transform(test_input)  # 테스트 scale도 train data로 fit한 객체로 변환

train_scaled[:5]
```

```
train_shape:  (119, 5)
test_shape :  (40, 5)
```

Out [20]:
```
array([[-0.75628803, -0.66065677, -0.62357446, -0.78015159, -0.45043644],
       [-0.45991057, -0.1248453 , -0.24414603, -0.4293487 ,  0.03516919],
       [ 0.07356886,  0.0212851 ,  0.2165885 ,  0.79541208,  0.37481797],
       [ 1.54063728,  1.0441979 ,  1.23743166,  2.29283234,  1.34130358],
       [-0.87483902, -0.75807703, -0.82232269, -0.80672937, -0.5697143 ]])
```

In [21]:
```python
lr = LogisticRegression(C =10,max_iter = 1000) #C 1/Lambda

lr.fit(train_scaled, train_target)
```

Out [21]:
```
LogisticRegression(C=10, max_iter=1000)
```

In [22]:
```python
print(lr.score(train_scaled, train_target))
print(lr.score(test_scaled, test_target))
```

```
0.8991596638655462
0.925
```

In [23]:
```python
print(lr.predict(test_scaled[11:20]))
print(test_target[11:20])
```

```
['Perch' 'Bream' 'Perch' 'Smelt' 'Bream' 'Roach' 'Bream' 'Pike' 'Perch']
['Perch' 'Bream' 'Perch' 'Perch' 'Bream' 'Roach' 'Bream' 'Pike' 'Perch']
```

In [24]:
```python
lr.predict_proba(test_scaled[:5]).round(3)
```

Out [24]:
```
array([[0.001, 0.045, 0.321, 0.006, 0.58 , 0.012, 0.035],
       [0.001, 0.043, 0.562, 0.002, 0.351, 0.004, 0.036],
       [0.   , 0.078, 0.527, 0.002, 0.345, 0.024, 0.024],
       [0.008, 0.878, 0.005, 0.   , 0.09 , 0.002, 0.017],
       [0.003, 0.82 , 0.013, 0.   , 0.138, 0.008, 0.018]])
```

In [25]:
```python
print(lr.classes_)
```

```
['Bream' 'Parkki' 'Perch' 'Pike' 'Roach' 'Smelt' 'Whitefish']
```

확률적 경사하강법 (Stochastic gradient Descent)

Derivatives of cost function of logistic regression

확률적 경사하강법

Let's see the cases of cost function of logistic regression
This equation does not have a closed-form solution x

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y_i log(h_\theta(x_i)) + (1-y_i)log(1-h_\theta(x_i))$$ ← 근사값 구하기

where, $h_\theta(x_i) = \frac{1}{1+e^{-\theta x}}$, $y \in 0, 1$ ⇒ 경사하강법

Linear Regression

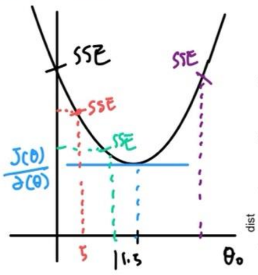
$J(\theta) = \sum_{i=1}^{n}(y_i - (\theta_0 + \theta_1 x_1 + \cdots + \theta_k x_n))^2$ closed-form

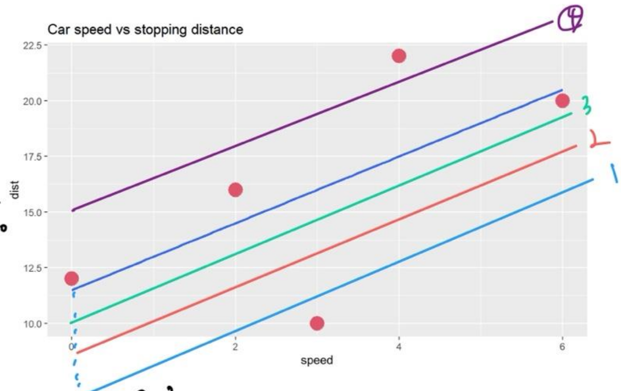
OLS

$\rightarrow \frac{J(\theta)}{\partial(\theta)} = 0$, $\frac{J(\theta)}{\partial(\theta_1)} = 0$

The concept of gradient descent (GD) algorithm

```
# A tibble: 5 x 2
  speed  dist
  <dbl> <dbl>
1     0    12
2     2    16
3     3    10
4     6    20
5     4    22
```






