Lessons: when where who

FRANCESCO TOMBA

MARTEDI' 16-18 Aula Morin [Edificio H3bis]

MERCOLEDI' 10-12 Aula I [Edificio C1]

GIOVEDI'' 15-17 Aula Morin [Edificio H3bis] (lab)

Lessons: what

Weekly: 4h theory + 2 h practice

Challenges (2): 4h

Small groups. Each group has a dataset and apply ML methods to solve a task.

Exam, sum of:

- Short reports after each challenge
- Propose a *final project* (to be evaluated):
  *Dataset
  *Algorithm
  *Theory
- Exam: reports + project presentation + questions on course material

Exam, Evaluation

Progetto: max 12

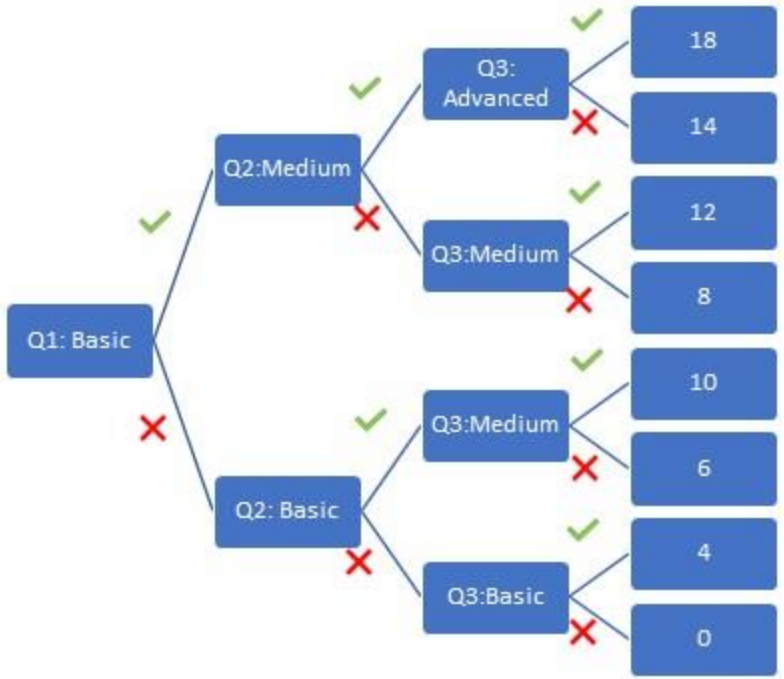
3 Domande: max. 18

Prima domanda: max. 8

Seconda domanda: max. 6

Terza domanda: max. 4

La prima domanda sarà scelta fra quelle considerate di base. Se lo studente risponde correttamente, la difficoltà della domanda successiva sarà aumentata, altrimenti la difficoltà rimarrà la stessa. Questo schema porta al alboro decisionale mostrato sotto.



Challenge: + 3 punti se fatte bene

Presentations should be a max of 10 mins!!!!

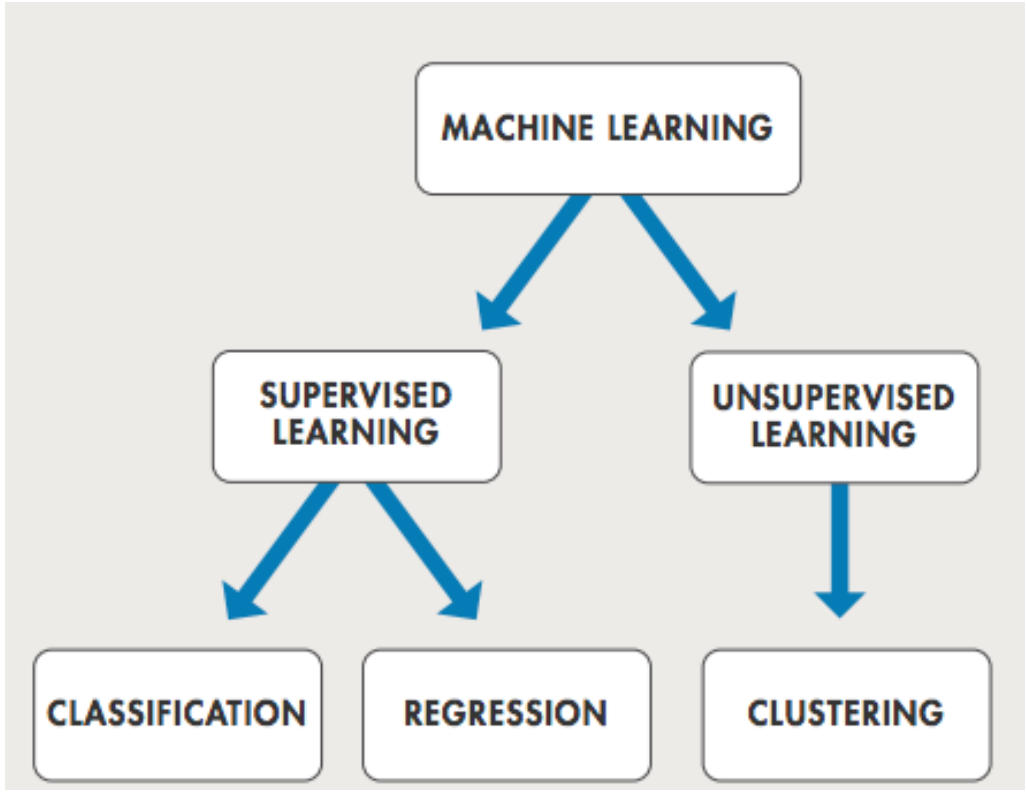
Course content

Main idea: get to know most common techniques (and theory) used in ML together with (some) coding skills.

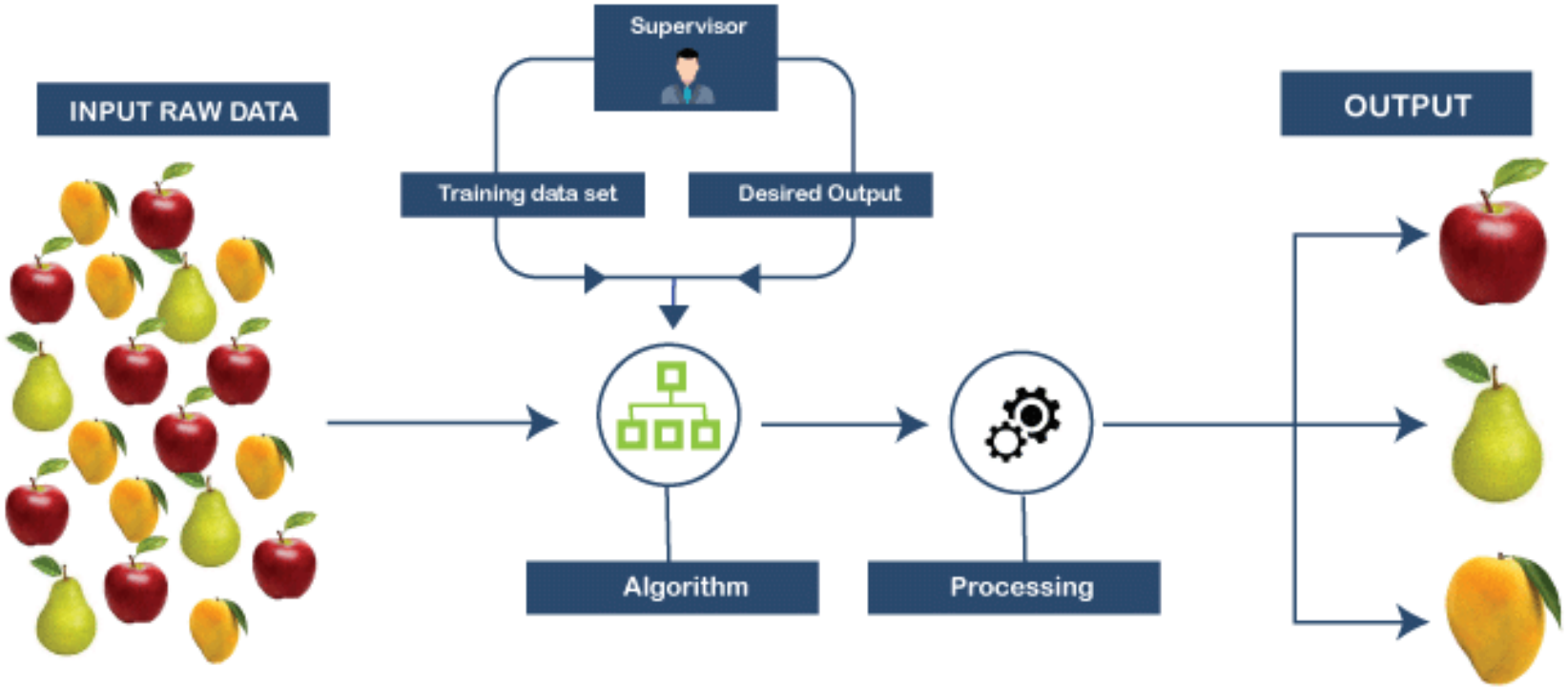
Intro

- Basic notions (dataset, task…) + Regression and classification models (A+F)
- Model assessment (A)
- Unsupervised learning: dimensionality reduction (A)
- Supervised learning: K-Nearest Neighbor, Trees (A)
- Kernels (F)
- Artificial Neural Networks (F)
- (Probabilistic formulation, recommender systems, RL)

Supervised and unsupervised methods

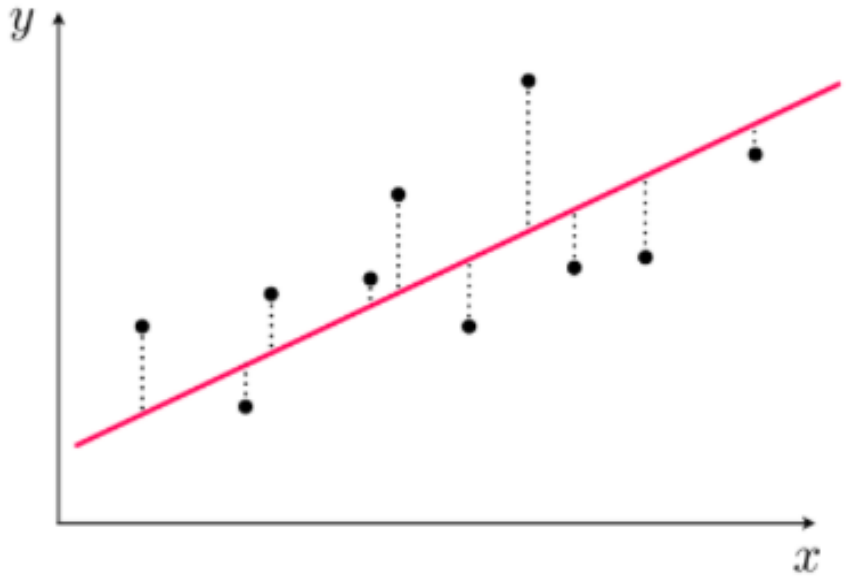


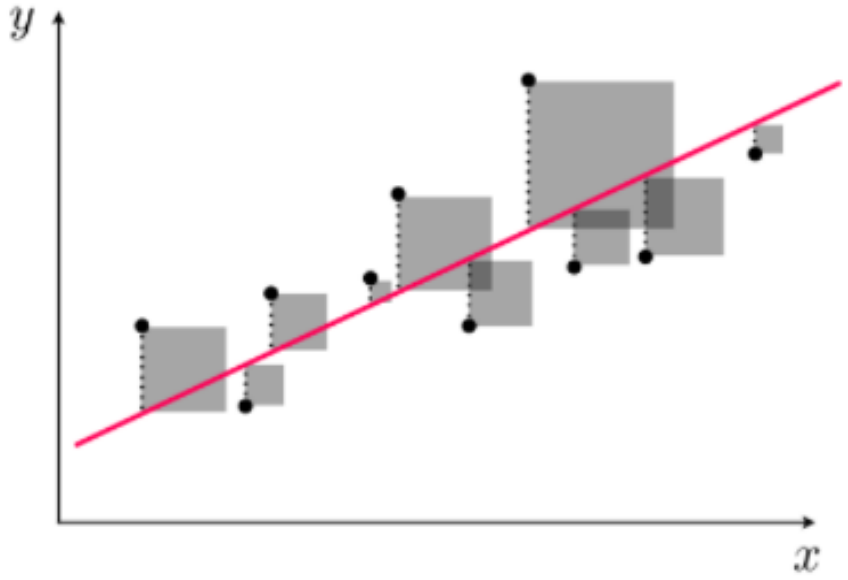
Supervised learning: classification



What's the role of the supervisor?

Measuring errors: square cost



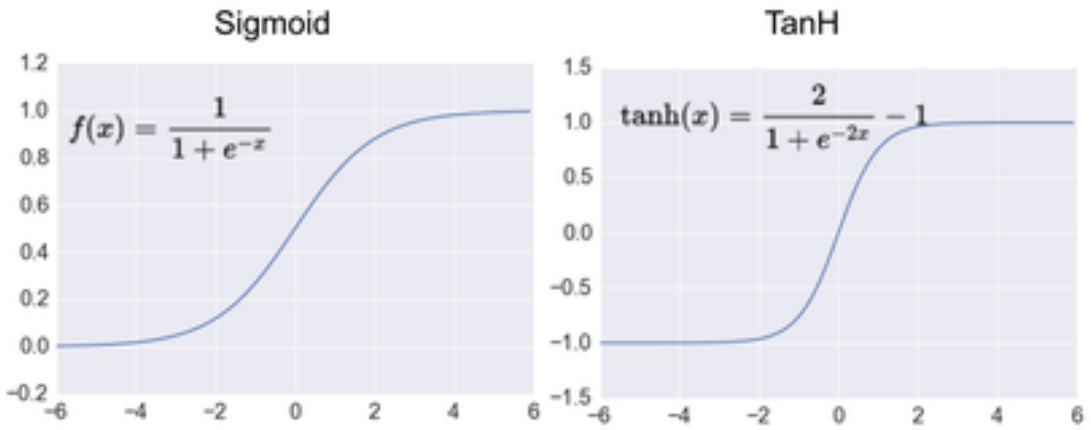


$$\mathcal{E}(w, x, y) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \sum_j w_j x_j^i)^2$$

How do we find a solution?

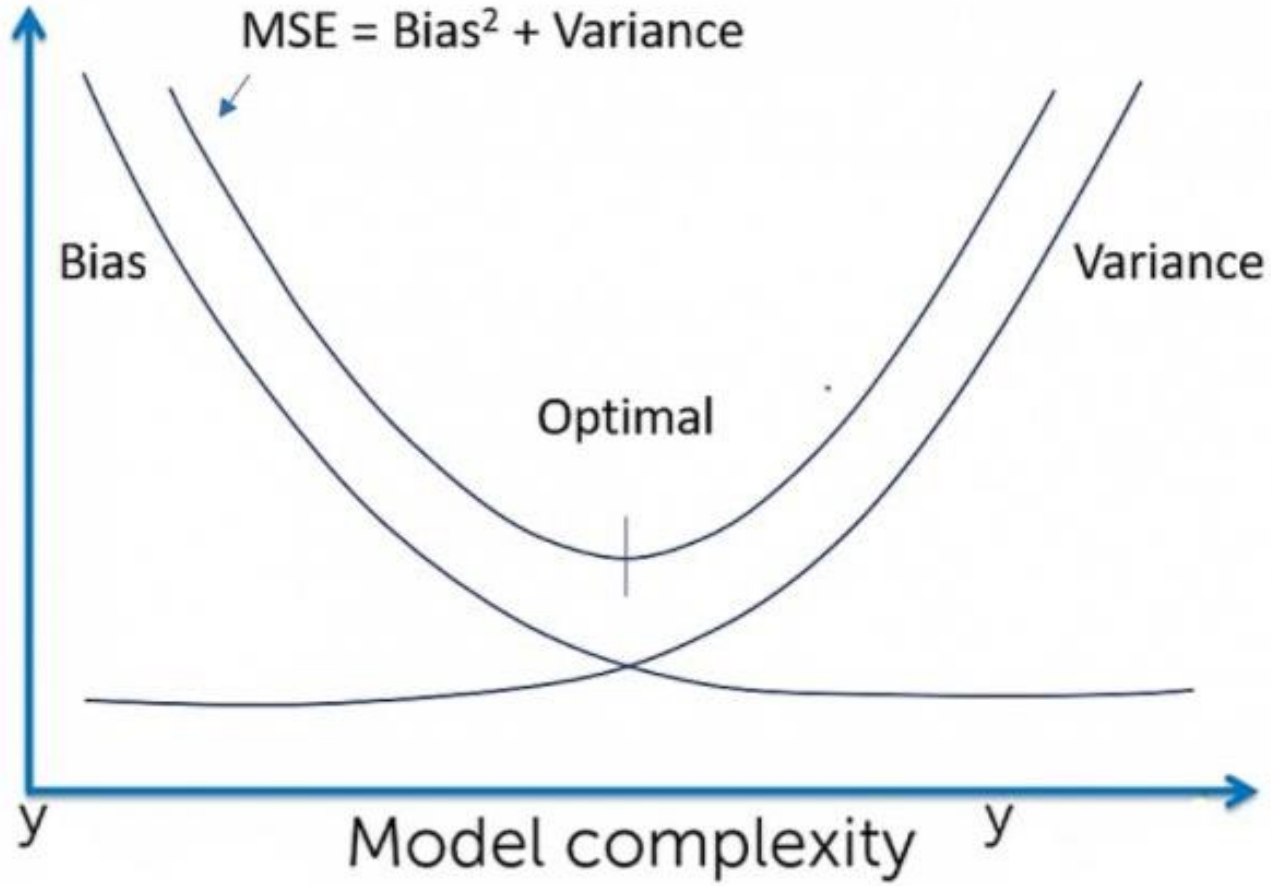
How do I write the RHS in a compact form?

