기계학습 - 231017

6.1 결정 트리 학습과 시각화

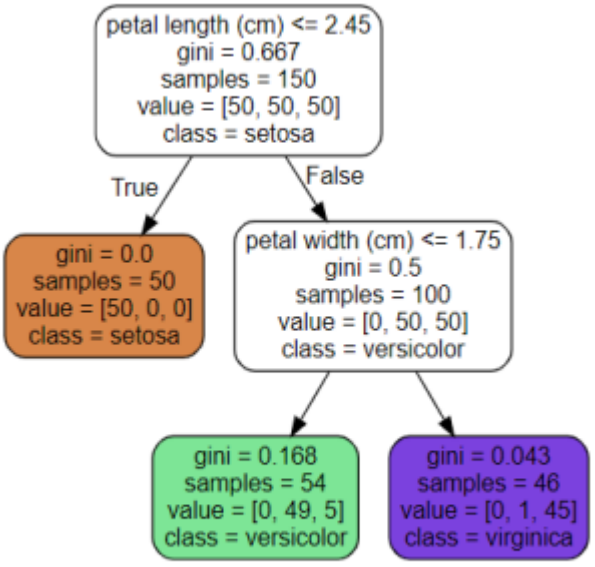
6.1 (코드)결정 트리 학습과 시각화

- DecisionTreeClassifier() :

→scikit-learn이 제공하는 의사결정트리 학습 API

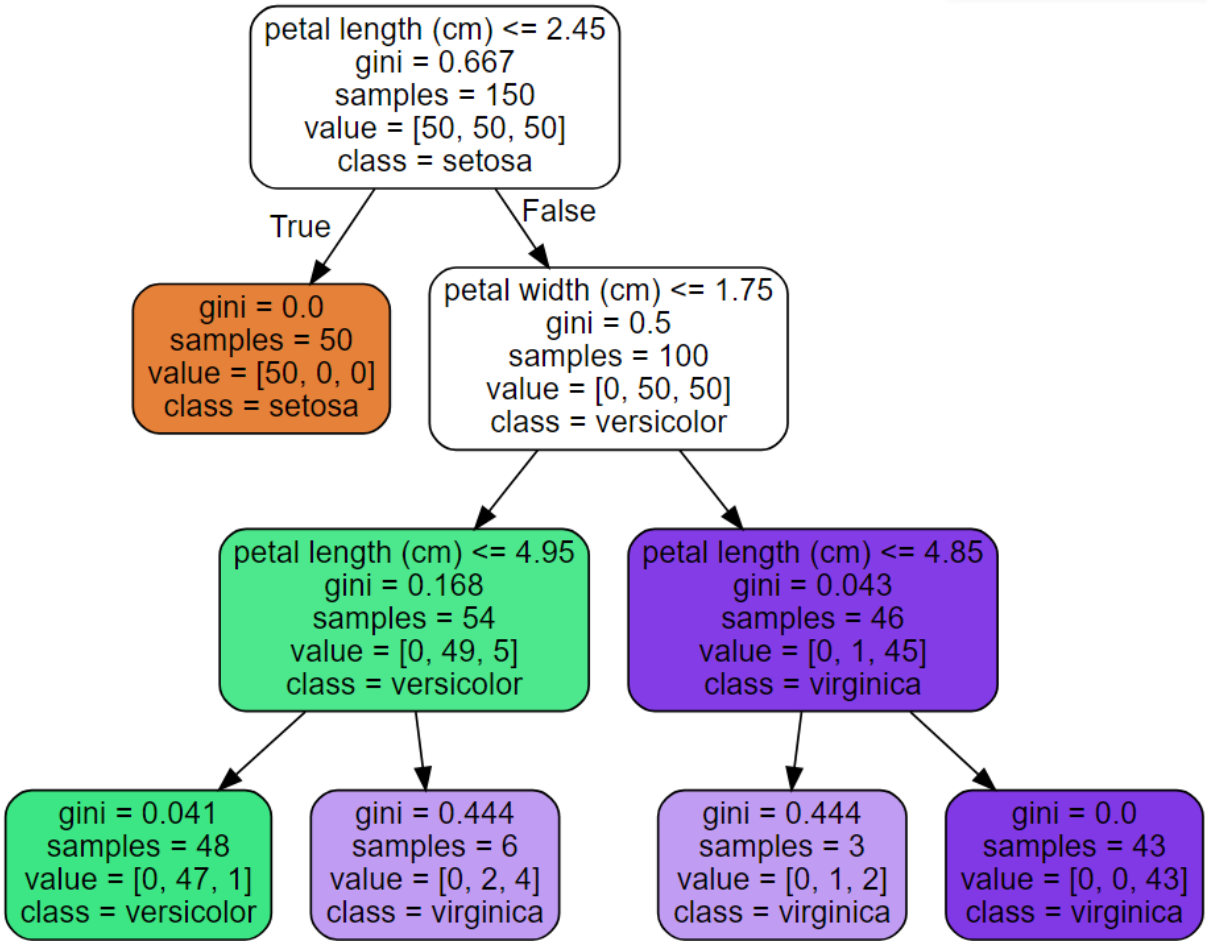
- export_graphviz()의 인자인 tree_clf는 우리가 학습시킨 데이터
- max_depth :
  → 연속된 가지치기 수

