Support Vector Machine Practice

Hanwool Jeong

hwjeong@kw.ac.kr

Hyperparameter

- A model hyperparameter is a configuration that is external to the model and whose value cannot be estimated from data.

  - They are often used in processes to help estimate model parameters.
  - They are often specified by the practitioner.
  - They can often be set using heuristics.
  - They are often tuned for a given predictive modeling problem.

Revisit Soft Margin

- Maximize the number of samples 1) while the minimizing the number of samples of 2) & 3)
- We can revise the hard margin example into

Minimize $\frac{\|\boldsymbol{w}\|^2}{2}$

$g(\mathbf{x_i}) = 1 - y_i(\mathbf{w}^T\mathbf{x_i} + b) \le 0$

➡

Minimize $\frac{\|\boldsymbol{w}\|^2}{2} + C\sum_{i=1}^{N} \xi_i$

subject to $0 \le \xi_i$

$g(\mathbf{x_i}) = 1 - \xi_i - y_i(\mathbf{w}^T\mathbf{x_i} + b) \le 0$

- We can repeat the procedure by the Lagrange aux. function of

$$L = \frac{\|\boldsymbol{w}\|^2}{2} + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \mu_i \{y_i(\mathbf{w}^T\mathbf{x_i} + b) - 1 + \xi_i\}$$

Revisit Famous Kernel Function

- Radial basis function or **RBF** kernel or Gaussian kernel

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

- Polynomial kernel

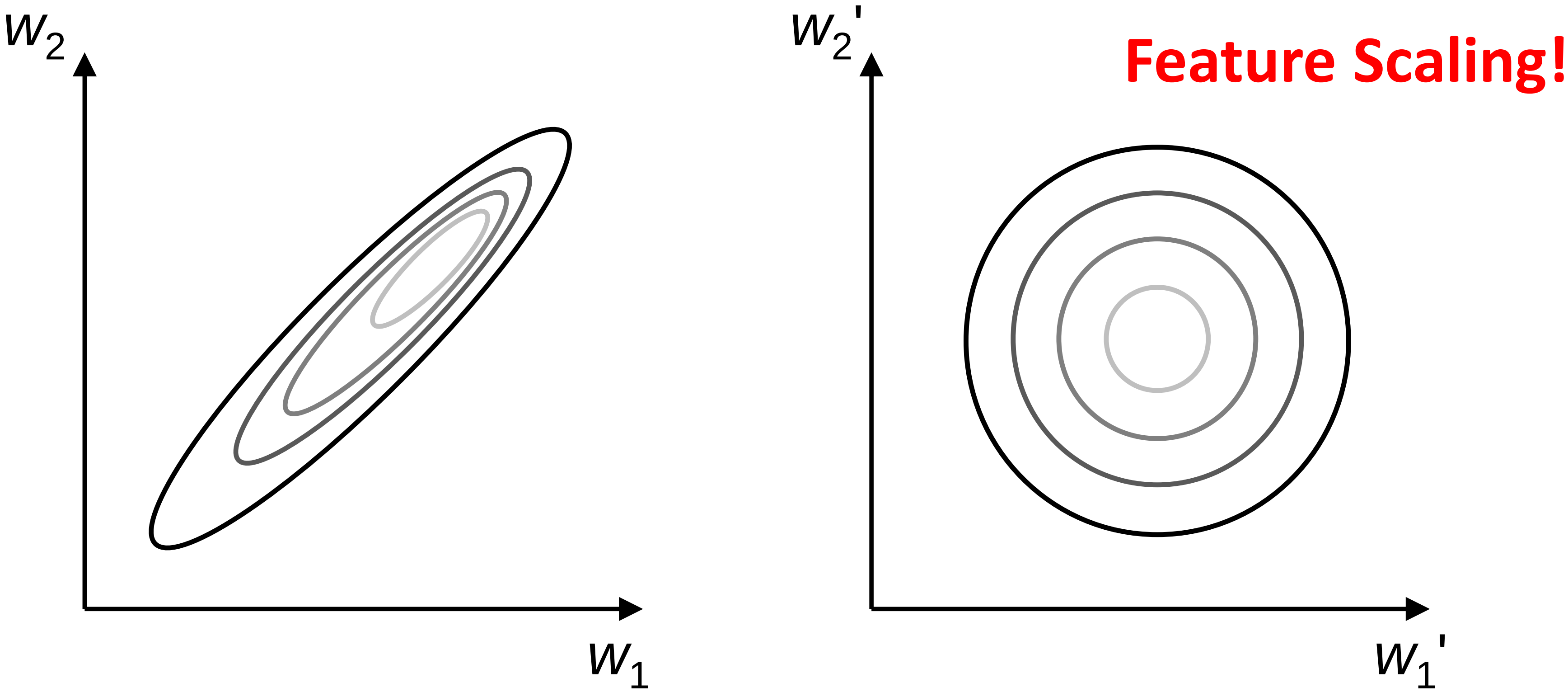
$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x}^T \mathbf{x}' + r)^M$$