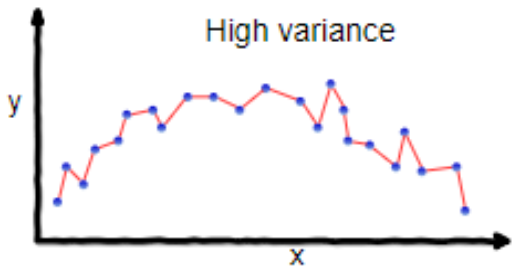
Logistic regression



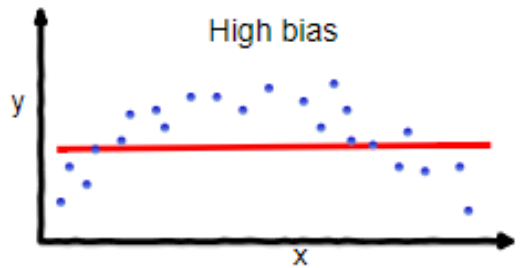
Logistic Function

$$p(y|x) = \frac{1}{1 + e^{w^T x}}$$

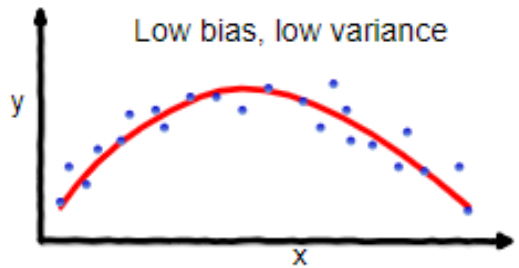




overfitting



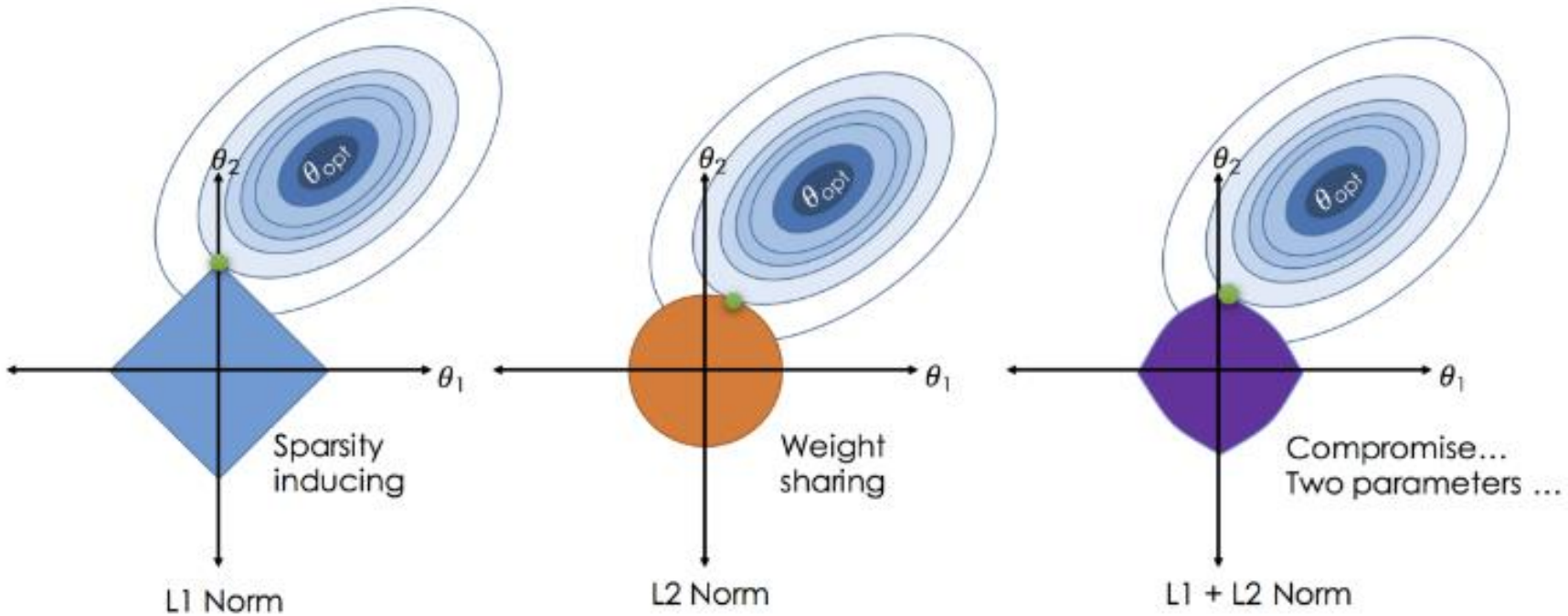
underfitting



Good balance

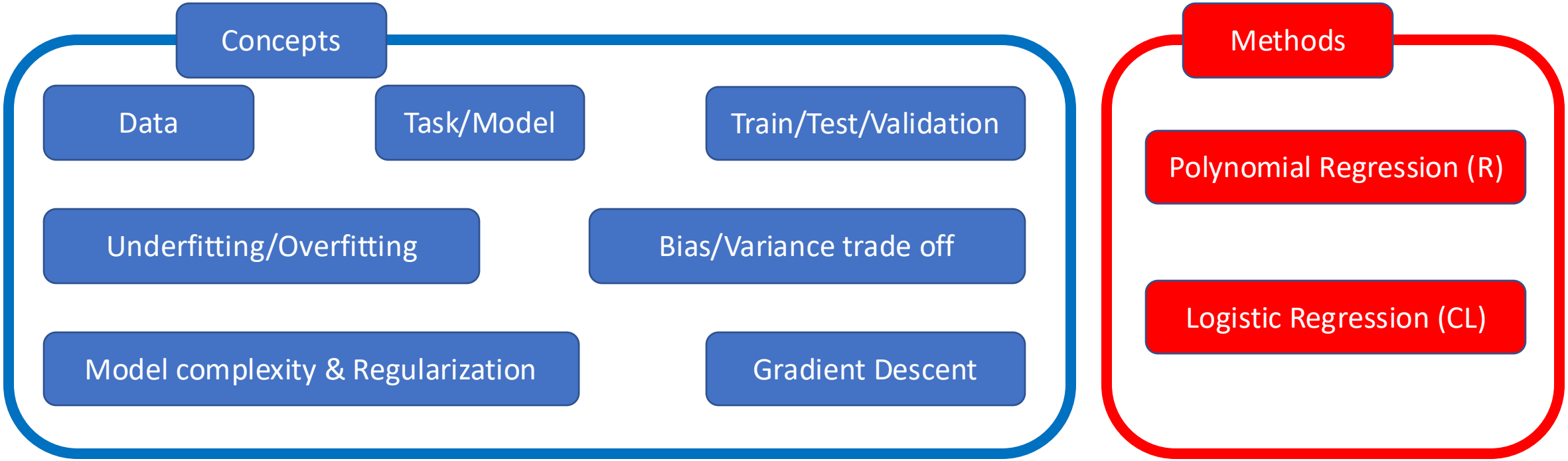
Regularization

$$\arg\min_{w} \|y - X^T w\|_2^2 + \lambda(\alpha\|w\|_1 + (1-\alpha)\|w\|_2)$$



Which is the advantage?

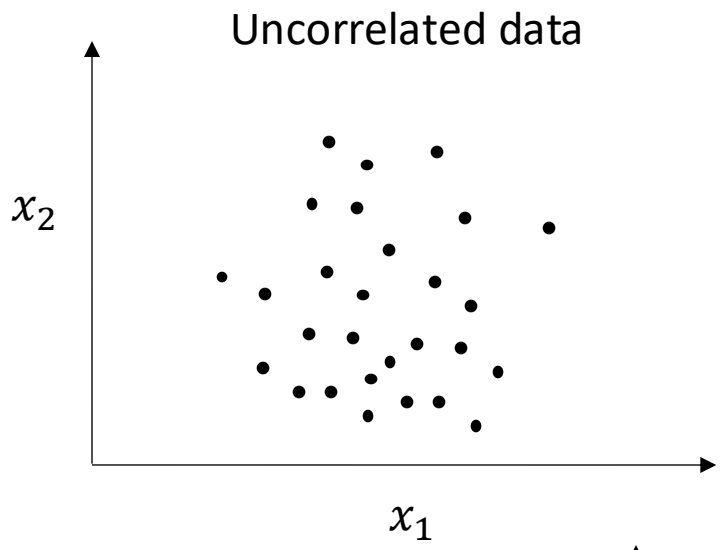
Supervised Learning

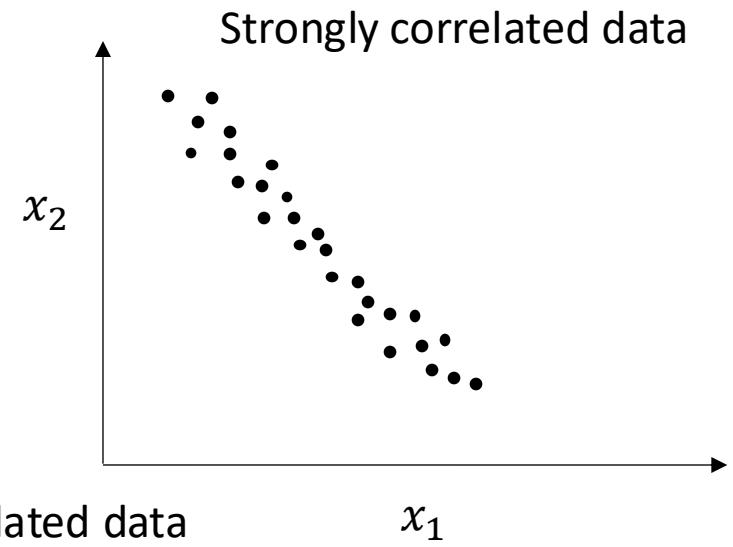


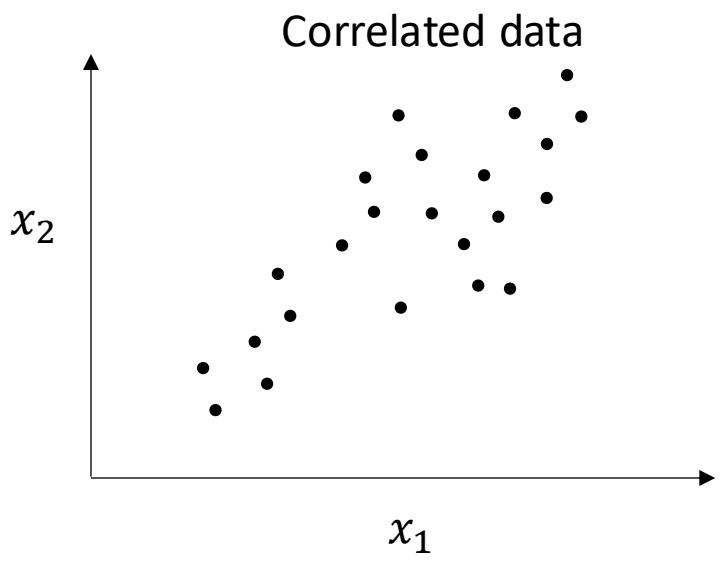
Algorithms for Unsupervised Learning (Alex)

- Manifold Learning (Dim. Red. & ID est.):
  - PCA
  - K-PCA
  - ISOMAP
  - t-SNE
  - Autoencoders
- Density Estimation:
  - Histograms
  - Kernel Density Estimation
  - k-Nearest Neighbor
  - Generative Adversarial NN
- Clustering:
  - k-means/c-means, kernel k-means, spectral clustering…
  - Hierarchical clustering
  - Density Based clustering
  - Self Organizig Maps

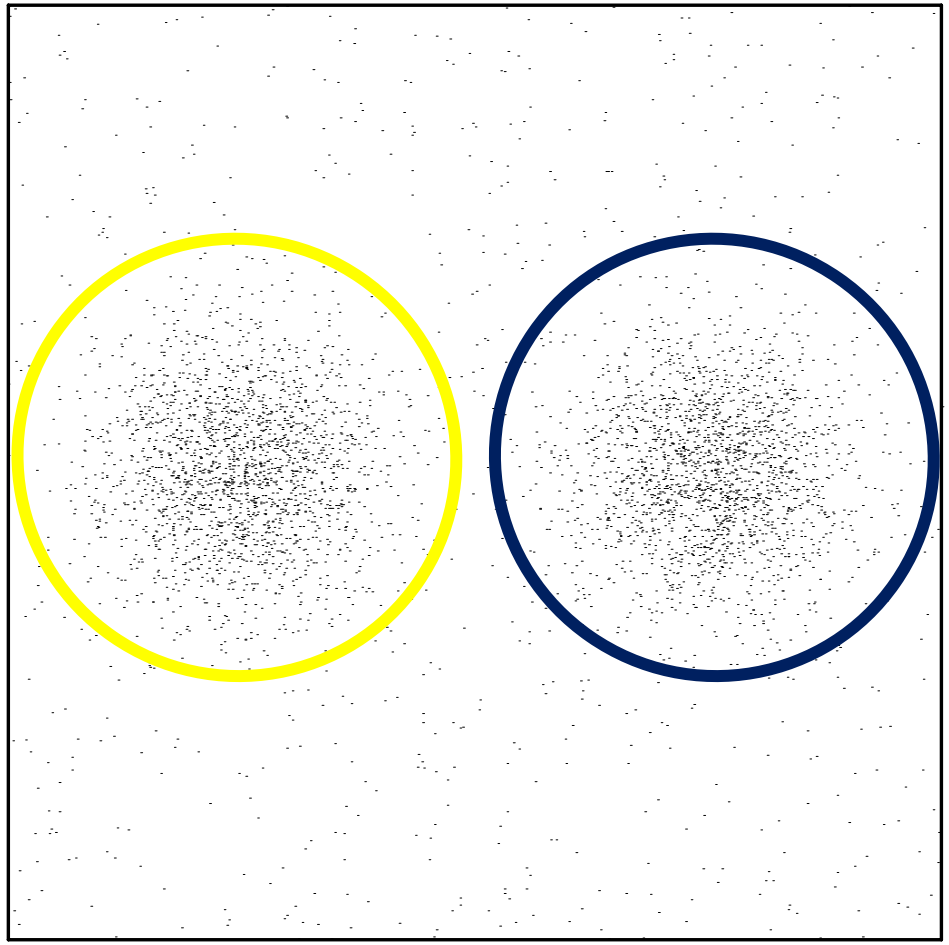
STRONG CORRELATIONS REDUCE DIMENSIONALITY



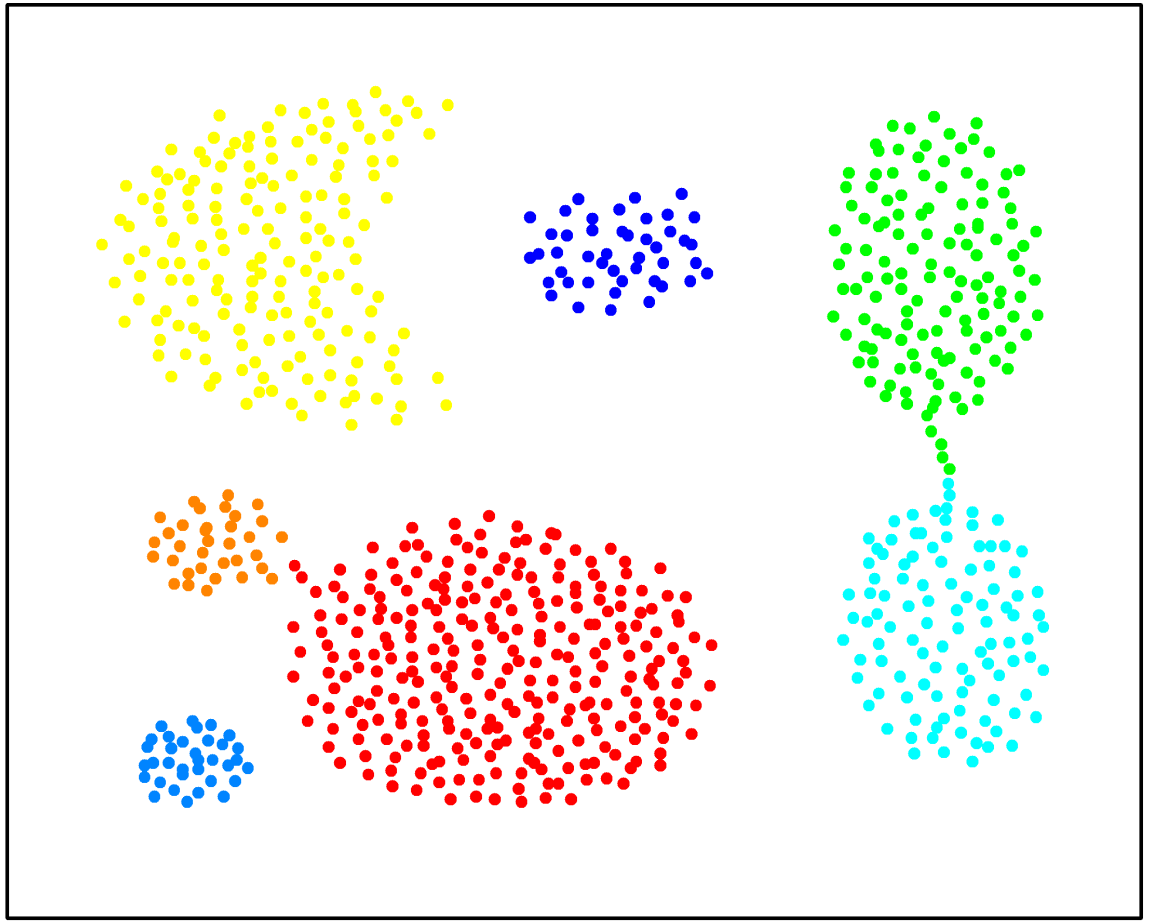




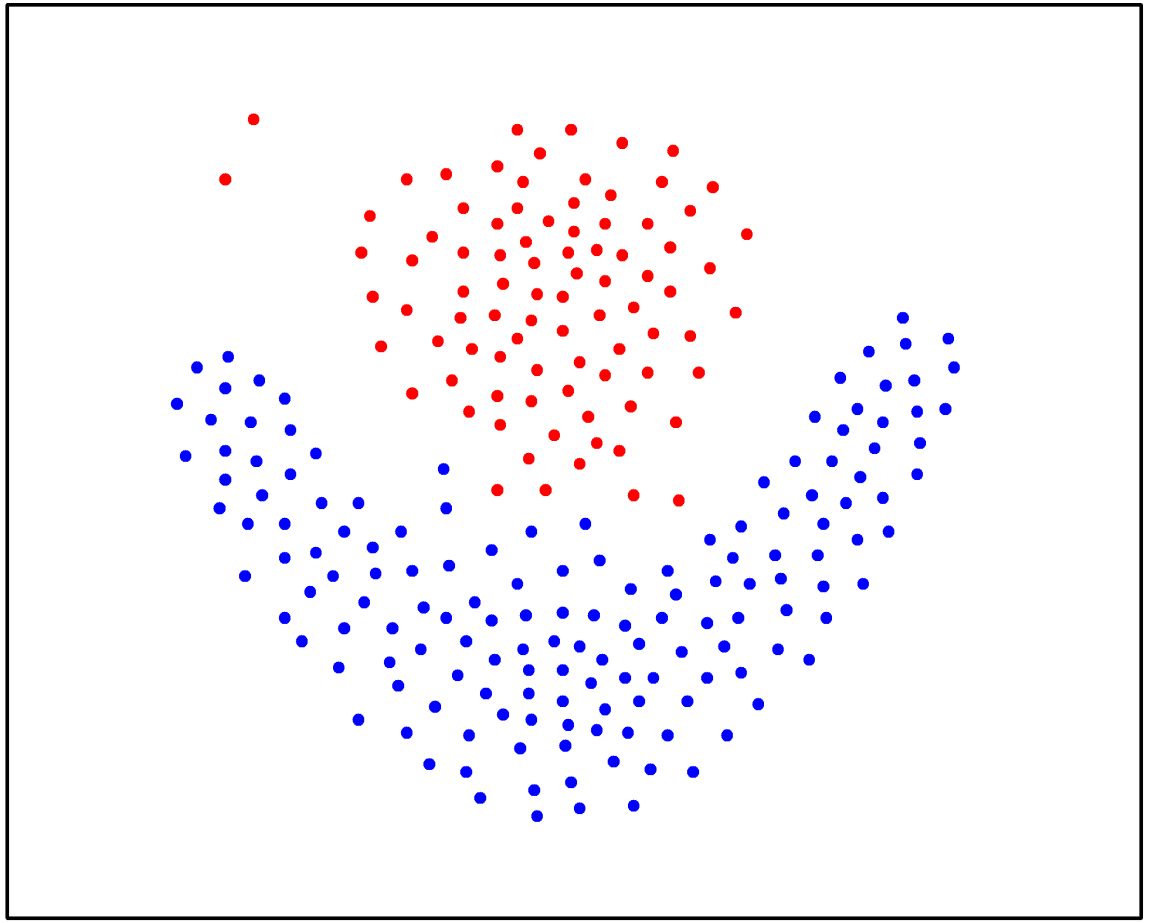
What is a cluster???



