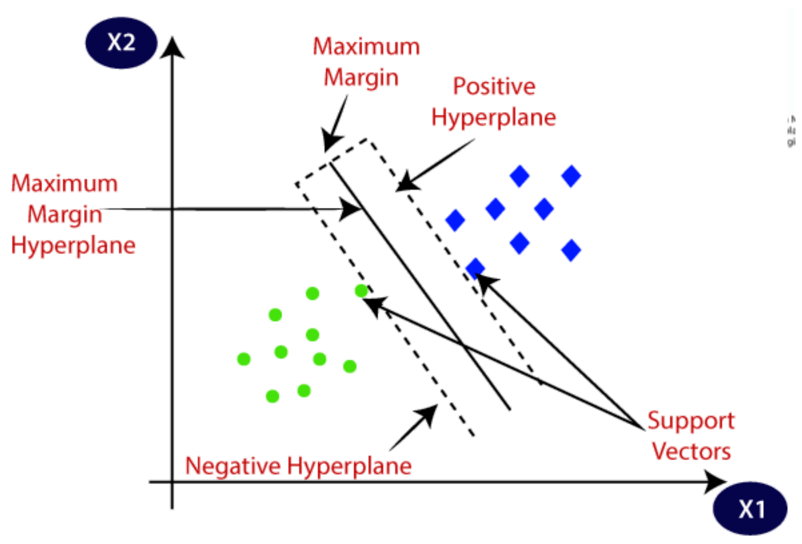
LESSON 13: SUPPORT VECTOR MACHINE



This lecture was refered by machinelearningcoban.com

1. Distance formular

In the 3D space, to calculate the distance between a point $X^* = (x_1^*, x_2^*, x_3^*)$ and a plane $w_1x_1 + w_2x_2 + w_3x_3 + b = 0$.

$$D = \frac{|w_1x_1^* + w_2x_2^* + w_3x_3^* + b|}{\sqrt{w_1^2 + w_2^2 + w_3^2}}$$

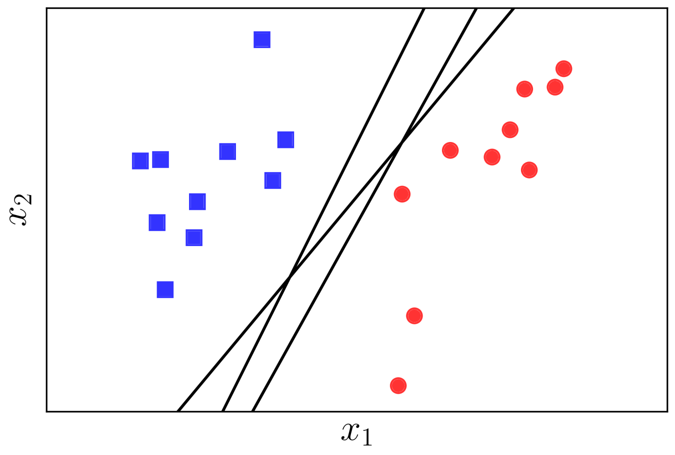
Generally to nD space, to calculate the distance between a point $X^* = (x_1^*, x_2^*, \ldots, x_n^*)$ and a hyper-plane $w_1x_1 + w_2x_2 + \cdots + w_nx_n + b = W^TX + b = 0$.

$$D = \frac{|W^TX^* + b|}{\sqrt{\sum_{i=1}^n w_i^2}}$$

2. SVM introduction

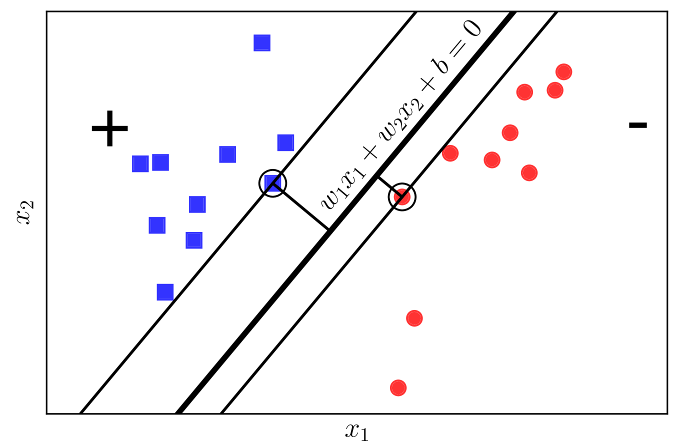
With a classification model, like Logistics Regression, we can find lots of solutions to classify data samples exactly.

But one problem is which solution is the best among all of these solutions?



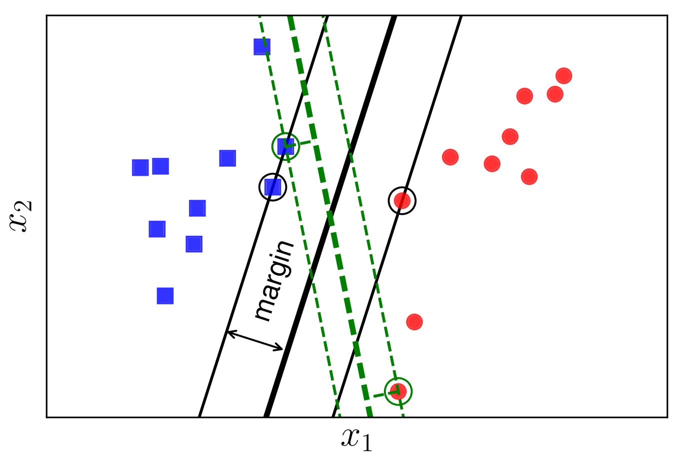
This solution show a problem is that the distance between a class point and the solution is larger than the distance between another class point and the solution.

=> That's unfair between the two classes



This solution show a problem is that the distances between each class point and the solution are equal but this distance is too small.

