AI VIETNAM
All-in-One Course
(TA Session)

Support Vector Machine

Exercise

AI VIET NAM
@aivietnam.edu.vn

Dinh-Thang Duong – TA

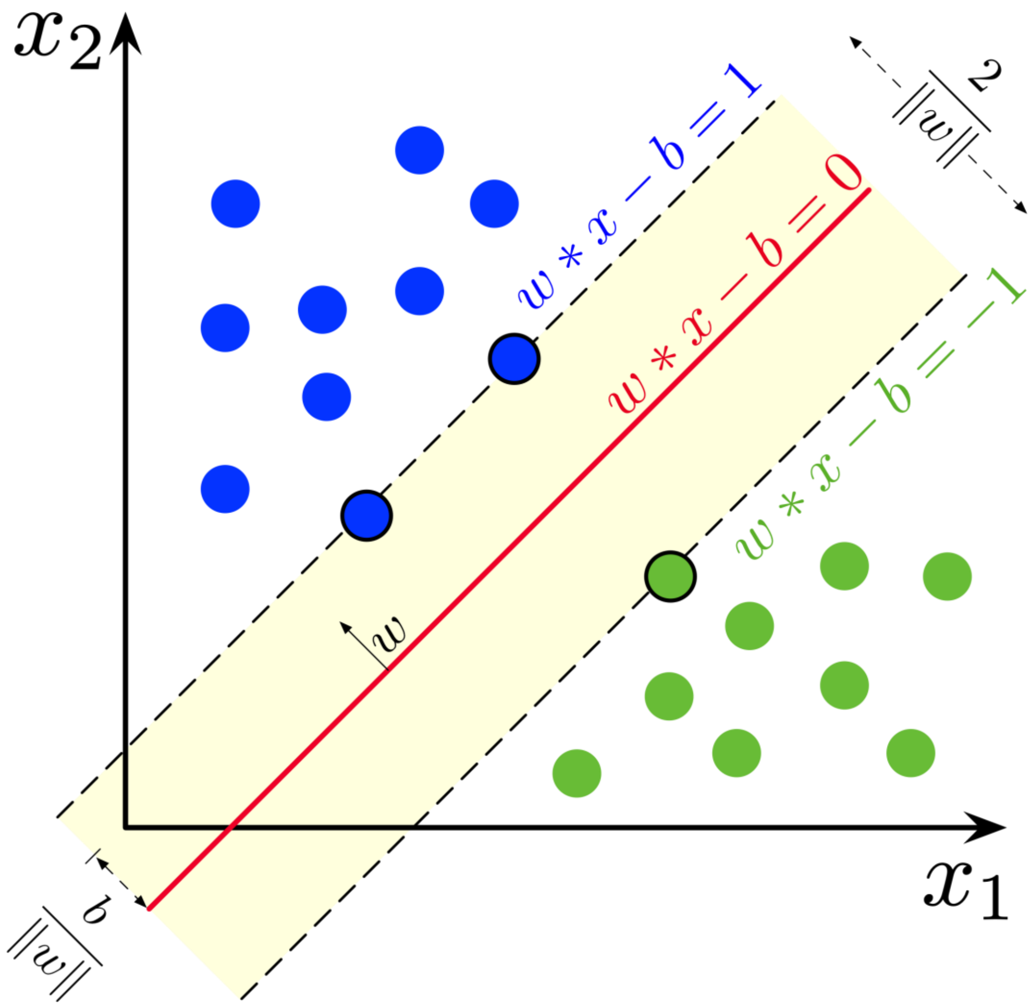
Year 2023

Outline

- Review
- Code Exercises
- Question

Review

❖ Introduction

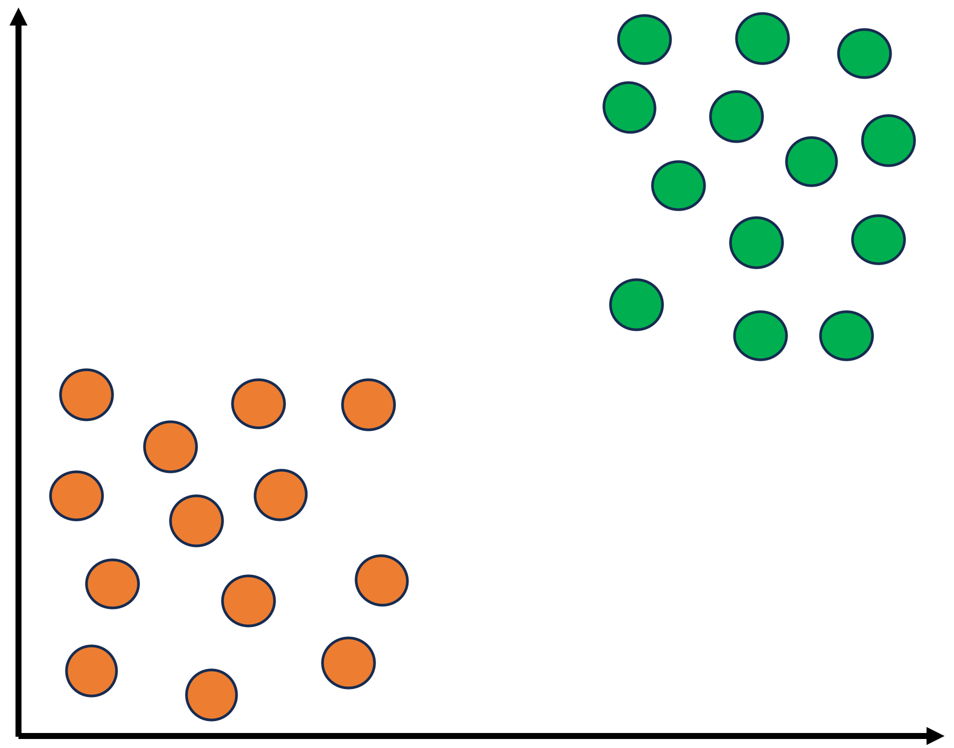


Support Vector Machine (SVM): A supervised-learning ML algorithm that works by identifying the optimal hyperplane that best separates data into different classes.

SVM was originally built for classification task (SVC) but was later modified to fit for regression task (SVR) too.

Support Vector Machine

❖ Getting Started

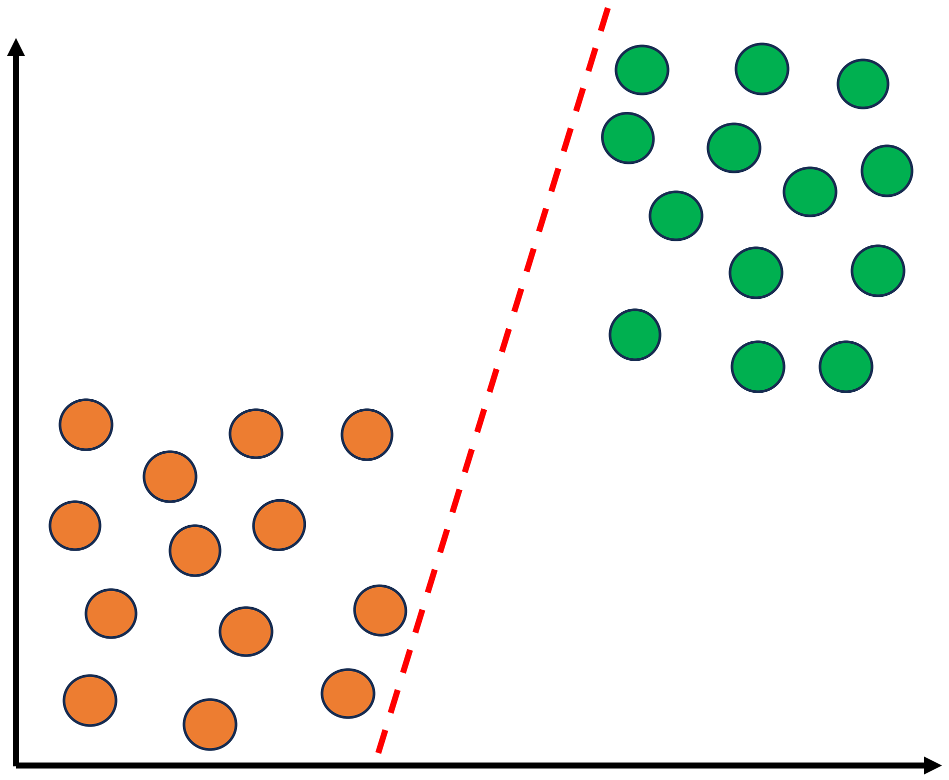


How should we draw a line so that we can perfectly separate this dataset into 2 classes?

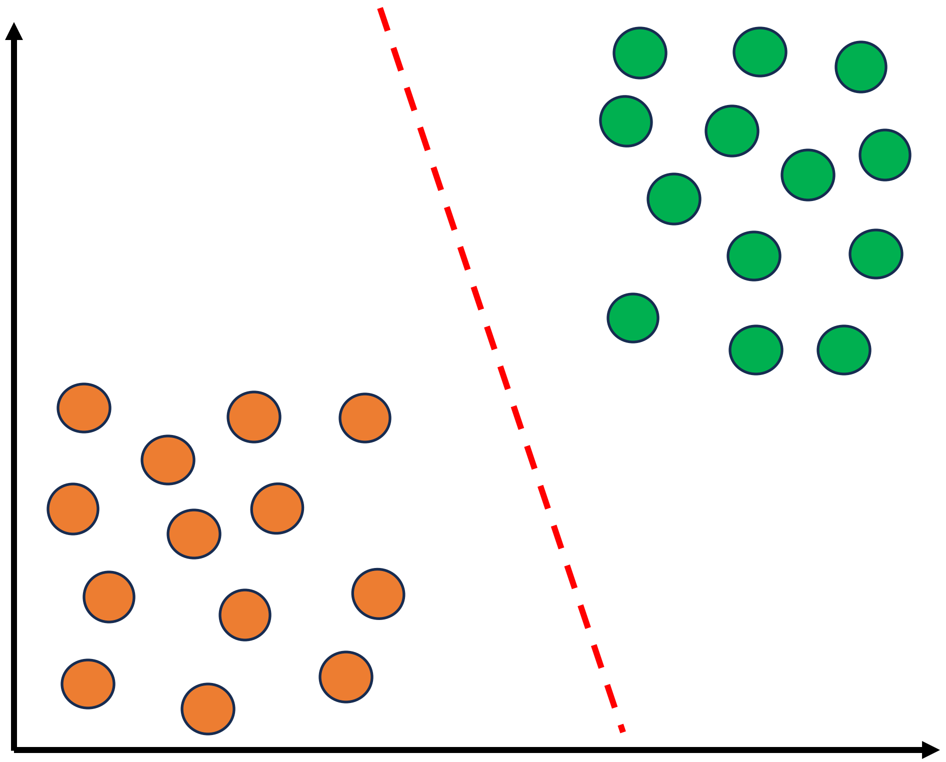
Support Vector Machine

❖ Getting Started

Example 1



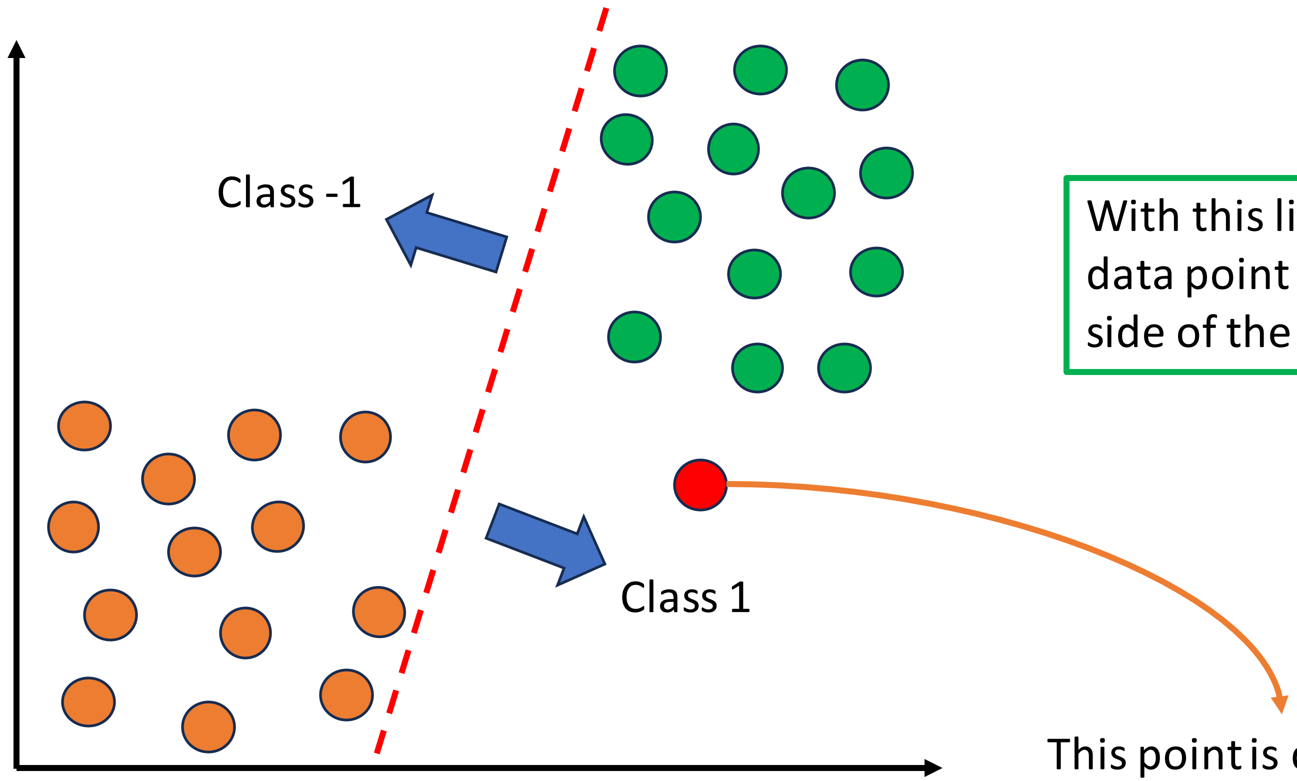
Example 2



There are many ways to draw the line

Support Vector Machine

❖ Getting Started

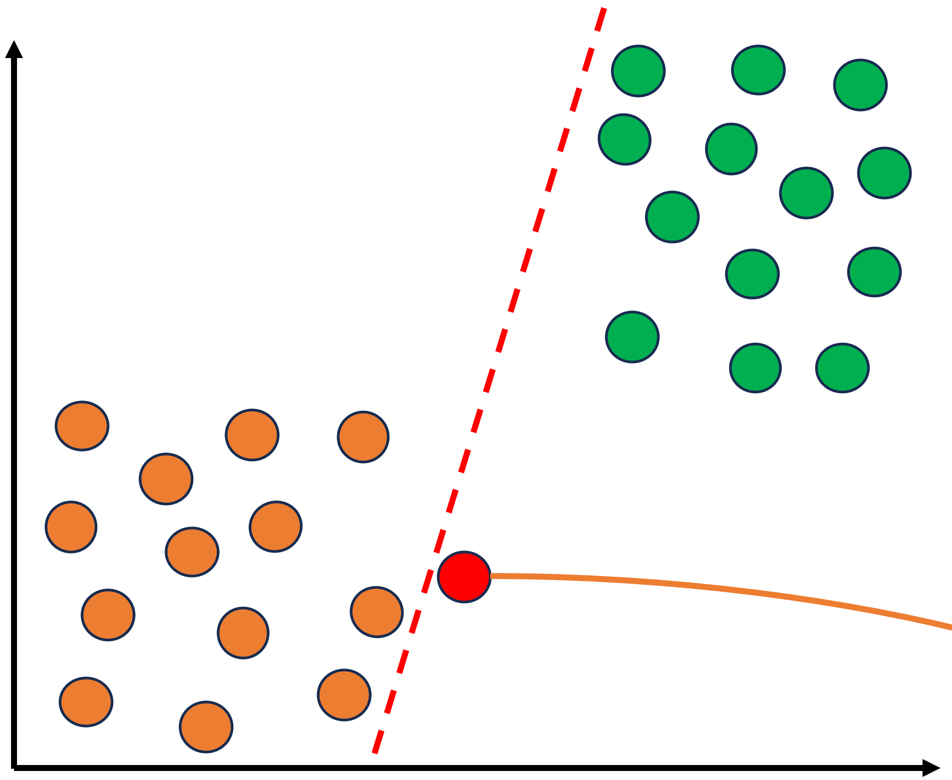


With this line, we can now determine whether a new data point belongs to Class -1 or Class 1 based on which side of the line it falls on.

This point is classified as **Class 1** since it lies on right hand side of the line.

Support Vector Machine

❖ Getting Started



However, in this situation, the result seems wrong since the point is more closer to Class -1.

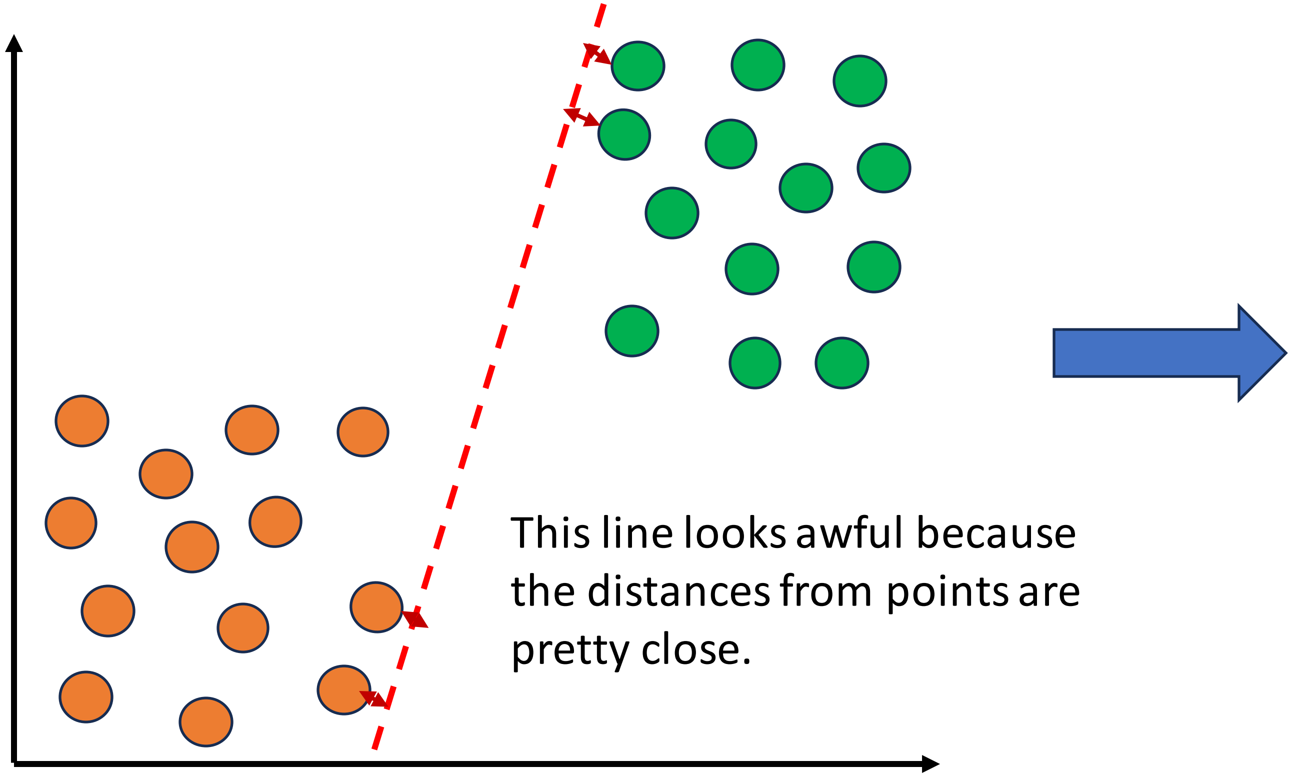
=> This line is not really optimal.