ex) max_depth = 2

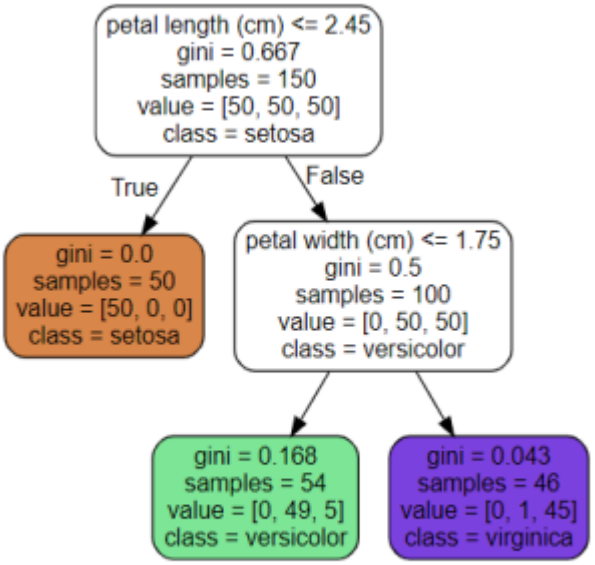


▲ 그림 6-1 붓꽃 결정 트리

ex) max_depth = 3



6.1 (결과해석)결정 트리 학습과 시각화



▲ 그림 6-1 붓꽃 결정 트리

gini : 노드들의 불순도

→ 동일클래스일시 0

→낮을수록 분류 잘됨

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^{\ 2}$$

▲ 식 6-1 지니 불순도

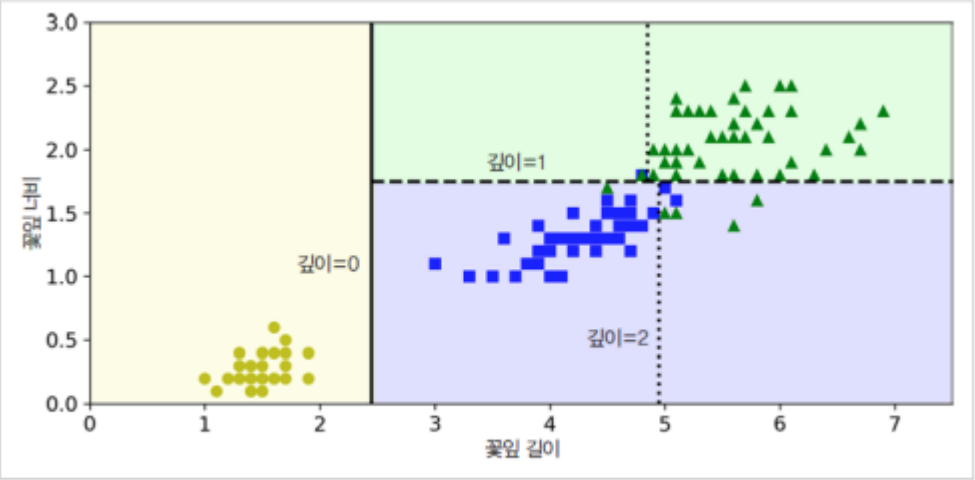
→ 계산법 :

샘플 54개, velue3개(setosa,Versicolor,virginica)

$$1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$$

6.2 예측하기

만들어 낸 예측



▲ 식 6-2 트리의 결정 경계

6.3 클래스 확률 추정

- 결정 트리는 한 샘플이 특정 클래스 k에 속할 확률을 추정 할수 있음.

