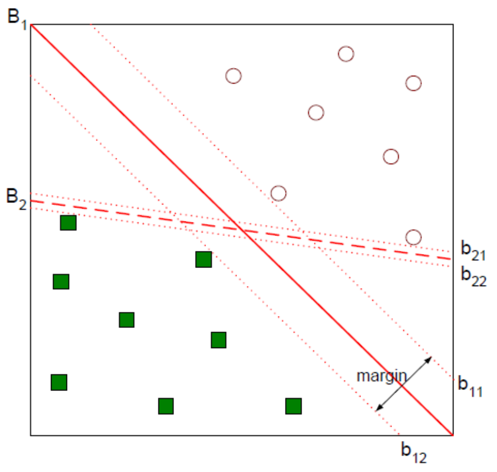
08. SVM과 결정트리

- SVM(Support Vector Machine)
- 결정트리(Decision Tree)

SVM(Support Vector Machine)

SVM이란?

- 다목적 기계학습 모델
  - 선형, 비선형 분류(linear, non-linear classification), 회귀(regression), 이상치(Outlier) 탐색 등 복잡한 분류문제에 적절함
  - 중간이하 크기의 데이터셋에 적합



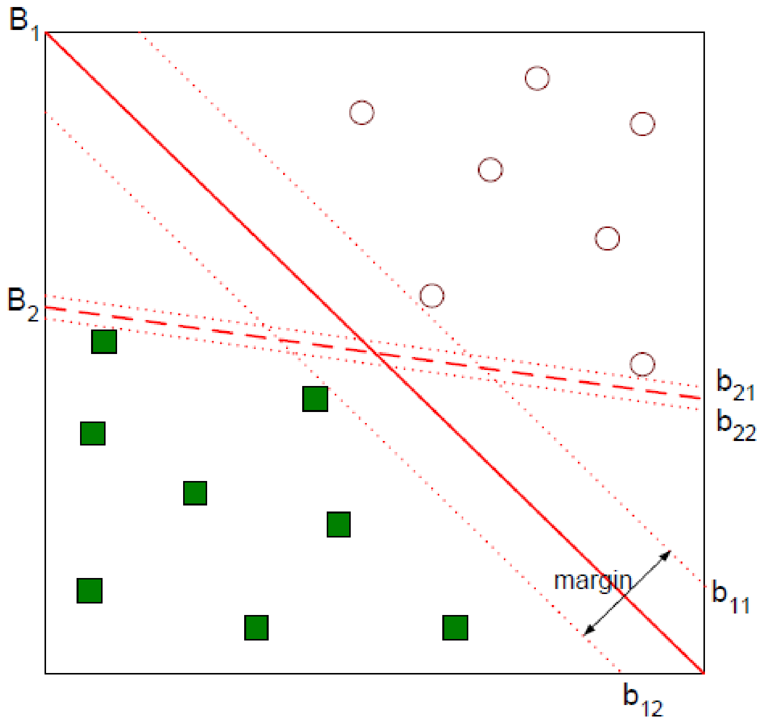
- 작동원리
  - 분류를 위한 최적의 결정경계(decision boundary) 즉, 다수의 결정경계 후보들 중에서 **최대의 마진을 갖는 결정경계**를 찾음
    - 마진(margin) : minus-plane과 plus-plane 사이의 거리
    - 서포트벡터(support vectors) : 마진 결정에 영향을 끼치는 샘플데이터

결정경계(decision boundary)

- 샘플들의 카테고리를 구분할 수 있는 초평면(hyperplane)
- N차원 공간상의 결정경계 차원 : N-1
  - 원점에서 시작하는 벡터 w와 직교하고, 거리가 b인 직선의 방정식($w^T x + b = 0$)

마진(Margin)

- 결정경계와 서포트벡터 사이의 거리
  - b_{11} : plus-plane ($w^T x + b = 1$)
  - b_{12} : minus-plane ($w^T x + b = -1$)
- 양의샘플(x^+)과 음의샘플(x^-)의 관계
  - $x^+ = x^- + \lambda w$
    - x^-를 w방향으로 λ만큼 평행이동



참조: https://imgur.com/DrcoGVQ

마진(Margin)

- λ의 값은?

$$\lambda = \frac{2}{w^T w}$$

$$w^T x^+ + b = 1$$
$$w^T(x^- + \lambda w) + b = 1$$
$$w^T x^- + b + \lambda w^T w = 1$$
$$-1 + \lambda w^T w = 1$$

- 따라서 margin $= distance(x^+, x^-)$

$$= \| x^+ - x^- \|_2$$
$$= \frac{2}{\| w \|_2}$$

- SVM의 목적은 마진을 최대화하는 결정경계를 찾는 것
  - 계산의 편의상 다음과 같이 최적화 문제로 변환

$$max \frac{2}{\| w \|^2} \Rightarrow \min \frac{1}{2} \| w \|_2^2$$

참고: SVM 클래스(numpy로 직접 구현한 것)

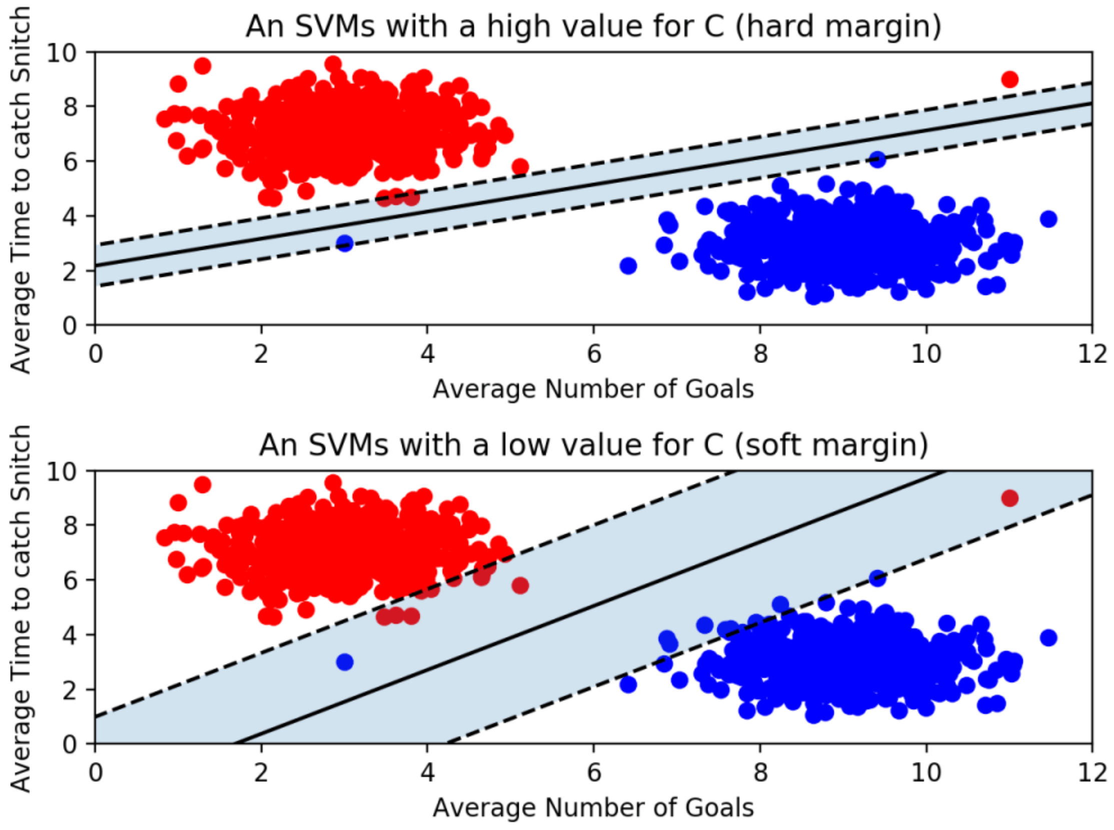
```
class SVM:
    def __init__(self, X, y, epochs, lr, C):
        self.X = X;     self.y = y;     self.epochs = epochs
        self.lr = lr;   self.C = C
        # Add column vector of ones for computational convenience
        self.X = np.column_stack((np.ones(len(X)), X))
        # Initialize normal vector
        self.w = np.ones(len(self.X[0]))

    def distances(self, w, with_lagrange=True):
        distances = self.y * (np.dot(self.X, w)) - 1
        # get distance from the current decision boundary
        # by considering 1 width of margin
        if with_lagrange:  # if lagrange multiplier considered
            # if distance is more than 0, sample is not on the support vector
            # Lagrange multiplier will be 0
            distances[distances > 0] = 0
        return distances
```

