



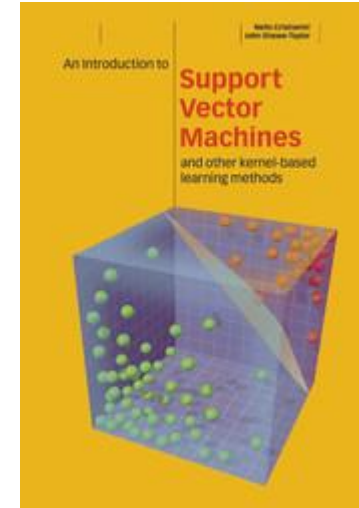
Support Vector Machines and Other Kernel Methods

Assoc. Prof. Karl Ezra Pilario, Ph.D.

Process Systems Engineering Laboratory
Department of Chemical Engineering
University of the Philippines Diliman

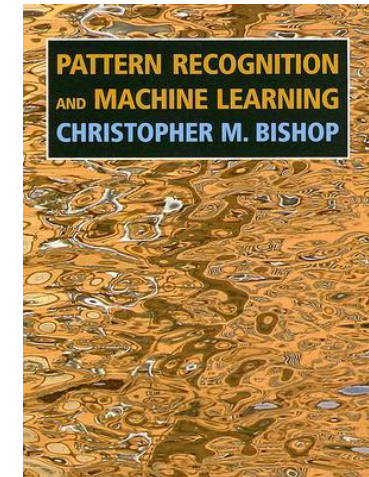
Outline

- Support Vector Machines
 - Large-Margin Classifiers
 - Quadratic Optimization Problem
 - Nonlinear SVM using Kernels
 - Mercer's Theorem
 - Multiple Kernel Learning
- Multi-class Classification
- Kernel Methods for Regression
 - Support Vector Regression
 - Kernel Ridge Regression



Cristianini & Shawe-Taylor (2000)
An Introduction to Support Vector Machines and other kernel-based learning methods.
Cambridge University Press.

Bishop (2006)
Pattern Recognition and Machine Learning. Springer.



Review

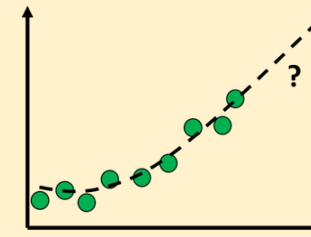
Supervised Learning

Learn a mapping or a function:

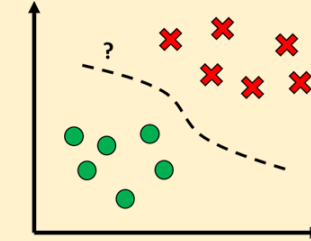
$$y = f(x)$$

from inputs (x) to outputs (y),
given a labelled set of input-output
examples (● or ✕).

Regression



Classification



Prediction Model

Cost Function

Linear Regression

$$y = w_0 + w_1 x$$

$$y = \mathbf{w}^T \boldsymbol{\phi}(x)$$

$$\min_{\mathbf{w}} f(\mathbf{w}) = \sum_{i=1}^N (y_i - \mathbf{w}^T \boldsymbol{\phi}(x_i))^2$$

Sum-of-squares loss

Logistic Regression

$$y = g(w_0 + w_1 x) \quad g(z) = \frac{1}{1 + e^{-z}}$$

$$y = g(\mathbf{w}^T \boldsymbol{\phi}(x))$$

$$\min_{\mathbf{w}} f(\mathbf{w}) = \sum_{i=1}^N -y_i \log(g(\mathbf{w}^T \boldsymbol{\phi}(x_i))) - (1 - y_i) \log(1 - g(\mathbf{w}^T \boldsymbol{\phi}(x_i)))$$

Cross-entropy loss or log-loss

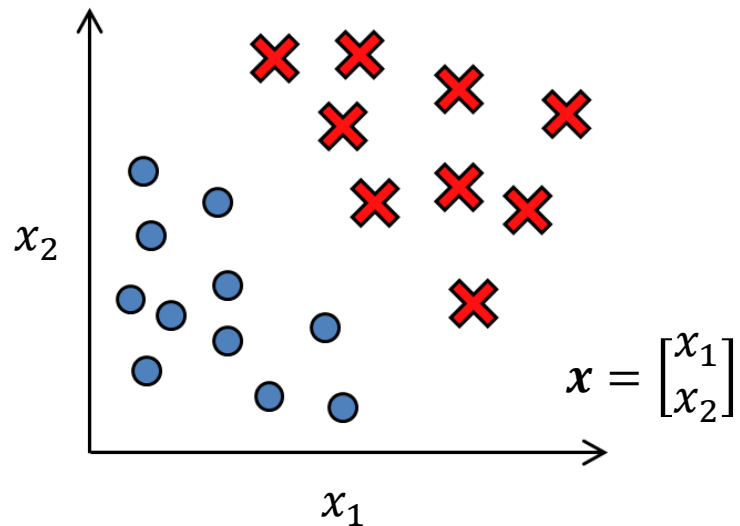
Support Vector Machine

???

???

Large-margin Classifiers

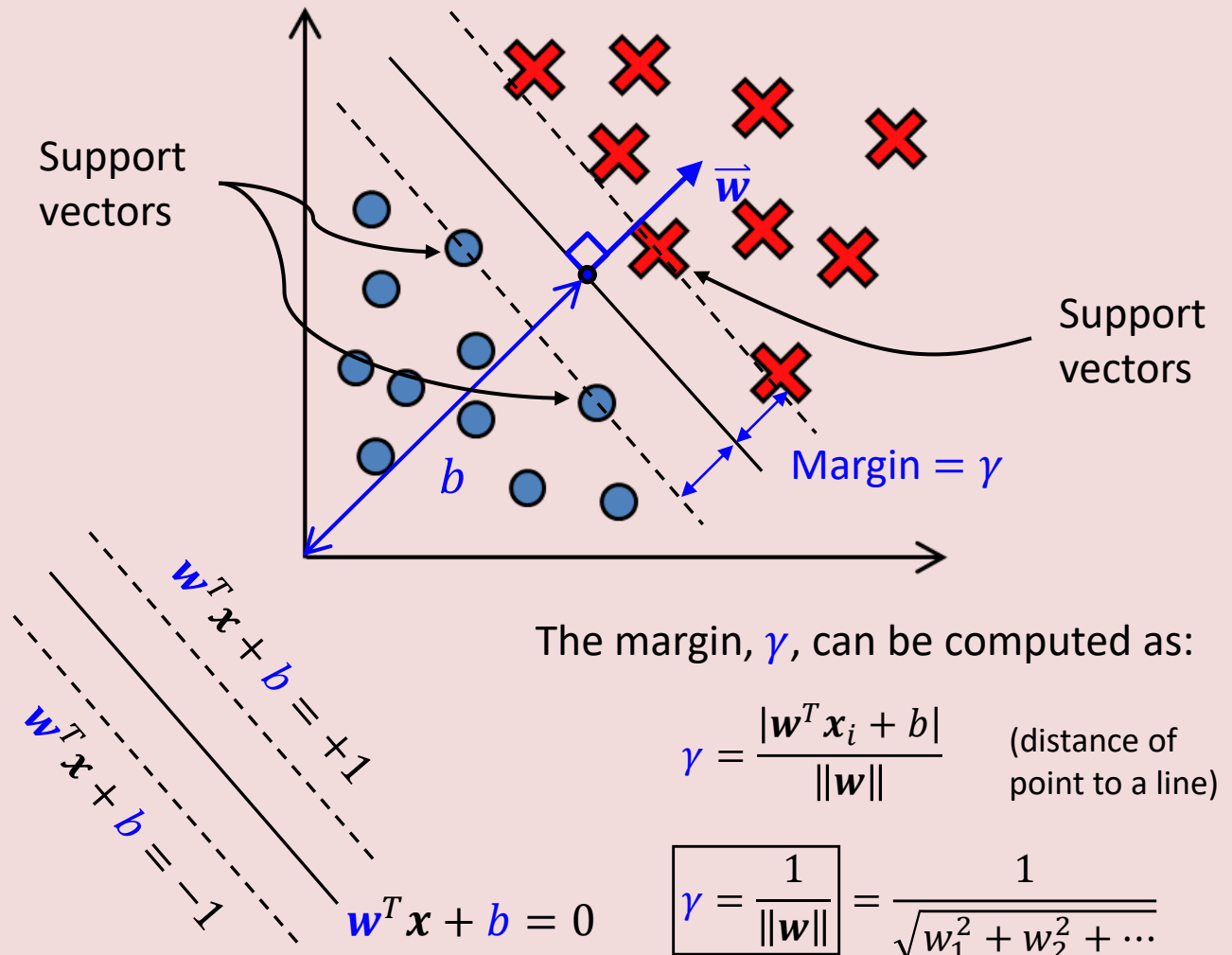
Consider a classification data set whose samples from different classes *can be separated by a line* (linearly separable):



There exists many straight lines that can separate the positive from the negative samples.

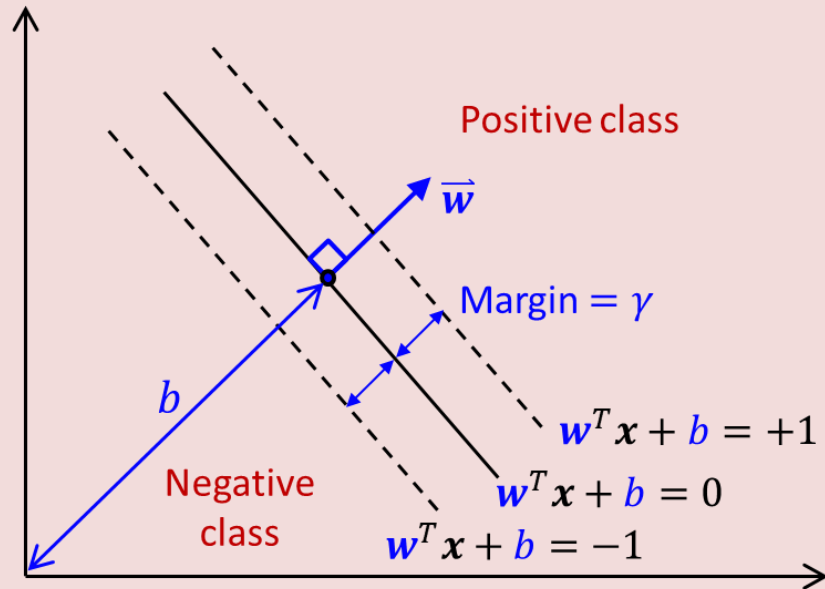
What is the equation of the *best-separating line*?