```
(Intercept)      speed
       11.5        1.5
```
← OLS

$y = 1.5x + 11.5$

[1] "Sum of squard error = 59"

$\Sigma(y - \hat{y}_i)^2$

$\Sigma(y_i - (1.5x))^2$

1. $y = 1.5x + 0$
2. $y = 1.5x + 5$
3. $y = 1.5x + 10$
4. $y = 1.5x + 15$

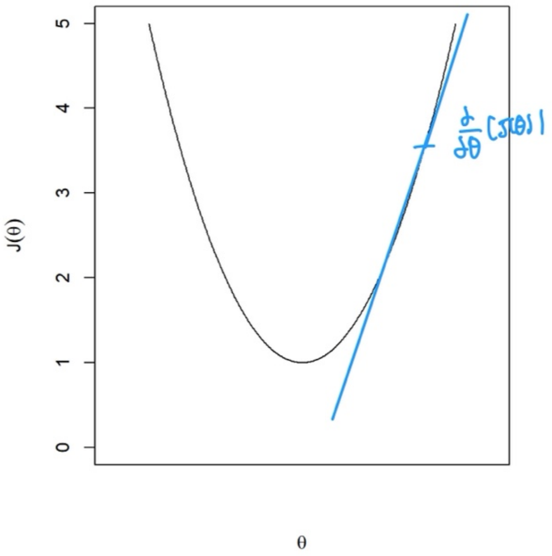
Gradient descent (GD) algorithm

- Objective (cost) function =
$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$
$= \frac{1}{2m}\sum_{i=1}^{m}(y_i - h_\theta(x_i))^2$

- Parameter update :
Repeat until convergence {

$$\theta_j^{(n+1)} = \theta_j^{(n)} - \gamma \frac{\partial}{\partial \theta_j} J(\theta^{(n)})$$

}



경사하강법

In [37]:
```
fish_input = fish[fish.columns[1:]].to_numpy()
fish_target = fish[fish.columns[0]].to_numpy()
```

In [38]:
```
train_input, test_input, train_target, test_target = train_test_split(
    fish_input,
    fish_target,
    stratify = fish_target,
)
```

In [39]:
```
ss = StandardScaler()
ss.fit(train_input)

train_scaled =  ss.transform(train_input)
test_scaled = ss.transform(test_input)
```

In [ ]:
```
from sklearn.linear_model import SGDClassifier