참조: https://bitbucket.org/tarlanahad/myneatcodes/src/master/SVM/Python/

```python
    def get_cost_grads(self, X, w, y):
        distances = self.distances(w)
        # Get current cost
        L = 1 / 2 * np.dot(w, w) - self.C * np.sum(distances)
        dw = np.zeros(len(w))
        for ind, d in enumerate(distances):
            if d == 0:  # if sample is not on the support vector
                di = w  # (alpha * y[ind] * X[ind]) = 0
            else:
                # (alpha * y[ind] * X[ind]) = y[ind] * X[ind]
                di = w - (self.C * y[ind] * X[ind])
            dw += di
        return L, dw / len(X)

    def fit(self):
        for i in range(self.epochs):
            L, dw = self.get_cost_grads(self.X, self.w, self.y)
            self.w = self.w - self.lr * dw

    def predict(self, X):
        X = np.column_stack((np.ones(len(X)), X))
        return np.sign(X @ self.w) # X @ self.w => X.dot(self.w)
```

Soft margin Classification

- Soft margin
  - 이상치(outlier)가 있는 경우에 민감한 하드 마진 분류의 문제점을 피하기 위한 것
  - 파라메터 C
    - SVM이 오류를 어느정도 허용?
    - 작을수록 많은 오류를 허용
    - overfitting되는 경우에 작은 값을 지정하여 규제
  - 파라메터 gamma
    - 결정경계를 얼마나 유연하게?
    - 너무 크면 overfitting



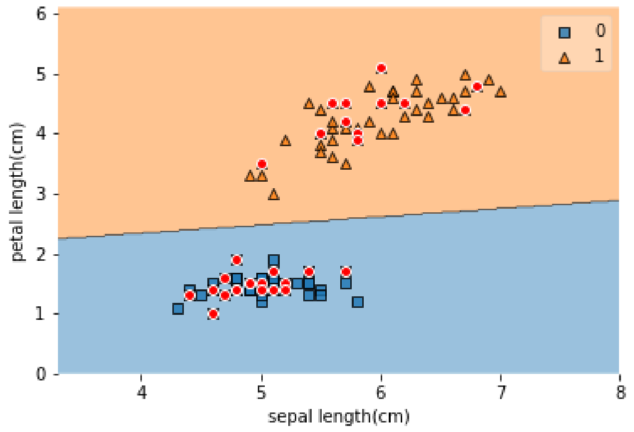
참조: https://hleecaster.com/ml-svm-concept/

sklearn.svm.**SVC**

```
SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale’,
           coef0=0.0, shrinking=True, probability=False,
           tol=0.001, cache_size=200, class_weight=None,
           verbose=False, max_iter=-1,
           decision_function_shape='ovr’,
           break_ties=False, random_state=None)
```

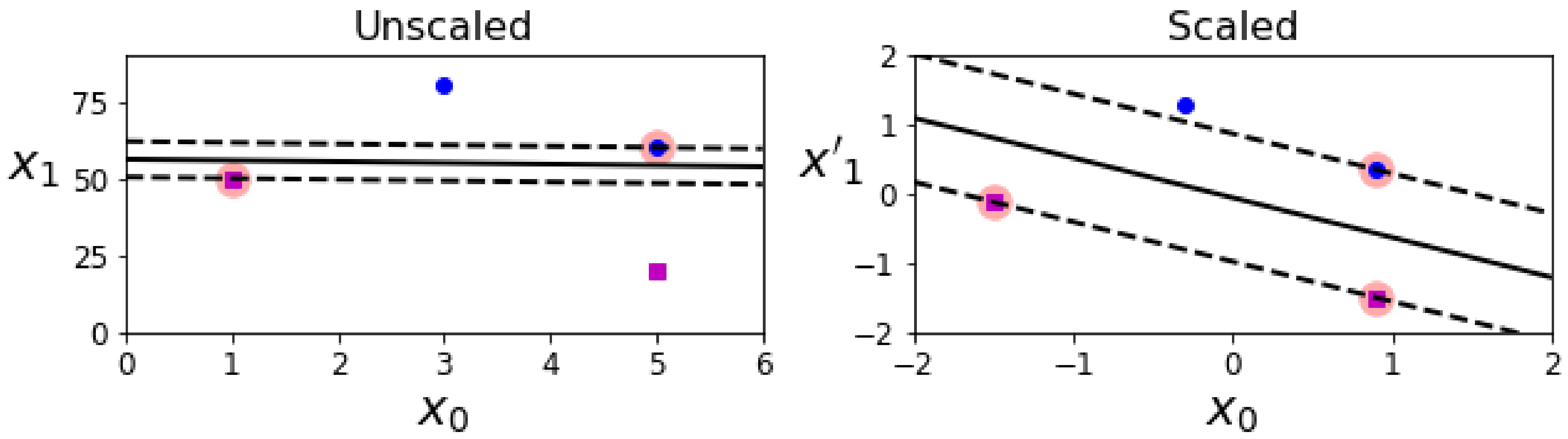
- **kernel: 커널 유형 지정**
  - {‘linear’, ‘poly’, **‘rbf’**, ‘sigmoid’, ‘precomputed’}
- **C: 오류허용 정도의 역수**
  - 작을수록 오류를 많이 허용(이상치에 덜 민감함)
- **gamma: Kernel 계수**
  - for ‘rbf’, ‘poly’, ‘sigmoid’

```python
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt

iris = load_iris()
X = iris["data"][0:100, (0,2)] # 처음 100개의 데이터 중에서 꽃받침 길이, 꽃잎 길이
y = iris["target"][0:100] # 처음 50개는 Iris-setosa, 다음 50개는 Iris-versicolor
y = np.where(y==0, 0, 1) # 만일 Iris-setosa이면 0, 아니면 1로 변경

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

clf = SVC(kernel='linear', C=0.5)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(f"accuracy: {accuracy_score(y_test, y_pred)}")

plot_decision_regions(X, y, clf=clf)
plt.scatter(X_test[:,0], X_test[:,1], marker="o", color="r", edgecolor="w")
plt.xlabel('sepal length(cm)')
plt.ylabel('petal length(cm)')
plt.show()
```



SVM은 Scale에 민감하다...