One possible answer: The one with the *largest margin for error*.



Large-margin Classifiers

Large-margin Classifiers:



Margin:
$$\gamma = \frac{1}{\|\mathbf{w}\|} = \frac{1}{\sqrt{w_1^2 + w_2^2 + \dots}}$$

If we want to maximize the margin, we can form the following **quadratic minimization problem**:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 = \frac{1}{2} \mathbf{w}^T \mathbf{w}$$

Subject to the following **constraints**, for all $i = 1, \dots, N$:

If a sample is from the **positive** class ($y_i = 1$):

$$\mathbf{w}^T \mathbf{x}_i + b \geq +1$$

If a sample is from the **negative** class ($y_i = -1$):

$$\mathbf{w}^T \mathbf{x}_i + b \leq -1$$

This solves the maximal margin classifier for the linearly separable case:

$$\min_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \mathbf{w}$$

$$\text{Subject to: } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

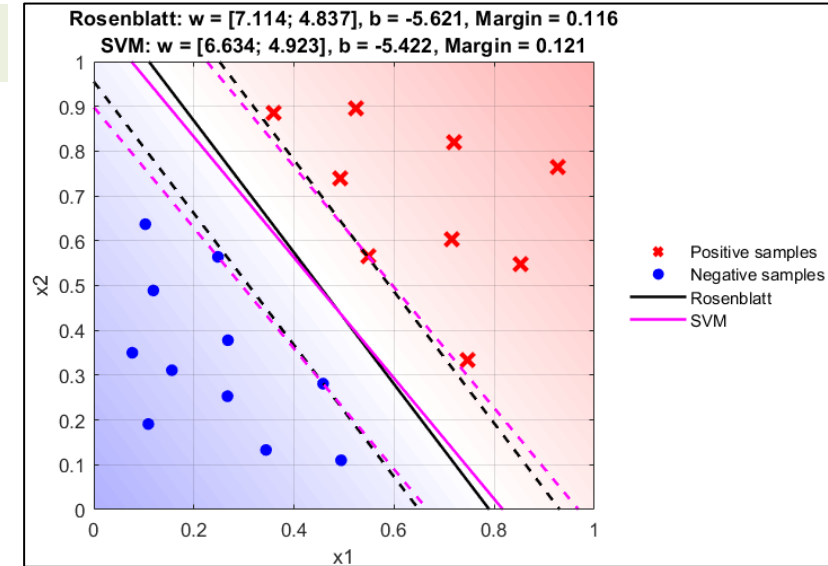
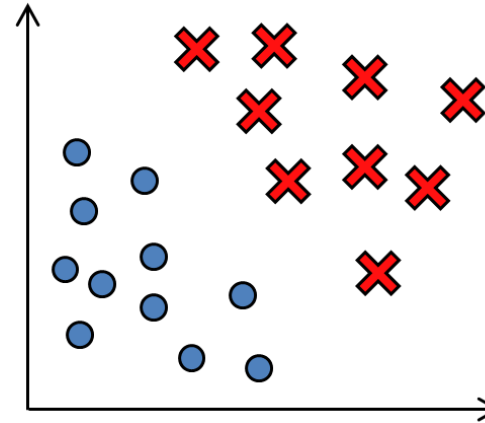
$$\text{for } i = 1, 2, \dots, N$$

Rosenblatt's Perceptron

An alternative to the maximum-margin classifier

- The first iterative algorithm for learning linear classifications (Frank Rosenblatt, 1956).
- An on-line, mistake-driven procedure
- Initialize \mathbf{w} , b , then update their values each time a training point is misclassified.
- Has a learning rate for updating weights.
- Guaranteed to converge but only for *linearly separable* data points.
- The final separating hyperplane can change depending on the order by which the points are iterated.
- Rosenblatt's perceptron is the basis of a modern neuron in **neural networks**.

In our example...



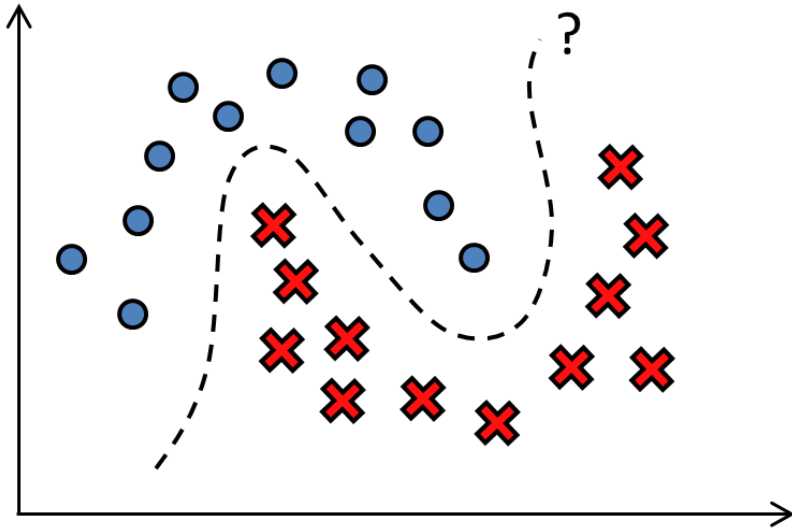
Rosenblatt's Perceptron Algorithm

```
Given training set  $\{\mathbf{x}_i, y_i\}$ ,  $y_i \in \{-1, 1\}$ 
Initialize:  $\mathbf{w} \leftarrow \mathbf{0}$ ,  $b \leftarrow 0$ ,  $k \leftarrow 0$ ,  $R \leftarrow \max_{1 \leq i \leq N} \|\mathbf{x}_i\|$ 
Repeat:
  for  $i = 1 \dots N$ 
    if  $y_i(\mathbf{w}^T \mathbf{x}_i + b) < 0$ 
       $\mathbf{w} \leftarrow \mathbf{w} + \eta y_i \mathbf{x}_i$ 
       $b \leftarrow b + \eta y_i R^2$ 
       $k \leftarrow k + 1$ 
    end if
  end for
Until no mistakes are made (i.e.  $k = 0$ )
Return  $(\mathbf{w}, b)$ 
```

In this implementation, I changed < 0 to < 1 so that we can enforce the rule that points are classified correctly only if they are *strictly outside the margin*.

Large-margin Classifiers

How about samples that cannot be separated by a line?



Kernel functions (or kernels):

- A *measure of similarity* between a sample x and any point in its surrounding space, x' .

- The **linear kernel** is given by:

$$K(x, x') = x^T x'$$

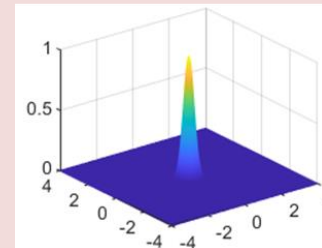
- A commonly used *nonlinear* kernel is the **Gaussian radial basis function (RBF) kernel**:

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

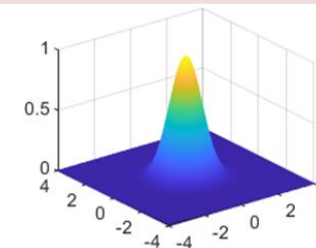
where σ = kernel width

If, say, $x = [0, 0]$:

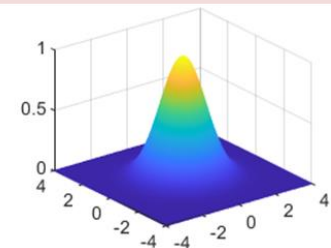
$\sigma = 0.1$



$\sigma = 1$



$\sigma = 2$



Large-margin Classifiers

Primal form:

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ \text{Subject to: } & y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \\ & \text{for } i = 1, 2, \dots, N \end{aligned}$$

This optimization problem has a dual form that can be derived using **Lagrange multipliers**:

Let α be the Lagrange multipliers (or dual variables):

$$\text{(Eq. 1)} \quad \mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1]$$

Differentiating with respect to \mathbf{w} and b :

$$\left. \begin{aligned} \frac{\partial \mathcal{L}(\mathbf{w}, b, \alpha)}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^N y_i \alpha_i \mathbf{x}_i = 0 \\ \frac{\partial \mathcal{L}(\mathbf{w}, b, \alpha)}{\partial b} &= \sum_{i=1}^N y_i \alpha_i = 0 \end{aligned} \right\}$$

Substituting Eq. 2 and 3 into Eq. 1:

$$\text{(Eq. 4)} \quad \mathcal{L}(\mathbf{w}, b, \alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j$$

We can now write the dual form:

Dual form:

$$\max_{\alpha} \quad \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{Subject to: } \sum_{i=1}^N y_i \alpha_i = 0$$

$$\alpha_i \geq 0, \quad \text{for } i = 1, 2, \dots, N$$

$$\mathbf{w} = \sum_{i=1}^N y_i \alpha_i \mathbf{x}_i \quad \text{(Eq. 2)}$$

$$\sum_{i=1}^N y_i \alpha_i = 0 \quad \text{(Eq. 3)}$$

The data set \mathbf{x} now only appears as a dot product, $\mathbf{x}_i^T \mathbf{x}_j$.
This is the **linear kernel**!