→ 계산된 클래스별 비율을 이용하여 새로운 샘플에 대한 예측 실행

❓ 다음 과제)
뎁스가 길으질수록 제대로 분류하지 못하는 이유

!6.4 **CART 훈련 알고리즘**

- CART(Classification and Regression Tree) 알고리즘 = 탐욕적 알고리즘

→여러 경우 중 하나를 결정해야 할 때마다 그 순간에 최적이라고 생각되는 것을 선택해 나가는 방식.
→ 사이킷런 결정 트리 훈련시키기 위해CART 알고리즘을 사용

→비용함수를 최소화 하는 특성 k 와 해당 특성의 임곗값 tk 을 결정

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

여기에서 $\begin{cases} G_{\text{left/right}} \text{는 왼쪽/오른쪽 서브셋의 불순도} \\ m_{\text{left/right}} \text{는 왼쪽/오른쪽 서브셋의 샘플 수} \end{cases}$

▲ 식 6-2 분류에 대한 CART 비용 함수

!6.5

예측을 하려면 결정 트리를 루트 노드에서부터 리프 노드까지 탐색해야 함
▪ 일반적으로 결정 트리는 거의 균형을 이루고 있으므로 결정 트리를 탐색하기 위해서는 약 O(log2
(m)) 개의 노드를 거쳐야 함.
▪ 각 노드는 하나의 특성값만 확인하기 때문에 예측에 필요한 전체 복잡도는 특성 수와 무관하게 O(log2
(m)) 임.
▪ 큰 훈련 세트를 다룰 때도 예측 속도가 매우 빠름.

6.6 지니 불순도 또는 엔트로피?

불순도 종류 : gini, entropy …

엔트로피 :

→ 결과가 0일수록 잘 나뉨 (불순도 최소)

$$H_i = -\sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log_2(p_{i,k})$$

▲ 식 6-3 엔트로피

어떤걸 사용해야할까? : 엄청큰 차이는 없음

→ 지니가 쪼금 더빠름(기본값으로 사용됨),엔트로피가 조금더 균형잡힌 트리

6.7 규제 매개변수

비 매개변수 : 훈련전 파라미터수 결정X

→ 과대적합 위험

매개변수 : 미리 정의된 모델 파라미터 존재

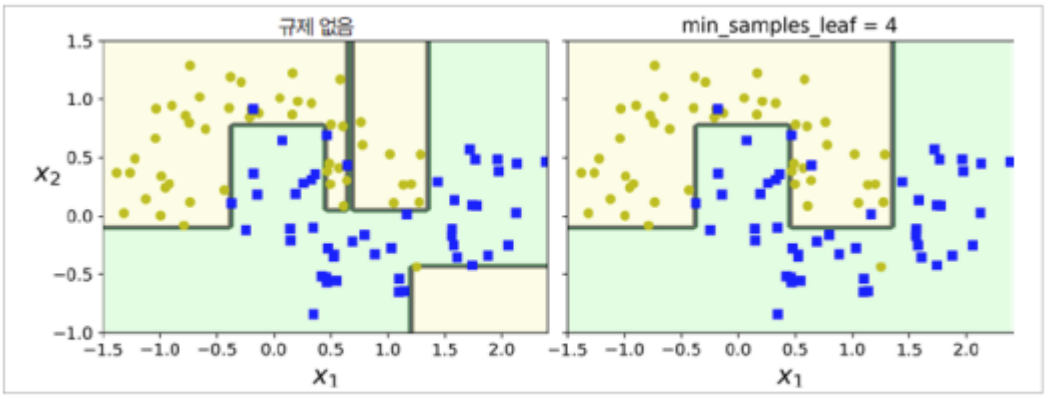
→자유도 제한, 과대적합위험도 줄음, 과소적합위험 커짐

6.7.(DecisionTreeClassifier) 규제 매개변수

→ 이러한 규제변수로 일반화시키는데 사용

규제를 높이는 방법
→ min_ 접두사 사용 규제: 매개변수를 증가시킴.
→ max_ 접두사 사용 규제: 매개변수를 감소시킴.

6.7.(코드) 규제 매개변수



▲ 그림 6-3 min_samples_leaf 매개변수를 사용한 규제

왼쪽: 규제 전혀 없음
→보다 정교함, 과대적합됨.
오른쪽: min_samples_leaf=4
→일반화 성능 보다 좋음

!6.8 회귀

DecisionTreeRegressor() :

→

ex) x1 = 0.6인 샘플 타깃을 예측 → 결국 value = 0.111인 리프 노드에 도달.