=> That's not good for classification



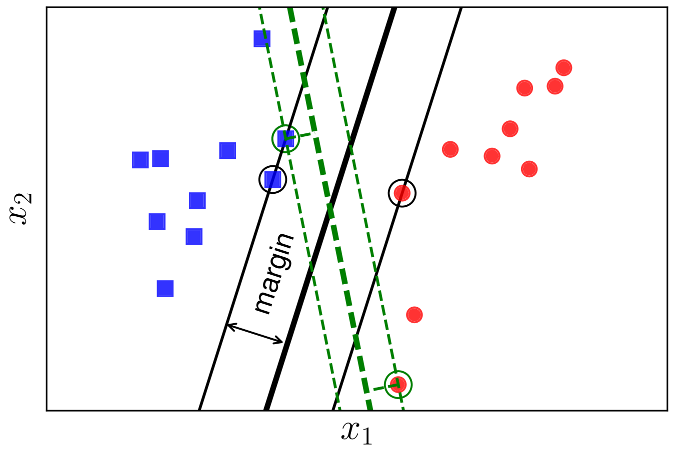
These are the reasons why SVM try to find a classification boundary which assure: **The distances (or can be called `margin`) between the solution and nearest point of each class are equal and largest.**

SVM is better than Neural network with only one layer - Logistic Regression.

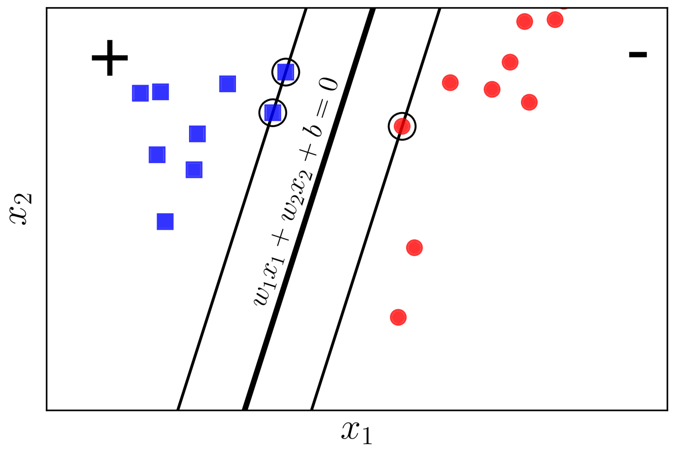
3. Optimization

We have a training set $(X_1, y_1), (X_2, y_2), \ldots, (X_N, y_N)$ while X_i is the data sample vector n dimension and y_i is the label ($y_i = 1$ or $y_i = -1$)

To simplify, we consider 2D space,



We assume that $W^TX + b = w_1x_1 + w_2x_2 + b = 0$ is the classification boundary. We have the distance between a data point (X_i, y_i) and the classification boundary



$$D = \frac{y_i(W^TX_i + b)}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

and margin is the minimum distance of all data points,

$$margin = \min_i \frac{y_i(W^TX_i + b)}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

We have to optimize the value of margin or, in other words, we have to find W and b to maximize the margin.

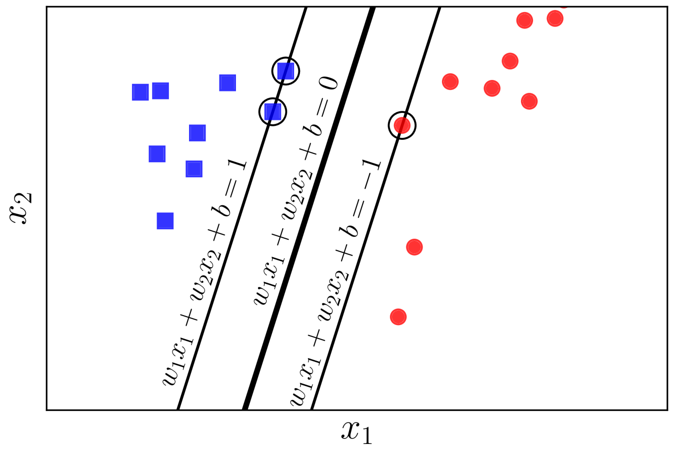
$$(W,b) = \arg\max_{W,b}\left\{\min_i \frac{y_i(W^TX_i+b)}{\sqrt{\sum_{i=1}^d w_i^2}}\right\}$$

It's complex to optimize the above formular, so we consider the classification boundary

$$W^TX+b = w_1x_1+w_2x_2+b=0$$
$$kW^TX+kb = kw_1x_1+kw_2x_2+kb=0$$

If we multiply both side by $k>0$, the classification boundary is not changed. So, we get the numerator $y_i(W^TX_i+b)$ from `margin formular` and assume that

$$y_i(W^TX_i+b)=v$$
$$y_i(kW^TX_i+kb)=1$$



With that assumption, we have

$$\forall i, y_i(kW^TX_i+kb)\geq 1$$

and we optimize

$$(W,b) = \arg\max_{W,b}\left\{\min_i \frac{y_i(W^TX_i+b)}{\sqrt{\sum_{i=1}^n w_i^2}}\right\}$$
$$= \arg\max_{W,b}\frac{1}{\sqrt{\sum_{i=1}^n w_i^2}}$$
$$\forall n, y_i(kW^TX_i+kb)\geq 1$$

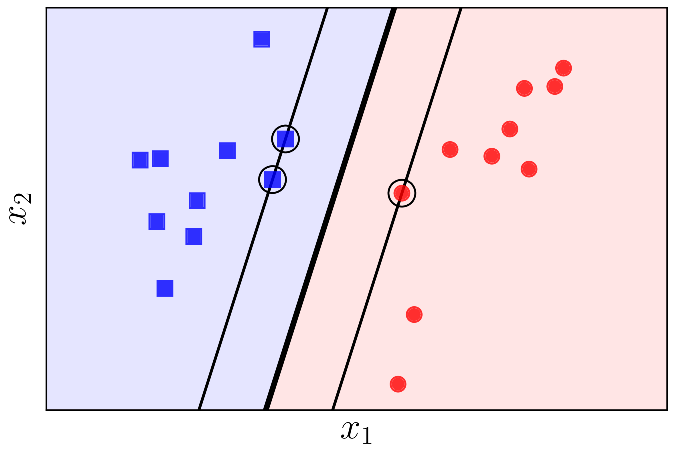
We modify to get the more simple optimization formular

$$(W,b) = \arg\min_{W,b}\sqrt{\sum_{i=1}^n w_i^2}$$
$$= \arg\min_{W,b}\sum_{i=1}^n w_i^2$$
$$\forall n, y_i(kW^TX_i+kb)\geq 1$$

To optimize this formular, we use Convex optimization algorithms.

