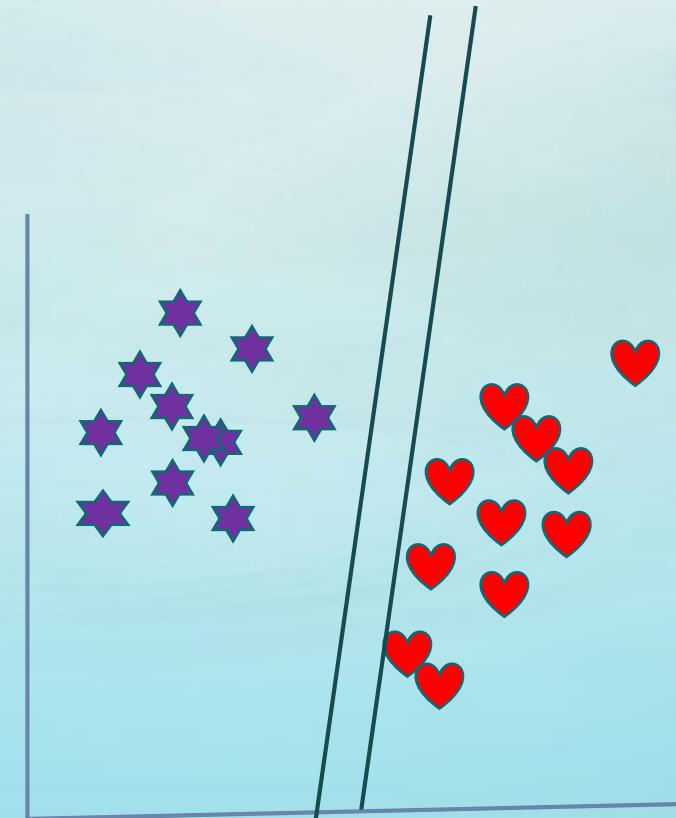
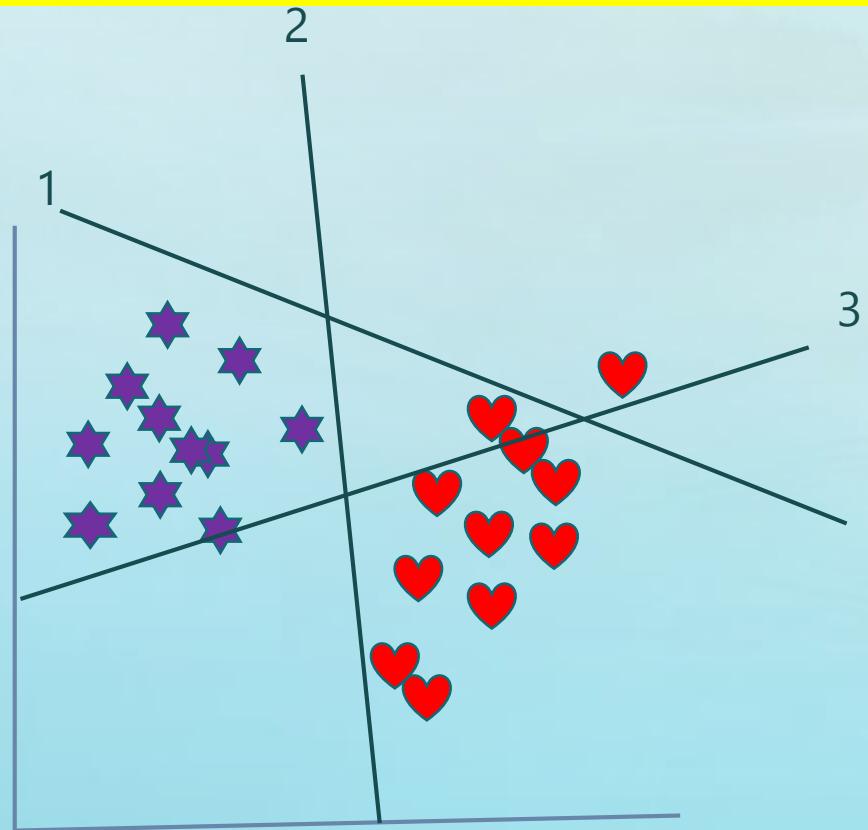


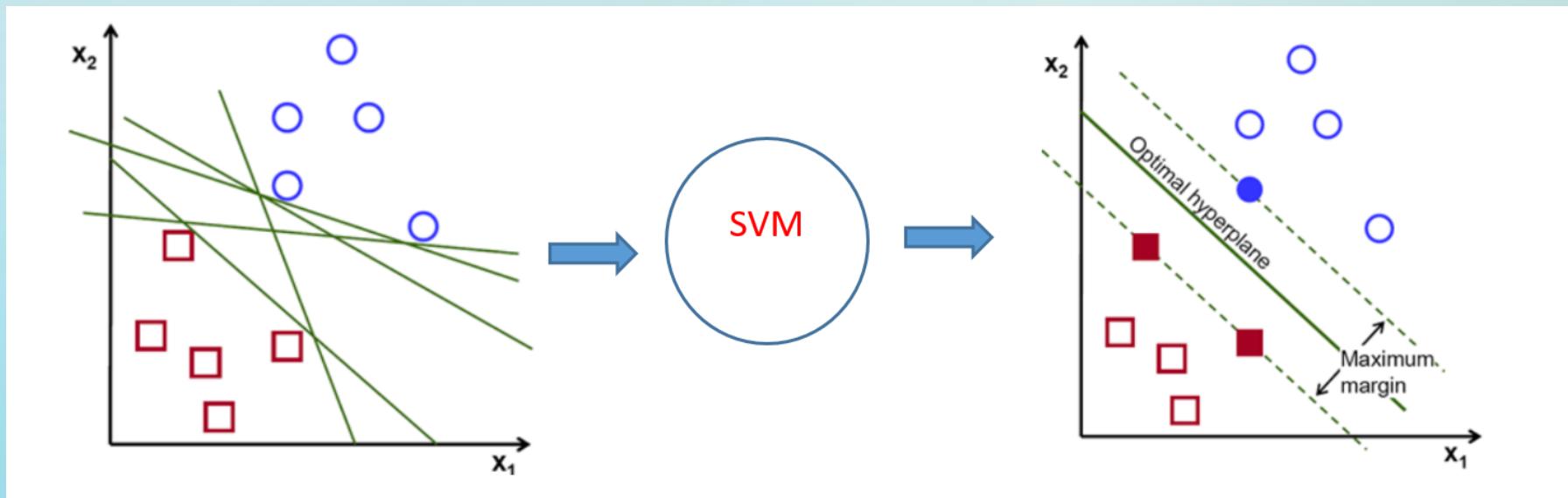
Support Vector Machines

Associate Prof. Nuanwan Soonthornphisaj

Which line is the best separation for 2 class?