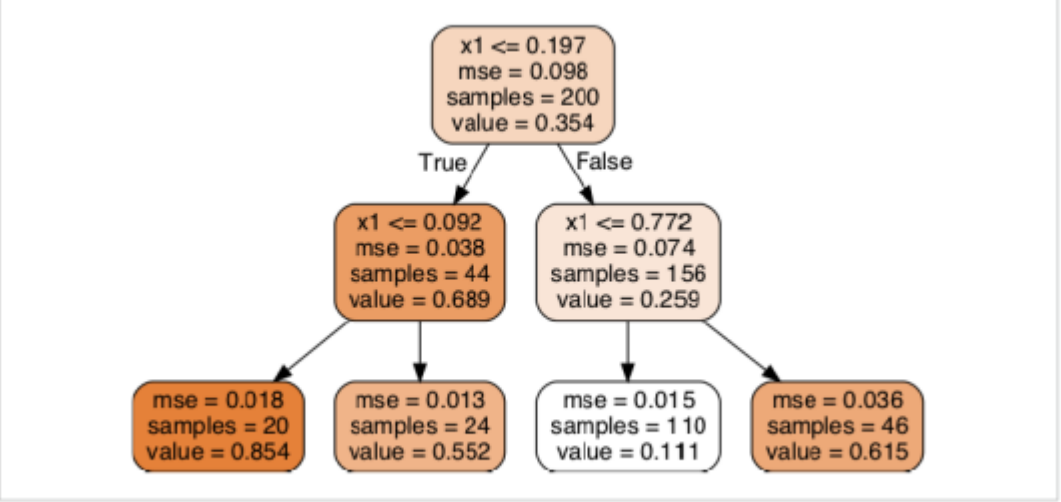


그림 6-4 회귀 결정 트리

→ samples: 해당 마디에 속한 훈련 샘플 수
→ value: 해당 마디에 속한 훈련 샘플의 평균 타깃값
→ mse: 해당 마디에 속한 훈련 샘플의 평균제곱오차

6.8.(결정경계) 회귀

왼편: max_depth=2
오른편: max_depth=3

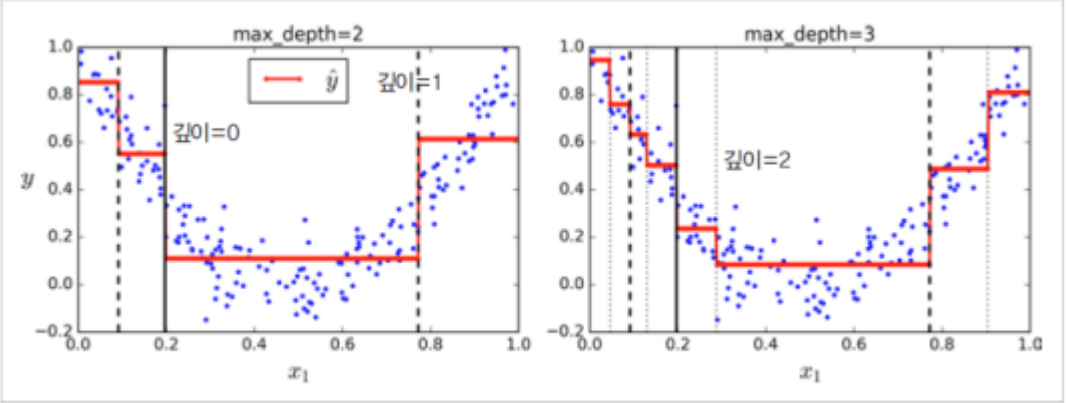


그림 6-5 두 개의 결정 트리 회귀 모델의 예측

왼쪽 그래프 볼 때 하단 그래프를 참고 (각 네모의 맨 윗줄,맨아래줄)