- Sigmoid kernel or tanh kernel

$$\kappa(\mathbf{x}, \mathbf{x}') = \tanh(\gamma \mathbf{x}^T \mathbf{x}' + r)$$

Gradient Descent in Different Data Scale

$$w_{next} = w_{present} - \eta \nabla NLL(w)$$



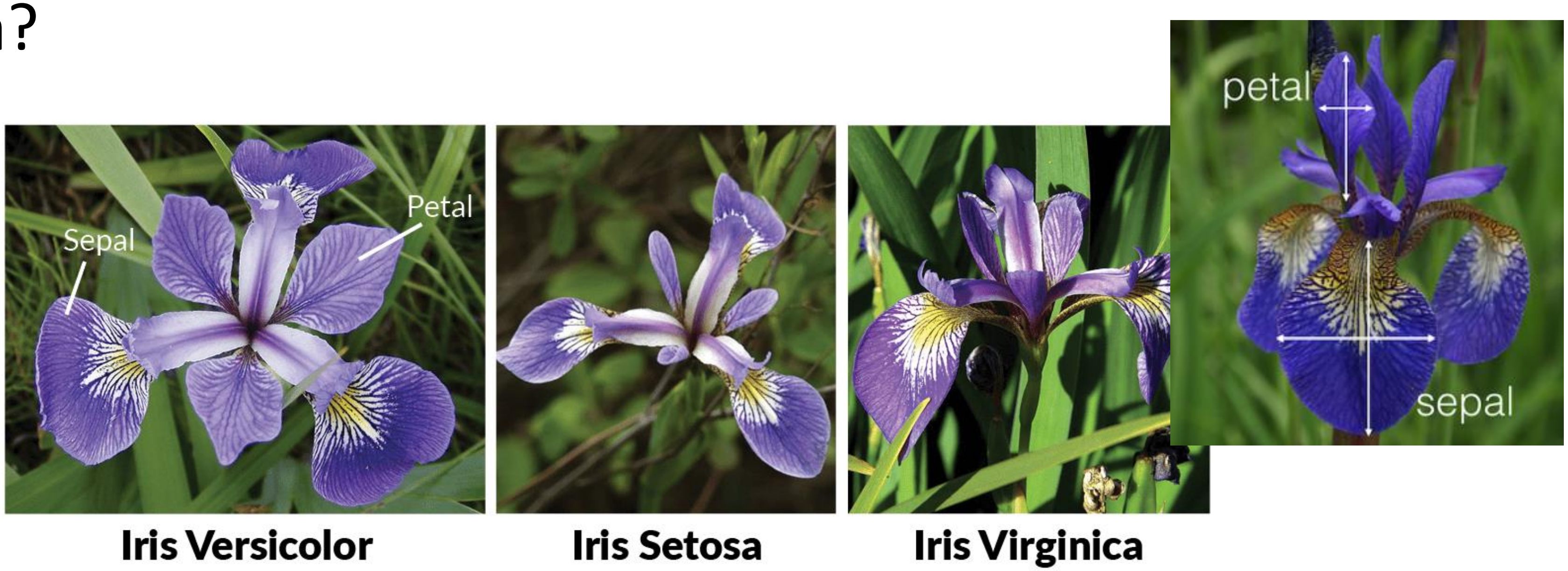
Feature Scaling in Python

- Min-max scaling (Normalization)
  - Make the data limited to [0,1]
  - Vulnerable to outlier noise
- Standardization
  - Does not limited to certain range
  - Robust to outlier noise
- Usage:

```python
from sklearn.preprocessing import StandardScaler

X = np.array([
        [2, -3],
        [4, 1],
        [0, -2],
        [10, 3]])

scaler = StandardScaler()
scaler.fit(X)
X_std = scaler.transform(X)
```