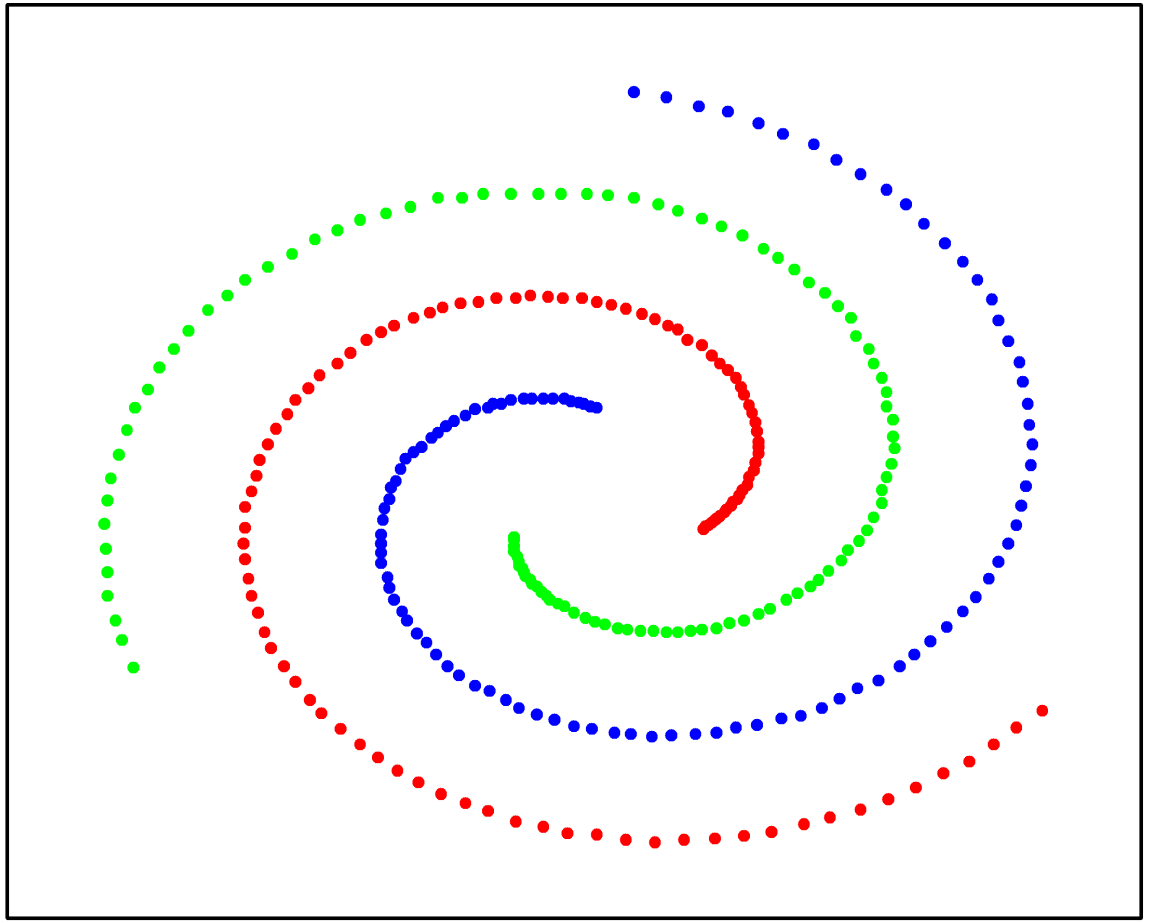
Other cluster examples…



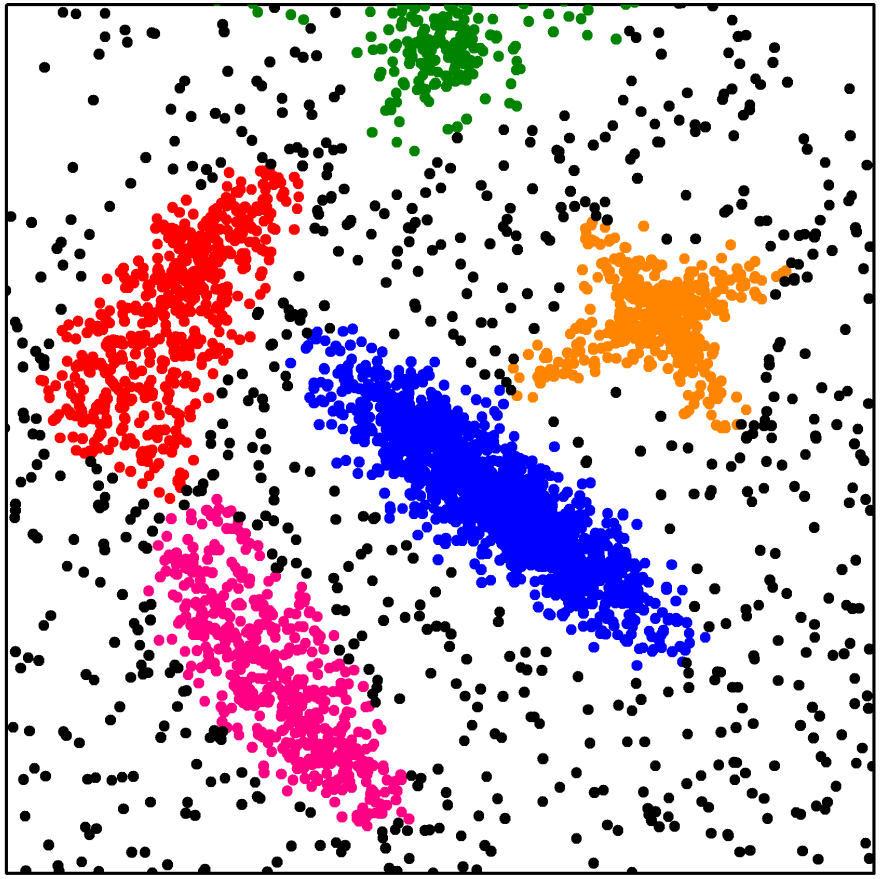
Other cluster examples…



Other cluster examples…



Other cluster examples…



Perceptron, support vector machines and kernels

Plan of part 1 of the course

PERCEPTRON and SVM: linearly separable problems

- Linearly separable problems: separating hyperplane learning with perceptron
- Support Vector machines and maximal margin classifier
- Soft margin classifier and kernels.

KERNELS: non-lineary separable problems

- Non linearly separable problems; feature augmentation and intuition.
- Kernel regression, kernel trick
- Kernel SVM
- Main options (linear, polynomial, radial); intuition behind radial kernel
- Application: Kernel regression, Kernel Nearest Neighbor, Kernel PCA.

Main goal of the course


.fun

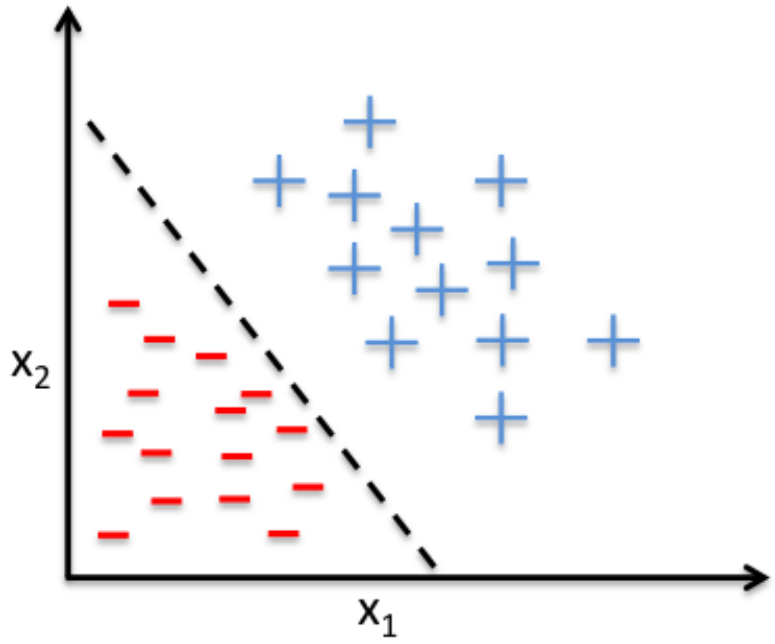
Class1: Perceptron

Linear separability

Separating hyperplane

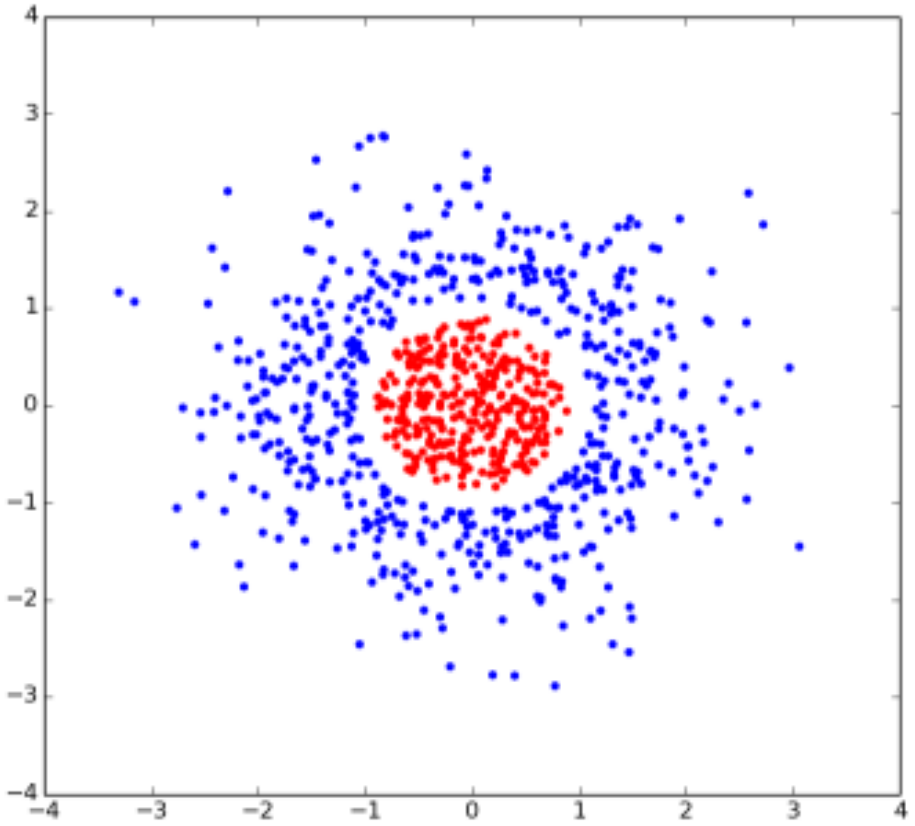
Loss and convergence of the algorithm

The problem: Linear separability



Assumptions

1. Binary classification (i.e. $y_i \in \{-1, +1\}$)
2. Data is linearly separable



Is this a linearly separable problem?

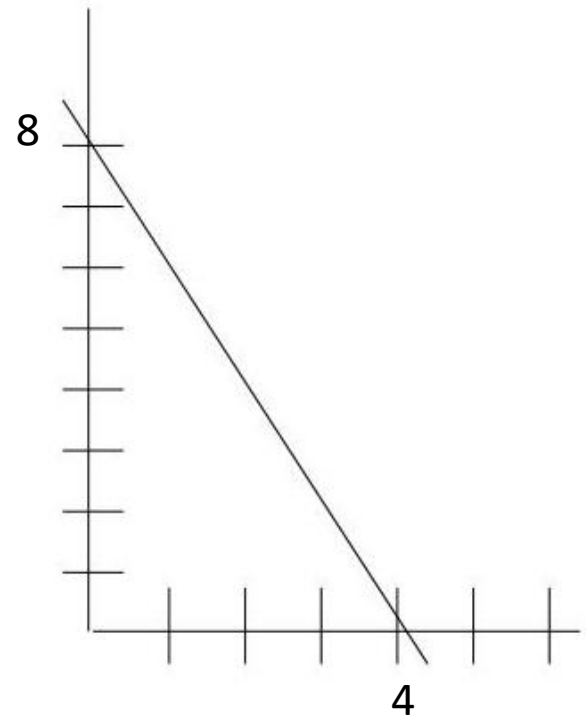
Hyperplane separators

- Consider equation of line shown on right

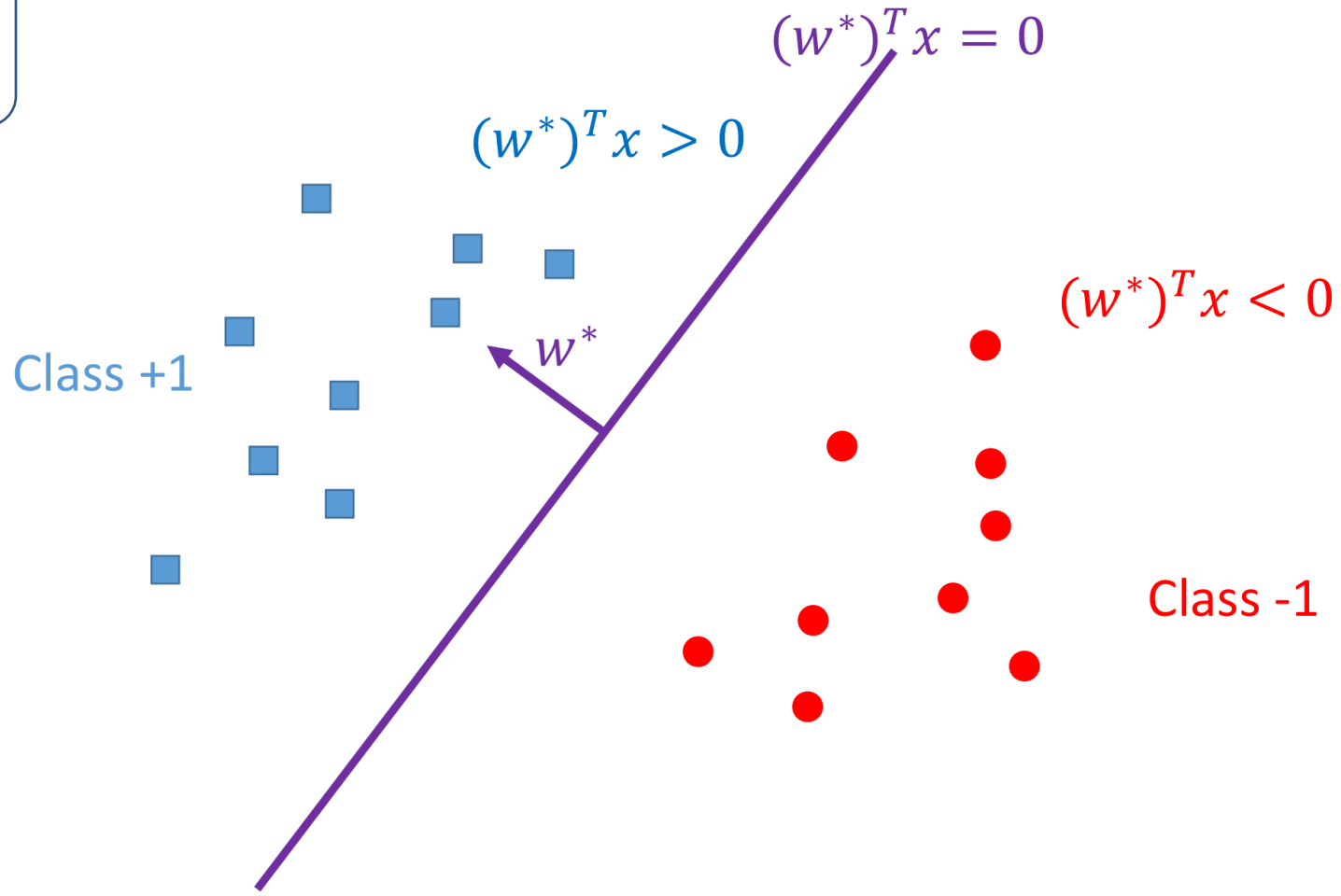
$$x_1 = -2x_0 + 8$$

- If we let $w = (2,1)$ and $w_0 = -8$ then we can rewrite it as

$$w^T x + w_0 = 0$$

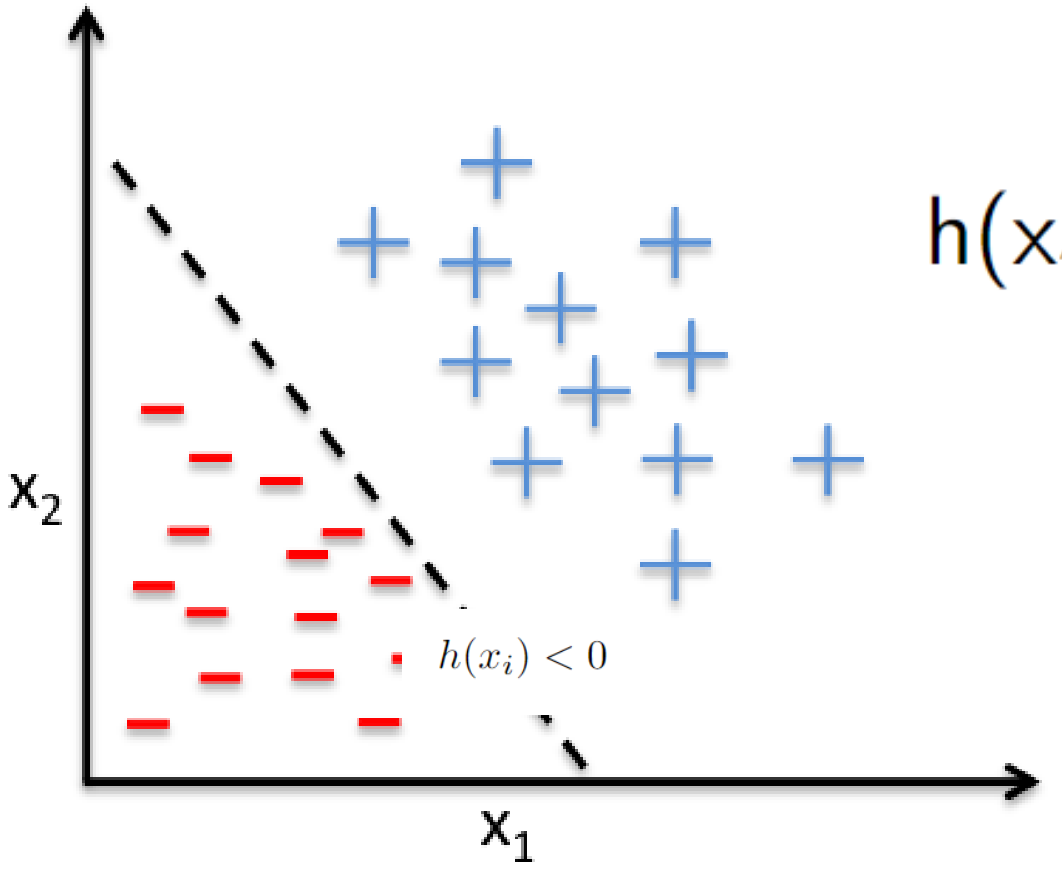


Binary Task



This is an hyperplane passing through…?