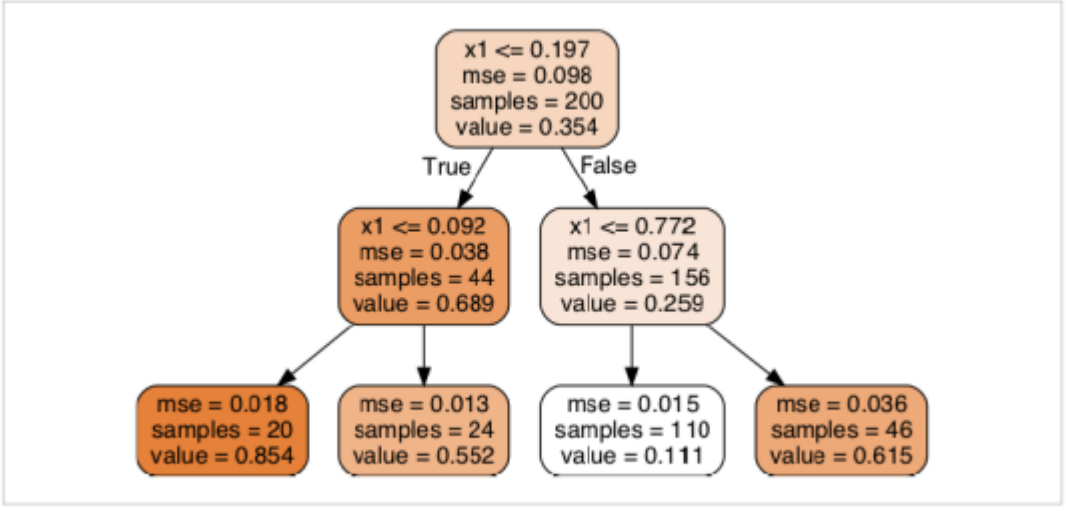


그림 6-4 회귀 결정 트리

6.8.(비용함수) 회귀

- CART 알고리즘
→훈련 세트를 평균제곱오차(MSE)를 최소화하도록 분할
→아래 비용함수를 최소화 하는 특성 k와 해당 특성의 임곗값 tk을 결정
→각 노드의 평균제곱오차 MSE를 최소화하는 방향으로 학습

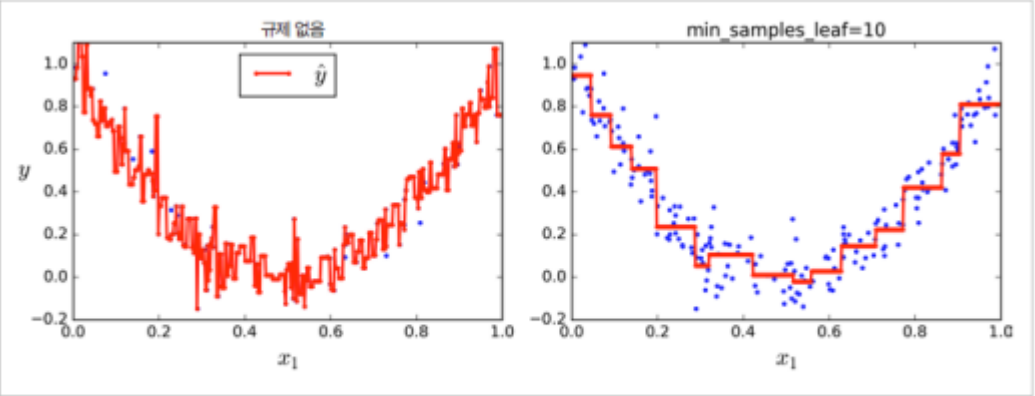
$$J(k, t_k) = \frac{m_{\text{left}}}{m} \text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m} \text{MSE}_{\text{right}}$$

$$\text{MSE}_{\text{node}} = \sum_{i \in \text{node}} (\hat{y}_{node} - y^{(i)})^2$$