Output of SVM

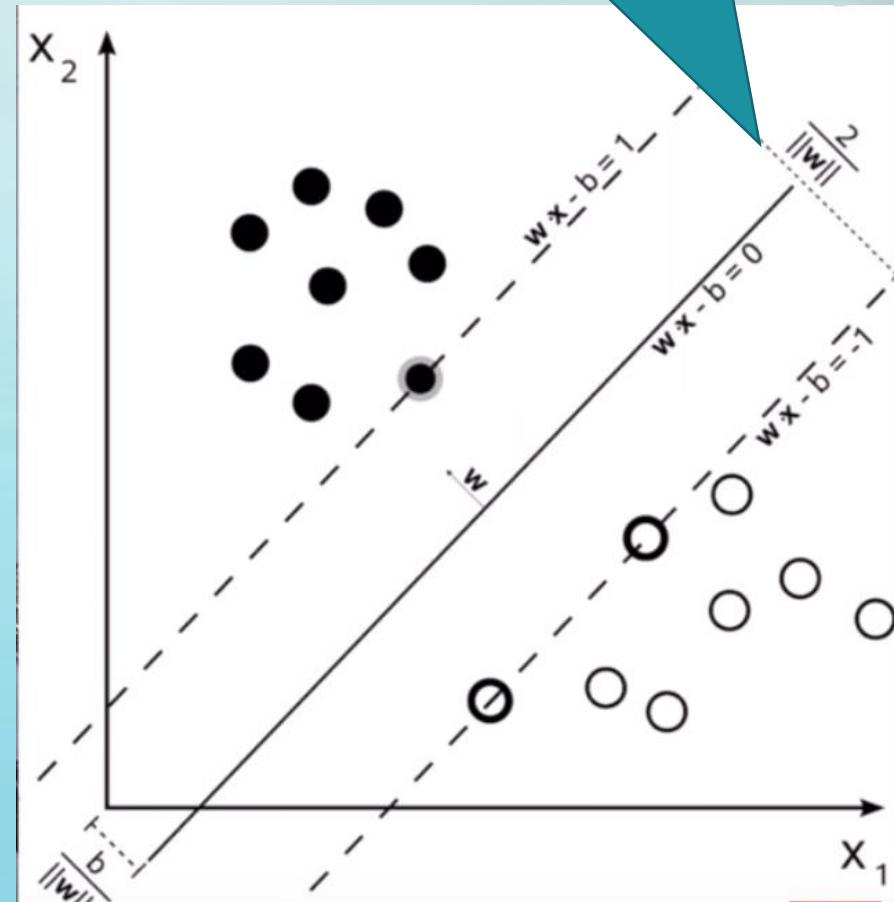


Introduction of SVM

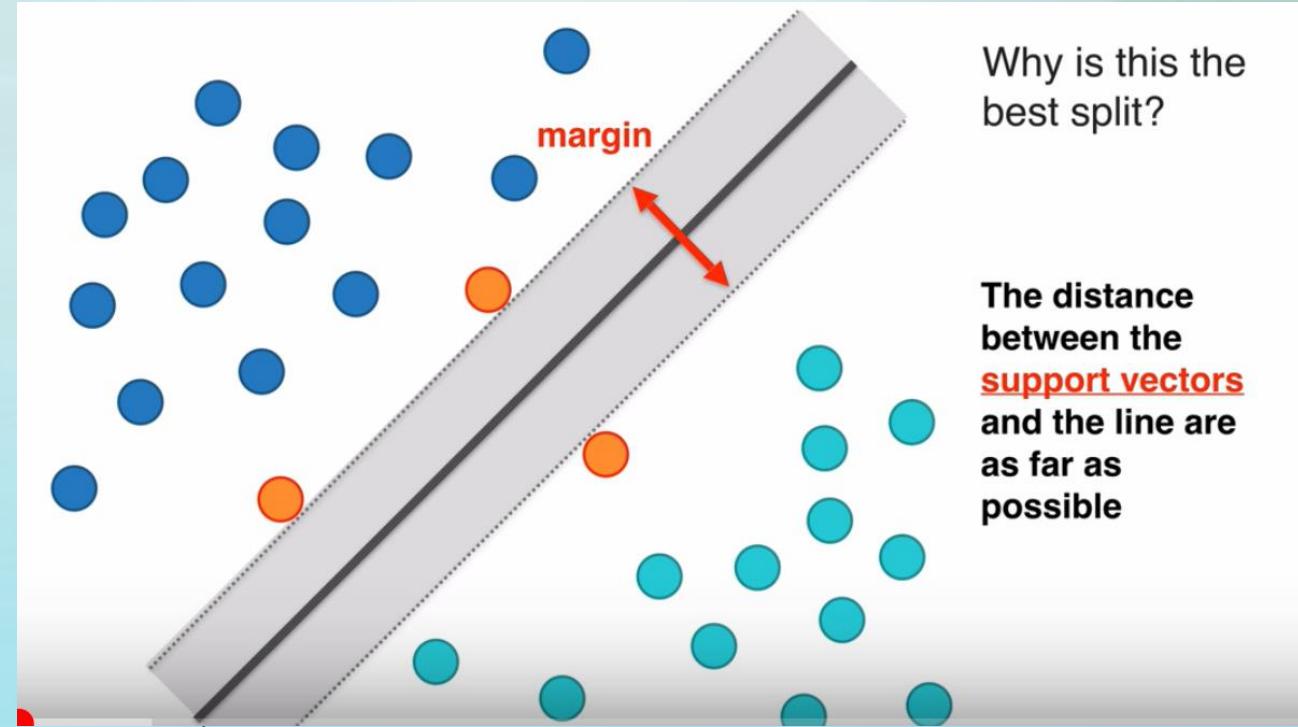
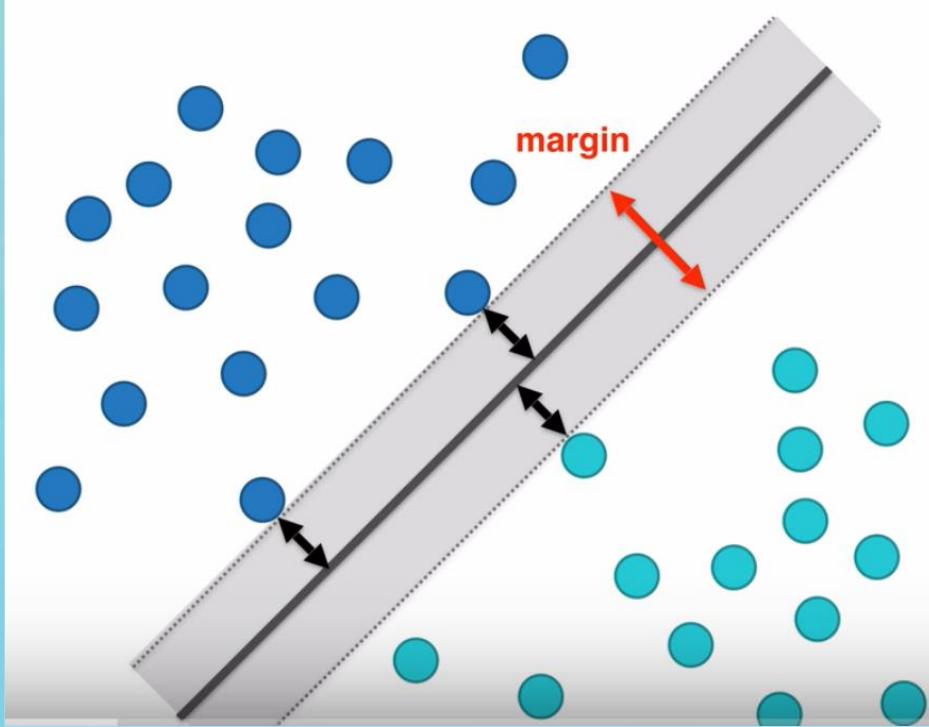
Support Vector Machines is a supervised learning algorithm that analyses data for the classification or regression

- SVM create the hyperplane or a set of hyperplane in high dimensional space.
- A good separation of hyperplane achieves by the hyperplane that has **the largest distance** to the nearest training set of any class

The margin is the maximum distance from the line to the nearest points



The best split



Why is this the
best split?

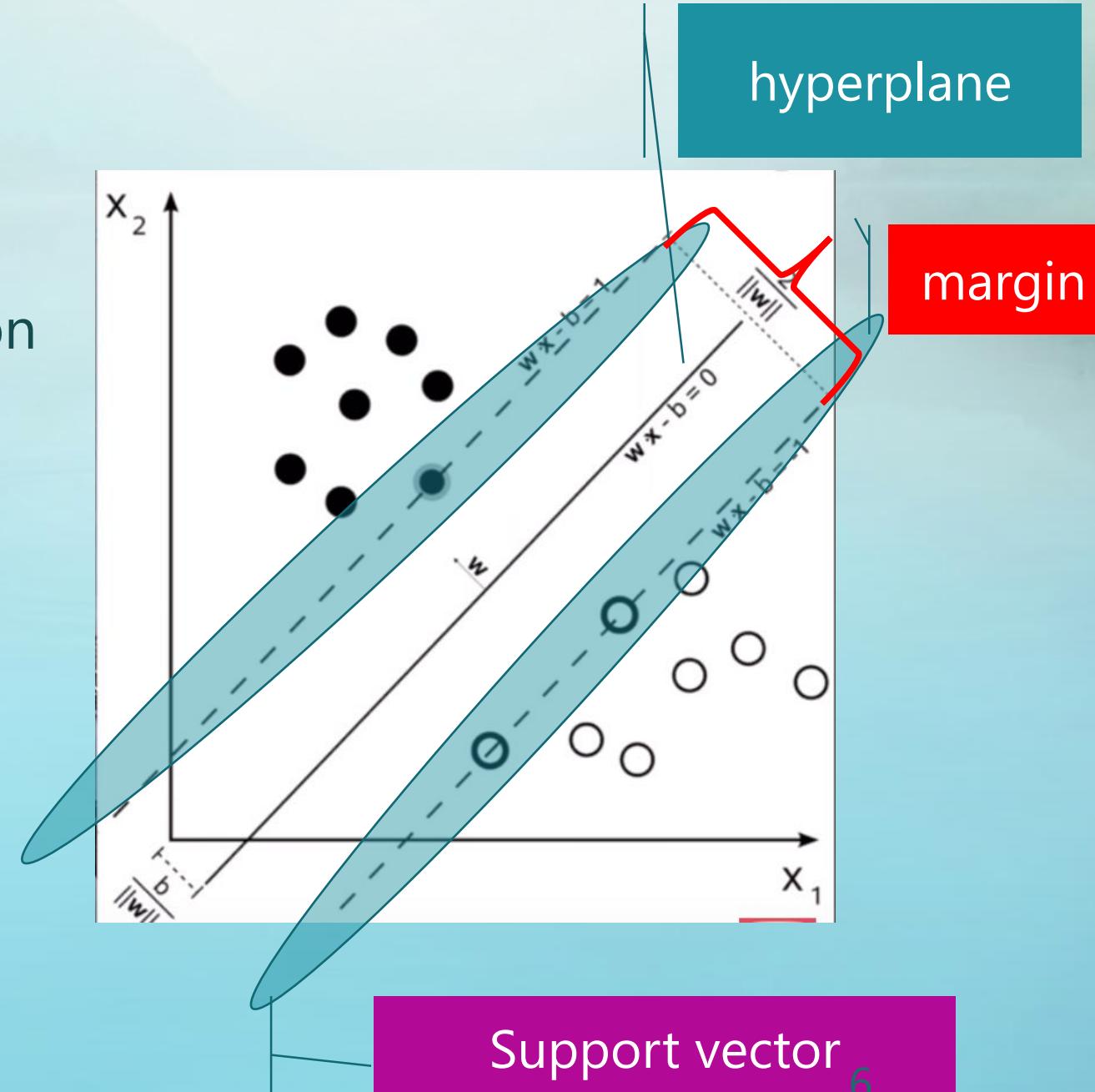
The distance
between the
support vectors
and the line are
as far as
possible

This **hyperplane**
best splits the data
because it is as far
as possible from
these **support**
vectors

hyperplane

Support Vectors

- We need to find support vectors that help the optimum separation of hyperplane
- These two support vectors must have maximum margin
- No instance is allow within the area = Hard margin



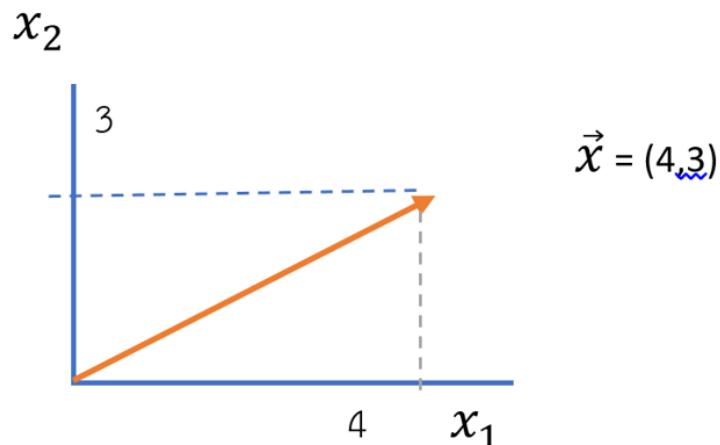
How to maximize the margin?

- It is the constrained optimization problem
- The way to solve this problem is Lagrange Multipliers

ทฤษฎี Linear Algebra ที่เกี่ยวข้อง

- Vector

เวกเตอร์ประกอบด้วยขนาดและทิศทาง เราสามารถแทนจุดในรูปแบบสองมิติได้ด้วยเวกเตอร์ที่สร้างจากจุดกำเนิด (Origin) ไปยังจุดนั้น จากภาพแสดงเวกเตอร์ X เขียนแทนด้วยสัญลักษณ์ \vec{x}

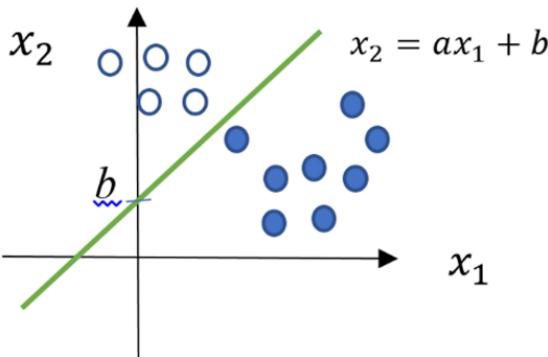


ความยาวของเวกเตอร์ \vec{x} แทนด้วยสัญลักษณ์ $\|x\|$ มีค่าเท่ากับ $\sqrt{x_1^2 + x_2^2}$ ดังนั้น เวกเตอร์ \vec{x} จึงมีความยาวเท่ากับ $\sqrt{4^2 + 3^2} = 5$

การหาความสัมพันธ์ระหว่างเวกเตอร์ด้วย dot product

กำหนดให้ $\vec{a} = (a_1, a_2, \dots, a_n)$ และ $\vec{b} = (b_1, b_2, \dots, b_n)$ ของเวกเตอร์ a และ b
แสดงได้ดังสมการที่ 8.1

$$\vec{a}\vec{b} = \sum_{i=0}^n a_i b_i \quad (8.1)$$



ภาพที่ 8.3 การจำแนกคลาสด้วยสมการเส้นตรง

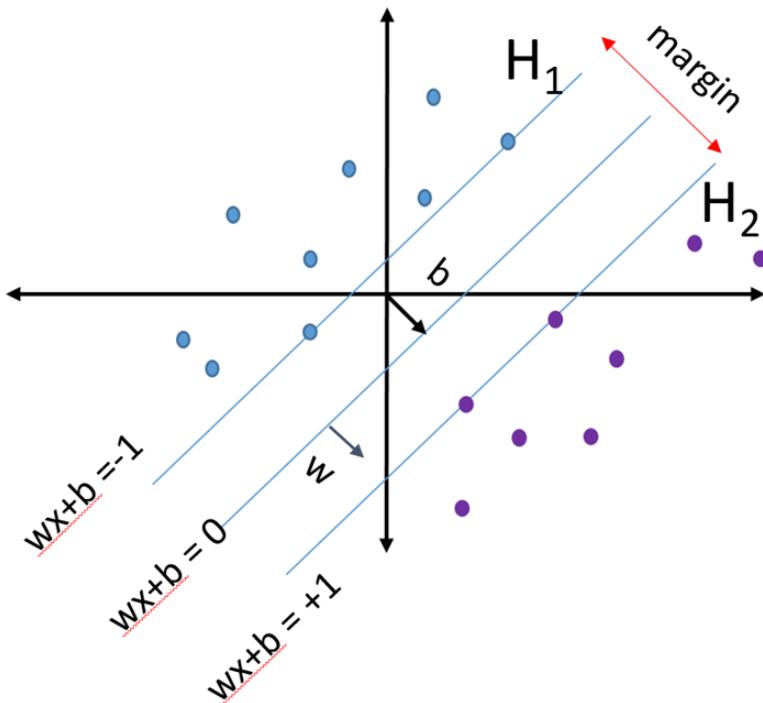
จากภาพแสดงเส้นตรงที่แบ่งแยกข้อมูลฝึกซ้อมี 2 ฟีเจอร์คือ x_1 และ x_2