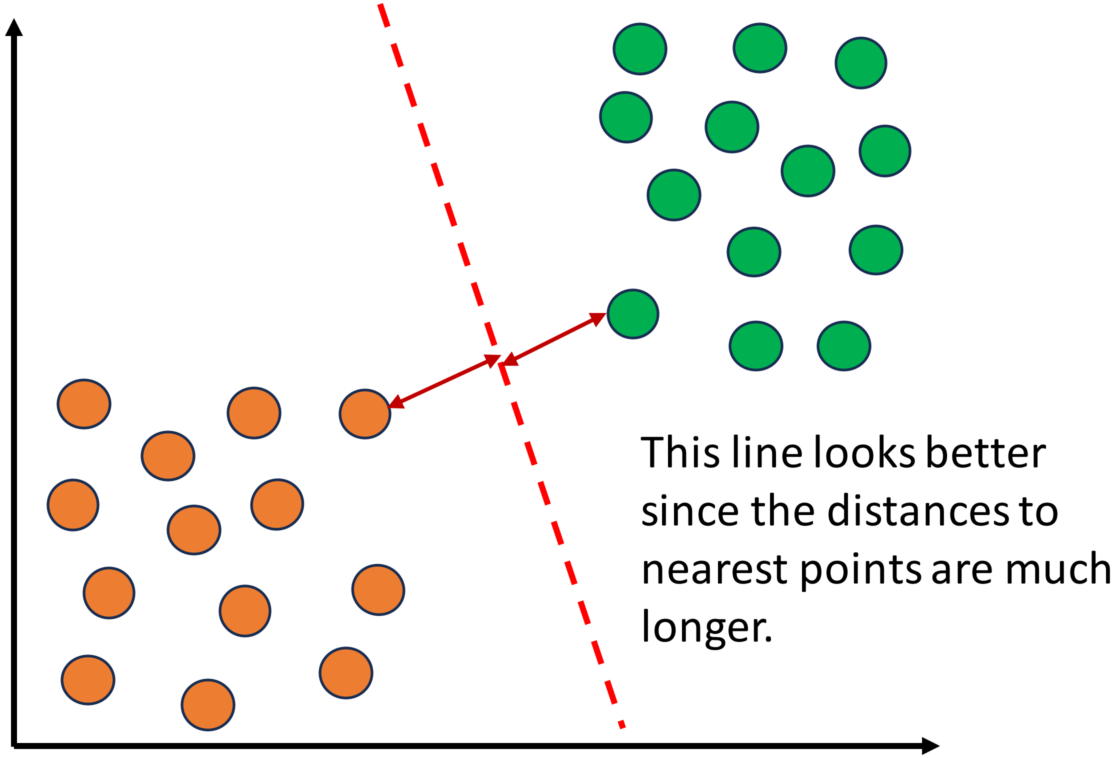
What would be the best line?

This point is classified as **Class 1** since it lies on right hand side of the line.

Support Vector Machine

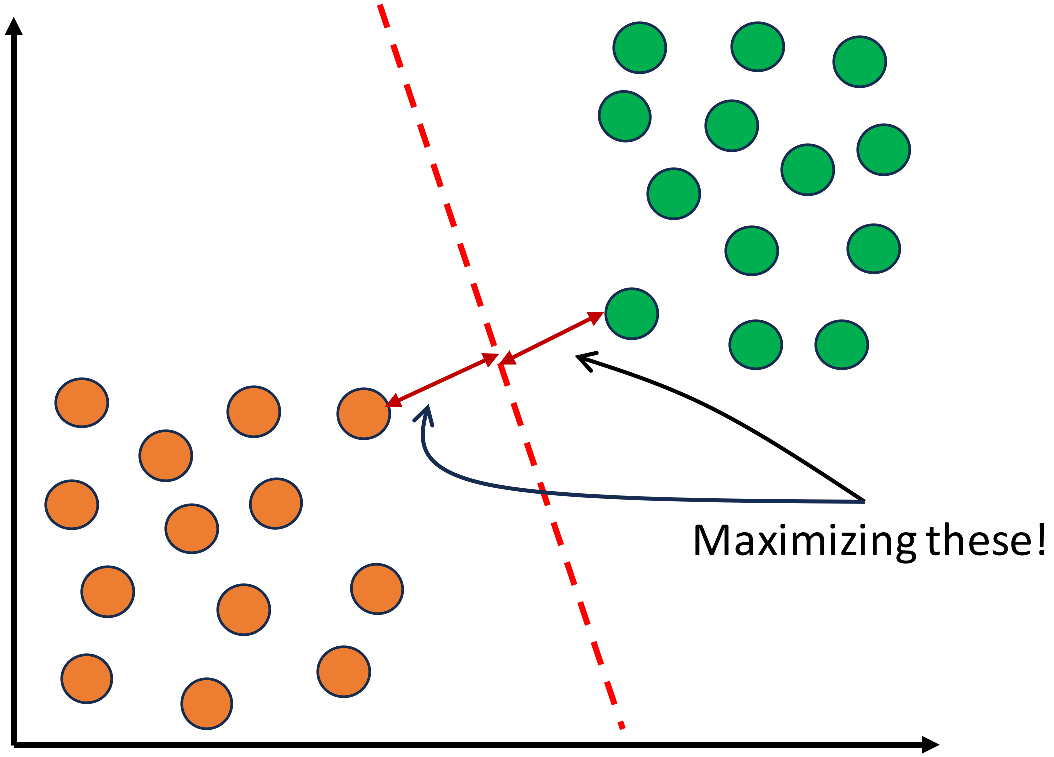
❖ Getting Started





Support Vector Machine

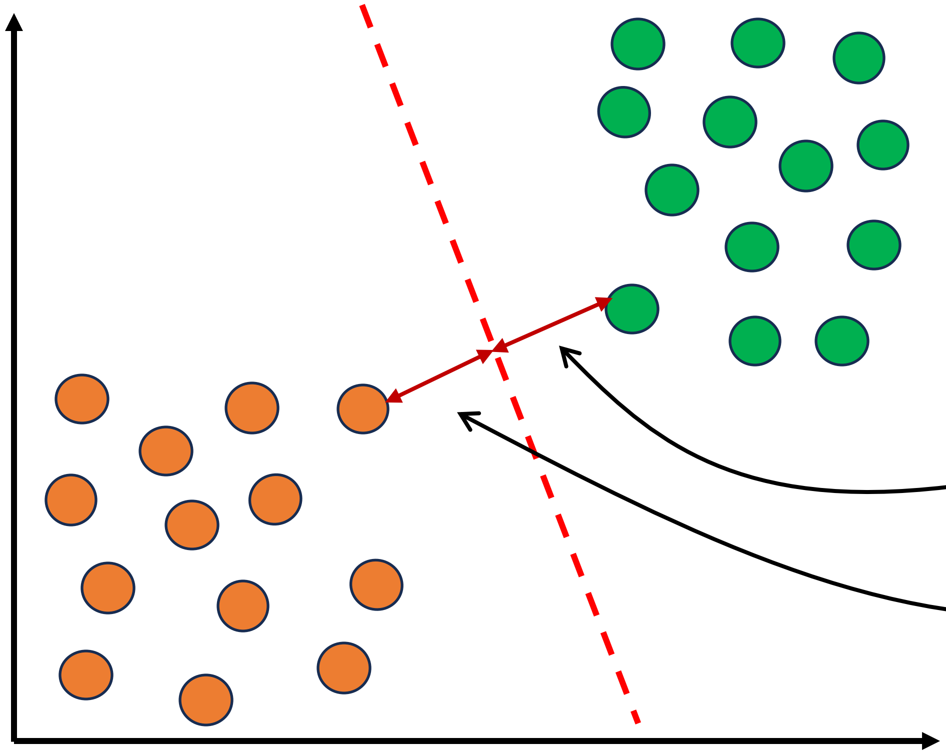
❖ SVM idea



SVM idea: Find the line that best separates the data into classes while maximizing the distances between nearest points.

Support Vector Machine

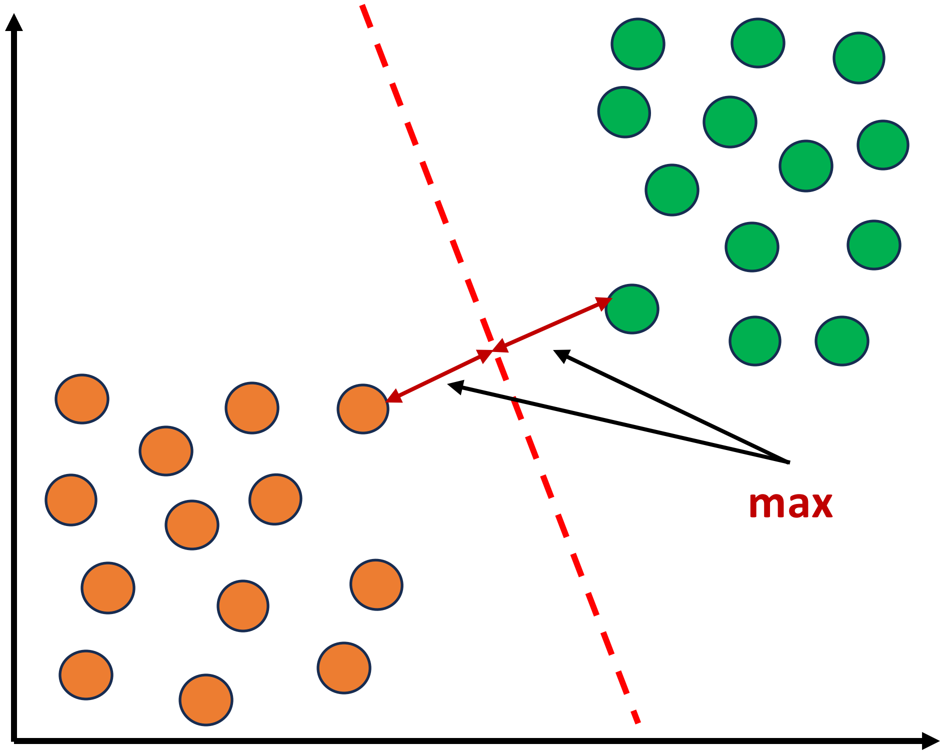
❖ Margin



The distance between the line and the nearest data point from either of the two classes is called **Margin**.

Support Vector Machine

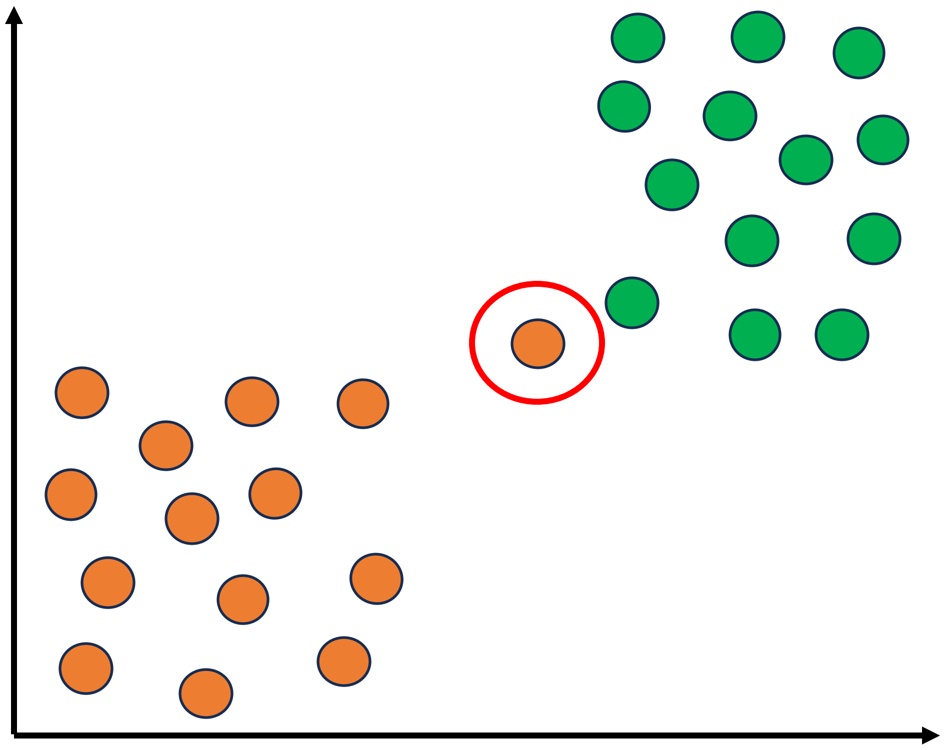
❖ Hard Margin SVM Idea



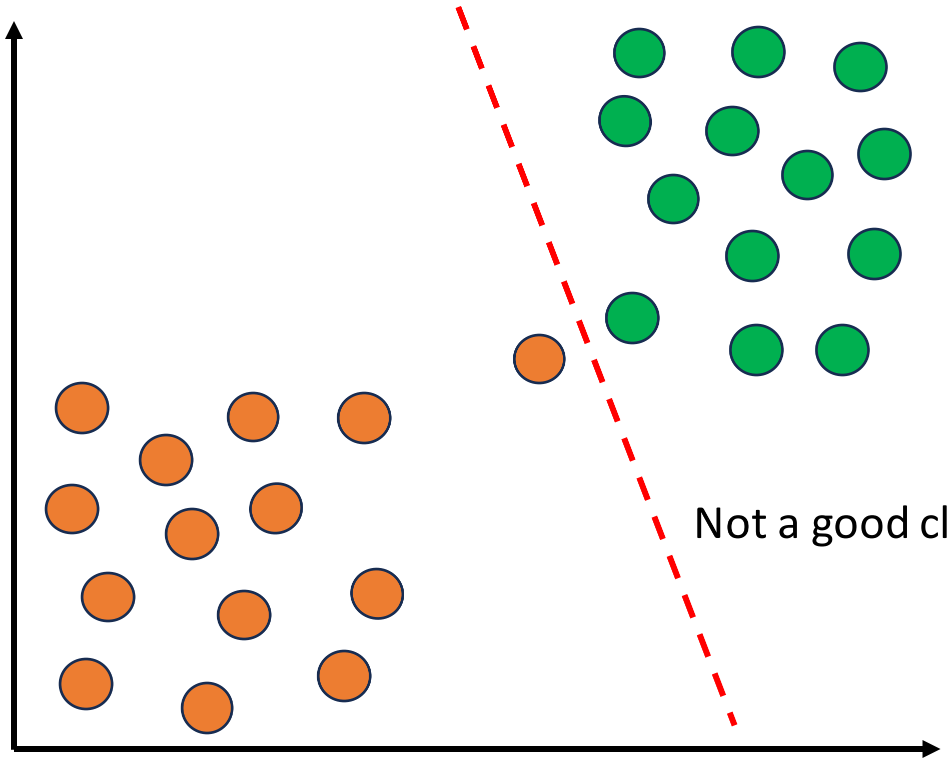
SVM idea: Find the line that best separates the data into classes while **maximizing the margin**. This is called **Hard Margin SVM**.

Support Vector Machine

❖ Hard Margin SVM Problem



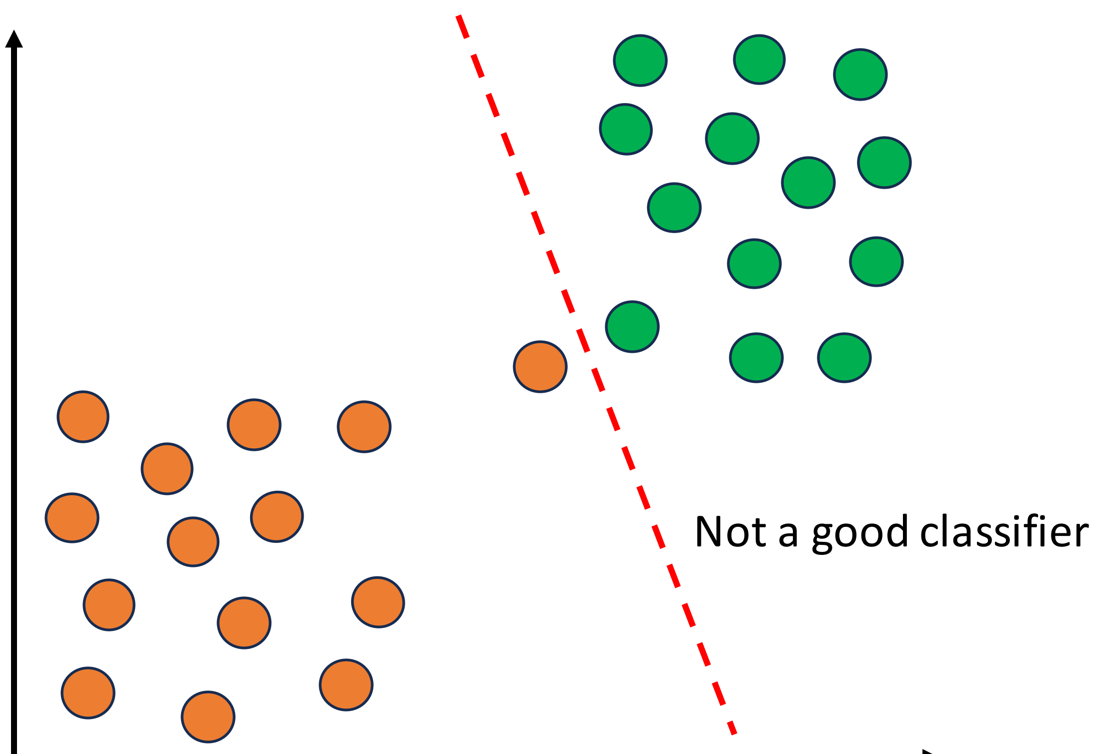
However, assume we have **an outlier**

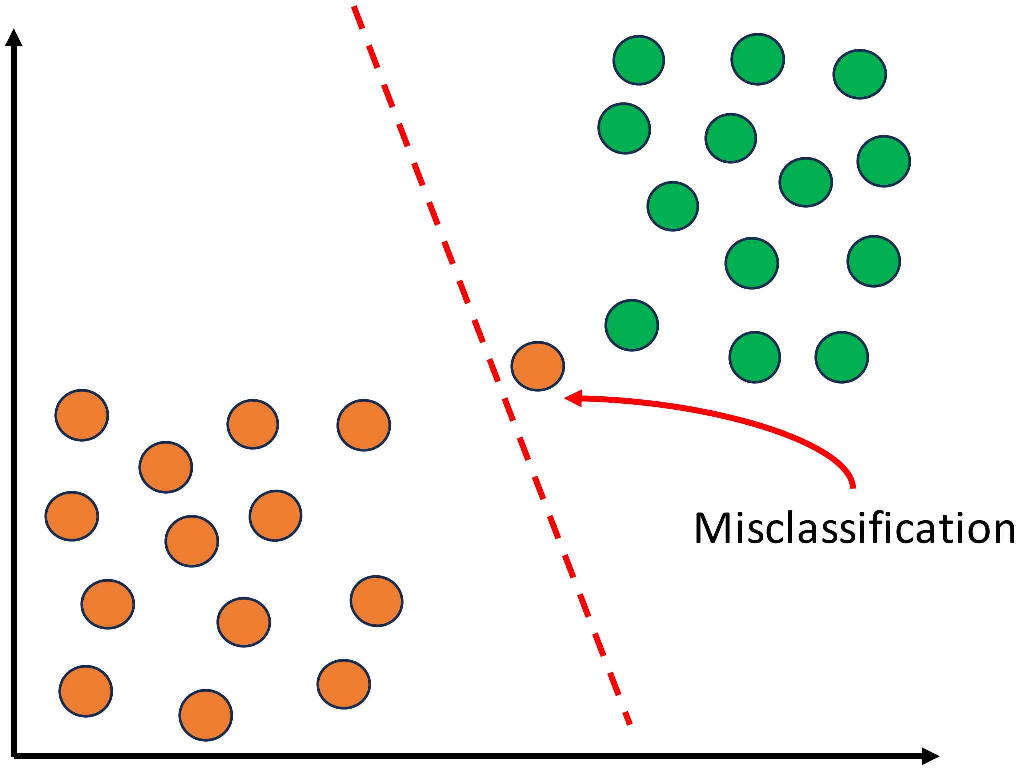


Using Hard Margin SVM, we might have a line like this.