Support Vector Machines

Support Vector Machines

Linear SVM algorithm:

Dual form:

$$\max_{\alpha} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j \mathbf{x}_i^T \mathbf{x}_j$$

This is the
linear kernel!

$$\text{Subject to: } \sum_{i=1}^N y_i \alpha_i = 0$$

$$\alpha_i \geq 0, \quad \text{for } i = 1, 2, \dots, N$$

Kernel SVM algorithm:

Find α_i that
solves the
optimization
problem:

$$\max_{\alpha} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Can be any kernel,
e.g. RBF

$$\text{Subject to: } \sum_{i=1}^N y_i \alpha_i = 0$$

$$\alpha_i \geq 0, \quad \text{for } i = 1, 2, \dots, N$$

Mercer's Theorem

A necessary and sufficient condition for a symmetric function $K(\mathbf{x}_i, \mathbf{x}_j)$ to be a kernel is that it should be *positive definite*.

i.e. For any set of samples $\mathbf{x}_1, \dots, \mathbf{x}_l$ and any set of real numbers $\lambda_1, \dots, \lambda_l$, a positive definite function must satisfy:

$$\sum_{i=1}^l \sum_{j=1}^l \lambda_i \lambda_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

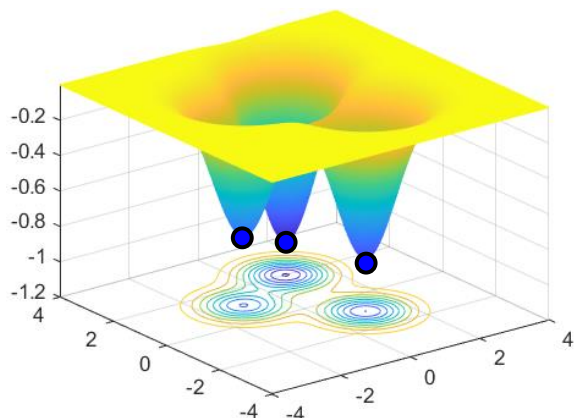
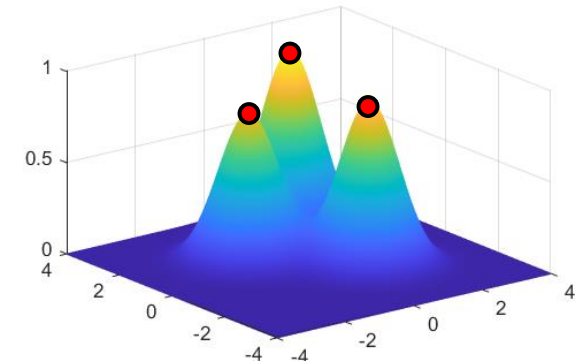
In that case, any Mercer kernel $K(\mathbf{x}_i, \mathbf{x}_j)$ can act as a dot product in nonlinear space.

“kernel trick”

Support Vector Machines

Why does the **RBF kernel function** lead to nonlinear decision boundaries?

Each point in the **positive** class is fitted to a “mountain” RBF.



Each point in the **negative** class is fitted to a “valley” RBF.

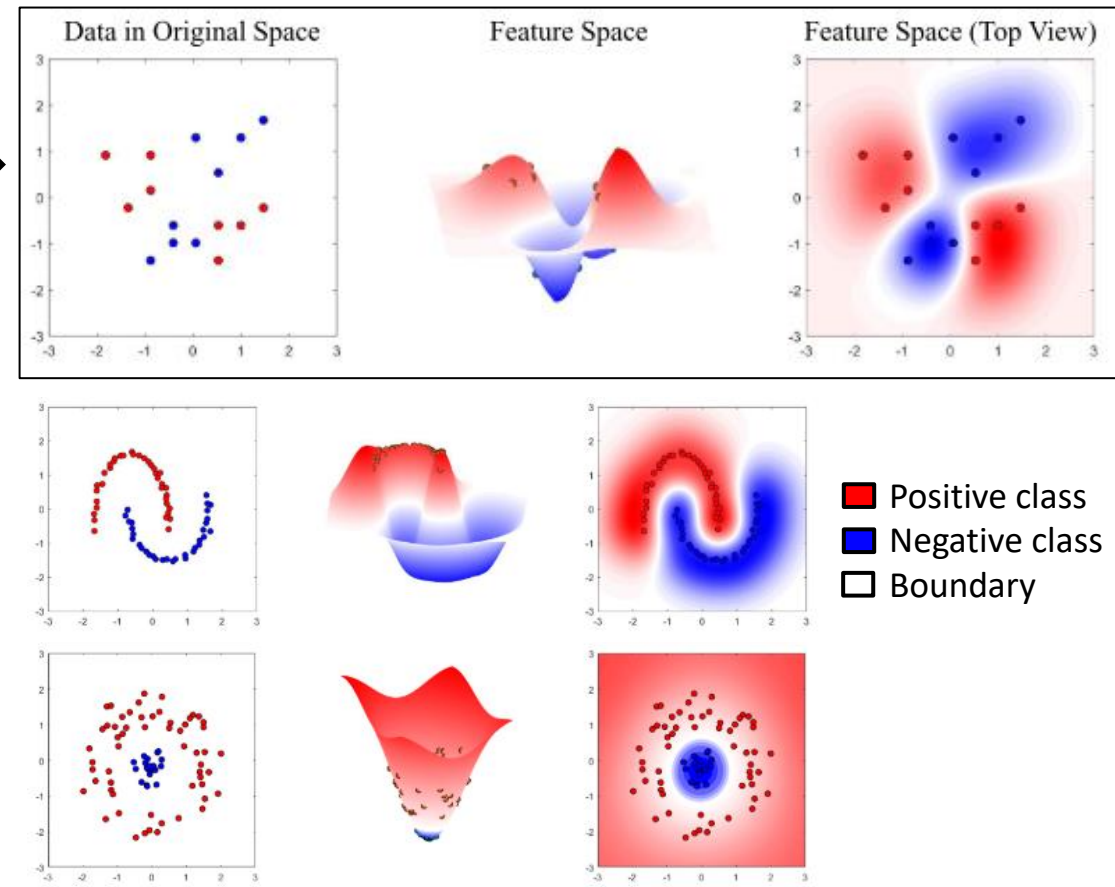
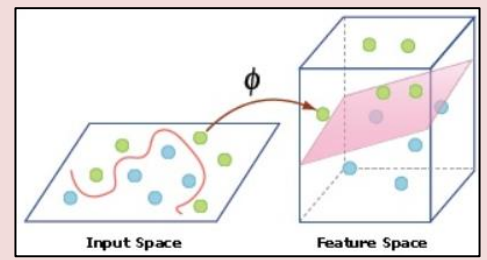
The *prediction model* in SVM is the **SUM** of all the **mountains** $+\alpha_i^* K(x_i, x')$ and **valleys** $-\alpha_i^* K(x_i, x')$:

$$f(x) = \sum_{i \in SV} y_i \alpha_i^* K(x_i, x') + b^*$$

(Representer Theorem) SV = set of support vectors

where:
 x_i are the input data
 $y_i \in \{-1, +1\}$ are their labels
 α_i^*, b^* are SVM parameters
 $K(x, x')$ is the kernel




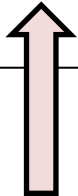
Main idea:
Kernel functions project the original data onto a *higher-dimensional feature space* where a *linear hyperplane* can more likely separate them (Cover, 1965).



Support Vector Machines

	Prediction Model	Cost Function	
Linear regression (Regression)	$y = \mathbf{w}^T \boldsymbol{\phi}(x_i)$	$\min_{\mathbf{w}} \sum_{i=1}^N (y_i - \mathbf{w}^T \boldsymbol{\phi}(x_i))^2$	(sum-of-squares loss)
Logistic regression (Classification)	$y = \frac{1}{1 + e^{-\mathbf{w}^T \boldsymbol{\phi}(x_i)}}$	$\min_{\mathbf{w}} \sum_{i=1}^N -y_i \log \left(\frac{1}{1 + e^{-\mathbf{w}^T \boldsymbol{\phi}(x_i)}} \right) - (1 - y_i) \log \left(1 - \frac{1}{1 + e^{-\mathbf{w}^T \boldsymbol{\phi}(x_i)}} \right)$	(cross-entropy loss or log-loss)
Support Vector Machines (Classification)	$y = \sum_{i \in SV} y_i \alpha_i^* K(x_i, x') + b^*$	$\max_{\alpha} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j K(x_i, x_j)$	Subject to: $\sum_{i=1}^N y_i \alpha_i = 0, \alpha_i \geq 0,$ for $i = 1, 2, \dots, N$ (maximum margin or hinge loss)
Artificial Neural Networks	????	????	
Etc...			

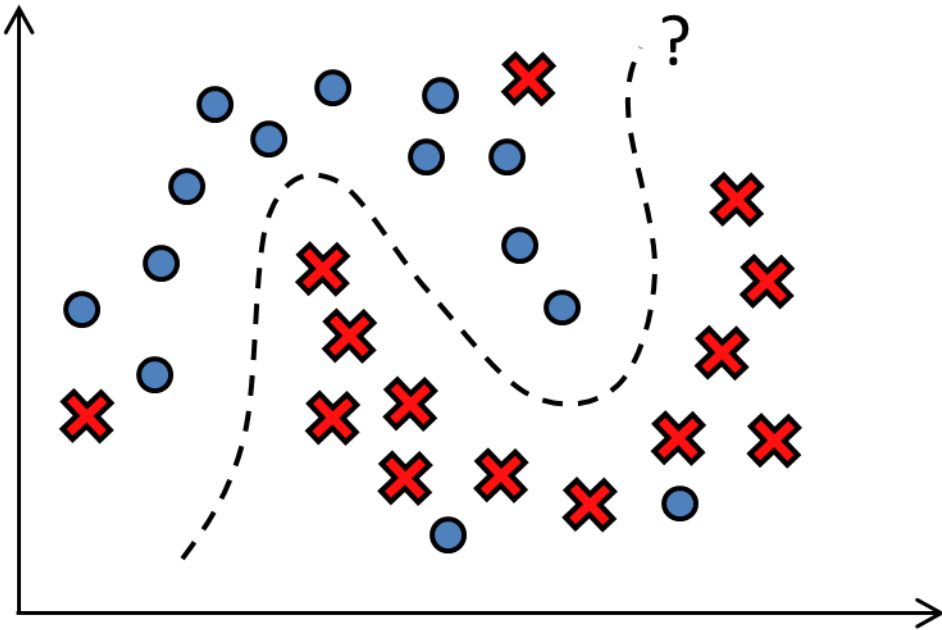
Parametric vs. Non-parametric Models

	Parametric Model	Non-Parametric Model
	<ul style="list-style-type: none">Model parameters (\mathbf{w}) are visible in the model, and they are explicitly fitted to data.Number of parameters are fixed with respect to the size of the training set.	<ul style="list-style-type: none">Only dual variables α_i are optimized, but the exact \mathbf{w} are unknown.Number of parameters α_i can grow with the size of the training data set.
Linear regression (Regression)	$y = \mathbf{w}^T \boldsymbol{\phi}(x)$ 	
Logistic regression (Classification)	$y = \frac{1}{1 + e^{-\mathbf{w}^T \boldsymbol{\phi}(x)}}$ 	
Support Vector Machines (Classification)	$y = \sum_{i \in SV} y_i \alpha_i^* K(\mathbf{x}_i, \mathbf{x}) + b^*$ 	
<p>Note: Locally Weighted Linear Regression is also a non-parametric model.</p>  <p>At test time, we need all the support vectors \mathbf{x}_i to measure their similarity with test data, \mathbf{x}</p>		

Support Vector Machines

What about outliers / noise in the data?