เส้นตรงนี้สามารถเขียนได้ว่า $x_2 = ax_1 + b$ หรือ $ax_1 + b - x_2 = 0$

ถ้ากำหนดให้ $w = (a, -1)$ ดังนั้นในกรณีที่ข้อมูลฝึกมีหลายมิติ เราสามารถเขียนสมการของระนาบหลายมิติโดยใช้การดอทโปรดักส์ ได้ดังสมการที่ 8.2

$$W \cdot X + b = 0 \quad (8.2)$$

การค้นหาชั้พพอร์ตเวกเตอร์



สมมติว่าชุดข้อมูลฝึกคือเซต \mathbf{X} ที่ประกอบไปด้วยตัวอย่าง x_i จำนวน n ตัวอย่าง แต่ละตัวอย่าง มีจำนวนฟีเจอร์เท่ากับ D และมีป้ายคลาสคือ y_i โดยที่ $y_i \in \{-1,1\}$ ดังนั้นตัวอย่างสามารถเขียนได้ดังนี้

$$\{x_i, y_i\} \text{ โดยที่ } i = 1..n, \text{ และ } x \in R^D$$

ดังนั้นระนาบหลายมิติที่แบ่งแยกคลาสของข้อมูลฝึก สามารถเขียนในรูปสมการที่ 8.4 และ 8.4 ตามลำดับ

$$w \cdot x_i + b \geq 1 \text{ กรณีที่ } y_i = +1 \quad (8.3)$$

$$w \cdot x_i + b \leq -1 \text{ กรณีที่ } y_i = -1 \quad (8.4)$$

จากภาพที่ 8.4 จะพบว่าระนาบจำนวนสองระนาบที่สร้างจากชั้พพอร์ตเวกเตอร์ของตัวอย่างในคลาส +1 และ -1 เขียนแทนด้วยสมการดังนี้

ระนาบ H1

$$w \cdot x_i + b = -1 \quad (8.5)$$

ระนาบ H2

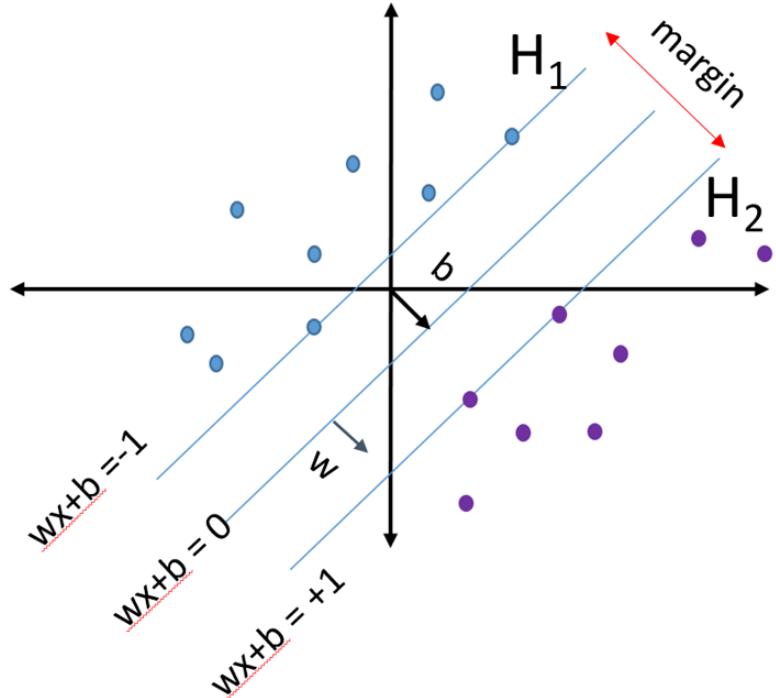
$$w \cdot x_i + b = +1 \quad (8.6)$$

ระนาบ H1

ระนาบ H2

$$w \cdot x_i + b = -1 \quad (8.5)$$

$$w \cdot x_i + b = +1 \quad (8.6)$$



ระยะห่างระหว่างระนาบ H_1 และ จุดกำเนิดคือ $\frac{(-1-b)}{|w|}$

ระยะห่างระหว่างระนาบ H_2 และ จุดกำเนิดคือ $\frac{(1-b)}{|w|}$

ดังนั้นระยะห่างระหว่างระนาบ H_1 และระนาบ H_2 เขียนได้ด้วยสมการ 8.7 และ 8.8

$$M = \frac{(1-b)}{|w|} - \frac{(-1-b)}{|w|} \quad (8.7)$$

$$M = \frac{2}{|w|} \quad (8.8)$$

จะได้ว่าครึ่งหนึ่งของระยะห่าง (M) เป็นไปตามสมการที่ 8.9

$$Margin = \frac{1}{|w|} \quad (8.9)$$

ดังนั้นขั้นตอนวิธีซึ่งพอร์ตเวกเตอร์จะมีเป้าหมายในการ maximize ค่า margin ซึ่งคือการ minimize ค่า $\|w\|$ นั้นเอง เขียนได้ว่า $\min\|w\|$ เราสามารถหาค่าเหมาะสมที่สุดด้วยวิธี Dual Optimization (Dual Optimization) ดังนั้น สมการจึงเปลี่ยนเป็น

$$\min \frac{\|w\|^2}{2} \text{ โดยที่ } y_i(w \cdot x_i + b) - 1 \geq 0 \text{ for } i = 1..n$$

จากทฤษฎีแคลคูลัส เราสามารถหาค่าต่ำสุดของฟังก์ชันได้ด้วยการหาอนุพันธ์ อย่างก็ตามเมื่อ ฟังก์ชันที่เราคำนวนหาค่าต่ำที่สุดมีเงื่อนไข (Constraint) ระหว่างตัวแปร เราจำเป็นต้องใช้ตัวคูณ ลากรองจ์ (Lagrange multiplier) หรือ λ กำกับที่เงื่อนไข เพื่อคำนวนค่าเหมาะสมที่สุด (optimization) จากปัญหาการหาระนาบหลายมิติเพื่อจำแนกข้อมูล เราต้องการ minimize สมการที่ 8.10

$$l = \frac{\|w\|^2}{2} - \sum_{i=1}^n \lambda_i (y_i(w \cdot x_i + b) - 1)$$

ตัวคูณลากรองจ์

เงื่อนไข

$$l = \frac{\|w\|^2}{2} - \sum_{i=1}^n \lambda_i (y_i(w \cdot x_i + b) - \sum_{i=1}^n \lambda_i)$$

หาอนุพันธ์ของ l

$$\frac{\partial l}{\partial w} = w - \sum_{i=1}^n \lambda_i y_i x_i = 0$$

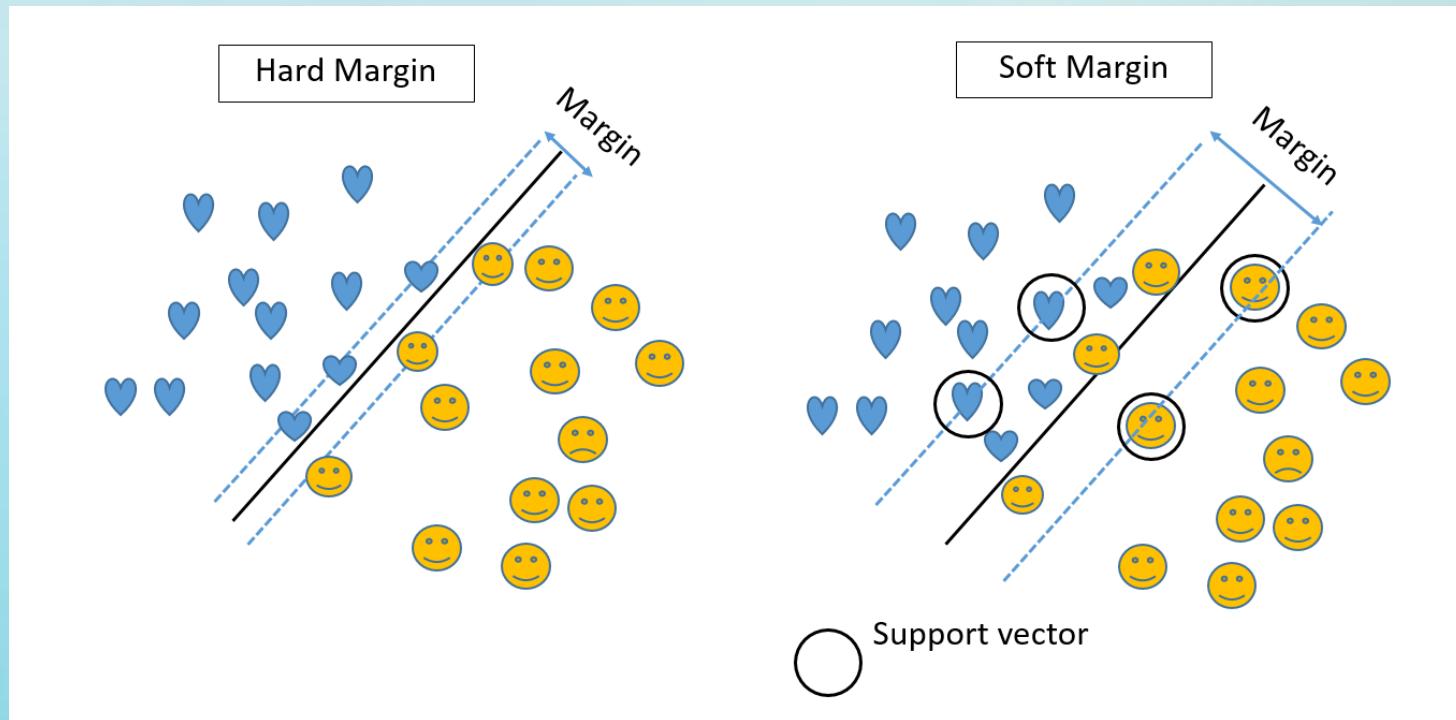
$$\frac{\partial l}{\partial \lambda} = \sum_{i=1}^n y_i (w \cdot x_i + b) - 1 = 0$$

$$\frac{\partial l}{\partial b} = \sum_{i=1}^n \lambda_i y_i = 0$$

หาค่า λ, w
เมื่อทราบค่า w จะหาค่า b
ทำให้หา support vector ได้

Hard margin ทำให้เกิดปัญหา Overfitting

วิธีแก้ = ยอมให้ตัวอย่างฝึกอยู่ภายในการ์จิน (training error) →
ลดความกว้างของมาร์จิน → Soft margin