$$\hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)}$$

6.8. 결정 트리 회귀 모델의 규제

-왼편: 규제가 없는 경우
→ 과대적합 발생
-오른편: min_samples_leaf=10
→일반화됨

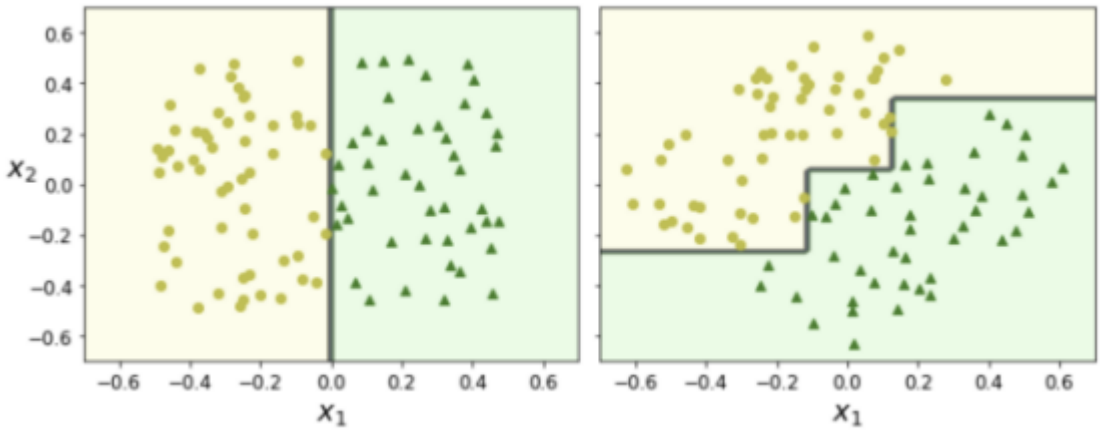


▲ 그림 6-6 결정 트리 회귀 모델의 규제

6.9 불안정성 (회전에 민감)

- 결정 트리: 여러 용도로 사용가능하고, 성능이 매우 우수

→ 하지만 훈련 세트에 민감
→ 계단모양 결정경계로 인해 회전에 민감



문제 해결

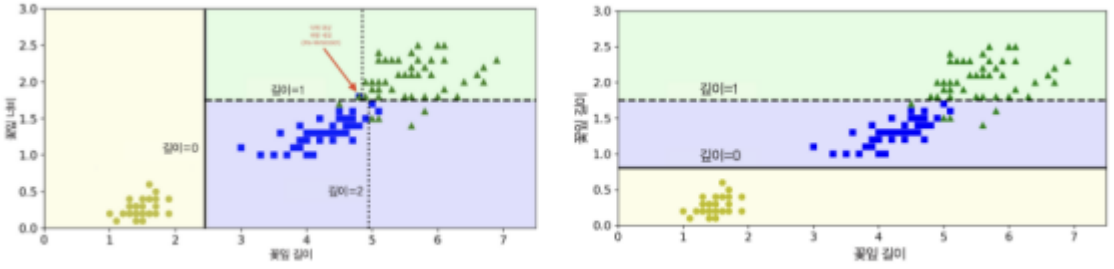
→훈련 데이터를 더 좋은 방향으로 회전시키는 PCA 기법 사용

6.9. 불안정성 (작은변화에 민감)

