지도 학습 : 분류

지도학습 : 분류

분류(Classification)란?

- 지도학습은 레이블(Label, 명시적인 정답)이 있는 데이터가 주어진 상태에서 학습하는 머신러닝
- 주어진 데이터의 피처(Feature)와 레이블 값을 머신러닝 알고리즘으로 학습해 모델을 생성
- 모델에 새로운 데이터 값이 주어지면 이 알 수 없는 레이블 값을 예측

분류 알고리즘

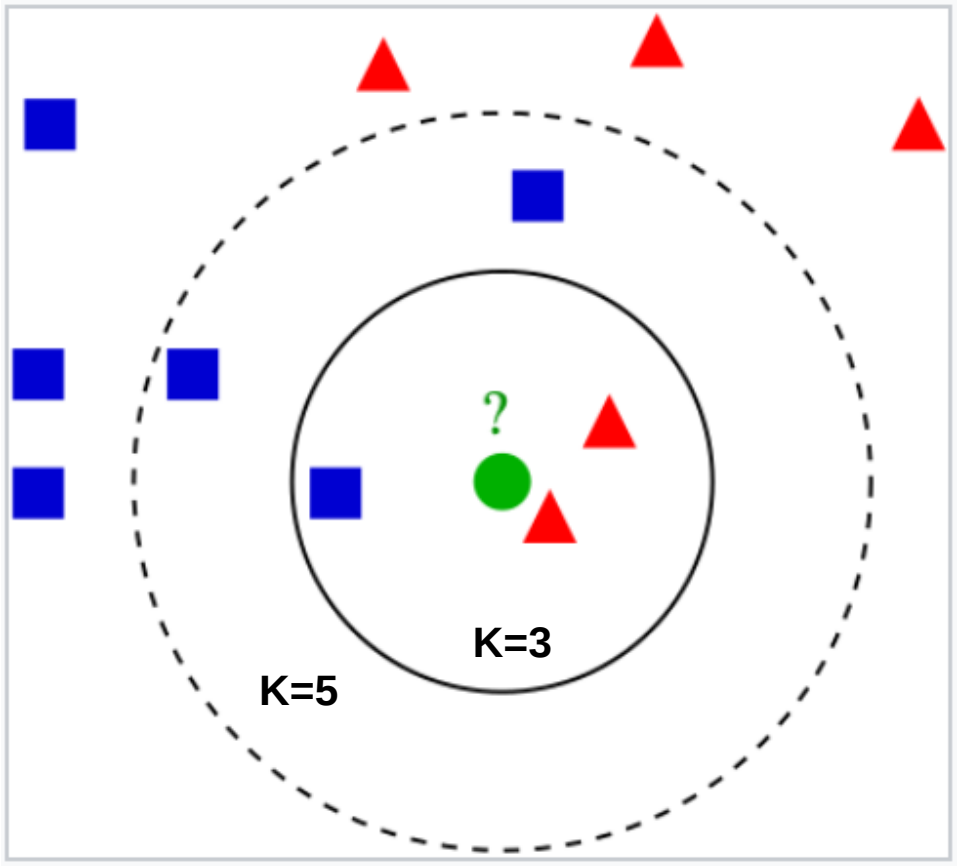
- 로지스틱 회귀(Logistic Regression) : 독립변수와 종속변수의 선형 관계성
- 결정 트리(Decision Tree) : 데이터 균일도에 따른 규칙
- 나이브 베이즈(Naïve-Bays) : 베이즈 통계와 생성 모델
- 최소 근접(Nearest Neighbor) : 근접 거리를 기준으로 하는 모델
- 신경망(Neural Network) : 심층 연결
- 서포트 벡터 머신(Support Vector Machine) : 개별 클래스 간의 최대 마진을 효과적 활용

분류 모델 평가

- 정확도(Accuracy)
- 오차행렬(Confusion Matrix)
- 정밀도(Preicision)
- 재현율(Recall)
- F1 스코어
- ROC 곡선과 AUC

지도학습 : 분류

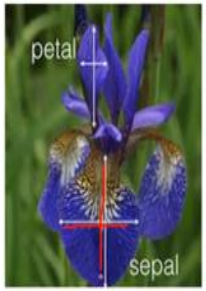
- K-최근접 이웃 알고리즘
  - 알고리즘의 훈련단계는 오직 훈련 표본이 특징 벡터와 항목 분류명으로 저장하는 것
  - k의 역할은 몇 번째로 가까운 데이터까지 살펴볼 것인가를 정한 숫자



지도학습 : 분류

- 실습 예제) 붓꽃 데이터 분류 - iris dataset

붓꽃 데이터 피처



- Sepal length
- Sepal width
- Petal length
- Petal width

붓꽃 데이터 품종(레이블)


Setosa


Vesicolor


Virginica

피처 / 레이블

학습 데이터

번호	꽃받침 길이	꽃받침 너비	꽃잎 길이	꽃잎 너비	Iris 꽃 종류
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
.....					....
50	6.4	3.5	4.5	1.2	Versicolor
.....					.....
150	5.9	3.0	5.0	1.8	Virginica

학습 데이터로 모델 학습

모델 학습

테스트 데이터

번호	꽃받침 길이	꽃받침 너비	꽃잎 길이	꽃잎 너비	Iris 꽃 종류는?
1	5.1	3.5	1.4	0.2	?
.....					?
50	6.4	3.5	4.5	1.2	?

학습 모델 통해 테스트 데이터의 레이블 값 예측

학습된 모델로 테스트 데이터 예측

예측된 레이블 값과 실제 레이블 값 예측 정확도 평가

평가

지도학습 : 분류

k-nn(최근접 이웃) 분석

1) 데이타셋 불러오기

패키지 설치

```
1  import numpy as np
2  import pandas as pd
3
4  import sklearn
5  import matplotlib.pyplot as plt
```

iris 데이타셋 불러오기

```
1  from sklearn import datasets
2
3  # dataset 불러오기
4  iris = datasets.load_iris()
```

```
1  # iris 데이타셋 살펴보기
2  irisdt = pd.DataFrame(iris['data'], columns = iris['feature_names'])
3  print(irisdt.shape)
```

```
(150, 4)
```

```
1  irisdt.head(2)
```

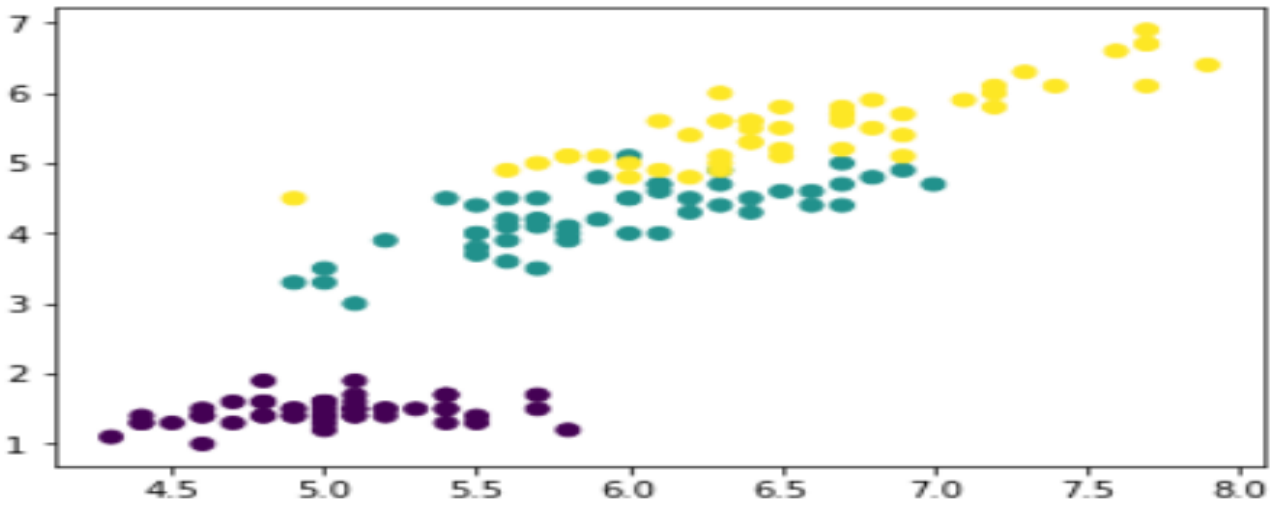
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2

지도학습 : 분류

- k-nn(최근접 이웃) 분석
  - 2) 데이터 시각화 - 산점도 그리기

```
1  # 산점도 그리기
2  import matplotlib.pyplot as plt
3
4  %matplotlib inline
5
6  plt.scatter(irisdt['sepal length (cm)'],
7              irisdt['petal length (cm)'],
8                    c = iris['target'])
```

```
<matplotlib.collections.PathCollection at 0x26786db9278>
```