การคำนวณเพื่อหาซอฟต์มาร์จิน

$$w \cdot x_i + b \geq 1 - \xi_i \text{ กรณีที่ } y_i = +1 \quad (8.30)$$

$$w \cdot x_i + b \leq -1 + \xi_i \text{ กรณีที่ } y_i = -1 \quad (8.31)$$

เราสามารถรวม 2 สมการข้างต้นได้เป็น

$$y_i(w \cdot x_i + b) - 1 + \xi_i \geq 0 \text{ กรณีที่ } y_i = +1, -1$$

$$\xi_i \geq 0 \text{ สำหรับ } i = 1..n$$

$$\sum_{i=1}^n \xi_i \geq C$$

$$0 \leq \xi_i \leq 1$$

ค่า ξ_i แสดง ตำแหน่งของตัวอย่าง i เช่น ถ้า

$0 \leq \xi_i \leq 1$ หมายความว่า ตัวอย่าง i อยู่ในบริเวณมาร์จินแต่ยังคงอยู่ในด้านที่ระบบจำแนก
ตัวอย่างนี้ได้ถูกต้อง

กรณีที่ $\xi_i > 1$ หมายความว่า ตัวอย่าง i อยู่ในบริเวณมาร์จินด้านตรงข้ามของระบบส่งผลให้ระบบ
จำแนกตัวอย่างนี้ผิด

ค่า C คือพารามิเตอร์ที่ถ่วงดุล (trade off) ระหว่างขนาดของมาร์จินกับจำนวนตัวอย่างในข้อมูล ฝึกที่ถูกจำแนกผิด ถ้า $C = 0$ หมายถึงการไม่ยอมให้มีการจำแนกตัวอย่างฝึก ส่งผลให้มาร์จินนี้เป็น hard margin ถ้า $C > 0$ หมายถึงมีจำนวนตัวอย่างฝึกจำนวนไม่เกิน C ที่อยู่ในบริเวณมาร์จิน (C ยิ่งมาก มาร์จินยิ่งกว้าง) การกำหนดค่า C สามารถทำได้โดยการตรวจสอบแบบไขว้ (cross validation)

การหาระนาบที่เหมาะสมที่สุดสำหรับ Soft margin

การหาระนาบที่เหมาะสมที่สุดสำหรับ Soft Margin มีลักษณะเป็นการแก้ปัญหาการหาค่าเหมาะสมที่สุดโดยเพิ่ม loss function ซึ่งเป็น พจน์ของ penalty สำหรับการจำแนกตัวอย่างฝึกที่ผิด จะได้สมการที่ 8.32

$$l = \min \frac{\|w\|^2}{2} + C \sum_{i=1}^n \xi_i \quad (8.32)$$

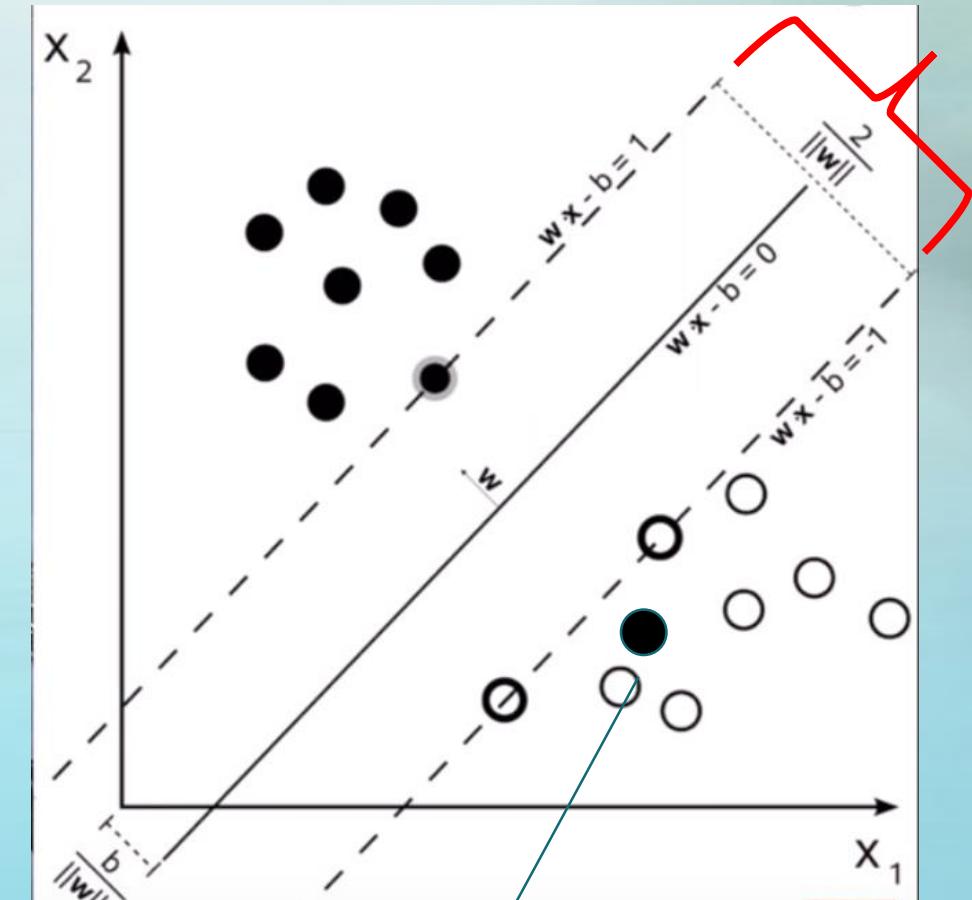
โดยที่

$$y_i(w \cdot x_i + b) - 1 + \xi_i \geq 0 \text{ และ } \xi_i \geq 0$$

เราสามารถหาค่าเหมาะสมที่สุดสำหรับปัญหานี้โดยไม่ต้องใช้ตัวคุณลักษณะ ถ้าเราแปลงสมการให้อยู่ในรูปแบบจากเงื่อนไข ส่งผลให้สามารถใช้ขั้นตอนวิธี iterative ลงตามความชัน (Gradient descent) ในการหาค่าได้ ซึ่งวิธีนี้ทำให้ชัพพอร์ตเวกเตอร์แมชีนสามารถเรียนรู้ได้บนชุดข้อมูลขนาดใหญ่

Slack variable (ξ)

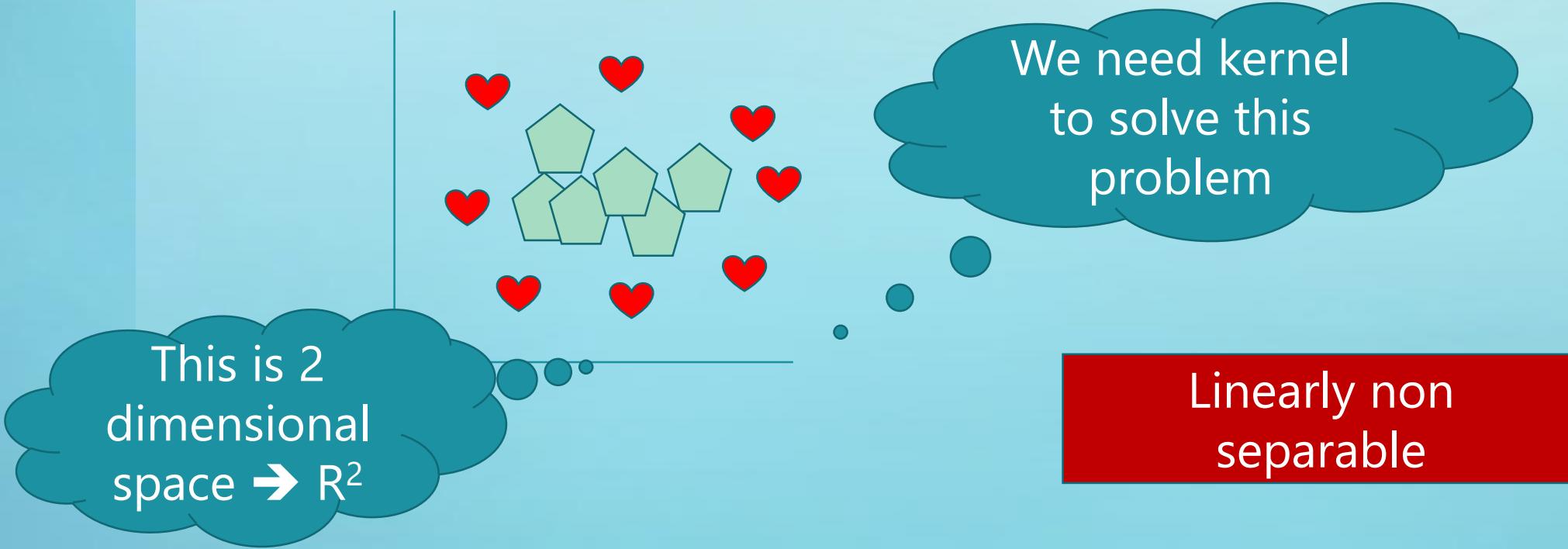
- If we allow some instances occurs inside the margin that means we allow error for the model
- The slack variable is used to control the amount of error
- Slack variable make SVMs easy to converge during the training phase since we allow training error