- 결정 트리: 여러 용도로 사용가능하고, 성능이 매우 우수

→하지만 훈련 세트의 작은 변화에 매우 민감함

ex) 화살표 값 제거하니 많이 바뀌어버림



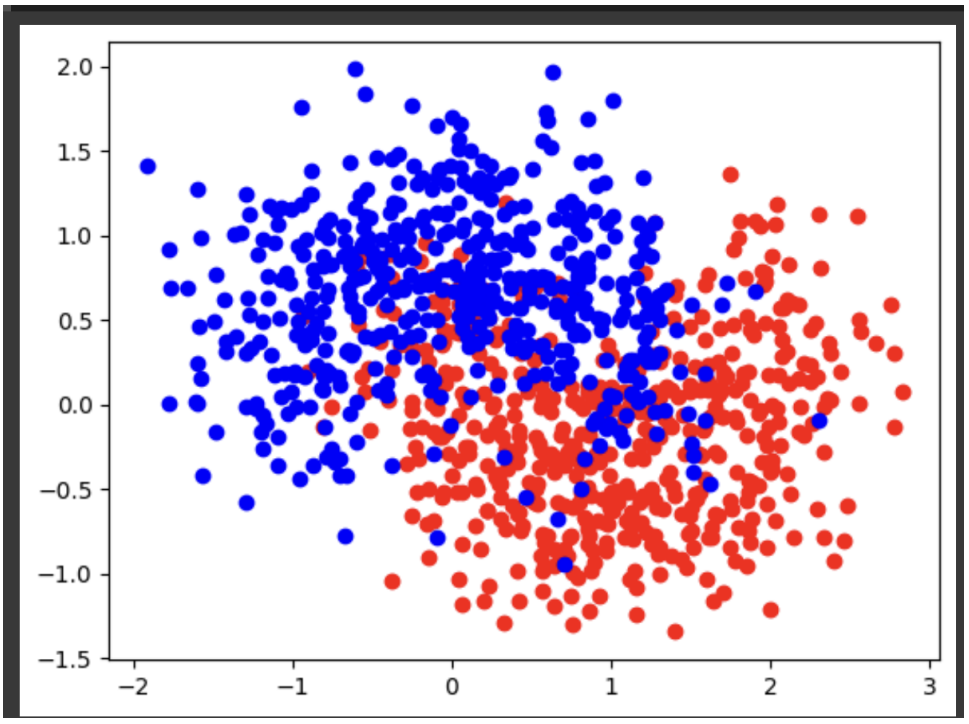
연습문제

a) make_moons(n_sample=1000, noise=0.4)를 사용해 데이터 셋을 나눈다

```
# 가상데이터 생성(sklearn.make_moons 이용, 샘플과 노이즈 설정)
x, y = make_moons(n_samples=1000, noise=0.4)

#가상데이터 시각화
```

```
plt.plot(x[y==1,0],x[y==1,1],marker='o',linestyle='',color='red')
plt.plot(x[y==0,0],x[y==0,1],marker='o',linestyle='',color='blue')
plt.show()
```



b) 이를 train_test_split()을 사용해 훈련 세트와 테스트 세트로 나눈다

```
#훈련세트와 테스트세트를 분리하는 문제를 sklearn.train_test_split을 이용하여 쉽게 가능
#random_state를 사용한 난수생성
#stratify를 통한 y클래스 유지
#테스트셋은 전체 데이터의 25%로 설정
x_train, x_test, y_train, y_test = train_test_split(
x, y, stratify=y, random_state=42, test_size=0.25
)
```

c) DecisionTreeClassifier의 최적의 매개변수를 찾기 위해 교차 검증과 함께 그리드 탐색을 수행한다

(GridSearchCV 를 사용하면 됨. 여러가지 max_leaf_nodes 값을 시도)

```
#sklearn.GridSearchCV를 이용한 그리드 탐색
#max_leaf_nodes 값을 조절하여 리프노드의 최대 수 탐색

params = {'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2, 3, 4]}
gs = GridSearchCV(DecisionTreeClassifier(random_state=42), params, n_jobs=-1)
```

```
print('최적 파라미터 값 : ', gs.best_params_)
print('점수 : ', gs.best_score_)
```

d) 찾은 매개변수를 사용해 전체 훈련 세트에 대해 모델을 훈련시키고 테스트 세트에서 성능을 측정한다. 대략 85~87의 정확도가 나옴

```
gs.fit(x_train, y_train)
gs.score(x_test, y_test)
```

```
#그리드탐색
params = {'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2, 3, 4]}
gs = GridSearchCV(DecisionTreeClassifier(random_state=42), params, n_jobs=-1)

gs.fit(x_train, y_train)
gs.score(x_test, y_test)


print('최적 파라미터 값 : ', gs.best_params_)
print('점수 : ', gs.best_score_)
```

```
최적 파라미터 값 :  {'max_leaf_nodes': 4, 'min_samples_split': 2}
점수 :  0.8506666666666668
```

sklearn의 결정트리와 그리드 탐색을 통한 교차검증 결과비교

(기본)

```
dt = DecisionTreeClassifier(random_state=42)
dt.fit(x_train, y_train)
dt.score(x_test, y_test)
#결과 = 0.82
```

(그리드 탐색)

```
params = {'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2, 3, 4]}
gs = GridSearchCV(DecisionTreeClassifier(random_state=42), params, n_jobs=-1)

gs.fit(x_train, y_train)
gs.score(x_test, y_test)

#결과 = 0.85
```

결론:

그리드 탐색을 통해 최적의 파라미터를 찾았을 때, 그냥 모델을 사용했을 때에 비해 0.03 높은 Score를 보여줌

#하고있는중

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

Xm, ym = make_moons(n_sample=1000, random_state=3, test_size=0.25)

Xm_train, Xm_test, ym_train, ym_test = train_test_split(Xm,ym)

dtree = DecisionTreeClassifier()

parameters = {'max_leaf_nodes':[1, 2, 3]}

grid_model = GridSearchCV(dtree,
                          param_grid=parameters,
                          cv=3,
                          refit=True,
                          return_train_score=True)

grid_model.fit(Xm_train,ym_train)

print('final params', gcv.best_params_)   # 최적의 파라미터 값 출력
print('best score', gcv.best_score_)      # 최고의 점수
```