Support Vector Machine

❖ Soft Margin SVM



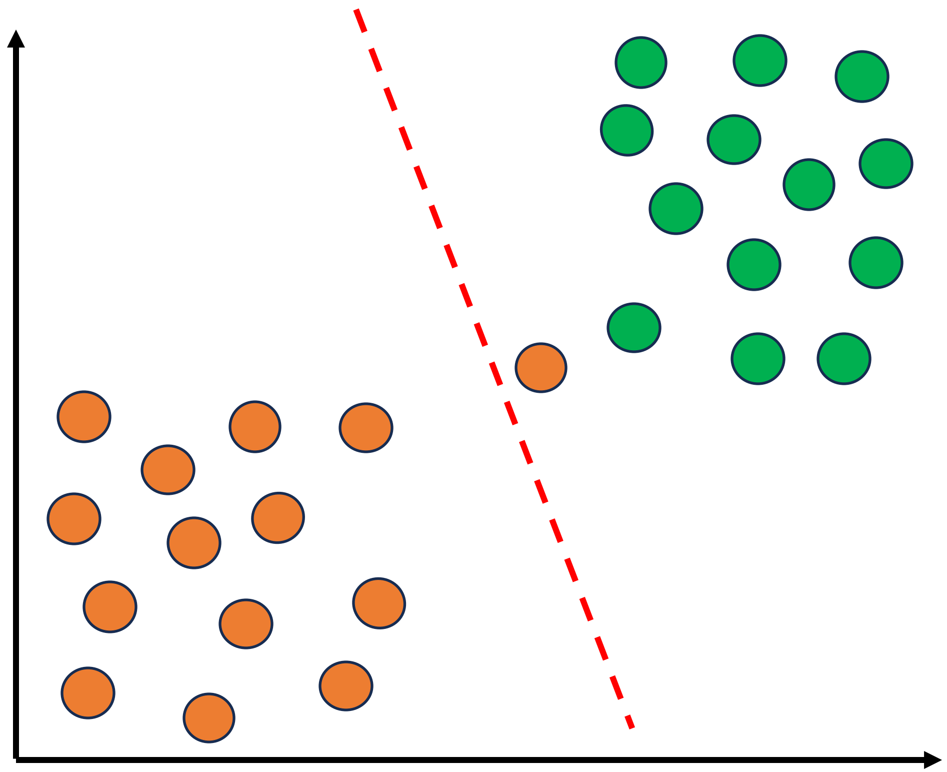


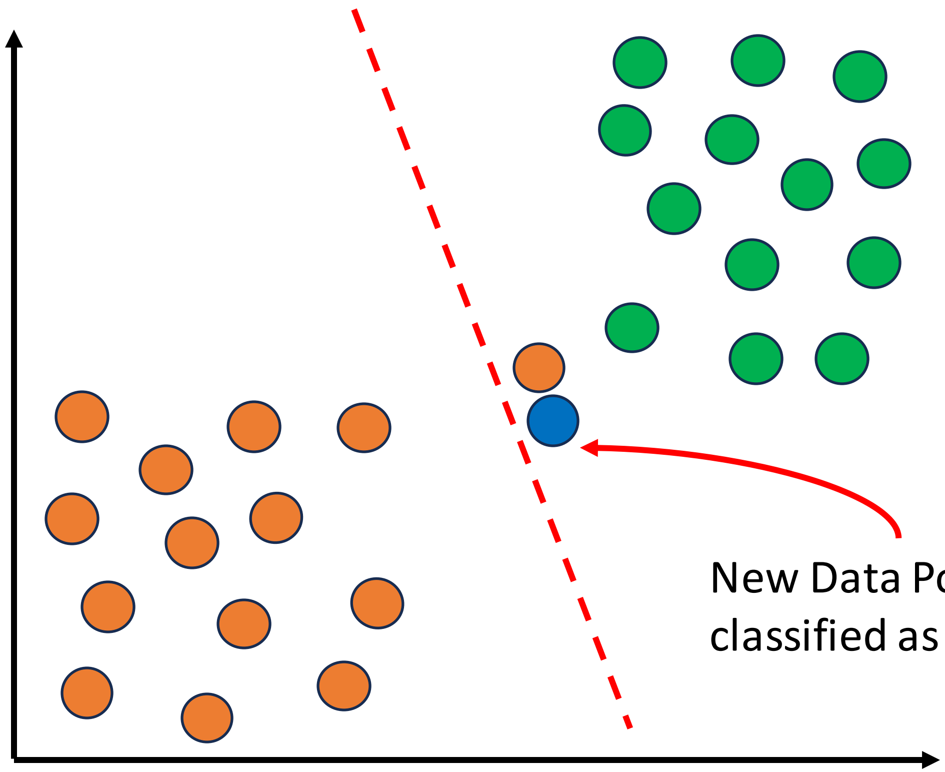
How to avoid this case?

To avoid this, we should **allow misclassifications.**

Support Vector Machine

❖ Soft Margin SVM

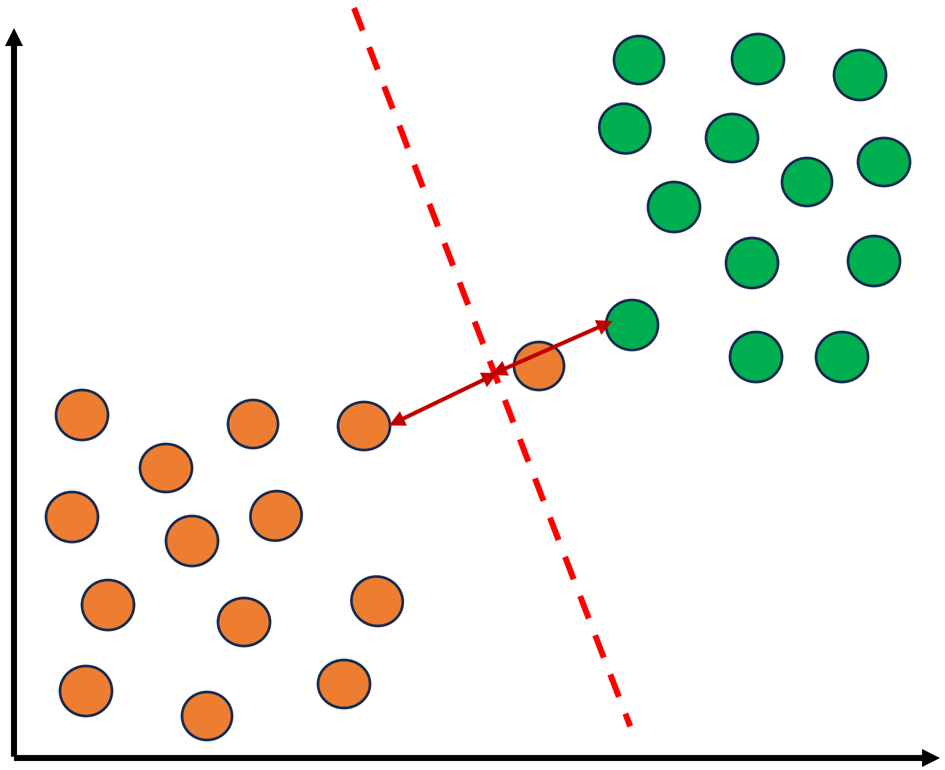




However, when we have a new data point, we might get it right.

Support Vector Machine

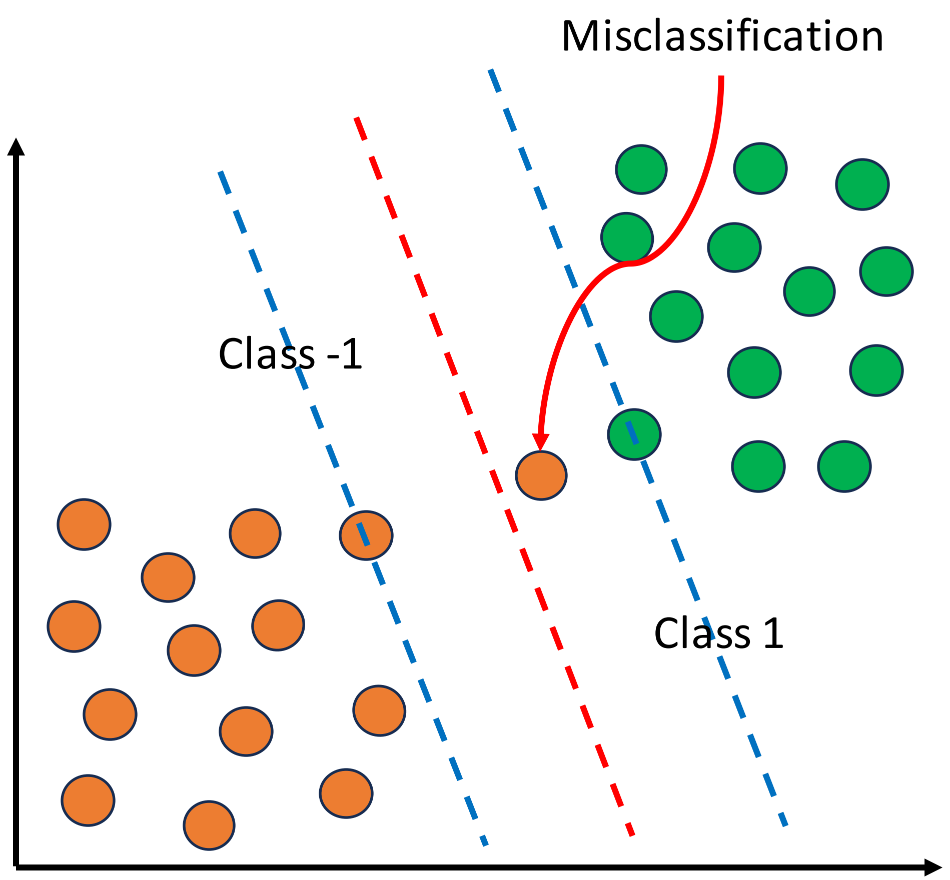
❖ Soft Margin SVM



When we **allow misclassifications**, the distance between the observations and the decision boundary is called **Soft Margin**
=> Soft Margin SVM (Support Vector Classifier).

Support Vector Machine

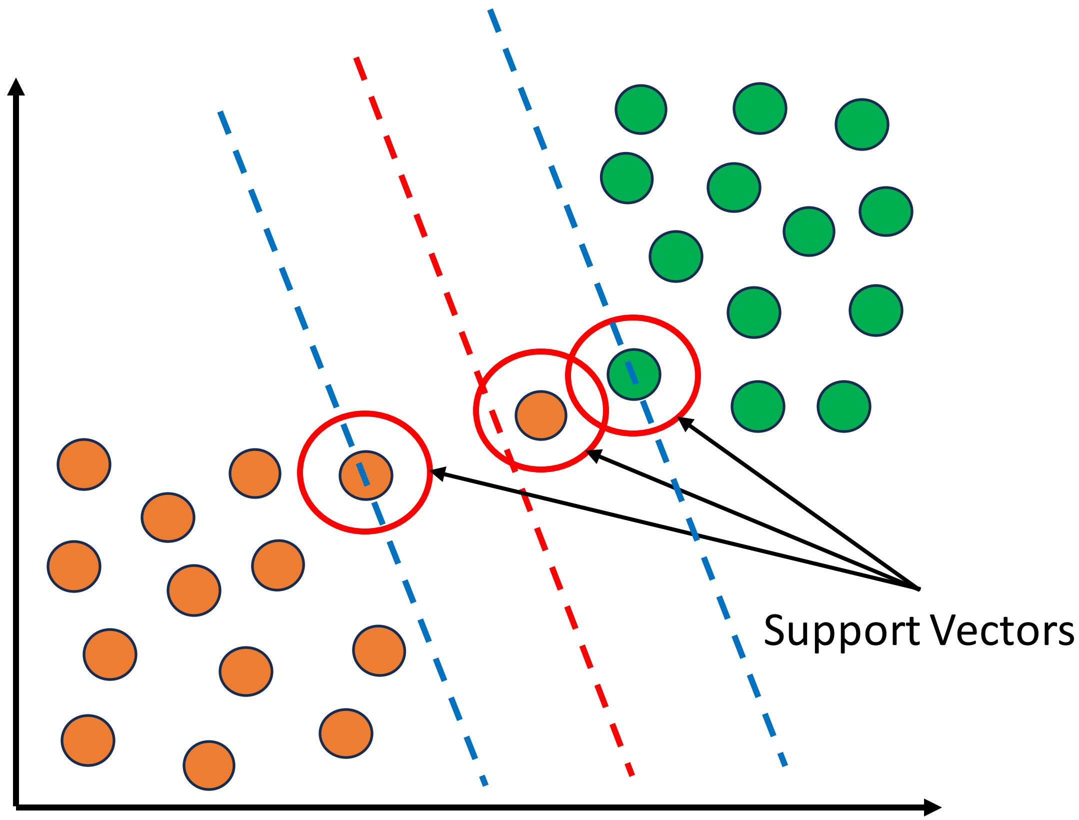
❖ Soft Margin SVM



To better have a sense of relation between data points and Soft Margin, we draw two parallel lines to the Decision Boundary on **Support Vectors**.

Support Vector Machine

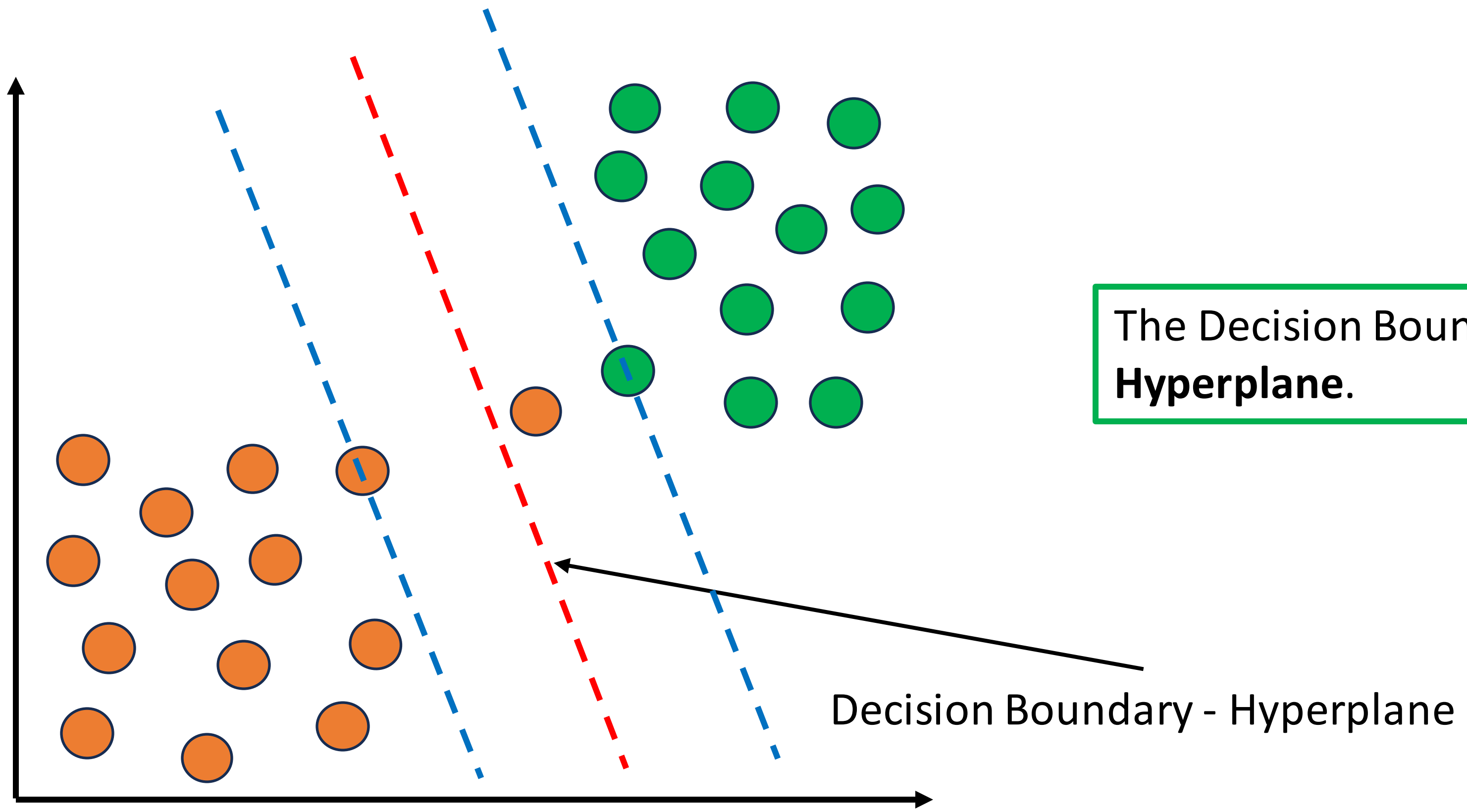
❖ Why "Support" Vector Machine



We called "Support" Vector Classifier because the **data points on the edge and within the Soft Margin** are called Support Vectors.