Real-world data are **noisy**. If we acknowledge that fact, we must tell the SVM that *a few misclassifications is okay*.



Solution: *Soft-margin SVM*

The 1-norm C-SVM employs a **box constraint**, C , on the values of α_i in the optimization problem:

$$\max_{\alpha} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{Subject to: } \sum_{i=1}^N y_i \alpha_i = 0$$

$$0 \leq \alpha_i \leq C, \quad \text{for } i = 1, 2, \dots, N$$

The C is a parameter that can be adjusted:

- If $C = \infty$, then hard-margin SVM is employed:
no misclassifications are tolerated.
- If $C < \infty$, then soft-margin SVM is employed:
the smaller the C , the more misclassifications are *tolerated*.

Support Vector Machines

Support Vector Machine: Summary

Given: Training Data, $\{x_i, y_i\}$, $y_i \in \{-1, 1\}$, $i = 1, \dots, N$

Initialization: Set the value of box constraint, C
Set a kernel function (e.g. RBF)
Set the kernel parameter value, σ

1: Solve the maximization problem:

$$\max_{\alpha} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N y_i y_j \alpha_i \alpha_j K(x_i, x_j)$$

Subject to: $\sum_{i=1}^N y_i \alpha_i = 0$

$$0 \leq \alpha_i \leq C, \quad \text{for } i = 1, 2, \dots, N$$

2: Compute the bias, b^* , such that on average, $y_i [\alpha_i^* K(x_i, x') + b^*] = 1$.

3: Return the classifier: $y_{\text{pred}} = \sum_{i \in SV} y_i \alpha_i^* K(x_i, x') + b^*$

Decision rule: If $y_{\text{pred}} > 0$, predict positive class.
If $y_{\text{pred}} < 0$, predict negative class.

In Python Scikit-learn, the **C-SVM** is implemented as **sklearn.svm.SVC**, with the following *defaults*:

- Box constraint, $C = 1$
- Kernel = RBF (Radial Basis Function)

List of possible kernels:

- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma \langle x, x' \rangle + r)^d$, where d is specified by parameter `degree`, r by `coef0`.
- rbf: $\exp(-\gamma \|x - x'\|^2)$, where γ is specified by parameter `gamma`, must be greater than 0.
- sigmoid $\tanh(\gamma \langle x, x' \rangle + r)$, where r is specified by `coef0`.

- Gamma, $\gamma = \frac{1}{\text{no. of features} \times \text{var}(x)}$

Other implementations of SVM in Scikit-learn are **sklearn.svm.NuSVC** and **sklearn.svm.LinearSVC**.

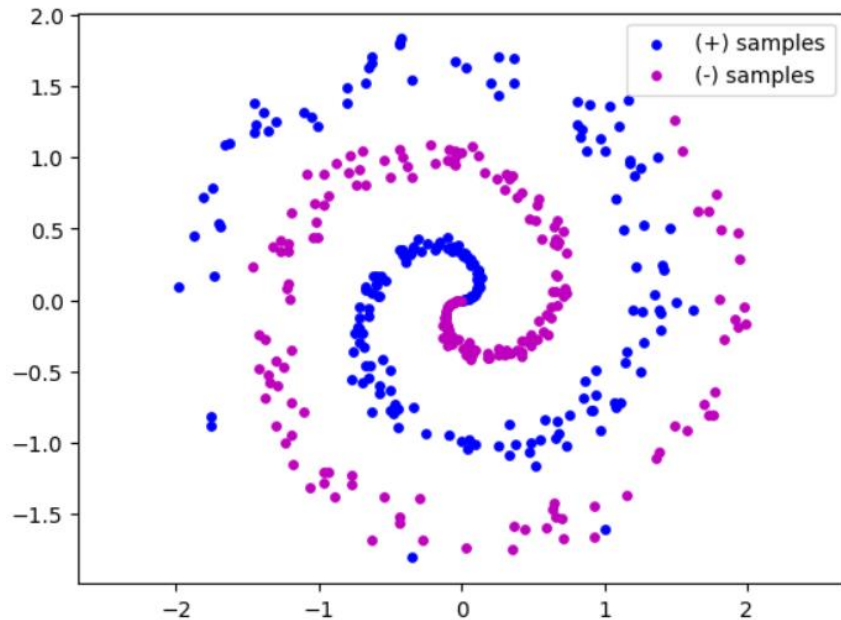
The maximization problem in SVMs is mainly solved using **SQP** (sequential quadratic programming) as implemented in `libsvm`.

Support Vector Machines

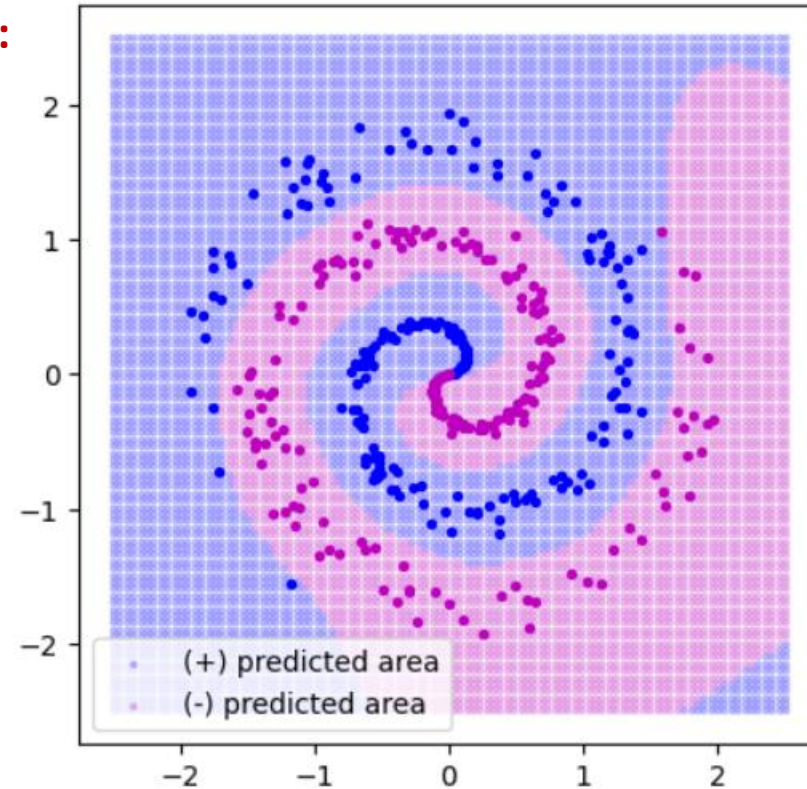
Example 1: Spiral Data Set

Find a maximal margin boundary between the blue and red points in the spiral data set below. Use an RBF kernel with a width of 1 and $C = 5$.

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{1.0}\right)$$



Answer:



where:

\mathbf{x}_i are the input data
 $y_i \in \{-1, +1\}$ are their labels
 α_i^*, b^* are SVM parameters
 $K(\mathbf{x}, \mathbf{x}')$ is the kernel
SV = set of support vectors

$$f(\mathbf{x}) = \sum_{i \in SV} y_i \alpha_i^* K(\mathbf{x}_i, \mathbf{x}') + b^*$$

Underfitting vs. Overfitting

Underfitting

- Performance is too low on the training data set.
- Patterns were not learned at all.
- The model is too simple (high bias, low variance).
- The model cannot generalize to unseen test data.

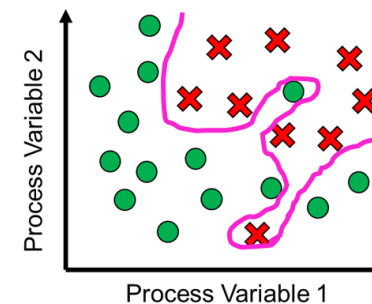
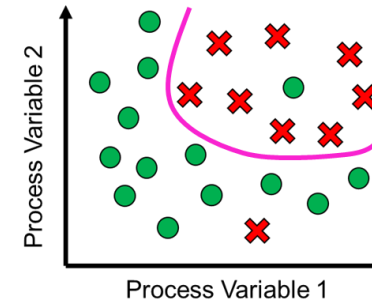
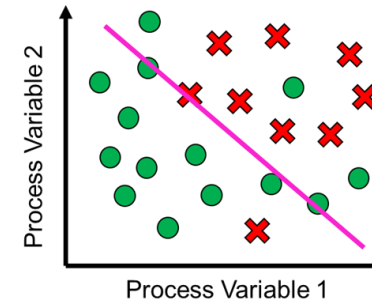
Good fit

- Performance on training data is decent.
- The pattern was learned.
- Bias and variance are balanced.
- Model is not too complex nor simple.
- The model *may* generalize well to unseen test data.

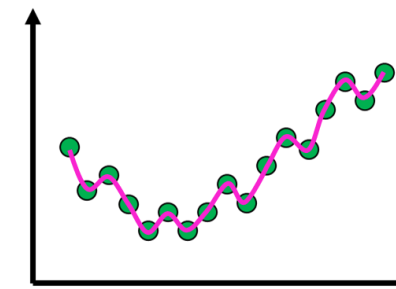
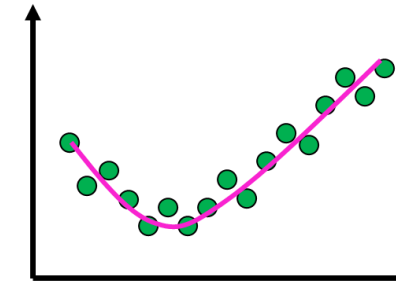
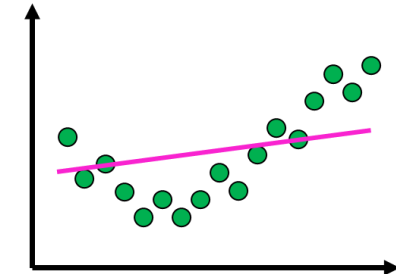
Overfitting

- Performance is too good to be true on the training data set.
- Patterns “learned” from noise.
- The model is too complex (low bias, high variance).
- The model cannot generalize to unseen test data.

Classification



Regression



Multiple Kernel Learning

- The name “kernel” is derived from *integral operator theory* in math.
- There are many kernel functions, other than the Gaussian RBF.
- One can create new Mercer kernels by combining base Mercer kernels.

Common kernels:

Gaussian RBF

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{\sigma}\right)$$

Sigmoid kernel

$$K(\mathbf{x}, \mathbf{x}') = \tanh(a\langle \mathbf{x}, \mathbf{x}' \rangle + b)$$

Polynomial kernel

$$K(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + 1)^d$$

Linear kernel

$$K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$$

Creating kernels: If $K_1(\mathbf{x}, \mathbf{x}')$ and $K_2(\mathbf{x}, \mathbf{x}')$ are valid **Mercer kernels**, then the following functions are also valid Mercer kernels.

- $K(\mathbf{x}, \mathbf{x}') = K_1(\mathbf{x}, \mathbf{x}') + K_2(\mathbf{x}, \mathbf{x}')$
- $K(\mathbf{x}, \mathbf{x}') = K_1(\mathbf{x}, \mathbf{x}')K_2(\mathbf{x}, \mathbf{x}')$
- $K(\mathbf{x}, \mathbf{x}') = aK_1(\mathbf{x}, \mathbf{x}')$
- $K(\mathbf{x}, \mathbf{x}') = \exp(K(\mathbf{x}, \mathbf{x}'))$
- $K(\mathbf{x}, \mathbf{x}') = p(K(\mathbf{x}, \mathbf{x}'))$, p is a polynomial with non-negative coefficients.

Related readings:

- Multiple kernel learning
- Compositional kernel
- Mixed kernels
- Hierarchical kernels
- Neural kernel blocks

Other kernels in other fields:

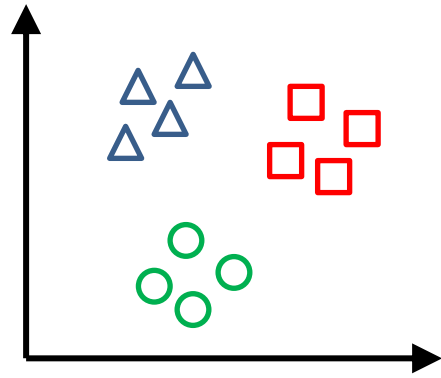
- **String kernels** are used in text mining and bioinformatics (gene analysis).
- **Matern kernels** are used for Gaussian process regression / kriging in geostatistics.
- **Clinical kernels** are used in analyzing clinical data / survival analysis in medicine.

Outline

- Support Vector Machines
 - Large-Margin Classifiers
 - Quadratic Optimization Problem
 - Nonlinear SVM using Kernels
 - Mercer's Theorem
 - Multiple Kernel Learning
- **Multi-class Classification**
- Kernel Methods for Regression
 - Support Vector Regression
 - Kernel Ridge Regression

Multi-class Classification

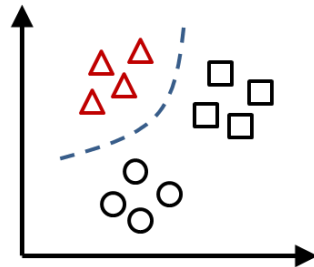
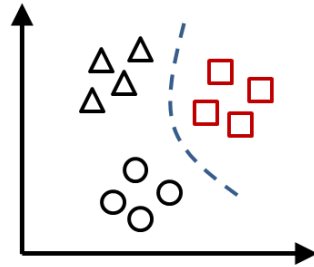
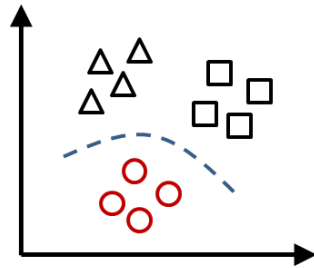
Sample data set:



One-vs.-Rest

(OvR, One-vs.-All, 1-v-R)

Given K classes, train K binary classifiers, each one treating one class as positive and the rest as negative.

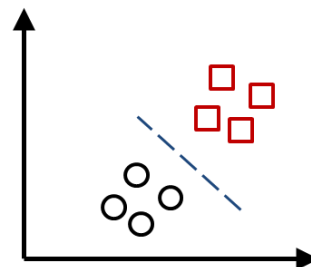
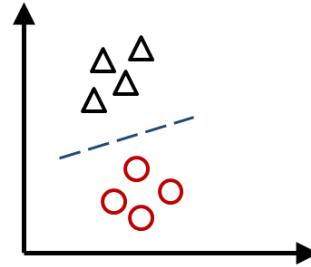
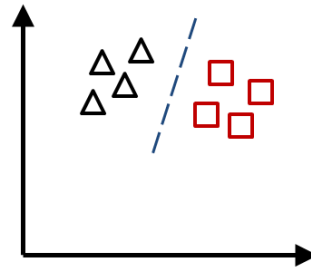


Note: The default multi-class strategy of `sklearn.svm.SVC` is OvO.

One-vs.-One

(OvO, 1-v-1)

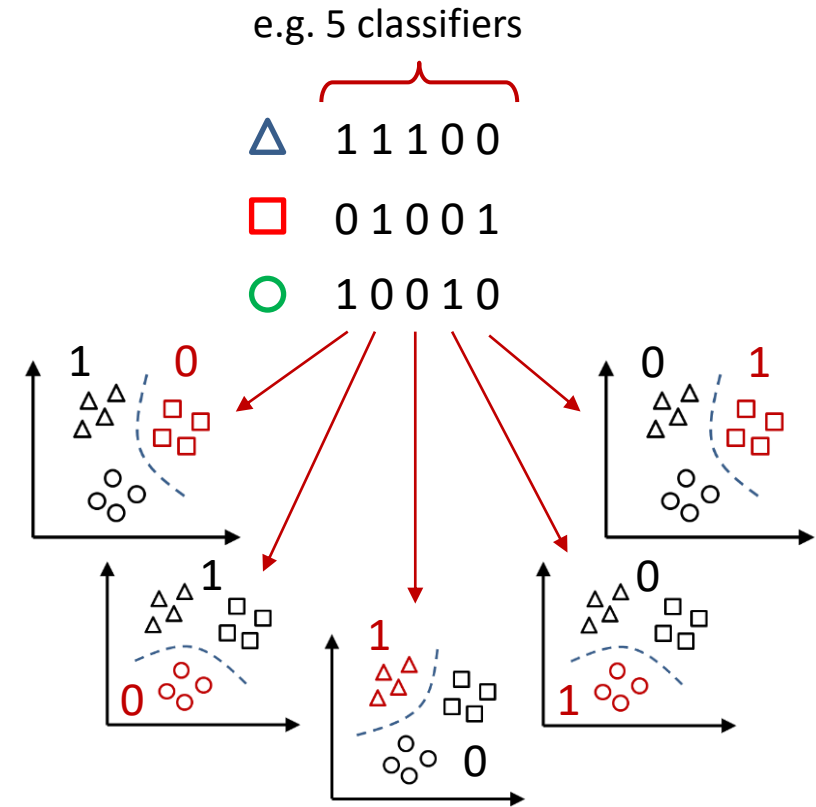
Given K classes, train $K(K - 1)/2$ binary classifiers, one for each pair of classes. Predict by majority voting.



Output Codes

(Error-correcting output codes, ECOC)

Given K classes, assign a unique binary codeword for each class, then train a binary classifier for each bit position in the codeword. Predict by generating the output codeword of a new sample, then report the class with the closest codeword to it.



Multi-class Classification

Example 2: Iris Flowers Data Set

- 150 flowers from 3 different Iris species (50 each)
- Measurements of sepal and petal lengths and widths.

Using only the **sepal length** and **petal length** features, train an SVM to classify all 150 flowers into their correct species.

Compare the results between the OvO, OvR, and ECOC strategies.

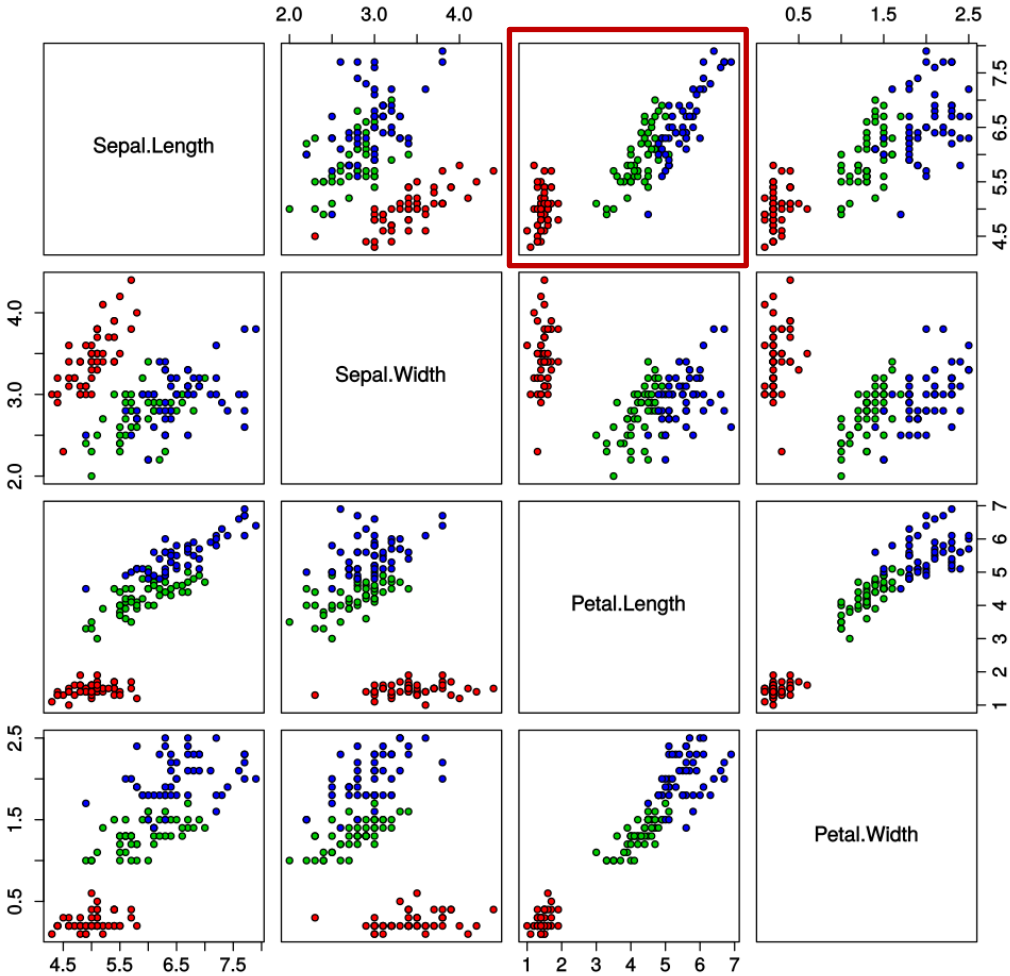
Use default SVM settings. For ECOC, use codewords of length 9.