- 스케일이 많이 다른 경우의 예

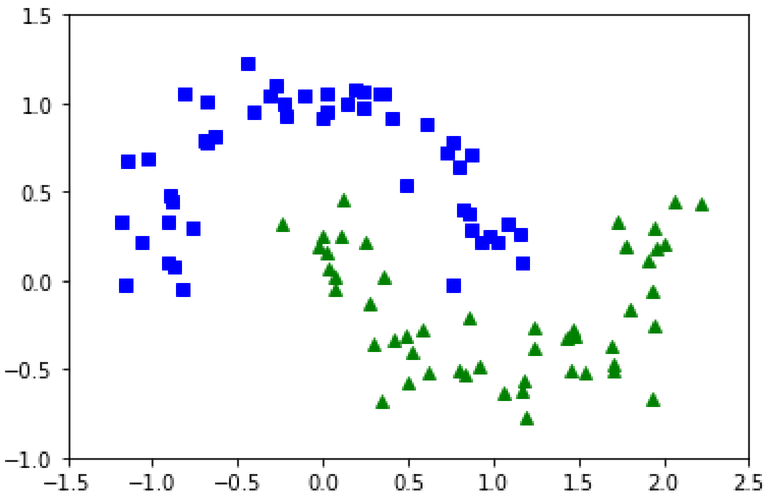


- 스케일러를 사용하면 결정경계를 개선할 수 있음
  - StandatdScaler
  - MinMaxScaler

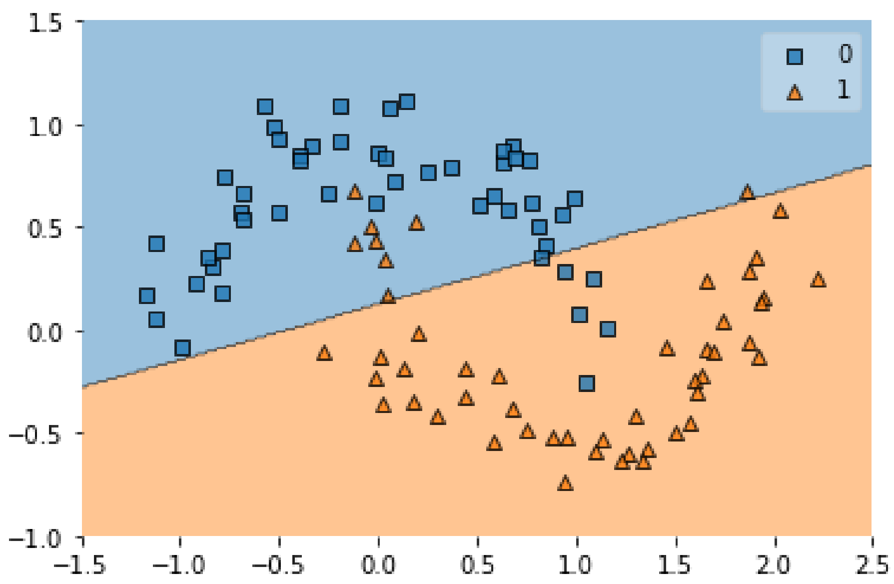
비선형 SVM

- 선형적으로 분류할 수 없는 경우
- (예)

```
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt

X, y = make_moons(n_samples=100, noise=0.15)

plt.plot(X[:,0][y==0], X[:,1][y==0], "bs")
plt.plot(X[:,0][y==1], X[:,1][y==1], "g^")
plt.axis([-1.5, 2.5, -1.0, 1.5])
plt.show()
```



```python
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt

X, y = make_moons(n_samples=100, noise=0.15)
svm_clf = make_pipeline(
    StandardScaler(),
    SVC(kernel='linear', C=10))

svm_clf.fit(X, y)

plot_decision_regions(X, y, clf=svm_clf)
plt.axis([-1.5, 2.5, -1.0, 1.5])
plt.show()
```



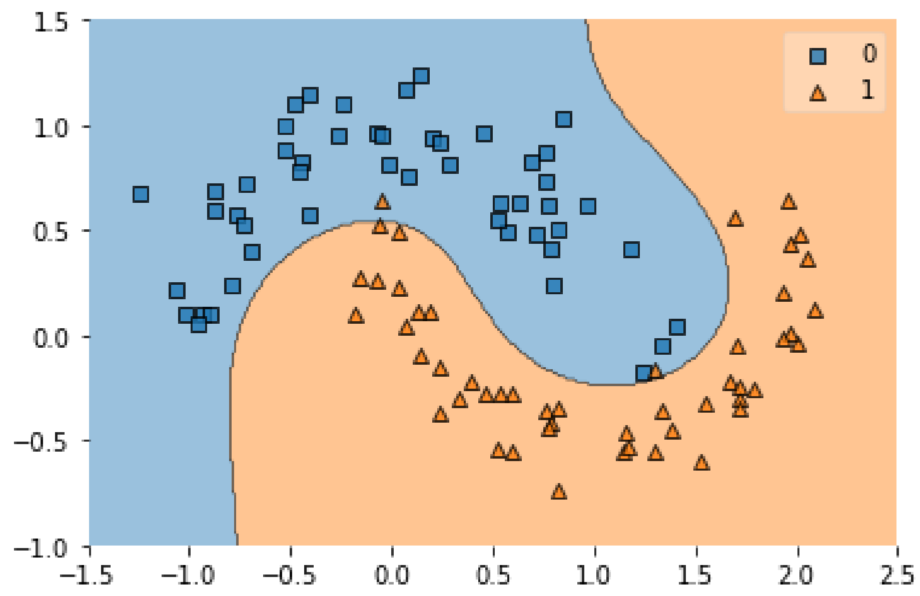
sklearn.preprocessing.**PolynomialFeatures**

- **다항회귀(polynomial regression)**
  - 비선형 데이터를 학습 가능
  - 각 특성의 거듭제곱을 새로운 특성으로 추가하고, 이 데이터셋에 대해 선형모델을 학습시키는 방법

- **PolynomialFeatures**
  - 각 특성을 주어진 degree에 따라 제곱하여 새로운 특성으로 추가

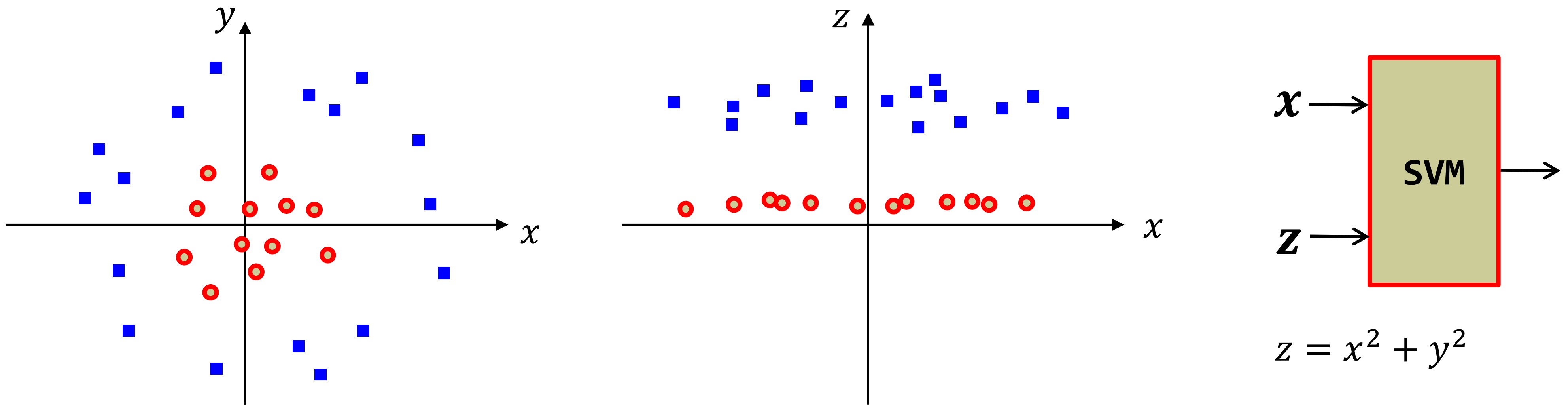
```
from sklearn.preprocessing import PolynomialFeatures

feature_poly = PolynomialFeatures(degree=2)
X_poly = feature_poly.fit_transform(X)
```

```python
from sklearn.datasets import make_moons
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt

X, y = make_moons(n_samples=100, noise=0.15)
svm_clf = make_pipeline(
    PolynomialFeatures(degree=3),
    StandardScaler(),
    SVC(kernel='linear', C=10))

svm_clf.fit(X, y)

plot_decision_regions(X, y, clf=svm_clf)
plt.axis([-1.5, 2.5, -1.0, 1.5])
plt.show()
```