error

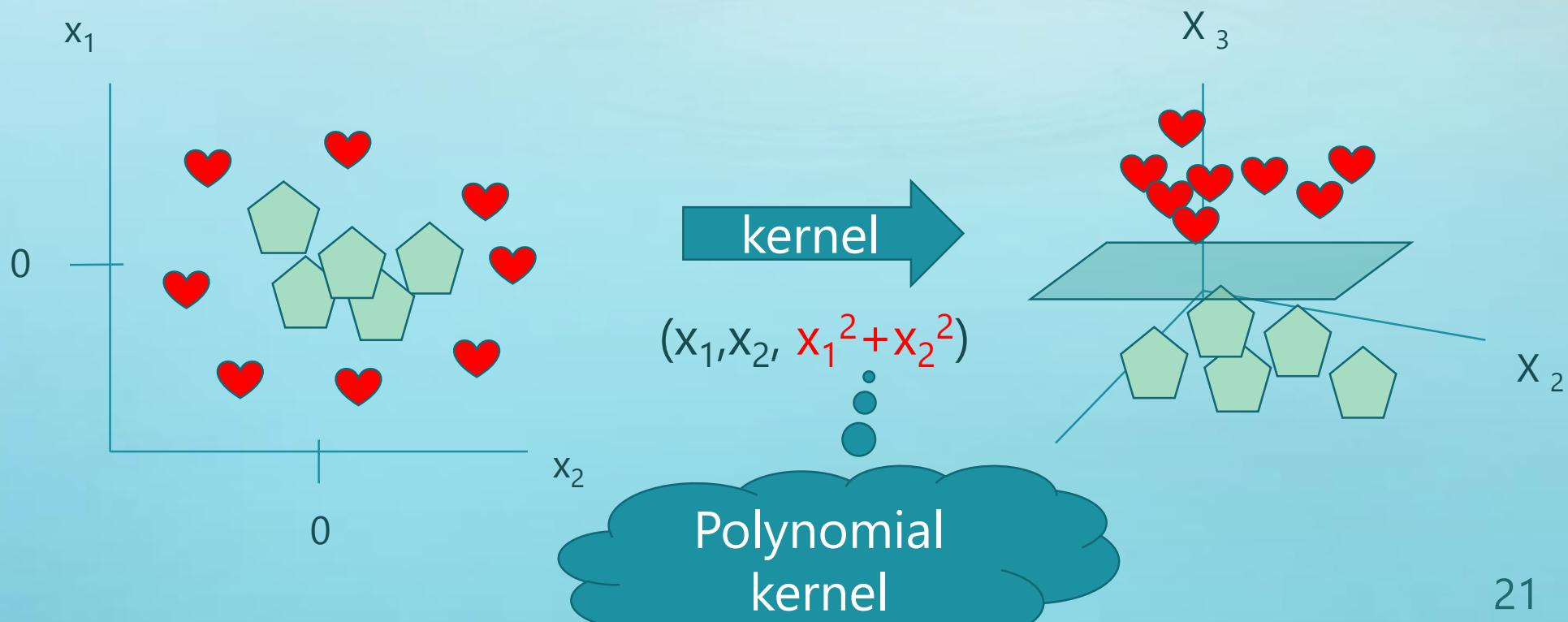
Kernel

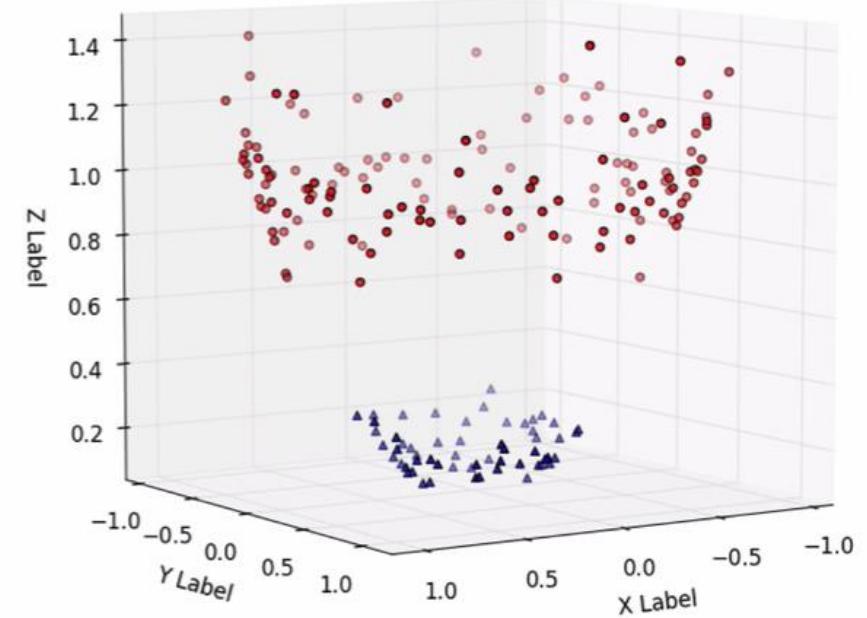
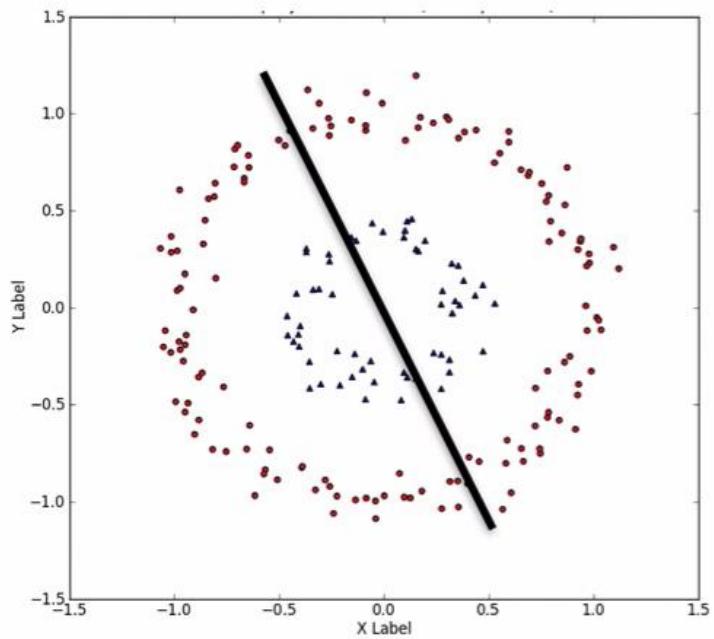
- Kernel → use for convert the feature space in high dimension with the hope that we can find the hyperplane to separate the training set



Polynomial Kernel

- If we convert the 2 dimensional space to 3 dimensional space





Kernel tricks



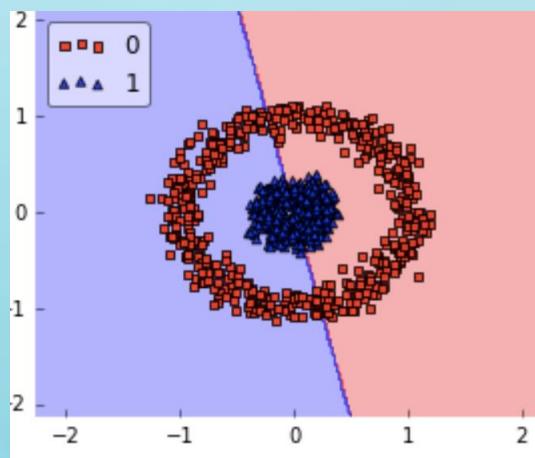
NOT separable

Linearly separable

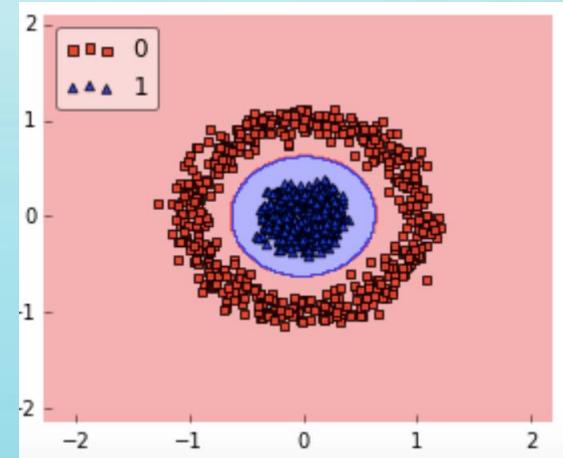
Non
linearly
separable

Non linearly separable dataset

- There are many kinds of kernel functions, we can try to find the one that fit to our training set
- Radial basis function (RBF), Sigmoid, polynomial, Gaussian



RBF kernel



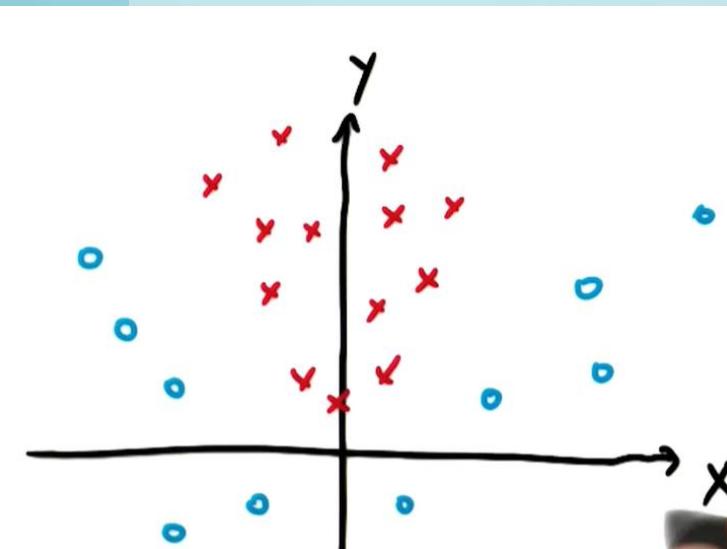
Original Code (linear)

```
# Fit basic SVC model (linear kernel)
model = svm.SVC(kernel='linear')
model.fit(sugar_butter, type_label)
```

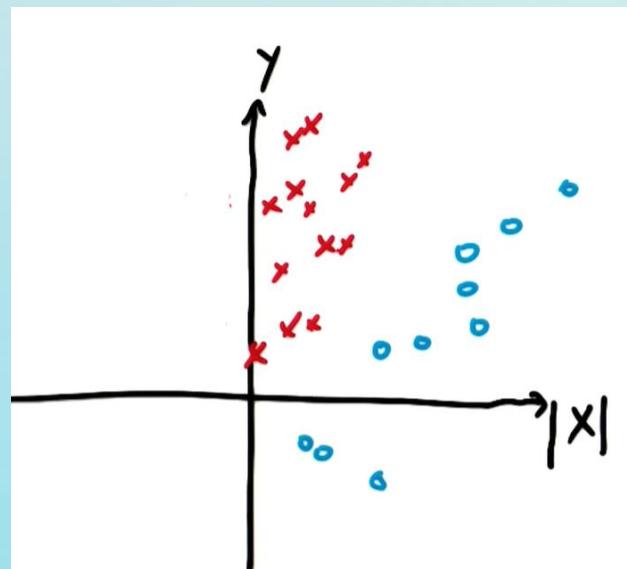
Updated Code (RBF)

```
# Fit the SVC model with radial kernel
model = svm.SVC(kernel='rbf', C=1, gamma=2**-5)
model.fit(sugar_butter, type_label)
```