Support Vector Machine

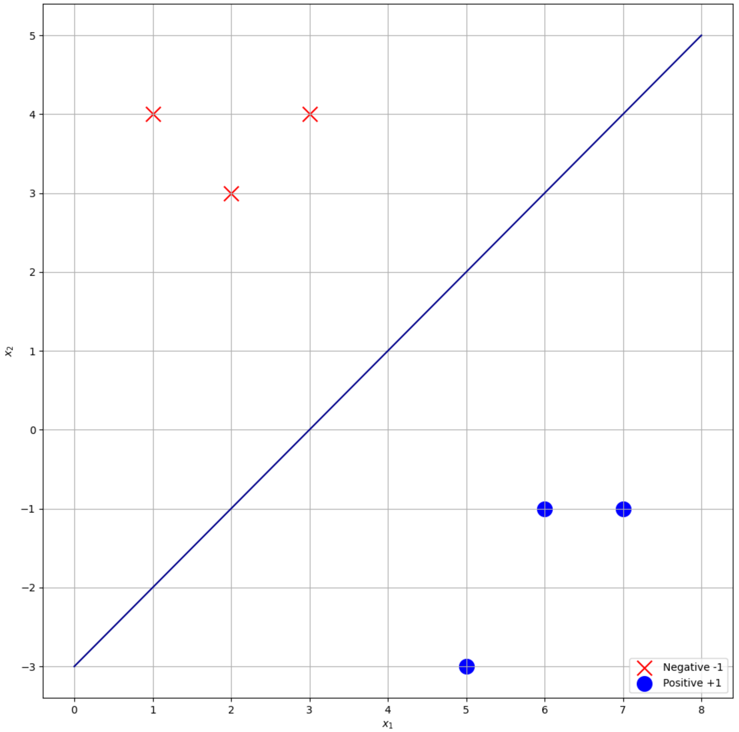
❖ SVM: Hyperplane



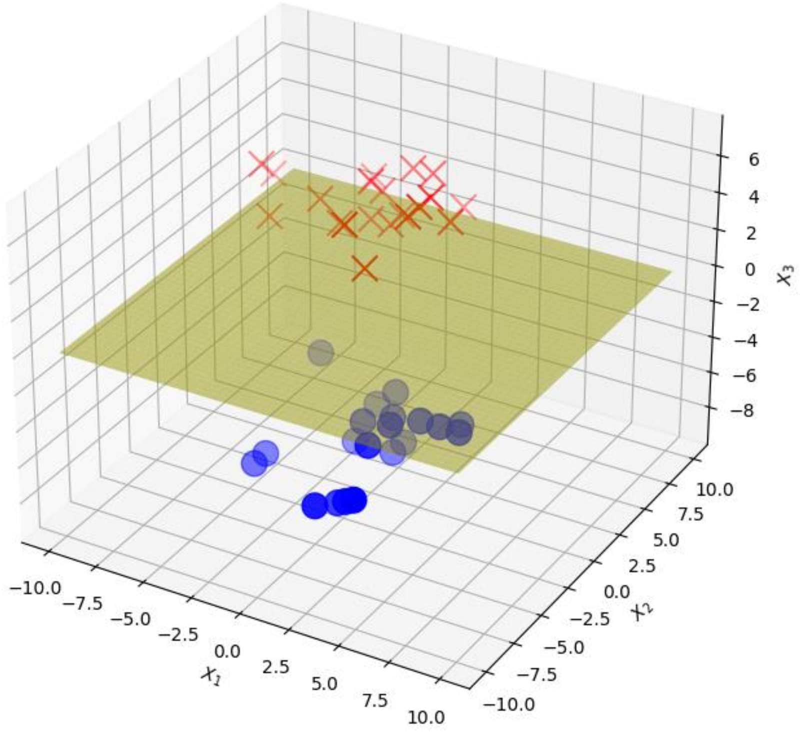
The Decision Boundary (the red line) in SVM is also called **Hyperplane**.

Support Vector Machine

❖ SVM: Hyperplane



In 2D space, decision boundary is a line



But in 3D, decision boundary is instead a plane

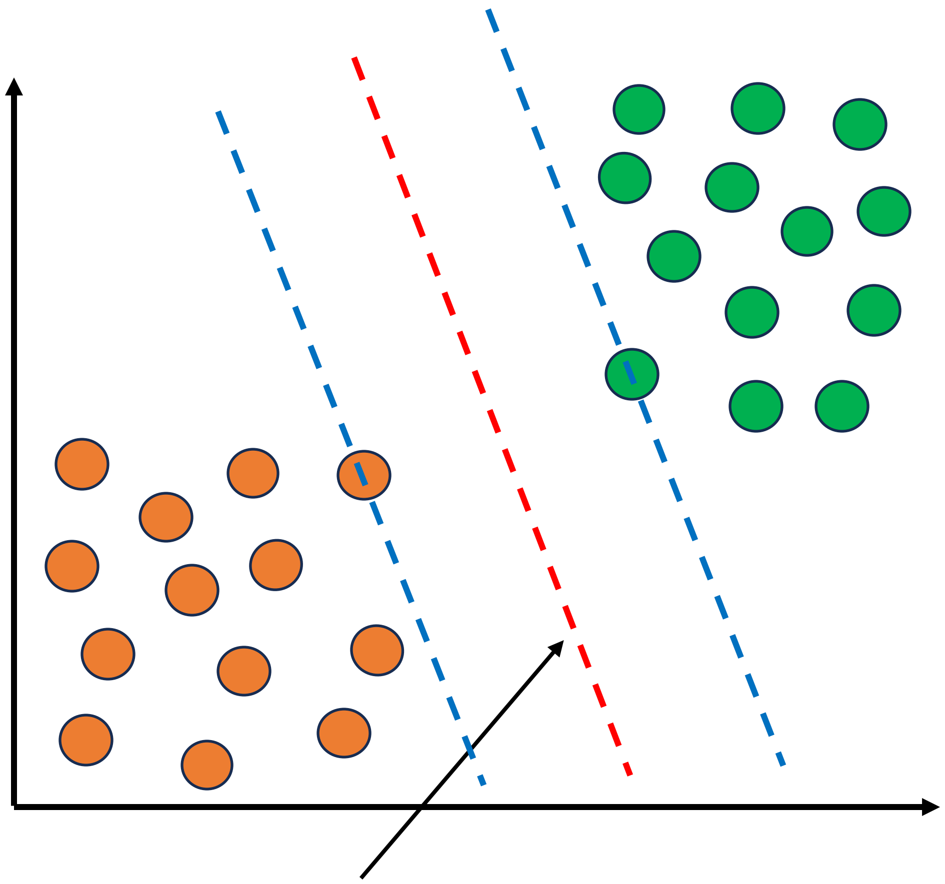
2D → line

3D → plane

Higher → **Hyperplane**

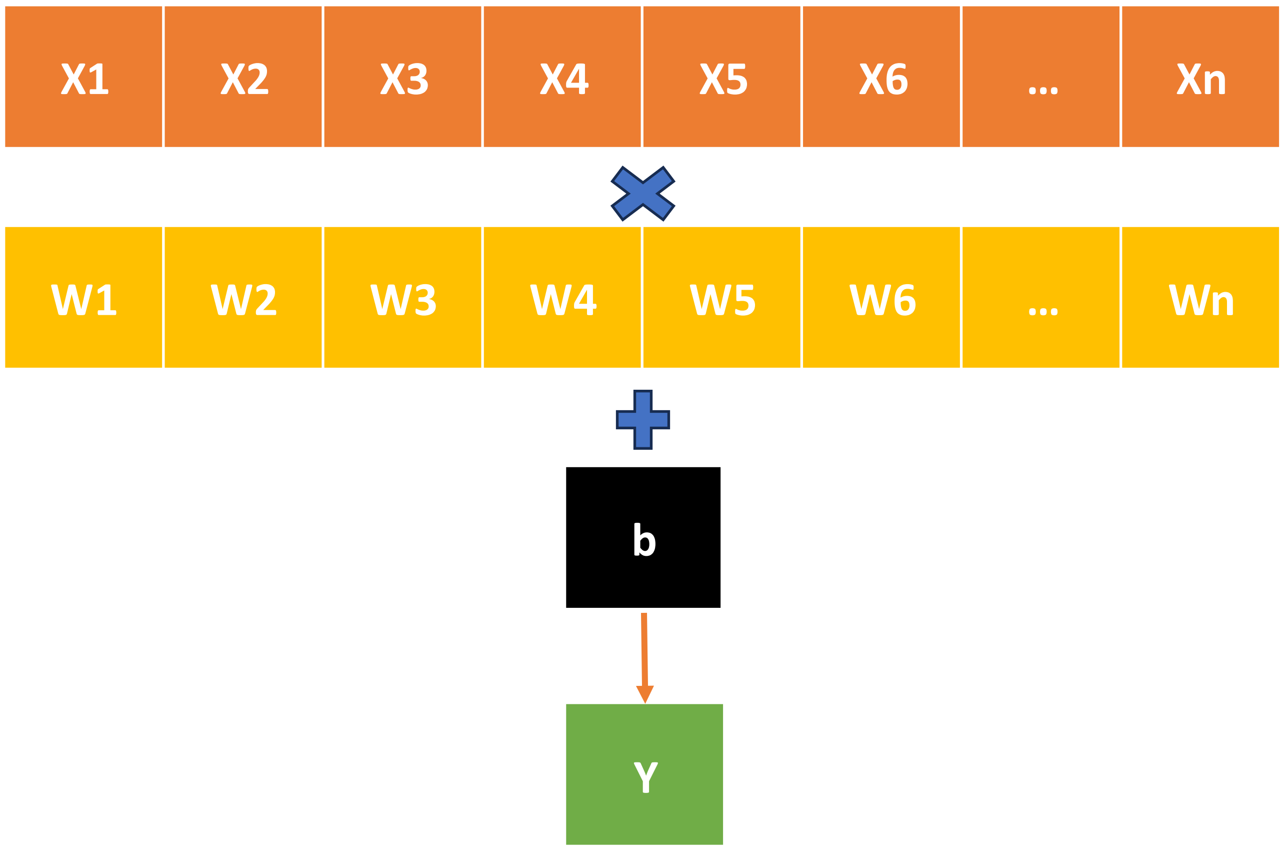
Support Vector Machine

❖ Hard Margin SVM: Prediction



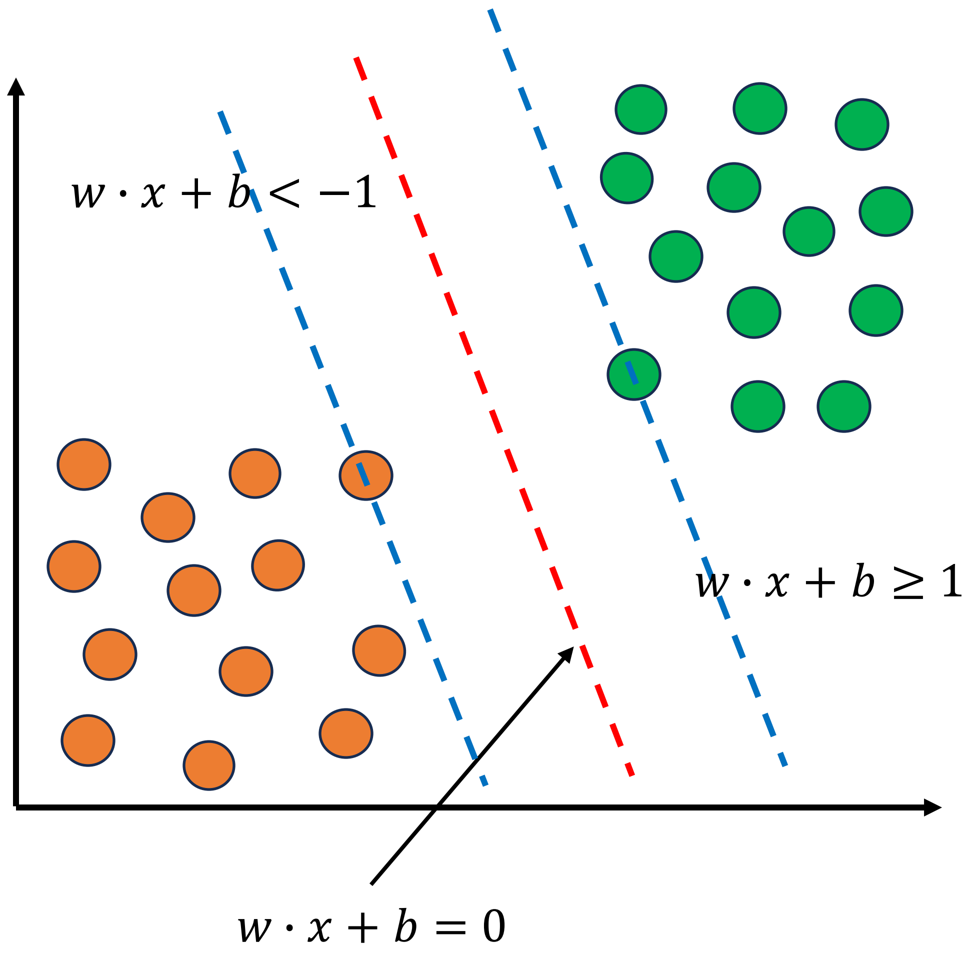
$w \cdot x + b = 0$

How to use Hard Margin SVM for making prediction?

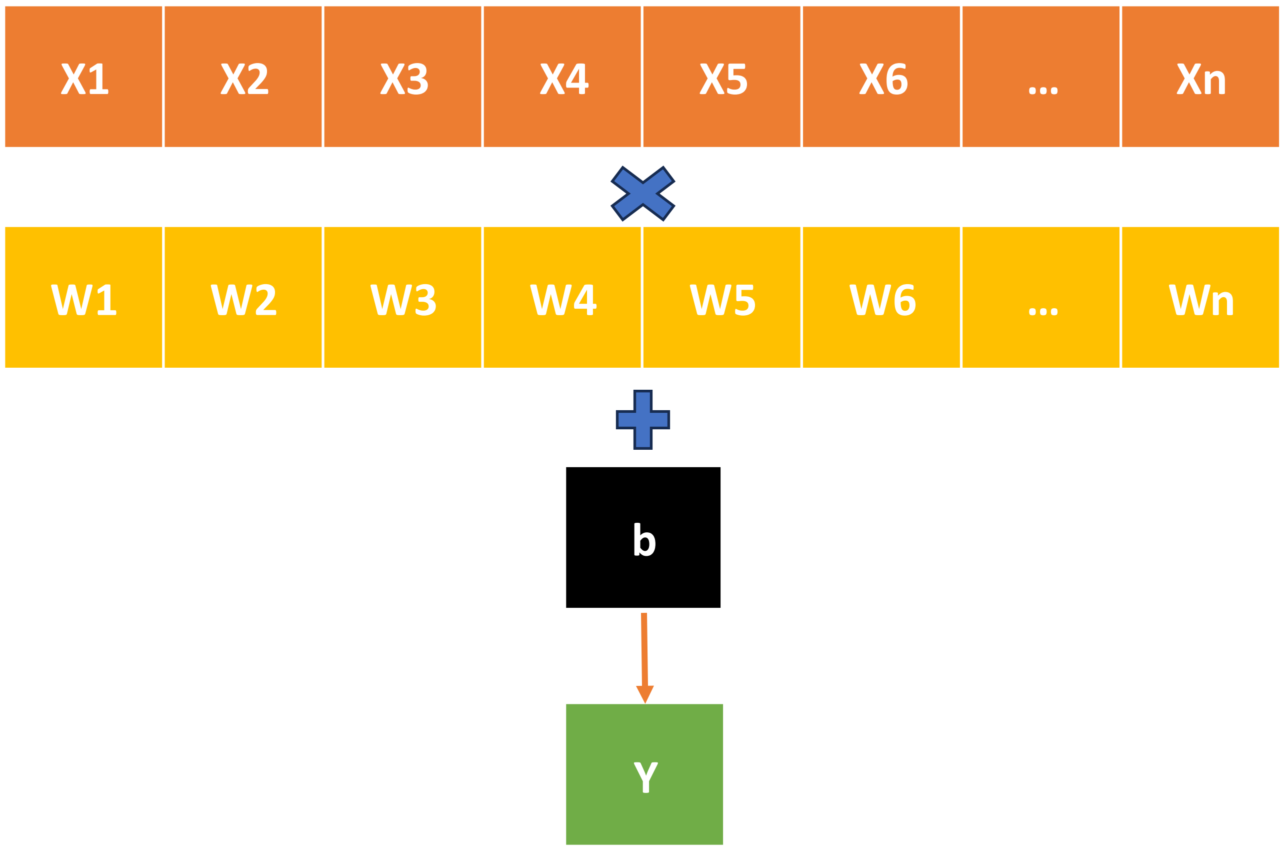


Support Vector Machine

❖ Hard Margin SVM: Prediction

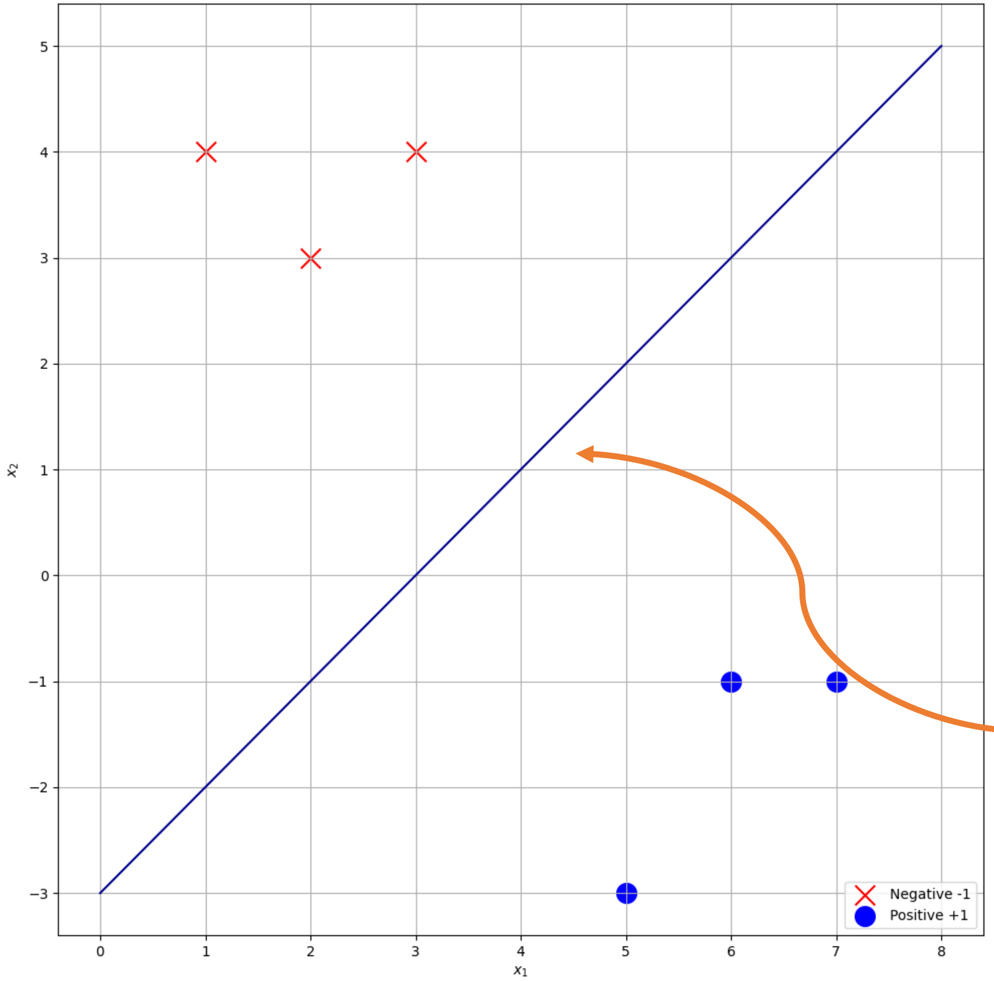


How to use Hard Margin SVM for making prediction?



Support Vector Machine

❖ Hard Margin SVM: Prediction



Equation of Hyperplane

$$w \cdot x + b = 0$$

With w = (1, -1) and b = -3 we get this hyperplane.

Hypothesis Function *h(x)*

- $h(x_i) = \begin{cases} +1 \; if \; w \cdot x + b \geq 1 \\ -1 \; if \; w \cdot x + b < -1 \end{cases}$
- $h(x_i) = sign(w \cdot x + b)$

We use the hypothesis function to predict the class of a data point.

Support Vector Machine

❖ Hard Margin SVM: Prediction

With w = (1, -1) and b = -3, the equation of hyperplane becomes:

$$w \cdot x + b = x_1 - x_2 - 3 = 0$$

X1	X2	Y
3	4	-1
1	4	-1
2	3	-1
6	-1	1
7	-1	1
5	-3	1

(1*3) + (-1*4) + (-3) = -7 < 0 → $y_{predict} = -1$

Classifying a data point using the hyperplane.