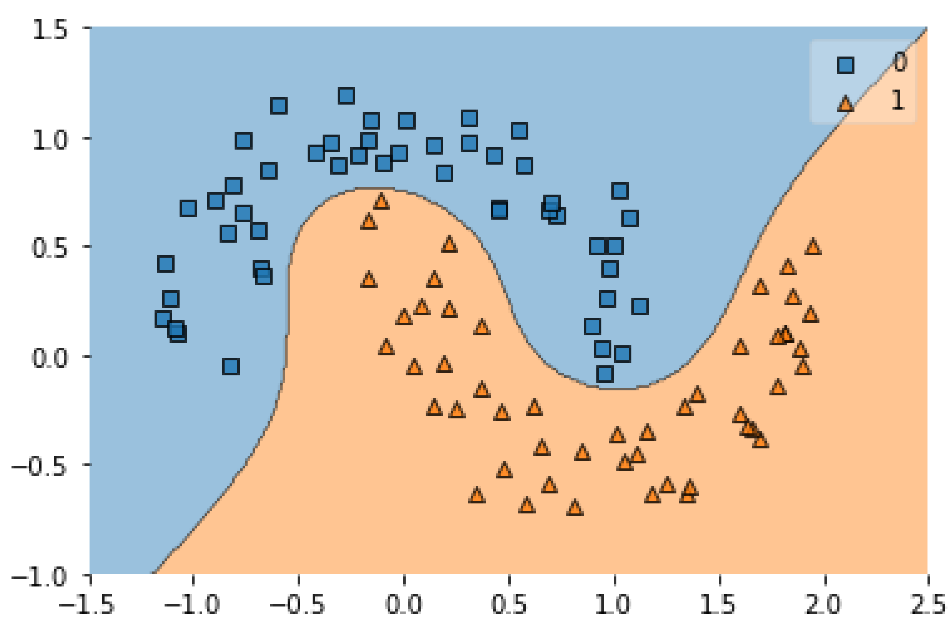
다항식 커널(kernel=‘poly’)

- 다항식 특성의 추가
  - 만일, PolynomialFeatures와 같은 것을 이용하여 높은 차수의 다항식 특성을 추가하는 경우에는 모델이 느려질 수 있음
- 커널트릭(Kernel Trick)
  - 높은 차수의 특성을 추가한 것과 같은 효과를 가지면서도 실제로 특성을 추가하지 않음



$z = x^2 + y^2$

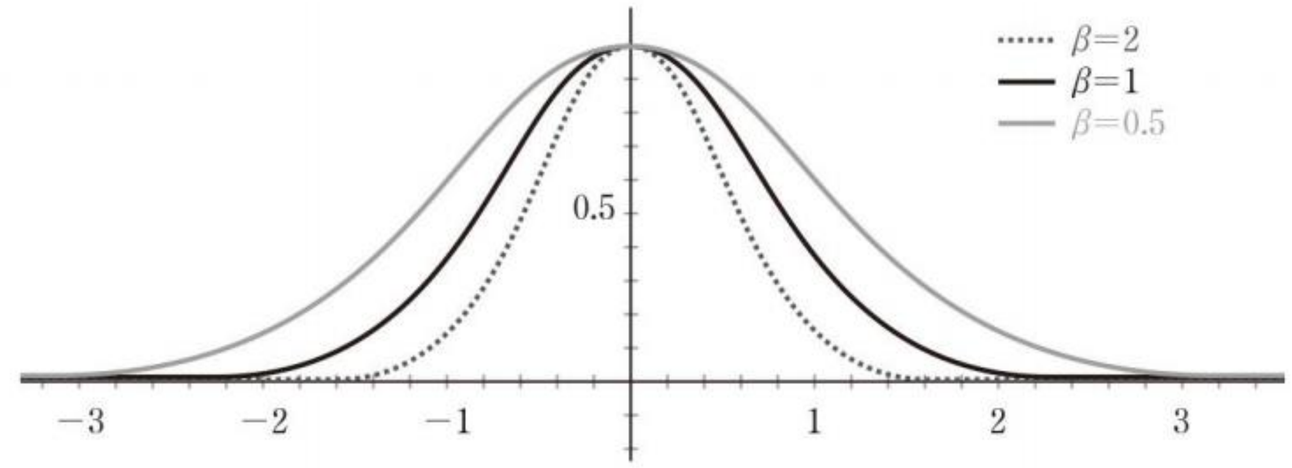
```python
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt

X, y = make_moons(n_samples=100, noise=0.15)
svm_clf = make_pipeline(
    StandardScaler(),
    SVC(kernel='poly', degree=3, coef0=1, C=5))

svm_clf.fit(X, y)

plot_decision_regions(X, y, clf=svm_clf)
plt.axis([-1.5, 2.5, -1.0, 1.5])
plt.show()
```



가우시안 RBF 커널(kernel='rbf')

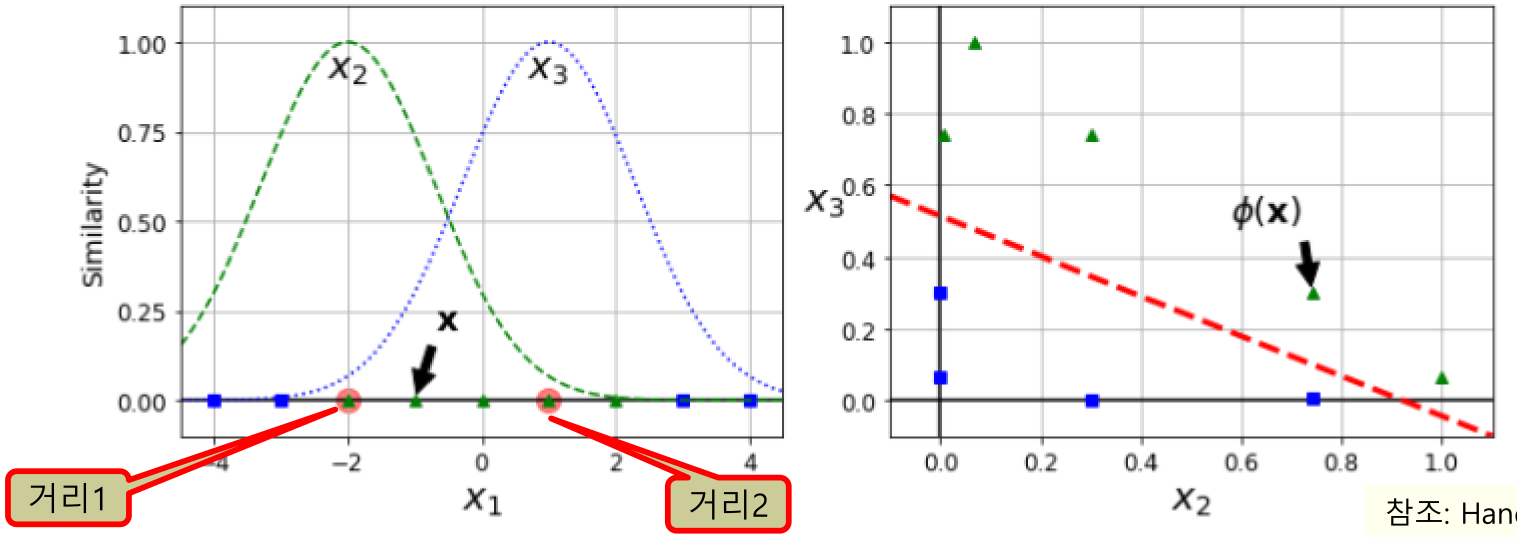
- RBF(Radial Basis Fuction)
  - 기존 벡터 μ와 입력벡터 x의 유사도(similarity) 측정 함수
  - 가우시안 함수