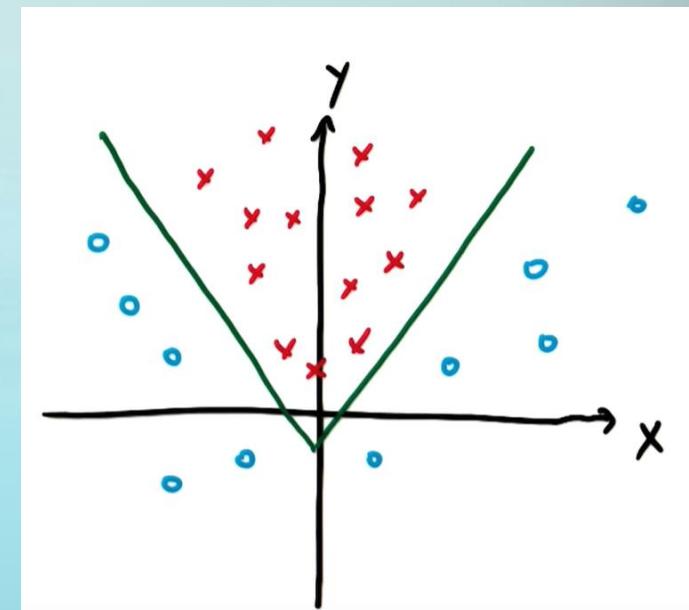
Original data



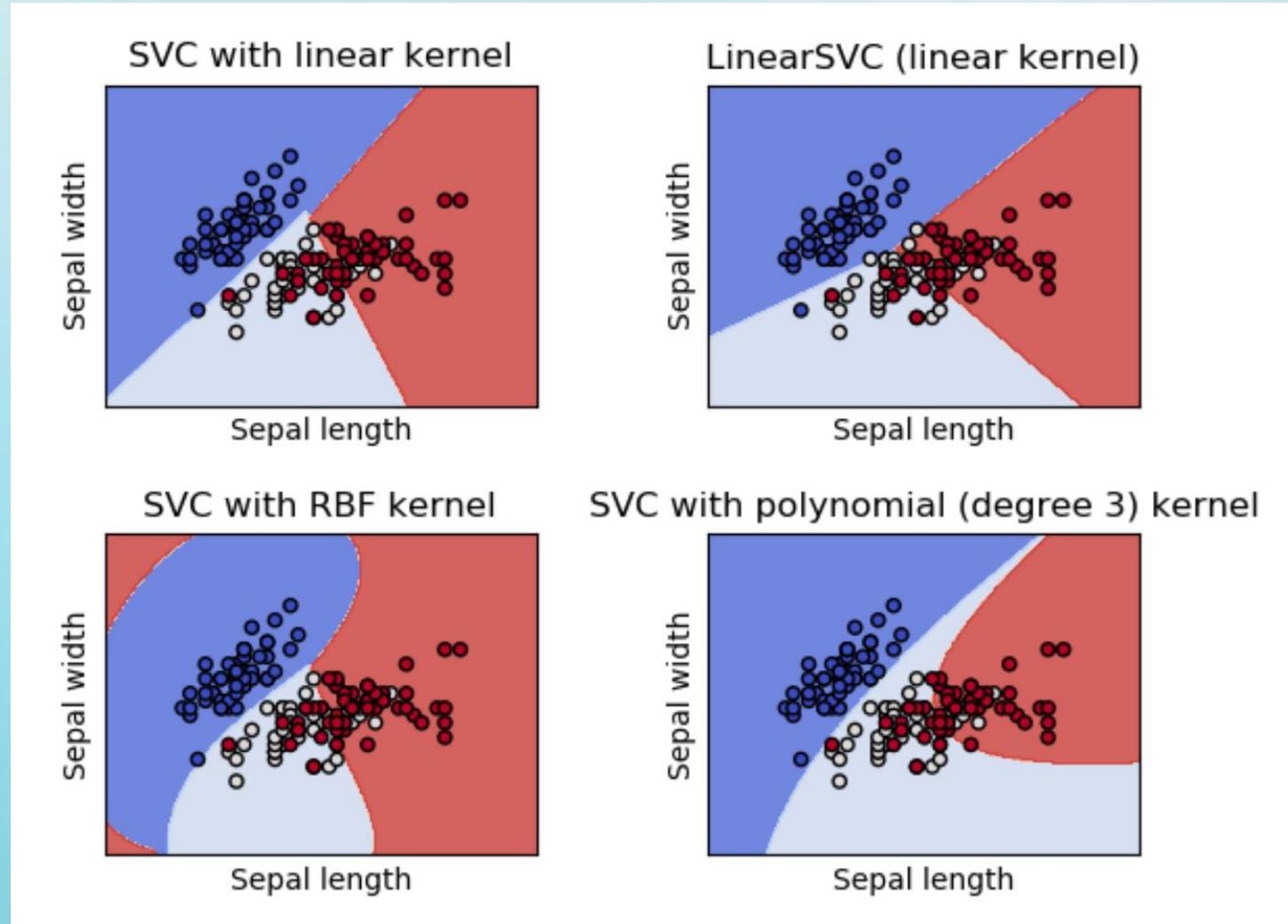
ADD FEATURE $|x|$



Decision boundary of SVM



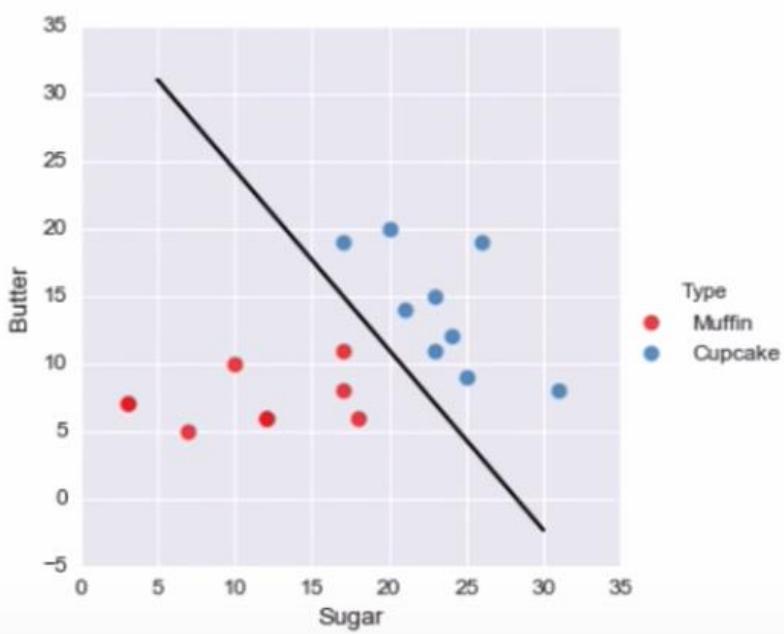
Different decision boundary using Different kernels



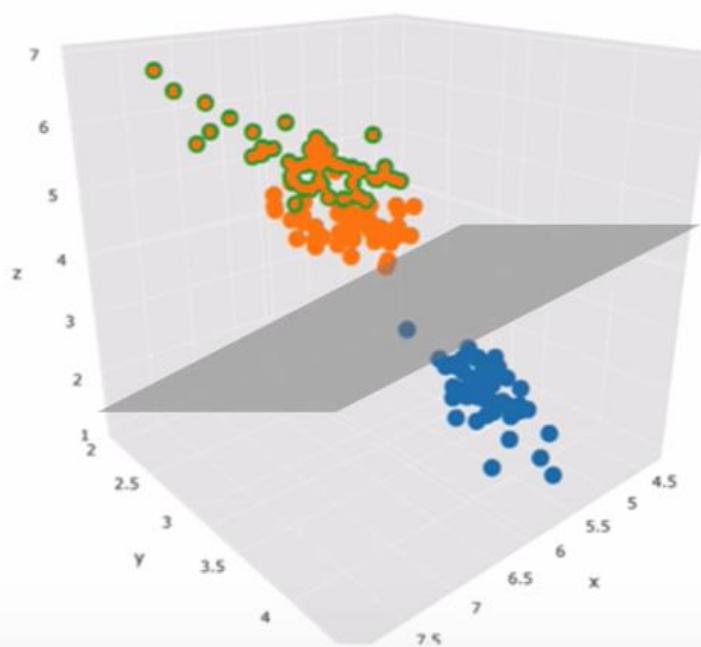
Python code

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC(gamma='scale')
>>> clf.fit(X, y)
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

2D: Separate with Line



3D: Separate with Plane



4D+: Separate with Hyperplane

Hard to Visualize

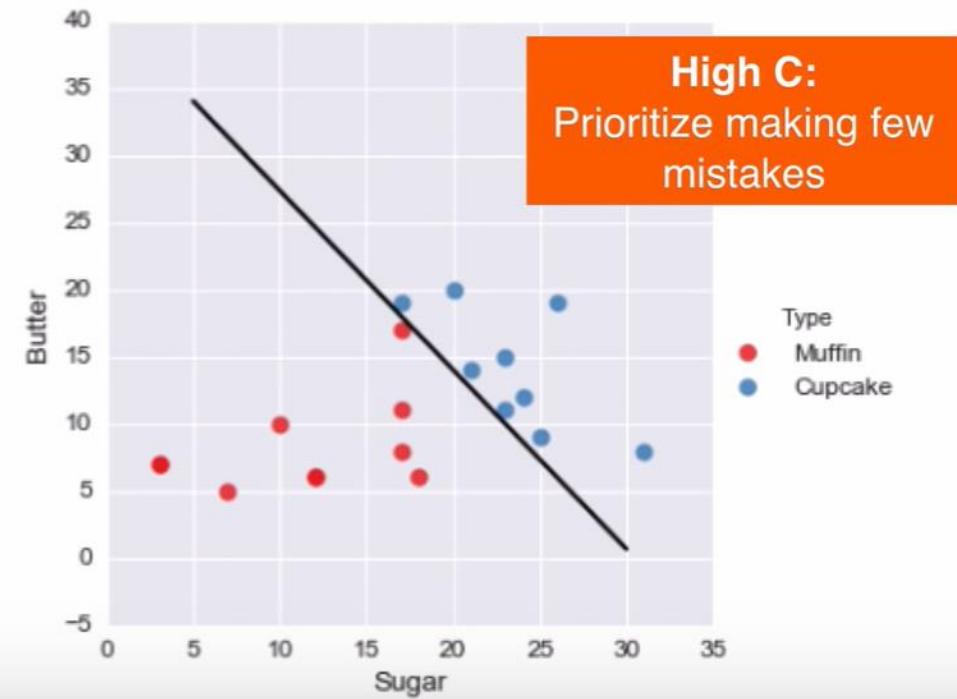
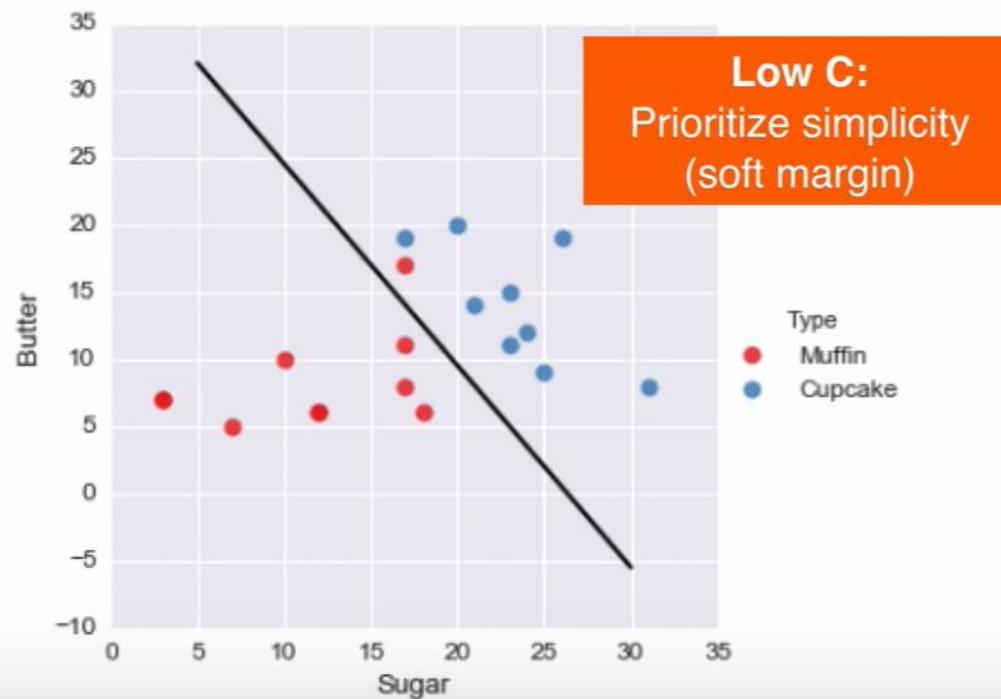
Parameter in SVM

- Kernel
- Gamma
- C parameter (to tradeoff between a smooth decision boundary and the correct classification of training data)



Default
 $C = 1.0$

Decision boundary using different C



The C parameter allows you to decide how much you want to penalize misclassified points

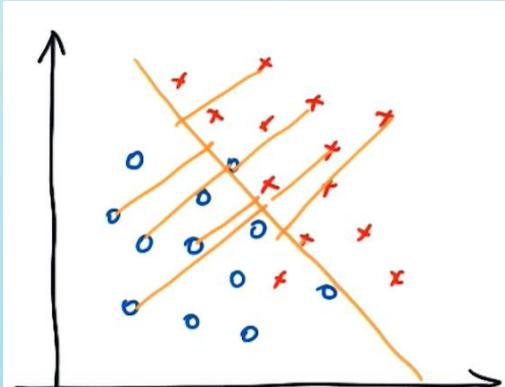
Avoid overfitting

- C
- Gamma
- kernel



Gamma parameter

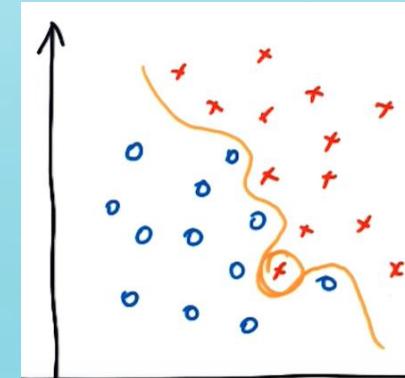
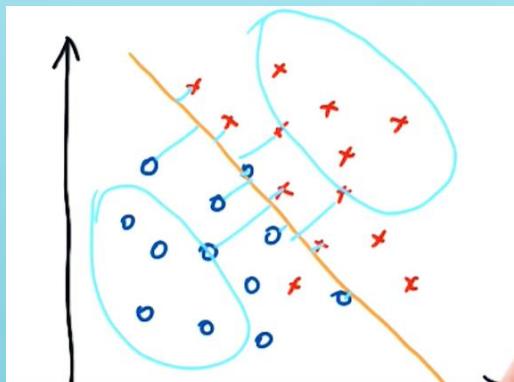
- Gamma define how far the influence of the single training example reach
- If Gamma has a low value → every point has a far reach