Load iris Data for SVM Practice

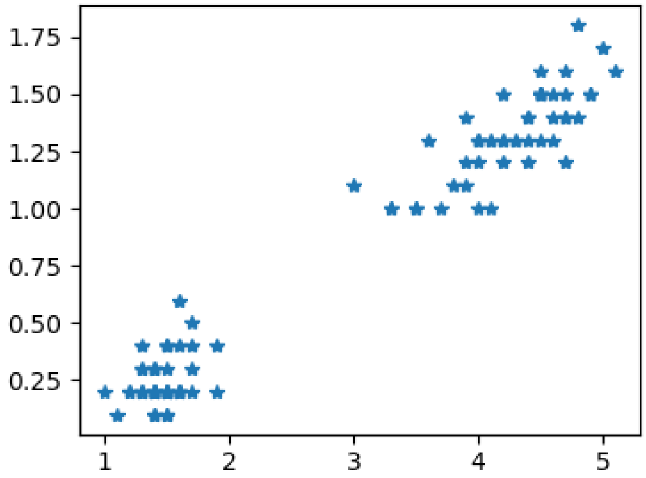
- Iris data?



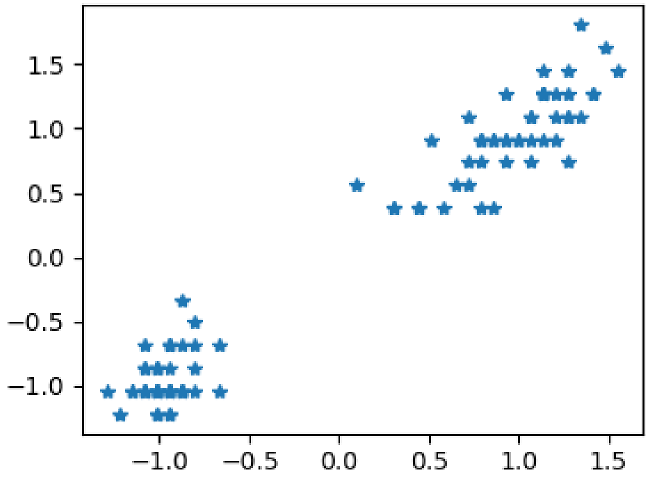

```
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()

X = iris["data"][0:100, (2,3)] #petal length & width
Y = iris["target"][0:100]

f1, ax1 = plt.subplots()
ax1.plot(X[:,0], X[:,1], '*')
```

Classified iris Datasets w/ Standardization

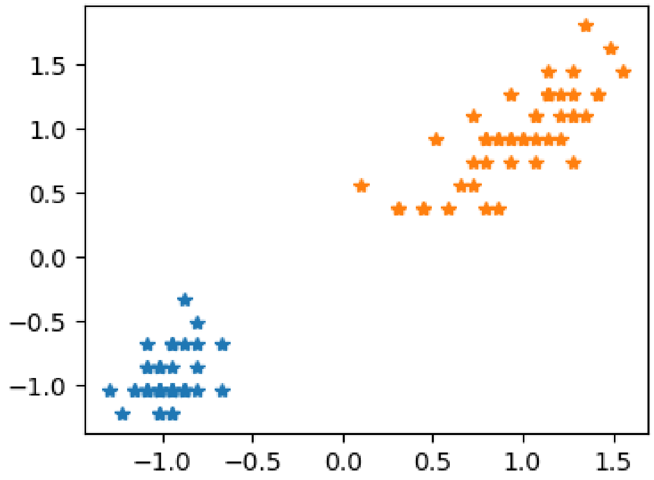
iris petal width vs. petal length



Standardization



Classified



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

from sklearn import datasets
from sklearn.preprocessing import StandardScaler

plt.close("all")

iris = datasets.load_iris()
X = iris["data"][0:100, (2,3)] #petal length & width
Y = iris["target"][0:100]
f1, ax1 = plt.subplots()
ax1.plot(X[:,0], X[:,1], '*')