sc = SGDClassifier(
    loss = 'log',
    max_iter = 100,
    ) # logistic Regression Loss
```

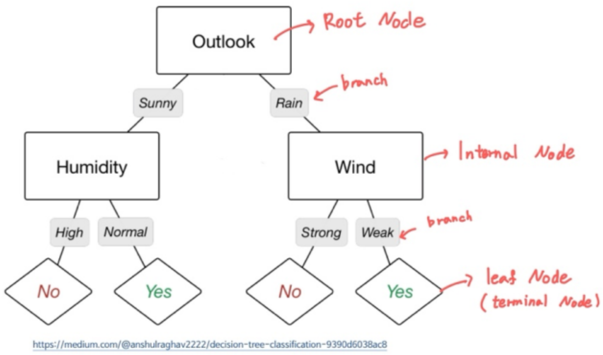
결정트리(Decision Tree)

Structure

- Decision tree for conditions to play tennis

outlook	humide	wind	T
Sunny	Normal	Strong	yes
Rain	Normal	week	No
:	:	:	:

↳ yes or No 를 자동으로 응답해주는



Partitioning rule of classification case

- Maximize the reduction in entropy or the Gini index
  - Gini index: $\boxed{I_G = 1 - \boxed{\sum_{j=1}^{c} p_j^2}}$, c is number of class → 감소되는 방향으로
    Impurity (불순도)
  - Entropy: $Entropy(p_j) = -\sum_{j=1}^{c} p_j log(p_j)$

Disease	Hypertension 고혈압	chestPain 가슴통증	Cholesterol 콜레스테롤
Yes	No	Yes	208
Yes	No	No	282
Yes	No	Yes	235
No	Yes	Yes	277
Yes	No	Yes	280
No	Yes	Yes	242
No	No	No	275
No	Yes	No	214
Yes	Yes	Yes	231
Yes	Yes	No	206

ex)

Hyper tension → yes: 5 [$\frac{2}{5}$ $\frac{3}{5}$], No: 5 [$\frac{4}{5}$ $\frac{1}{5}$]

$I_G = 1 - \left(\frac{6}{10}\right)^2 - \left(\frac{4}{10}\right)^2 = \frac{100-36-16}{100} = \frac{48}{100} = 0.48$

positive rate Negative rate

$I_G = 1 - \left(\frac{4}{5}\right)^2 - \left(\frac{1}{5}\right)^2 = 0.32$

$I_G = 1 - \left(\frac{2}{5}\right)^2 - \left(\frac{3}{5}\right)^2 = \frac{25-4-9}{25} = \frac{12}{25} = 0.48$

combine $I_G = 0.48 \times \frac{1}{2} + 0.32 \times \frac{1}{2} = 0.385$

Node1 Node2
$0.48 - 0.385 = 0.095$ → Information Gain

ex)

chest pain → $I_G = 0.48$

yes [$\frac{4}{6}$ $\frac{2}{6}$], No [$\frac{2}{4}$ $\frac{2}{4}$]

$I_G = 1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2 = 0.5$

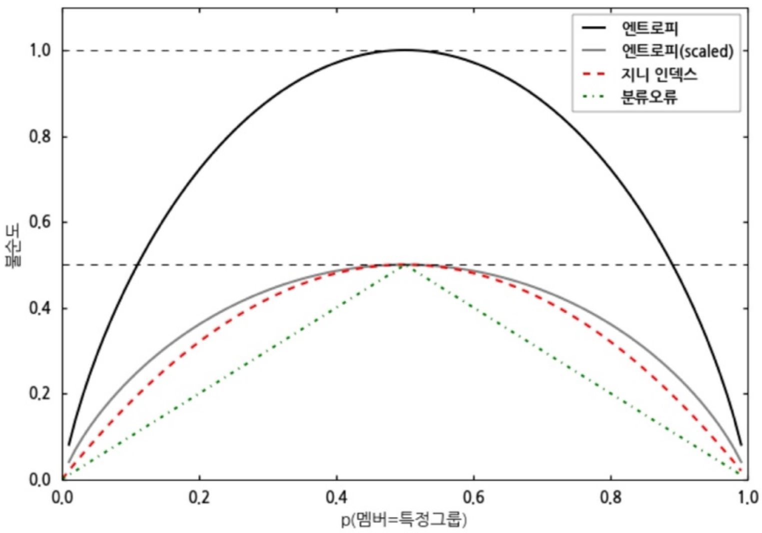
$I_G = 1 - \left(\frac{4}{6}\right)^2 - \left(\frac{2}{6}\right)^2 = 0.44$

weight-sum $= 0.44 \times 0.6 + 0.5 \times 0.4 = 0.464$

Information Gain $= 0.48 - 0.464 = 0.016$

Hypertension I_G > Chest pain I_G
0.095 0.016

Ginni and Entropy



https://m.blog.naver.com/samsjang/220978650404

The decision tree boundary of Iris data

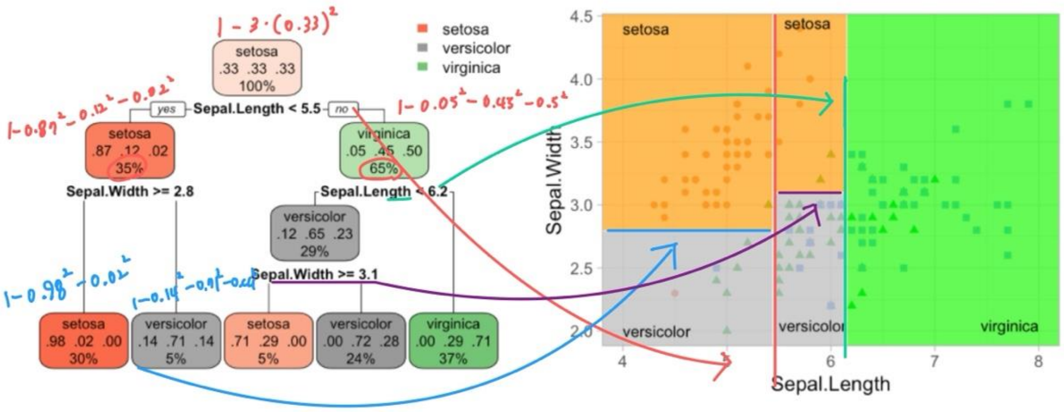


Figure 9.5: Decision tree for the iris classification problem (left). The decision boundary results in rectangular regions that enclose the observations. The class with the highest proportion in each region is the predicted value (right).

https://bradleyboehmke.github.io/HOML