$$\phi(x,\mu) = \exp(-\beta \parallel x - \mu \parallel^2)$$



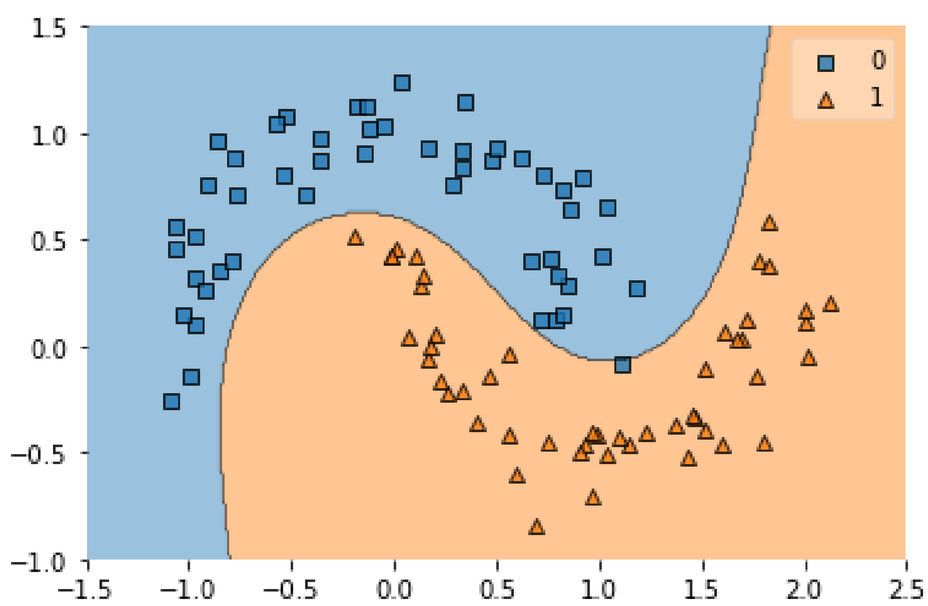
RBF 커널

- 각 샘플이 특정 randmark와 얼마나 유사한지 계산한 값을 추가
- (예) 1차원인 경우,
  - 두 개의 randmark 추가($x_1 = -2$, $x_1 = 1$), 실제로는 모든 샘플위치
  - $x_1 = -1$에 대해 각각의 randmark 사이의 유사도(RBF)계산($\beta = 0.3$인 경우)
  - $x_2 = \exp(-0.3 * (1)^2) = 0.74$, $x_3 = \exp(-0.3 * (2)^2) = 0.3$
  - 따라서 샘플 $\phi(x) = (0.74, 0.3)$ 추가



참조: Hands-On Machine Learning(박해선)

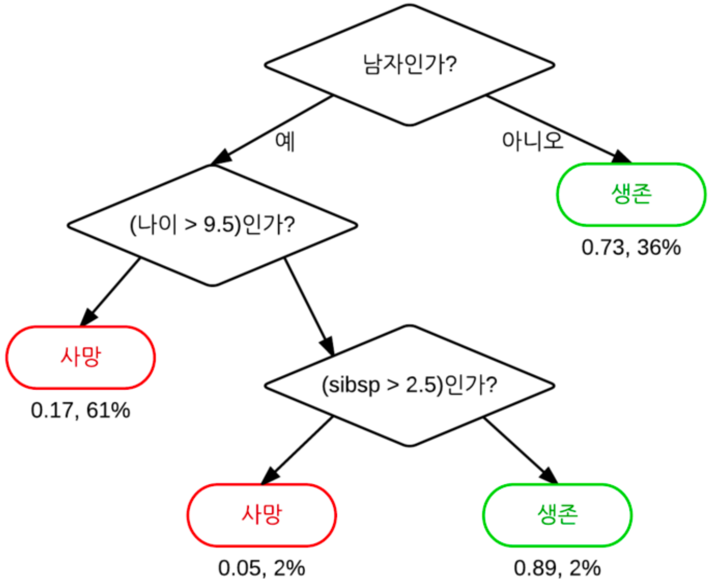
```python
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from mlxtend.plotting import plot_decision_regions
import matplotlib.pyplot as plt

X, y = make_moons(n_samples=100, noise=0.15)
svm_clf = make_pipeline(
    StandardScaler(),
    SVC(kernel='rbf', gamma=0.1, C=100))

svm_clf.fit(X, y)

plot_decision_regions(X, y, clf=svm_clf)
plt.axis([-1.5, 2.5, -1.0, 1.5])
plt.show()
```



결정트리(Decision Tree)

결정트리(Decision Tree)란?

- 일련의 분류 규칙을 통해 데이터를 분류, 회귀하는 지도 학습 모델 중 하나
- 작동방식
  - Root Node로 부터 시작하여 분류기준(조건)에 따라 전체 영역을 두 개로 구분하며, 맨 마지막 노드를 Terminal Node 혹은 Leaf Node라고 함
- 기본 아이디어
  - Leaf Node가 섞이지 않은 상태로
    즉, 복잡성(entropy)이
    낮아지도록 만드는 것
  - 복잡성이 낮아지는 방향으로 분기



참조: https://wooono.tistory.com/104

- 불순도(impurity)
  - 해당 카테고리(범주)안에 서로 다른 데이터가 얼마나 섞여 있는지를 나타내는 것
- 불순도를 수치적으로 나타내기 위한 함수
  - Gini

$$G = 1 - \sum_{k=1}^{m} p_k^2$$

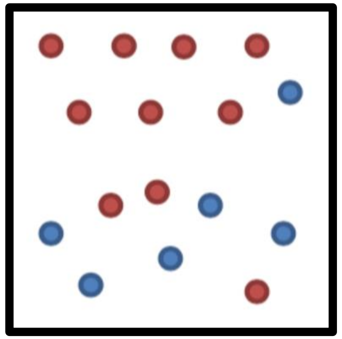
Entropy

$$E = -\sum_{k=1}^{k} p_i \log_2(p_i)$$

  - (예)카테고리 안에 빨간색 공이 10개, 파란색 공이 6개 있을 때

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

$$E = -\sum_{k=1}^{k} p_i \log_2(p_i) = -\frac{6}{16}\log_2\left(\frac{6}{16}\right) - \frac{10}{16}\log_2\left(\frac{10}{16}\right) = 0.95$$



결정트리 구성단계

1. Root Node의 불순도 계산
2. 나머지 특징들에 대해 분할한 후의 Child Node들의 불순도 계산
3. 각 특징들에 대한 정보획득(Information Gain)이 최대가 되는 조건을 찾아 분할(처음에는 Root Node와 Child Node의 불순도 차이가 최대가 되는 특징으로 먼저 분류)
4. 모든 Leaf Node들의 불순도가 0이 될 때까지 2~3단계 반복

정보획득(Information Gain)