Sepal Length	Sepal Width	Petal Length	Petal Width	Classification
cm	cm	cm	cm	
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
...
7	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
...
6.3	3.3	6	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
7.1	3	5.9	2.1	Iris-virginica
...



Versicolor Setosa Virginica

Iris Data (red=setosa,green=versicolor,blue=virginica)



Multi-class Classification

Example 2: Iris Flowers Data Set

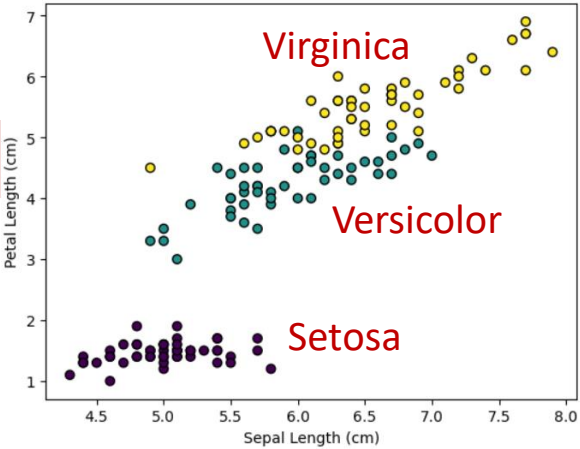
- 150 flowers from 3 different Iris species (50 each)
- Measurements of sepal and petal lengths and widths.

Using only the **sepal length** and **petal length** features, train an SVM to classify all 150 flowers into their correct species.

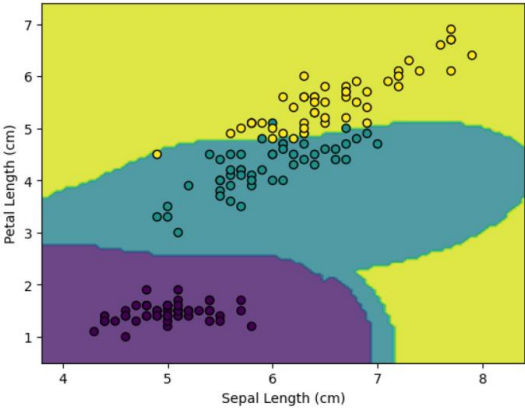
Compare the results between the OvO, OvR, and ECOC strategies.

Use default SVM settings. For ECOC, use codewords of length 9.

Data Set:



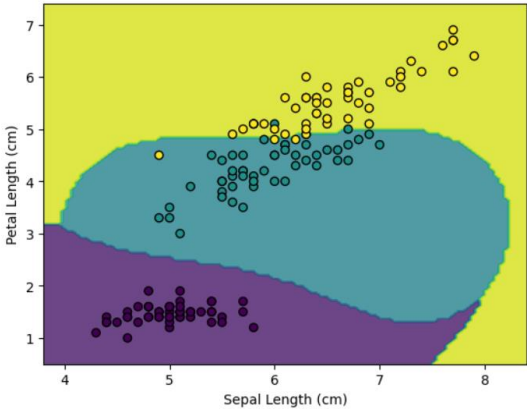
OvO Strategy



Actual	50	0	0
	0	47	3
	0	3	47

Predicted

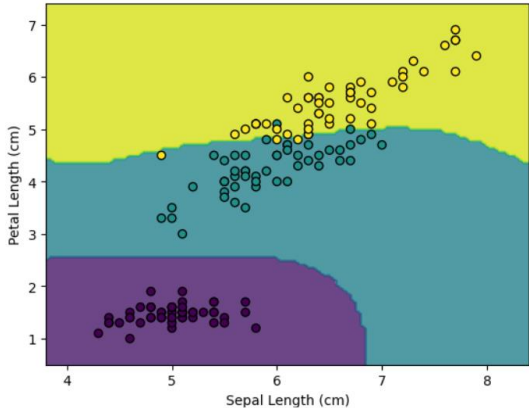
OvR Strategy



Actual	50	0	0
	0	48	2
	0	4	46

Predicted

ECOC Strategy



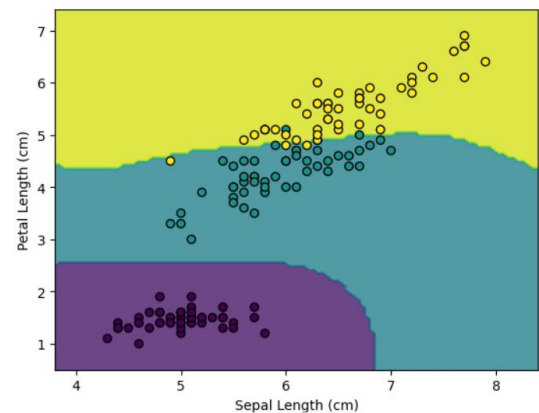
Actual	50	0	0
	0	48	2
	0	4	46

Predicted

Multi-class Classification

Example 2: Iris Flowers Data Set

ECOC Strategy



Actual	setosa	50	0	0
	versicolor	0	48	2
	virginica	0	4	46
		setosa	versicolor	virginica
		Predicted		

Performance Metrics for Multi-class Classification

Classification Report:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	50
versicolor	0.92	0.96	0.94	50
virginica	0.96	0.92	0.94	50
accuracy			0.96	150
macro avg	0.96	0.96	0.96	150
weighted avg	0.96	0.96	0.96	150

Sample calculations:

Precision = $\frac{48}{48 + 4} = 0.92$
(Versicolor)

50	0	0
0	48	2
0	4	46

Recall = $\frac{48}{48 + 2} = 0.96$
(Versicolor)

50	0	0
0	48	2
0	4	46

Overall Metrics

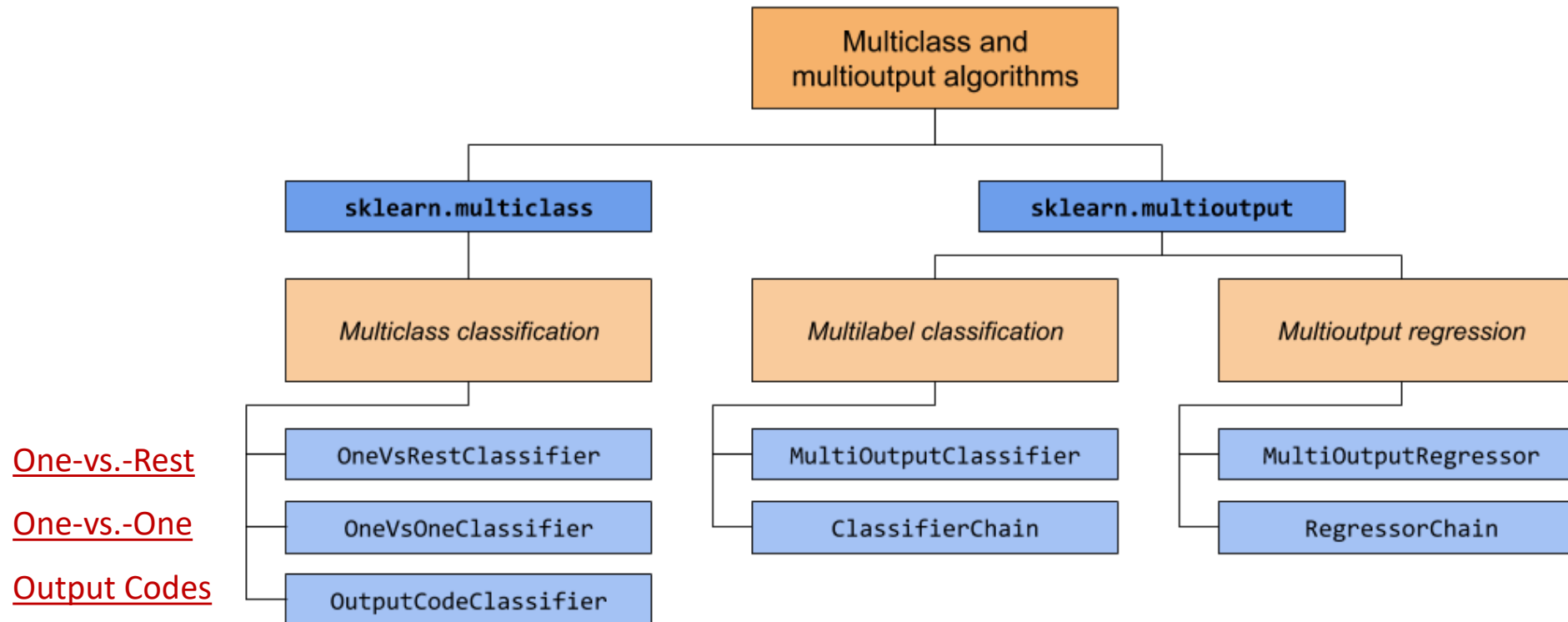
- Macro Average
(mean of metrics)
- Weighted Average
(mean weighted by number of samples)

Multi-class, multi-label, multi-output

In Python Scikit-learn, there are other kinds of classification learning tasks available:

- **Multi-class:** Two or more classes are available; each sample is labeled to one and only one class.
- **Multi-label:** Two or more classes are available; a sample can be labelled to one or more classes at once.
- **Multi-class, Multi-output:** Two or more classes are available;
A model can predict multiple outputs for one sample simultaneously.

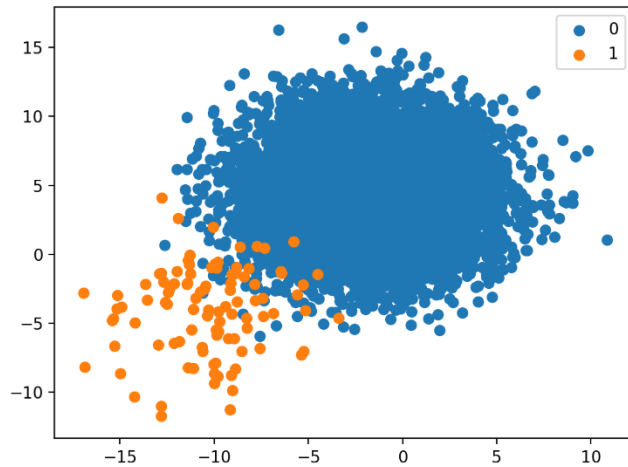
Multi-class, multi-output classification is also known as **multi-task classification**.



What about class imbalance?

Class imbalance occurs when classes are not approximately equally represented.

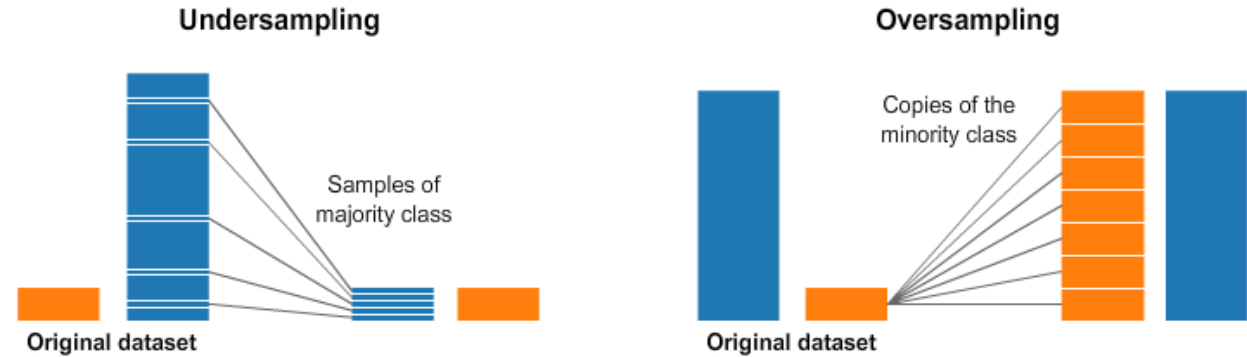
The majority class becomes easy to predict, while the minority class becomes hard.



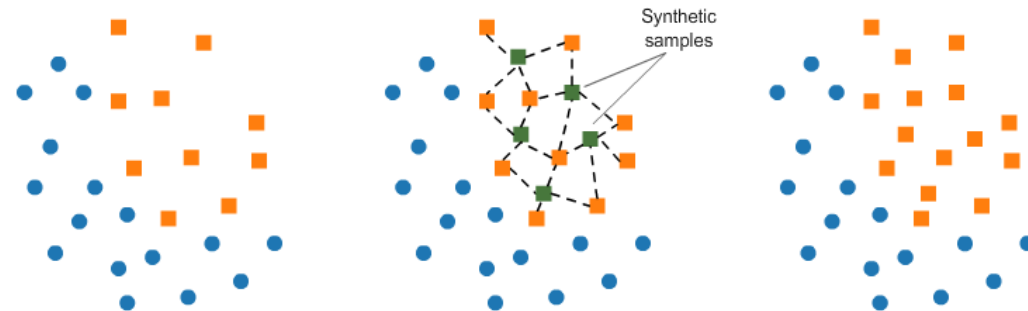
Examples:

- Some images are rare.
- There are only a few fraudulent checks.
- There are only a few faulty equipment data.
- There are more SFW content than NSFW content.
- Some words appear more often than others.

Solution 1: Random undersampling and oversampling



Solution 2: SMOTE (Synthetic Minority Oversampling Technique)



Other solutions:

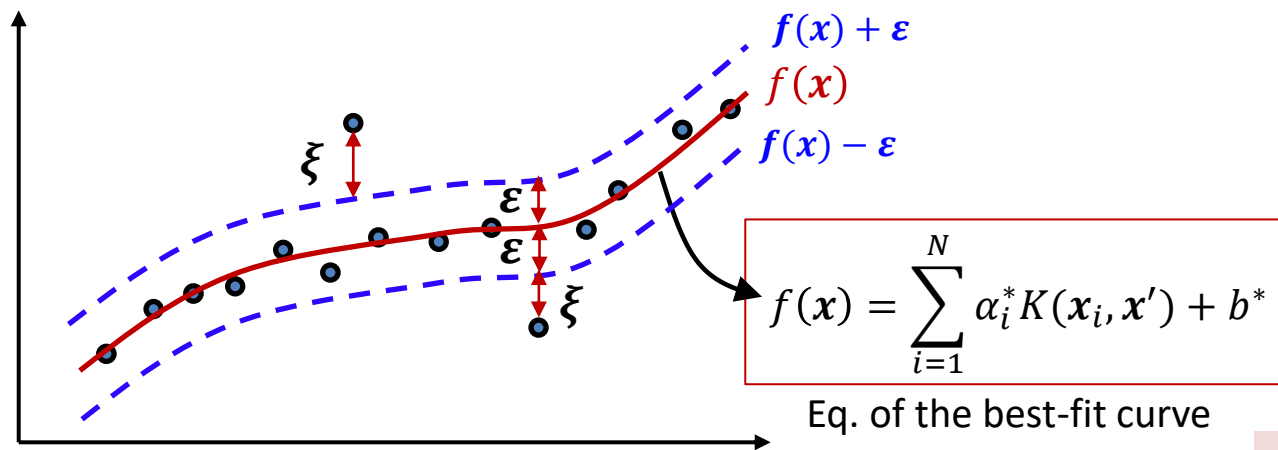
- **Do nothing.**
- Collect more data.
- Treat the problem as anomaly detection rather than classification.
- Include *class weights* in the classifier's cost function.
- Consider other evaluation metrics, like F1-score.

Outline

- Support Vector Machines
 - Large-Margin Classifiers
 - Quadratic Optimization Problem
 - Nonlinear SVM using Kernels
 - Mercer's Theorem
 - Multiple Kernel Learning
- Multi-class Classification
- **Kernel Methods for Regression**
 - Support Vector Regression
 - Kernel Ridge Regression

Support Vector Regression

- Support Vector Machines can also be formulated to solve regression instead of classification.
- Large-margin intuition \rightarrow ϵ -insensitive cost function (ϵ is 'epsilon')



In Support Vector Regression, points outside the ε -tube are the **support vectors**.

SVR: Dual Form of the Optimization Problem

$$\max_{\alpha} \sum_{i=1}^N y_i \alpha_i - \varepsilon \sum_{i=1}^N |\alpha_i| - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Subject to: $\sum_{i=1}^N \alpha_i = 0$

$$-C \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N$$

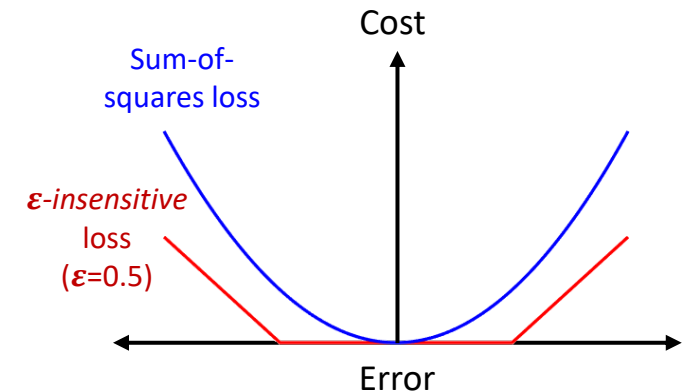
This optimization problem was derived from minimizing the *linear ϵ -insensitive loss / cost function*:

$$L_\varepsilon = \begin{cases} 0 & \text{if } |y - f(x)| \leq \varepsilon \\ |y - f(x)| - \varepsilon & \text{otherwise} \end{cases}$$

A more compact way to write it:

$$L_\varepsilon = \max(0, |y - f(x)| - \varepsilon)$$

The *linear ϵ -insensitive cost function* ignores errors that are within ϵ distance of the observed value by treating them as equal to zero.



Kernel Ridge Regression

From Linear Ridge Regression to Kernel Ridge Regression:

**Linear Basis
Function Model:**

$$y = w_0 + w_1\phi_1(x) + w_2\phi_2(x) + \cdots + w_m\phi_m(x)$$

$$y = \mathbf{w}^T \boldsymbol{\phi}(x) = \boldsymbol{\phi}(x)^T \mathbf{w}$$

By minimizing the cost
function with penalty:

$$\min_{\mathbf{w}} (\mathbf{y} - \boldsymbol{\Phi}\mathbf{w})^T (\mathbf{y} - \boldsymbol{\Phi}\mathbf{w}) + \lambda(\mathbf{w}^T \mathbf{w})$$

The solution is given by:

$$\hat{\mathbf{w}} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \mathbf{I})^{-1} \boldsymbol{\Phi}^T \mathbf{y}$$

Substituting the $\hat{\mathbf{w}}$ into the
model:

$$y_{\text{pred}} = \boldsymbol{\phi}(x)^T ((\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \mathbf{I})^{-1} \boldsymbol{\Phi}^T \mathbf{y})$$

Using the Woodbury matrix
identity:

$$y_{\text{pred}} = \boldsymbol{\phi}(x)^T \boldsymbol{\Phi} (\boldsymbol{\Phi} \boldsymbol{\Phi}^T + \lambda \mathbf{I})^{-1} \mathbf{y}$$