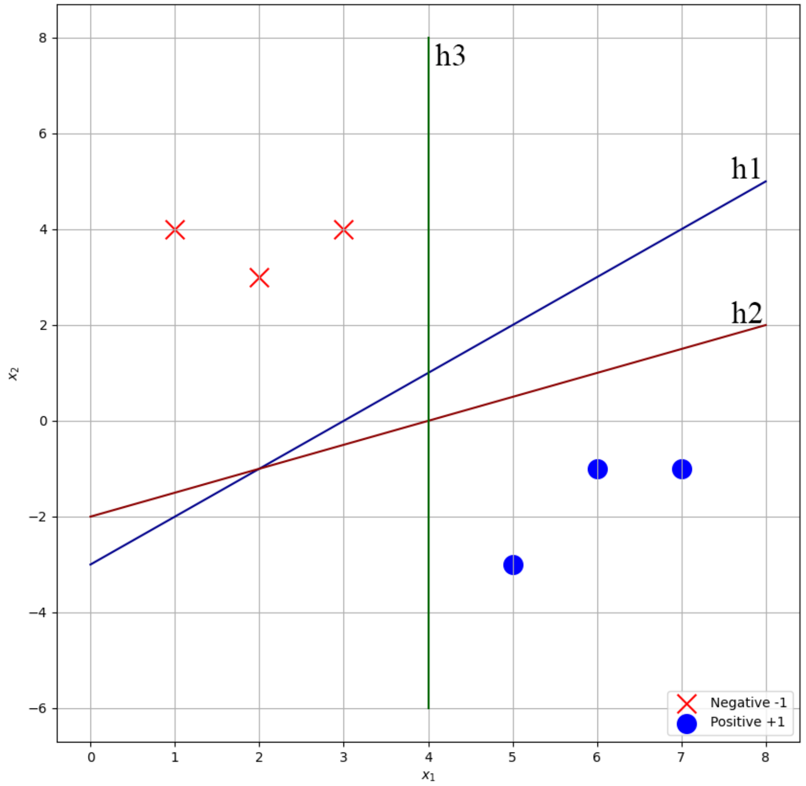
(1*7) + (-1*-1) + (-3) = 5 > 0 → $y_{predict} = +1$

In this example, we use X1 and X2 to predict Y.

Support Vector Machine

❖Hard Margin SVM: Optimal Hyperplane

Changing the value of **W** and **b** gives us different hyperplanes



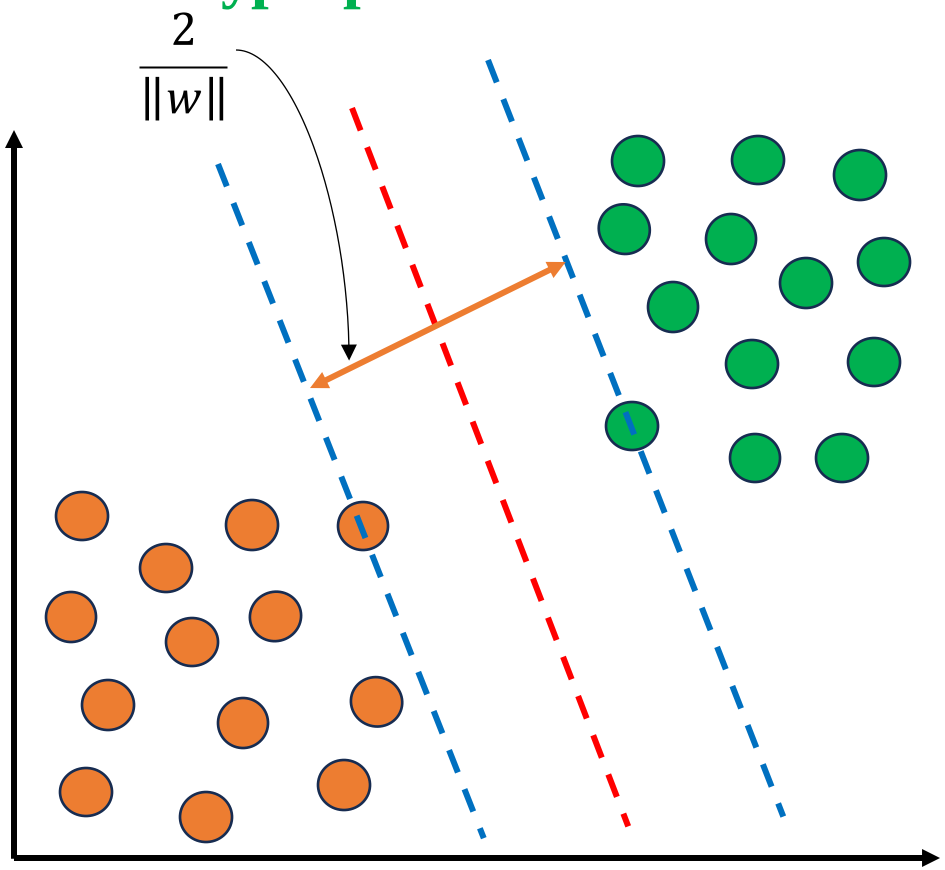
Find **W** and **b** such that Hyperplane best separates the data

Hyperplane = $\boldsymbol{w} \cdot x + \boldsymbol{b} = 0$

Support Vector Machine

❖Hard Margin SVM: Optimal Hyperplane

Find **W** and **b** such that Hyperplane best separates the data



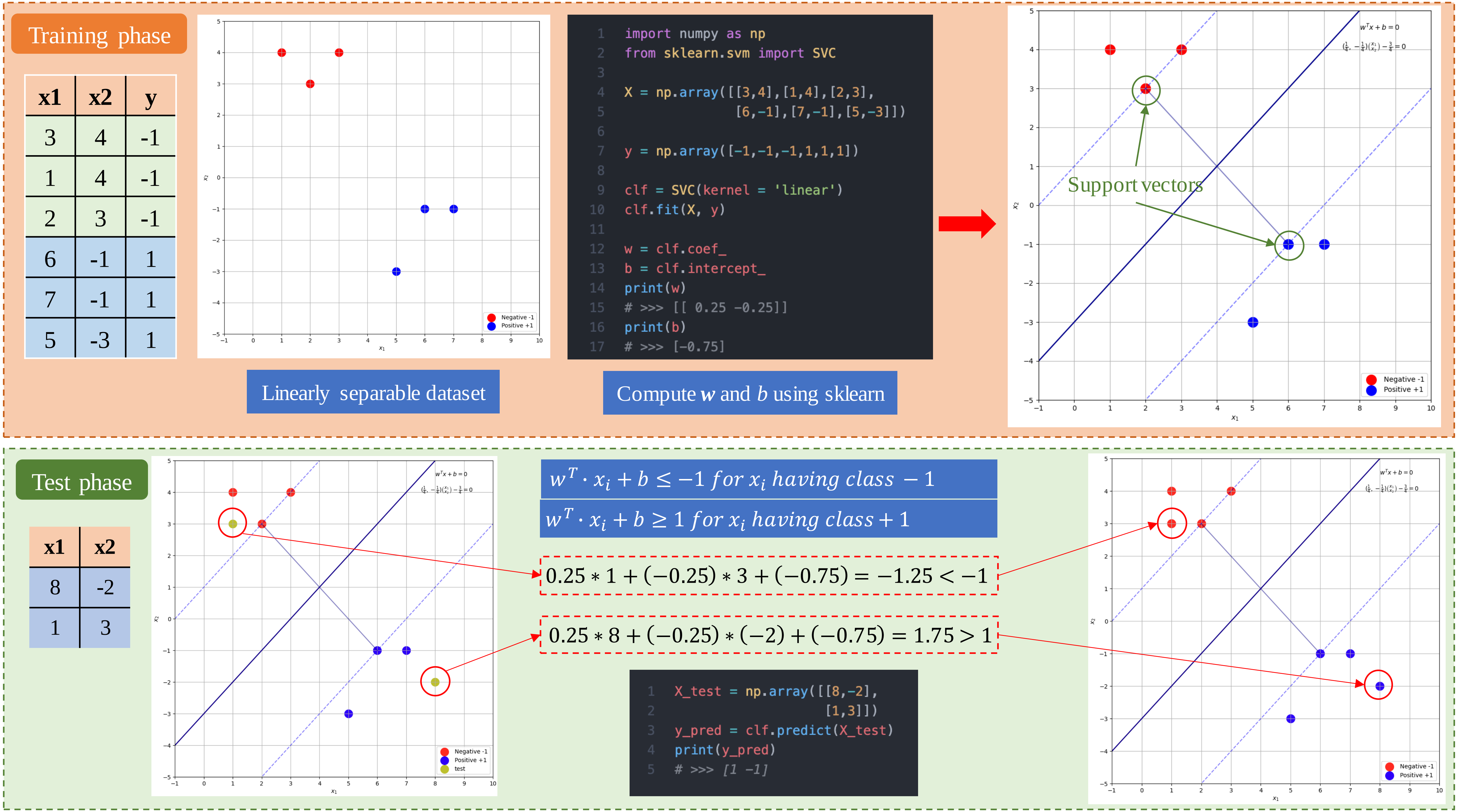
Objective: $\max_{w,b} \frac{2}{\|w\|}$

Objective: $\min_{w,b} \frac{1}{2}\|w\|^2$

Constraints:

- For y=1: $w \cdot x + b \geq 1$
- For y=-1: $w \cdot x + b \leq -1$

=> $y(w \cdot x + b) \geq 1$



Training phase
x1 | x2 | y
3 | 4 | -1
1 | 4 | -1
2 | 3 | -1
6 | -1 | 1
7 | -1 | 1
5 | -3 | 1
Negative -1
Positive +1
Linearly separable dataset
import numpy as np
from sklearn.svm import SVC
X = np.array([[3,4],[1,4],[2,3],
[6,-1],[7,-1],[5,-3]])
y = np.array([-1,-1,-1,1,1,1])
clf = SVC(kernel = 'linear')
clf.fit(X, y)
w = clf.coef_
b = clf.intercept_
print(w)
# >>> [[0.25 -0.25]]
print(b)
# >>> [-0.75]
Compute w and b using sklearn
$w^Tx + b = 0$
Support vectors
Test phase
x1 | x2
8 | -2
1 | 3
test
$w^T \cdot x_i + b \leq -1$ for x_i having class -1
$w^T \cdot x_i + b \geq 1$ for x_i having class $+1$
$0.25 * 1 + (-0.25) * 3 + (-0.75) = -1.25 < -1$
$0.25 * 8 + (-0.25) * (-2) + (-0.75) = 1.75 > 1$
X_test = np.array([[8,-2],
[1,3]])
y_pred = clf.predict(X_test)
print(y_pred)
# >>> [1 -1]

Support Vector Machine

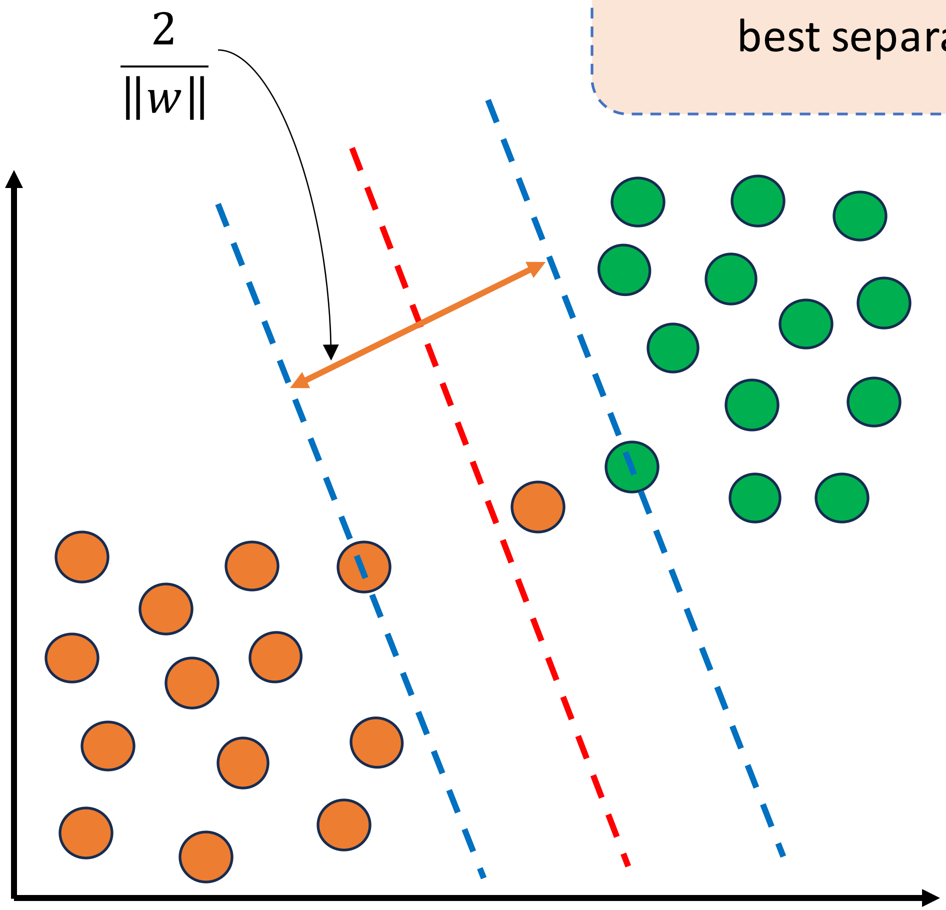
❖Soft Margin SVM: Optimal Hyperplane

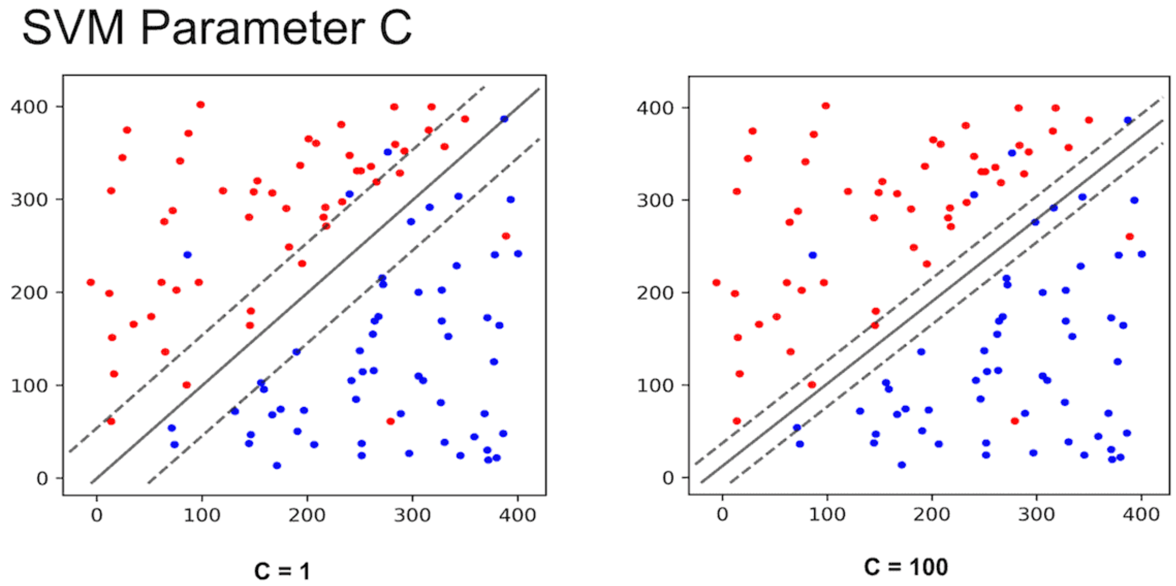
Find **W** and **b** such that Hyperplane best separates the data

Objective: $\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\zeta_i$

Constraint: $y_i(w \cdot x_i + b) \geq 1 - \zeta_i$

$(\zeta_i \geq 0, i = 1, \ldots, m)$

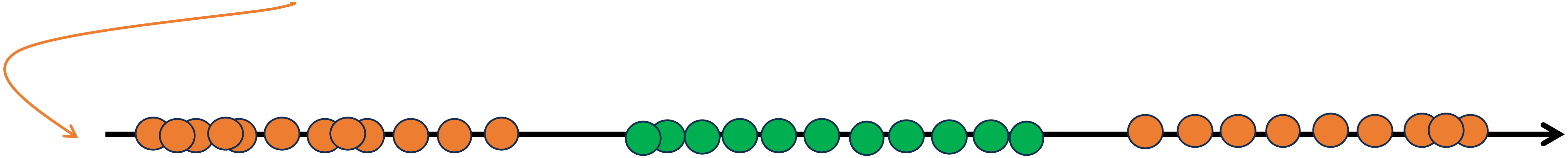




Support Vector Machine

❖ SVC Problem

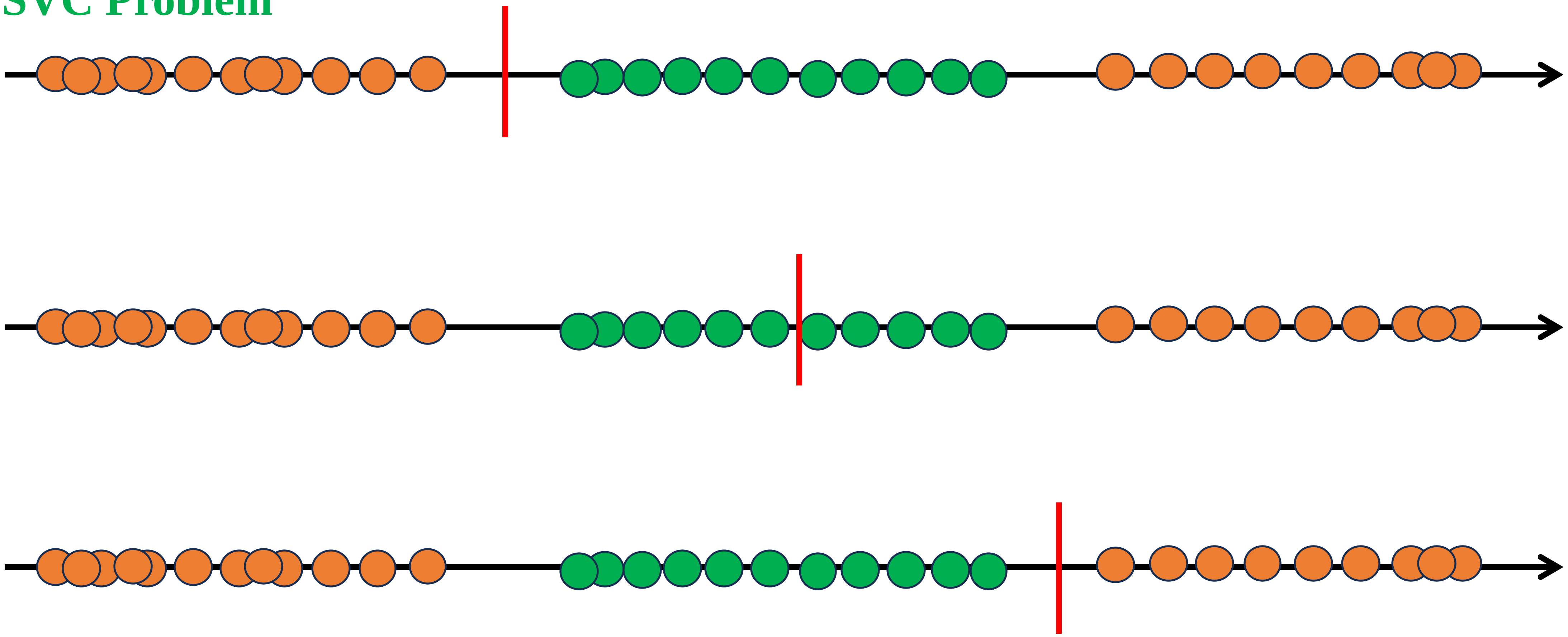
1-Dimensional Space



Can SVC handle this kind of data?