4. Soft-margin SVM

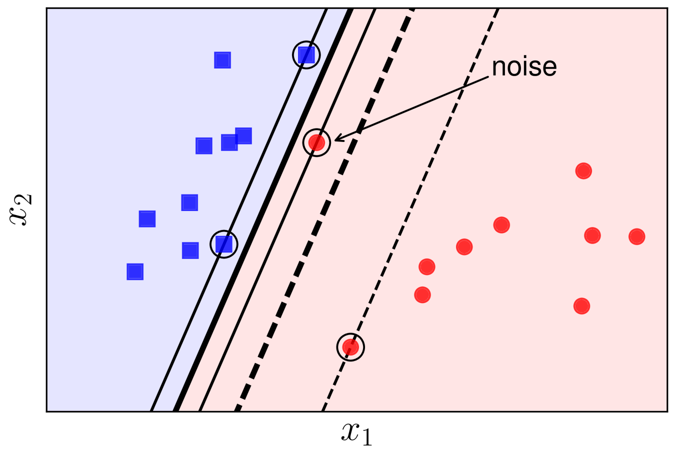
One problem with SVM is that this model works with only linearly separable dataset.



How about linearly separable dataset but contains noise?

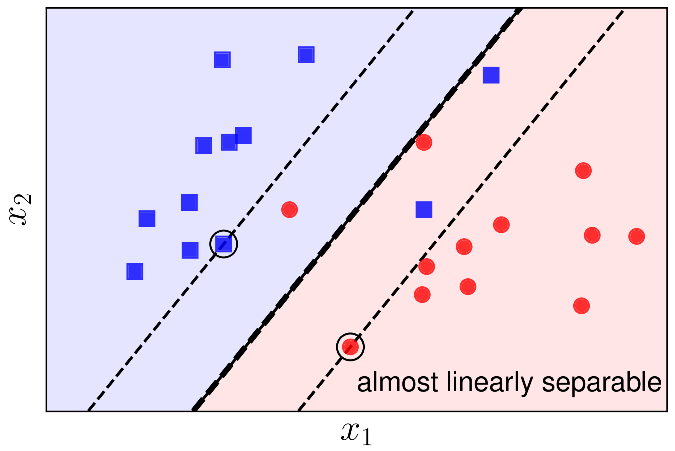
SVM can find a solution in this case, but the solution is not so good because of the margin is too small.

That's why we call SVM is sensitive to noise.



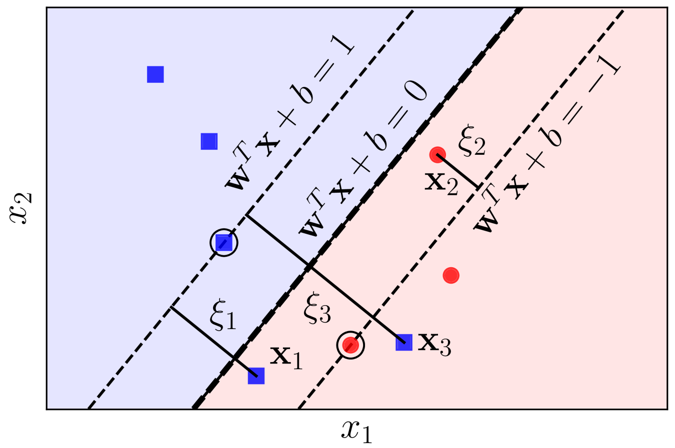
And, now about almost linearly separable dataset?

SVM cannot find a solution in this case!



But, on both the above cases, if we ignore some data points (allow them to fall inside the margin line), SVM can propose a really good solution with a large margin (dot line).

That's why we need another version of SVM - **Soft-margin SVM**, and the original version of SVM become **Hard-margin SVM**.



In Soft-margin SVM, we can ignore some data points, but we have to minimize the number of ignored data points.

So, the optimization problem of Soft-margin SVM is the combination of `maximizing the margin` and `minimizing the number of ignored data points`.

To maximize the margin, like Hard-margin SVM, we optimize $(W, b) = \arg\min_{W,b} \sum_{i=1}^{d} w_i^2$.

To minimize the number of ignored data points, for each data point x_i in the dataset, we propose a variable called ξ to validate their position respected to the margin.

For each data point x_i in the dataset,

- if x_i is classified exactly and x_i is outside of the margin, $\xi_i = 0$.
- if x_i is classified exactly and x_i is inside of the margin, $0 < \xi_n < 1$.
- if x_i is not classified, $1 < \xi_n$.