scaler = StandardScaler()
scaler.fit(X)
X_std = scaler.transform(X)
f2, ax2 = plt.subplots()
ax2.plot(X_std[:,0], X_std[:,1], '*')

df_clf = pd.DataFrame(np.c_[X_std, Y])
df_clf.columns = ["petalLength", "petalWidth", "target"]
df_clf_group = df_clf.groupby("target")
f3, ax3 = plt.subplots()
for target, group in df_clf_group:
        ax3.plot(group.petalLength, group.petalWidth, '*', label = "target")
```

SVM by sklearn

- Although we can simply write a code for SVM by realizing:

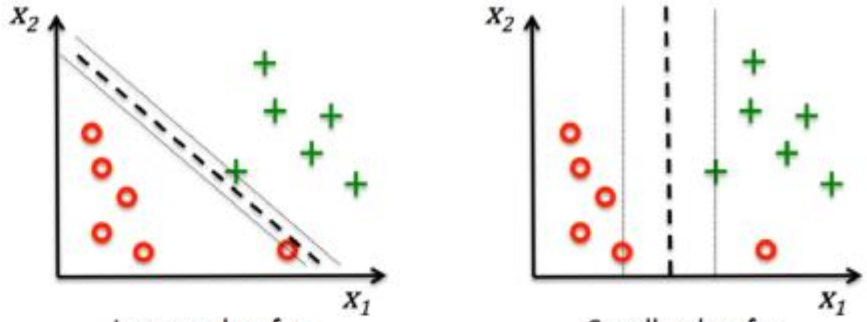
$$L(\mu)= \sum_{i=1}^{N} \mu_i - \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} \mu_i\mu_j y_i y_j \mathbf{x_i}^T\mathbf{x_j}}{2}$$

Then, w & b from optimal μ

- We can utilize the pre established code:
- sklearn.svm.SVC — scikit-learn 0.24.2 documentation

Usage of SVM

```python
svm_clf = SVC(C=0.01, kernel="linear")
svm_clf.fit(X_std, Y)
```

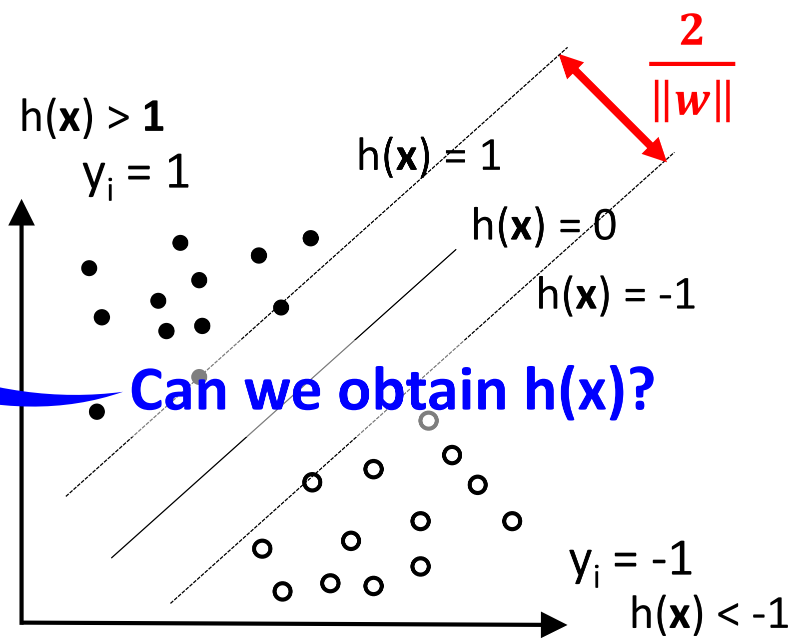


Large value for parameter *C*　　Small value for parameter *C*

```
fit(X, y, sample_weight=None)