Support Vector Machine

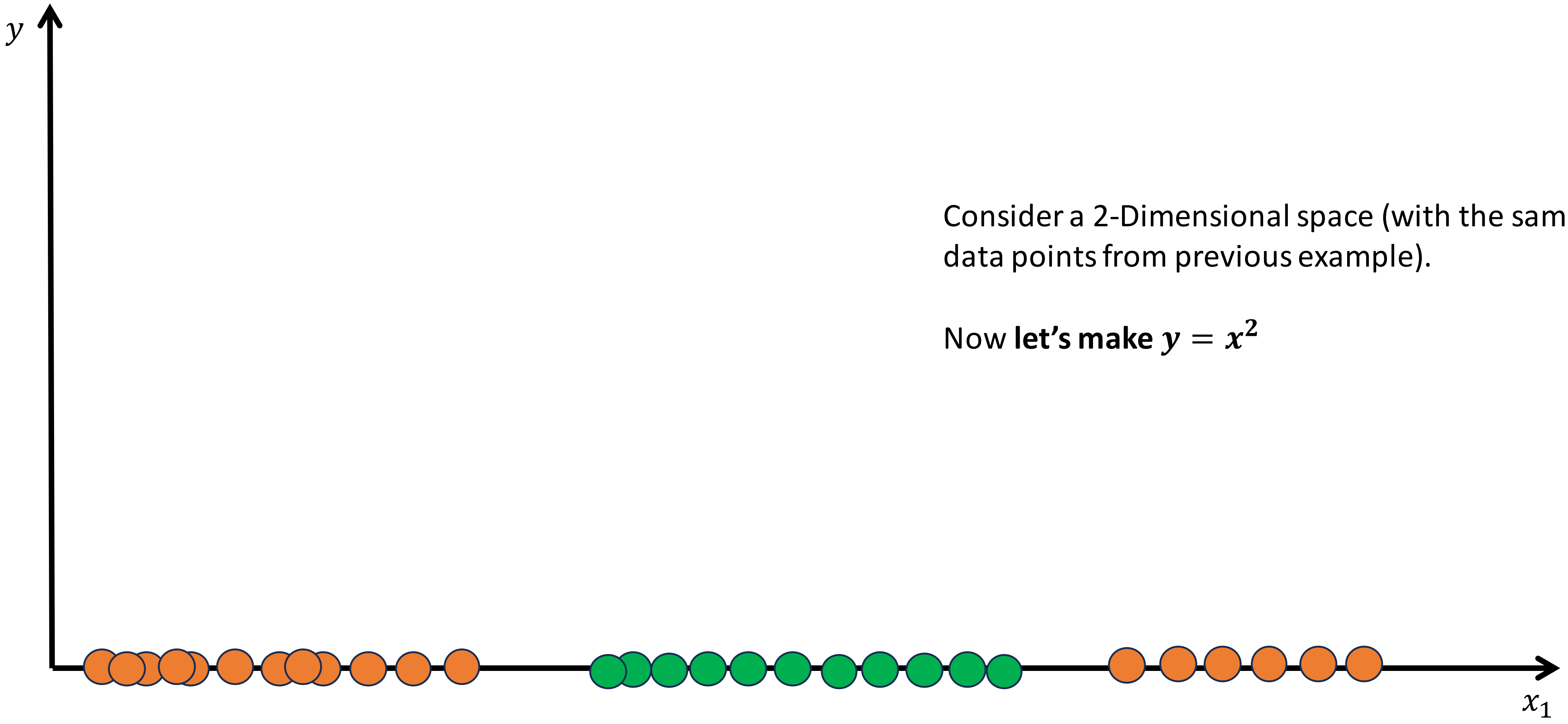
❖ SVC Problem



In general, it is hard for Hard/Soft Margin Classifier to handle this kind of data

Support Vector Machine

❖ Kernel SVM Idea

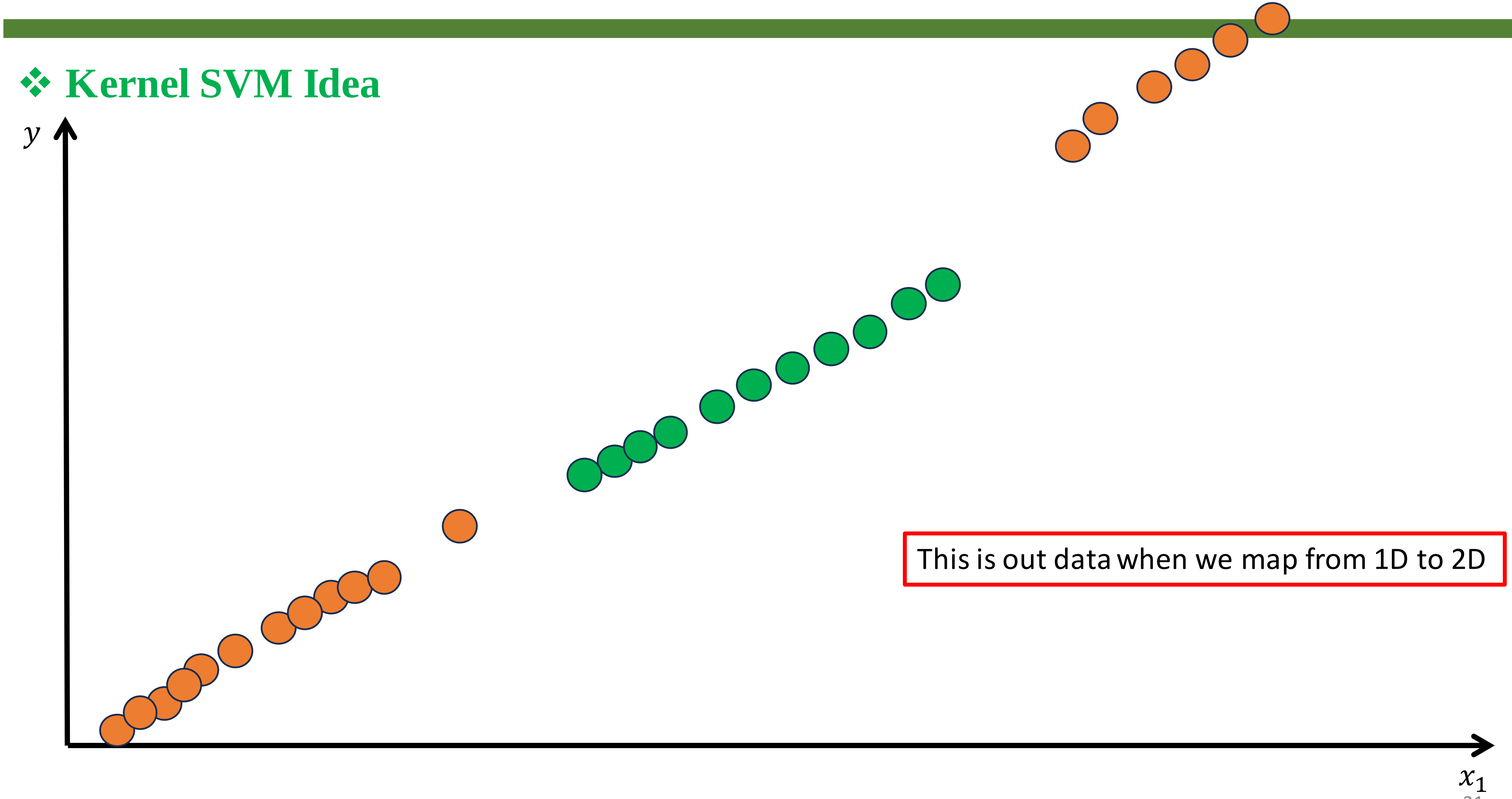


Consider a 2-Dimensional space (with the same data points from previous example).

Now **let's make** $\boldsymbol{y = x^2}$

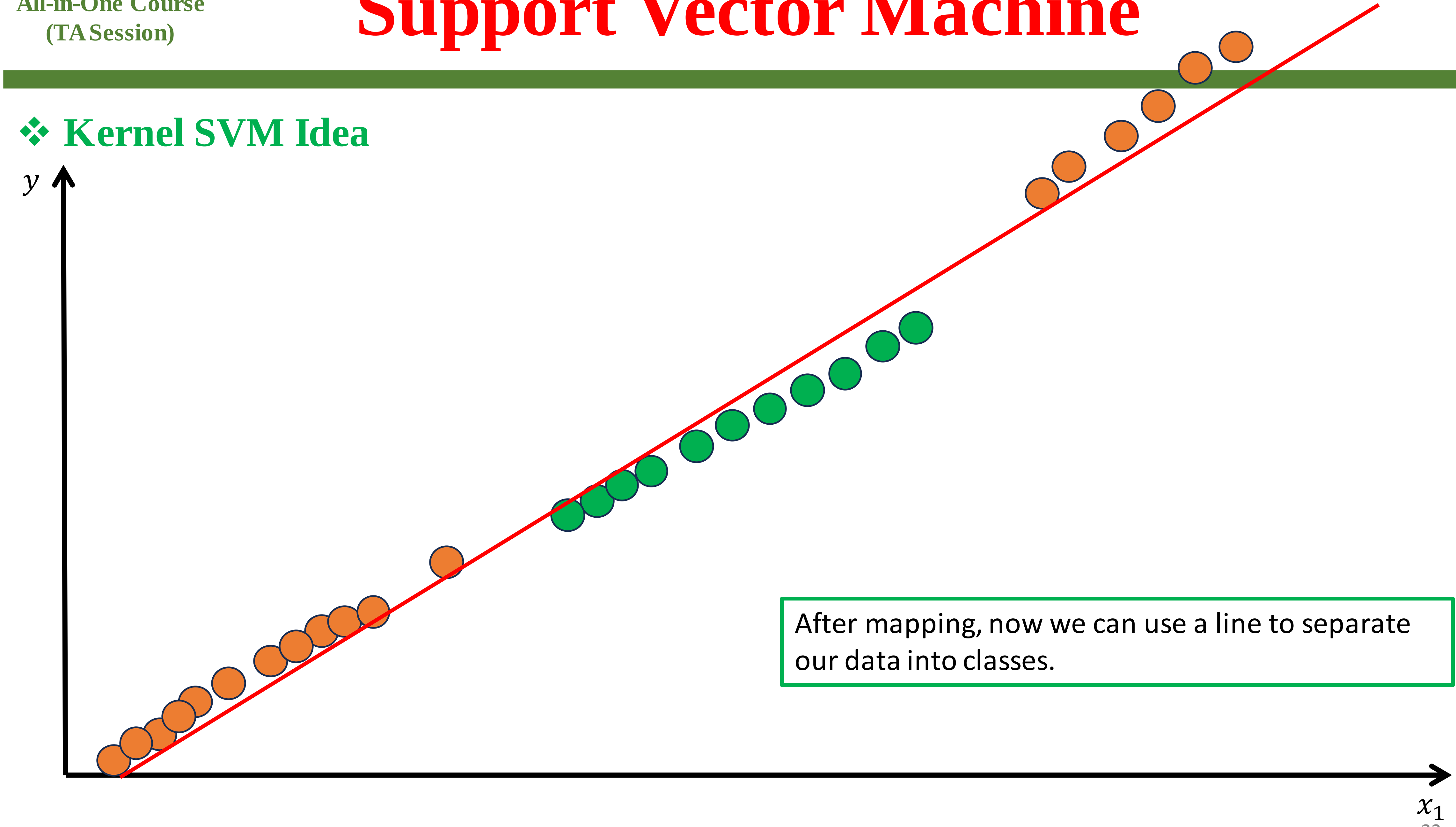
Support Vector Machine

❖ Kernel SVM Idea



Support Vector Machine

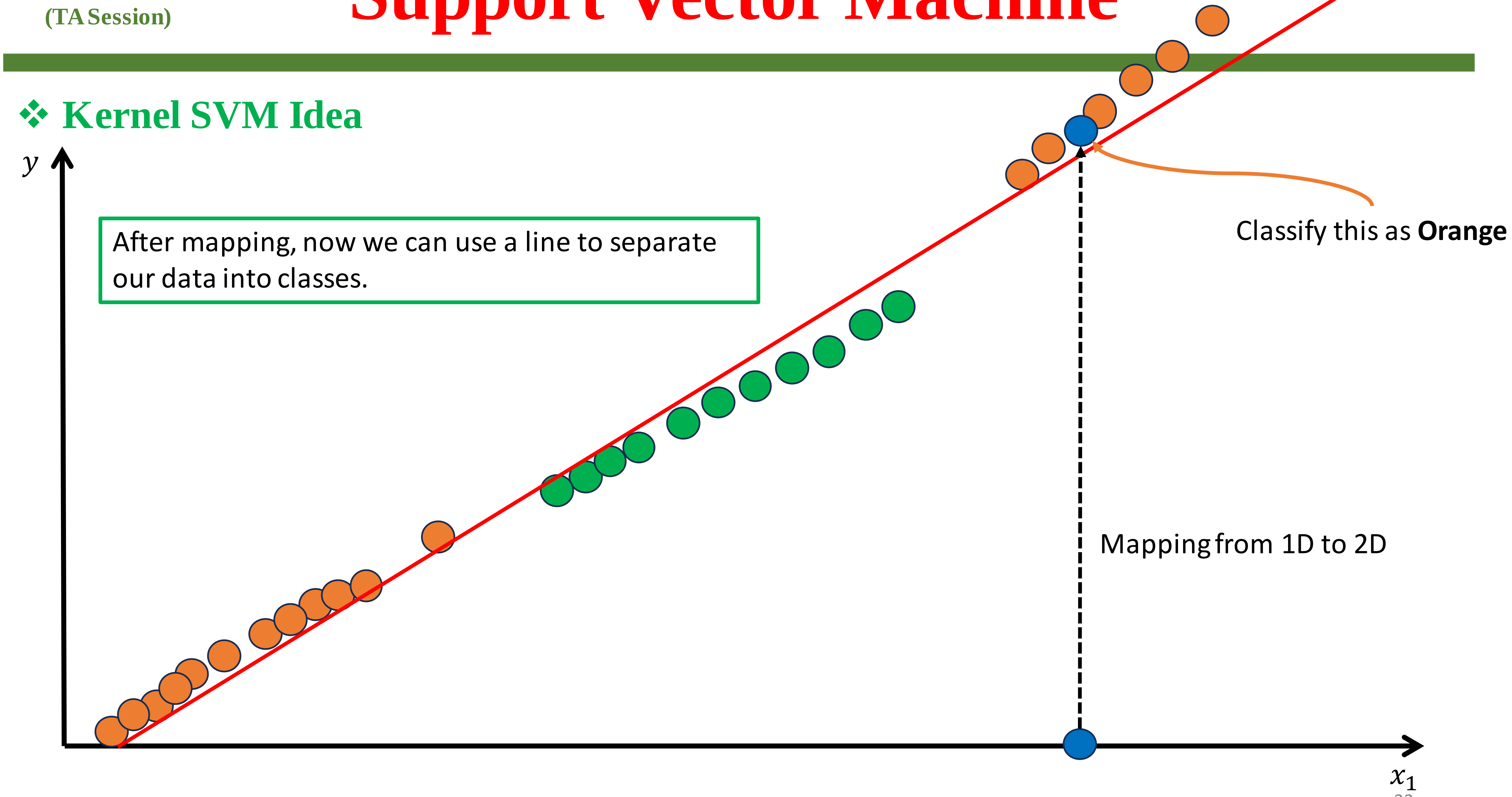
❖ Kernel SVM Idea



After mapping, now we can use a line to separate our data into classes.