- Parent Node와 Child Node들의 불순도 차이

결정트리 구성의 예

■ 테니스 경기 참가 여부

참조: https://wooono.tistory.com/104

(1) Root Node의 불순도

$$E(\text{경기}) = -\sum_{k=1}^{k} p_i \log_2(p_i)$$

$$= -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right)$$

$$= 0.94$$

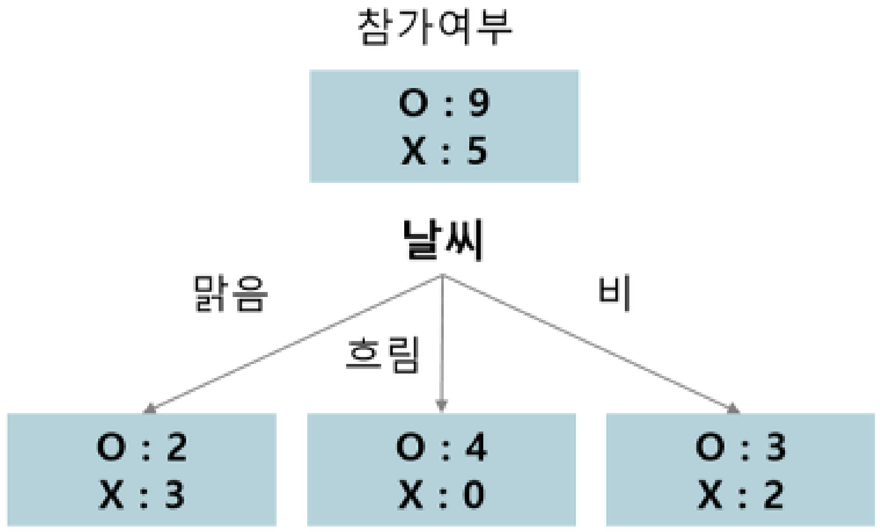
날짜	날씨	온도	습도	바람	참가여부
D1	맑음	더움	높음	약함	X
D2	맑음	더움	높음	강함	X
D3	흐림	더움	높음	약함	O
D4	비	포근	높음	약함	O
D5	비	서늘	정상	약함	O
D6	비	서늘	정상	강함	X
D7	흐림	서늘	정상	강함	O
D8	맑음	포근	높음	약함	X
D9	맑음	서늘	정상	약함	O
D10	비	포근	정상	약함	O
D11	맑음	포근	정상	강함	O
D12	흐림	포근	높음	강함	O
D13	흐림	더움	정상	약함	O
D14	비	포근	높음	강함	X

(2) 나머지 특징들에 대해 분할한 후의 Child Node들의 불순도 계산

날짜	날씨	온도	습도	바람	참가여부
D1	맑음	더움	높음	약함	X
D2	맑음	더움	높음	강함	X
D3	흐림	더움	높음	약함	O
D4	비	포근	높음	약함	O
D5	비	서늘	정상	약함	O
D6	비	서늘	정상	강함	X
D7	흐림	서늘	정상	강함	O
D8	맑음	포근	높음	약함	X
D9	맑음	서늘	정상	약함	O
D10	비	포근	정상	약함	O
D11	맑음	포근	정상	강함	O
D12	흐림	포근	높음	강함	O
D13	흐림	더움	정상	약함	O
D14	비	포근	높음	강함	X

- 날씨

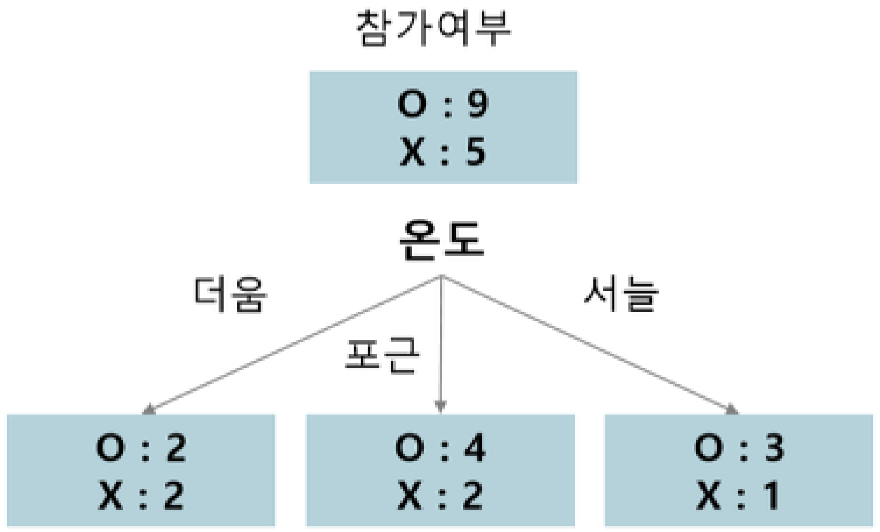
$$E(\text{경기}|\text{날씨}) = \frac{5}{14}\left(-\frac{2}{5}log_2\left(\frac{2}{5}\right)-\frac{3}{5}log_2\left(\frac{3}{5}\right)\right)$$
$$+\frac{4}{14}\left(-\frac{4}{4}log_2\left(\frac{4}{4}\right)-\frac{0}{4}log_2\left(\frac{0}{4}\right)\right)$$
$$+\frac{5}{14}\left(-\frac{3}{5}log_2\left(\frac{3}{5}\right)-\frac{2}{5}log_2\left(\frac{2}{5}\right)\right)$$
$$= 0.694$$



날짜	날씨	온도	습도	바람	참가여부
D1	맑음	더움	높음	약함	X
D2	맑음	더움	높음	강함	X
D3	흐림	더움	높음	약함	O
D4	비	포근	높음	약함	O
D5	비	서늘	정상	약함	O
D6	비	서늘	정상	강함	X
D7	흐림	서늘	정상	강함	O
D8	맑음	포근	높음	약함	X
D9	맑음	서늘	정상	약함	O
D10	비	포근	정상	약함	O
D11	맑음	포근	정상	강함	O
D12	흐림	포근	높음	강함	O
D13	흐림	더움	정상	약함	O
D14	비	포근	높음	강함	X

- 온도

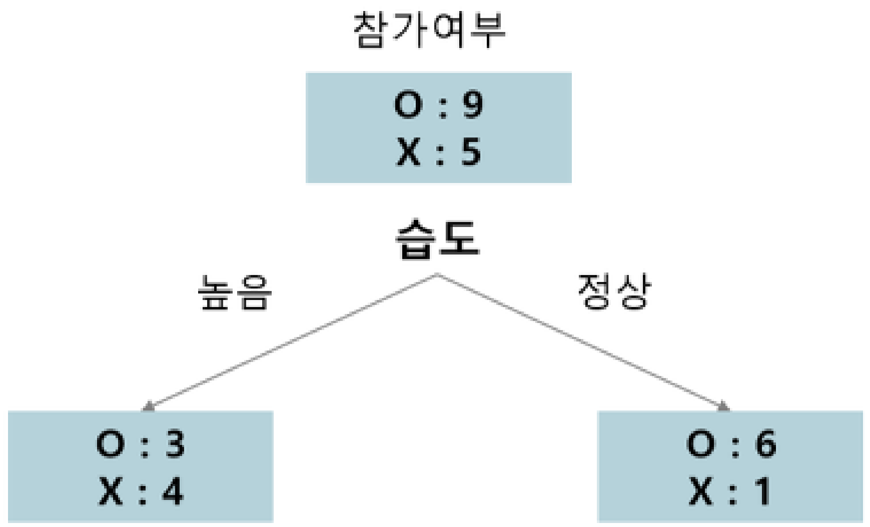
$$
\begin{aligned}
E(\text{경기}|\text{온도}) &= \frac{4}{14}\left(-\frac{2}{4}log_2\left(\frac{2}{4}\right)-\frac{2}{4}log_2\left(\frac{2}{4}\right)\right)\\
&+\frac{6}{14}\left(-\frac{4}{6}log_2\left(\frac{4}{6}\right)-\frac{2}{6}log_2\left(\frac{2}{6}\right)\right)\\
&+\frac{4}{14}\left(-\frac{3}{4}log_2\left(\frac{3}{4}\right)-\frac{1}{4}log_2\left(\frac{1}{4}\right)\right)\\
&= 0.911
\end{aligned}
$$