We have the following optimization problem for Hard-margin SVM,

$$(W, b) = \arg\min_{W,b} \sum_{i=1}^{d} w_i^2$$

$$\forall n, y_i(kW^T X_i + kb) \geq 1$$

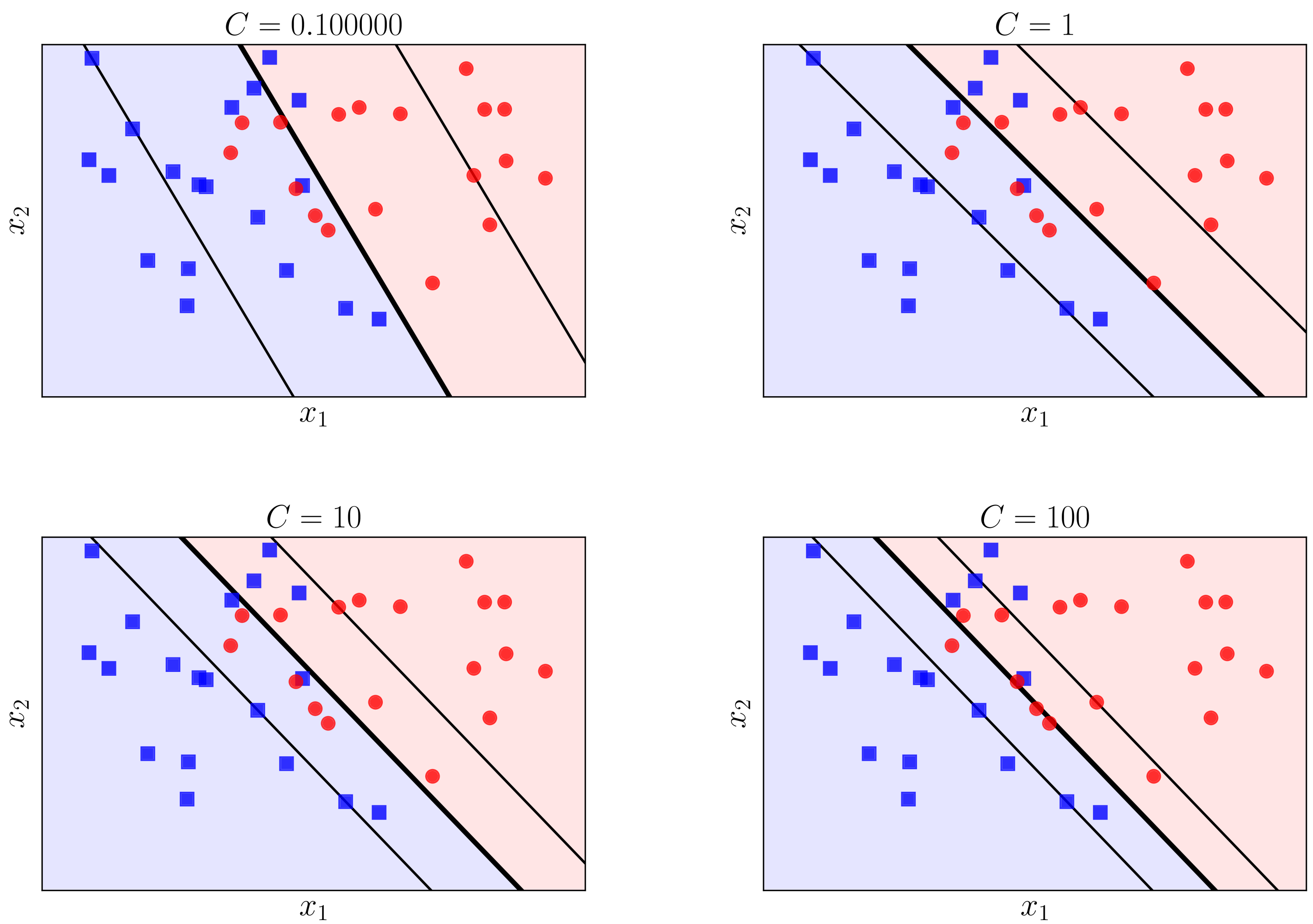
We have the following optimization problem for Soft-margin SVM with ξ in the optimization condition.

$$(W, b) = \arg\min_{W,b} \sum_{i=1}^{d} w_i^2 + C \sum_{i=1}^{N} \xi_i$$

$$\forall n, y_i(kW^T X_i + kb) \geq 1 - \xi_i$$

The role of C here is to balance the terms in the function.

- With small C, Soft-margin SVM will ignore lots of points to maximize the margin.
- With big C, Soft-margin SVM will try to ignore zero point. In case of linearly separable dataset, Soft-margin SVM will optimize to become the optimal Hard-margin SVM.

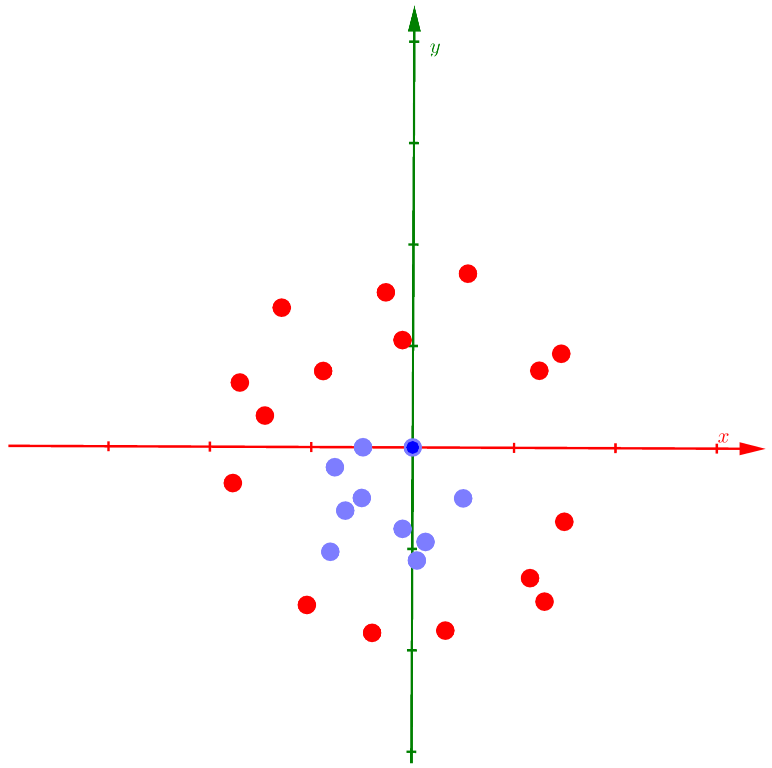


5. Kernel SVM

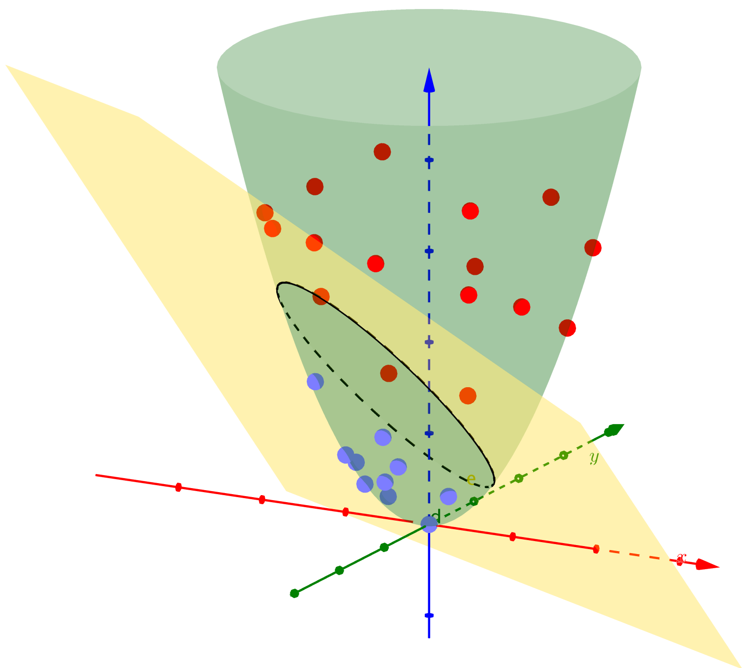
The original SVM (Hard-margin SVM) works with linearly separable dataset

The Soft-margin SVM works with linearly separable dataset with noise or almost linearly separable dataset.