Classifying points



$$\mathrm{h}(\mathrm{x}_i) = sign(w^T x_i + b)$$

Absorbing the bias

b is the bias term (without the bias term, the hyperplane that $\mathbf{w}$ defines would always have to go through the origin). Dealing with b can be a pain, so we 'absorb' it into the feature vector $\mathbf{w}$ by adding one additional *constant* dimension. Under this convention,

$$\mathbf{x}_i \text{ becomes } \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix}$$

$$\mathbf{w} \text{ becomes } \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$$

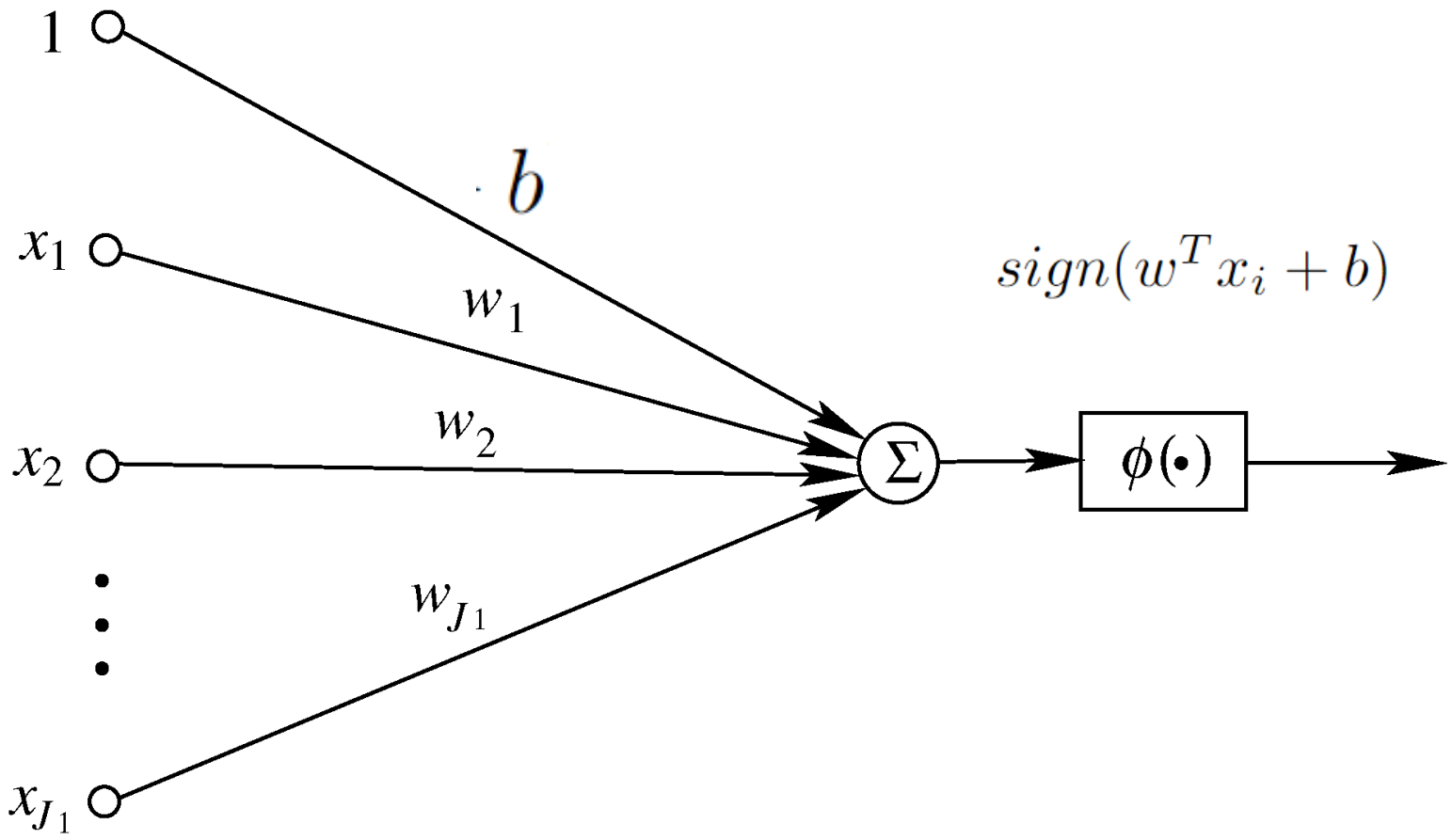
We can verify that

$$\begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix}^\top \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} = \mathbf{w}^\top \mathbf{x}_i + b$$

Using this, we can simplify the above formulation of $h(\mathbf{x}_i)$ to

$$h(\mathbf{x}_i) = \text{sign}(\mathbf{w}^\top \mathbf{x})$$

Simplest Neuron model



Summarizing:

- Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
- Hypothesis $f_w(x) = w^T x$
  - $y = +1$ if $w^T x > 0$
  - $y = -1$ if $w^T x < 0$
- Prediction: $y = \text{sign}(f_w(x)) = \text{sign}(w^T x)$

- Goal: minimize classification error

How do we minimize the error?

Perceptron Algorithm

- Assume for simplicity: all x_i has length 1

1. Start with the all-zeroes weight vector $\mathbf{w}_1 = \mathbf{0}$, and initialize t to 1.
2. Given example $\mathbf{x}$, predict positive iff $\mathbf{w}_t \cdot \mathbf{x} > 0$.
3. On a mistake, update as follows:
   - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}$.
   - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}$.

   $t \leftarrow t + 1$.

Why is this a reasonable strategy?

Intuition: correct the current mistake

- If mistake on a positive example

$$w_{t+1}^T x = (w_t + x)^T x = w_t^T x + x^T x = w_t^T x + 1$$

- If mistake on a negative example

$$w_{t+1}^T x = (w_t - x)^T x = w_t^T x - x^T x = w_t^T x - 1$$

1. Start with the all-zeroes weight vector $\mathbf{w}_1 = \mathbf{0}$, and initialize t to 1.
2. Given example $\mathbf{x}$, predict positive iff $\mathbf{w}_t \cdot \mathbf{x} > 0$.
3. On a mistake, update as follows:
   - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}$.
   - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}$.

   $t \leftarrow t + 1$.

Does the algorithm converge? If so how fast?

The Perceptron Theorem

- Suppose there exists w^* that correctly classifies $\{(x_i, y_i)\}$
- W.L.O.G., all x_i and w^* have length 1, so the minimum distance of any example to the decision boundary is

$$\gamma = \min_i |(w^*)^T x_i|$$

Need not be i.i.d. !

- Then Perceptron makes at most $\left(\frac{1}{\gamma}\right)^2$ mistakes

Do not depend on n, the length of the data sequence!

Analysis

- First look at the quantity $w_t^T w^*$

- Claim 1: $w_{t+1}^T w^* \geq w_t^T w^* + \gamma$
- Proof: If mistake on a positive example x

$$w_{t+1}^T w^* = (w_t + x)^T w^* = w_t^T w^* + x^T w^* \geq w_t^T w^* + \gamma$$

- If mistake on a negative example

$$w_{t+1}^T w^* = (w_t - x)^T w^* = w_t^T w^* - x^T w^* \geq w_t^T w^* + \gamma$$

Analysis

- Next look at the quantity $||w_t||$

Negative since we made a mistake on x

- Claim 2: $||w_{t+1}||^2 \leq ||w_t||^2 + 1$
- Proof: If mistake on a positive example x

$$||w_{t+1}||^2 = ||w_t + x||^2 = ||w_t||^2 + ||x||^2 + 2w_t^T x$$

Analysis: putting things together

- Claim 1: $w_{t+1}^T w^* \geq w_t^T w^* + \gamma$
- Claim 2: $\left||w_{t+1}|\right|^2 \leq \left||w_t|\right|^2 + 1$

After M mistakes:

- $w_{M+1}^T w^* \geq \gamma M$
- $\left||w_{M+1}|\right| \leq \sqrt{M}$
- $w_{M+1}^T w^* \leq \left||w_{M+1}|\right|$ Apply chauchy swartz

So $\gamma M \leq \sqrt{M}$, and thus $M \leq \left(\frac{1}{\gamma}\right)^2$

Intuition

- Claim 1: $w_{t+1}^T w^* \geq w_t^T w^* + \gamma$
- Claim 2: $\left||w_{t+1}|\right|^2 \leq \left||w_t|\right|^2 + 1$

The correlation gets larger. Could be:

1. w_{t+1} gets closer to w^*
2. w_{t+1} gets much longer

Rules out the bad case "2. w_{t+1} gets much longer"

History

Frank Rosenblatt
1928–1969

Rosenblatt's perceptron played an important role in the history of machine learning. Initially, Rosenblatt simulated the perceptron on an IBM 704 computer at Cornell in 1957, but by the early 1960s he had built special-purpose hardware that provided a direct, parallel implementation of perceptron learning. Many of his ideas were encapsulated in "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" published in 1962. Rosenblatt's work was criticized by Marvin Minsky, whose objections were published in the book "Perceptrons", co-authored with Seymour Papert. This book was widely misinterpreted at the time as showing that neural networks were fatally flawed and could only learn solutions for linearly separable problems. In fact, it only proved such limitations in the case of single-layer networks such as the perceptron and merely conjectured (incorrectly) that they applied to more general network models. Unfortunately, however, this book contributed to the substantial decline in research funding for neural computing, a situation that was not reversed until the mid-1980s. Today, there are many hundreds, if not thousands, of applications of neural networks in widespread use, with examples in areas such as handwriting recognition and information retrieval being used routinely by millions of people.

Figure from *Pattern Recognition and Machine Learning*, Bishop

Class 1 (second part): Support vector machines

- Gradient point of view
- Maximal margin classifier
- Maximal-margin optimization
- Support vectors

Gradient descent view of the perceptron algorithm

Matrix Calculus

Let $y, x \in \mathbb{R}$ be scalars, $\mathbf{y} \in \mathbb{R}^M$ and $\mathbf{x} \in \mathbb{R}^P$ be vectors, and $\mathbf{Y} \in \mathbb{R}^{M \times N}$ and $\mathbf{X} \in \mathbb{R}^{P \times Q}$ be matrices

Types of Derivatives (Denominator ↓ / Numerator →)	**scalar**	**vector**	**matrix**
scalar	$\frac{\partial y}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{Y}}{\partial x}$
vector	$\frac{\partial y}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$
matrix	$\frac{\partial y}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$

Types of Derivatives	**scalar**
scalar	$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x}\right]$
vector	$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$
matrix	$\frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial X_{11}} & \frac{\partial y}{\partial X_{12}} & \cdots & \frac{\partial y}{\partial X_{1Q}} \\ \frac{\partial y}{\partial X_{21}} & \frac{\partial y}{\partial X_{22}} & \cdots & \frac{\partial y}{\partial X_{2Q}} \\ \vdots & & & \vdots \\ \frac{\partial y}{\partial X_{P1}} & \frac{\partial y}{\partial X_{P2}} & \cdots & \frac{\partial y}{\partial X_{PQ}} \end{bmatrix}$

Types of Derivatives	**scalar**	**vector**
scalar	$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x}\right]$	$\frac{\partial \mathbf{y}}{\partial x} = \left[\frac{\partial y_1}{\partial x} \quad \frac{\partial y_2}{\partial x} \quad \cdots \quad \frac{\partial y_N}{\partial x}\right]$
vector	$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_N}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_N}{\partial x_2} \\ \vdots & & & \\ \frac{\partial y_1}{\partial x_P} & \frac{\partial y_2}{\partial x_P} & \cdots & \frac{\partial y_N}{\partial x_P} \end{bmatrix}$

Vector Gradient

$$\frac{\partial}{\partial w_j}(w^T x) = \frac{\partial}{\partial w_j}\sum_k w_k x_k = x_j$$

$$\nabla_w(w^T x) = x$$

Other examples

The Matrix Cookbook

[http://matrixcookbook.com]

Kaare Brandt Petersen
Michael Syskind Pedersen

$$\frac{\partial \mathbf{x}^T\mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T\mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

$$\frac{\partial \mathbf{a}^T\mathbf{X}\mathbf{b}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{b}^T$$

$$\frac{\partial \mathbf{a}^T\mathbf{X}^T\mathbf{b}}{\partial \mathbf{X}} = \mathbf{b}\mathbf{a}^T$$

$$\frac{\partial \mathbf{a}^T\mathbf{X}\mathbf{a}}{\partial \mathbf{X}} = \frac{\partial \mathbf{a}^T\mathbf{X}^T\mathbf{a}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{a}^T$$

Gradient descent view: constructing a loss

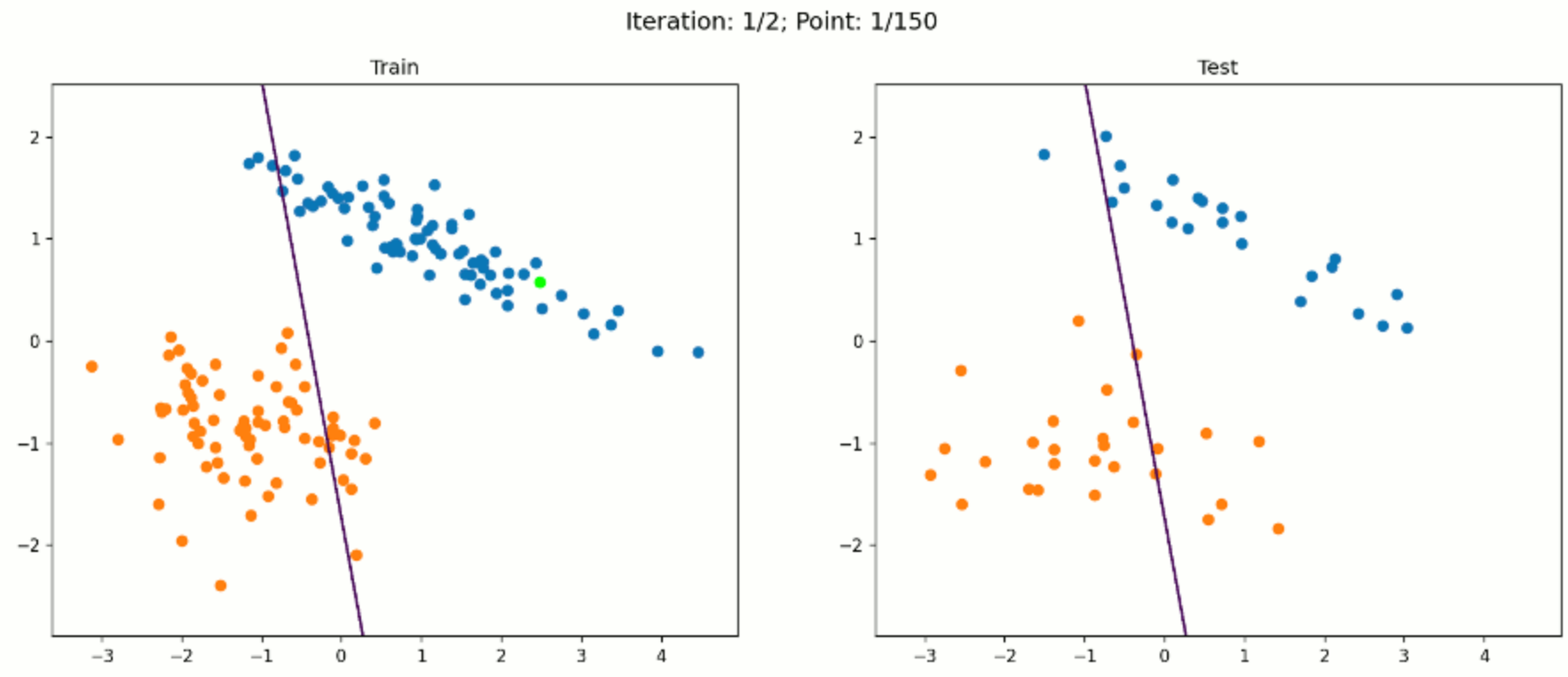
Perceptron loss:

$$L_{\boldsymbol{w}}(y^{(i)}) = \max(0, -y^{(i)}\boldsymbol{w}^\top \boldsymbol{x}^{(i)}).$$

Why is a good Loss?

$$\partial L_{\boldsymbol{w}}(y^{(i)}) = \begin{cases} \{0\}, & \text{if } y^{(i)}\boldsymbol{w}^\top \boldsymbol{x}^{(i)} > 0 \\ \{-y^{(i)}\boldsymbol{x}^{(i)}\}, & \text{if } y^{(i)}\boldsymbol{w}^\top \boldsymbol{x}^{(i)} < 0 \\ [-1, 0] \times y^{(i)}\boldsymbol{x}^{(i)}, & \text{if } \boldsymbol{w}^\top \boldsymbol{x}^{(i)} = 0 \end{cases}.$$

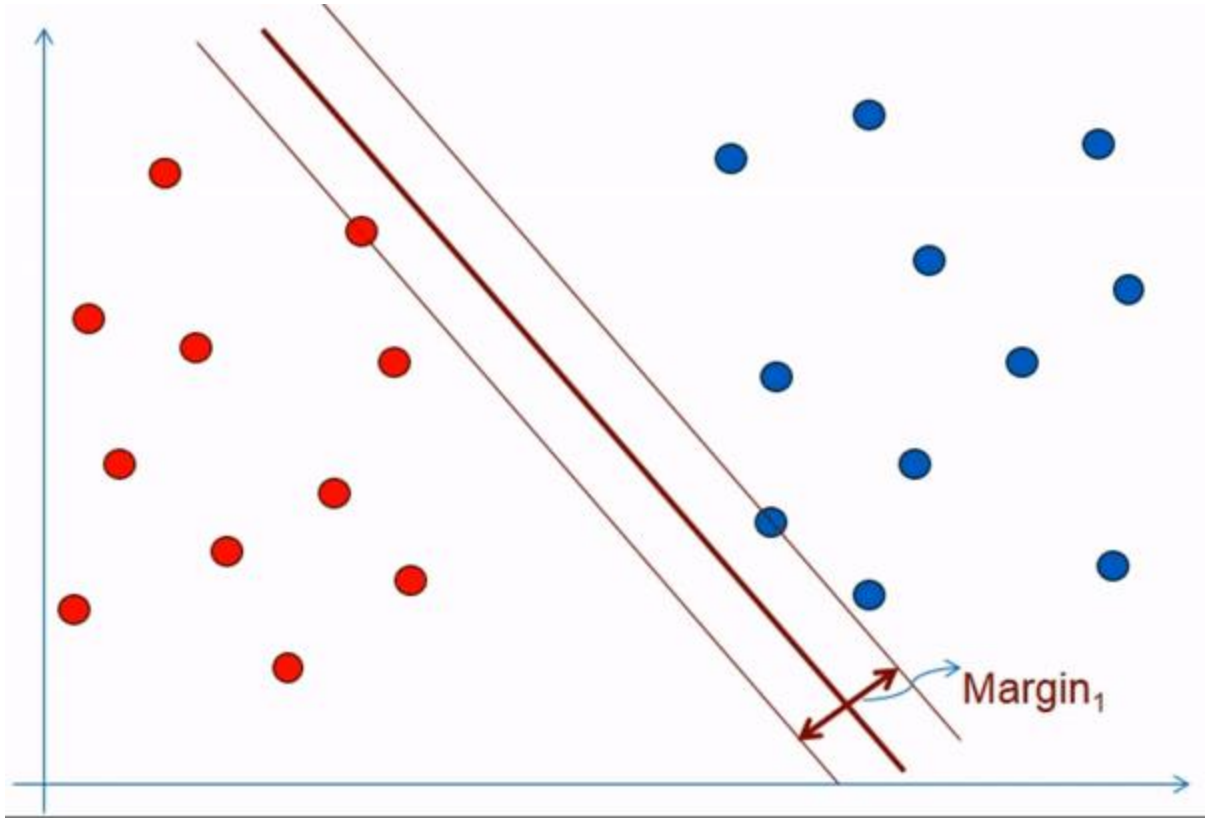
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \sum_{i=1}^{n} \partial L_{\boldsymbol{w}^{(t)}}(y^{(i)}).$$



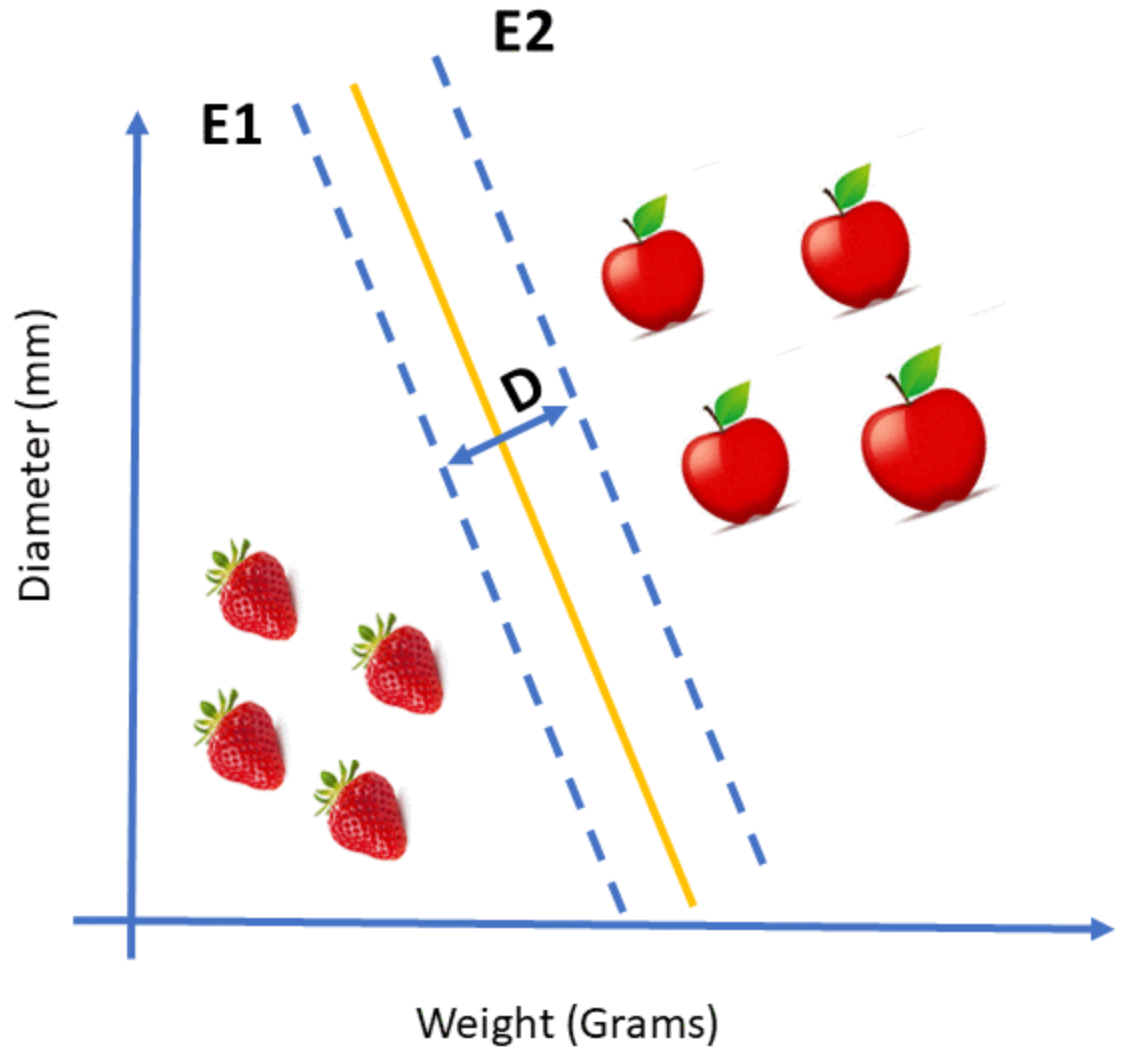
What is the problem?

What happened if we add noise to the points?

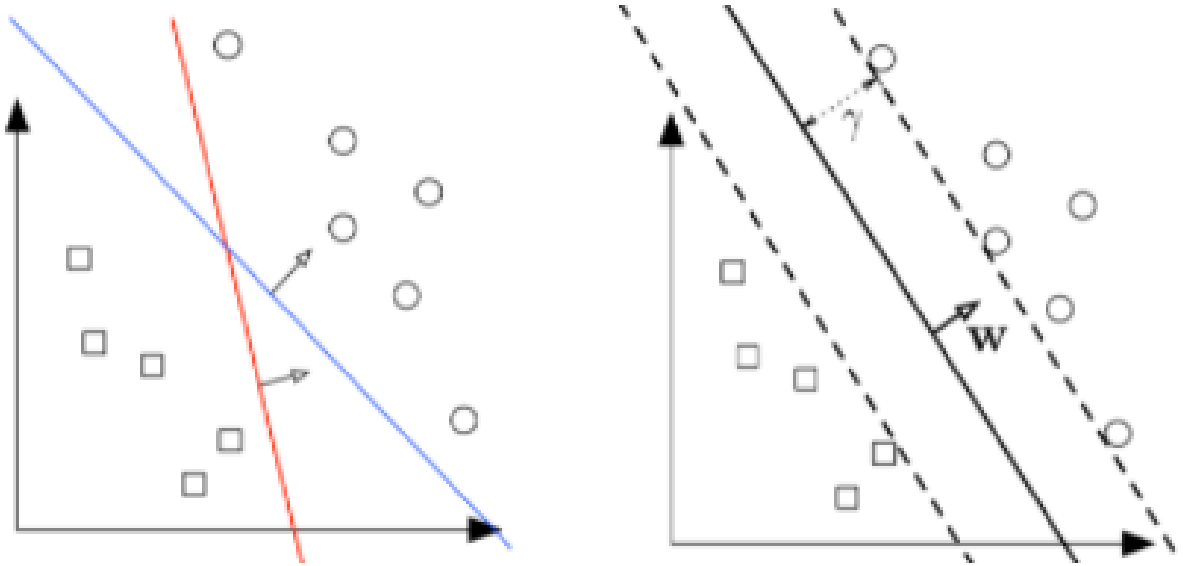
Maximum margin



Maximum margin



Maximum margin



What is the best separating hyperplane?
What about robustness?

MAXIMUM MARGINE HYPERPLANE

The hyperplane that maximizes the distances to the closest data points from both classes.

Let's define more precisely the concept of margin

Distance point-hyperplane

Consider a point x and call d its distance from an hyperplane H. Let x^P the projection of x onto H. We have

- $x^P = x - d$ and $d = \alpha w$
- $x^P \in H$ so

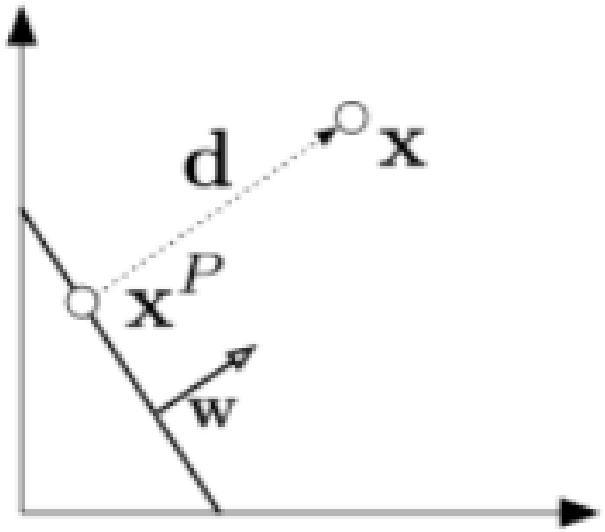
$$w^T x^P + b = 0 \rightarrow w^T(x-d)+b = 0 \rightarrow w^T(x-\alpha w)+b = 0 \rightarrow \alpha = \frac{w^T x + b}{w^T w}$$

- Therefore the length of d is:

$$\|d\|_2 = \alpha\sqrt{w^T w} = \frac{|w^T x + b|}{\sqrt{w^T w}} = \frac{|w^T x + b|}{\|w\|_2}$$

- Thus we are looking for:

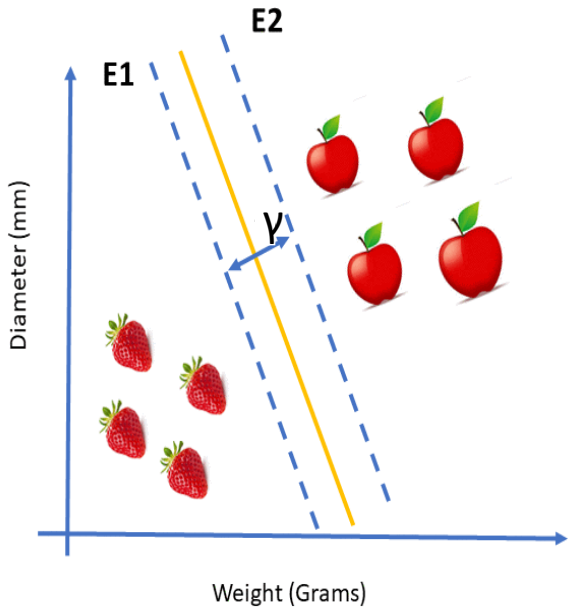
$$\gamma(w,b) = \min_{x \in Data} \frac{|w^T x + b|}{\|w\|_2}$$



Maximal margin classifier optimization (1)

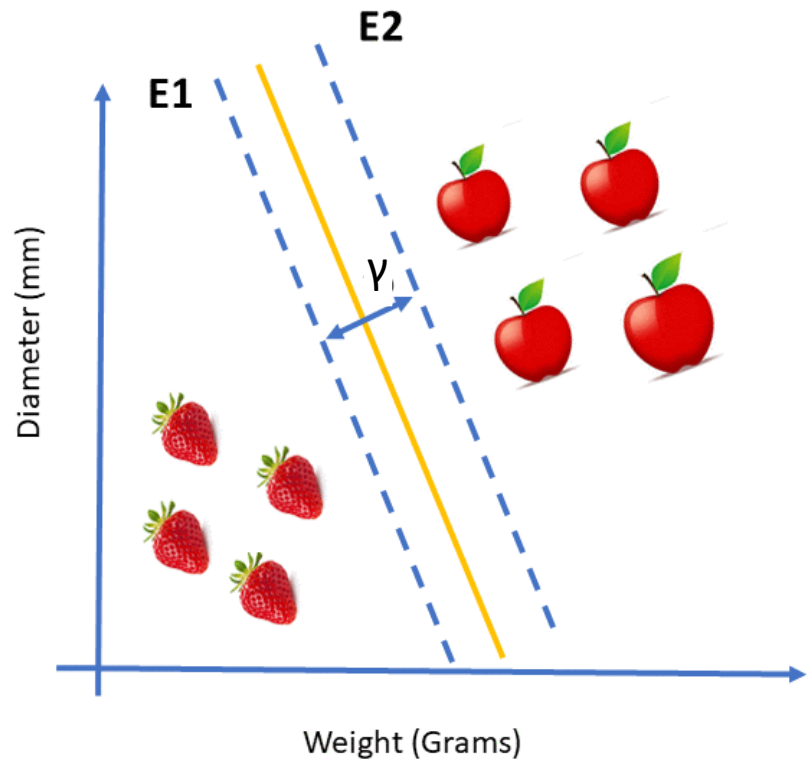
We can formulate our search for the maximum margin separating hyperplane as a constrained optimization problem. The objective is to maximize the margin under the constraints that all data points must lie on the correct side of the hyperplane:

$$\underbrace{\max_{\mathbf{w},b} \gamma(\mathbf{w}, b)}_{\textit{maximize margin}} \text{ such that } \underbrace{\forall i \; y_i(\mathbf{w}^T x_i + b) \geq 0}_{\textit{separating hyperplane}}$$



By definition, the margin and hyperplane are scale invariant: $\gamma(\beta\mathbf{w}, \beta b) = \gamma(\mathbf{w}, b), \forall \beta \neq 0$

The hyperplane lies in the middle of the two classes. Why?



Note that if the hyperplane is such that γ is maximized, it must lie right in the middle of the two classes. In other words, γ must be the distance to the closest point within both classes. (If not, you could move the hyperplane towards data points of the class that is further away and increase γ, which contradicts that γ is maximized.)

Maximal margin classifier optimization (2)

If we plug in the definition of γ we obtain:

$$\underbrace{\max_{\mathbf{w},b} \underbrace{\frac{1}{\|\mathbf{w}\|_2} \min_{\mathbf{x}_i \in D} \left|\mathbf{w}^T \mathbf{x}_i + b\right|}_{\gamma(\mathbf{w},b)}}_{\textit{maximize margin}} \quad s.t. \quad \underbrace{\forall i \; y_i(\mathbf{w}^T x_i + b) \geq 0}_{\textit{separating hyperplane}}$$

Because the hyperplane is scale invariant, we can fix the scale of $\mathbf{w}, b$ anyway we want. Let's be clever about it, and choose it such that

$$\min_{\mathbf{x} \in D} \left|\mathbf{w}^T \mathbf{x} + b\right| = 1.$$

We can add this re-scaling as an equality constraint. Then our objective becomes:

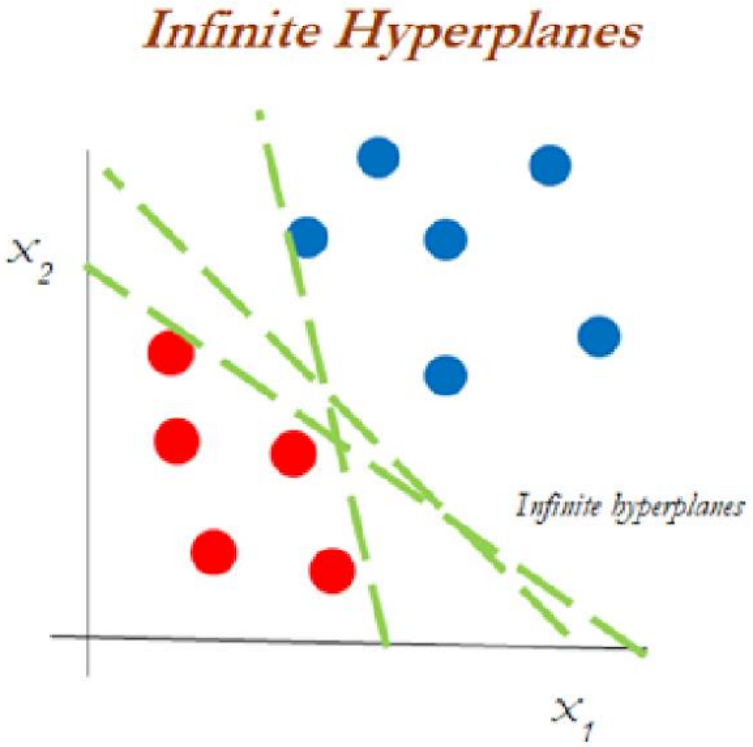
$$\max_{\mathbf{w},b} \frac{1}{\|\mathbf{w}\|_2} \cdot 1 = \min_{\mathbf{w},b} \|\mathbf{w}\|_2 = \min_{\mathbf{w},b} \mathbf{w}^\top \mathbf{w}$$

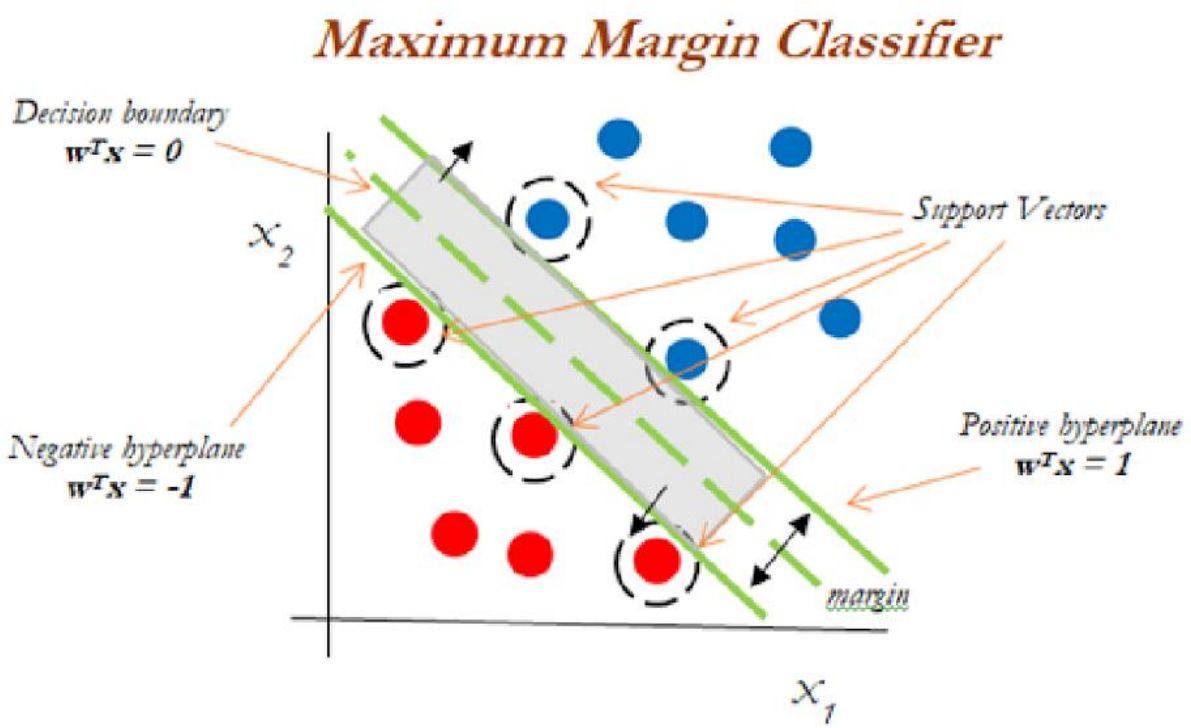
(Where we made use of the fact $f(z) = z^2$ is a monotonically increasing function for $z \geq 0$ and $\|\mathbf{w}\| \geq 0$; i.e. the $\mathbf{w}$ that maximizes $\|\mathbf{w}\|_2$ also maximizes $\mathbf{w}^\top \mathbf{w}$.)

Support vectors

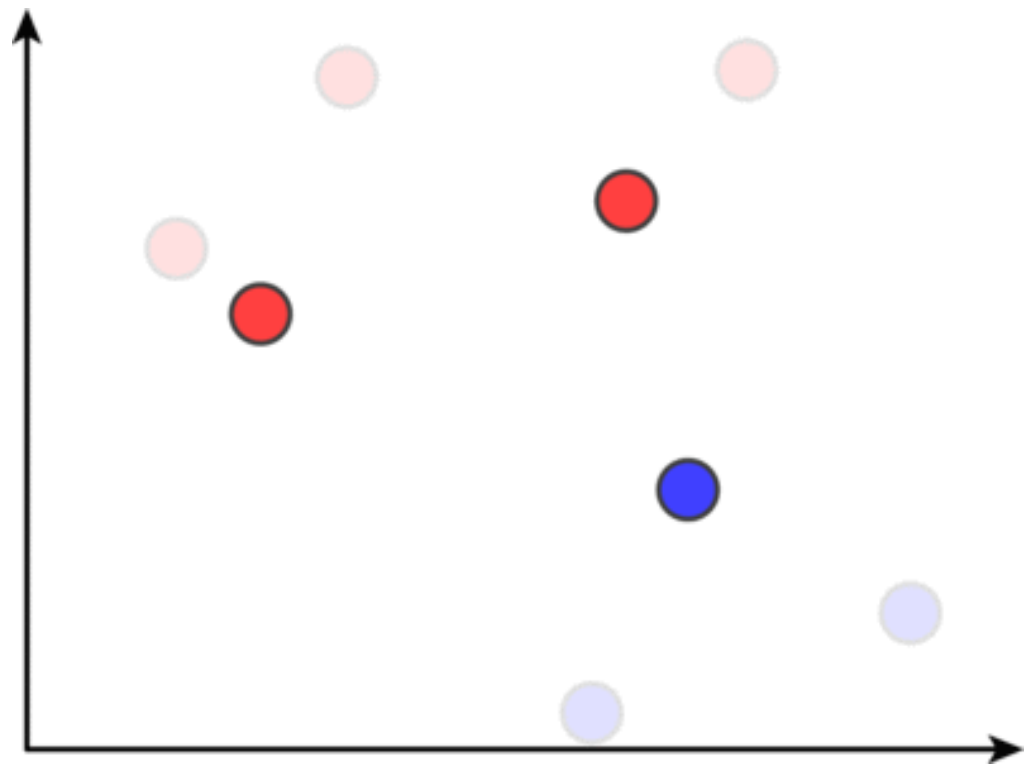
For the optimal $\mathbf{w}, b$ pair, some training points will have tight constraints, i.e.

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1.$$





These points determine the shape of the hyperplane!!!



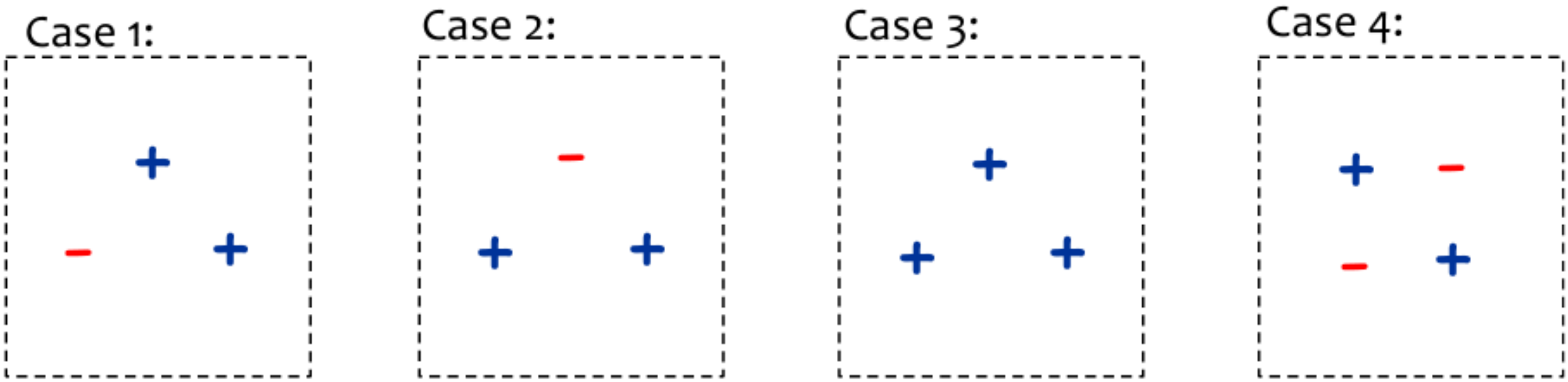
Class 2: beyond maximal margin classifiers

Maximal margin classifier limitations.

Soft margin classifier, slack parameters: learning, role of the parameter C

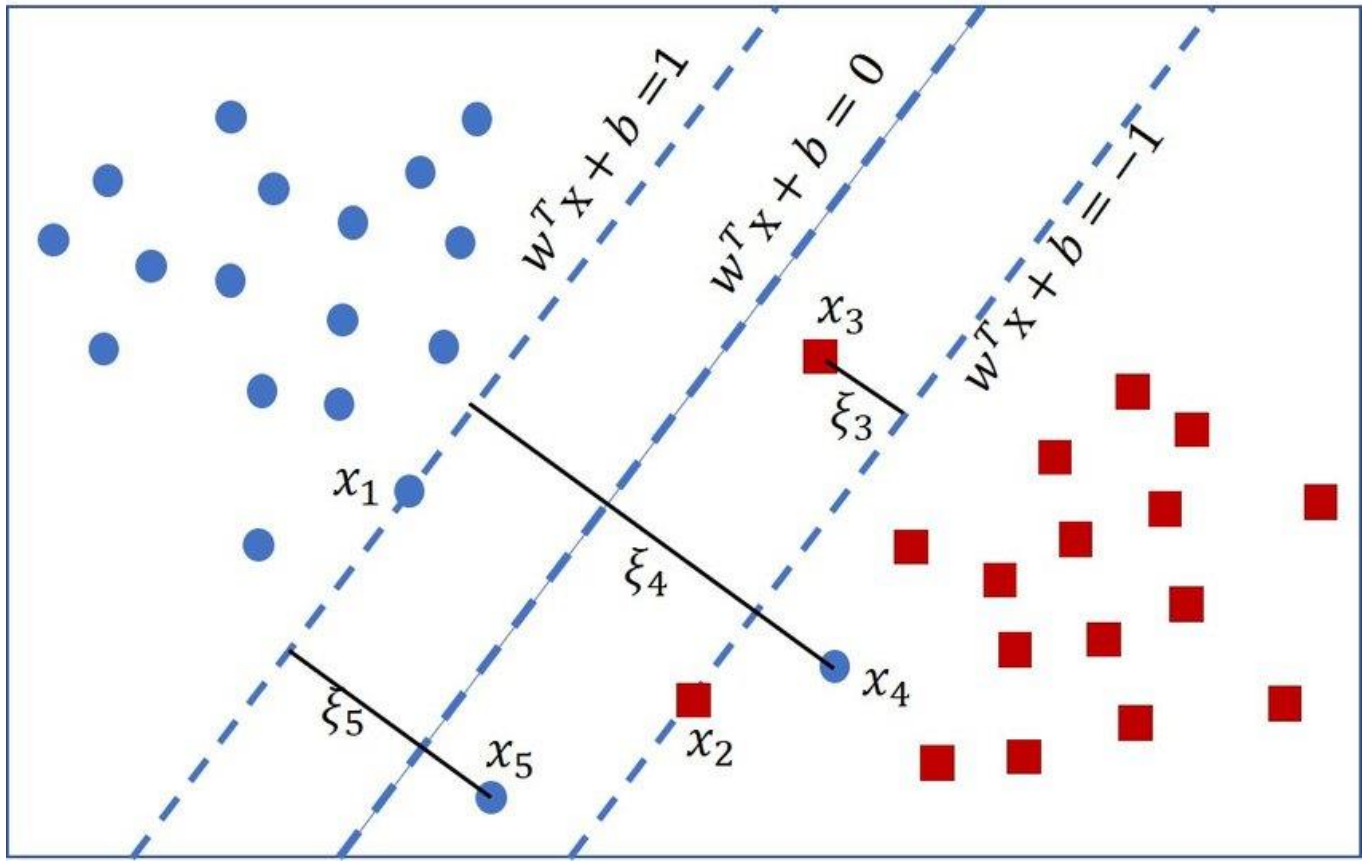
Dual problem and optimization

Linear separability

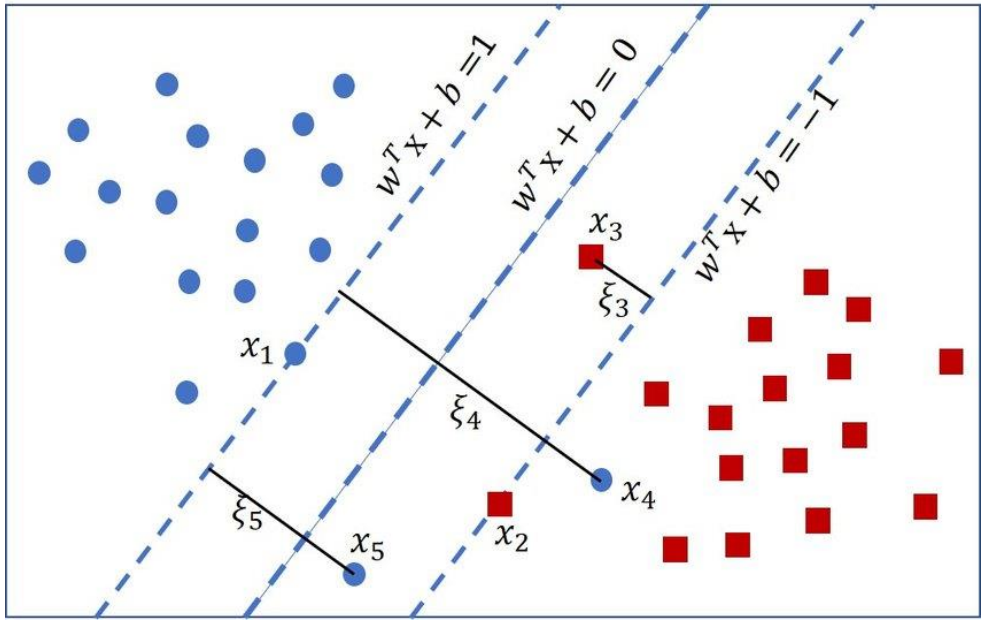


Can you separate + and - with a plane in all cases?

Which is the problem here?



Maximal margin classification and limitations



If the data is low dimensional it is often the case that there is no separating hyperplane between the two classes. In this case, there is no solution to the optimization problems stated above. We can fix this by allowing the constraints to be violated ever so slight with the introduction of slack variables:

$$\min_{\mathbf{w},b} \mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n} \xi_i$$
$$s.t.\ \forall i\ y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1 - \xi_i$$
$$\forall i\ \xi_i \geq 0$$

The slack variable ξ_i allows the input $\mathbf{x_i}$ to be closer to the hyperplane (or even be on the wrong side), but there is a penalty in the objective function for such "slack". If C is very large, the SVM becomes very strict and tries to get all points to be on the right side of the hyperplane. If C is very small, the SVM becomes very loose and may "sacrifice" some points to obtain a simpler (i.e. lower $\|\mathbf{w}\|_2^2$) solution.

New optimization problem

Let us consider the value of ξ_i for the case of $C \neq 0$. Because the objective will always try to minimize ξ_i as much as possible, the equation must hold as an *equality* and we have:

$$\xi_i = \begin{cases} 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) & \text{if } y_i(\mathbf{w}^T\mathbf{x}_i + b) < 1 \\ 0 & \text{if } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \end{cases}$$

This is equivalent to the following closed form:

$$\xi_i = \max(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b), 0).$$

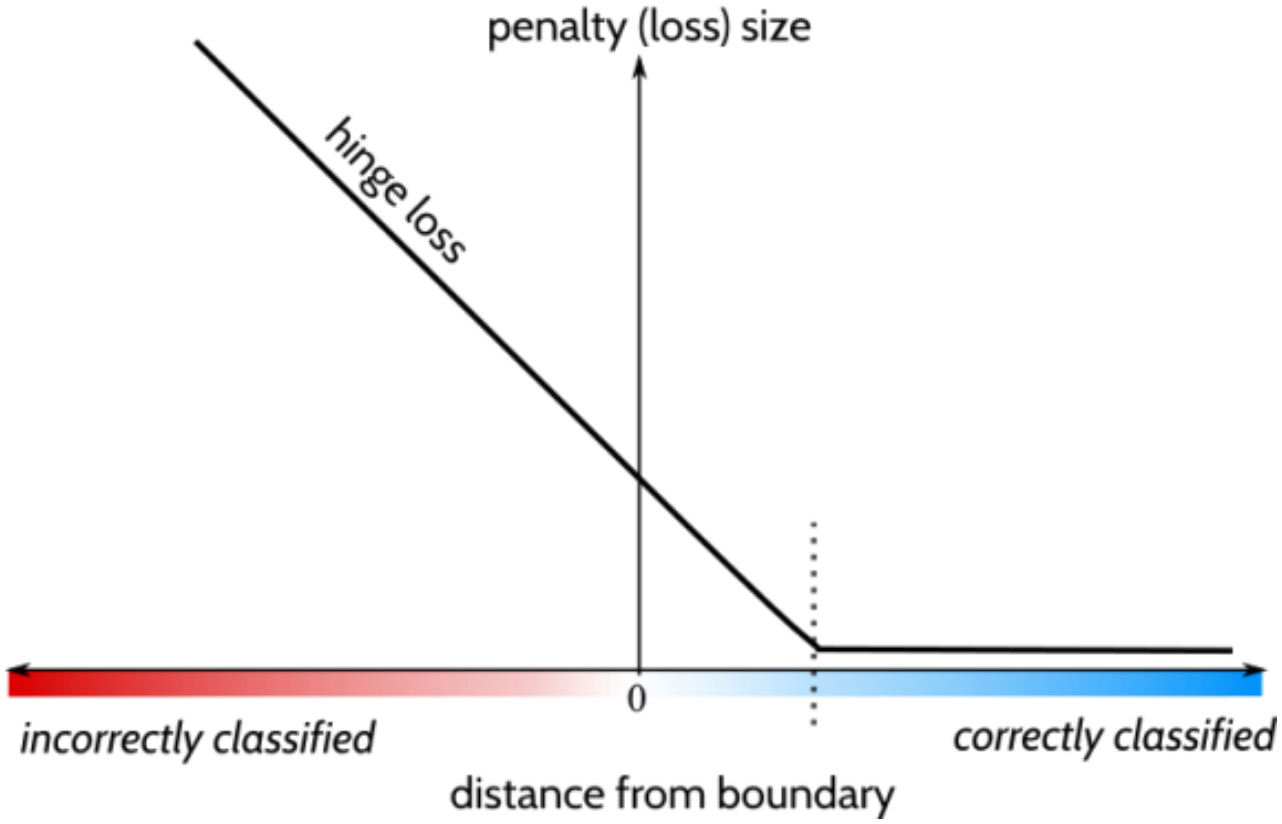
If we plug this closed form into the objective of our SVM optimization problem, we obtain the following *unconstrained* version as loss function and regularizer:

$$\min_{\mathbf{w},b} \underbrace{\mathbf{w}^T\mathbf{w}}_{l_2-regularizer} + C \sum_{i=1}^{n} \underbrace{\max\left[1 - y_i(\mathbf{w}^T\mathbf{x} + b), 0\right]}_{hinge-loss}$$

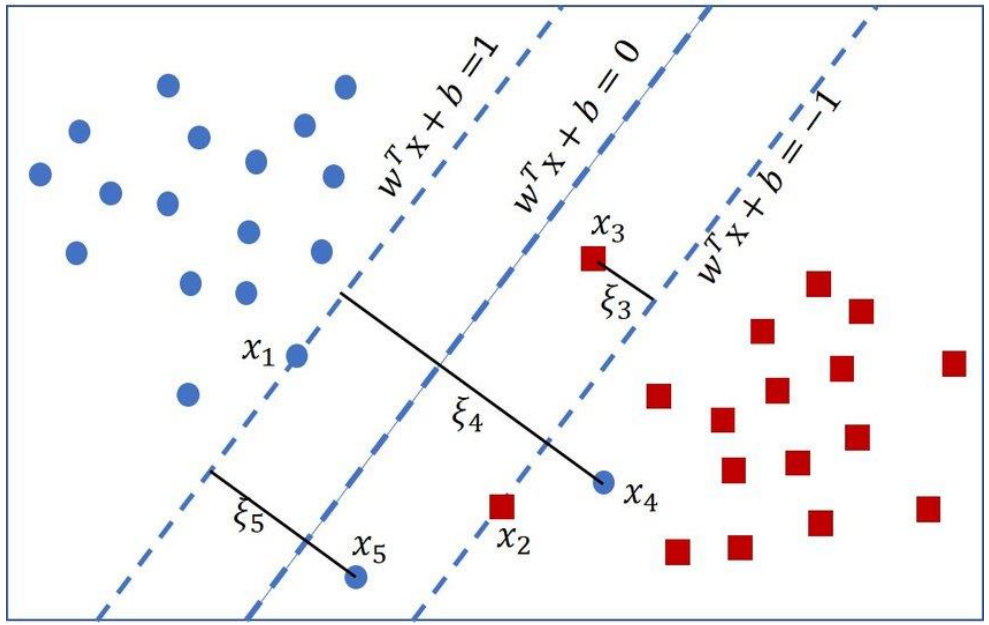
We minimize the errors

This formulation allows us to optimize the SVM paramters $(\mathbf{w}, b)$ just like logistic regression (e.g. through gradient descent). The only difference is that we have the **hinge-loss** instead of the **logistic loss**.

Hinge loss


penalty (loss) size
hinge loss
0
incorrectly classified
correctly classified
distance from boundary

Solving for the new optimization problem



slack penalty

$$\min_{\substack{w \in \mathbb{R}^d \\ \xi \geq 0}} \quad \frac{1}{2} w^\top w + C \sum_{i=1}^{n} \xi_i$$

$$\text{s.t.} \quad y_i(w^\top x_i + b) \geq 1 - \xi_i \qquad \forall i \in \{1, \ldots, n\}$$

The Lagrangian: $L(w, b, \xi, \alpha, \lambda)$

$$= \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i\left(1 - y_i\left(w^\top x_i + b\right) - \xi_i\right) + \sum_{i=1}^{n}\lambda_i(-\xi_i)$$

with variable constraints

$$\alpha_i \geq 0, \qquad \lambda_i \geq 0.$$

How do you minimize a function? Derive w.r.t. w, b, and ξ

Derivative wrt w:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \qquad w = \sum_{i=1}^{n} \alpha_i y_i x_i.$$

w is a linear combination of the training points!

Derivative wrt b:

$$\frac{\partial L}{\partial b} = \sum_i y_i \alpha_i = 0.$$

Derivative wrt ξ_i:

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \lambda_i = 0 \qquad \alpha_i = C - \lambda_i.$$

Noting that $\lambda_i \geq 0$,

$$\alpha_i \leq C.$$

Substituting…

$$
\begin{aligned}
g(\alpha,\lambda) &= \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i\left(1 - y_i\left(w^\top x_i + b\right) - \xi_i\right) \\
&\qquad + \sum_{i=1}^{n}\lambda_i(-\xi_i) \\
&= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^\top x_j + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^\top x_j \\
&\qquad - b\underbrace{\sum_{i=1}^{m}\alpha_i y_i}_{0} + \sum_{i=1}^{m}\alpha_i - \sum_{i=1}^{m}\alpha_i\xi_i - \sum_{i=1}^{m}\underbrace{(C-\alpha_i)}_{\lambda_i}\xi_i \\
&= \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^\top x_j.
\end{aligned}
$$

Finally

$$g(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^\top x_j,$$

subject to the constraints

$$0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} y_i \alpha_i = 0$$

So far only math….

Support vectors

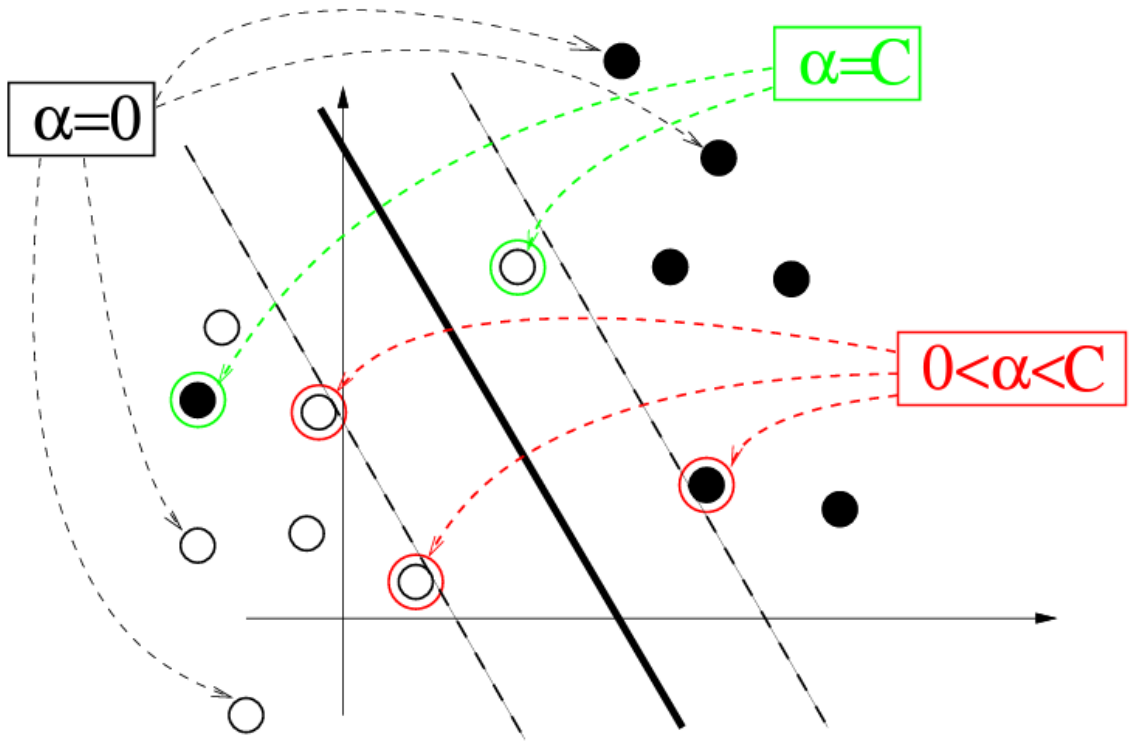
- The solution w can be written as a linear combination of the data

- Only those data for which $0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} y_i \alpha_i = 0$ are counted for the solution

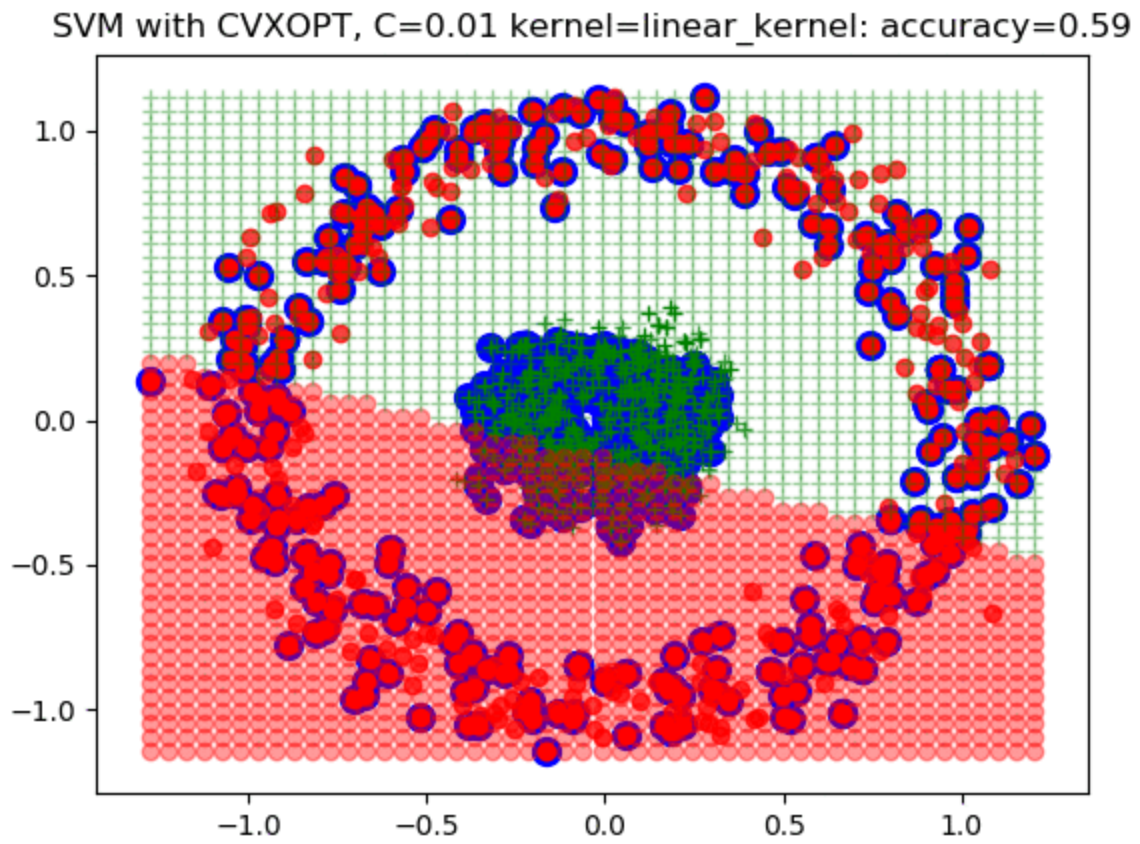
Support vectors

$$\min_{\substack{w \in \mathbb{R}^d \\ \xi \geq 0}} \quad \frac{1}{2} w^\top w + C \sum_{i=1}^{n} \xi_i$$

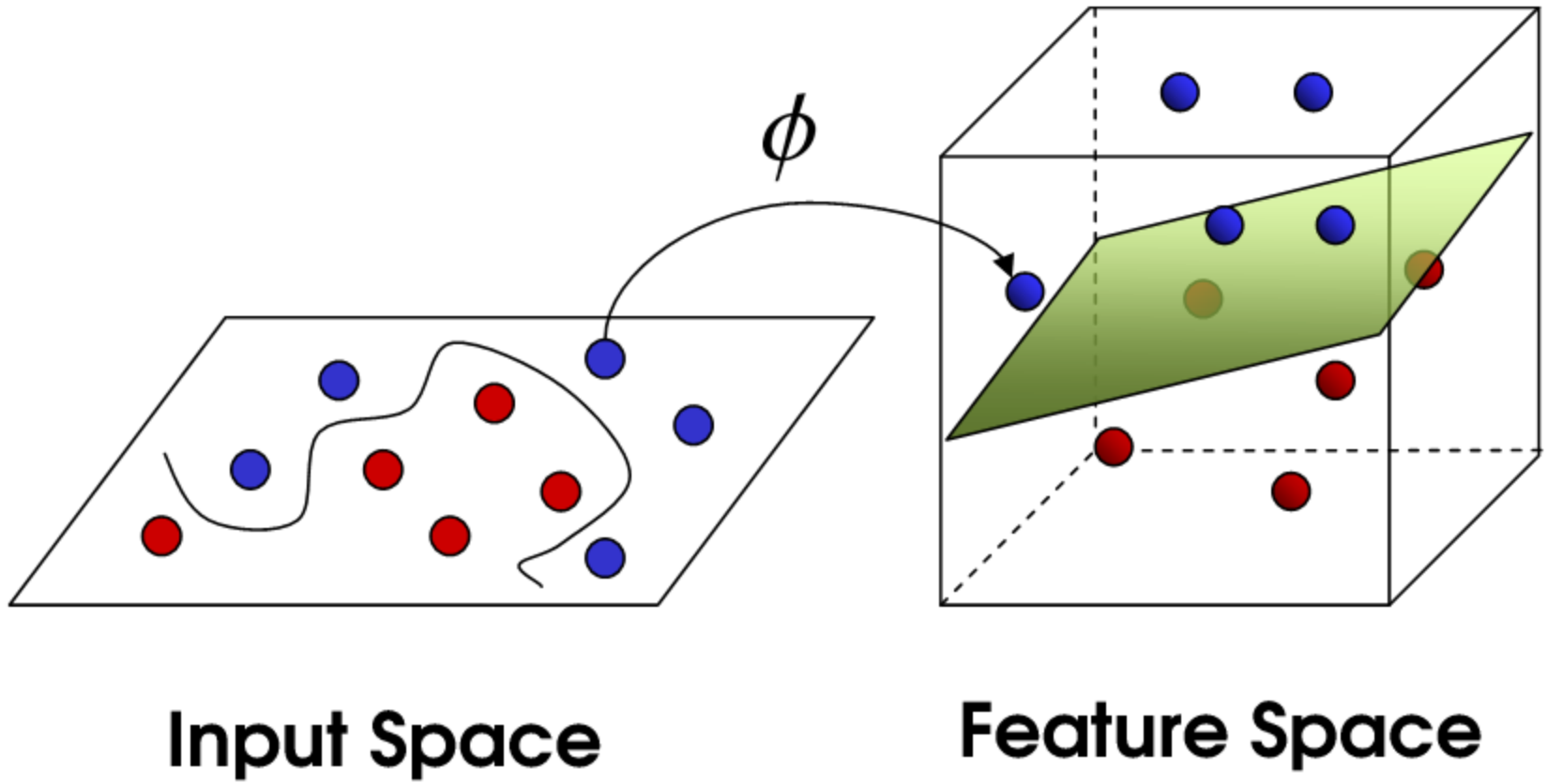
$$\text{s.t.} \quad y_i(w^\top x_i + b) \geq 1 - \xi_i$$

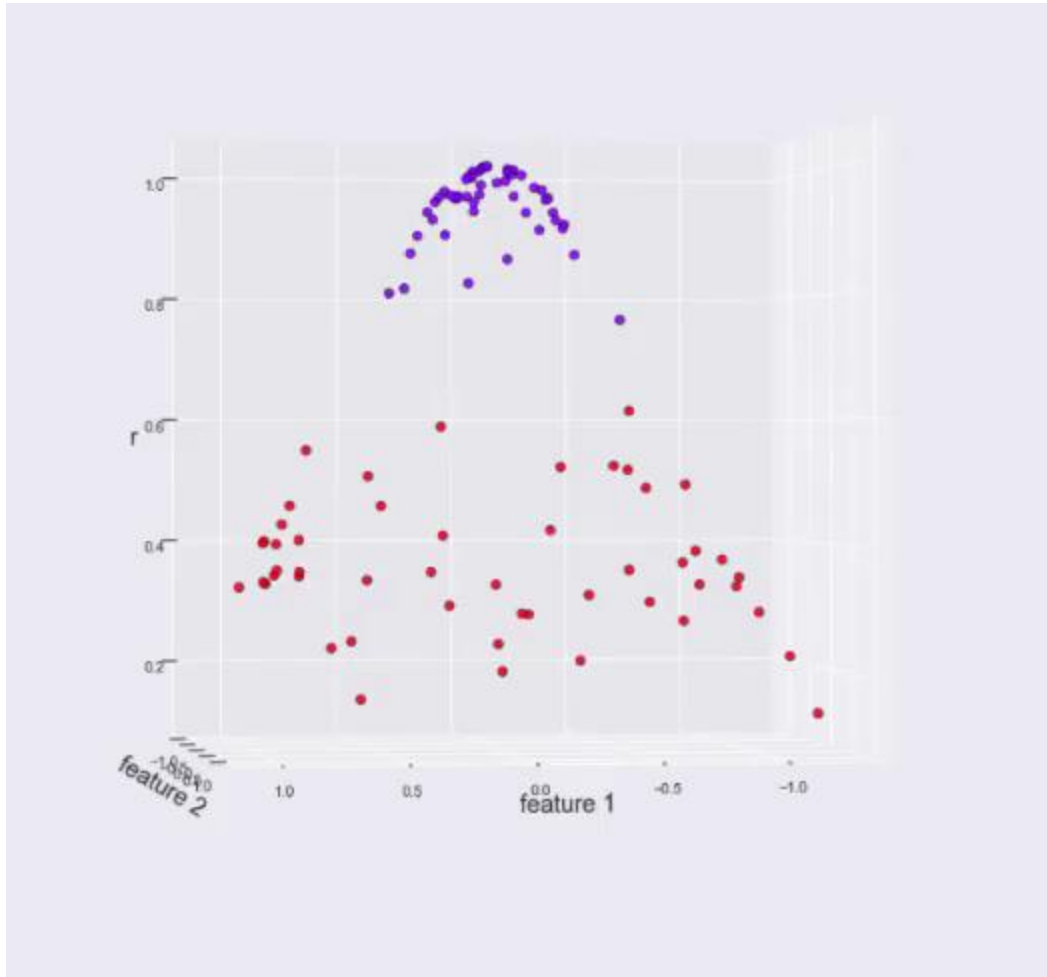


Problem of linear SVMs



Idea




1.0
0.8
0.6
r
0.4
0.2
1.0
0.5
0.0
-0.5
-1.0
feature 1
feature 2

Class 3

Kernel trick for linear regression and non linearly separable problems

Feature augmentation and intuition

So far:

- Perceptron
- Maximal margin classifiers SVM
- Maximal margin with errors
- Beyond linear separators

Important observation

- All quantities (Loss and y) can be expressed as products of data

$$\sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^\top x_j. \qquad w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

- We want to show that the solution of the optimization problem can be expressed in terms of the data inner products only (kernel trick).
- Next we show that this is true also for regression

Regression example

We want to prove that:
using gradient descent with regression we get a solution that is a linear combination of the training points

LOSS:

$$\ell(\mathbf{w}) = \sum_{i=1}^{n} (\mathbf{w}^\top \mathbf{x}_i - y_i)^2$$

GRADIENT:

$$w_{t+1} \leftarrow w_t - s\left(\frac{\partial \ell}{\partial \mathbf{w}}\right) \text{ where: } \frac{\partial \ell}{\partial \mathbf{w}} = \sum_{i=1}^{n} \underbrace{2(\mathbf{w}^\top \mathbf{x}_i - y_i)}_{\gamma_i \text{ : function of } \mathbf{x}_i, y_i} \mathbf{x}_i = \sum_{i=1}^{n} \gamma_i \mathbf{x}_i$$

We will now show that we can express $\mathbf{w}$ as a linear combination of all input vectors,

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i.$$

Since the loss is convex, the final solution is independent of the initialization, and we can initialize $\mathbf{w}^0$ to be whatever we want. For convenience, let us pick $\mathbf{w}_0 = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$. For this initial choice of $\mathbf{w}_0$, the linear combination in $\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$ is trivially $\alpha_1 = \cdots = \alpha_n = 0$. We now show that throughout the entire gradient descent optimization such coefficients $\alpha_1, \ldots, \alpha_n$ must always exist, as we can re-write the gradient updates entirely in terms of updating the α_i coefficients:

$$\begin{aligned}
\mathbf{w}_1 &= \mathbf{w}_0 - s \sum_{i=1}^{n} 2(\mathbf{w}_0^\top \mathbf{x}_i - y_i)\mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^0 \mathbf{x}_i - s \sum_{i=1}^{n} \gamma_i^0 \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^1 \mathbf{x}_i && (\text{with } \alpha_i^1 = \alpha_i^0 - s\gamma_i^0) \\
\mathbf{w}_2 &= \mathbf{w}_1 - s \sum_{i=1}^{n} 2(\mathbf{w}_1^\top \mathbf{x}_i - y_i)\mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^1 \mathbf{x}_i - s \sum_{i=1}^{n} \gamma_i^1 \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^2 \mathbf{x}_i && (\text{with } \alpha_i^2 = \alpha_i^1 \mathbf{x}_i - s\gamma_i^1) \\
\mathbf{w}_3 &= \mathbf{w}_2 - s \sum_{i=1}^{n} 2(\mathbf{w}_2^\top \mathbf{x}_i - y_i)\mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^2 \mathbf{x}_i - s \sum_{i=1}^{n} \gamma_i^2 \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^3 \mathbf{x}_i && (\text{with } \alpha_i^3 = \alpha_i^2 - s\gamma_i^2) \\
&\ldots \qquad\qquad \ldots && \ldots \\
\mathbf{w}_t &= \mathbf{w}_{t-1} - s \sum_{i=1}^{n} 2(\mathbf{w}_{t-1}^\top \mathbf{x}_i - y_i)\mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^{t-1} \mathbf{x}_i - s \sum_{i=1}^{n} \gamma_i^{t-1} \mathbf{x}_i = \sum_{i=1}^{n} \alpha_i^t \mathbf{x}_i && (\text{with } \alpha_i^t = \alpha_i^{t-1} - s\gamma_i^{t-1})
\end{aligned}$$

Formally, the argument is by induction. $\mathbf{w}$ is trivially a linear combination of our training vectors for $\mathbf{w}_0$ (base case). If we apply the inductive hypothesis for $\mathbf{w}_t$ it follows for $\mathbf{w}_{t+1}$.

The update-rule for α_i^t is thus

$$\alpha_i^t = \alpha_i^{t-1} - s\gamma_i^{t-1}, \text{ and we have } \alpha_i^t = -s \sum_{r=0}^{t-1} \gamma_i^r.$$

Let us now calculate the regression function for a point

$$\mathbf{w}^\top \mathbf{x}_j = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i^\top \mathbf{x}_j.$$

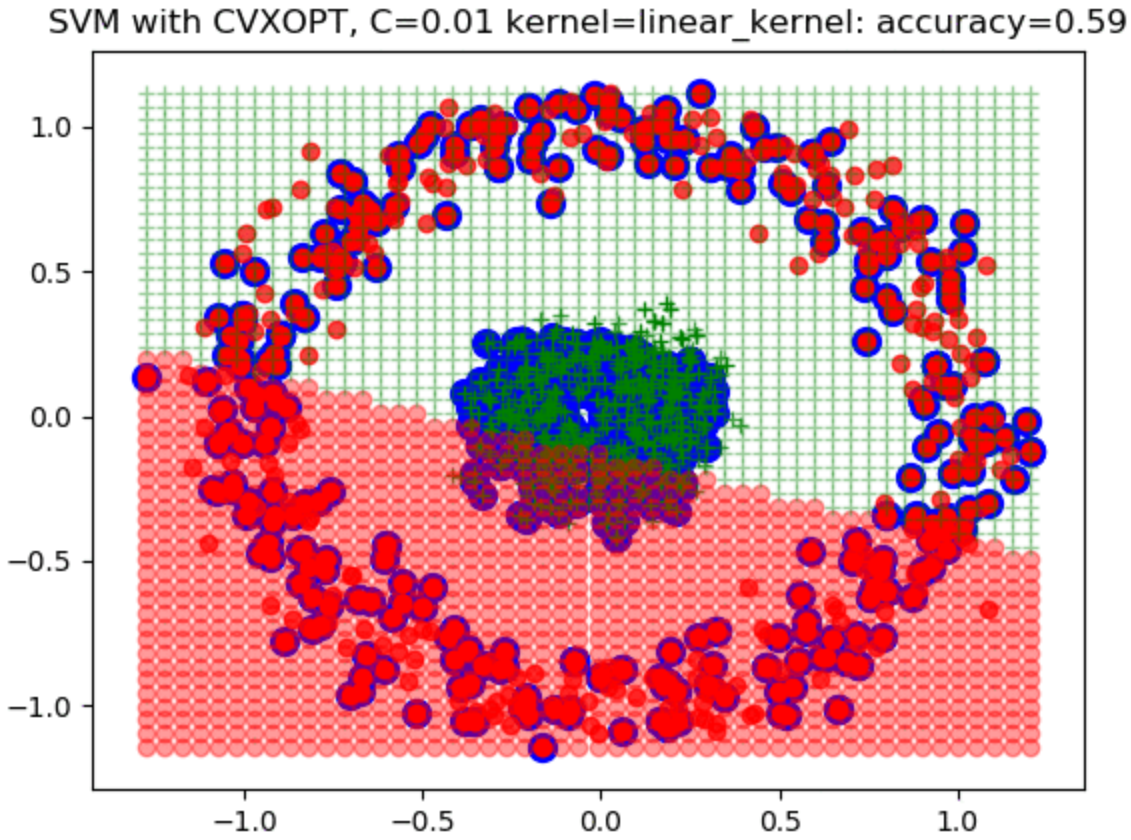
Is a linear combination of the dot product of the point with the training data