지도학습 : 분류

- k-nn(최근접 이웃) 분석
  - 3) 사이킷런 라이브러리를 이용한 k-nn 알고리즘 적용
    - (1) 전체 데이타에 적용

```
1  import pandas as pd
2  import numpy as np
3  from sklearn import neighbors
4
5  # 1. 모델 적용
6  knn = neighbors.KNeighborsClassifier()
7
8  # 2. 모델 훈련 - 4개의 특징(변수)을 가지고 3개로 분류(0, 1, 2)
9  knn.fit(iris['data'], iris['target'])
10
11 # 3. 모델 정확도 평가
12 knn.score(iris['data'], iris['target'])
```

```
0.9666666666666667
```

```
1  # 모델 정확도 확인 : 예측값 vs 정답
2  pre = knn.predict(iris['data'])  # 예측값(0, 1, 2)
3  iris['target'] # 정답(0, 1, 2)
4
5  pre == iris['target'] # 150개 중 5개 틀림
```

```
1  # 정확도 계산
2  145/150
```

```
0.9666666666666667
```

지도학습 : 분류

- k-nn(최근접 이웃) 분석
  - 3) 사이킷런 라이브러리를 이용한 k-nn 알고리즘 적용
    - (2) 학습/테스트 데이터 나누어 진행

```
from sklearn.model_selection import train_test_split

iris_x = iris.data      # train data
iris_y = iris.target    # test data

x_train, x_test, y_train, y_test = train_test_split(iris_x, iris_y,
                                                    test_size = 0.2, random_state = 0)
```

```
len(x_train), len(x_test)
```

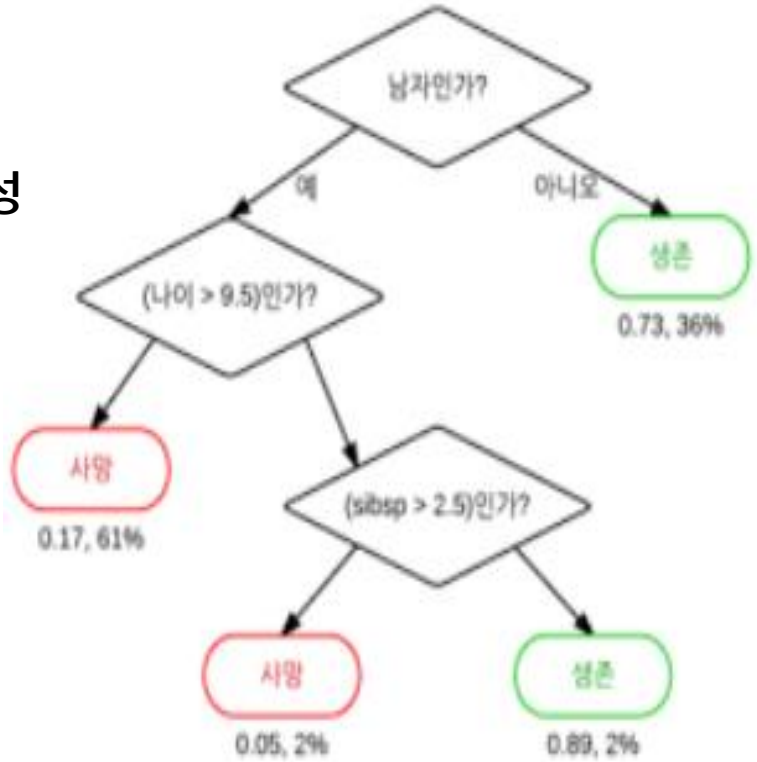
```
(120, 30)
```

```
import pandas as pd
import numpy as np
from sklearn import neighbors

# 1. 모델 적용
knn = neighbors.KNeighborsClassifier()

# 2. 모델 훈련 - 4개의 특징(변수)을 가지고 3개로 분류(0, 1, 2)
knn.fit(x_train, y_train)

# 3. 모델 정확도 평가
knn.score(x_test, y_test)
```

```
0.9666666666666667
```

지도학습 : 분류

- Decision Tree(의사 결정 나무)
  - 가장 직관적인 알고리즘으로 이해가 쉬움
  - 입력변수를 바탕으로 목표 변수의 값을 예측하는 모델을 생성



〈타이타닉 탑승객의 생존여부 결정트리〉

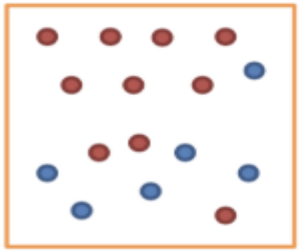
  - 장점
    - 결과를 해석하고 이해하기가 쉽다.
    - 자료를 가공(정규화, 결측치, 이상치)할 필요가 거의 없다.
    - 수치 자료와 범주 자료 모두에 적용할 수 있다.
    - 화이트박스 모델을 사용한다.
    - 대규모의 데이터 셋에도 잘 동작한다.
  - 단점
    - 한 번에 하나의 변수만을 고려하므로 변수간 상호작용을 파악하기가 어렵다.
    - 결정경계(Decision Boundary)가 데이터 축에 수직이어서 비선형(Non-Linear) 데이터 분류에는 적합하지 않다.
    - Full Tree일 경우 오버피팅이 발생하기 쉽다.

지도학습 : 분류

Decision Tree(의사 결정 나무)

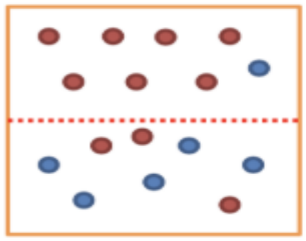
결정 트리 분기의 기준 - 지니 계수, 엔트로피

- 각 영역의 **순도(homogeneity)**가 증가, **불순도(impurity)** 혹은 **불확실성(uncertainty)**이 최대한 감소하도록 하는 방향으로 학습을 진행
- 순도가 증가/불확실성이 감소하는 걸 두고 정보이론에서는 **정보획득(information gain)**이라고 함
- 1) **지니계수**(Gini Index)
  - **불순도(Impurity)와 다양성(Diversity)를 계산하는 방법**
  - 영역내에서 특정 클래스에 속하는 관측치들의 비율을 제외한 값 : $I(A) = 1 - \sum_{k=1}^{m} p_k^2$



$$I(A) = 1 - \sum_{k=1}^{m} p_k^2$$
$$= 1 - \left(\frac{6}{16}\right)^2 - \left(\frac{10}{16}\right)^2$$
$$\approx 0.47$$

- 2개 이상의 영역에서의 식 : $I(A) = \sum_{i=1}^{d} \left(R_i \left(1 - \sum_{k=1}^{m} p_{ik}^2\right)\right)$



$$I(A)$$
$$= 0.5 \times \left(1 - \left(\frac{7}{8}\right)^2 - \left(\frac{1}{8}\right)^2\right) + 0.5 \times \left(1 - \left(\frac{3}{8}\right)^2 - \left(\frac{5}{8}\right)^2\right)$$
$$= 0.34$$

- 분기 후 정보 획득량 (Information Gain) 은 0.47 - 0.34 = 0.13

지도학습 : 분류

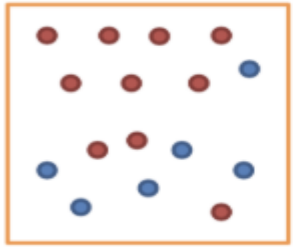
Decision Tree(의사 결정 나무)

결정 트리 분기의 기준

2) 엔트로피(entropy)

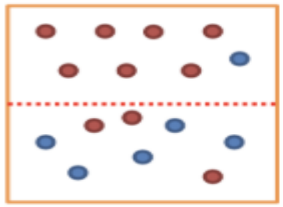
- 엔트로피는 열역학에서 나온 개념으로써 무질서에 대한 측도 역할을 한다.
- 지니 지수와 비슷하지만 log를 취함으로써 정규화 과정을 거치게 된다.

$$Entropy(A) = -\sum_{k=1}^{m} p_k \log_2(p_k)$$ Pk는 한 영역에 속한 데이터 중 k범주에 속하는 데이터의 비율., m은 범주의 개수



$$\begin{aligned} E(A) &= -\sum_{k=1}^{m} p_k \log_2(p_k) \\ &= -\frac{6}{16}\log_2\left(\frac{6}{16}\right) - \frac{10}{16}\log_2\left(\frac{10}{16}\right) \\ &\approx 0.95 \end{aligned}$$

- **두 개 이상 영역**



$$\begin{aligned} &E(A) \\ &= 0.5 \times \left(-\frac{1}{8}\log_2\left(\frac{1}{8}\right) - \frac{7}{8}\log_2\left(\frac{7}{8}\right)\right) \\ &\quad + 0.5 \times \left(-\frac{3}{8}\log_2\left(\frac{3}{8}\right) - \frac{5}{8}\log_2\left(\frac{5}{8}\right)\right) \\ &= 0.75 \end{aligned}$$

