Support Vector Machine (SVM) 2

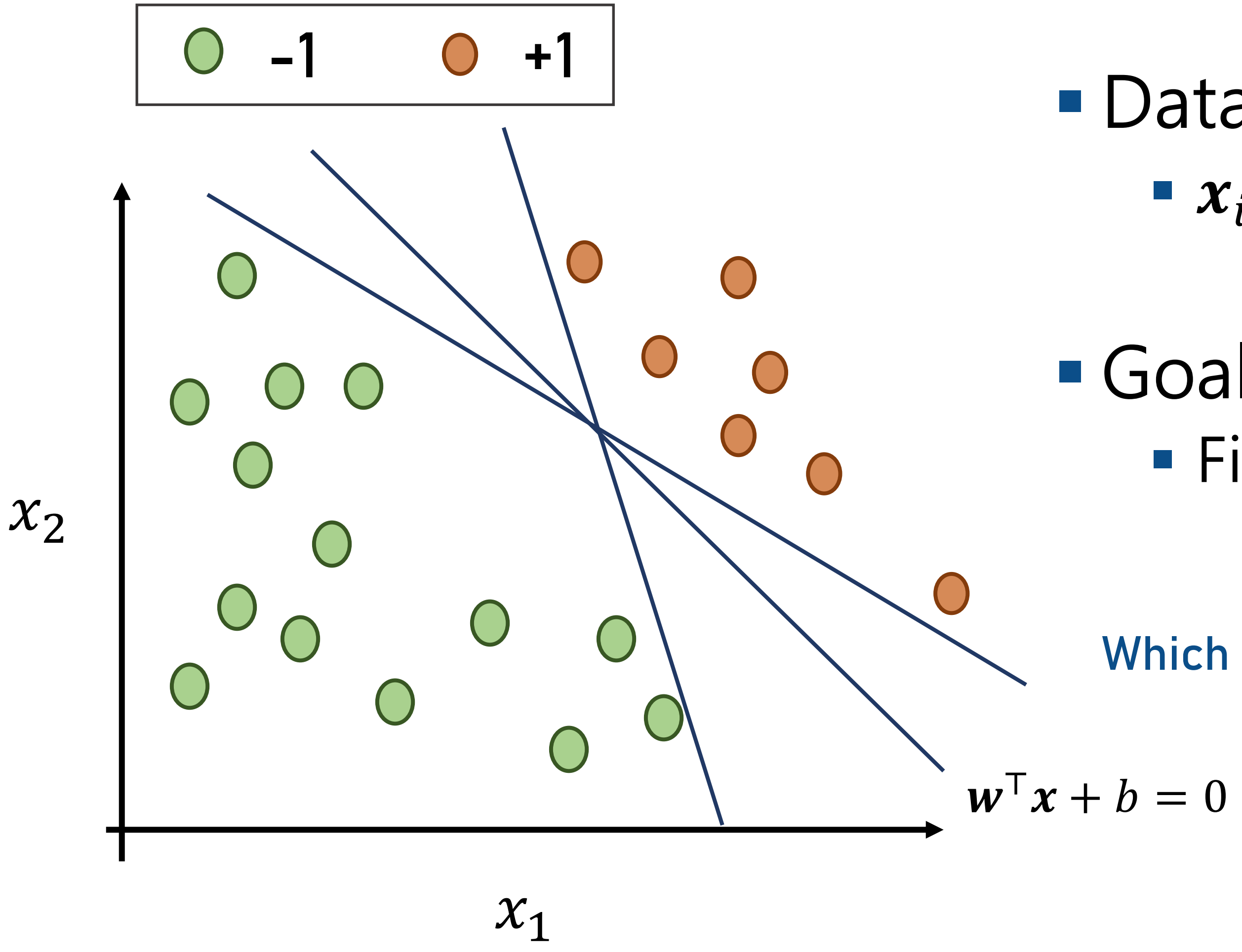
한양대학교 ERICA
소프트웨어융합대학
COLLEGE OF COMPUTING

인공지능학과
Department of
Artificial Intelligence

정 우 환 (whjung@hanyang.ac.kr)

Fall 2021

Linear SVM

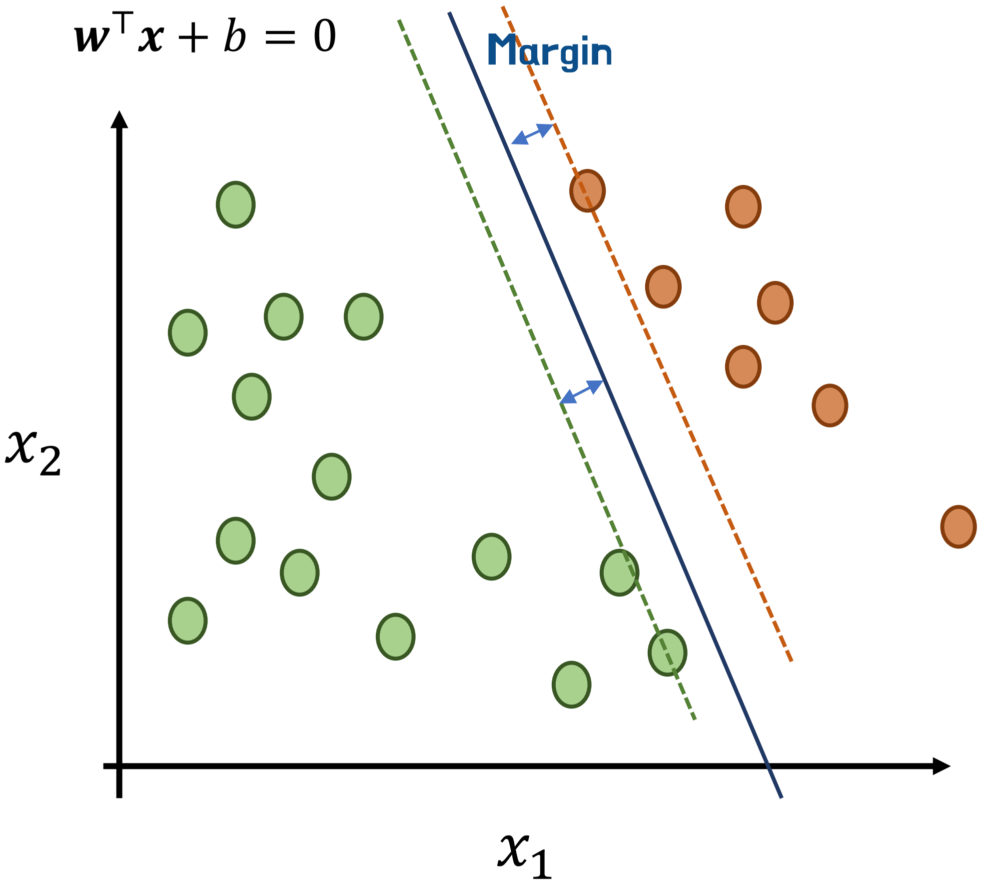


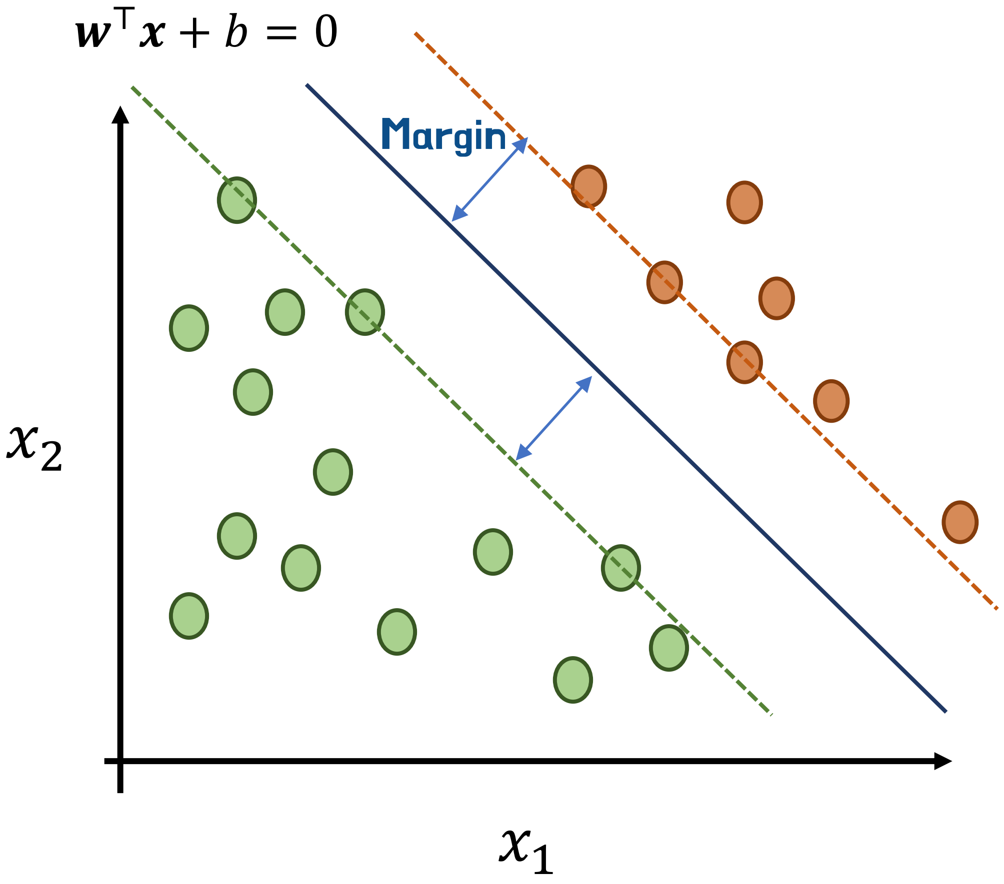
- Data: $< \boldsymbol{x}_i, y_i >$ for $i = 1, \dots n$
  - $\boldsymbol{x}_i \in \mathbb{R}^d$, $y_i \in \{-1, +1\}$
- Goal
  - Finding a good separating hyperplane

Which is best?

Margin

Maximizing margin over the training set
= Minimizing generalization error





Constrained Optimization Problem

$$\underset{\boldsymbol{w},b}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

$$subject\ to\ y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1, i = 1,2,\dots,n$$

- Learnable parameter는 $\boldsymbol{w}, b$
- Margin $\frac{1}{\|\boldsymbol{w}\|_2}$ 을 최대화
- Constraint 만족하면 training data를 완벽하게 separating하게 됨

Original formulation

- 직관적인 formulation
- Solution구하기 어려움

$$\underset{\boldsymbol{w},b}{\text{minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

$$subject\ to\ y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1, i = 1,2,\ldots,n$$

Linear SVM classifier

$$\boldsymbol{x} \longrightarrow \boxed{f} \longrightarrow \hat{y}$$

$$\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{w}^*, b^*) \ = \ sign({\boldsymbol{w}^*}^\top \boldsymbol{x} + b^*)$$

where $\boldsymbol{w}^* = \sum_{i=1}^{n} \alpha_i^* y_i \boldsymbol{x}_i$

Lagrangian dual

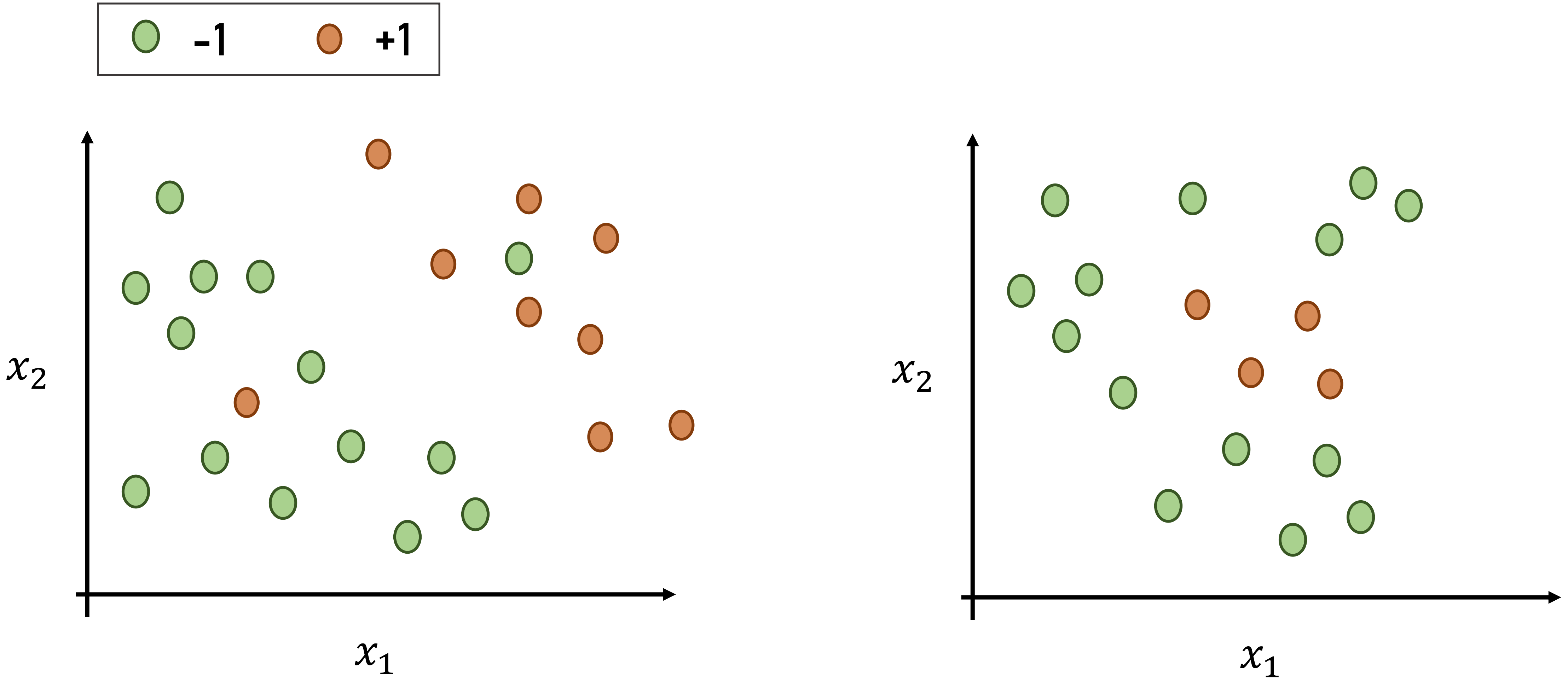
- Quadratic programming formulation
- Convex optimization을 통해 풀 수 있음

$$\underset{\alpha}{\text{minimize}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j {\boldsymbol{x}_i}^\top \boldsymbol{x}_j$$

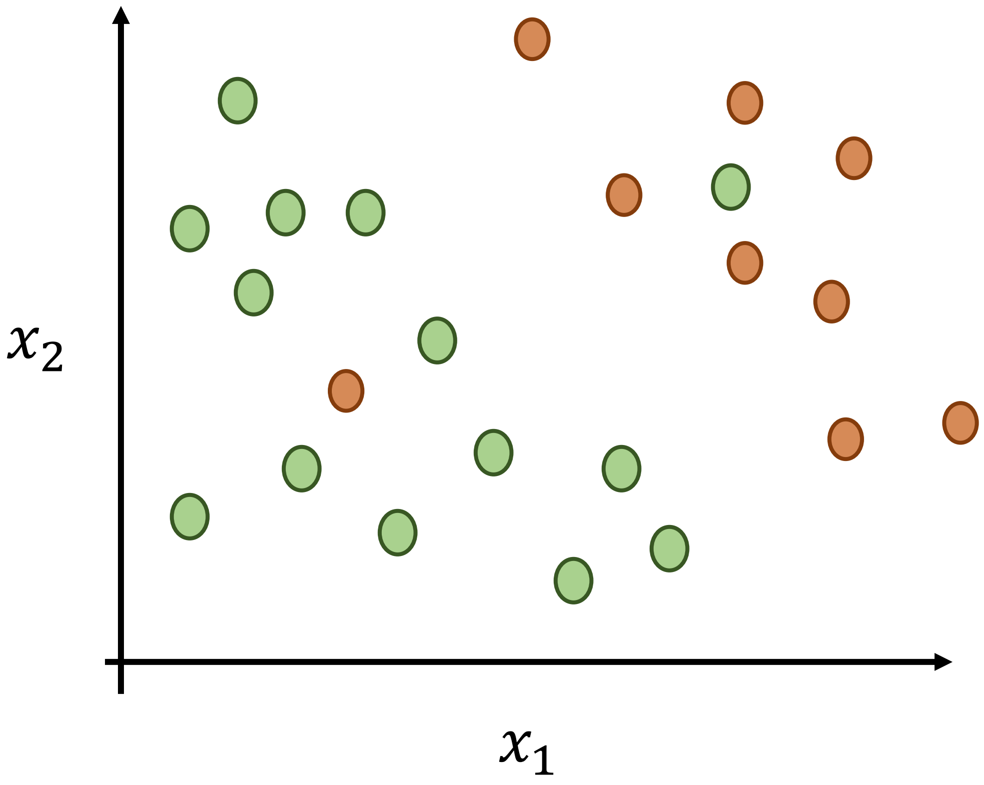
$$subject\ to\ \alpha_i \geq 0, i = 1,2,\ldots,n$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

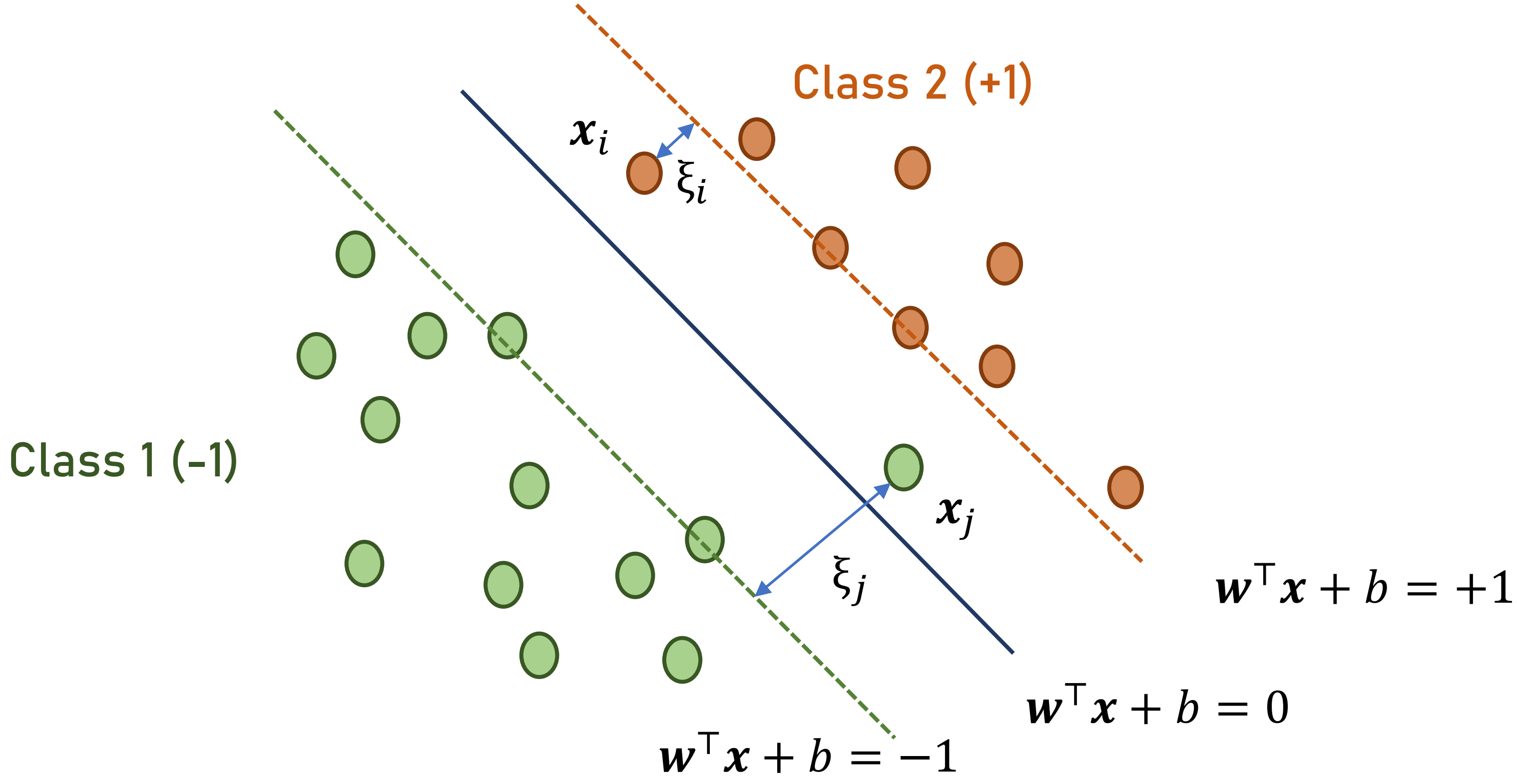
Linearly Non-separable Data



Soft Margin SVM



Soft Margin



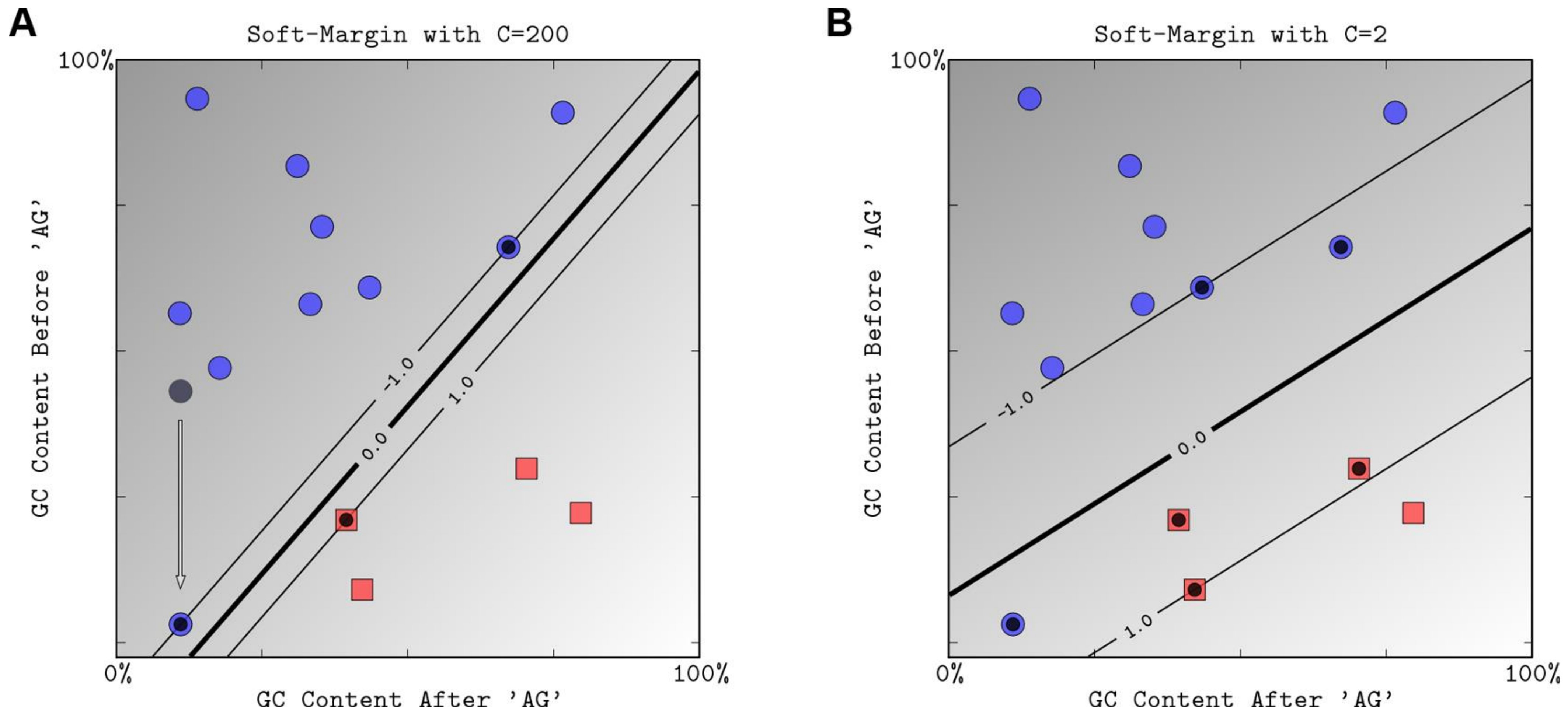
Class 2 (+1)
$\boldsymbol{x}_i$
ξ_i
Class 1 (-1)
$\boldsymbol{x}_j$
ξ_j
$\boldsymbol{w}^\top \boldsymbol{x} + b = +1$
$\boldsymbol{w}^\top \boldsymbol{x} + b = 0$
$\boldsymbol{w}^\top \boldsymbol{x} + b = -1$

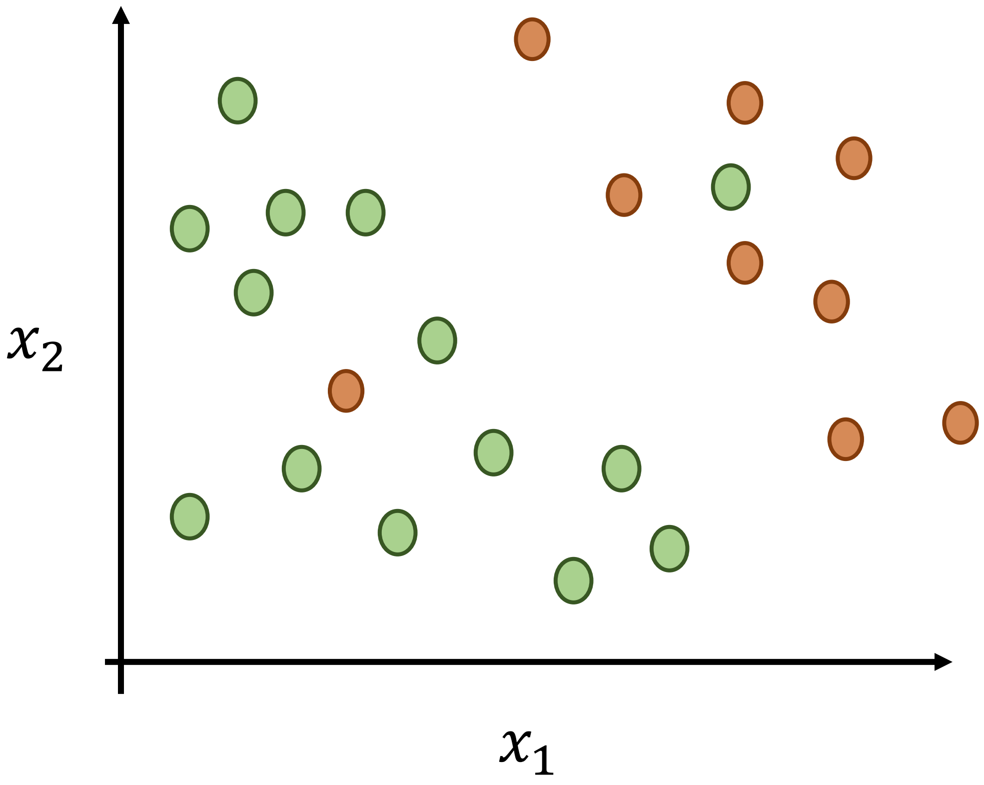
Optimization Problem

$$
\begin{aligned}
\underset{\boldsymbol{w},b}{\text{minimize}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i \\
\textit{subject to} \quad & y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i \qquad \text{For } i = 1,2,\ldots,n \\
& \xi_i \geq 0 \qquad \text{For } i = 1,2,\ldots,n
\end{aligned}
$$

- $C\sum_{i=1}^{n}\xi_i$: 예외의 최소화
  - C를 활용해 허용할 training error를 결정
  - C ↑: training error를 적게허용
  - C ↓: training error를 많이허용
- Linearly separable 하지 않더라도 해가 존재

Soft-margin for Linearly Separable Problem

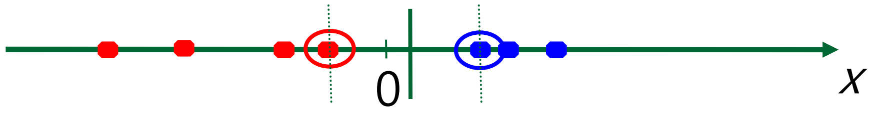




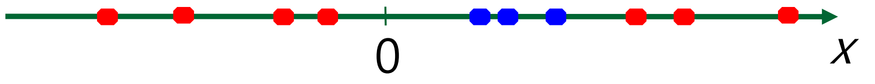
Kernel Trick

Non-linearly Separable Problems

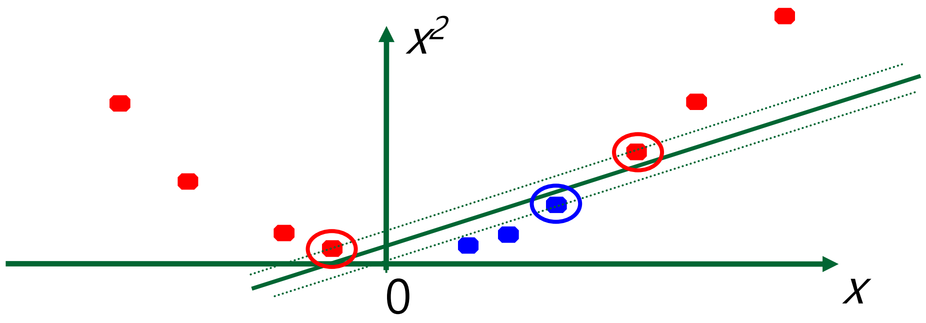
- Datasets that are linearly separable with noise work out great:



- But what are we going to do if the dataset is just too hard?

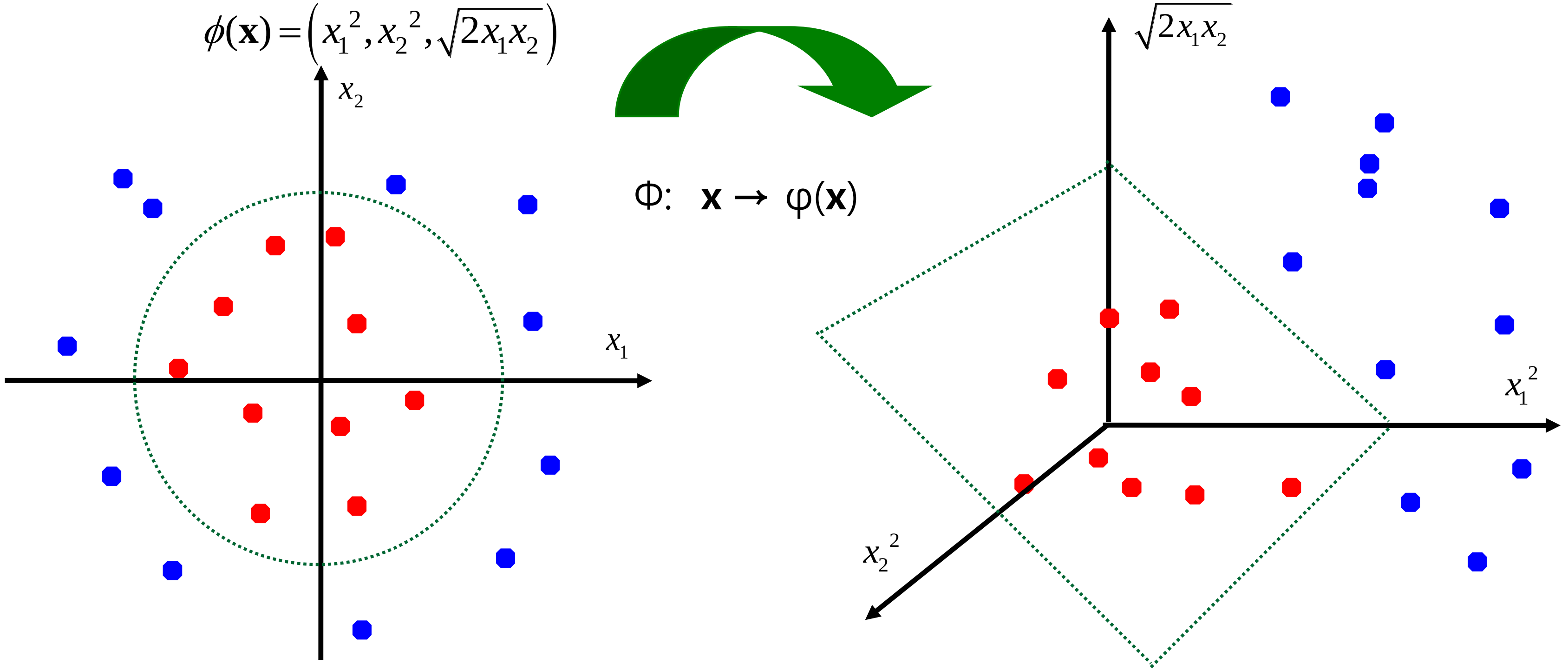


- How about… mapping data to a higher-dimensional space:



Non-linearly Separable Problems

- General idea: the original input space(x) can be mapped to some higher-dimensional feature space(φ(x)) where the training set is separable:



If data are mapped into higher a space of sufficiently high dimension,
then they will in general be linearly separable;
N data points are in general separable in a space of N-1 dimensions or more!!!

Kernel Mapping

Linear SVM formulation (Lagrangian dual)

$$\underset{\alpha}{\text{minimize}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i{}^{\top} \boldsymbol{x}_j$$

$$subject\ to\ \alpha_i \geq 0, i = 1,2, \dots, n$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

SVM formulation (transformation)

$$\underset{\alpha}{\text{minimize}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(\boldsymbol{x}_i)^{\top} \phi(\boldsymbol{x}_j)$$

$$subject\ to\ \alpha_i \geq 0, i = 1,2, \dots, n$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

Kernel Mapping

SVM formulation (transformation)

$$\underset{\alpha}{\text{minimize}} \; \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(\boldsymbol{x}_i)^\top \phi(\boldsymbol{x}_j)$$

$$subject\ to\ \alpha_i \geq 0, i = 1,2, \dots, n$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

SVM formulation (kernel)

$$\underset{\alpha}{\text{minimize}} \; \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

$$subject\ to\ \alpha_i \geq 0, i = 1,2, \dots, n$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

Kernel Functions

- Linear
  - $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{x}_j$
  - Mapping Φ: $\boldsymbol{x} \rightarrow \varphi(\boldsymbol{x})$, where $\varphi(\boldsymbol{x})$ is $\boldsymbol{x}$ itself
- Polynomial of power p
  - $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{x}_j + 1\right)^P$
- Gaussian (radial-basis function)
  - *Most widely used
  - $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}}$
- Sigmoid (Neural net style)
  - $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh\left(\kappa\, \boldsymbol{x}_i^{\mathsf{T}} \boldsymbol{x}_j - \delta\right)$

Polynomial Kernel

- Example) $p = 4$, d = 1 (scalar)

$$K(x_i, x_j) = \left(x_i^{\top} x_j + 1\right)^4 = x_i^4 x_j^4 + 4x_i^3 x_j^3 + 6x_i^2 x_j^2 + 4x_i \; x_j \; + 1$$

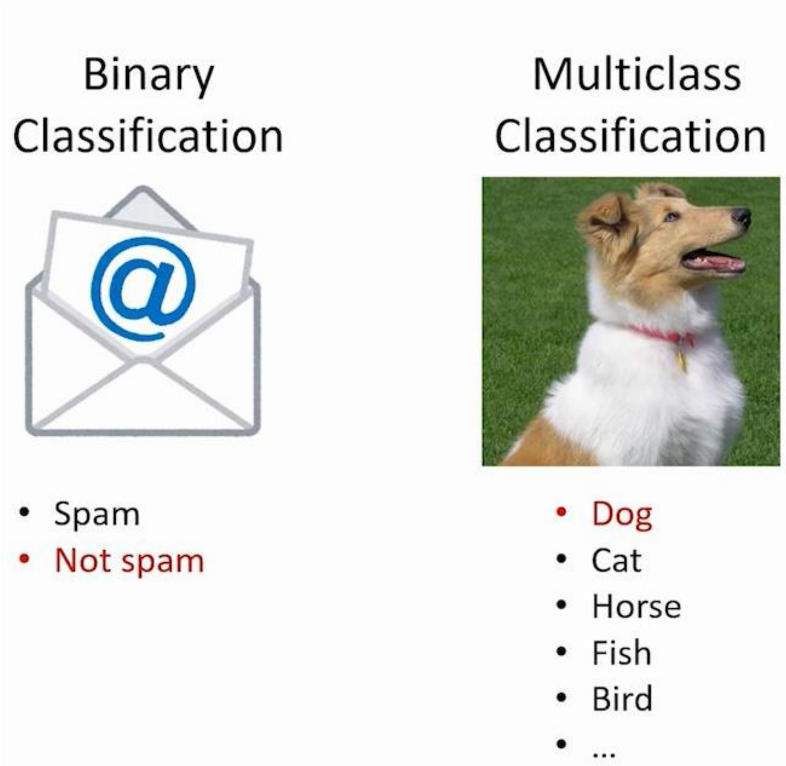
$$= \left(x_i^4, 2x_i^3, \sqrt{6}x_i^2, 2x_i \;\, , 1\right)^{\top} \left(x_j^4, 2x_j^3, \sqrt{6}x_j^2, 2x_j \;\, , 1\right)$$

$$\Phi(x) = \left(x^4, 2x^3, \sqrt{6}x^2, 2x, 1\right)$$

SVM for Multi-class Classification

One-to-One

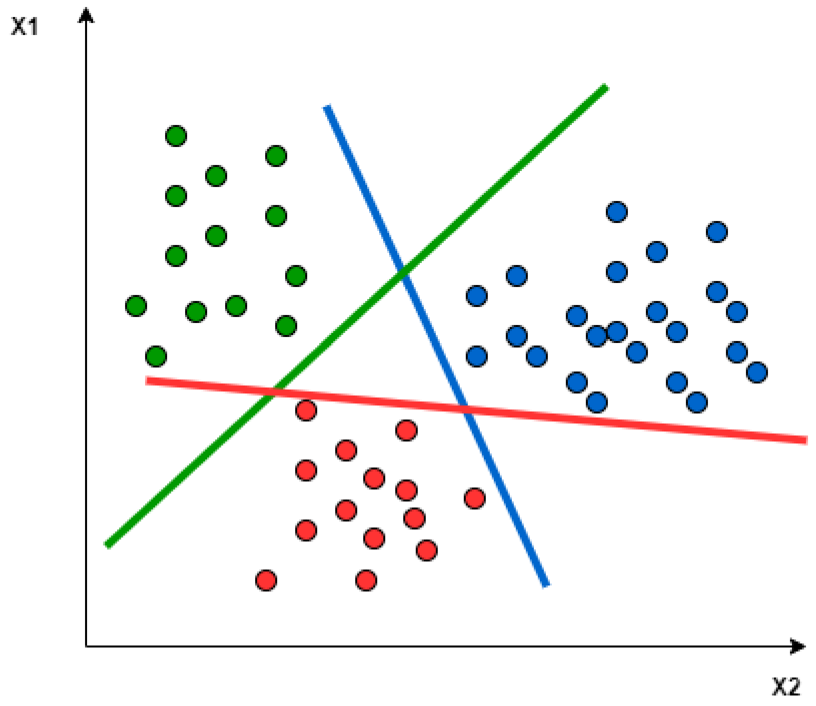
One-to-Rest



그림출처: Ertuğrul Demir https://www.kaggle.com/c/lish-moa/discussion/180500

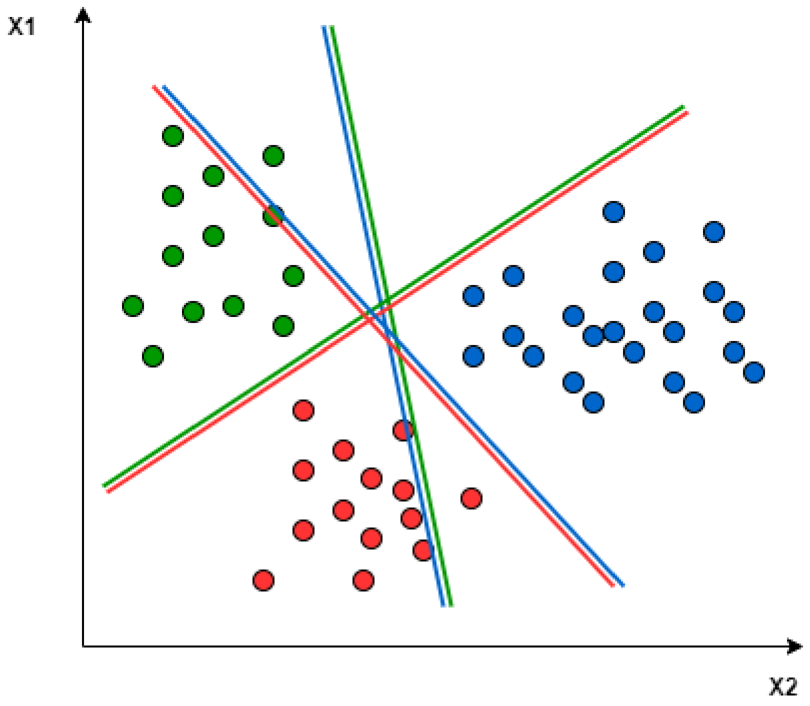
One-to-Rest

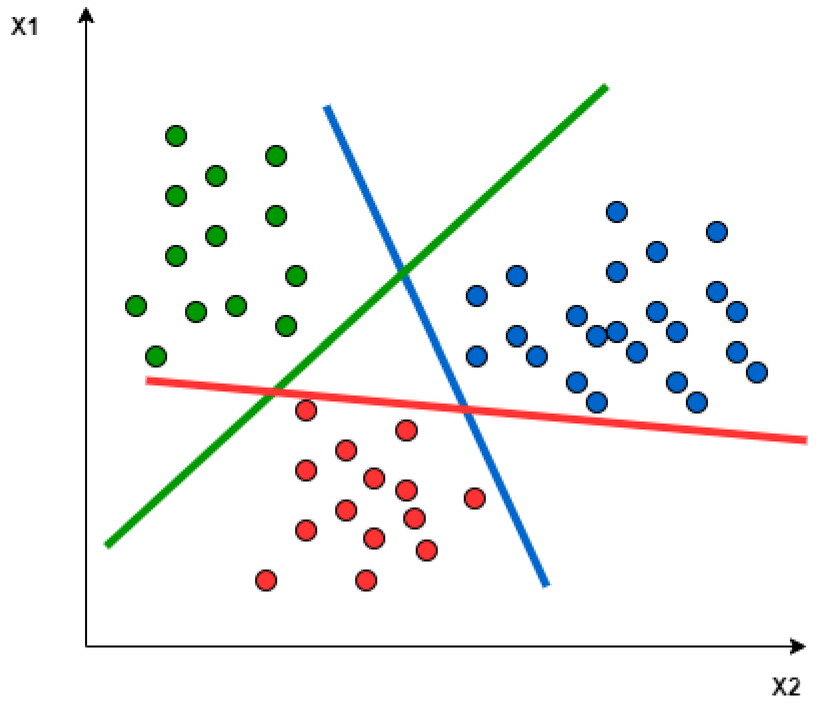
- Splitting the multi-class dataset into multiple binary classification problems
  - Example) Multi-class problem: 'red', 'blue', 'green'
    - **Binary Classification 1**: red vs [blue, green]
    - **Binary Classification 2**: blue vs [red, green]
    - **Binary Classification 3**: green vs [red, blue]
- Number of datasets (models): $\#\, classes$
- Predictions are made using the model with the highest confidence



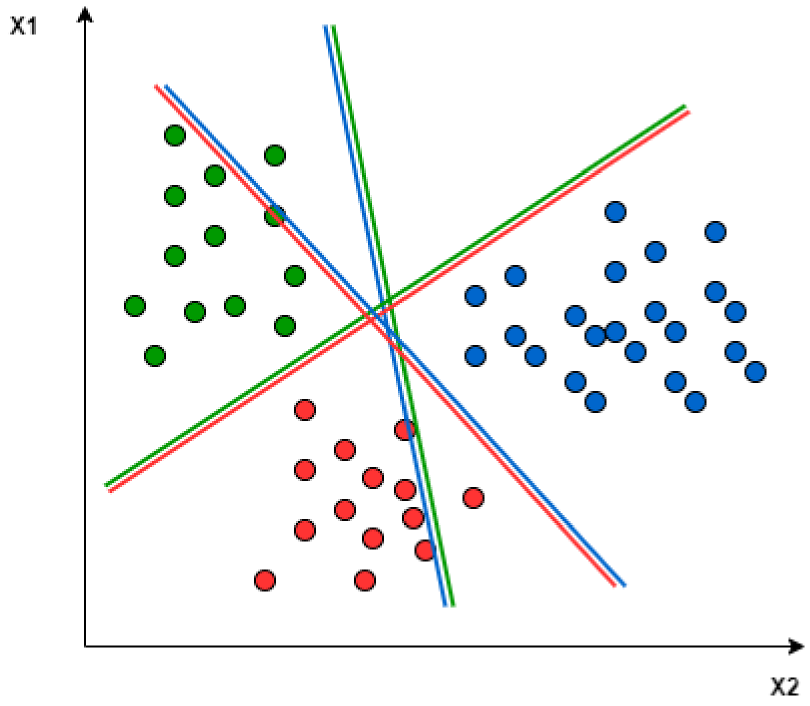
One-to-One

- Splitting the multi-class dataset into multiple binary classification problems
  - Example) Multi-class problem: 'red', 'blue', 'green'
    - **Binary Classification 1**: red vs. blue
    - **Binary Classification 2**: red vs. green
    - **Binary Classification 3**: red vs. yellow
    - **Binary Classification 4**: blue vs. green
    - **Binary Classification 5**: blue vs. yellow
    - **Binary Classification 6**: green vs. yellow
- Number of datasets (models): $\frac{n_{class}(n_{class}-1)}{2}$
- Prediction
  - Voting





One-to-Rest



One-to-One

https://www.baeldung.com/cs/svm-multiclass-classification

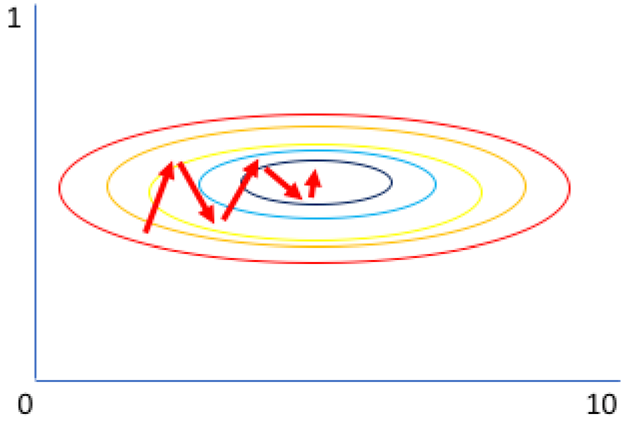
SVM vs. Neural Network

- **SVM**
  - Deterministic algorithm
  - Nice generalization properties
  - Hard to learn – learned in batch mode using quadratic programming techniques
  - Using kernels can learn very complex functions
- **Neural Network**
  - Nondeterministic algorithm
  - Generalizes well but doesn't have strong mathematical foundation
  - Can easily be learned in incremental fashion
  - To learn complex functions—use multilayer perceptron (nontrivial)

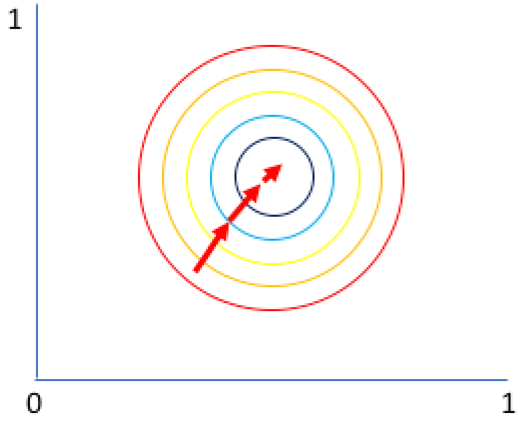
Implementing SVM with Scikit-learn

Input normalization

$$X' := \frac{X - \mu}{\sigma^2}$$



Gradient of larger parameter
dominates the update



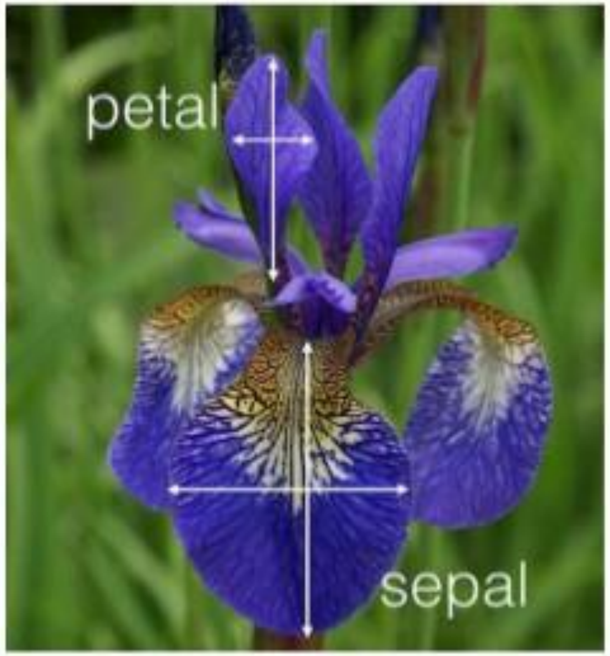
Both parameters can be
updated in equal proportions

www.jeremyjordan.me

https://archive.ics.uci.edu/ml/datasets/iris

Iris Dataset

Supervised learning *classification* problem
(using the Iris flower data set)



Training / test data

Features | Labels

Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	Iris setosa
4.9	3.0	1.4	0.2	Iris setosa
7.0	3.2	4.7	1.4	Iris versicolor
6.4	3.2	4.5	1.5	Iris versicolor
6.3	3.3	6.0	2.5	Iris virginica
5.8	3.3	6.0	2.5	Iris virginica

https://medium.com/analytics-vidhya/exploration-of-iris-dataset-using-scikit-learn-part-1-8ac5604937f8 https://archive.ics.uci.edu/ml/datasets/iris

Iris Dataset

Setosa Versicolor Virginica

https://medium.com/analytics-vidhya/exploration-of-iris-dataset-using-scikit-learn-part-1-8ac5604937f8

Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.preprocessing import FunctionTransformer

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
```

Load data

```
iris =load_iris()
#['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)'] 타겟 외 컬럼
X=iris.data[:,[2,3]]
```

```
Y=iris.target
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=0)
```

```
sc = StandardScaler()
sc.fit(X_train)
X_train_std=sc.transform(X_train)
X_test_std=sc.transform(X_test)
X_combined_std=np.vstack((X_train_std,X_test_std))
Y_combined_std=np.hstack((Y_train,Y_test))
```

sklearn.preprocessing.StandardScaler

class sklearn.preprocessing.StandardScaler(*, *copy=True*, *with_mean=True*, *with_std=True*) ¶ [source]

Standardize features by removing the mean and scaling to unit variance.

The standard score of a sample `x` is calculated as:

z = (x - u) / s

where `u` is the mean of the training samples or zero if `with_mean=False`, and `s` is the standard deviation of the training samples or one if `with_std=False`.

Model - Train

```
## Linear SVM
model1=SVC(kernel='linear').fit(X_test_std,Y_test)
## Polynomial SVM
model2=SVC(kernel='poly', random_state=0,gamma=10,C=1).fit(X_test_std,Y_test)
## Gaussian SVM
model3=SVC(kernel='rbf',random_state=0,gamma=1,C=1).fit(X_test_std,Y_test)
```

sklearn.svm.SVC

class sklearn.svm.SVC(**, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None*) ¶ [source]

C-Support Vector Classification.

Parameters:

C : *float, default=1.0*

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.

kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'*

Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

degree : *int, default=3*

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

Test

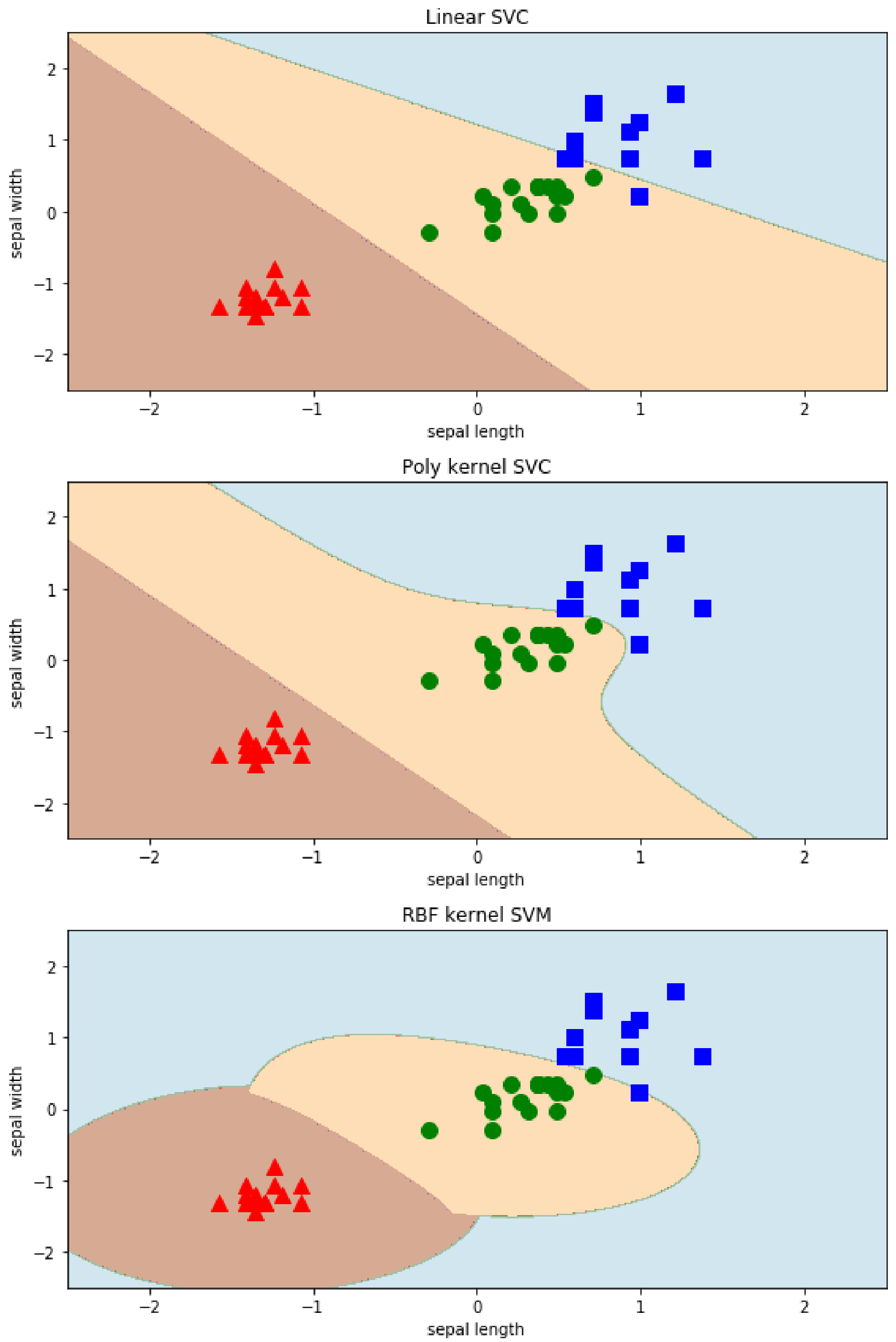
```
print("Accuracy")
## Linear SVM
Y_pred = model1.predict(X_test_std)
accuracy1 = accuracy_score(Y_test, Y_pred)
print(f"Linear SVM₩t{accuracy1:.3f}")
## Polynomial SVM
Y_pred = model2.predict(X_test_std)
accuracy2 = accuracy_score(Y_test, Y_pred)
print(f"Polynomial SVM₩t{accuracy2:.3f}")
## Gaussian SVM
model3=SVC(kernel='rbf',random_state=0,gamma=1,C=1).fit(X_test_std,Y_test)
Y_pred = model3.predict(X_test_std)
accuracy3 = accuracy_score(Y_test, Y_pred)
print(f"Gaussian SVM₩t{accuracy3:.3f}")
```

```
Accuracy
Linear SVM      0.933
Polynomial SVM  1.000
Gaussian SVM    0.978
```

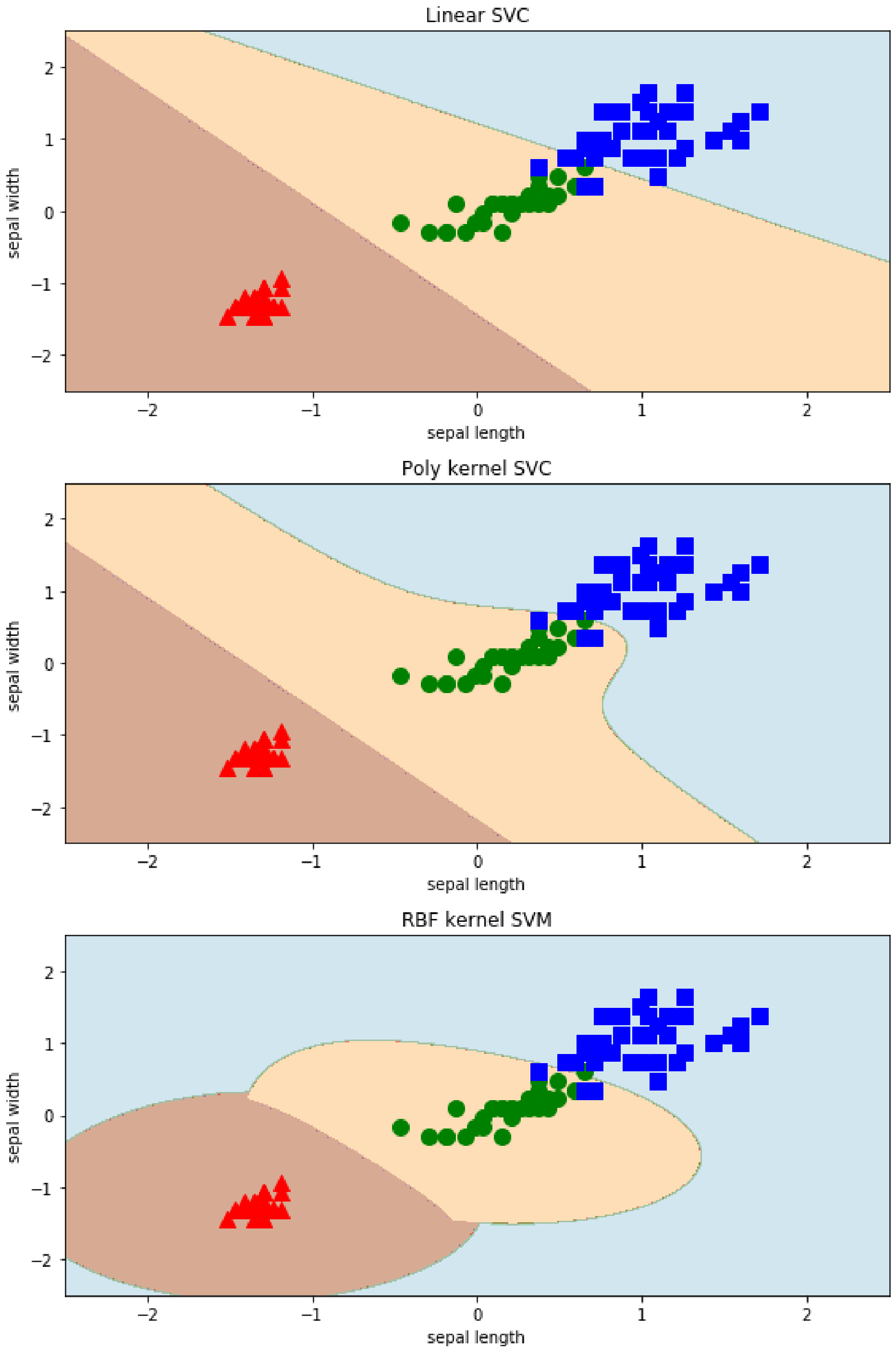
Visualization

```python
def plot_iris(X,Y,model,title,xmin=-2.5,xmax=2.5, ymin=-2.5,ymax=2.5):
    XX,YY=np.meshgrid(
        np.arange(xmin,xmax,(xmax-xmin)/1000),
        np.arange(ymin,ymax,(ymax-ymin)/1000))
    ZZ=np.reshape(model.predict(np.array([XX.ravel(),YY.ravel()]).T),XX.shape)
    plt.contourf(XX,YY,ZZ,cmap=plt.cm.Paired_r,alpha=0.5)
    plt.scatter(X[Y == 0, 0], X[Y == 0, 1], c='r', marker='^', label='0', s=100)
    plt.scatter(X[Y == 1, 0], X[Y == 1, 1], c='g', marker='o', label='1', s=100)
    plt.scatter(X[Y == 2, 0], X[Y == 2, 1], c='b', marker='s', label='2', s=100)
    plt.xlim(xmin, xmax)
    plt.ylim(ymin, ymax)
    plt.xlabel("sepal length")
    plt.ylabel("sepal width")
    plt.title(title)

plt.figure(figsize=(8, 12))
plt.subplot(311)
plot_iris(X_test_std, Y_test, model1, "Linear SVC")
plt.subplot(312)
plot_iris(X_test_std, Y_test, model2, "Poly kernel SVC")
plt.subplot(313)
plot_iris(X_test_std, Y_test, model3, "RBF kernel SVM")
plt.tight_layout()
plt.show()
```

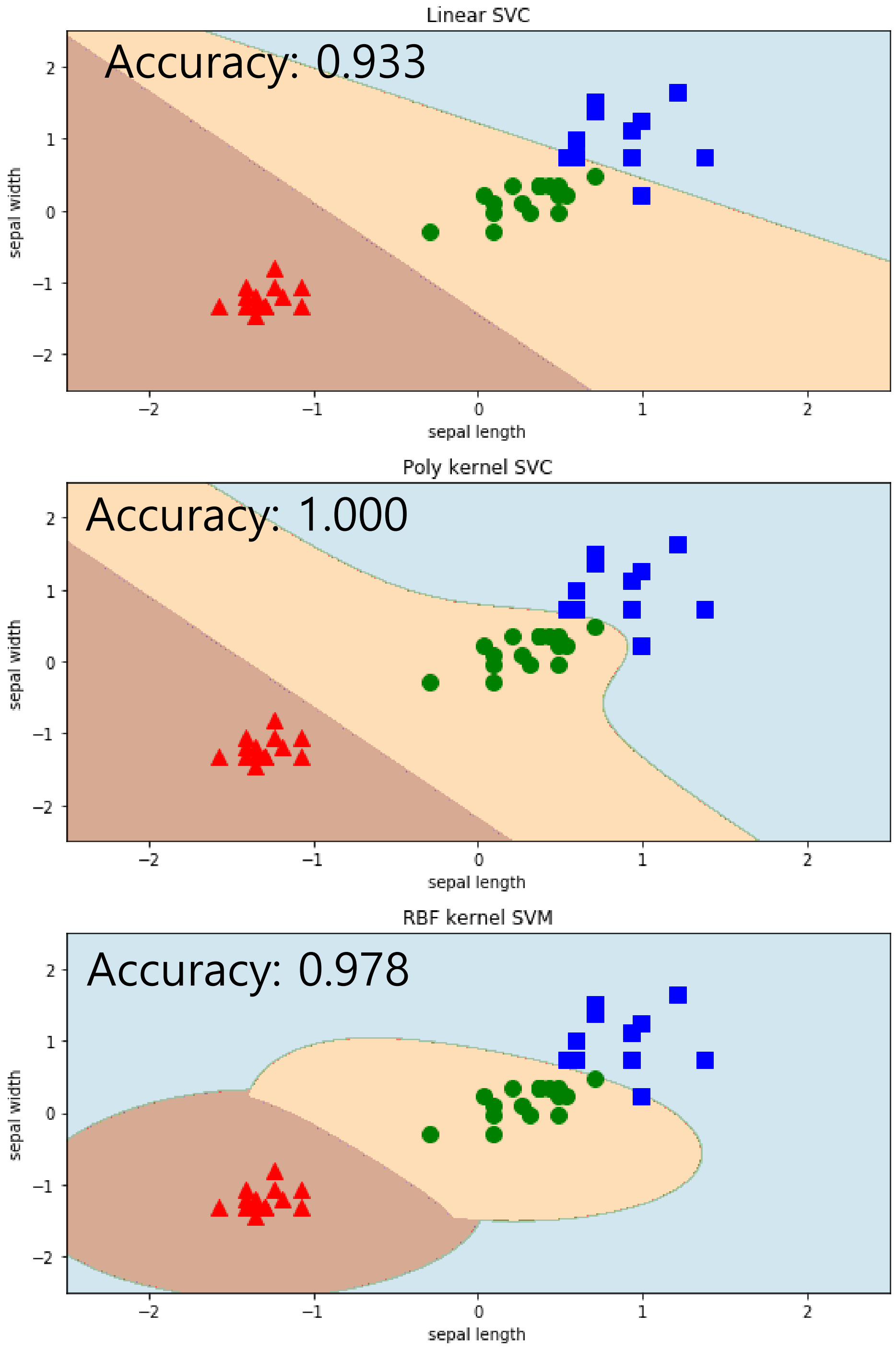


Train



Linear SVC
Poly kernel SVC
RBF kernel SVM
sepal width
sepal length

Test



Linear SVC
Accuracy: 0.933
Poly kernel SVC
Accuracy: 1.000
RBF kernel SVM
Accuracy: 0.978
sepal width
sepal length

References

- Andrew W. Moore's slides:
  - http://www.cs.cmu.edu/~awm/tutorials
- Seoung Bum Kim's slides:
  - https://youtu.be/qFg8cDnqYCI
- Kyuseok Shim's slides
- Classic! (tistory)
  - https://icefree.tistory.com/entry/Machine-Learning-Kernel-SVMSupport-Vector-Machine