Support Vector Machine

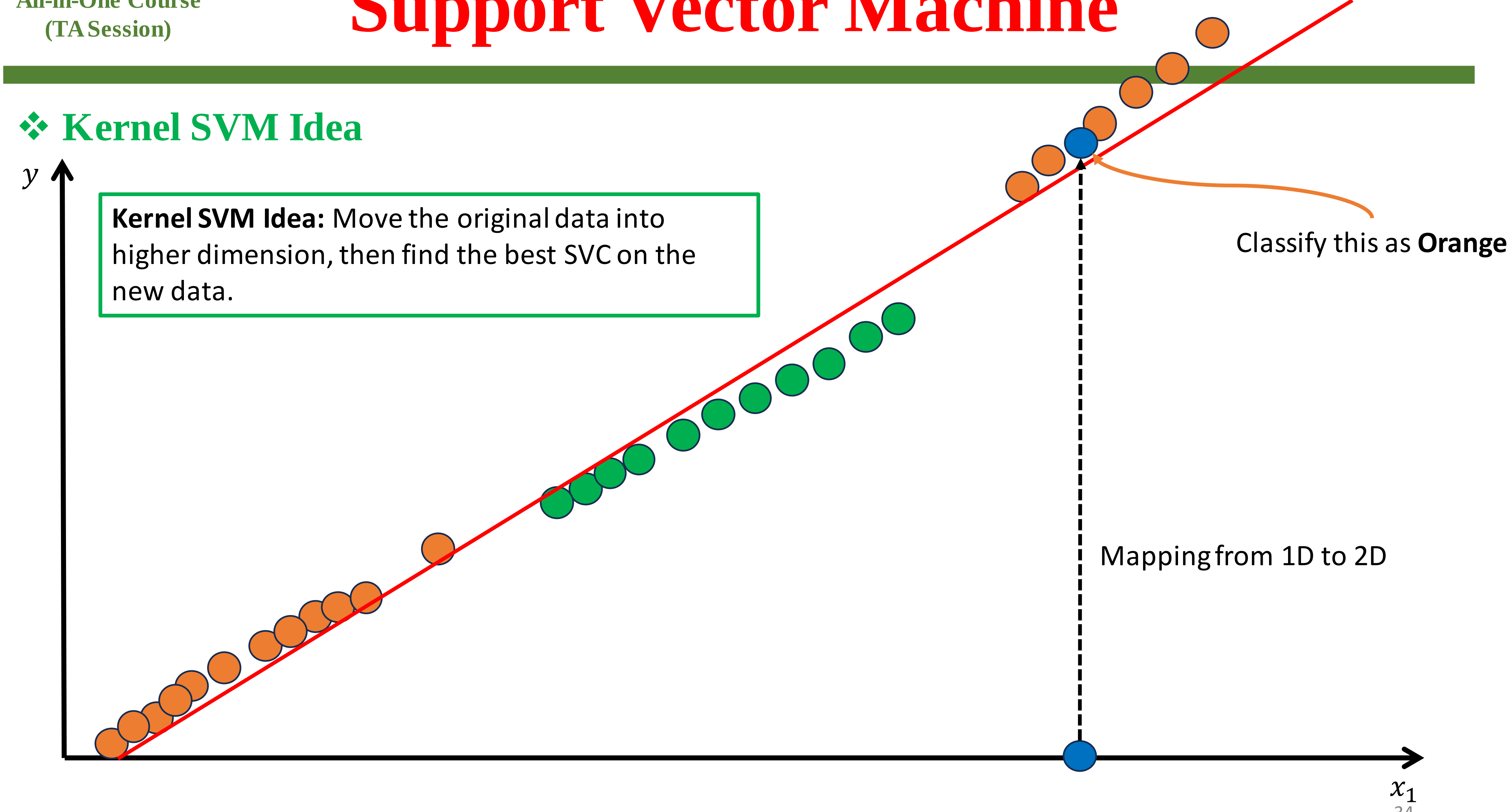
❖ Kernel SVM Idea



Support Vector Machine

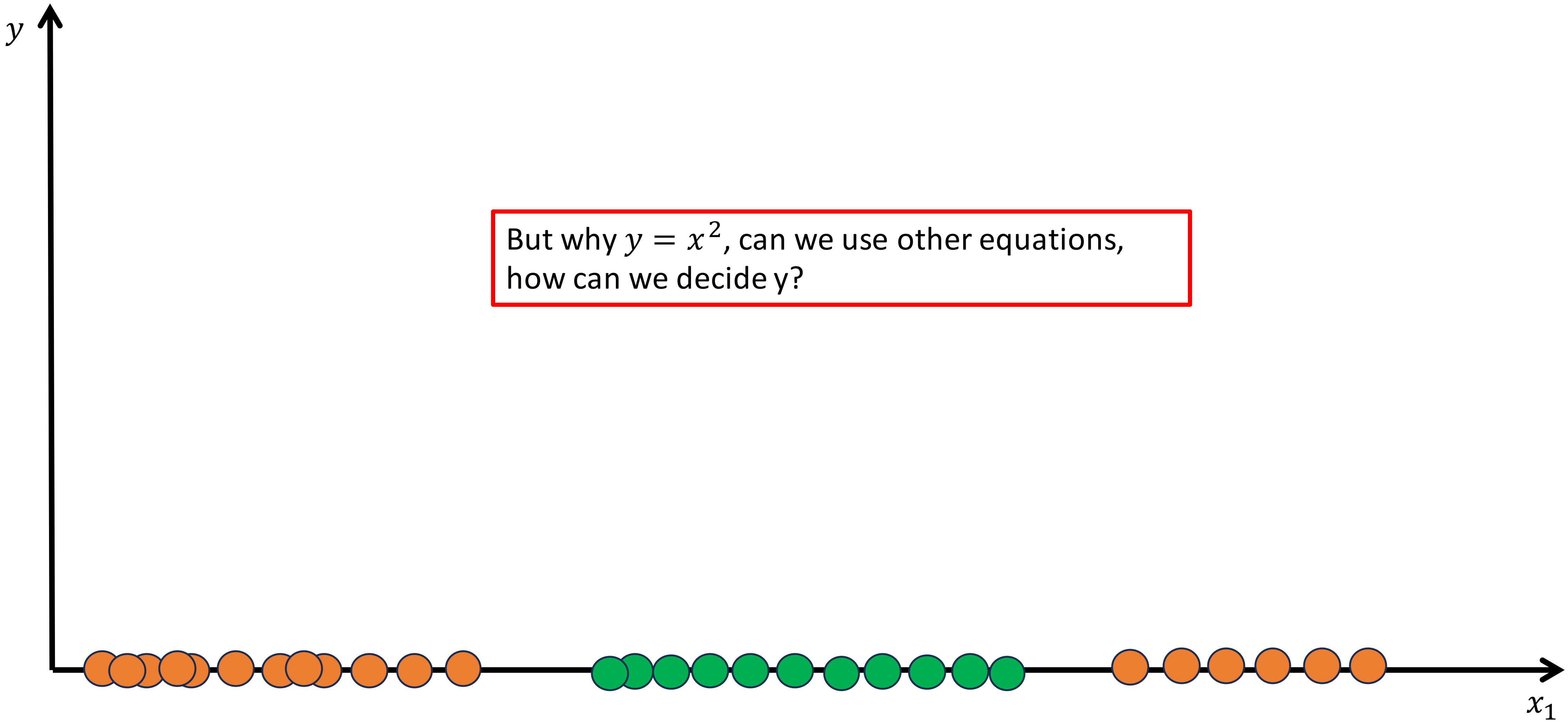
❖ Kernel SVM Idea

Kernel SVM Idea: Move the original data into higher dimension, then find the best SVC on the new data.



Support Vector Machine

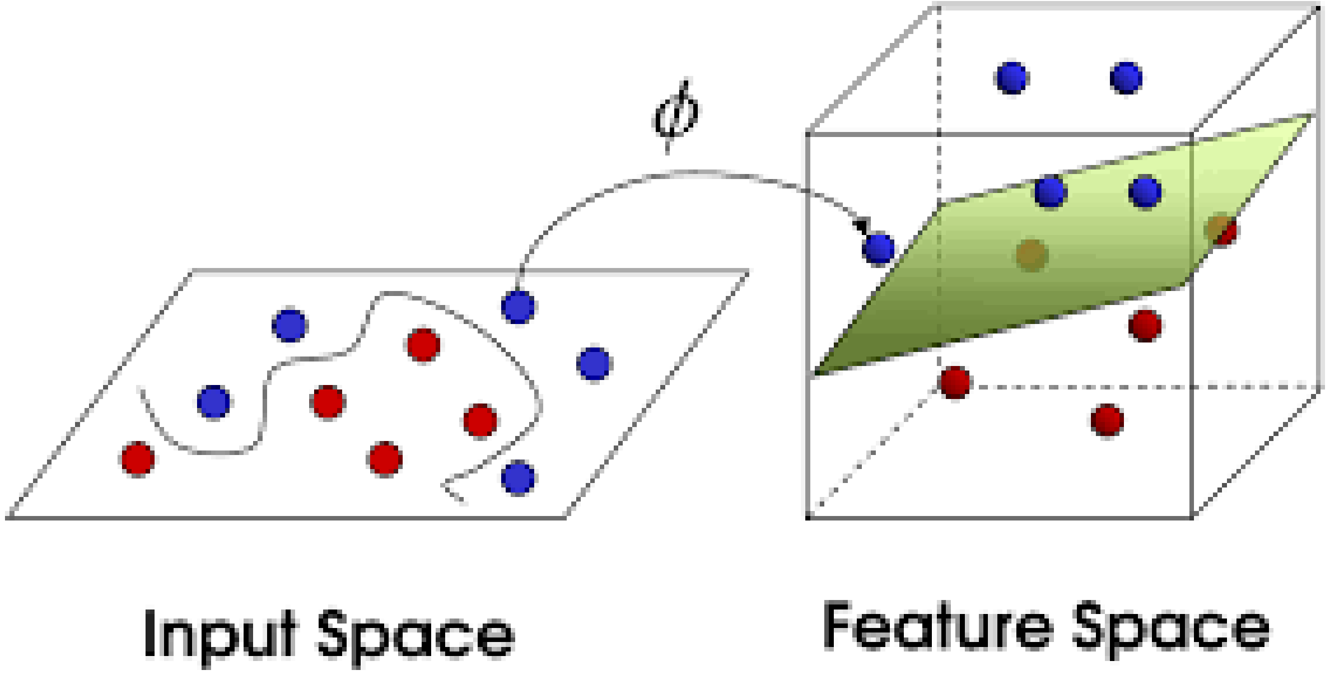
❖ Kernel SVM Idea



But why $y = x^2$, can we use other equations, how can we decide y?

Support Vector Machine

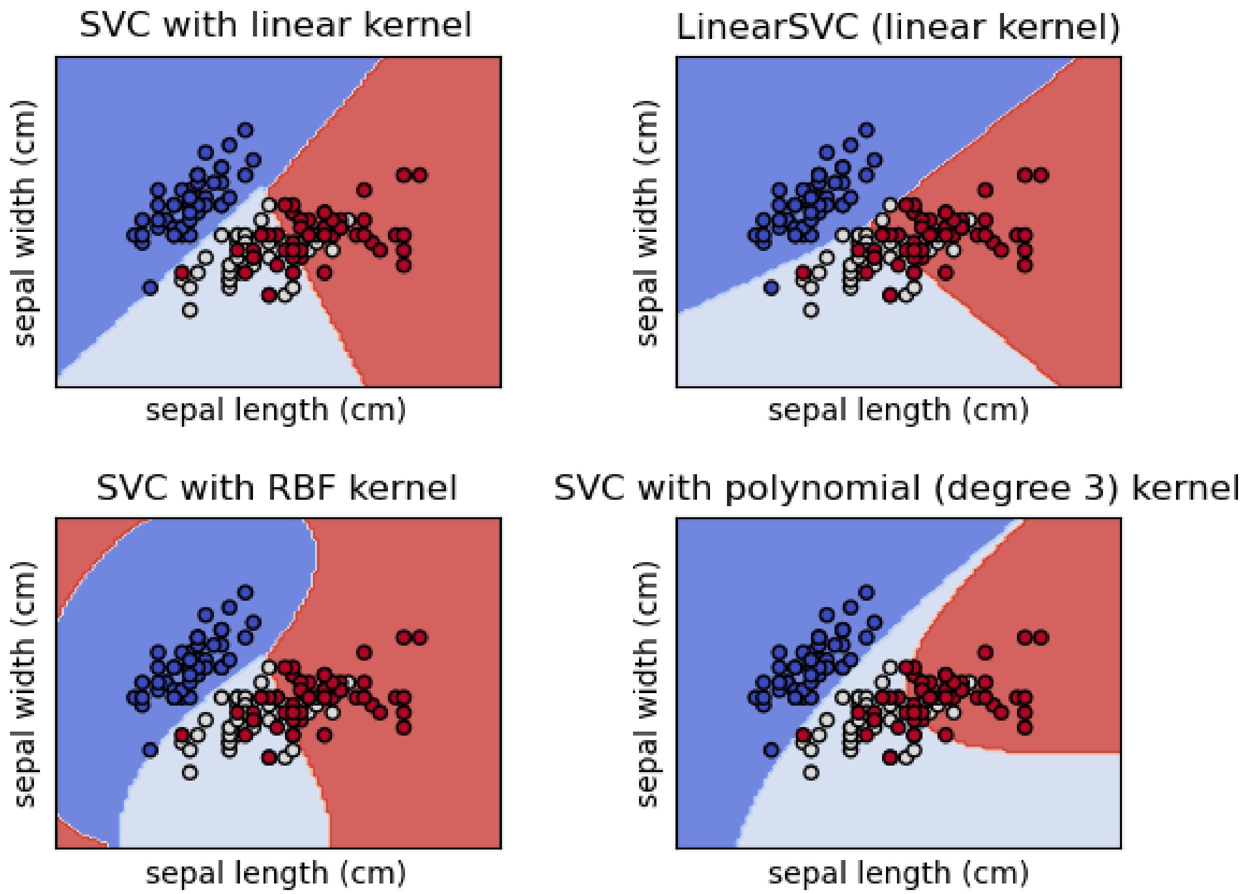
❖ Kernel



To decide the y, or to decide SVC in higher dimensions, we use **Kernel Functions.**

Support Vector Machine

❖ Type of kernels



Different results from different kernels using sklearn

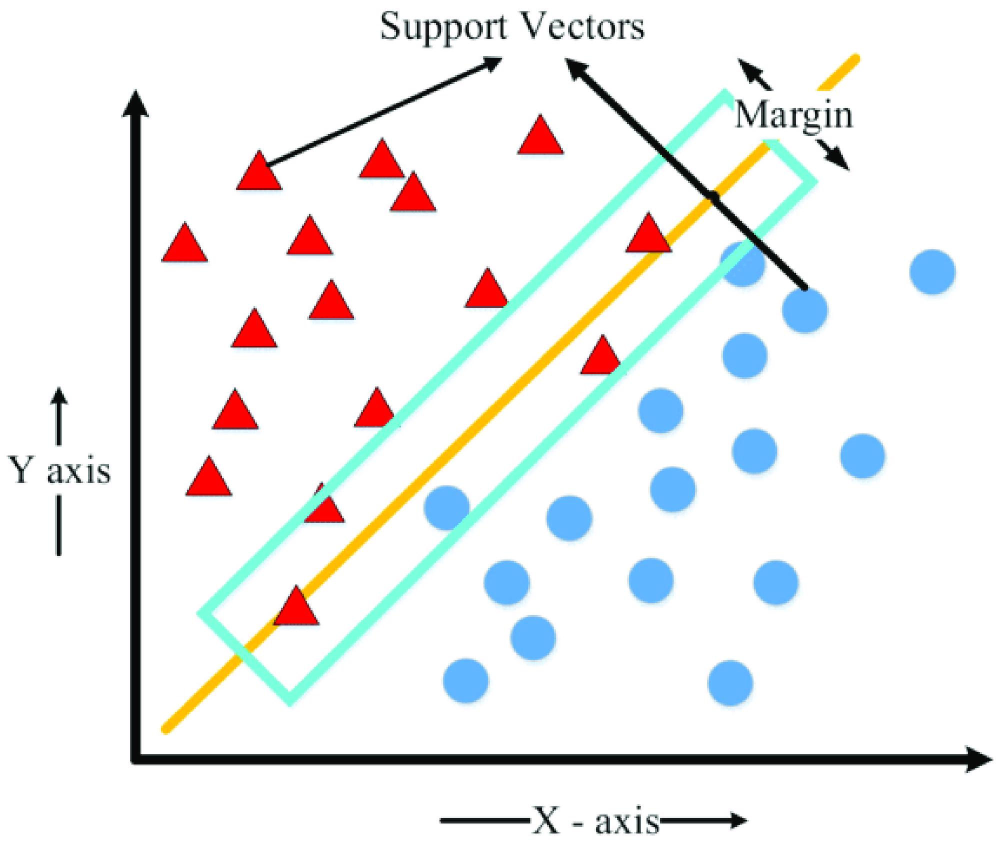
In general, we have some kernel types:

- Linear
- Polynomial
- Radial Basis Function (RBF)
- Sigmoid

Code Exercises

❖ Introduction

Description: Build a Support Vector Classifier for Breast Cancer Recurrence Classification and a Support Vector Regression for Auto Insurance Prediction.



Code Examples

❖ BCR Prediction Step 1: Import libraries

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

from sklearn.svm import SVC
from sklearn.preprocessing import (
    StandardScaler,
    LabelEncoder,
    OneHotEncoder,
    OrdinalEncoder
)
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Code Examples

❖ BCR Prediction Step 2: Read dataset

```python
dataset_path = './breast-cancer.csv'
df = pd.read_csv(
    dataset_path,
    names=[
        'age',
        'meonpause',
        'tumor-size',
        'inv-nodes',
        'node-caps',
        'deg-malig',
        'breast',
        'breast-quad',
        'irradiat',
        'label'
    ]
)
df
```

```
7. Attribute Information:
   1. Class: no-recurrence-events, recurrence-events
   2. age: 10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90-99.
   3. menopause: lt40, ge40, premeno.
   4. tumor-size: 0-4, 5-9, 10-14, 15-19, 20-24, 25-29, 30-34, 35-39, 40-44,
                  45-49, 50-54, 55-59.
   5. inv-nodes: 0-2, 3-5, 6-8, 9-11, 12-14, 15-17, 18-20, 21-23, 24-26,
                 27-29, 30-32, 33-35, 36-39.
   6. node-caps: yes, no.
   7. deg-malig: 1, 2, 3.
   8. breast: left, right.
   9. breast-quad: left-up, left-low, right-up, right-low, central.
  10. irradiat: yes, no.
```

Detail information of the dataset here

Code Examples

❖ BCR Prediction Step 2: Read dataset

	age	meonpause	tumor-size	inv-nodes	node-caps	deg-malig	breast	breast-quad	irradiat	label
0	'40-49'	'premeno'	'15-19'	'0-2'	'yes'	'3'	'right'	'left_up'	'no'	'recurrence-events'
1	'50-59'	'ge40'	'15-19'	'0-2'	'no'	'1'	'right'	'central'	'no'	'no-recurrence-events'
2	'50-59'	'ge40'	'35-39'	'0-2'	'no'	'2'	'left'	'left_low'	'no'	'recurrence-events'
3	'40-49'	'premeno'	'35-39'	'0-2'	'yes'	'3'	'right'	'left_low'	'yes'	'no-recurrence-events'
4	'40-49'	'premeno'	'30-34'	'3-5'	'yes'	'2'	'left'	'right_up'	'no'	'recurrence-events'
...	...	...	...	...	...	...	...	...	...	...
281	'50-59'	'ge40'	'30-34'	'6-8'	'yes'	'2'	'left'	'left_low'	'no'	'no-recurrence-events'
282	'50-59'	'premeno'	'25-29'	'3-5'	'yes'	'2'	'left'	'left_low'	'yes'	'no-recurrence-events'
283	'30-39'	'premeno'	'30-34'	'6-8'	'yes'	'2'	'right'	'right_up'	'no'	'no-recurrence-events'
284	'50-59'	'premeno'	'15-19'	'0-2'	'no'	'2'	'right'	'left_low'	'no'	'no-recurrence-events'
285	'50-59'	'ge40'	'40-44'	'0-2'	'no'	'3'	'left'	'right_up'	'no'	'no-recurrence-events'

286 rows × 10 columns

Code Examples

❖ BCR Prediction Step 3: Dataset Information

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 286 entries, 0 to 285
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   age          286 non-null    object
 1   meonpause    286 non-null    object
 2   tumor-size   286 non-null    object
 3   inv-nodes    286 non-null    object
 4   node-caps    278 non-null    object
 5   deg-malig    286 non-null    object
 6   breast       286 non-null    object
 7   breast-quad  285 non-null    object
 8   irradiat     286 non-null    object
 9   label        286 non-null    object
dtypes: object(10)
memory usage: 22.5+ KB
```

```
df.describe()
```

	age	meonpause	tumor-size	inv-nodes	node-caps	deg-malig
count	286	286	286	286	278	286
unique	6	3	11	7	2	3
top	'50-59'	'premeno'	'30-34'	'0-2'	'no'	'2'
freq	96	150	60	213	222	130

breast	breast-quad	irradiat	label
286	285	286	286
2	5	2	2
'left'	'left_low'	'no'	'no-recurrence-events'
152	110	218	201

Code Examples

❖ BCR Prediction Step 4: Filling missing values

pandas

	Postal Address	Permanent Address
0	New York	Miami
1	NaN	Amsterdam
2	London	London
3	Mumbai	Rajkot
4	NaN	Sydney

	Postal Address	Permanent Address
0	New York	Miami
1	Amsterdam	Amsterdam
2	London	London
3	Mumbai	Rajkot
4	Sydney	Sydney

```
df['node-caps'] = df['node-caps'].fillna(
    df['node-caps'].mode()[0]
)
df['breast-quad'] = df['breast-quad'].fillna(
    df['breast-quad'].mode()[0]
)
```

Fill with the **most appears value**.

X1	X2	X3
1	**A**	**yes**
1	B	**yes**
2	Y	no
5	T	**yes**
1	**A**	**yes**
1	**A**	**yes**

Code Examples

❖ BCR Prediction Step 4: Filling missing values

```
df.describe()
```

	age	meonpause	tumor-size	inv-nodes	node-caps	deg-malig	breast	breast-quad	irradiat	label
count	286	286	286	286	286	286	286	286	286	286
unique	6	3	11	7	2	3	2	5	2	2
top	'50-59'	'premeno'	'30-34'	'0-2'	'no'	'2'	'left'	'left_low'	'no'	'no-recurrence-events'
freq	96	150	60	213	230	130	152	111	218	201

Missing values are filled

Code Examples

❖ BCR Prediction Step 5: Encode categorical features

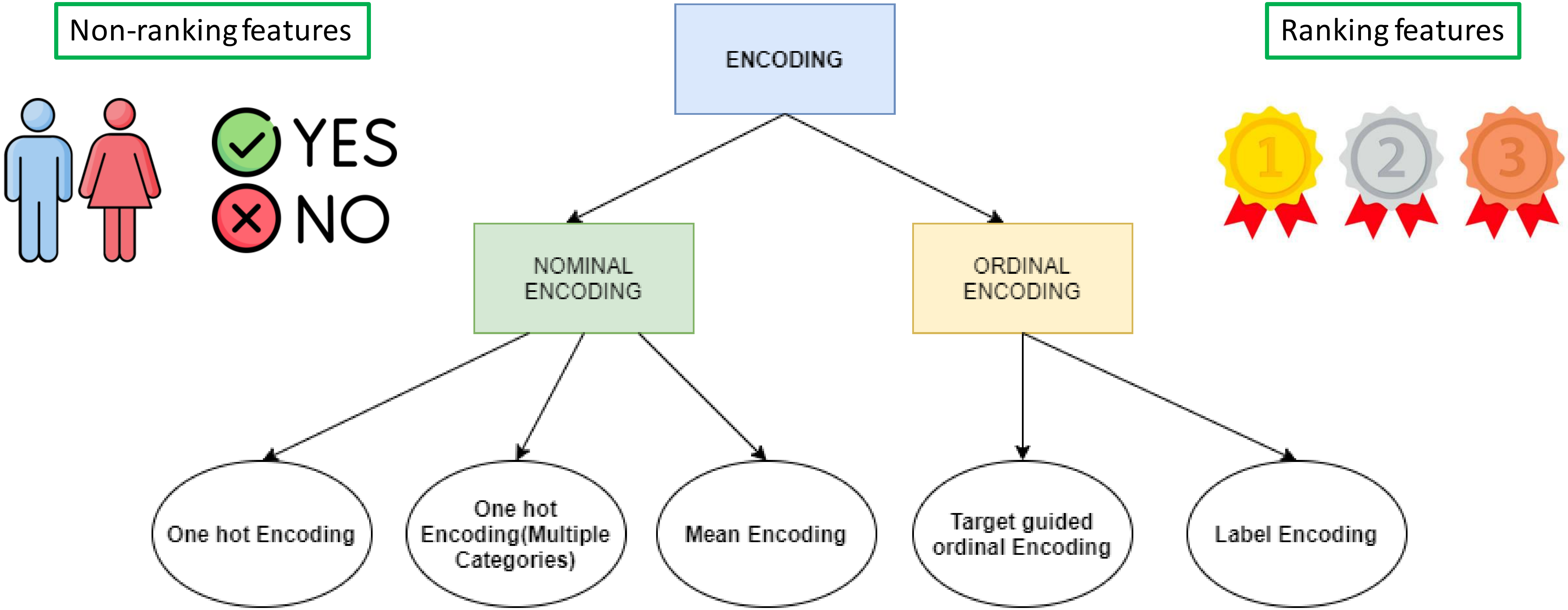
```
for col_name in df.columns:
    n_uniques = df[col_name].unique()
    print(f'Unique values in {col_name}: {n_uniques}')
```

```
Unique values in age: ["'40-49'" "'50-59'" "'60-69'" "'30-39'" "'70-79'" "'20-29'"]
Unique values in meonpause: ["'premeno'" "'ge40'" "'lt40'"]
Unique values in tumor-size: ["'15-19'" "'35-39'" "'30-34'" "'25-29'" "'40-44'" "'10-14'" "'0-4'"
 "'20-24'" "'45-49'" "'50-54'" "'5-9'"]
Unique values in inv-nodes: ["'0-2'" "'3-5'" "'15-17'" "'6-8'" "'9-11'" "'24-26'" "'12-14'"]
Unique values in node-caps: ["'yes'" "'no'"]
Unique values in deg-malig: ["'3'" "'1'" "'2'"]
Unique values in breast: ["'right'" "'left'"]
Unique values in breast-quad: ["'left_up'" "'central'" "'left_low'" "'right_up'" "'right_low'"]
Unique values in irradiat: ["'no'" "'yes'"]
Unique values in label: ["'recurrence-events'" "'no-recurrence-events'"]
```

All features are categorical feature

Code Examples

❖ BCR Prediction Step 5: Encode categorical features



Code Examples

❖ BCR Prediction Step 5: Encode categorical features

```
Unique values in age: ["'40-49'" "'50-59'" "'60-69'" "'30-39'" "'70-79'" "'20-29'"]
Unique values in meonpause: ["'premeno'" "'ge40'" "'lt40'"]
Unique values in tumor-size: ["'15-19'" "'35-39'" "'30-34'" "'25-29'" "'40-44'" "'10-14'" "'0-4'"
 "'20-24'" "'45-49'" "'50-54'" "'5-9'"]
Unique values in inv-nodes: ["'0-2'" "'3-5'" "'15-17'" "'6-8'" "'9-11'" "'24-26'" "'12-14'"]
Unique values in node-caps: ["'yes'" "'no'"]
Unique values in deg-malig: ["'3'" "'1'" "'2'"]
Unique values in breast: ["'right'" "'left'"]
Unique values in breast-quad: ["'left_up'" "'central'" "'left_low'" "'right_up'" "'right_low'"]
Unique values in irradiat: ["'no'" "'yes'"]
Unique values in label: ["'recurrence-events'" "'no-recurrence-events'"]
```

- **Non-ranking features:** ['meonpause', 'node-caps', 'breast', 'breast-quad', 'irradiat’]
- **Ranking features:** ['age', 'tumor-size', 'inv-nodes', 'deg-malig']

Code Examples

❖ BCR Prediction Step 5: Encode categorical features

```python
non_rank_features = [
    'meonpause',
    'node-caps',
    'breast',
    'breast-quad',
    'irradiat'
]
rank_features = [
    'age',
    'tumor-size',
    'inv-nodes',
    'deg-malig'
]

y = df['label']
X = df.drop('label', axis=1)
```

```python
transformer = ColumnTransformer(
    transformers=[
        ("OneHot", OneHotEncoder(drop='first'), non_rank_features),
        ("Ordinal", OrdinalEncoder(), rank_features)
    ],
    remainder='passthrough'
)
X_transformed = transformer.fit_transform(X)
```

For non-ranking features: Apply One-hot Encoding

For ranking features: Apply Ordinal Encoding

Code Examples

❖ BCR Prediction Step 5: Encode categorical features

X1	X2
0	yes
1	no
2	no
3	yes
4	yes
5	yes

Encoding

X1	X2
0	1
1	0
2	0
3	1
4	1
5	1

Ordinal Encoder

X1	X2_Yes	X2_No
0	1	0
1	0	1
2	0	1
3	1	0
4	1	0
5	1	0

Onehot Encoder

Code Examples

❖ BCR Prediction Step 5: Encode categorical features

```
onehot_features = transformer.named_transformers_['OneHot'].get_feature_names_out(non_rank_features)
all_features = onehot_features.tolist() + rank_features

X_encoded = pd.DataFrame(
    X_transformed,
    columns=all_features
)
```

	meonpause_'lt40'	meonpause_'premeno'	node-caps_'yes'	breast_'right'	breast-quad_'left_low'	breast-quad_'left_up'	breast-quad_'right_low'	breast-quad_'right_up'	irradiat_'yes'	age	tumor-size	inv-nodes	deg-malig
0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	2.0	2.0	0.0	2.0
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	3.0	2.0	0.0	0.0
2	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	3.0	6.0	0.0	1.0
3	0.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	2.0	6.0	0.0	2.0
4	0.0	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	2.0	5.0	4.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
281	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	3.0	5.0	5.0	1.0
282	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	3.0	4.0	4.0	1.0
283	0.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	5.0	5.0	1.0
284	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	3.0	2.0	0.0	1.0
285	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	3.0	7.0	0.0	2.0

286 rows × 13 columns

Code Examples

❖ BCR Prediction Step 6: Encode label

X1	Y
0	yes
1	no
2	no
3	yes
4	yes
5	yes

→

Label Encoder	
‘no’	0
‘yes’	1

→

X1	Y
0	1
1	0
2	0
3	1
4	1
5	1

Code Implementation

❖ BCR Prediction Step 7: Normalization

Using sklearn.preprocessing.StandardScaler() to scale all values in dataset.

```
normalizer = StandardScaler()
X_normalized = normalizer.fit_transform(
    X_encoded
)
```

$$z = \frac{x_i - \mu}{\sigma}$$

```
X_normalized
```

```
array([[-0.15839699,  0.95219046,  2.02660871, ..., -0.96065727,
        -0.55562266,  1.29056424],
       [-0.15839699, -1.05021006, -0.49343516, ..., -0.96065727,
        -0.55562266, -1.42341644],
       [-0.15839699, -1.05021006, -0.49343516, ...,  0.90204089,
        -0.55562266, -0.0664261 ],
       ...,
       [-0.15839699,  0.95219046,  2.02660871, ...,  0.43636635,
         2.03245684, -0.0664261 ],
       [-0.15839699,  0.95219046, -0.49343516, ..., -0.96065727,
        -0.55562266, -0.0664261 ],
       [-0.15839699, -1.05021006, -0.49343516, ...,  1.36771543,
        -0.55562266,  1.29056424]])
```

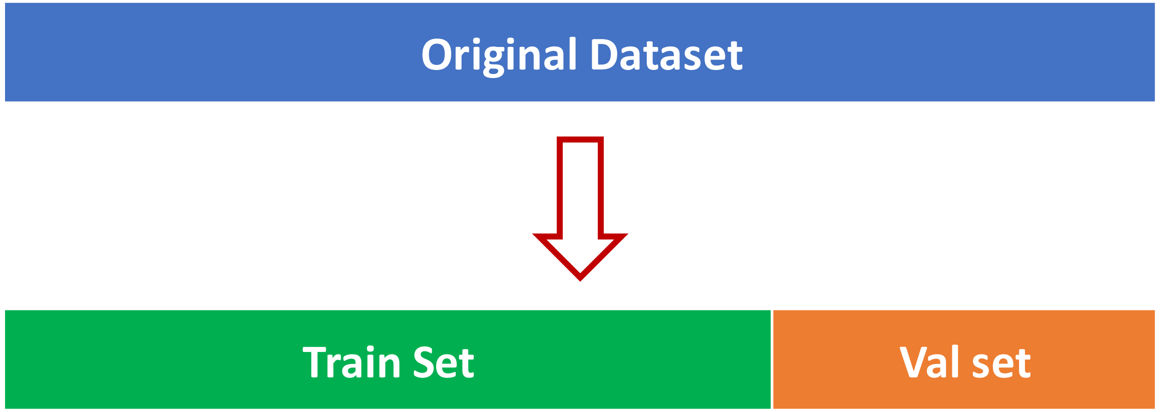
Code Examples

❖ BCR Prediction Step 8: Split train, val set

```python
test_size = 0.3
random_state = 1
is_shuffle = True
X_train, X_val, y_train, y_val = train_test_split(
    X_normalized, y_encoded,
    test_size=test_size,
    random_state=random_state,
    shuffle=is_shuffle
)
```

```python
print(f'Number of training samples: {X_train.shape[0]}')
print(f'Number of val samples: {X_val.shape[0]}')
```

```
Number of training samples: 200
Number of val samples: 86
```



Code Examples

❖ BCR Prediction Step 8: Train and evaluate SVM model

```python
classifier = SVC(
    random_state=random_state
)
classifier.fit(X_train, y_train)
```

```
▼ SVC
SVC(random_state=1)
```

```python
y_pred = classifier.predict(X_val)
scores = accuracy_score(y_pred, y_val)

print('Evaluation results on validation set:')
print(f'Accuracy: {scores}')
```

```
Evaluation results on validation set:
Accuracy: 0.686046511627907
```

Code Examples

❖ Auto Insurance Prediction Step 1: Import libraries

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

from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error
)
```

Code Examples

❖ Auto Insurance Prediction Step 2: Read dataset

```python
dataset_path = './auto-insurance.csv'
df = pd.read_csv(
    dataset_path,
    names=[
        'n_claims',
        'total_payment'
    ]
)
```

```
Auto Insurance in Sweden

In the following data
X = number of claims
Y = total payment for all the claims in thousands of Swedish Kronor
for geographical zones in Sweden
```

Detail information of the dataset here

Code Examples

❖ Auto Insurance Prediction Step 3: Dataset information

	n_claims	total_payment
0	108	392.5
1	19	46.2
2	13	15.7
3	124	422.2
4	40	119.4
...	...	...
58	9	87.4
59	31	209.8
60	14	95.5
61	53	244.6
62	26	187.5

63 rows × 2 columns

```
df.describe()
```

	n_claims	total_payment
count	63.000000	63.000000
mean	22.904762	98.187302
std	23.351946	87.327553
min	0.000000	0.000000
25%	7.500000	38.850000
50%	14.000000	73.400000
75%	29.000000	140.000000
max	124.000000	422.200000

```
df.info()
```

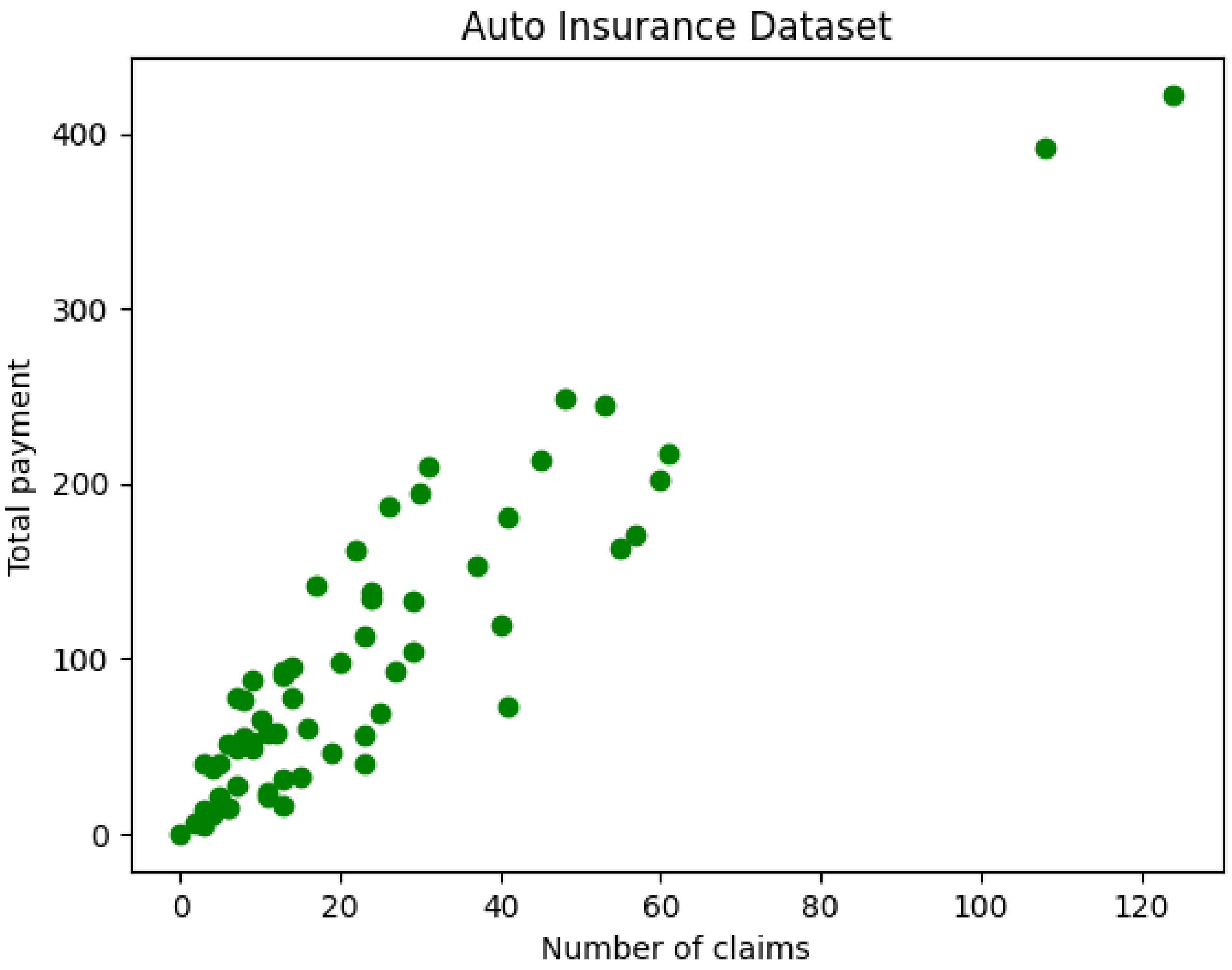
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 63 entries, 0 to 62
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   n_claims       63 non-null     int64  
 1   total_payment  63 non-null     float64
dtypes: float64(1), int64(1)
memory usage: 1.1 KB
```

Code Examples

❖ Auto Insurance Prediction Step 4: Dataset Visualization

	n_claims	total_payment
0	108	392.5
1	19	46.2
2	13	15.7
3	124	422.2
4	40	119.4
...	...	...
58	9	87.4
59	31	209.8
60	14	95.5
61	53	244.6
62	26	187.5

63 rows × 2 columns



Code Implementation

❖ Auto Insurance Prediction Step 5: Normalization

Using sklearn.preprocessing.StandardScaler() to scale all values in dataset.

```
normalizer = StandardScaler()
df_normalized = normalizer.fit_transform(
    df
)
```

$$z = \frac{x_i - \mu}{\sigma}$$

```
df_normalized
```

```
array([[ 3.67330185e+00,  3.39728625e+00],
       [-1.68556660e-01, -6.00095564e-01],
       [-4.27558357e-01, -9.52160668e-01],
       [ 4.36397304e+00,  3.74011686e+00],
       [ 7.37949280e-01,  2.44860684e-01],
       [ 1.47178742e+00,  8.39331269e-01],
       [ 4.11113805e-03, -4.76584200e-01],
       [-3.84391408e-01, -2.38795966e-01],
       [ 9.53784027e-01,  1.33683966e+00],
       [-5.57059206e-01, -3.79622008e-01],
       [-7.72893953e-01, -8.92136453e-01],
       [ 1.08328488e+00,  1.73045999e+00],
       [-5.13892256e-01, -8.62124346e-01],
       [ 4.11113805e-03, -6.76280144e-01],
       [-6.86560054e-01, -5.70083457e-01],
```

Code Examples

❖ Auto Insurance Prediction Step 6: Split X, y and create train, val dataset

```
X, y = df_normalized[:, 0], df_normalized[:, 1]
X = X.reshape(-1, 1)
```

```
test_size = 0.3
random_state = 1
is_shuffle = True
X_train, X_val, y_train, y_val = train_test_split(
    X, y,
    test_size=test_size,
    random_state=random_state,
    shuffle=is_shuffle
)
```

```
print(f'Number of training samples: {X_train.shape[0]}')
print(f'Number of val samples: {X_val.shape[0]}')
```

```
Number of training samples: 44
Number of val samples: 19
```

Code Examples

❖ Auto Insurance Prediction Step 7: Train and evaluate SVM model

```python
regressor = SVR()
regressor.fit(
    X_train,
    y_train
)
```

```
▾ SVR
SVR()
```

```python
y_pred = regressor.predict(X_val)
mae = mean_absolute_error(y_pred, y_val)
mse = mean_squared_error(y_pred, y_val)

print('Evaluation results on validation set:')
print(f'Mean Absolute Error: {mae}')
print(f'Mean Squared Error: {mse}')
```

```
Evaluation results on validation set:
Mean Absolute Error: 0.4549655045116023
Mean Squared Error: 0.5406791138567528
```

Question

?


Thank you!