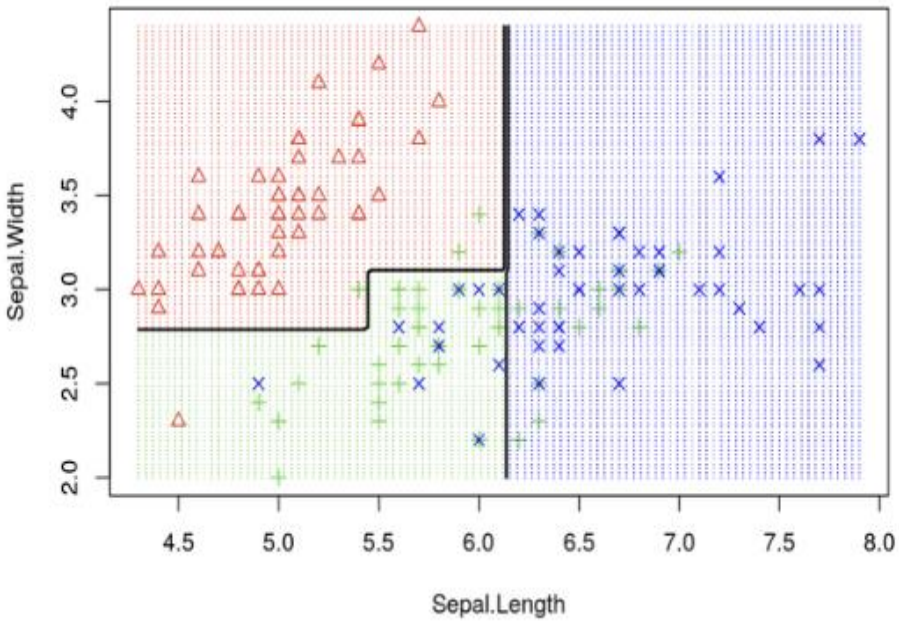
- **분기 후 정보 획득량 (Information Gain) 은 0.95 - 0.75 = 0.20 이다**

지도학습 : 분류

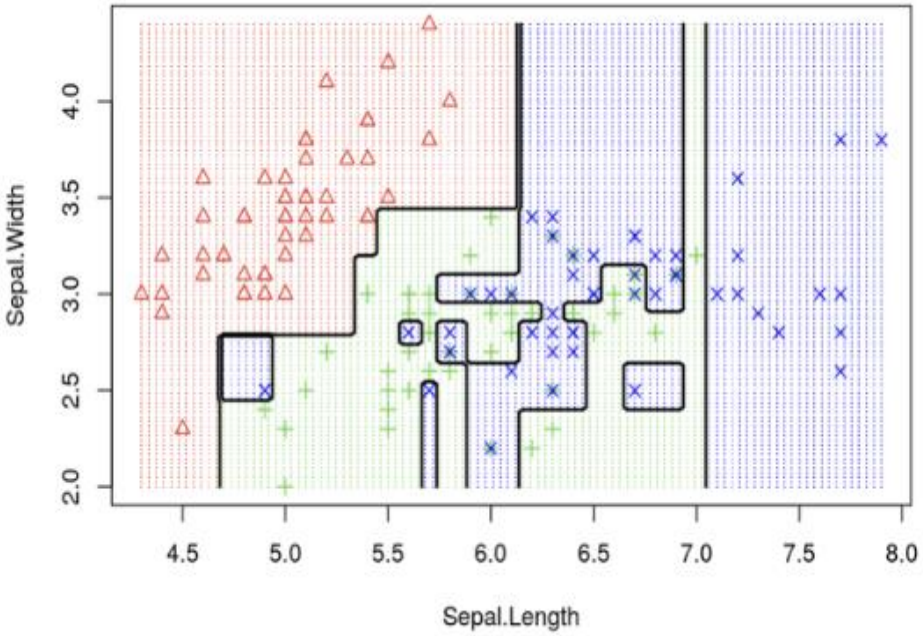
- Decision Tree(의사 결정 나무)
- 의사 결정 트리모델 학습 과정
  - 1) 재귀적 분기(recursive partitioning)
    - 재귀적 분기는 특정 영역인 하나의 노드 내에서 하나의 변수 값을 기준으로 분기하여 새로 생성된 자식 노드들의 동질성이 최대화되도록 분기점을 선택하는 것이다.
    - 동질성이 최대화된다는 말은 즉, 불순도는 최소화 된다는 의미이다.
    - **불순도를 측정하는 기준**
      - **범주형 변수 : 지니 계수 (Gini Index)**
      - **수치형 변수 : 분산 (Variance)**
  - 2) 가지치기 (Pruning)
    - 가지치기란 나무의 가지를 치는 것과 같아서 이름이 붙여졌으며, 가지를 잘라서 버리는 것이 아닌, 합치는 것이라고 이해해야 한다.
    - Decision Tree 분기 수가 증가할 때, 초기에는 오류율이 감소하지만 어느 시점부터는 증가하기 시작한다.
    - full tree란 Terminal node의 순도가 100%인 상태인 트리를 말한다.
    - 보통 full tree의 경우 분기가 너무 많아 과적합 (Overfitting)이 발생하기 쉽다. 따라서 먼저 full tree를 생성한 뒤 적절한 수준에서 하위 노드와 상위 노드를 결합해주어야 한다.

지도학습 : 분류

- Decision Tree(의사 결정 나무)
- 의사 결정 트리모델 학습 과정
  - 2) 가지치기 (Pruning)



적절한 가지치기를 한 뒤의 Tree



Full -Tree
Terminal node의 순도가 100%인 상태인 트리
→ 오버피팅(과적합)

지도학습 : 분류

■ 실습) Decision Tree(의사 결정 나무)

1) 모델 불러오기 및 정의하기

```
1  from sklearn.tree import DecisionTreeClassifier
2
3  clf = DecisionTreeClassifier(random_state = 0)
```

2) 모델 학습하기(훈련 데이타)

```
1  clf.fit(x_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

3) 예측하기(테스트 데이타)

```
1  y_pred = clf.predict(x_test)
```

4) 모델 평가 : 정확도(Accuracy)

```
1  clf.score(x_test, y_test)
```

```
1.0
```

지도학습 : 분류

- 실습) Decision Tree(의사 결정 나무)

결정 트리 모델의 시각화(Decision Tree Visualization)

```
from sklearn.tree import export_graphviz

# export_graphviz()의 호출 결과로 out_file로 지정된 tree.dot 파일을 생성함.
export_graphviz(clf, out_file="./tree.dot", class_names=iris.target_names , ₩
feature_names = iris.feature_names, impurity=True, filled=True)
```

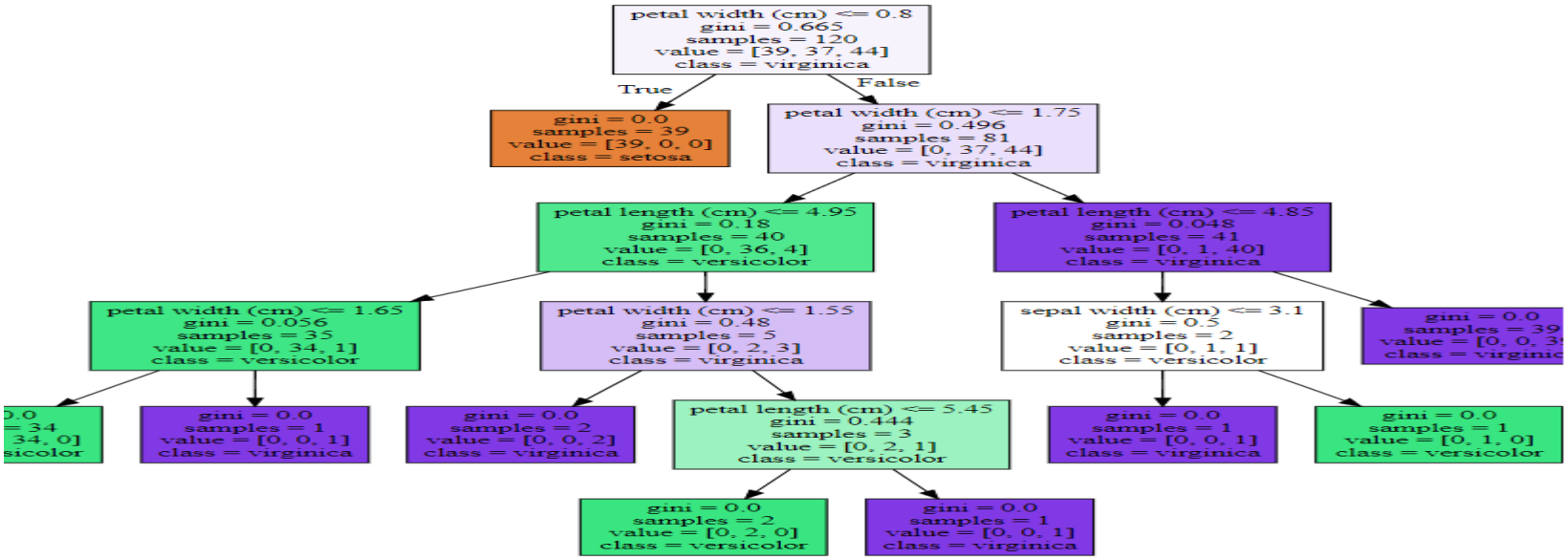
graphviz 설치

1. 파일 다운로드 : https://graphviz.gitlab.io/_pages/Download/Download_windows.html
2. pip install graphviz
3. 환경변수 설정 - PATH에 새로만들기로 path 추가

```
import graphviz

# 위에서 생성된 tree.dot 파일을 Graphviz 읽어서 Jupyter Notebook상에서 시각화
with open("./tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

지도학습 : 분류

- 실습) Decision Tree(의사 결정 나무)



그래프 설명

- petal width (cm) <= 0.8와 같이 피처의 조건이 있는 것은 자식 노드를 만들기 위한 규칙 조건, 조건이 없으면 리프 노드
- gini는 다음의 value=[]로 주어진 데이타 분포의 지니 계수
- samples는 현 규칙에 해당하는 데이타 건수
- value = []는 클래스 값 기반의 데이타 건수
- iris 데이타 세트는 클래스 값으로 0, 1, 2를 가지고 있으며,
  - 0 : Setosa, 1:Versicolor, 2: Virginica 품종을 가리킴
- 만일 Value=[41,40,39]라면 클래스 값의 순서로 Setosa 41개, Versicolor 40개, Virginica 39개로 데이타가 구성
- 색깔이 짙어질수록 지니계수가 낮고 해당 레이블에 속하는 샘플 데이타가 많다는 의미

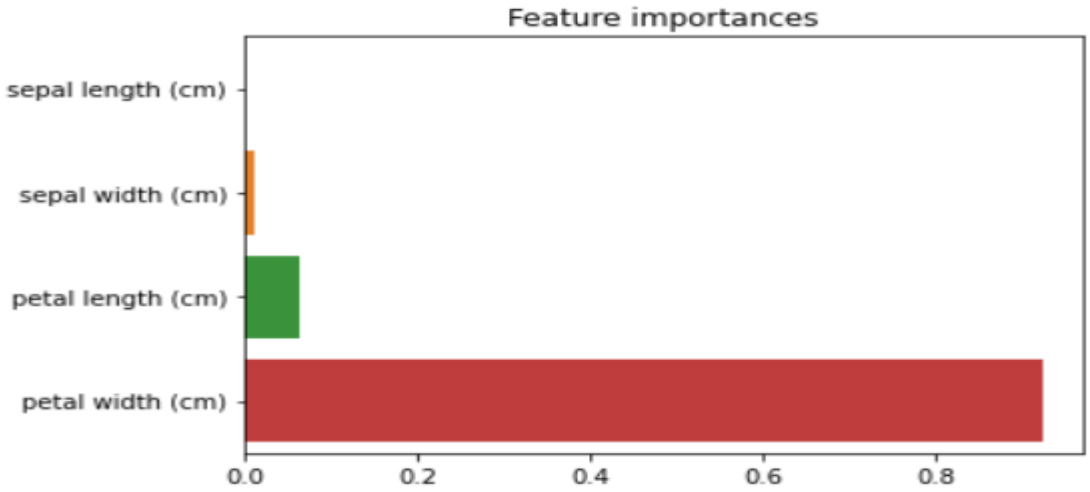
지도학습 : 분류

- 실습) Decision Tree(의사 결정 나무)

피처 중요도 시각화

```
1  import seaborn as sns
2  import numpy as np
3  %matplotlib inline
4
5  # feature importance 추출
6  print("Feature importances:₩n{0}".format(np.round(clf.feature_importances_, 3)))
7
8  # feature별 importance 매핑
9  for name, value in zip(iris.feature_names , clf.feature_importances_):
10     print('{0} : {1:.3f}'.format(name, value))
11
12 # feature importance를 column 별로 시각화 하기
13 sns.barplot(x=clf.feature_importances_ , y=iris.feature_names)
14 plt.title('Feature importances')
```

```
Feature importances:
[0.    0.013 0.064 0.923]
sepal length (cm) : 0.000
sepal width (cm) : 0.013
petal length (cm) : 0.064
petal width (cm) : 0.923
```



지도학습 : 분류

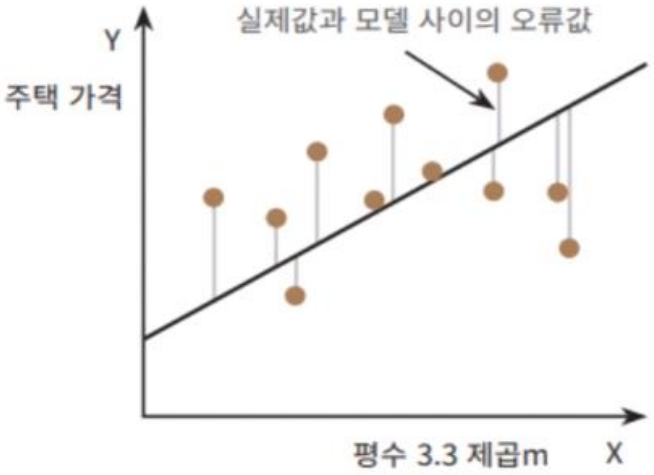
- 나이브 베이즈(Navie Bayes)
  - 일반적인 원칙에 근거한 여러 알고리즘들을 이용하여 훈련

  - 장점
    - 일부의 확률 모델에서 나이브 베이즈 분류는 지도학습 환경에서 매우 효율적인 훈련
    - 분류에 필요한 파라미터를 추정하기 위한 트레이닝 데이터 양이 매우 적다.
    - 간단한 디자인과 단순한 가정임에도 불구하고 많은 복잡한 실제 상황에서 잘 적용

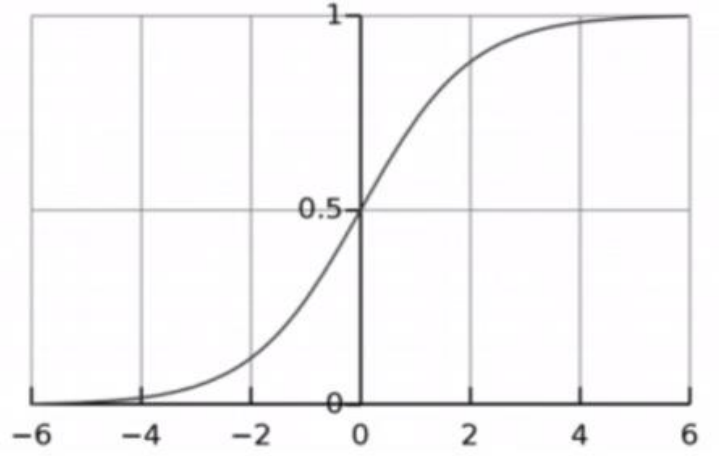
  - 단점
    - 나이브베이즈 분류는 독립성 가정이 종종 부정확한 결과를 내기도 한다.

지도학습 : 분류

- 로지스틱 회귀(Logistic Regression)
  - 선형 회귀 방식을 분류에 적용한 알고리즘
  - 선형 회귀가 학습을 통해 선형 함수의 회귀 최적선을 찾는 것과 로지스틱 회귀는 시그모이드(sigmoid) 함수 최적선을 찾고 이 시그모이드 함수의 반환 값을 확률로 간주해 확률에 따라 분류를 결정
  - 시그모이드 함수 $y = \frac{1}{1+e^{-x}}$
  - 시그모이드 함수는 x값이 +∞이거나 -∞여도 y 값은 항상 0과 1사이의 값을 반환
  - X값이 커지면 1에 근사, x값이 작아지면 0에 근사, x가 0일 때는 0.5



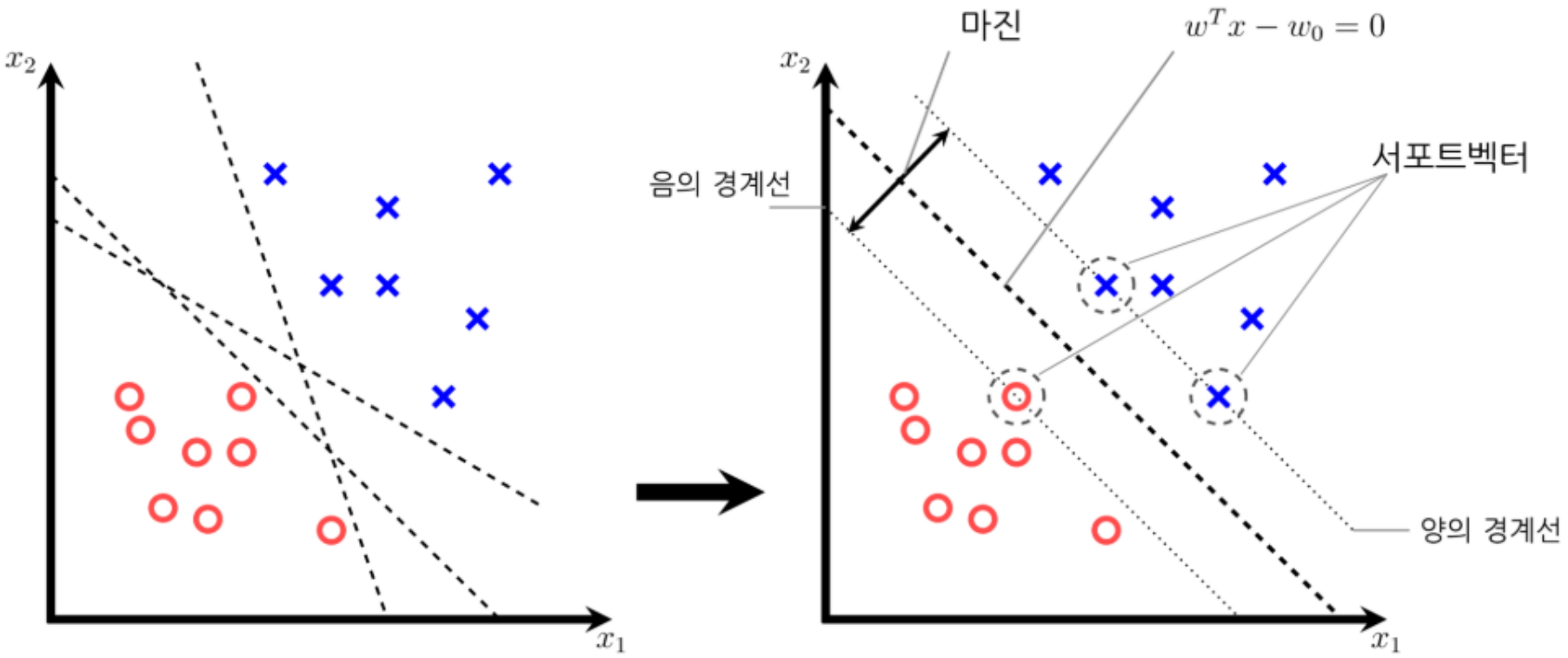
〈 선형 회귀의 선형 함수 〉



〈 로지스틱 회귀의 시그모이드 함수 〉

지도학습 : 분류

- SVC(Support Vector Model)
  - 초평면이 가장 가까운 학습 데이터 점과 큰 차이를 가지고 있으면 분류 오차가 작음
  - 좋은 분류를 위해서는 어떤 분류된 점에 대해서 가장 가까운 학습 데이터와 가장 먼 거리를 가지는 초평면 찾음



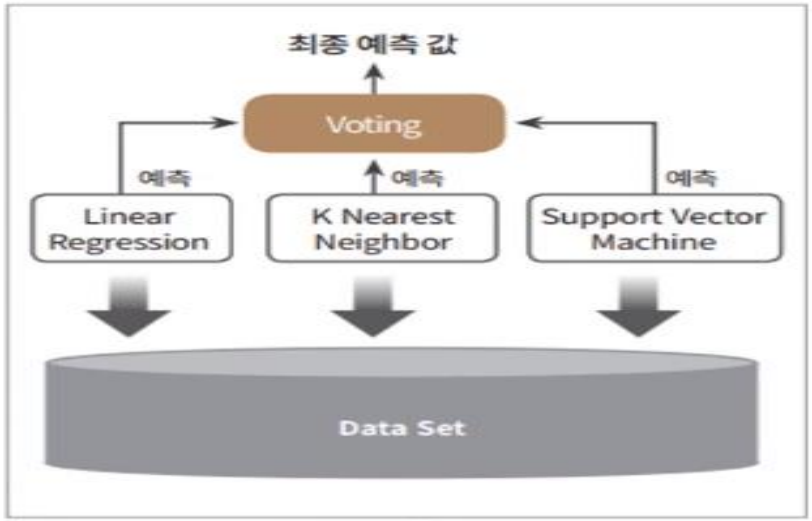
지도학습 : 분류 앙상블

- **분류 앙상블 학습**
  - 여러 개의 약한 분류기를 생성하여 그 예측을 결합
  - 단일 분류기보다 신뢰성이 높은 예측 값을 얻는 것이 목표
- **앙상블의 3가지 기법**

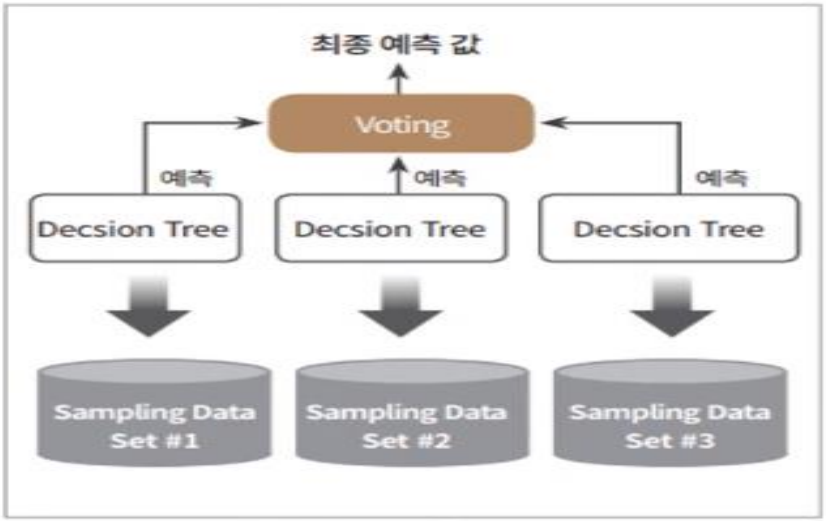
  1) 보팅 : 같은 데이터 셋을 학습한 서로 다른 모델링으로부터 투표 방식으로 결과 추출

  2) 배깅 : 서로 다른 독립된 데이터 셋으로 학습한 같은 모델링으로부터 투표방식으로 결과추출

  3) 부스팅 : 서로 다른 독립된 데이터 셋으로 학습한 서로 다른 모델로부터 가중치에 의한 투표 방식으로 결과 추출



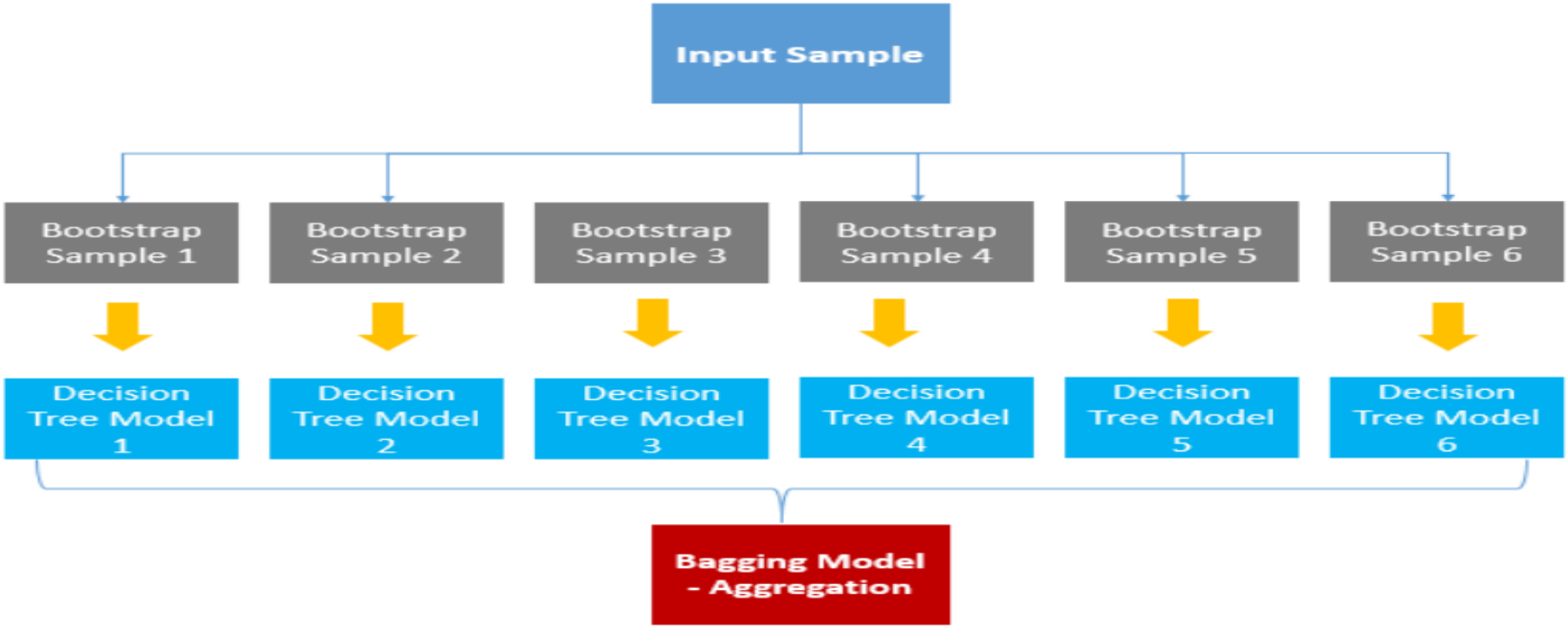
Voting 방식



Bagging 방식

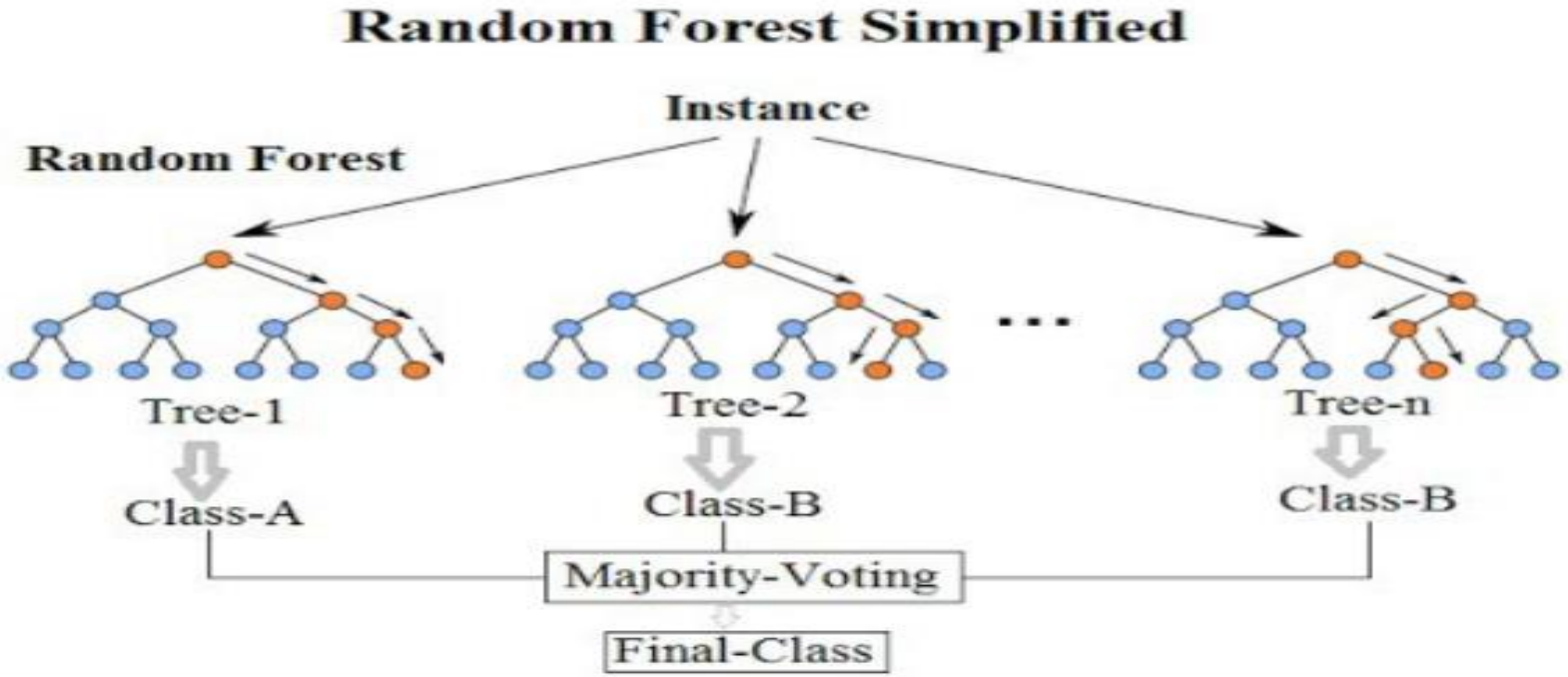
지도학습 : 분류 앙상블

- 배깅(bagging)
  - 서로 다른 독립된 데이터 셋으로 학습한 같은 모델링으로부터 투표방식으로 결과추출
  - 샘플을 여러 번 뽑아 각 모델을 학습시켜 결과를 집계(Aggregating) 하는 방법
  - 랜덤포레스트((Random Forest)



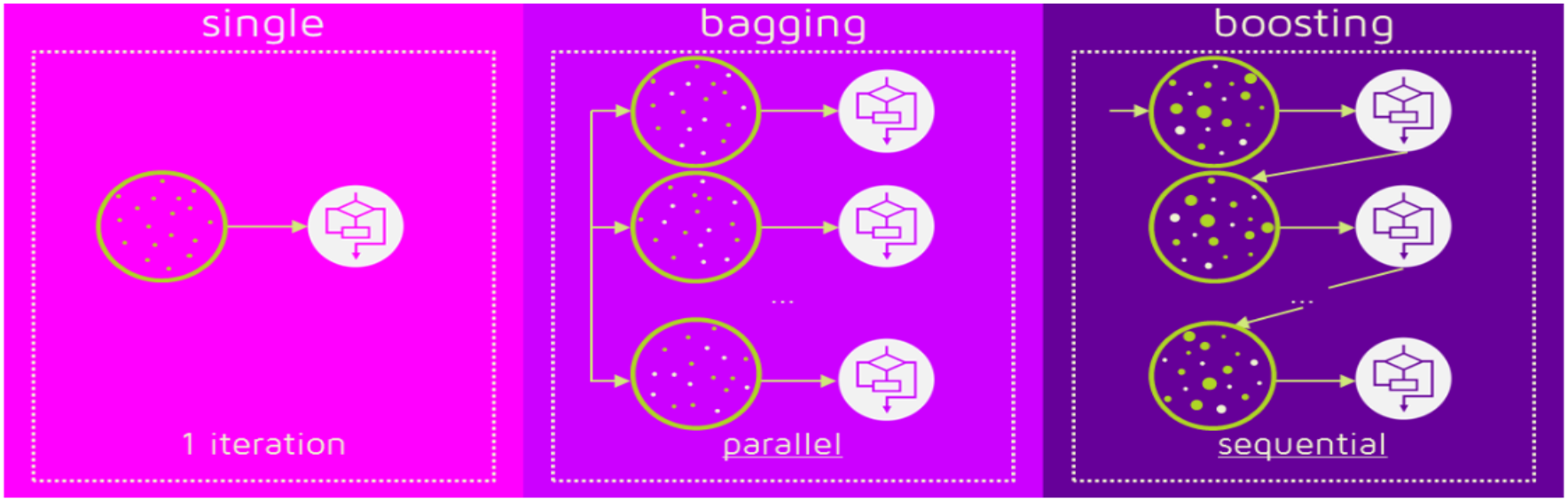
지도학습 : 분류 앙상블

- 랜덤 포레스트(Random Forest)
  - 배깅의 가장 대표적인 알고리즘
  - 앙상블 알고리즘 중 비교적 빠른 수행 속도
  - 다양한 영역에서 높은 정확도
  - 결정 트리 기반의 알고리즘



지도학습 : 분류 앙상블

- 부스팅(Boosting)
  - Bagging이 일반적인 모델을 만드는데 집중되어 있다면, Boosting은 맞추기 어려운 문제를 맞추는데 초점이 맞춰짐
  - Bagging과 동일하게 복원 랜덤 샘플링을 하지만, 가중치를 부여한다는 차이점
  - Bagging이 병렬로 학습하는 반면, Boosting은 순차적으로 학습하여, 학습이 끝나면 나온 결과에 따라 가중치가 재분배됨
  - AdaBoost, XGBoost, GradientBoost, XGBoost 모델은 강력한 성능을 보여줌



지도학습 : 분류

■ 분류 모델의 비교 - iris dataset 활용

```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd

import sklearn
import matplotlib.pyplot as plt
```

- 데이타 로드

```
from sklearn import datasets
from sklearn.model_selection import train_test_split

# dataset 불러오기
iris = datasets.load_iris()
```

- 학습/테스트 데이타 분리

```
iris_x = iris.data      # train data
iris_y = iris.target    # test data

x_train, x_test, y_train, y_test = train_test_split(iris_x, iris_y,
                                                    test_size = 0.2, random_state = 0)
```

지도학습 : 분류

- 분류 모델의 비교 - iris dataset 활용

1) 의사결정나무(Decision Tree)

```
result = dict()  # 각 모델마다 정확도 저장하기 위해 딕셔너리 선언

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state = 0)
clf.fit(x_train, y_train)
result['Decision Tree'] = clf.score(x_test, y_test)
result['Decision Tree']
```

```
1.0
```

2) 나이브베이즈

```
from sklearn.naive_bayes import GaussianNB

clf = GaussianNB()
clf.fit(x_train, y_train)
result['GaussianNB'] = clf.score(x_test, y_test)
result['GaussianNB']
```

```
0.9666666666666667
```

지도학습 : 분류

- 분류 모델의 비교 - iris dataset 활용

3) K-근접이웃

```
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier()
clf.fit(x_train, y_train)
result['KNN'] = clf.score(x_test, y_test)
result['KNN']
```

```
0.9666666666666667
```

4) 로지스틱 회귀

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
clf.fit(x_train, y_train)
result['LR'] = clf.score(x_test, y_test)
result['LR']
```

```
1.0
```

지도학습 : 분류

- 분류 모델의 비교 - iris dataset 활용

5) SVC

```
1  from sklearn.svm import SVC
2
3  clf = SVC()
4  clf.fit(x_train, y_train)
5  result['SVC'] = clf.score(x_test, y_test)
6  result['SVC']
```

```
1.0
```

6) RandomForest

```
1  from sklearn.ensemble import RandomForestClassifier
2
3  clf = RandomForestClassifier()
4  clf.fit(x_train, y_train)
5  result['RandomForestClassifier'] = clf.score(x_test, y_test)
6  result['RandomForestClassifier']
```

```
1.0
```

지도학습 : 분류

- 분류 모델의 비교 - iris dataset 활용

7) boosting

```
1  from sklearn.ensemble import GradientBoostingClassifier
2
3  clf = GradientBoostingClassifier()
4  clf.fit(x_train, y_train)
5  result['GradientBoostingClassifier'] = clf.score(x_test, y_test)
6  result['GradientBoostingClassifier']
```

```
1.0
```

8) xgboost

```
1  # pip install xgboost
2  from xgboost import XGBClassifier
3  clf =XGBClassifier()
4  clf.fit(x_train, y_train)
5  result['XGBClassifier'] = clf.score(x_test, y_test)
6  result['XGBClassifier']
```

```
1.0
```

지도학습 : 분류

- 분류 모델이 비교 - iris dataset 활용

분류 모델 성능평가 비교

```
# 전체 성능평가 결과
result
```

```
{'Decision Tree': 1.0,
 'GaussianNB': 0.9666666666666667,
 'KNN': 0.9666666666666667,
 'LR': 1.0,
 'SVC': 1.0,
 'RandomForestClassifier': 1.0,
 'GradientBoostingClassifier': 1.0,
 'XGBClassifier': 1.0}
```

```
# pandas로 결과보기
import pandas as pd
pd.Series(result)
```

```
Decision Tree                 1.000000
GaussianNB                    0.966667
KNN                           0.966667
LR                            1.000000
SVC                           1.000000
RandomForestClassifier        1.000000
GradientBoostingClassifier    1.000000
XGBClassifier                 1.000000
dtype: float64
```

지도학습 : 분류 모델 평가

분류 모델 평가

정확도(Accuracy)

$$\text{정확도(Accuracy)} = \frac{\text{예측 결과가 동일한 데이터 건수}}{\text{전체 예측 데이터 건수}}$$

- 실제 데이터에서 예측 데이터가 얼마나 같은지 판단
- 정확도는 직관적으로 모델 예측 성능을 나타내는 평가 지표
- 이진 분류의 경우 데이터의 구성에 따라 ML 모델의 성능을 왜곡할 수 있기 때문에 정확도 수치 하나만 가지고 성능을 평가하지 않는다.
- 특히 정확도는 불균형한(imbalanced) 레이블 값 분포에서 ML 모델의 성능을 판단할 경우, 적합한 평가지표가 아니다.

지도학습 : 분류 모델 평가

- 분류 모델 평가
  - 오차행렬(Confusion Matrix)
    - 이진 분류의 예측 오류가 얼마인지와 더불어 어떠한 유형의 예측 오류가 발생하고 있는지를 함께 나타내는 지표

	예측 클래스(Predicted Class)	
	Negative(0)	Positive (1)
Negative(0)	TN (True Negative)	FP (False Positive)
실제 클래스 (Actual Class) Positive(1)	FN (False Negative)	TP (True Positive)

정확도 (Accuracy)	$\frac{TP+TN}{TP+TN+FP+FN}$	특이도 (Specificity)	$\frac{TN}{TN+FP}$
민감도 (Sensitivity)	$\frac{TP}{TP+FN}$	정밀도 (Precision)	$\frac{TP}{TP+FP}$

$$\text{F1-Score} = \frac{2*Precision*Recall}{Precision+Recall}$$

지도학습 : 분류 모델 평가

- 분류 모델 평가 시 사용하는 지표

- 정확도(Accuracy)
  - 모든 예측 중 옳게 표현한 비율

$$\text{정확도} = \frac{TP + TN}{TP + TN + FP + FN}$$

- 정밀도(Precision)
  - 긍정으로 표시한 것 중 옳게 표시한 비율
  - 실제 Negative 음성 데이터 예측을 Positive 양성으로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우
  - 스팸메일을 분류하는 모델은 실제 Positive인 스팸 메일을 Negative인 일반 메일로 분류하더라도 사용자는 불편함을 느끼는 정도이지만, 실제 Negative인 일반 메일을 Positive인 스팸메일로 분류할 경우는 메일을 아예 받지 못하게 되어 업무에 차질이 생기는 경우

$$\text{정밀도} = \frac{TP}{TP + FP}$$

- 재현율(Recall) = 민감도
  - 답이 긍정적인 것들 중 옳게 표시한 비율
  - 실제 Positive 양성 데이터를 Negative로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우
  - 금융 사기나 암 진단 과 같이 반드시 탐지가 필요한 경우
  - 잘못 지나치면 큰 실수가 됨

$$\text{재현율} = \frac{TP}{TP + FN}$$

- F1 스코어
  - 정밀도와 재현율을 결합한 지표
  - 정밀도와 재현율의 차이가 없을 때 높은 값을 가짐

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

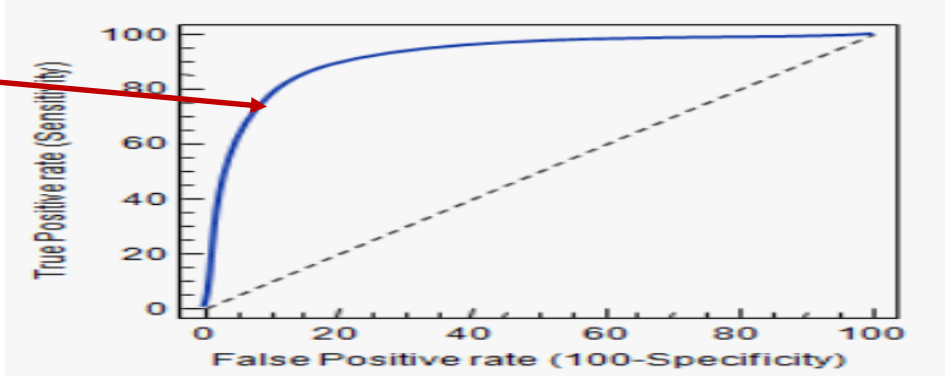
지도학습 : 분류 모델 평가

- 분류 모델 평가 시 사용하는 지표
  - ROC-AUC
    - 2차 대전 때 통신 장비 성능 평가를 위해 고안된 수치, 의학 분야에 많이 사용
    - 이진 분류 성능 평가를 위해 사용되는 중요한 지표
    - FPR(False Positive Rate) 변할 때 TPR(True Positive Rate,재현율, 민감도)이 어떻게 변하는지를 나타내는 곡선

  - ROC 곡선
    - Receiver Operation Characteristic Curve(ROC) 곡선 : 수신자 판단 곡선
    - FPR(False Positive Rate) = FP / (FP + TN) = 1 - TNR = 1 - 특이도

  - AUC(Area Under Curve)
    - ROC 커브의 밑 면적이 1에 가까울수록(즉, 왼쪽 상단 꼭지점에 다가갈수록) 좋은 성능
    - AUC 가 0.5(점선)면 성능이 전혀 없음, 1이면 최고의 성능

ROC 곡선



- ROC 곡선 - 사이킷런 roc_curve() 함수 제공
- AUC 스코어 - 사이킷런 roc_auc_score() 함수 제공

사이킷런 모델 평가

- 사이킷런 모델 평가
  - https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics

Classification metrics

See the Classification metrics section of the user guide for further details.

metrics.accuracy_score(y_true, y_pred[, ...])	Accuracy classification score.
metrics.auc(x, y[, reorder])	Compute Area Under the Curve (AUC) using the trapezoidal rule
metrics.average_precision_score(y_true, y_score)	Compute average precision (AP) from prediction scores
metrics.balanced_accuracy_score(y_true, y_pred)	Compute the balanced accuracy
metrics.brier_score_loss(y_true, y_prob[, ...])	Compute the Brier score.
metrics.classification_report(y_true, y_pred)	Build a text report showing the main classification metrics
metrics.cohen_kappa_score(y1, y2[, labels, ...])	Cohen's kappa: a statistic that measures inter-annotator agreement.
metrics.confusion_matrix(y_true, y_pred[, ...])	Compute confusion matrix to evaluate the accuracy of a classification
metrics.f1_score(y_true, y_pred[, labels, ...])	Compute the F1 score, also known as balanced F-score or F-measure
metrics.fbeta_score(y_true, y_pred, beta[, ...])	Compute the F-beta score
metrics.hamming_loss(y_true, y_pred[, ...])	Compute the average Hamming loss.
metrics.hinge_loss(y_true, pred_decision[, ...])	Average hinge loss (non-regularized)
metrics.jaccard_score(y_true, y_pred[, ...])	Jaccard similarity coefficient score
metrics.log_loss(y_true, y_pred[, eps, ...])	Log loss, aka logistic loss or cross-entropy loss.
metrics.matthews_corrcoef(y_true, y_pred[, ...])	Compute the Matthews correlation coefficient (MCC)
metrics.multilabel_confusion_matrix(y_true, ...)	Compute a confusion matrix for each class or sample
metrics.precision_recall_curve(y_true, ...)	Compute precision-recall pairs for different probability thresholds
metrics.precision_recall_fscore_support(...)	Compute precision, recall, F-measure and support for each class
metrics.precision_score(y_true, y_pred[, ...])	Compute the precision
metrics.recall_score(y_true, y_pred[, ...])	Compute the recall
metrics.roc_auc_score(y_true, y_score[, ...])	Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
metrics.roc_curve(y_true, y_score[, ...])	Compute Receiver operating characteristic (ROC)
metrics.zero_one_loss(y_true, y_pred[, ...])	Zero-one classification loss.

Regression metrics

See the Regression metrics section of the user guide for further details.

metrics.explained_variance_score(y_true, y_pred)	Explained variance regression score function
metrics.max_error(y_true, y_pred)	max_error metric calculates the maximum residual error.
metrics.mean_absolute_error(y_true, y_pred)	Mean absolute error regression loss
metrics.mean_squared_error(y_true, y_pred[, ...])	Mean squared error regression loss
metrics.mean_squared_log_error(y_true, y_pred)	Mean squared logarithmic error regression loss
metrics.median_absolute_error(y_true, y_pred)	Median absolute error regression loss
metrics.r2_score(y_true, y_pred[, ...])	R^2 (coefficient of determination) regression score function.

Multilabel ranking metrics

See the Multilabel ranking metrics section of the user guide for further details.

metrics.coverage_error(y_true, y_score[, ...])	Coverage error measure
metrics.label_ranking_average_precision_score(...)	Compute ranking-based average precision
metrics.label_ranking_loss(y_true, y_score)	Compute Ranking loss measure

Clustering metrics

See the Clustering performance evaluation section of the user guide for further details.

The sklearn.metrics.cluster submodule contains evaluation metrics for cluster analysis results. There are two forms of evaluation:

- supervised, which uses a ground truth class values for each sample.
- unsupervised, which does not and measures the 'quality' of the model itself.

metrics.adjusted_mutual_info_score(...[, ...])	Adjusted Mutual Information between two clusterings.
metrics.adjusted_rand_score(labels_true, ...)	Rand index adjusted for chance.
metrics.calinski_harabasz_score(X, labels)	Compute the Calinski and Harabasz score.
metrics.davies_bouldin_score(X, labels)	Computes the Davies-Bouldin score.
metrics.completeness_score(labels_true, ...)	Completeness metric of a cluster labeling given a ground truth.
metrics.cluster.contingency_matrix(...[, ...])	Build a contingency matrix describing the relationship between labels.
metrics.fowlkes_mallows_score(labels_true, ...)	Measure the similarity of two clusterings of a set of points.
metrics.homogeneity_completeness_v_measure(...)	Compute the homogeneity and completeness and V-Measure scores at once.
metrics.homogeneity_score(labels_true, ...)	Homogeneity metric of a cluster labeling given a ground truth.
metrics.mutual_info_score(labels_true, ...)	Mutual Information between two clusterings.
metrics.normalized_mutual_info_score(...[, ...])	Normalized Mutual Information between two clusterings.
metrics.silhouette_score(X, labels[, ...])	Compute the mean Silhouette Coefficient of all samples.
metrics.silhouette_samples(X, labels[, metric])	Compute the Silhouette Coefficient for each sample.
metrics.v_measure_score(labels_true, labels_pred)	V-measure cluster labeling given a ground truth.

지도학습 : 분류 – 로지스틱 회귀

피마 인디언 당뇨병 예측

- 데이타세트 다운 로드 : https://www.kaggle.com/uciml/pima-indians-diabetes-database
- 피마 인디언 당뇨병 데이타 세트 : 북아메리카 피마 지역 원주민의 Type-2 당뇨병 결과 데이타
- 20세기 후반 들어서 서구화된 식습관으로 많은 당뇨 환자가 발생
- 당뇨병 여부를 판단하는 머신러닝 예측 모델을 만들고, 분류 평가 모델의 평가 지표 사용

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score
from sklearn.metrics import f1_score, confusion_matrix, precision_recall_curve, roc_curve
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression


diabetes_data = pd.read_csv('diabetes.csv')
print(diabetes_data['Outcome'].value_counts())
diabetes_data.head(3)
```

```
0    500
1    268
Name: Outcome, dtype: int64
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1

- 전체 768개의 데이타 중에서 Negative 값 0이 500개, Positive 값 1이 268개
- Negative가 상대적으로 많음

지도학습 : 분류 – 로지스틱 회귀

피마 인디언 당뇨병 데이타 세트의 피처

- Pregnancies: 임신 횟수
- Glucose: 포도당 부하 검사 수치
- BloodPressure: 혈압(mm Hg)
- SkinThickness: 팔 삼두근 뒤쪽의 피하지방 측정값(mm)
- Insulin: 혈청 인슐린(mu U/ml)
- BMI: 체질량지수(체중(kg)/(키(m))^2)
- DiabetesPedigreeFunction: 당뇨 내력 가중치 값
- Age: 나이
- Outcome: 클래스 결정 값(0또는 1)

피처 타입과 Null 확인

```
1  diabetes_data.info( )
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
```

지도학습 : 분류 – 로지스틱 회귀

Logistic Regression으로 학습 및 예측 수행

- 오차행렬, 정밀도, 재현율, F1스코어, AUC 구하는 함수

```
def get_clf_eval(y_test, pred=None, pred_proba=None):
    confusion = confusion_matrix( y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred)
    recall = recall_score(y_test , pred)
    f1 = f1_score(y_test,pred)
    roc_auc = roc_auc_score(y_test, pred_proba)
    print('오차 행렬')
    print(confusion)
    # ROC-AUC print
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f},₩
          F1: {3:.4f}, AUC:{4:.4f}'.format(accuracy, precision, recall, f1, roc_auc))
```

지도학습 : 분류 – 로지스틱 회귀

- Logistic Regression으로 학습 및 예측 수행

```
# 피처 데이터 세트 X, 레이블 데이터 세트 y를 추출.
# 맨 끝이 결과 컬럼으로 레이블 값임. 컬럼 위치 -1을 이용해 추출
X = diabetes_data.iloc[:, :-1]
y = diabetes_data.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 156, stratify=y)

# 로지스틱 회귀로 학습,예측 및 평가 수행.
lr_clf = LogisticRegression()
lr_clf.fit(X_train , y_train)
pred = lr_clf.predict(X_test)
# roc_auc_score 추가
pred_proba = lr_clf.predict_proba(X_test)[:, 1]

get_clf_eval(y_test , pred, pred_proba)
```

```
오차 행렬
[[88 12]
 [23 31]]
정확도: 0.7727, 정밀도: 0.7209, 재현율: 0.5741,          F1: 0.6392, AUC:0.7919
```

지도학습 : 분류 – 로지스틱 회귀

ROC

```
def roc_curve_plot(y_test , pred_proba_c1):
    # 임곗값에 따른 FPR, TPR 값을 반환 받음.
    fprs , tprs , thresholds = roc_curve(y_test ,pred_proba_c1)

    # ROC Curve를 plot 곡선으로 그림.
    plt.plot(fprs , tprs, label='ROC')
    # 가운데 대각선 직선을 그림.
    plt.plot([0, 1], [0, 1], 'k--', label='Random')

    # FPR X 축의 Scale을 0.1 단위로 변경, X,Y 축명 설정등
    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1),2))
    plt.xlim(0,1); plt.ylim(0,1)
    plt.xlabel('FPR( 1 - Sensitivity )'); plt.ylabel('TPR( Recall )')
    plt.legend()
    plt.show()

roc_curve_plot(y_test, lr_clf.predict_proba(X_test)[:, 1] )
```

AUC

```
from sklearn.metrics import roc_auc_score

pred_proba = lr_clf.predict_proba(X_test)[:, 1]
roc_score = roc_auc_score(y_test, pred_proba)
print('ROC AUC 값: {0:.4f}'.format(roc_score))
```

```
ROC AUC 값: 0.7919
```