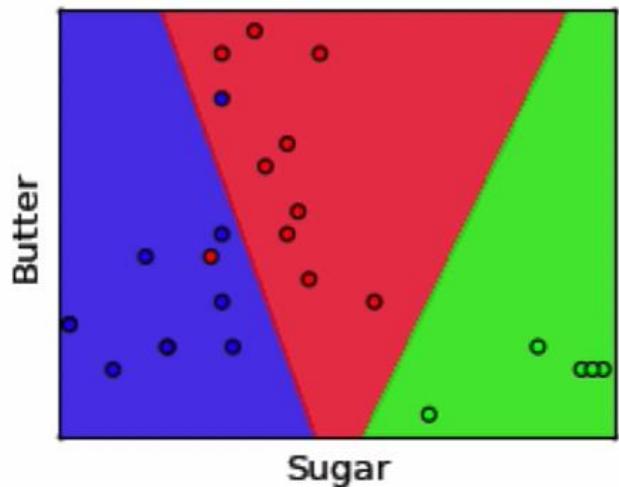
the near point has low weight

The decision boundary look more linear

- If Gamma has a high value → the decision boundary highly focus on near point has a high weight

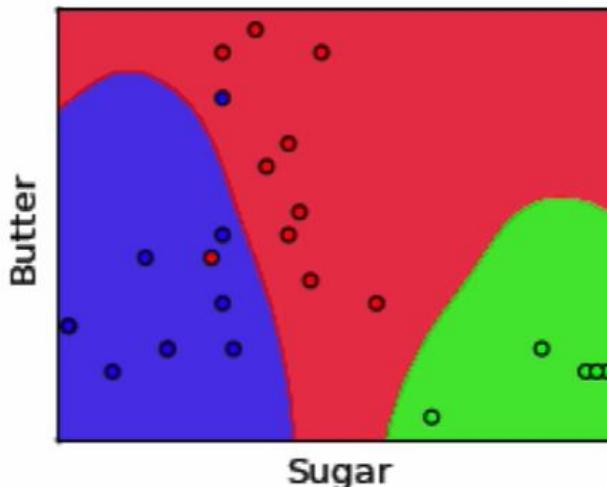


Kernel Trick: Comparison



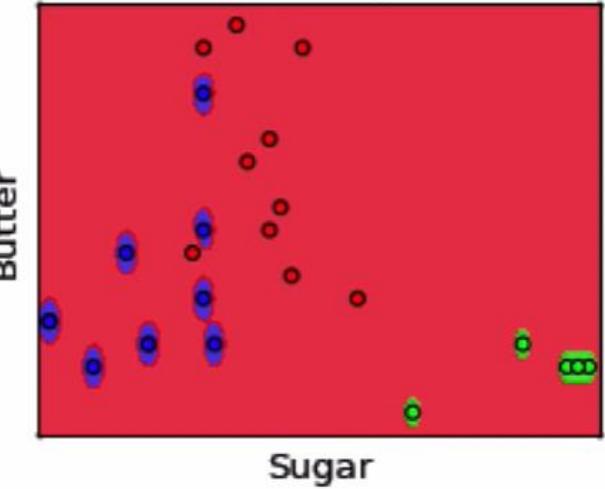
Kernel: Linear
C: 1

- Muffin
- Cupcake
- Scone



Kernel: RBF
C: 1
Gamma: 2^{-5}

Small Gamma:
Less complexity



Kernel: RBF
C: 1
Gamma: 2^1

Large Gamma:
More complexity

ข้อสังเกต SVM

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- ต้องมีการ normalize feature
- If the number of features is much greater than the number of samples, avoid over-fitting by choosing linear kernel
- SVMs is not suitable for noisy data

Example



The Challenge

Classify recipes as cupcakes or muffins. When given a new recipe, determine if it's a cupcake or a muffin.

Prepare data

Recipe	Flour	Sugar	Butter	Milk	Egg	Baking Powder	Salt	Vanilla
Muffin1	2 cups	1/2 cup	1/4 cup	1 cup	2 eggs	1 tbsp	1/2 tsp	1 tsp
Cupcake1	2 cups	3/4 cup	1/2 cup	1 cup	2 eggs	2 tsp	1/2 tsp	1 tsp
...

Amount-based

Recipe	Flour	Sugar	Other
Muffin1	2 cups	1/2 cup	...
Cupcake1	2 cups	3/4 cup	...



Percent-based

Recipe	Flour	Sugar	Other	Total Volume
Muffin1	47%	24%	...	100%
Cupcake1	42%	21%	...	100%

Python

Type	Flour	Milk	Sugar	Butter	Egg	Baking Powder	Vanilla	Salt
Muffin	55	28	3	7	5	2	0	0
Muffin	47	24	12	6	9	1	0	0
Muffin	47	23	18	6	4	1	0	0
Muffin	50	25	12	6	5	2	1	0
Muffin	55	27	3	7	5	2	1	0
Muffin	54	27	7	5	5	2	0	0
Muffin	47	26	10	10	4	1	0	0
Muffin	50	17	17	8	6	1	0	0
Muffin	50	17	17	11	4	1	0	0
Cupcake	39	0	26	19	14	1	1	0
Cupcake	34	17	20	20	5	2	1	0
Cupcake	39	13	17	19	10	1	1	0
Cupcake	38	15	23	15	8	0	1	0
Cupcake	42	18	25	9	5	1	0	0
Cupcake	36	14	21	14	11	2	1	0
Cupcake	38	15	31	8	6	1	1	0
Cupcake	36	16	24	12	9	1	1	0

- a. Import Libraries
- b. Import Data
- c. Prepare the Data
- d. Fit the Model
- e. Visualize Results
- f. Predict New Case

Import Library

Cupcakes vs Muffins with SVM

Step 1: Import Libraries

```
In [13]: # Allows charts to appear in the notebook
%matplotlib inline

# Libraries for analysis
import pandas as pd
import numpy as np
from sklearn import svm

# Libraries for visuals
import matplotlib.pyplot as plt
import seaborn as sns; sns.set(font_scale=1.2)
```

Import data

```
In [2]: # Read in muffin and cupcake ingredient data  
recipes = pd.read_csv('data_recipes.csv')  
recipes.head()
```

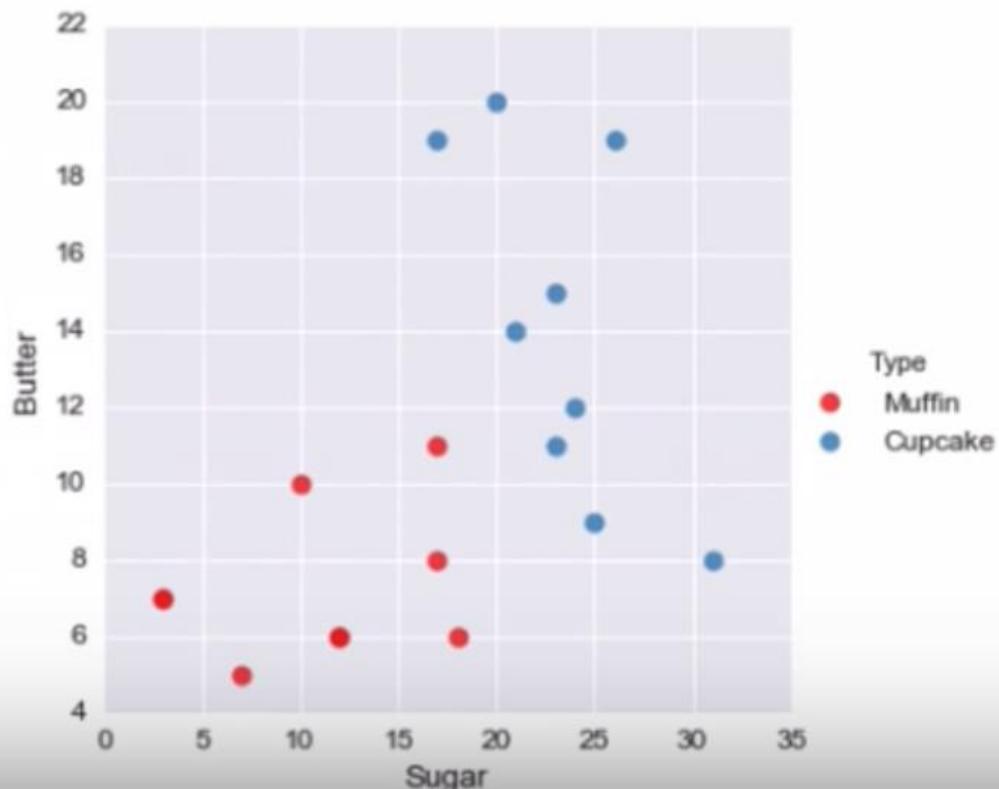
Out[2]:

	Type	Flour	Milk	Sugar	Butter	Egg	Baking Powder	Vanilla	Salt
0	Muffin	55	28	3	7	5	2	0	0
1	Muffin	47	24	12	6	9	1	0	0
2	Muffin	47	23	18	6	4	1	0	0
3	Muffin	50	25	12	6	5	2	1	0
4	Muffin	55	27	3	7	5	2	1	0

Prepare the data

```
In [3]: # Plot two ingredients  
sns.lmplot('Sugar', 'Butter', data=recipes, hue='Type',  
            palette='Set1', fit_reg=False, scatter_kws={"s": 70})
```

Out[3]: <seaborn.axisgrid.FacetGrid at 0xad7af28>



Fit the model

```
In [4]: # Specify inputs for the model
sugar_butter = recipes[['Sugar','Butter']].as_matrix()
type_label = np.where(recipes['Type']=='Muffin', 0, 1)

In [5]: # Fit the SVM model
model = svm.SVC(kernel='linear')
model.fit(sugar_butter, type_label)

Out[5]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
           decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
           max_iter=-1, probability=False, random_state=None, shrinking=True,
           tol=0.001, verbose=False)
```