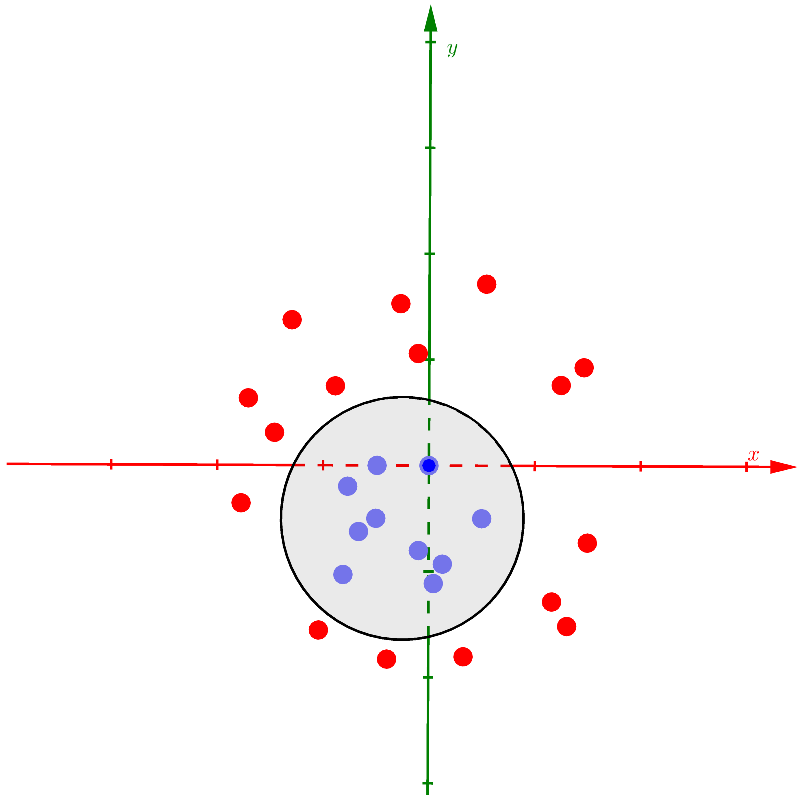
And, how about non-linearly separable dataset?



Kernel SVM in this case will propose a function Φ to map each data point x_i in the dataset to the new data space. And in this space, our non-linearly separable dataset can become linearly separable dataset or almost linearly separable dataset.



In the new data space, we can use Soft-margin SVM or Hard-margin SVM to find the classification boundary. With this classification boundary in the new data space, we can find the classification boundary in the original data space.



To save the computation cost, the function Φ can be **kernel function**. That's why we call **Kernel SVM**.

Instead of directly map data point x_i in the dataset to the new data space, kernel function try to calculate the relationship between data point in the new data space.

6. Implementation example

6.1. Prepare library and data

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

sns.set()
```

In [2]:
```
iris_df = sns.load_dataset('iris')
iris_df
```

Out[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica

	sepal_length	sepal_width	petal_length	petal_width	species
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

In [3]:
```
iris_df = iris_df.drop(columns=['sepal_length', 'sepal_width'])
iris_df
```

Out[3]:

	petal_length	petal_width	species
0	1.4	0.2	setosa
1	1.4	0.2	setosa
2	1.3	0.2	setosa
3	1.5	0.2	setosa
4	1.4	0.2	setosa
...	...	...	...
145	5.2	2.3	virginica
146	5.0	1.9	virginica
147	5.2	2.0	virginica
148	5.4	2.3	virginica
149	5.1	1.8	virginica

150 rows × 3 columns

In [4]:
```
sns.scatterplot(data=iris_df, x='petal_length', y='petal_width', hue='species')
plt.show()
```

6.2. Linearly separable dataset problem

In [5]:
```
iris_df['is_setosa'] = iris_df.species.apply(
    lambda x: True if x == 'setosa' else False)
iris_df
```

Out[5]:

	petal_length	petal_width	species	is_setosa
0	1.4	0.2	setosa	True
1	1.4	0.2	setosa	True
2	1.3	0.2	setosa	True
3	1.5	0.2	setosa	True
4	1.4	0.2	setosa	True
...	...	...	...	...
145	5.2	2.3	virginica	False
146	5.0	1.9	virginica	False
147	5.2	2.0	virginica	False
148	5.4	2.3	virginica	False
149	5.1	1.8	virginica	False

150 rows × 4 columns

In [6]:

```
iris_setosa_df = iris_df[iris_df.is_setosa]
iris_setosa_df.head()
```

Out[6]:

	petal_length	petal_width	species	is_setosa
0	1.4	0.2	setosa	True
1	1.4	0.2	setosa	True
2	1.3	0.2	setosa	True
3	1.5	0.2	setosa	True
4	1.4	0.2	setosa	True

In [7]:

```
iris_not_setosa_df = iris_df[~iris_df.is_setosa].sample(50, random_state=1)
iris_not_setosa_df.head()
```

Out[7]:

	petal_length	petal_width	species	is_setosa
130	6.1	1.9	virginica	False
134	5.6	1.4	virginica	False
83	5.1	1.6	versicolor	False
131	6.4	2.0	virginica	False
143	5.9	2.3	virginica	False

In [8]:

```
df = pd.concat([iris_setosa_df, iris_not_setosa_df])
df
```

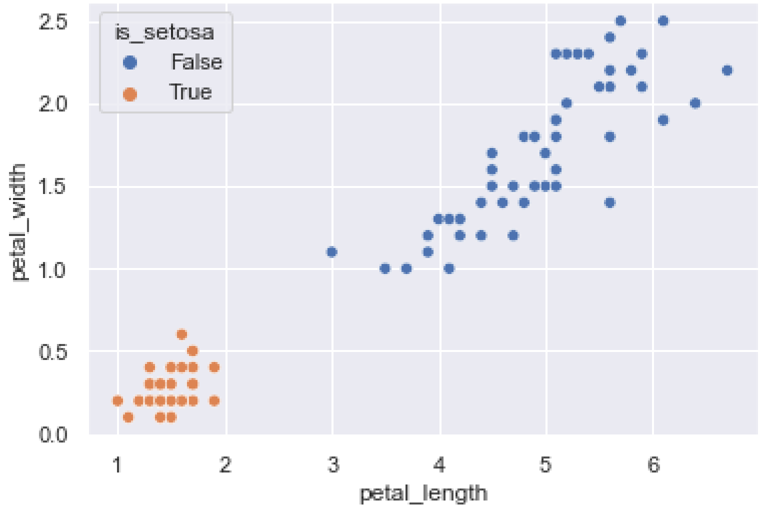
Out[8]:

	petal_length	petal_width	species	is_setosa
0	1.4	0.2	setosa	True
1	1.4	0.2	setosa	True
2	1.3	0.2	setosa	True
3	1.5	0.2	setosa	True

	petal_length	petal_width	species	is_setosa
4	1.4	0.2	setosa	True
...	...	...	...	...
91	4.6	1.4	versicolor	False
95	4.2	1.2	versicolor	False
141	5.1	2.3	virginica	False
76	4.8	1.4	versicolor	False
148	5.4	2.3	virginica	False

100 rows × 4 columns

In [9]:
```
sns.scatterplot(data=df, x='petal_length', y='petal_width', hue='is_setosa')
plt.show()
```



In [10]:
```
df
```

Out[10]:

	petal_length	petal_width	species	is_setosa
0	1.4	0.2	setosa	True
1	1.4	0.2	setosa	True
2	1.3	0.2	setosa	True
3	1.5	0.2	setosa	True
4	1.4	0.2	setosa	True
...	...	...	...	...
91	4.6	1.4	versicolor	False
95	4.2	1.2	versicolor	False
141	5.1	2.3	virginica	False
76	4.8	1.4	versicolor	False
148	5.4	2.3	virginica	False

100 rows × 4 columns

In [11]:
```
X = np.array(df.iloc[:, :2])
X.shape
```

Out[11]:
```
(100, 2)
```

In [12]:
```
X
```

Out[12]:
```
array([[1.4, 0.2],
       [1.4, 0.2],
       [1.3, 0.2],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.7, 0.4],
       [1.4, 0.3],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.5, 0.1],
       [1.5, 0.2],
       [1.6, 0.2],
       [1.4, 0.1],
       [1.1, 0.1],
       [1.2, 0.2],
       [1.5, 0.4],
       [1.3, 0.4],
       [1.4, 0.3],
       [1.7, 0.3],
       [1.5, 0.3],
       [1.7, 0.2],
       [1.5, 0.4],
       [1. , 0.2],
       [1.7, 0.5],
       [1.9, 0.2],
       [1.6, 0.2],
       [1.6, 0.4],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.6, 0.2],
       [1.6, 0.2],
       [1.5, 0.4],
       [1.5, 0.1],
       [1.4, 0.2],
       [1.5, 0.2],
       [1.2, 0.2],
       [1.3, 0.2],
       [1.4, 0.1],
       [1.3, 0.2],
       [1.5, 0.2],
       [1.3, 0.3],
       [1.3, 0.3],
       [1.3, 0.2],
       [1.6, 0.6],
       [1.9, 0.4],
       [1.4, 0.3],
       [1.6, 0.2],
       [1.4, 0.2],
       [1.5, 0.2],
       [1.4, 0.2],
       [6.1, 1.9],
       [5.6, 1.4],
       [5.1, 1.6],
       [6.4, 2. ],
       [5.9, 2.3],
       [4.1, 1. ],
       [4.7, 1.5],
       [5.6, 2.2],
       [5. , 1.5],
       [5.3, 2.3],
```

```
       [5.1, 1.9],
       [4. , 1.3],
       [4.5, 1.7],
       [5.9, 2.1],
       [5.1, 1.9],
       [3.9, 1.2],
       [3.7, 1. ],
       [4.2, 1.3],
       [5.6, 2.1],
       [3.5, 1. ],
       [4.9, 1.5],
       [4.9, 1.8],
       [5.2, 2. ],
       [5.5, 2.1],
       [3.9, 1.1],
       [4.5, 1.6],
       [5.7, 2.5],
       [5. , 1.7],
       [4.2, 1.3],
       [4.1, 1.3],
       [6.7, 2.2],
       [5.1, 1.8],
       [5.8, 2.2],
       [5.2, 2.3],
       [4.8, 1.8],
       [4.4, 1.2],
       [3. , 1.1],
       [6.1, 2.5],
       [4.7, 1.2],
       [4.5, 1.5],
       [5.6, 2.4],
       [5.6, 1.8],
       [4.9, 1.8],
       [4.4, 1.4],
       [5.1, 1.5],
       [4.6, 1.4],
       [4.2, 1.2],
       [5.1, 2.3],
       [4.8, 1.4],
       [5.4, 2.3]])
```

In [13]:

```
y = np.array(df['is_setosa'])
y.shape
```

Out[13]:

```
(100,)
```

In [14]:

```
y
```

Out[14]:

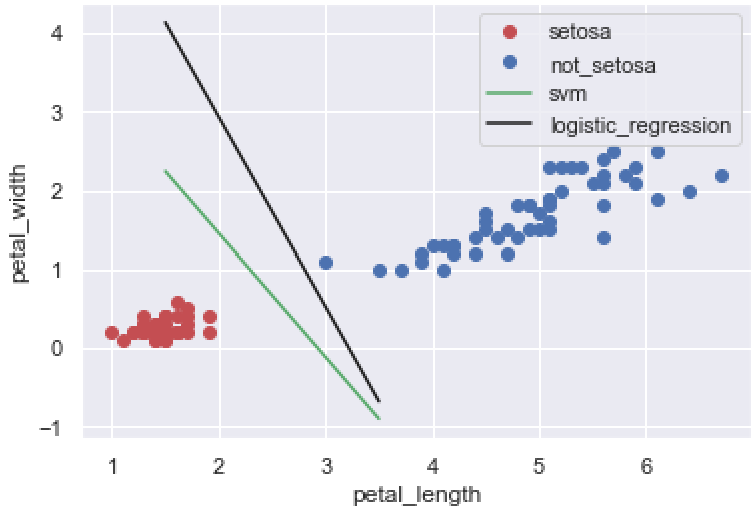
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,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

6.3. Use sklearn

In [15]:

```
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
```

In [16]:
```
sklearn_svm = SVC(kernel='linear', C=1e5) # big number to create Hard-margin SVM
sklearn_svm
```
Out[16]:
```
SVC(C=100000.0, kernel='linear')
```
In [17]:
```
sklearn_svm.fit(X, y)
```
Out[17]:
```
SVC(C=100000.0, kernel='linear')
```
In [18]:
```
svm_w = sklearn_svm.coef_
svm_w
```
Out[18]:
```
array([[-1.29411743, -0.82352928]])
```
In [19]:
```
svm_b = sklearn_svm.intercept_
svm_b
```
Out[19]:
```
array([3.7882347])
```
In [20]:
```
sklearn_logistic_regression = LogisticRegression()
sklearn_logistic_regression
```
Out[20]:
```
LogisticRegression()
```
In [21]:
```
sklearn_logistic_regression.fit(X, y)
```
Out[21]:
```
LogisticRegression()
```
In [22]:
```
lr_w = sklearn_logistic_regression.coef_
lr_w
```
Out[22]:
```
array([[-2.35093015, -0.97631903]])
```
In [23]:
```
lr_b = sklearn_logistic_regression.intercept_
lr_b
```
Out[23]:
```
array([7.56643687])
```
In [24]:
```
def display(X, y, w_1, b_1, w_2, b_2):
    plt.plot(X[:50, 0], X[:50, 1], 'ro', label='setosa')
    plt.plot(X[50:, 0], X[50:, 1], 'bo', label='not_setosa')

    line_types = ['g-', 'k-']
    model_name = ['svm', 'logistic_regression']
    reg_x = np.linspace(1.5, 3.5, 2)
    for idx, (w, b) in enumerate([[w_1, b_1], [w_2, b_2]]):

        reg_y = \
        (- b[0] / w[0][1]) + \
        (- w[0][0] / w[0][1]) * reg_x

        plt.plot(reg_x, reg_y, line_types[idx], label=model_name[idx])

    plt.xlabel('petal_length')
    plt.ylabel('petal_width')
    plt.legend()
    plt.show()
```
In [25]:
```
display(X, y, svm_w, svm_b, lr_w, lr_b)
```



In [26]:
```
lr_y_pred = sklearn_logistic_regression.predict(X)
lr_y_pred
```

Out[26]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False])
```

In [27]:
```
svm_y_pred = sklearn_svm.predict(X)
svm_y_pred
```

Out[27]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False])
```

In [28]:
```
print(classification_report(lr_y_pred, y))
```

```
              precision    recall  f1-score   support

       False       1.00      1.00      1.00        50
        True       1.00      1.00      1.00        50

    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
weighted avg       1.00      1.00      1.00       100
```

In [29]:
```
print(classification_report(svm_y_pred, y))
```

```
              precision    recall  f1-score   support

       False       1.00      1.00      1.00        50
        True       1.00      1.00      1.00        50
```

```
    accuracy                           1.00       100
   macro avg       1.00      1.00      1.00       100
weighted avg       1.00      1.00      1.00       100
```

6.4. Almost linearly separable dataset problem

In [30]:
```
iris_df
```

Out[30]:

	petal_length	petal_width	species	is_setosa
0	1.4	0.2	setosa	True
1	1.4	0.2	setosa	True
2	1.3	0.2	setosa	True
3	1.5	0.2	setosa	True
4	1.4	0.2	setosa	True
...	...	...	...	...
145	5.2	2.3	virginica	False
146	5.0	1.9	virginica	False
147	5.2	2.0	virginica	False
148	5.4	2.3	virginica	False
149	5.1	1.8	virginica	False

150 rows × 4 columns

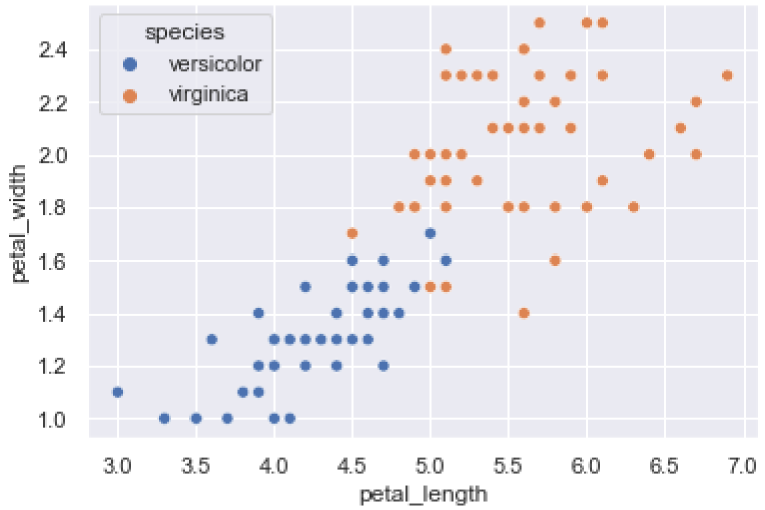
In [31]:
```
vv_df = iris_df[iris_df.species != 'setosa']
vv_df
```

Out[31]:

	petal_length	petal_width	species	is_setosa
50	4.7	1.4	versicolor	False
51	4.5	1.5	versicolor	False
52	4.9	1.5	versicolor	False
53	4.0	1.3	versicolor	False
54	4.6	1.5	versicolor	False
...	...	...	...	...
145	5.2	2.3	virginica	False
146	5.0	1.9	virginica	False
147	5.2	2.0	virginica	False
148	5.4	2.3	virginica	False
149	5.1	1.8	virginica	False

100 rows × 4 columns

In [32]:
```
sns.scatterplot(data=vv_df, x='petal_length', y='petal_width', hue='species')
plt.show()
```



In [33]:
```
X = vv_df[['petal_length', 'petal_width']]
X.shape
```

Out[33]: (100, 2)

In [34]:
```
X
```

Out[34]:

	petal_length	petal_width
50	4.7	1.4
51	4.5	1.5
52	4.9	1.5
53	4.0	1.3
54	4.6	1.5
...	...	...
145	5.2	2.3
146	5.0	1.9
147	5.2	2.0
148	5.4	2.3
149	5.1	1.8

100 rows × 2 columns

In [35]:
```
y = vv_df['species']
y.shape
```

Out[35]: (100,)

In [36]:
```
y
```

Out[36]:
```
50     versicolor
51     versicolor
52     versicolor
53     versicolor
54     versicolor
          ...
145     virginica
146     virginica
```