Applying the **kernel trick**:

$$K(\mathbf{x}', \mathbf{x}') = \mathbf{K} = \boldsymbol{\Phi} \boldsymbol{\Phi}^T \quad \text{Kernel matrix or Gram matrix}$$

$$k(x, x') = \mathbf{k}(x) = \boldsymbol{\phi}(x)^T \boldsymbol{\Phi}$$

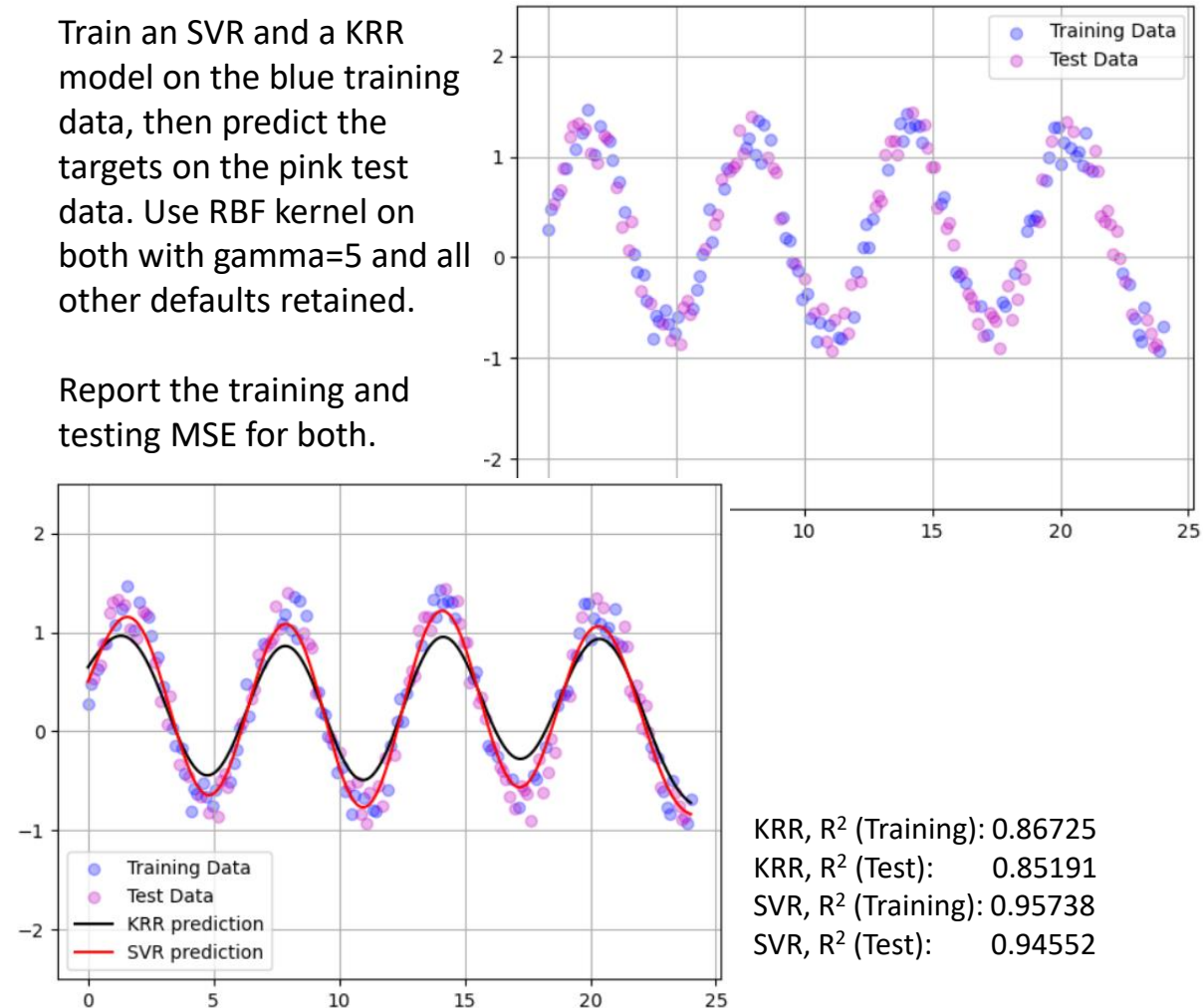
We have now derived
Kernel Ridge Regression:

$$y_{\text{pred}} = \mathbf{k}(x)^T (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}$$

Example 3: Sine Data Set (again)

Train an SVR and a KRR model on the blue training data, then predict the targets on the pink test data. Use RBF kernel on both with gamma=5 and all other defaults retained.

Report the training and testing MSE for both.



A closer look at the RBF kernel...

We learned that the reason why the **kernel trick** works is that the kernel function acts as a **dot product** of some **feature map** $\phi(\mathbf{x})$ of the input \mathbf{x} :

$$K(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

So, for the **RBF kernel**, what does the $\phi(\mathbf{x})$ correspond to?

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{\sigma}\right) = \langle ?, ? \rangle$$

Derivation:

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2}\right) = \exp\left(-\frac{1}{2}(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\langle \mathbf{x}, \mathbf{x}' \rangle)\right) \\ &= \exp\left(-\frac{1}{2}(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2)\right) \exp\left(-\frac{1}{2}(-2\langle \mathbf{x}, \mathbf{x}' \rangle)\right) \\ &= c \exp\left(-\frac{1}{2}(-2\langle \mathbf{x}, \mathbf{x}' \rangle)\right) = c \exp(\langle \mathbf{x}, \mathbf{x}' \rangle) \\ &= c \sum_{n=0}^{\infty} \frac{\langle \mathbf{x}, \mathbf{x}' \rangle^n}{n!} = c \left[1 + \langle \mathbf{x}, \mathbf{x}' \rangle + \frac{\langle \mathbf{x}, \mathbf{x}' \rangle^2}{2!} + \frac{\langle \mathbf{x}, \mathbf{x}' \rangle^3}{3!} + \dots \right] \end{aligned}$$

Since the Gaussian RBF kernel is an infinite power series, it corresponds to an **infinite-dimensional feature map**, $\phi(\mathbf{x}) \in \mathcal{R}^{\infty}$. More formally, the underlying feature space is a **Reproducing Kernel Hilbert Space (RKHS)**.

Smaller example: Given $\mathbf{x} = [x_1, x_2]$, the quadratic kernel $(\mathbf{x} \cdot \mathbf{x}')^2$ can be derived from the feature map:

$$\phi(\mathbf{x}) = [x_1^2, \quad x_2^2, \quad \sqrt{2} x_1 x_2].$$

Proof: $K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})\phi(\mathbf{x}')^T$

$$= [x_1^2, \quad x_2^2, \quad \sqrt{2} x_1 x_2] \begin{bmatrix} x_1'^2 \\ x_2'^2 \\ \sqrt{2} x_1' x_2' \end{bmatrix}$$

$$= x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2x_1 x_2 x_1' x_2'$$

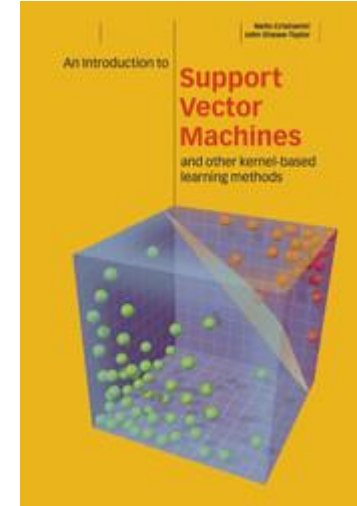
$$= (x_1 x_1' + x_2 x_2')^2$$

$$= \left([x_1 \quad x_2] \begin{bmatrix} x_1' \\ x_2' \end{bmatrix} \right)^2 = \boxed{(\mathbf{x} \cdot \mathbf{x}')^2}$$

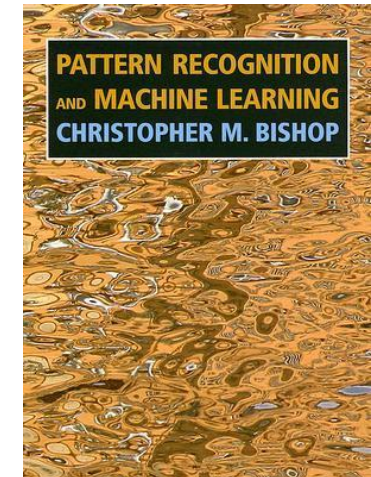
Quadratic kernel

Outline

- Support Vector Machines
 - Large-Margin Classifiers
 - Quadratic Optimization Problem
 - Nonlinear SVM using Kernels
 - Mercer's Theorem
 - Multiple Kernel Learning
- Multi-class Classification
- Kernel Methods for Regression
 - Support Vector Regression
 - Kernel Ridge Regression



Cristianini & Shawe-Taylor (2000)
An Introduction to Support Vector Machines and other kernel-based learning methods.
Cambridge University Press.



Bishop (2006)
Pattern Recognition and Machine Learning. Springer.

Other Kernel Machines you may be interested in:

- Least-squares SVM (LSSVM)
- Relevance Vector Machine (RVM)
- Core Vector Machines (CVM)
- Twin Support Vector Machines
- One-class SVM

Further Reading

- Cristianini & Shawe-Taylor (2000). *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press.
- Bishop (2006). *Pattern Recognition and Machine Learning*. Springer.
- Cortes and Vapnik (1995). Support-Vector Networks. *Machine Learning*, vol. 20, pp. 273-297.
- Andrew Ng CS 229 ML: <https://www.youtube.com/playlist?list=PLoROMvody4rMiGQp3WXShtMGgzqpfVfbU>
- Multi-class, Multi-label, Multi-output: <https://scikit-learn.org/stable/modules/multiclass.html>
- ECOC paper: <https://arxiv.org/pdf/cs/9501101.pdf>
- SMOTE paper: <https://arxiv.org/pdf/1106.1813.pdf>
- <https://www.kdnuggets.com/2020/01/5-most-useful-techniques-handle-imbalanced-datasets.html>
- KRR vs. SVR: https://scikit-learn.org/stable/auto_examples/miscellaneous/plot_kernel_ridge_regression.html
- K.-R. Muller, S. Mika, G. Ratsch, K. Tsuda, B. Scholkopf (2001). "An introduction to kernel-based learning algorithms", IEEE Transactions on Neural Networks, 12(2), 181-201. <https://ieeexplore.ieee.org/document/914517>
- Kernel Cookbook: <https://www.cs.toronto.edu/~duvenaud/cookbook/>
- Gonen and Alpaydin (2011). Multiple Kernel Learning Algorithms. Journal of Machine Learning Research, 12. <https://jmlr.org/papers/volume12/gonen11a/gonen11a.pdf>
- RKHS: <https://ngilshie.github.io/jekyll/update/2018/02/01/RKHS.html>
- Hofmann (2008). Kernel Methods in Machine Learning: <https://arxiv.org/pdf/math/0701907.pdf> ← More theory on Kernel Machines
- Tipping (1999). Relevance Vector Machines. <https://proceedings.neurips.cc/paper/1999/file/f3144cefe89a60d6a1afaf7859c5076b-Paper.pdf>