Fit the SVM model according to the given training data.
```

- Now, check svm_clf in Variable Explorer.
- What is matter for us?

svm_clf.decision_function(X)

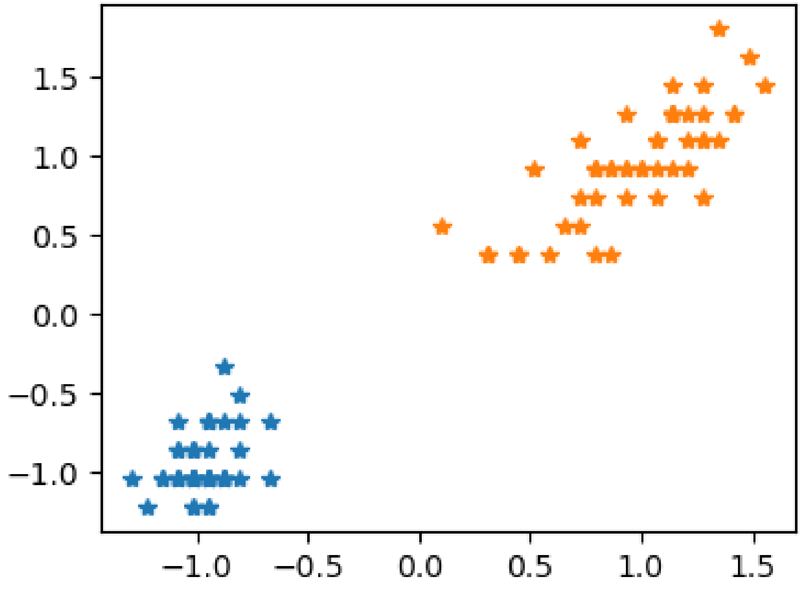


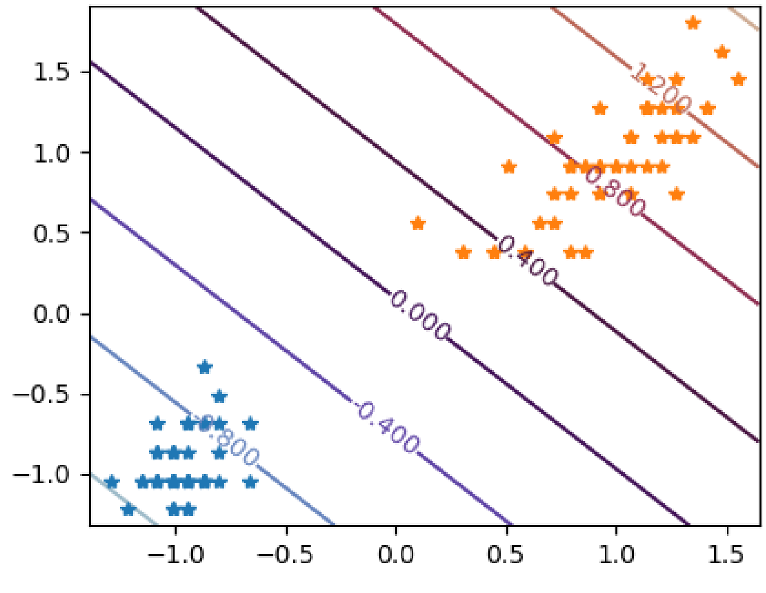
Visualization of SVM Training Results

- We have **svm_clf.decision_function(X)**

➔ **How can we visualize this?**

- We can utilize contour with meshgrid

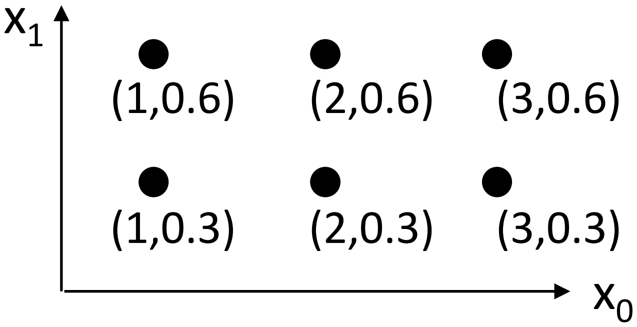




Contour

- ax3.contour(x0Plt, x1Plt, h)

[h(1, 0.3), h(2, 0.3), h(3, 0.3)
h(1, 0.6), h(2, 0.6), h(3, 0.6)]



How should x0Plt or x1Plt look like?

[1, 2, 3
1, 2, 3]

[0.3, 0.3, 0.3
0.6, 0.6, 0.6]

How can we make if we have
x0 = [1,2,3] and x1 = [0.3, 0.6]?

[x0Plt, x1Plt] = np.meshgrid(x0, x1)

		x0 [1	2	3]
x1	**[0.3**	1, 0.3	2, 0.3	3, 0.3
	0.6]	1, 0.6	2, 0.6	3, 0.6

↓

x0Plt

[1, 2, 3
1, 2, 3]

x1Plt

[0.3, 0.3, 0.3
0.6, 0.6, 0.6]

How Can We Achieve h(X)?

x0Plt	x1Plt		
[1, 2, 3 1, 2, 3]	[0.3, 0.3, 0.3 0.6, 0.6, 0.6]	➡	[h(1, 0.3), h(2, 0.3), h(3, 0.3) h(1, 0.6), h(2, 0.6), h(3, 0.6)]

We need [[1, 0.3]
[2, 0.3]
[3, 0.3]
[1, 0.6]
[2, 0.6]
[3, 0.6]]

Reshape vs. Ravel

a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])

a.reshape(3,4) ↓ ↑ **a.ravel()**

a = np.array([[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]])

x0Plt **x1Plt**

[1, 2, 3 [0.3, 0.3, 0.3
1, 2, 3] 0.6, 0.6, 0.6]

⬇

x0Plt.ravel() **x1Plt.ravel()**

[1, 2, 3, 1, 2, 3] [0.3, 0.3, 0.3, 0.6, 0.6, 0.6]

⬇

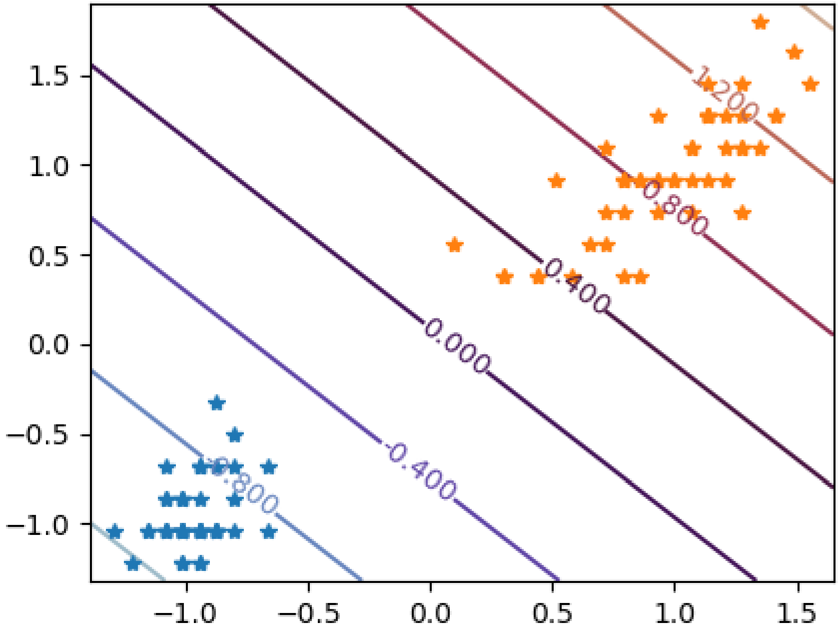
np.c_[**x0Plt.ravel()**, **x1Plt.ravel()**]

⬇

[[1, 0.3]
[2, 0.3]
[3, 0.3]
[1, 0.6]
[2, 0.6]
[3, 0.6]]

Code for Visualizing Contour

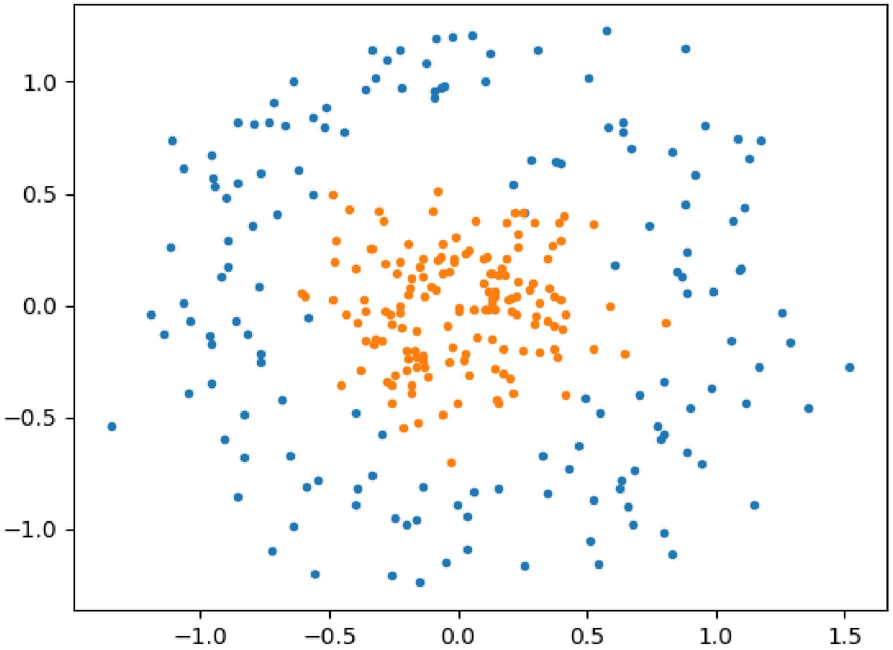
```
[x0Min, x0Max] = [min(X_std[:,0])-0.1, max(X_std[:,0])+0.1]
[x1Min, x1Max] = [min(X_std[:,1])-0.1, max(X_std[:,1])+0.1]
delta = 0.01
[x0Plt, x1Plt] = np.meshgrid(np.arange(x0Min, x0Max, delta),
np.arange(x1Min, x1Max, delta))
h = svm_clf.decision_function(np.c_[x0Plt.ravel(), x1Plt.ravel()])