날짜	날씨	온도	습도	바람	참가여부
D1	맑음	더움	높음	약함	X
D2	맑음	더움	높음	강함	X
D3	흐림	더움	높음	약함	O
D4	비	포근	높음	약함	O
D5	비	서늘	정상	약함	O
D6	비	서늘	정상	강함	X
D7	흐림	서늘	정상	강함	O
D8	맑음	포근	높음	약함	X
D9	맑음	서늘	정상	약함	O
D10	비	포근	정상	약함	O
D11	맑음	포근	정상	강함	O
D12	흐림	포근	높음	강함	O
D13	흐림	더움	정상	약함	O
D14	비	포근	높음	강함	X

- **습도**

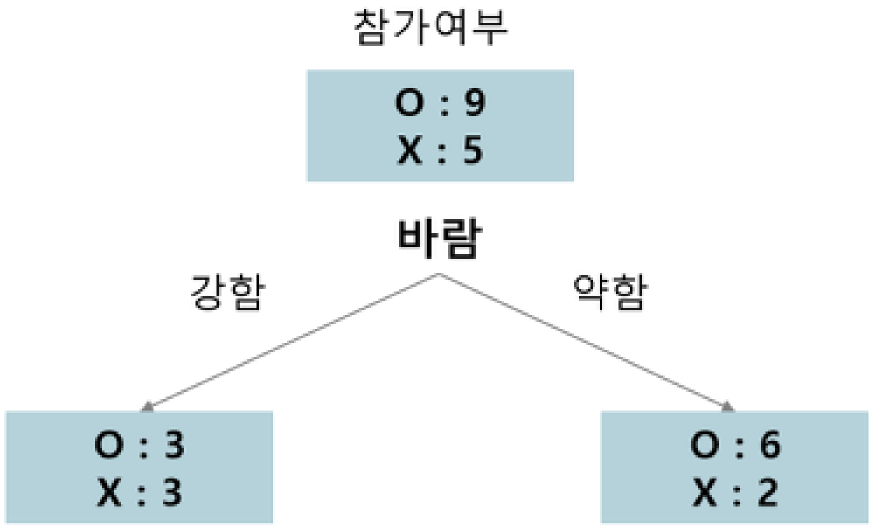
$$E(\text{경기}|\text{습도}) = \frac{7}{14}\left(-\frac{3}{7}log_2\left(\frac{3}{7}\right) - \frac{4}{7}log_2\left(\frac{4}{7}\right)\right) + \frac{7}{14}\left(-\frac{6}{7}log_2\left(\frac{6}{7}\right) - \frac{1}{7}log_2\left(\frac{1}{7}\right)\right) = 0.789$$



날짜	날씨	온도	습도	바람	참가여부
D1	맑음	더움	높음	약함	X
D2	맑음	더움	높음	강함	X
D3	흐림	더움	높음	약함	O
D4	비	포근	높음	약함	O
D5	비	서늘	정상	약함	O
D6	비	서늘	정상	강함	X
D7	흐림	서늘	정상	강함	O
D8	맑음	포근	높음	약함	X
D9	맑음	서늘	정상	약함	O
D10	비	포근	정상	약함	O
D11	맑음	포근	정상	강함	O
D12	흐림	포근	높음	강함	O
D13	흐림	더움	정상	약함	O
D14	비	포근	높음	강함	X

- **바람**

$$E(\text{경기}|\text{바람}) = \frac{6}{14}\left(-\frac{3}{6}log_2\left(\frac{3}{6}\right)-\frac{3}{6}log_2\left(\frac{3}{6}\right)\right)$$
$$+\frac{8}{14}\left(-\frac{6}{8}log_2\left(\frac{6}{8}\right)-\frac{2}{8}log_2\left(\frac{2}{8}\right)\right)$$
$$= 0.892$$



날짜	날씨	온도	습도	바람	참가여부
D1	맑음	더움	높음	약함	X
D2	맑음	더움	높음	강함	X
D3	흐림	더움	높음	약함	O
D4	비	포근	높음	약함	O
D5	비	서늘	정상	약함	O
D6	비	서늘	정상	강함	X
D7	흐림	서늘	정상	강함	O
D8	맑음	포근	높음	약함	X
D9	맑음	서늘	정상	약함	O
D10	비	포근	정상	약함	O
D11	맑음	포근	정상	강함	O
D12	흐림	포근	높음	강함	O
D13	흐림	더움	정상	약함	O
D14	비	포근	높음	강함	X

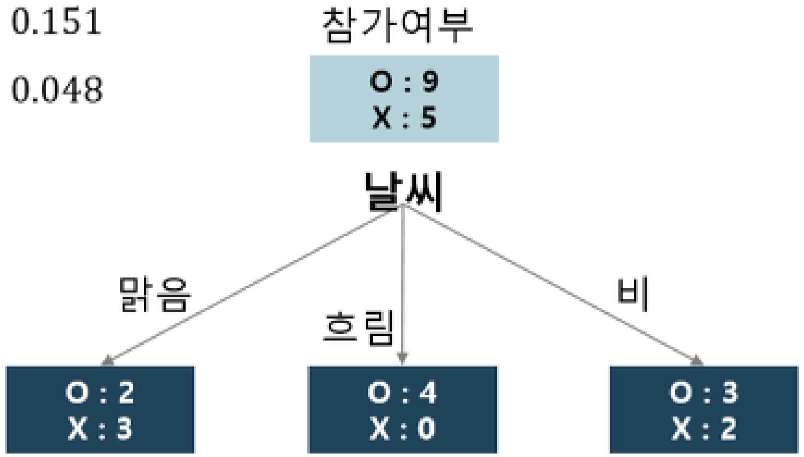
(3) 각 특징들에 대한 정보획득이 최대가 되는 조건을 찾아 분할

$IG(경기, 날씨) = E(경기) - E(경기|날씨) = 0.94 - 0.694 = \mathbf{0.246}$ → 날씨로 가장 먼저 분류

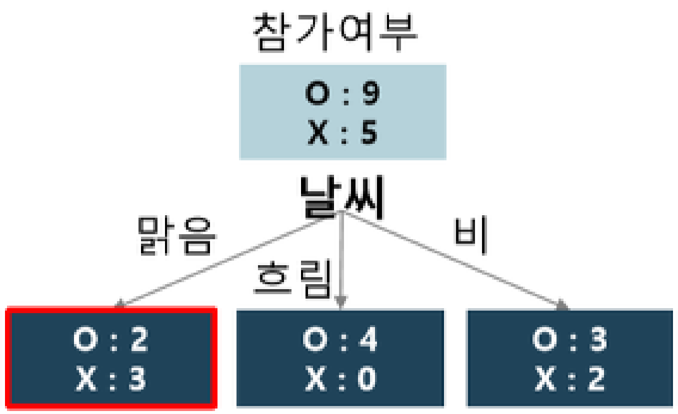
$IG(경기, 온도) = E(경기) - E(경기|온도) = 0.94 - 0.911 = 0.029$

$IG(경기, 습도) = E(경기) - E(경기|습도) = 0.94 - 0.789 = 0.151$

$IG(경기, 바람) = E(경기) - E(경기|바람) = 0.94 - 0.892 = 0.048$



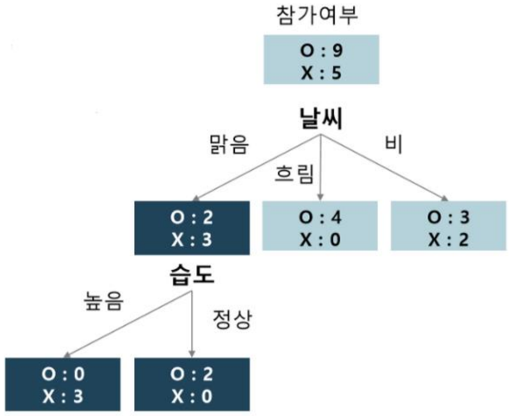
(4) 모든 Leaf Node들의 불순도가 0이 될 때까지 2~3단계 반복



날짜	날씨	온도	습도	바람	참가여부
D1	맑음	더움	높음	약함	X
D2	맑음	더움	높음	강함	X
D8	맑음	포근	높음	약함	X
D9	맑음	서늘	정상	약함	O
D11	맑음	포근	정상	강함	O

(Approach 1)

$$E(\text{경기}) = -\frac{2}{5}log_2\left(\frac{2}{5}\right) - \frac{3}{5}log_2\left(\frac{3}{5}\right) = 0.971$$



(Approach 2)

$$E(\text{경기}|\text{온도}) = \frac{2}{5}\left(-\frac{2}{2}log_2\left(\frac{2}{2}\right)\right) + \frac{2}{5}\left(-\frac{1}{2}log_2\left(\frac{1}{2}\right) - \frac{1}{2}log_2\left(\frac{1}{2}\right)\right) + \frac{1}{5}\left(-\frac{1}{1}log_2\left(\frac{1}{1}\right)\right) = 0.4$$

$$E(\text{경기}|\text{습도}) = \frac{3}{5}\left(-\frac{3}{3}log_2\left(\frac{3}{3}\right)\right) + \frac{2}{5}\left(-\frac{2}{2}log_2\left(\frac{2}{2}\right)\right) = 0$$

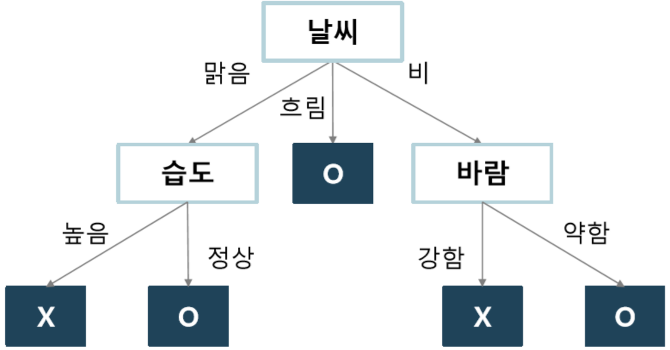
$$E(\text{경기}|\text{바람}) = \frac{2}{5}\left(-\frac{1}{2}log_2\left(\frac{1}{2}\right) - \frac{1}{2}log_2\left(\frac{1}{2}\right)\right) + \frac{3}{5}\left(-\frac{1}{3}log_2\left(\frac{1}{3}\right) - \frac{2}{3}log_2\left(\frac{2}{3}\right)\right) = 0.951$$

(Approach 3)

$$IG(\text{경기}, \text{온도}) = E(\text{경기}) - E(\text{경기}|\text{온도}) = 0.971 - 0.4 = 0.571$$

$$IG(\text{경기}, \text{습도}) = E(\text{경기}) - E(\text{경기}|\text{습도}) = 0.971 - 0 = \mathbf{0.971}$$

$$IG(\text{경기}, \text{바람}) = E(\text{경기}) - E(\text{경기}|\text{바람}) = 0.971 - 0.951 = 0.02$$



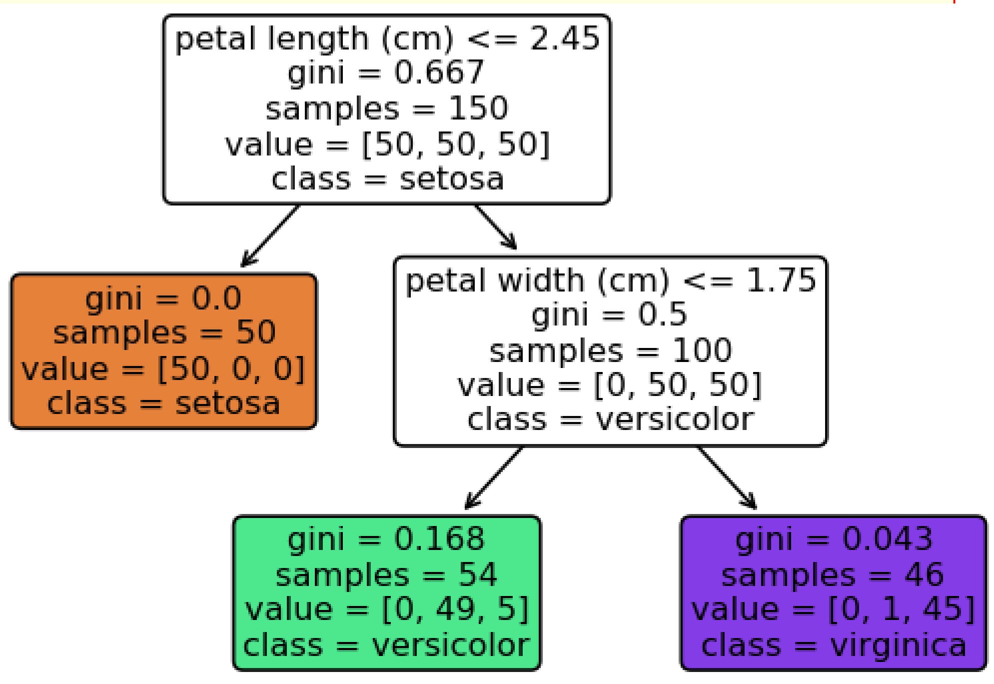
sklearn.tree.**plot_tree**

Decision tree의 시각화를 위한 함수

```
plot_tree(decision_tree, max_depth=None,
          feature_names=None, class_names=None,
          filled=False, proportion=False,
          rounded=False, precision=3, fontsize=None)
```

- max_depth: if None, the tree is fully generated.
- feature_names: list of strings
- class_names: list of str or bool
- proportion: True로 지정하면, 값 대신 비율로 표현
- precision: number of digits of precision
- filled, rounded, fontsize: 노드의 모양과 폰트크기 설정

```python
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data[:, 2:] # 꽃잎의 길이와 너비
y = iris.target

clf = DecisionTreeClassifier(max_depth=2)
clf.fit(X, y)

plt.figure(dpi=120)
plot_tree(clf, feature_names=iris.feature_names[2:],
            class_names=iris.target_names,
            filled=True, rounded=True)
plt.show()
```



과적합(overffiting) 방지

- 가지치기(pruning)
  - 결정트리의 특정 노드 밑의 하부 트리를 제거하여 일반화 성능을 높히는 것
  - $Cost(T) = ERR(T) + \alpha\, L(T)$
    - $ERR(T)$: 검증데이터에 대한 오분류율
    - $L(T)$: Leaf Node의 개수(구조의 복잡성)
  - Scikit-learn 옵션
    - min_samples_split : 하나의 node에 들어있는 샘플의 최소 개수
    - min_samples_leaf : 하나의 leaf node에 들어있는 샘플의 최소 개수
    - max_depth : 분리의 최대 깊이

- 결정트리의 주요 단점
  - 과적합으로 알고리즘 성능이 떨어질 수 있다.
  - 한 번에 하나의 변수만을 고려하므로 변수간 상호작용을 파악하기가 어렵다.
  - 약간의 차이에 따라(레코드의 개수의 약간의 차이) 트리의 모양이 많이 달라질 수 있다.
  - 계단모양의 결정경계를 만드므로, 학습데이터의 회전에 민감하다.

- 이러한 문제를 해결하기 위한 모델
  - **랜덤 포레스트(Random Forest)** : 다수의 결정트리들을 학습하는 앙상블(ensemble) 방법