```
147     virginica
148     virginica
149     virginica
Name: species, Length: 100, dtype: object
```

6.5. Use sklearn

In [37]:
```
sklearn_svm_c1 = SVC(kernel='linear', C=1)
sklearn_svm_c1
```

Out[37]:
```
SVC(C=1, kernel='linear')
```

In [38]:
```
sklearn_svm_c1.fit(X, y)
```

Out[38]:
```
SVC(C=1, kernel='linear')
```

In [39]:
```
sklearn_svm_c1.coef_
```

Out[39]:
```
array([[2.1829247 , 2.25365588]])
```

In [40]:
```
sklearn_svm_c1.intercept_
```

Out[40]:
```
array([-14.41486828])
```

In [41]:
```
svm_c1_pred = sklearn_svm_c1.predict(X)
svm_c1_pred
```

Out[41]:
```
array(['versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'virginica', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'virginica',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'virginica', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'versicolor', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'versicolor', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica'], dtype=object)
```

In [42]:
```
sklearn_svm_c0p1 = SVC(kernel='linear', C=0.1)
sklearn_svm_c0p1
```

Out[42]:
```
SVC(C=0.1, kernel='linear')
```

In [43]:
```
sklearn_svm_c0p1.fit(X, y)
```

Out[43]:
```
SVC(C=0.1, kernel='linear')
```

In [44]:
```
sklearn_svm_c0p1.coef_
```

Out[44]: array([[1.19016375, 0.95213016]])

In [45]:
```
sklearn_svm_c0p1.intercept_
```

Out[45]: array([-7.4265328])

In [46]:
```
svm_c0p1_pred = sklearn_svm_c0p1.predict(X)
svm_c0p1_pred
```

Out[46]:
```
array(['versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'virginica', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'virginica',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'virginica', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'versicolor', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'versicolor', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica'], dtype=object)
```

In [47]:
```
sklearn_lr = LogisticRegression()
sklearn_lr
```

Out[47]: LogisticRegression()

In [48]:
```
sklearn_lr.fit(X, y)
```

Out[48]: LogisticRegression()

In [49]:
```
sklearn_lr.coef_
```

Out[49]: array([[2.77743512, 2.38548149]])

In [50]:
```
sklearn_lr.intercept_
```

Out[50]: array([-17.5471049])

In [51]:
```
lr_pred = sklearn_lr.predict(X)
lr_pred
```

Out[51]:
```
array(['versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'virginica', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'virginica',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'virginica', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
```

```
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'versicolor', 'versicolor', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'versicolor', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'versicolor', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica', 'virginica', 'virginica', 'virginica',
       'virginica', 'virginica'], dtype=object)
```

In [52]:

```
print(classification_report(lr_pred, y))
```

```
              precision    recall  f1-score   support

  versicolor       0.94      0.96      0.95        49
   virginica       0.96      0.94      0.95        51

    accuracy                           0.95       100
   macro avg       0.95      0.95      0.95       100
weighted avg       0.95      0.95      0.95       100
```

In [53]:

```
print(classification_report(svm_c1_pred, y))
```

```
              precision    recall  f1-score   support

  versicolor       0.94      0.96      0.95        49
   virginica       0.96      0.94      0.95        51

    accuracy                           0.95       100
   macro avg       0.95      0.95      0.95       100
weighted avg       0.95      0.95      0.95       100
```

In [54]:

```
print(classification_report(svm_c0p1_pred, y))
```

```
              precision    recall  f1-score   support

  versicolor       0.94      0.96      0.95        49
   virginica       0.96      0.94      0.95        51

    accuracy                           0.95       100
   macro avg       0.95      0.95      0.95       100
weighted avg       0.95      0.95      0.95       100
```

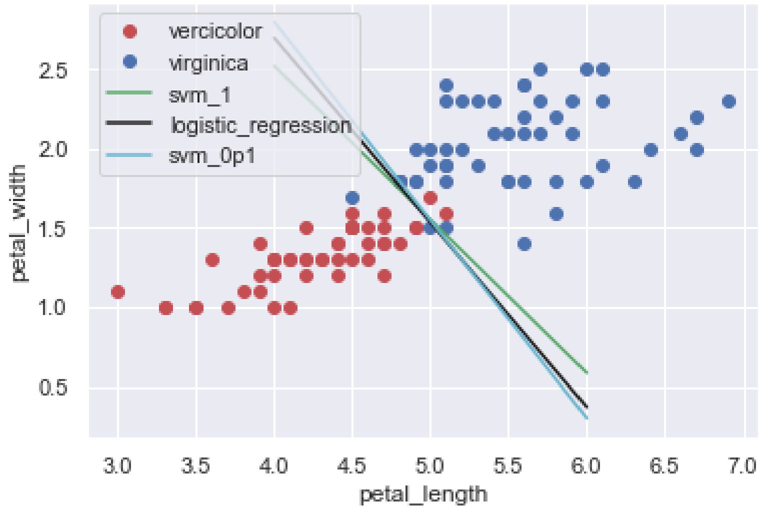
In [55]:

```
def display(X, y, w_1, b_1, w_2, b_2, w_3, b_3):
    plt.plot(X[:50, 0], X[:50, 1], 'ro', label='vercicolor')
    plt.plot(X[50:, 0], X[50:, 1], 'bo', label='virginica')

    line_types = ['g-', 'k-', 'c-']
    model_name = ['svm_1', 'logistic_regression', 'svm_0p1']
    reg_x = np.linspace(4, 6, 2)
    for idx, (w, b) in enumerate([[w_1, b_1], [w_2, b_2], [w_3, b_3]]):

        reg_y = \
        (- b[0] / w[0][1]) + \
        (- w[0][0] / w[0][1]) * reg_x

        plt.plot(reg_x, reg_y, line_types[idx], label=model_name[idx])

    plt.xlabel('petal_length')
    plt.ylabel('petal_width')
```

```
    plt.legend()
    plt.show()
```

In [56]:

```
display(
    np.array(X),
    np.array(y),
    sklearn_svm_c1.coef_,
    sklearn_svm_c1.intercept_,
    sklearn_lr.coef_,
    sklearn_lr.intercept_,
    sklearn_svm_c0p1.coef_,
    sklearn_svm_c0p1.intercept_
)
```



In [ ]: