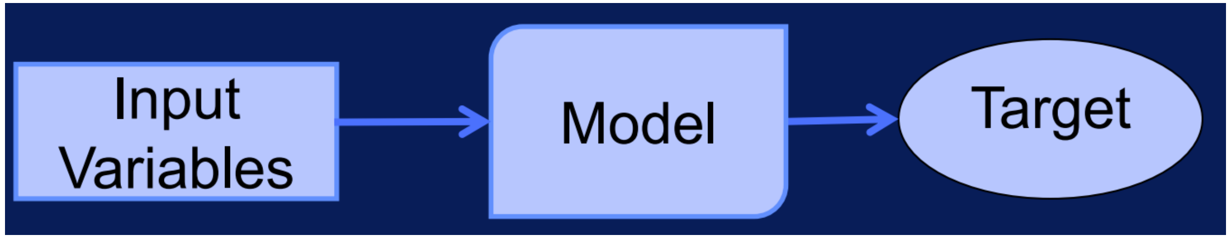
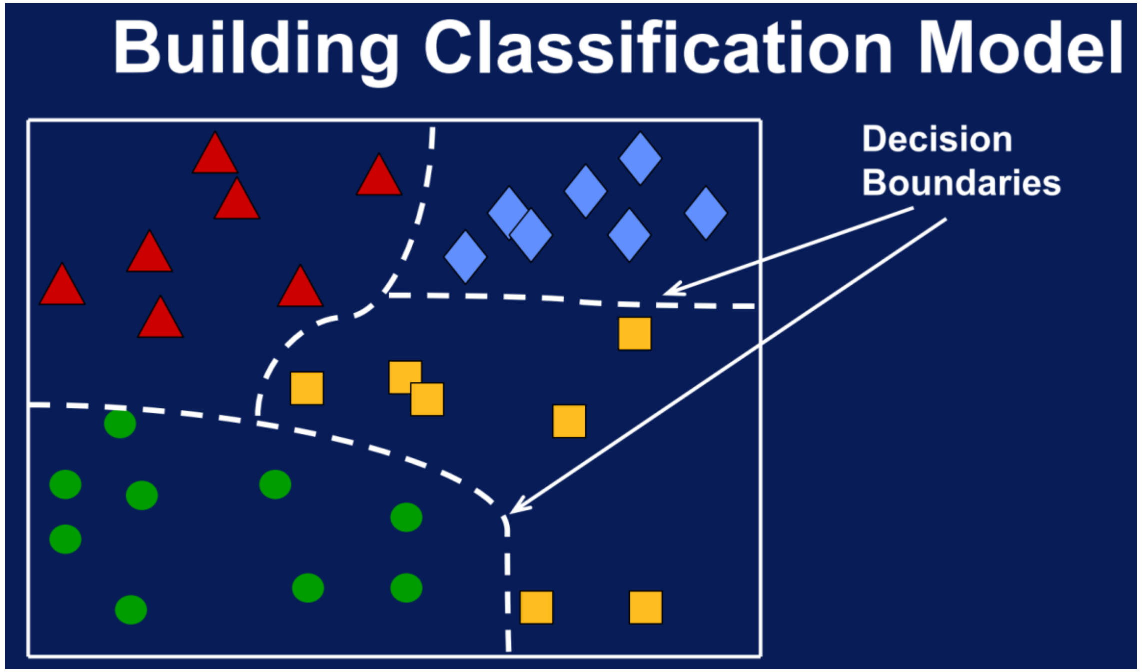
Classification Model

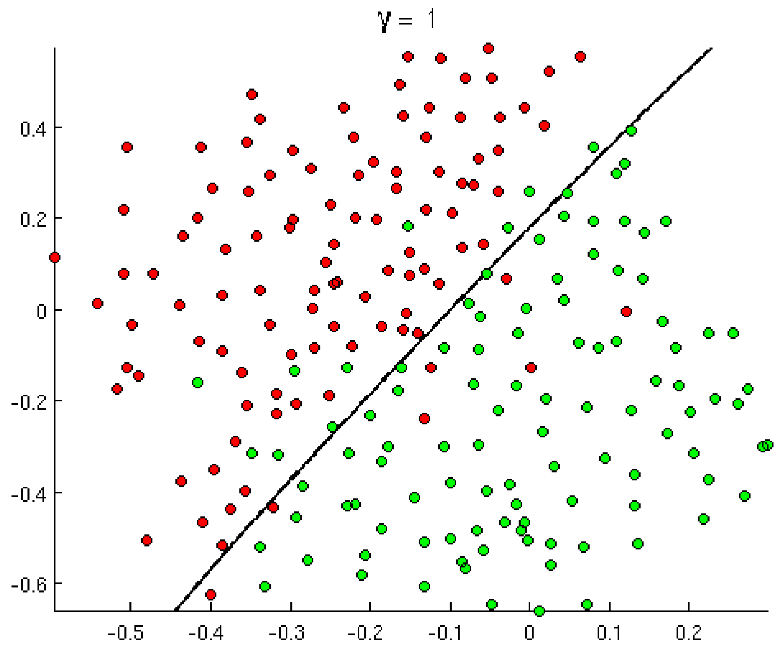
classification

- Predict: Category from input variables
- Goal: Match model outputs to targets (desired outputs)

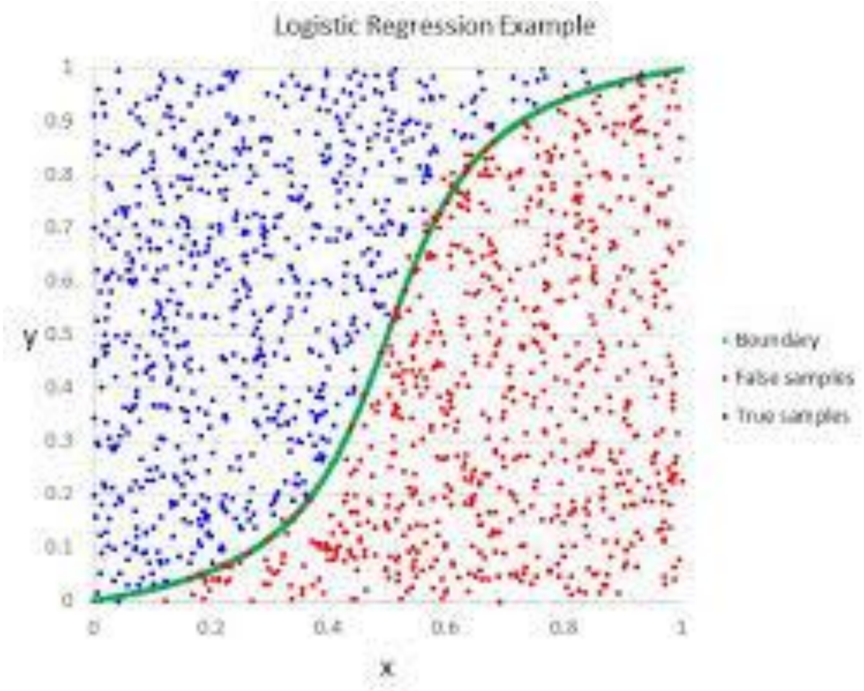




Classification (single dimension Input)

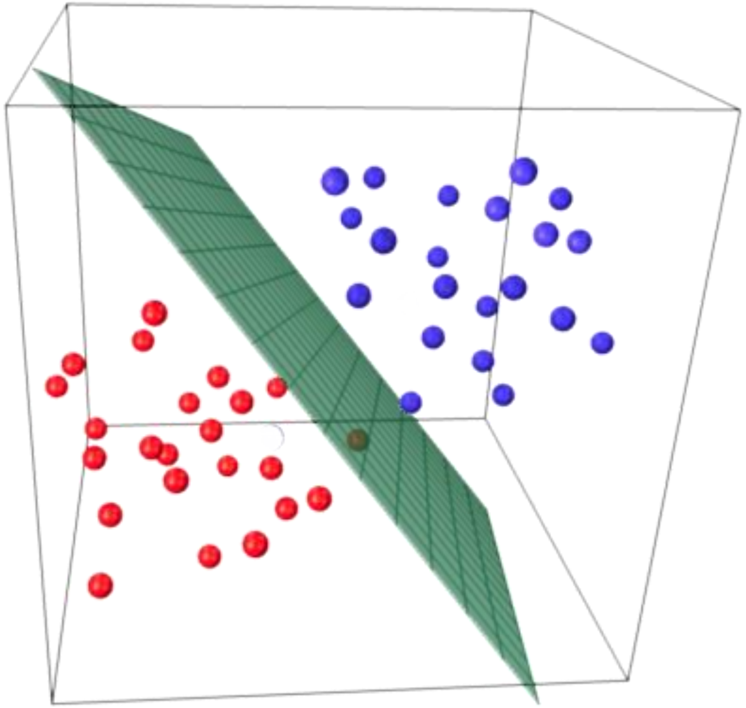


Linear Classifier

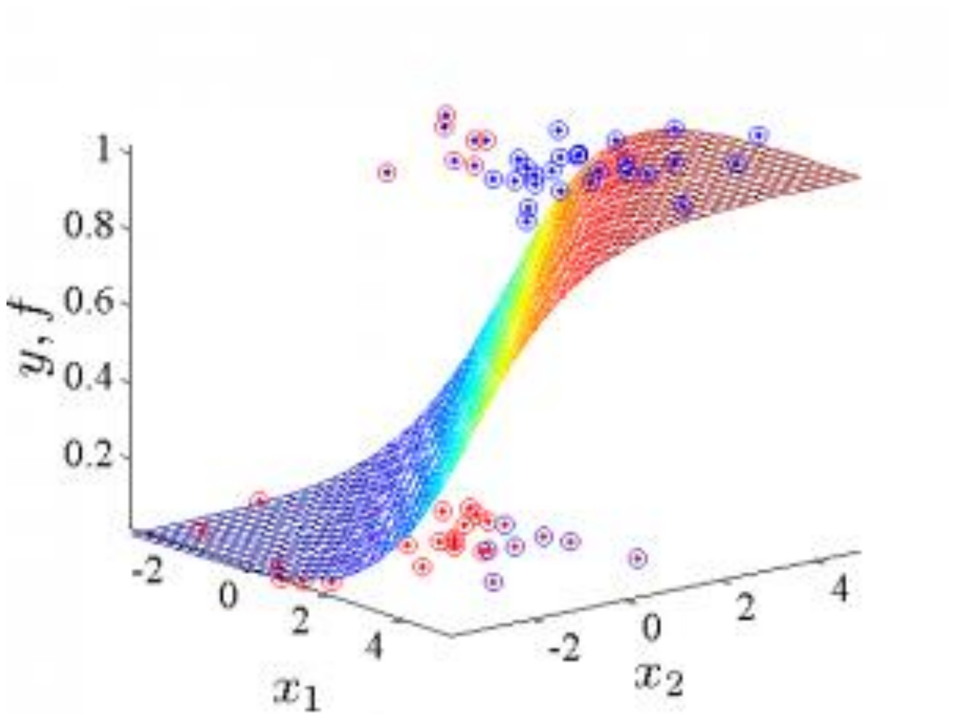


Nonlinear Classifier (Logistic / Sigmoid)

Classification (Multi-dimension Inputs)



Linear Classifier



Nonlinear Classifier (Logistic / Sigmoid)

Supervised vs Unsupervised Model

Supervised Classification	Unsupervised Clustering
• known number of classes	• unknown number of classes
• based on a training set	• no prior knowledge
• used to classify future observations	• used to understand (explore) data

Supervised Classification

Single Node MODEL

- kNN (k-Nearest Neighbor)
- Logistic Regression
- Support Vector Machine

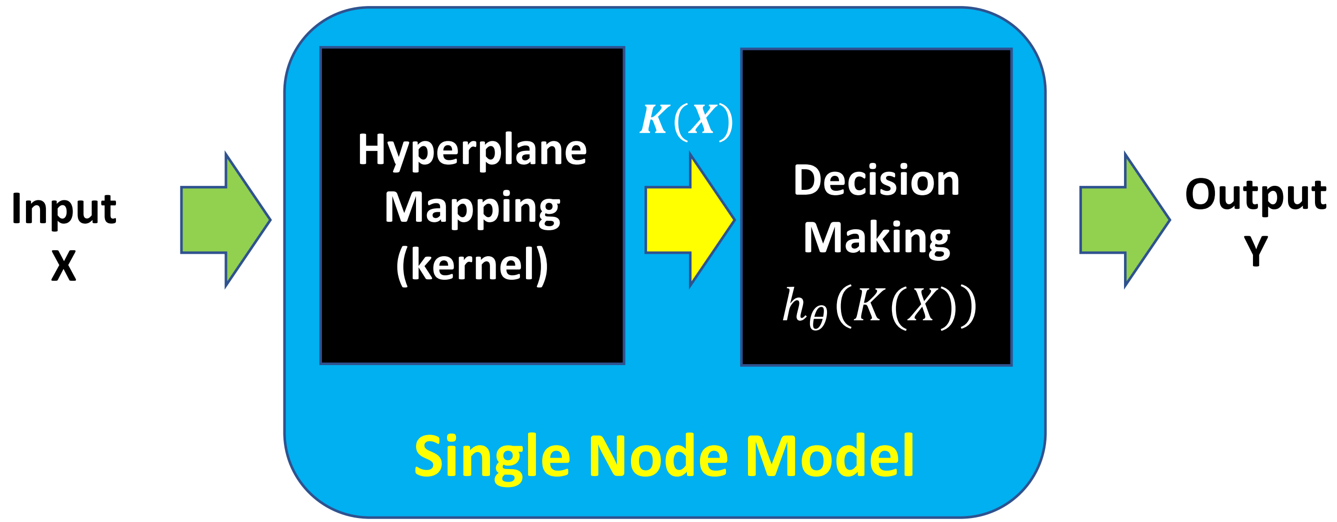
Multi-Node Model

- Neural Networks (Deep Networks)
- Convolutional Neural Network
- Recurrent Neural Network
  - Long Short Term Memory (LSTM)

SINGLE NODE MODEL

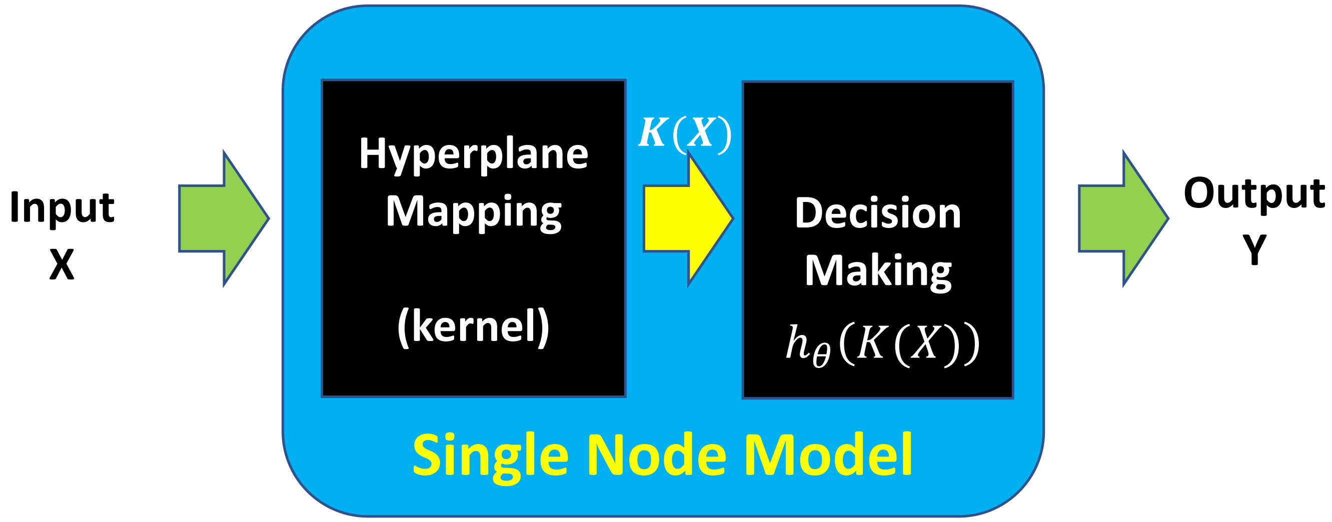
Logistic regression vs SVM

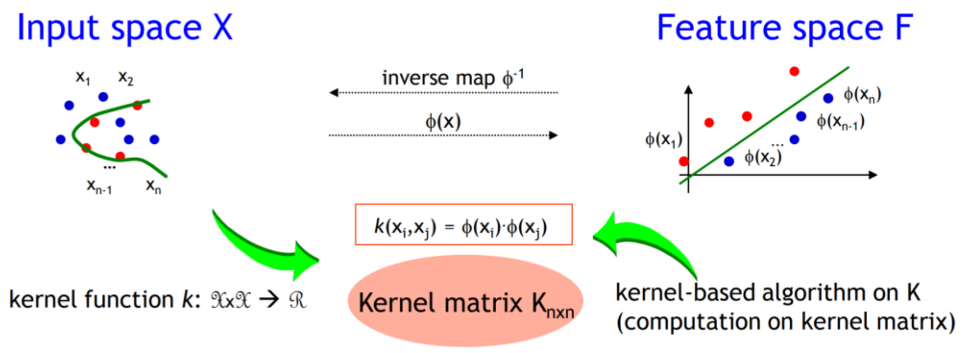
Single node Model Architecture



- **Model Architecture**
  - Hyperplane Mapping through
    - Model Kernel
  - Decision Making through
    - Hypothesis Function
    - Activation Function