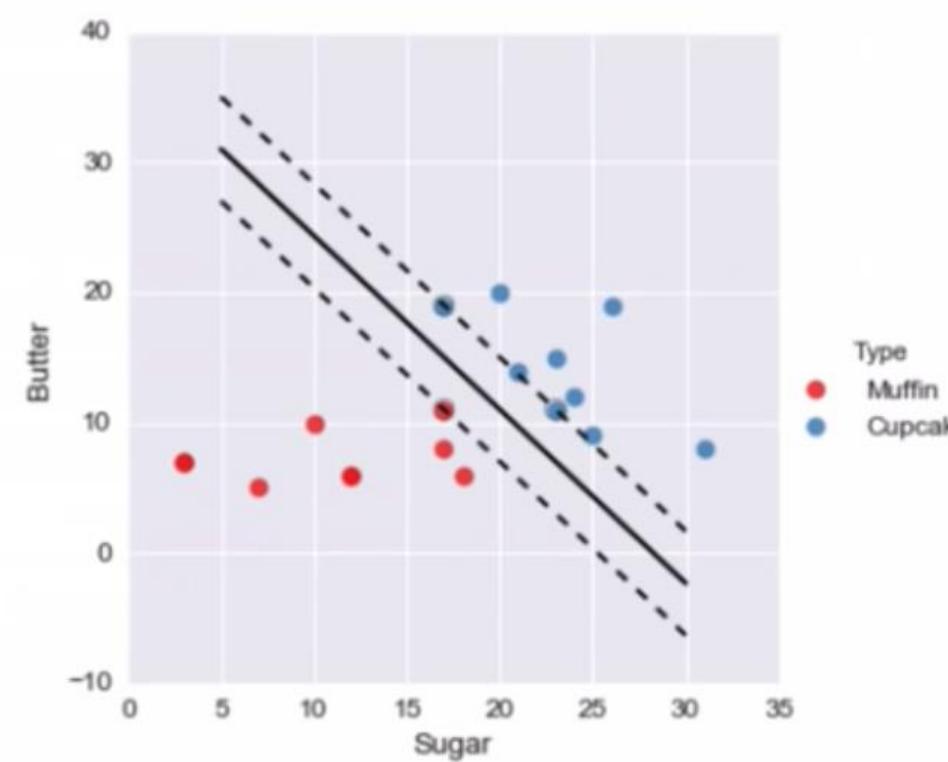
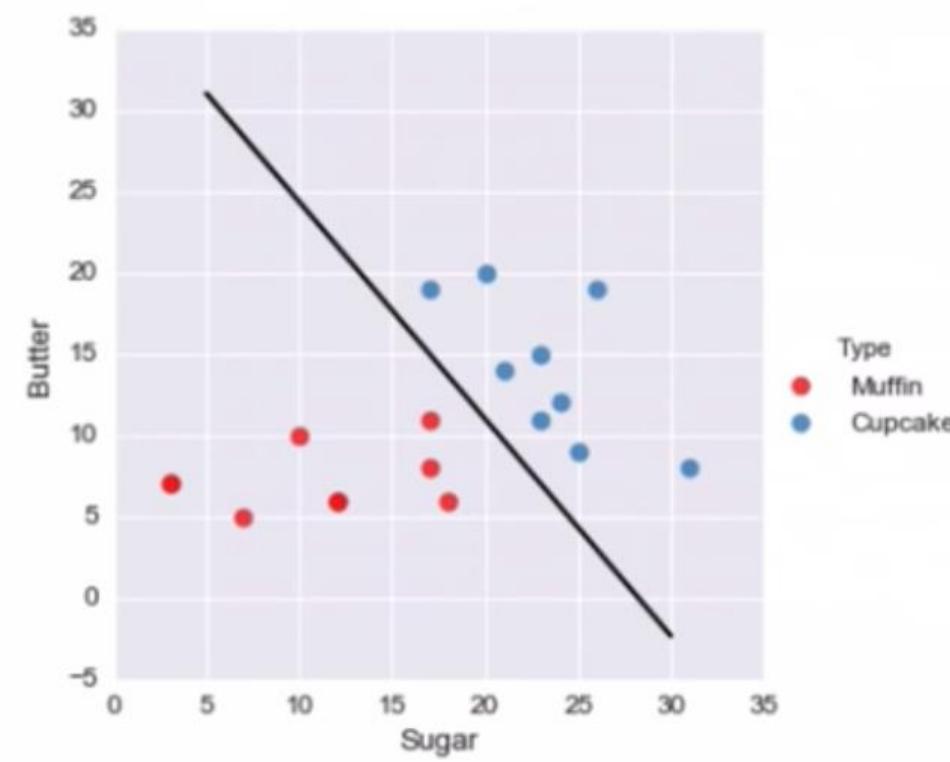
Visualize the result

```
In [6]: # Get the separating hyperplane
w = model.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(5, 30)
yy = a * xx - (model.intercept_[0]) / w[1]

# Plot the parallels to the separating hyperplane
# that pass through the support vectors
b = model.support_vectors_[0]
yy_down = a * xx + (b[1] - a * b[0])
b = model.support_vectors_[-1]
yy_up = a * xx + (b[1] - a * b[0])
```

```
In [7]: # Look at the margins and support vectors
sns.lmplot('Sugar', 'Butter', data=recipes, hue='Type',
            palette='Set1', fit_reg=False, scatter_kws={"s": 70})
plt.plot(xx, yy, linewidth=2, color='black')
plt.plot(xx, yy_down, 'k--')
plt.plot(xx, yy_up, 'k--')
plt.scatter(model.support_vectors_[:, 0], model.support_vectors_[:, 1],
            s=80, facecolors='none')
```

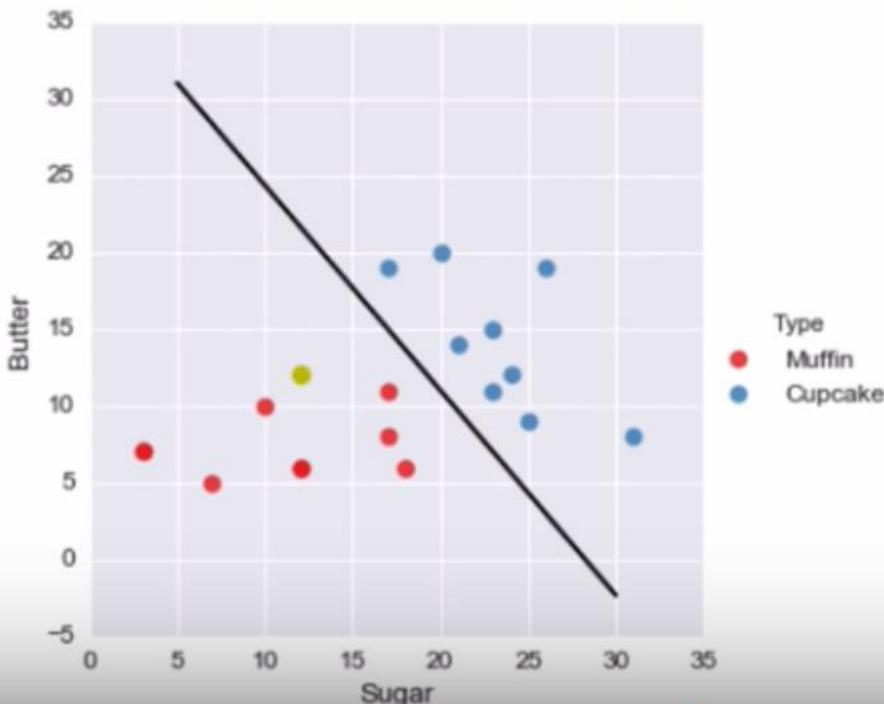
```
In [12]: # Plot the hyperplane
sns.lmplot('Sugar', 'Butter', data=recipes, hue='Type',
            palette='Set1', fit_reg=False, scatter_kws={"s": 70})
plt.plot(xx, yy, linewidth=2, color='black')
```



Predict the new case

```
In [10]: # Plot the point to visually see where the point lies
sns.lmplot('Sugar', 'Butter', data=recipes, hue='Type',
            palette='Set1', fit_reg=False, scatter_kws={"s": 70})
plt.plot(xx, yy, linewidth=2, color='black')
plt.plot(12, 12, 'yo', markersize='9')
```

```
Out[10]: [
```



Step 6: Predict New Case

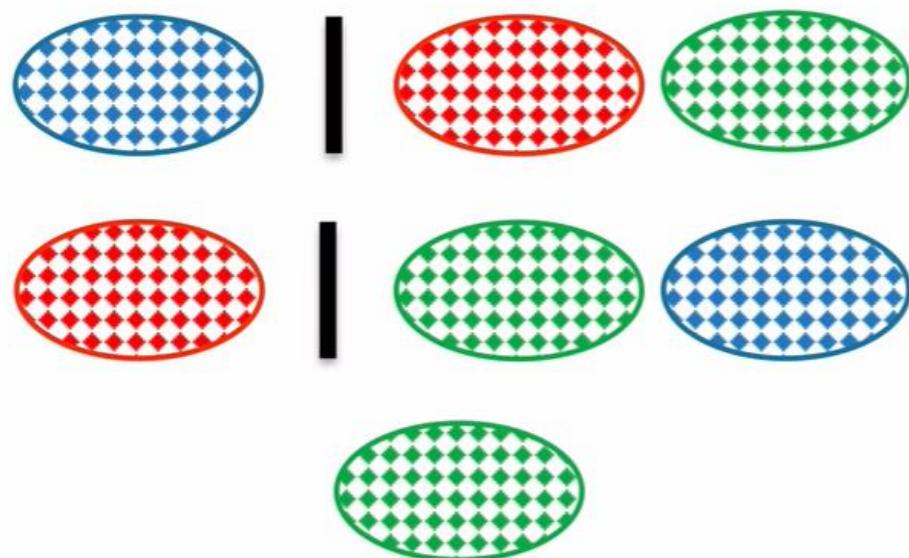
```
In [8]: # Create a function to guess when a recipe is a muffin
# or a cupcake using the SVM model we created
def muffin_or_cupcake(butter, sugar):
    if(model.predict([[butter, sugar]])==0:
        print('You\'re looking at a muffin recipe!')
    else:
        print('You\'re looking at a cupcake recipe!')
```

```
In [9]: # Predict if 12 parts butter and sugar
muffin_or_cupcake(12,12)
```

You're looking at a muffin recipe!

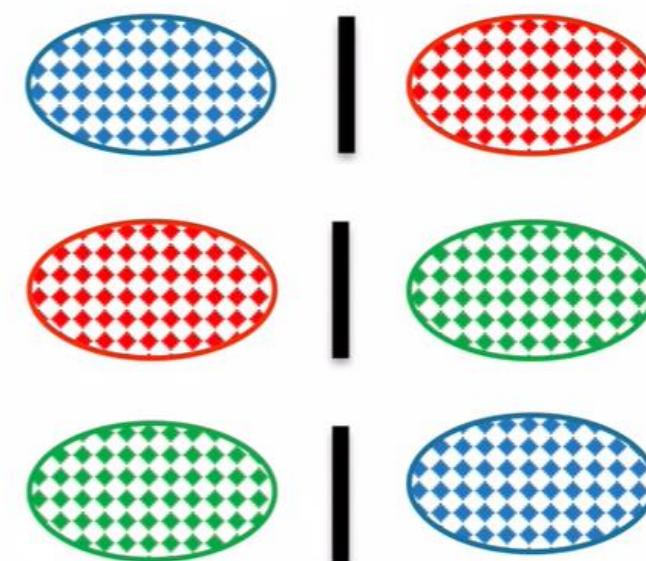
Multiple Classes: Comparison

OVR: One vs Rest



Pros: Fewer classifications
Cons: Classes may be imbalanced

OVO: One vs One



Pros: Less sensitive to imbalance
Cons: More classifications

SVM software tools

- Weka → function → SMO (Sequential Minimal Optimization)
- LibSVM is the name of a software and SMO is the name of an algorithm to solve the SVM.
- LIBSVM → one against one approach
- If k is the number of classes, then $k(k - 1)/2$ classifiers are constructed and each one trains data from two classes.

- In WEKA, SMO and LibSVM are different algorithms, but both can be used to perform SVM.
- Precisely, SMO implements John Platt's sequential minimal optimization algorithm for training a support vector classifier, while, LibSVM is a wrapper class for the libsvm library that supports the classifiers implemented in the libsvm library, including one-class SVMs.
- Therefore, getting different results is an intuitive issue may occur.

Video Clips

- Easiest way to UNDERSTAND the Support Vector Machine!
(16 mins)
- <https://www.youtube.com/watch?v=foWkxFlaigM>
- SVM in python (22 mins)
- <https://www.youtube.com/watch?v=N1vOgolbjSc> (22 mins)
- <https://www.youtube.com/watch?v=TtKF996oEI8>