h = h.reshape(x0Plt.shape)
CS = ax3.contour(x0Plt, x1Plt, h, cmap=plt.cm.twilight)
ax3.clabel(CS)
```

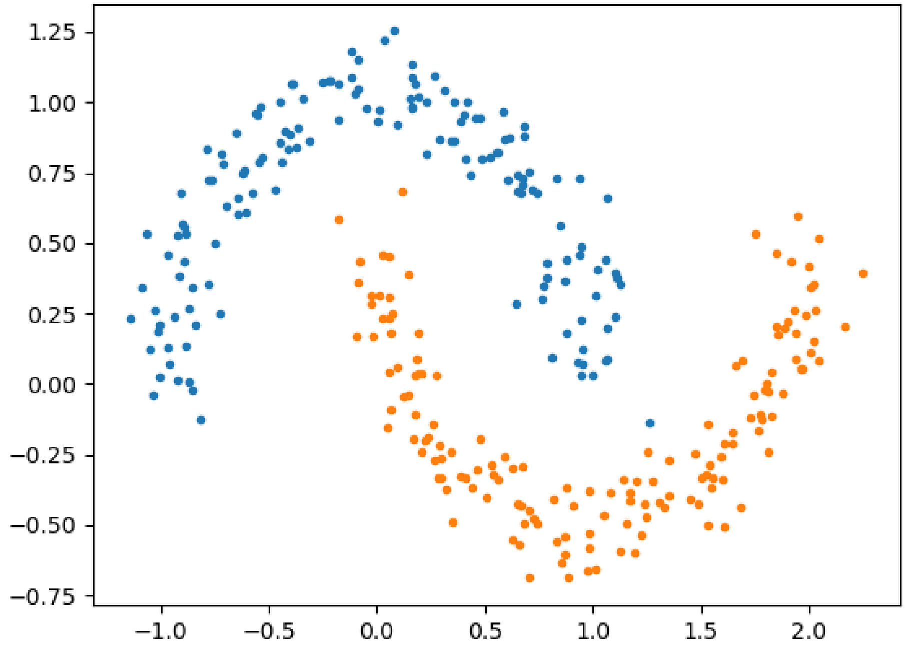


You also can use svm_clf.predict([[x1, x2]])

Moon-Data or Circle-Data

```
from sklearn import datasets
plt.close("all")
# [X,Y] = datasets.make_circles(n_samples = 300, shuffle = True , noise = 0.2, random_state = 15, factor = 0.3)
[X,Y] = datasets.make_moons(n_samples = 300, shuffle = True, noise = 0.1, random_state = 15)
```





Do it By Yourself!

- Apply “train_test_split” to test the SVM performance
- Apply kernel trick to data for **moon data** and **circle data** by changing
  - Kernel function ➔ rbf and poly
  - For rbf, change gamma
  - For poly, change degree/gamma/coef0
  - Repeat above various factor and noise of moon/circle data

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

$$\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x}^T \mathbf{x}' + r)^M$$