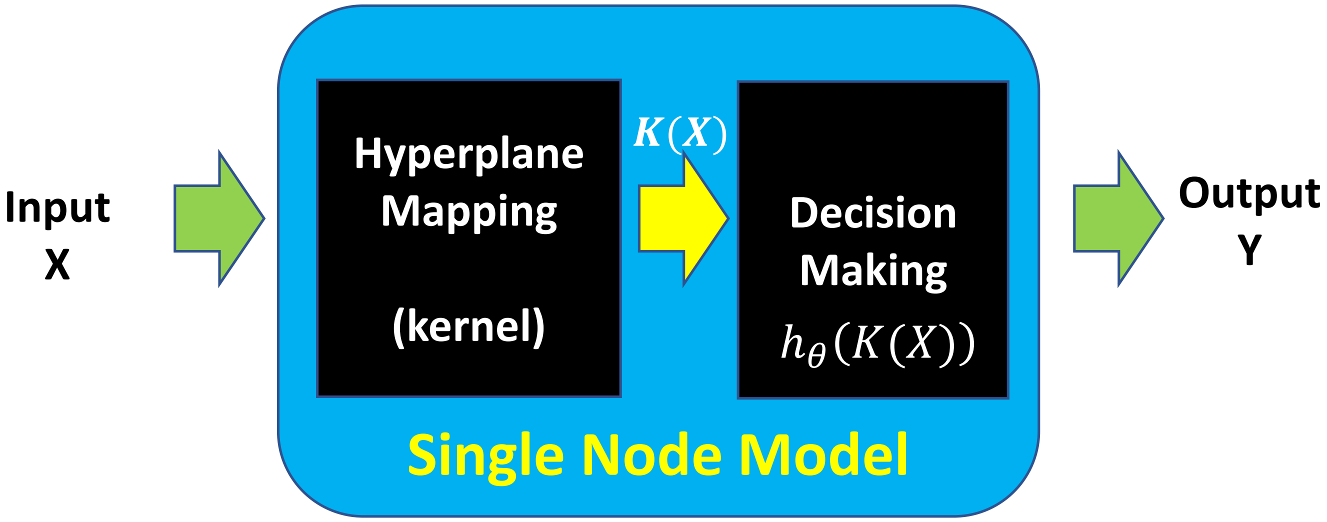
Single node Model Architecture



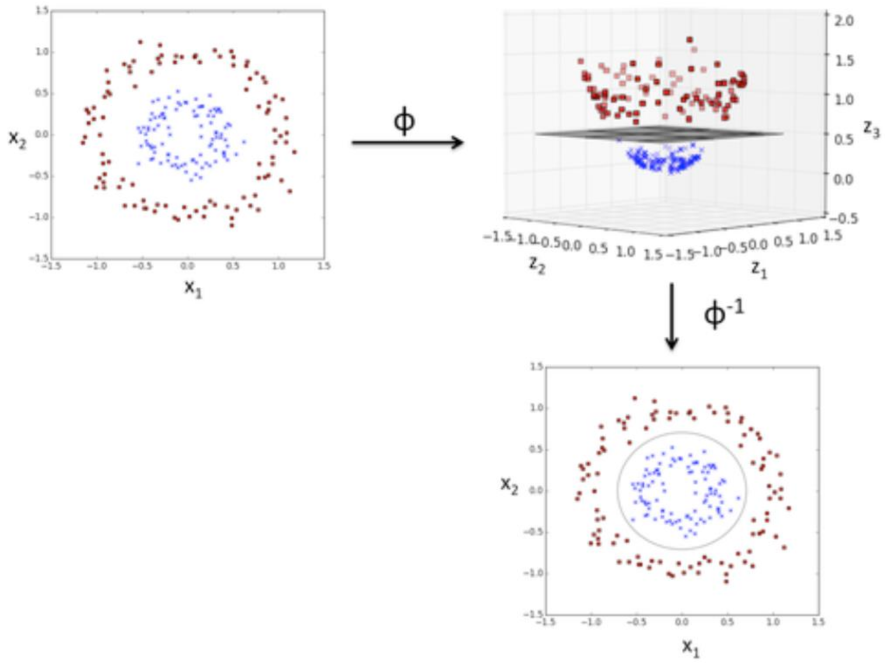


- **Model Architecture**
  - Hyperplane Mapping through
    - Kernel (K)
      - Linear kernel
      - Nonlinear kernel
        - Gaussian
        - Radial Basis Function (RBF)
        - Polynomial

Single node Model Architecture



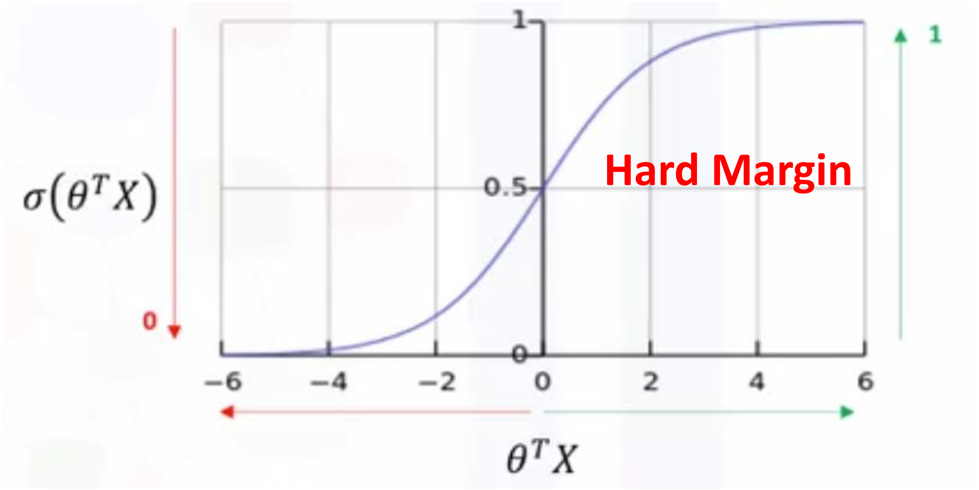
- **Model Architecture**
  - Hyperplane Mapping through
    - Kernel (K)



Model Kernel: Hyperplane Mapping

Logistic Regression

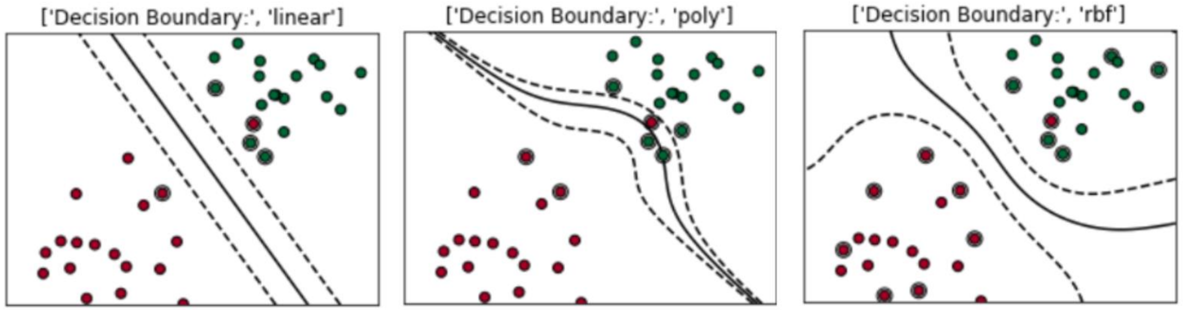
- **Sigmoid / Logistic Kernel Function**



$K(X) = \theta^T X = \theta_0 + \theta_1 x_1 + \theta_2\, x_2 + \cdots + \theta_n\, x_n$

Support Vector Machine (SVM)

- **Class Partition Kernels: Linear / Non-Linear**

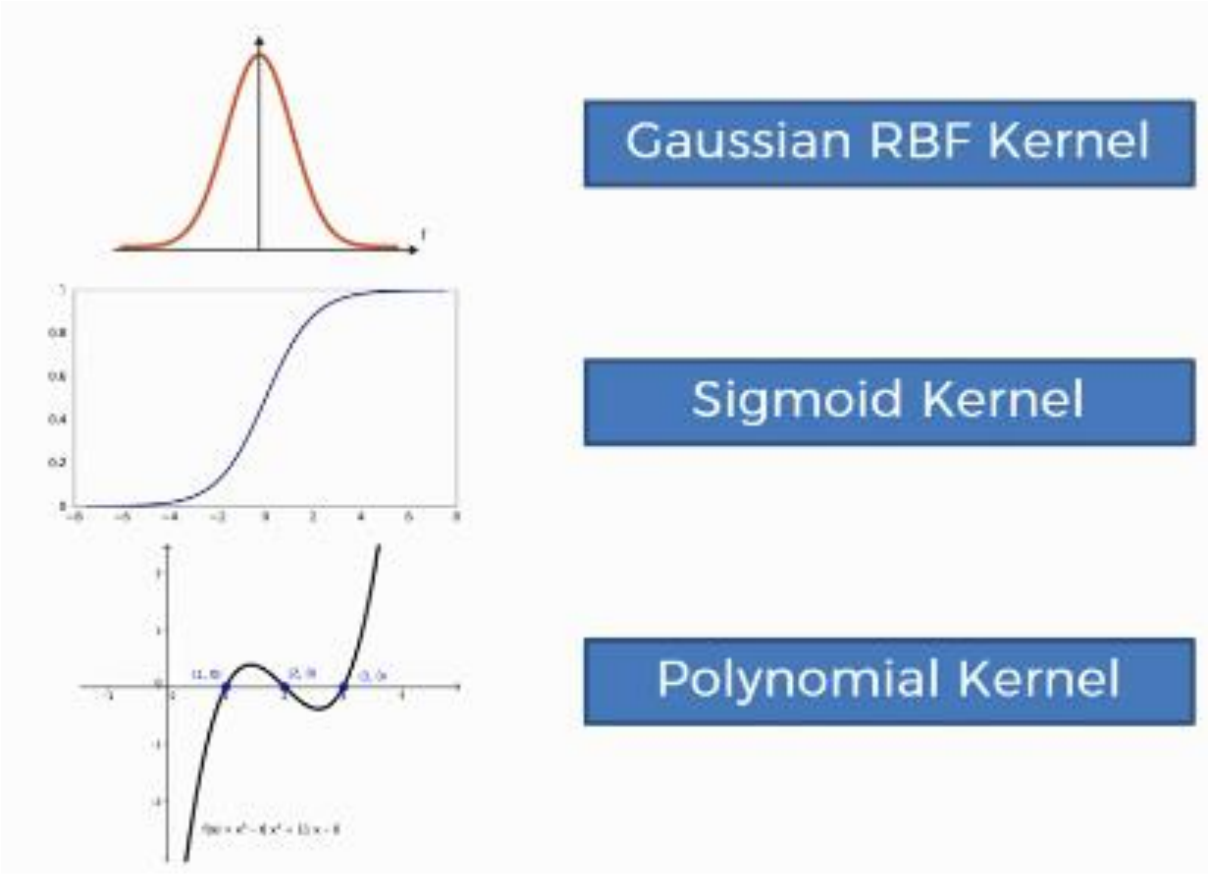


Soft Margin

$K(X) = x_i^T x_j$

$K(X) = exp\left(-\gamma \|x_i - x_j\|^2\right)$

SVC: kernels



Gaussian kernel

$$K(x_i, x_j) = exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

Gaussian radial basis function (RBF)

$$K(x_i, x_j) = exp\left(-\gamma\|x_i - x_j\|^2\right)$$

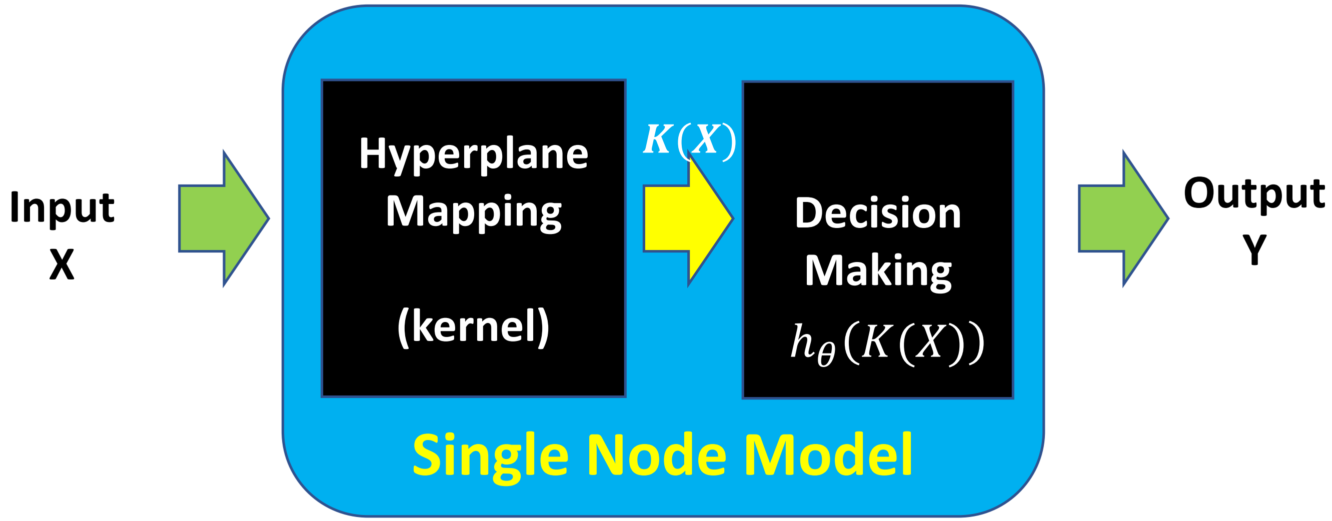
Polynomial kernel

$$K(x_i, x_j) = (x_i x_j)^d$$

Sigmoid kernel

$$K(x_i, x_j) = tanh(\alpha {x_i}^T x_j + c)$$

Single node Model Architecture

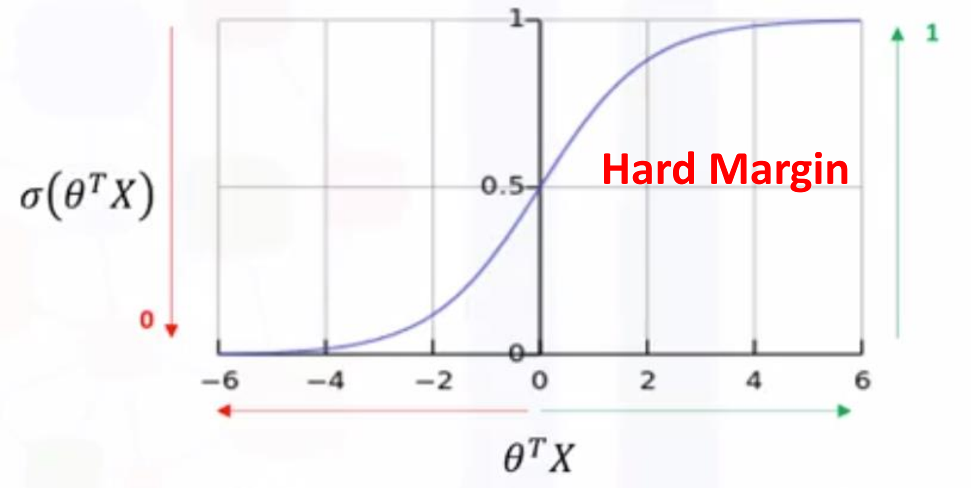


- **Model Architecture**
  - Hyperplane Mapping through
    - Kernel (K)
  - Decision Making though
    - Hypothesis function
      - Activation function
      - $h_\theta(K(X))$

Model Kernel: Decision Making

Logistic Regression

- **Sigmoid / Logistic Kernel Function**



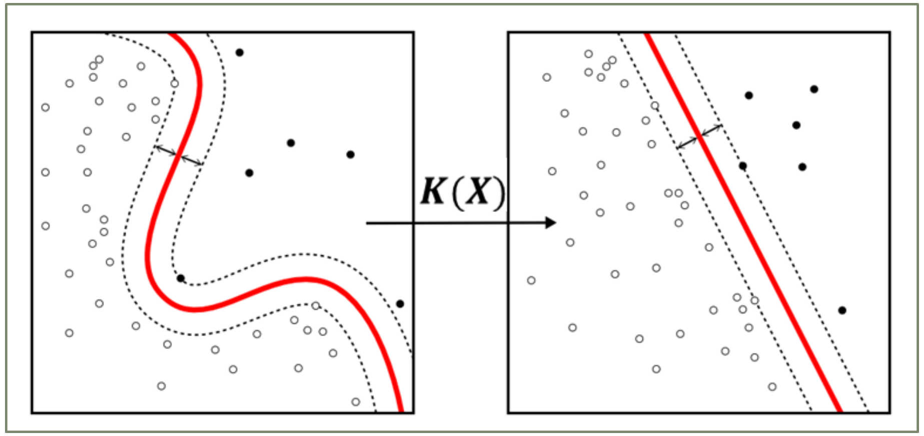
$$K(X) = \theta^T X = \theta_0 + \theta_1 x_1 + \theta_2\, x_2 + \cdots + \theta_n\, x_n$$

$$h_\theta(K(X)) = \sigma\left(\theta^T X\right) = \frac{1}{1 + \exp(\theta^T X)}$$

$$= \frac{1}{1 + \exp(\,\theta_1 x_1 + \theta_2\, x_2 + \theta_3\, x_3 + b)}$$

Support Vector Machine (SVM)

- **Class Partition Kernels: Linear / Non-Linear**



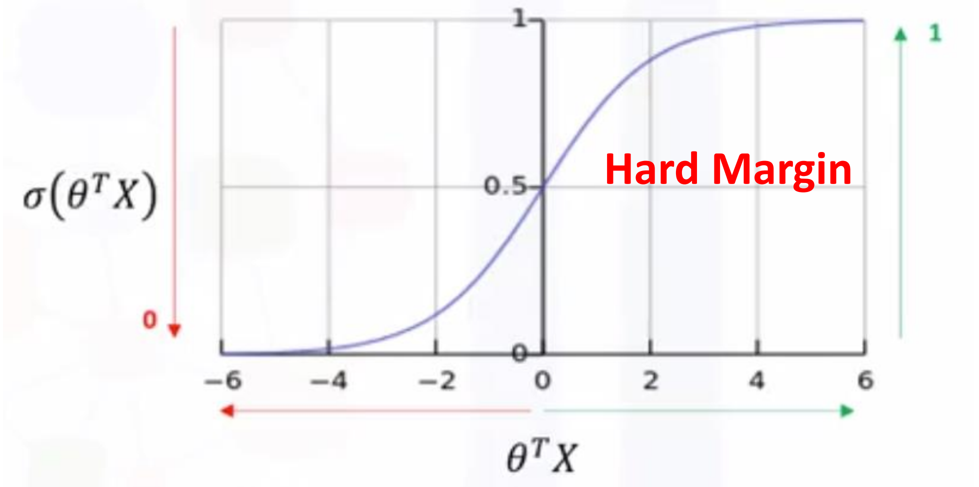
$$h_\theta(K(X)) = W^T \mathrm{K(X)} - b = \theta^T \mathrm{K(X)}$$

Soft Margin

Model Kernel: Decision Making

Logistic Regression

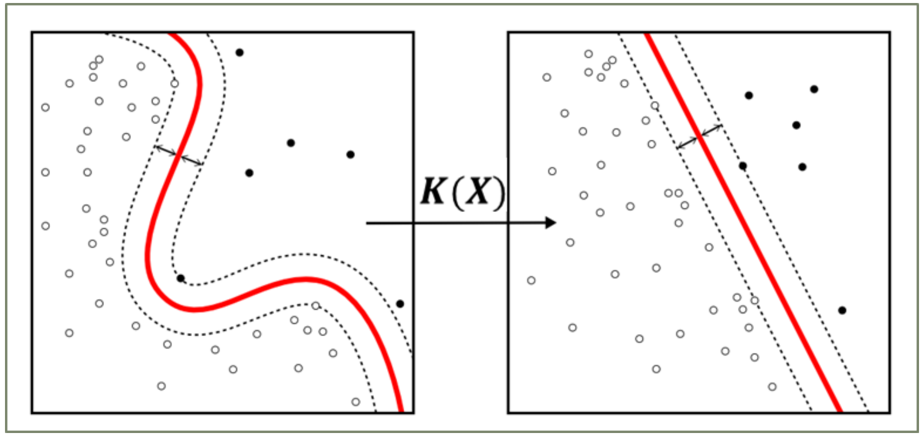
- **Sigmoid / Logistic Kernel Function**



$$P(y=1|x) = h_\theta(K(X)) = \sigma\left(\theta^T X\right)$$
$$P(y=0|x) = 1 - P(y=1|x)$$
$$\hat{y} = \begin{cases} 1, & P(y=1|x) \geq P(y=0|x) \\ 0, & Otherwise \end{cases}$$

Support Vector Machine (SVM)

- **Class Partition Kernels: Linear / Non-Linear**



$$h_\theta(K(X)) = \theta^T \mathrm{K(X)}$$

Soft Margin

$$\hat{y} = \begin{cases} 1, & h_\theta \geq 0 \\ 0, & Otherwise \end{cases}$$

Best Optimizing hyperplane parameters ($\theta_0, \theta_1, \ldots, \theta_N$ & kernel paprmeter)

Why do we need to tune these parameters?

ML: Single Node Processor (Loss Function)

Hyperplane Parameter (θ_i) Optimization

$$L(x, y) = \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2$$

- **Regression Loss** Function: **Error** Function
  - Normally use **L2 distance** between
    - y: real output / desired output / ground truth
    - $h_\theta(K(X))$: Hyperplane decision of kernel (K)

ML: Single Node Processor (Loss Function)

Hyperplane Parameter (θ_i) Optimization

- **Classification Loss** Function:
  - distance measure
    - Dot product between y and $h_\theta\big(K(X)\big)$
      - $y \cdot h_\theta$

What's the difference between SVMs and Logistic Regression? (Revisited)

	SVMs	Logistic Regression
Loss function	Hinge loss	Log-loss
Kernels	Yes!	Yes!
Solution sparse	Often yes!	Almost always no!
Semantics of learned model	Linear model from "Margin"	Probability Distribution

ML: Single Node Processor (Loss Function)

Hyperplane Parameter (θ_i) Optimization

- **Classification Loss** Function:
  - distance measure
    - Dot product between y and $h_\theta(K(X))$
      - $y \cdot h_\theta = y \cdot z$

What's the difference between SVMs and Logistic Regression? (Revisited)

	SVMs	Logistic Regression
Loss function	Hinge loss	Log-loss

$$\ell(y, z) = \max(0, 1 - yz)$$

$$\ell_{\log}(y, z) = \ln(1 + e^{-yz})$$

ML: Single Node Processor (Loss Function)

Hyperplane Parameter (θ_i) Optimization

$$\ell(y, z) = \max(0, 1 - yz)$$

$$\ell_{\log}(y, z) = \ln(1 + e^{-yz})$$

Optimizing on training data only

It is possible to cause **overfitted**

Not understand general inputs

ML: Single Node Processor (Logistic)

Loss function with regularization

$$L(x, y) + \lambda\ (Regularization)$$

Regularization Control (λ)

C=1/λ

ML: Single Node Processor

Hyperplane Parameter Optimization

Logistic Regression

Hard Margin

- **Tuning Optimization Parameter:**
  - **C: Regularization factor on**
    - **weight magnitude**
  - **Kernel parameter**
    - **Linear -> No parameter**

Support Vector Machine (SVM)

Soft Margin

- **Tuning Optimization Parameter:**
  - **C: Regularization factor on**
    - **Soft margin (Slag variable)**
  - **Kernel parameter**
    - **Linear -> no parameter**
    - **RBF -> γ: gamma**
    - ***Polynomial -: d: polynomial degree***

ML: Single Node Processor (Logistic)

Loss function with regularization

$$\ell_{\log}(y, z) = \ln(1 + e^{-yz})$$

$$L(x,y) \equiv \sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 + \boxed{\lambda \sum_{i=1}^{n} \theta_i^2}$$

C in Logistic

C=1/λ

Large C: less λ -> may be overfitted
Less C: Large λ -> may be underfitted

Regularization on Weight Magnitude

L1 Regularization

$$\ell_{\log}(y, z) = \ln(1 + e^{-yz}) + \boxed{\lambda \sum_{i=1}^{n} |\theta_i|}$$

L2 Regularization

$$\ell_{\log}(y, z) = \ln(1 + e^{-yz})$$

ML: Single Node Processor (Logistic)

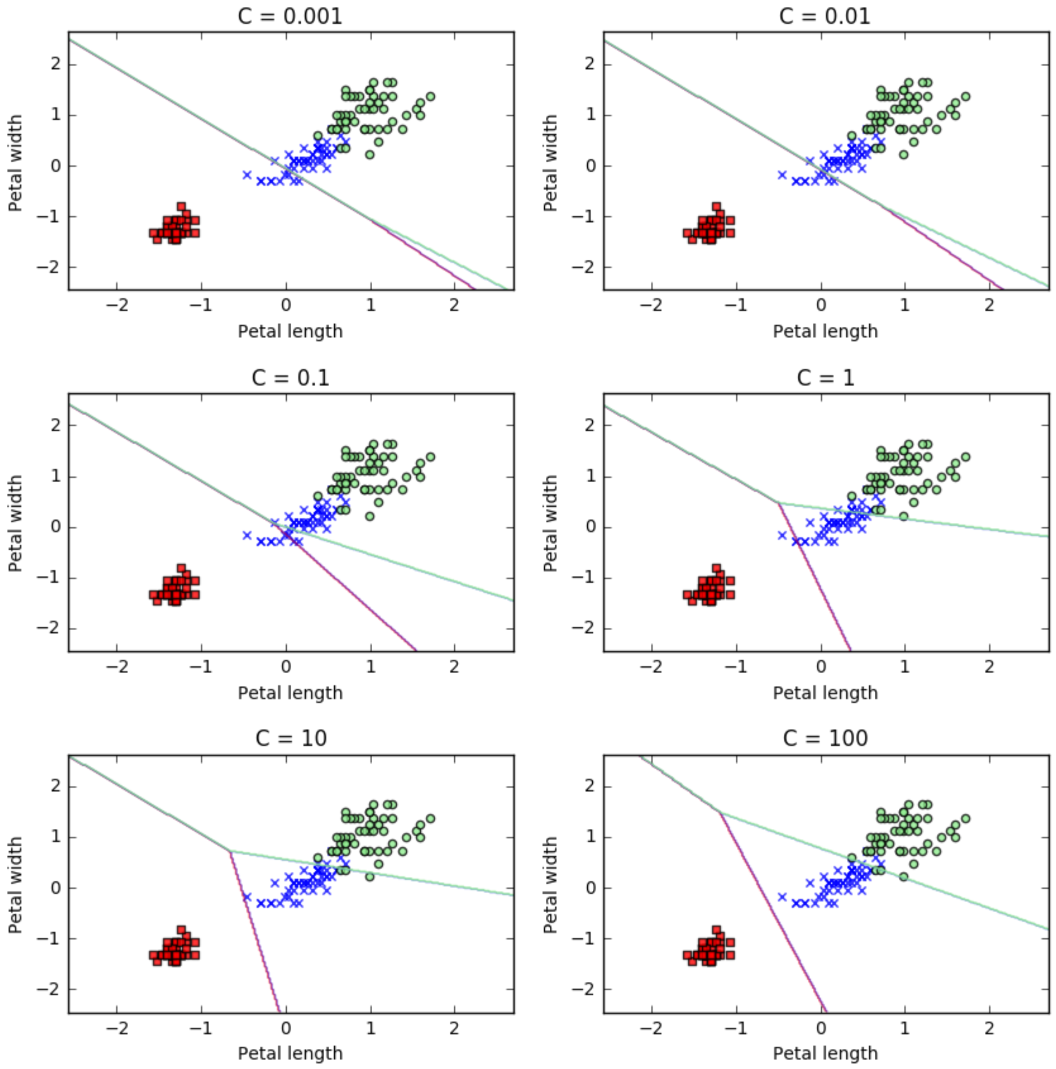
Loss function with regularization

$$\ell_{\log}(y, z) = \ln(1 + e^{-yz}) + \boxed{\lambda \sum_{i=1}^{n} \theta_i^2}$$

C in Logistic

C=1/λ

Large C: less λ -> may be overfitted
Less C: Large λ -> may be underfitted



ML: Single Node Processor (SVM)

Loss function with regularization

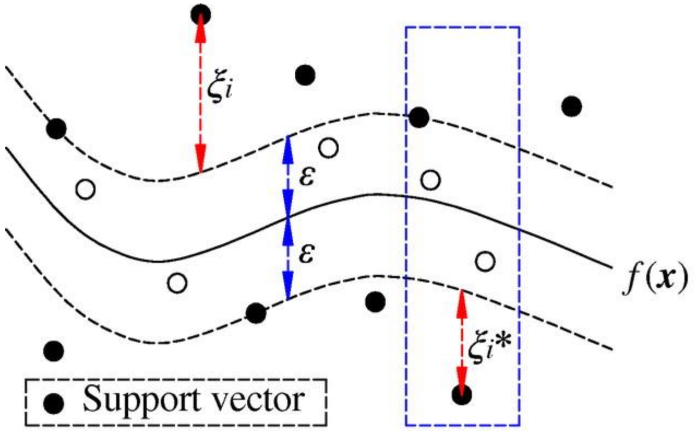
$$\ell(y, z) = \max(0, 1 - yz) + \boxed{C \sum_{i}^{N} \xi_i}$$

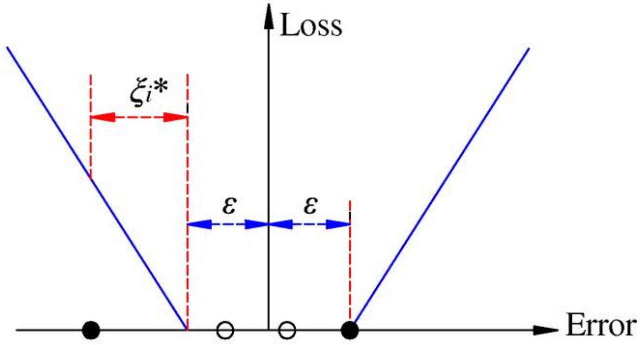
- L1 Regularization

C

Large C: may be overfitted
Less C: may be underfitted

Regularization on Soft Margin





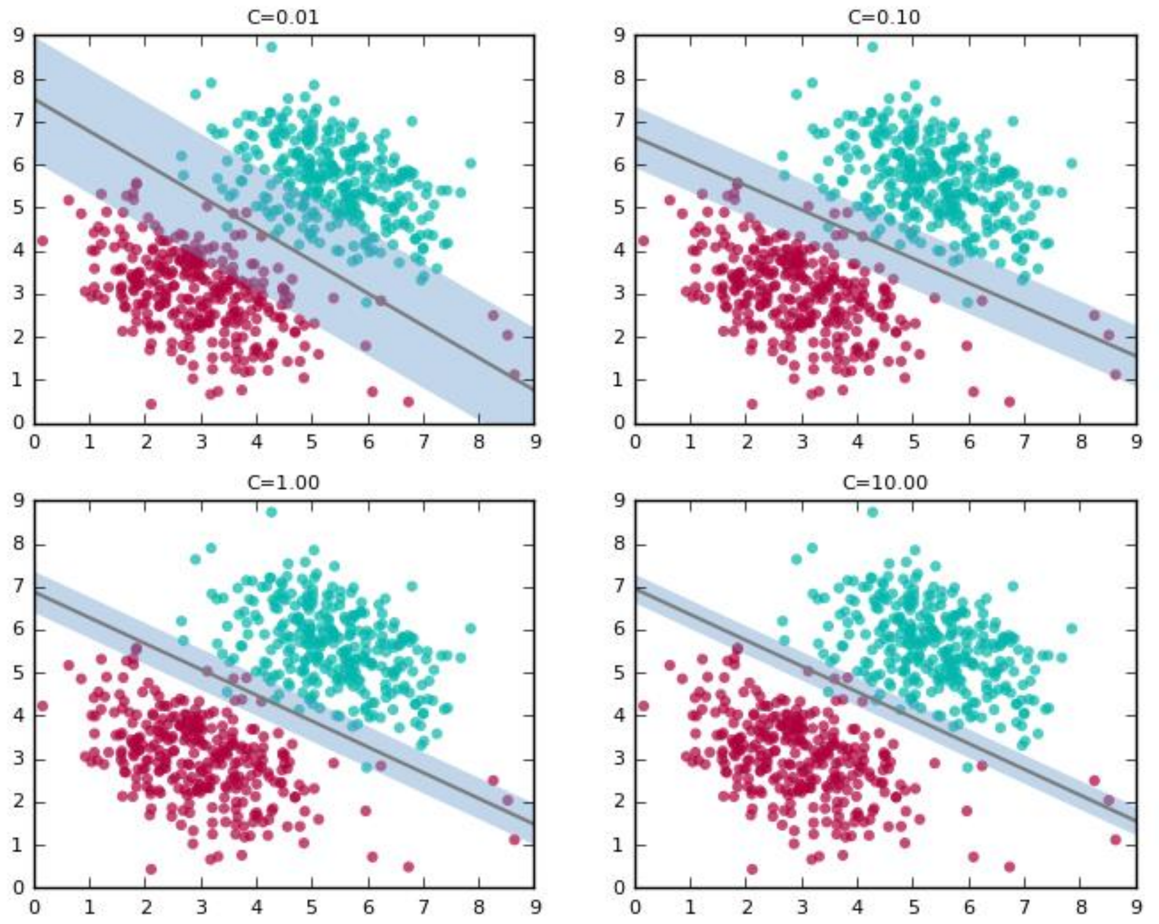
(b)

ML: Single Node Processor (SVM)

Loss function with regularization

- L1 Regularization

$$\ell(y, z) = \max(0, 1 - yz) + \boxed{C \sum_{i}^{N} \xi_i}$$



C in SVM

Large C: may be overfitted
Less C: may be underfitted

Regularization on Soft Margin

ML: Single Node Processor (SVM)

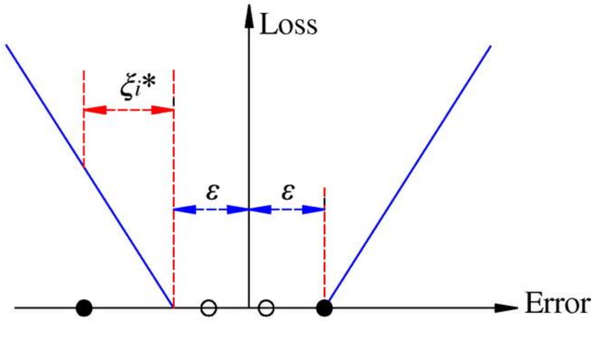
Loss function with regularization

- L1 Regularization

$$\ell(y, z) = \max(0, 1 - yz) + \boxed{C \sum_{i}^{N} \xi_i}$$

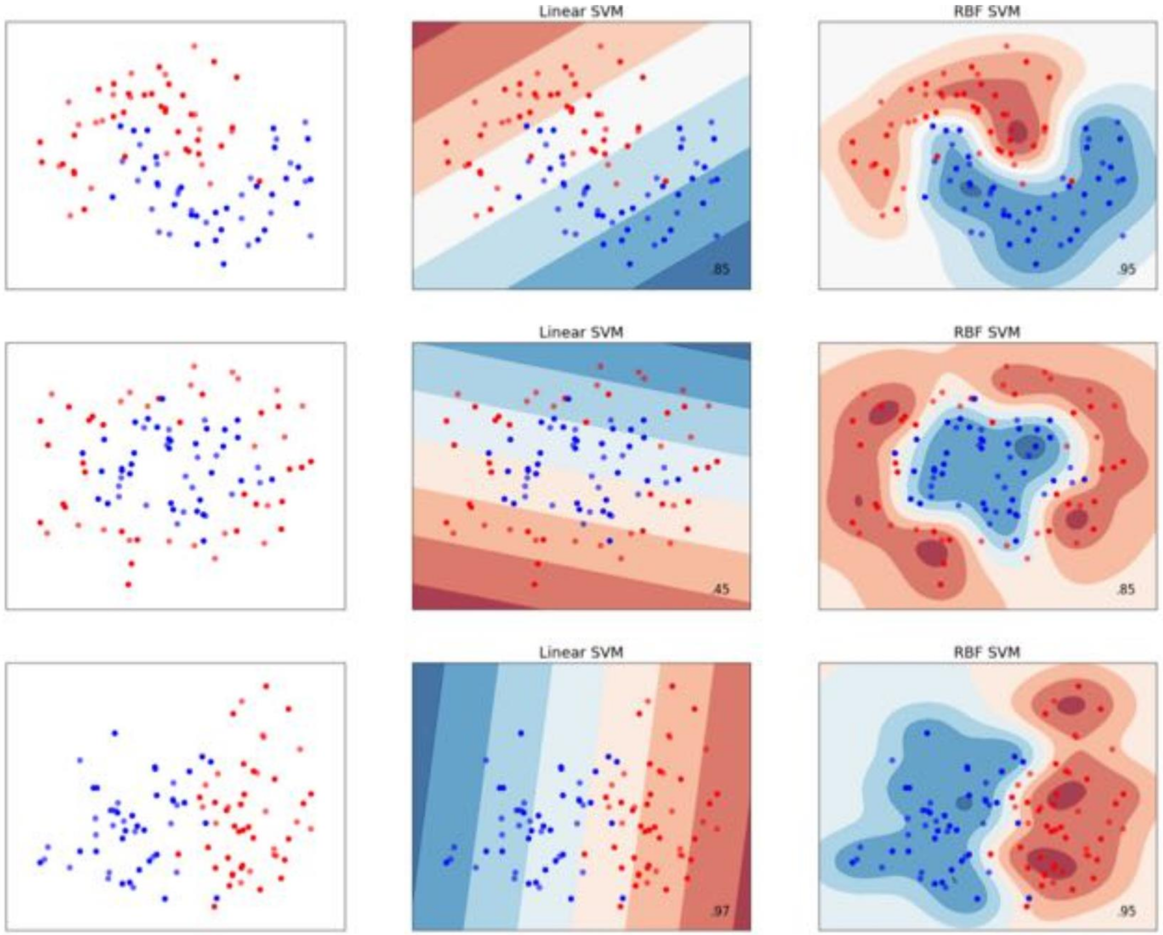
C in SVM

Large C: may be overfitted
Less C: may be underfitted



(b)

Regularization on Soft Margin



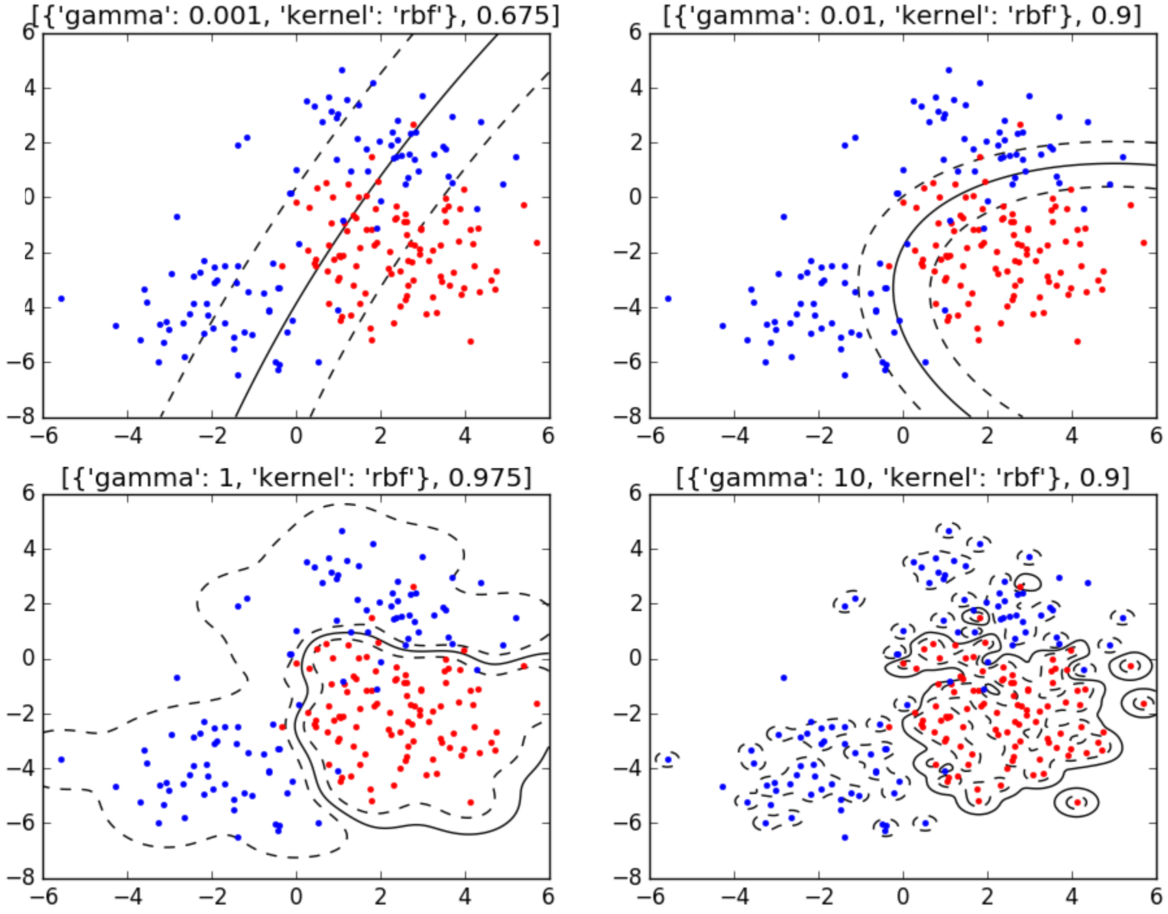
ML: Single Node Processor (SVM)

Loss function with regularization

- L1 Regularization

$$\ell(y, z) = \max(0, 1 - yz) + \boxed{C\sum_i^N \xi_i}$$

$$h_\theta(X) = exp\left(-\gamma\|x_i - C_j\|^2\right)$$

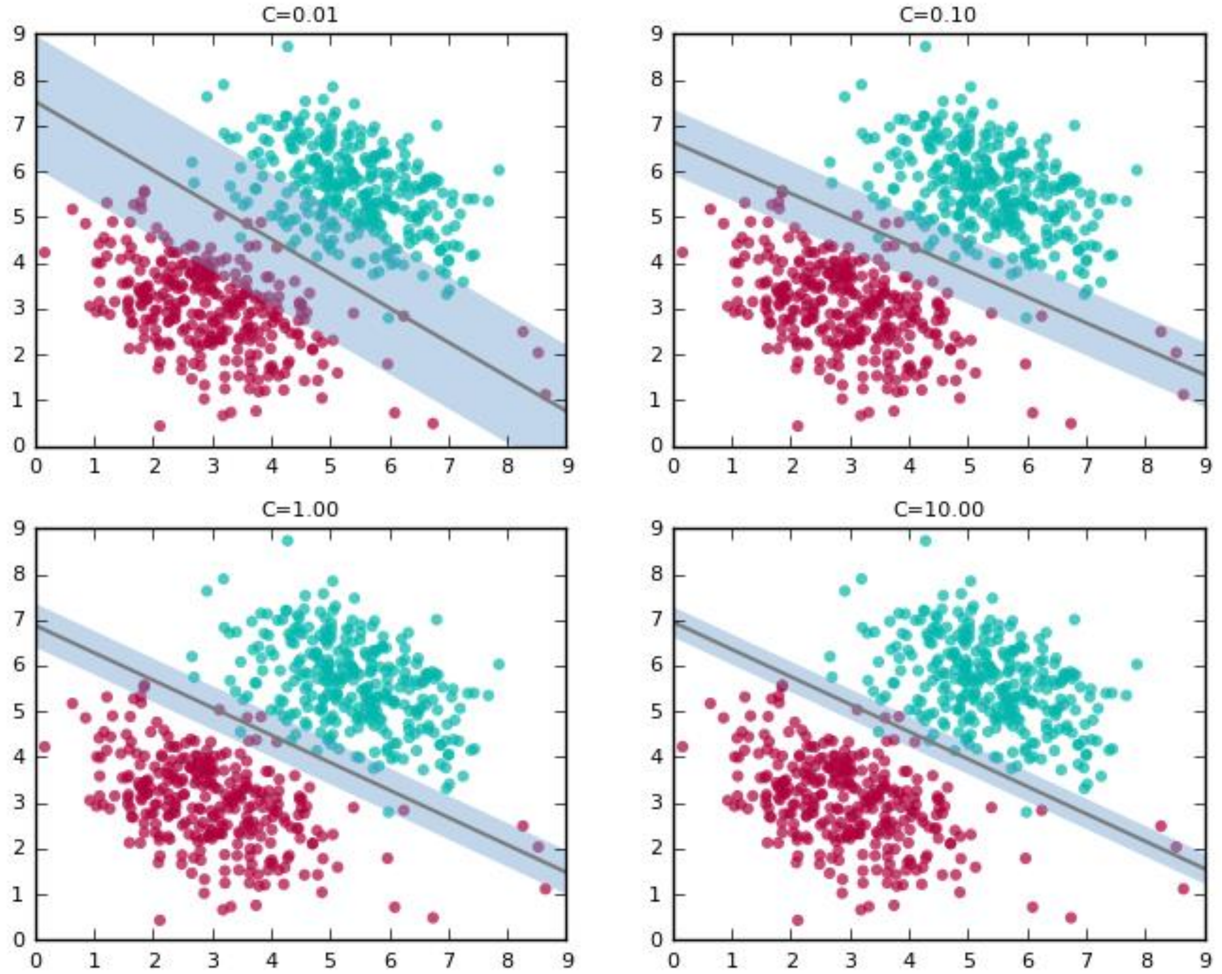


Gamma (γ) in SVM

Large γ : may be overfitted
Less γ : may be underfitted

Regularization on Soft Margin

SVC: Tuning parameters (Gamma)

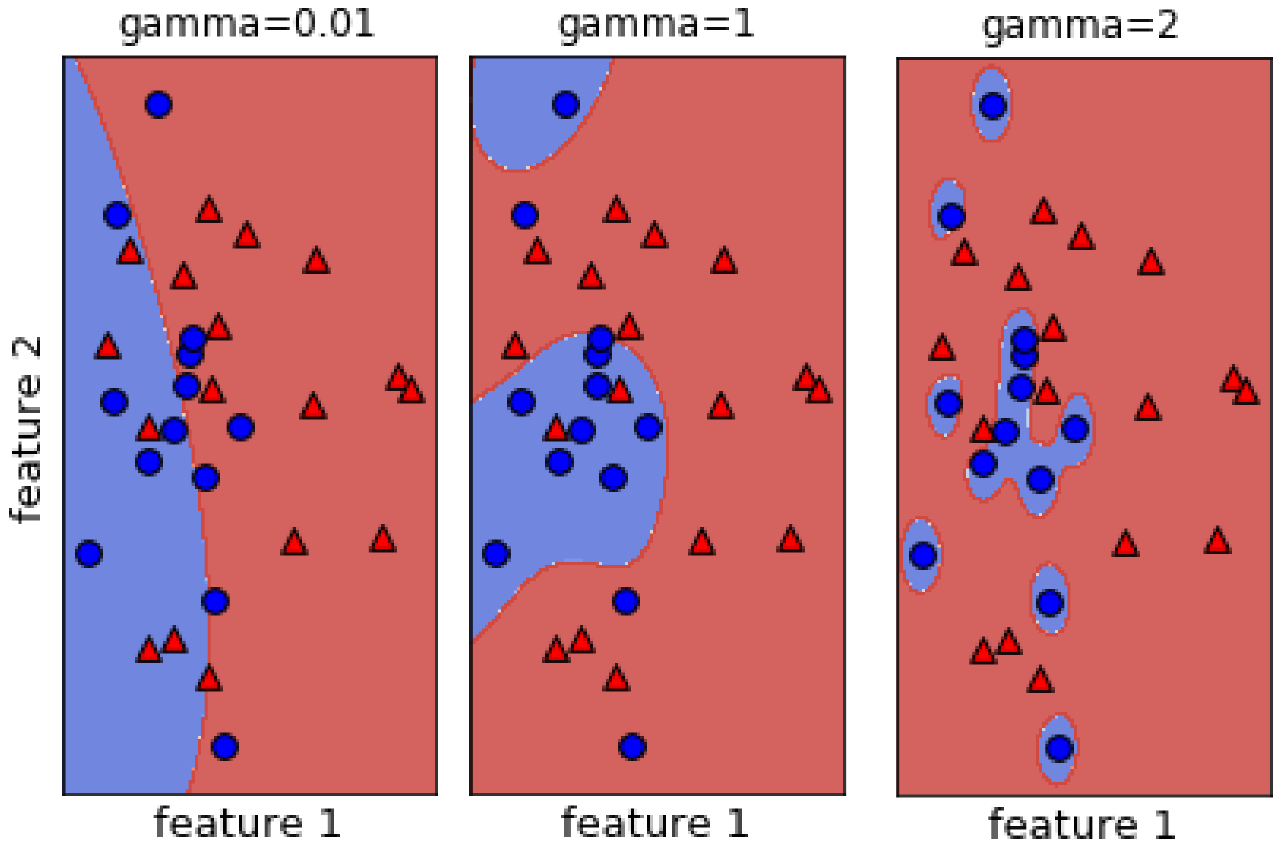


- General C parameters
  - Default: C=1

  - For large values of C,
    - choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly.

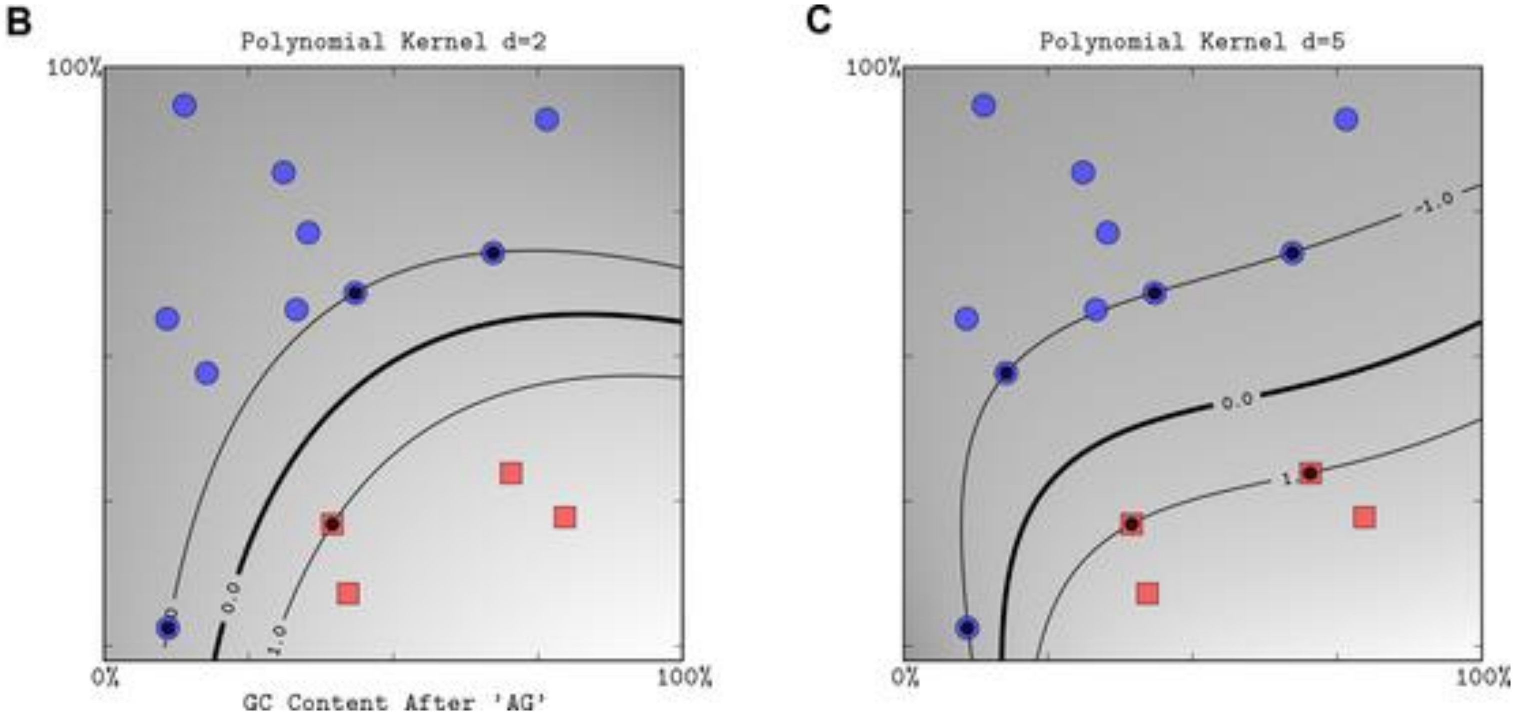
  - a very small value of C,
    - look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

SVC: Tuning parameters (Gamma)



- Gamma parameters
  - Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’
  - Higher the value of gamma,
    - try to exact fit the as per training data set
    - i.e. generalization error and cause over-fitting problem.
  - Default: ‘auto’ = 1/n_features
    - ‘scale’ = 1 / (n_features * X.var())

SVC: Tuning parameters (Degree)



- Degree parameters
  - Kernel coefficient for ‘poly’
  - Higher the value of gamma,
    - try to exact fit the as per training data set
    - i.e. generalization error and cause over-fitting problem.
  - Default: degree=3

SVM Classification (SVC)

- from sklearn.model_selection import train_test_split
  - X_train, X_test, y_train, y_test =train_test_split(X, y, test_size = 0.20)

- from sklearn.svm import SVC
  - svclassifier = SVC(kernel='kernel_name', parameters)
    - svclassifier = SVC(kernel='rbf', C=100, gamma=0.01)
  - svclassifier.fit(X_train, y_train)
  - y_pred = svclassifier.predict(X_test)

- cv = define cross validation you want to use with cross validation parameters
- C_range = [0.01, 0.1, 10, 100]
- gamma_range = [0.001, 0.01, 0.1]
- param_grid = dict(gamma=gamma_range, C=C_range)
- grid = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
- grid.fit(X, y)

Multi-class Classification

multi-class Classification

- Normally Logistic and SVC is only a binary classifier:
  - that is, it can only classify two classes at a time.
- Therefore, in order to classify multiple classes (more than two)
  - it has to train two or more binary classifiers
- Multi-class classification
  - “one-vs-one” approach
  - “one-vs-all” approach

Multi-class SVM training 'one-vs-one'

- Assume n_class as n different classes

- Two pairs of classes are selected at a time and a binary classifier trained for them.
  - If n_class is the number of classes,
    - then n_class * (n_class - 1) / 2 classifiers are constructed and
    - each one trains data from two classes.

  - During the classification phases
    - all the binary classifiers are tested.
    - The class with the most votes wins.

Multi-class SVM training ‘one-vs-all’

- train one classifier per class
  - n_class -> total n classifiers.
    - Ground truth = class ith
      - Class label ith -> positive
      - The rest -> negative

  - Generic SVM might not work,
    - but still there are some workarounds.