And the loss

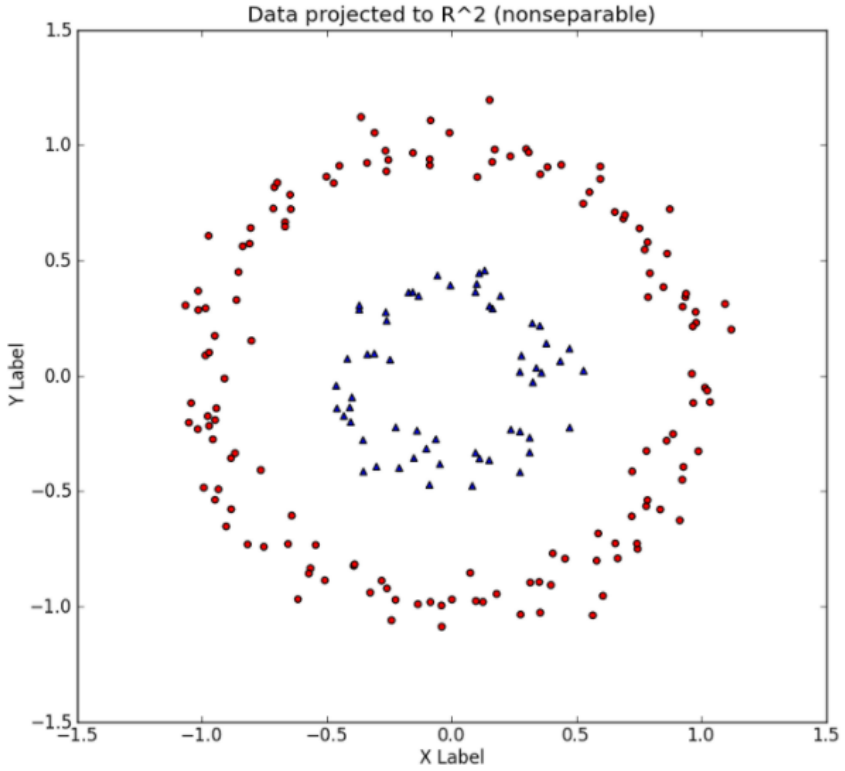
$$\ell(\alpha) = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_j \mathbf{x}_j^\top \mathbf{x}_i - y_i \right)^2$$

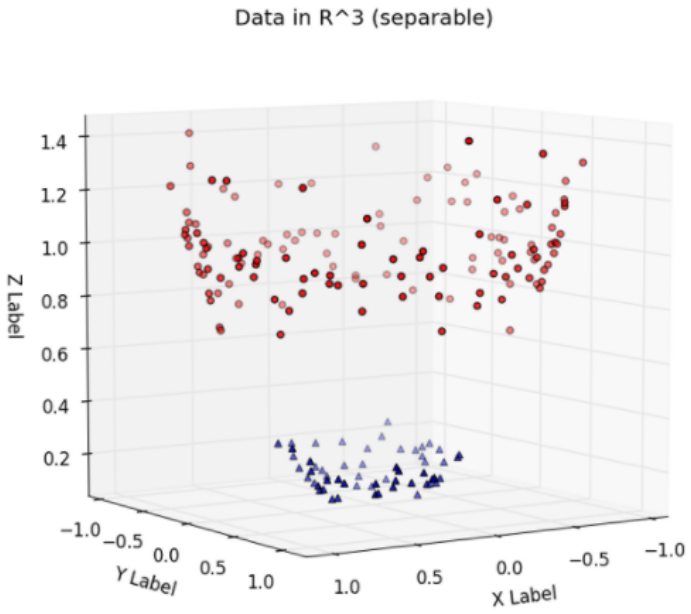
Do you notice a theme? The only information we ever need in order to learn a hyper-plane classifier with the squared-loss is inner-products between all pairs of data vectors.

Why are these observations important?
What if the problem is non linearly separable?



Adding new dimensions and non linear features

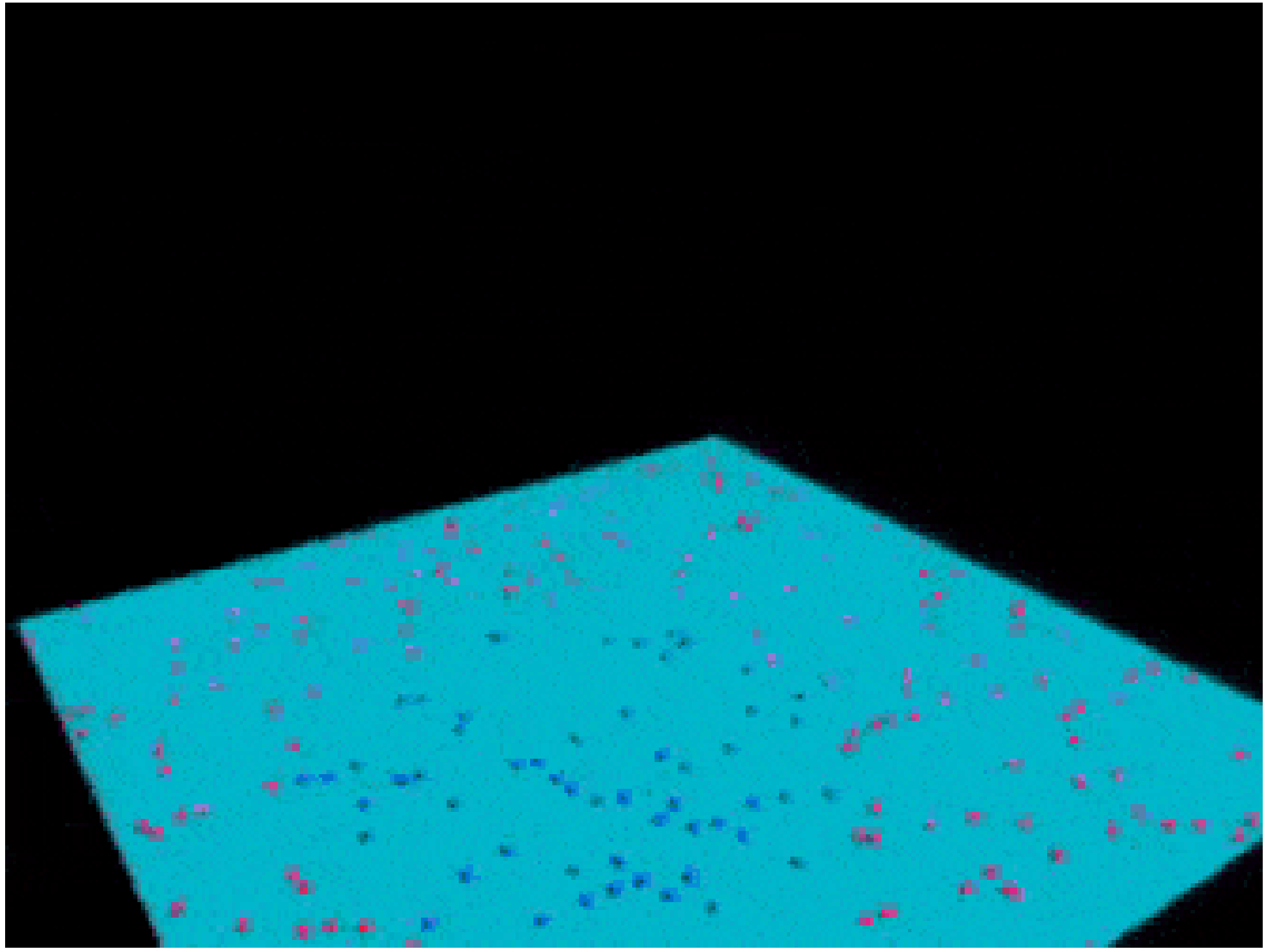


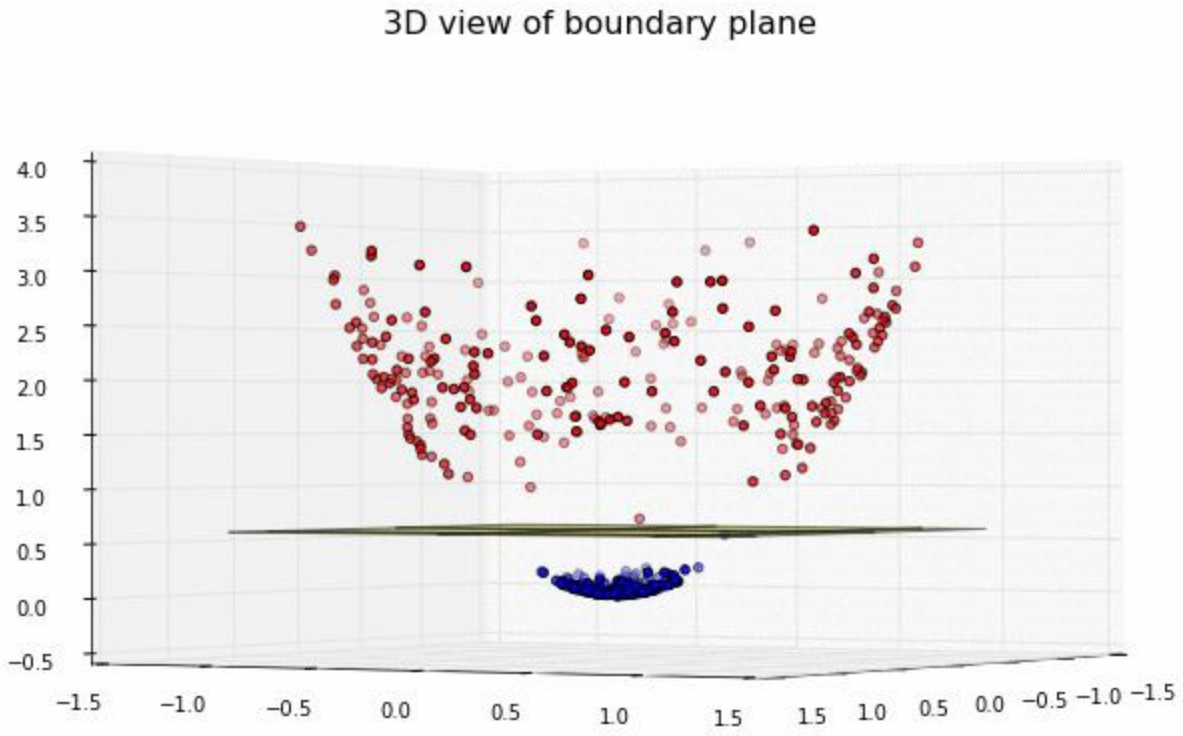


r

$$(x, y) \to (x, y, x^2 + y^2)$$

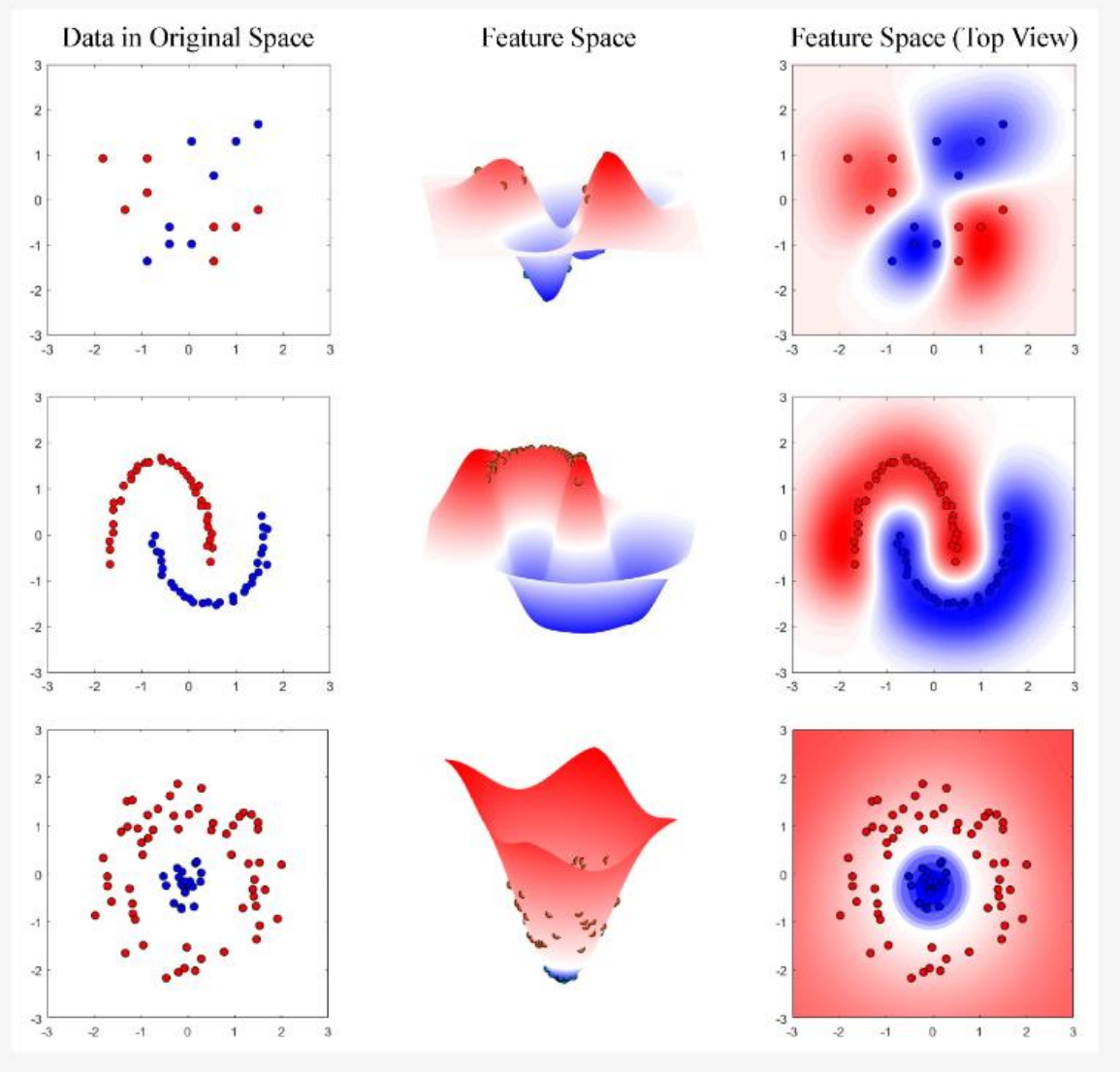
Idea: add dimensions and try to make the problem separable





In this case the solution is “obvious ”. But what about more complex shapes?

We would like to have a rich feature space such that e.g.



Which features should we use?

Use non-linear features to write non-linear separators

$$g(\boldsymbol{x}) = 0.5 - (x_1^2 + x_2^2). \tag{2.131}$$

For any $\boldsymbol{x}$ such that $g(\boldsymbol{x}) > 0$, we declare $\boldsymbol{x}$ belongs to class $+1$, and for any $\boldsymbol{x}$ such that $g(\boldsymbol{x}) < 0$ we declare $\boldsymbol{x}$ belongs to class -1. The hypothesis function is thus

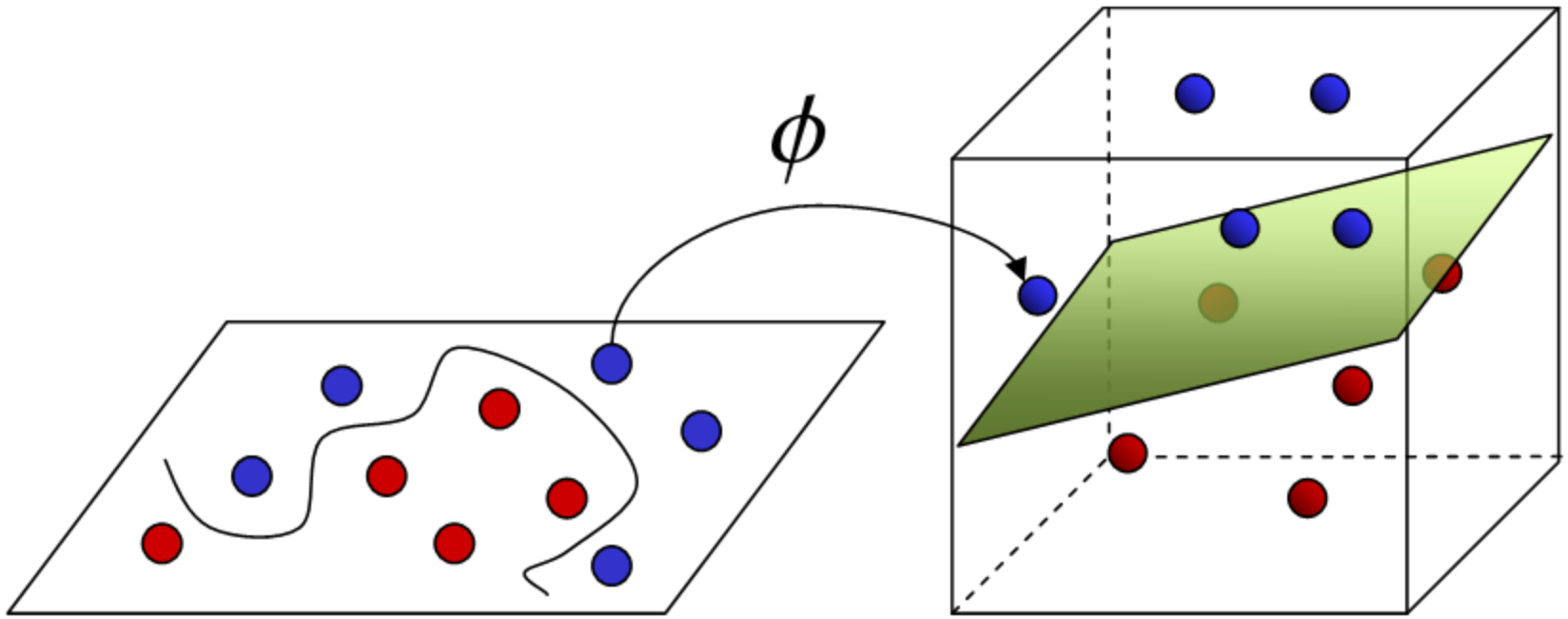
$$h(\boldsymbol{x}) = \text{sign}(g(\boldsymbol{x})) = \text{sign}\left(0.5 - x_1^2 - x_2^2\right). \tag{2.132}$$

Let us try to rewrite $h(\boldsymbol{x})$. Define $z_0 = 1$, $z_1 = x_1^2$ and $z_2 = x_2^2$. Then, $h(\boldsymbol{x})$ becomes

$$\begin{aligned} h(\boldsymbol{x}) &= \text{sign}\left(0.5 - x_1^2 - x_2^2\right) \\ &= \text{sign}\left(\begin{bmatrix} 0.5 & -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1^2 \\ x_2^2 \end{bmatrix}\right) = \text{sign}\left(\begin{bmatrix} \widetilde{w}_0 & \widetilde{w}_1 & \widetilde{w}_2 \end{bmatrix} \begin{bmatrix} z_0 \\ z_1 \\ z_2 \end{bmatrix}\right). \end{aligned} \tag{2.133}$$

Therefore, we are able to rewrite a nonlinear discriminant function into a linear form by using a transformation

$$\boldsymbol{z} = \Phi(\boldsymbol{x}) = \begin{bmatrix} 1 \\ x_1^2 \\ x_2^2 \end{bmatrix}. \tag{2.134}$$


ϕ
Input Space
Feature Space

Which features?

Let’s use them all!!!

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}, \text{ and define } \phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \cdots x_d \end{pmatrix}.$$

We expand the dimensionality with new correlation features

What's the problem of the approach?

- For example, how about using power series expansion?

$$(X, Y, Z) \rightarrow (X, Y, Z, X^2, Y^2, Z^2, XY, YZ, ZX, \ldots)$$

- But, many recent data are high-dimensional.
  e.g. microarray, images, etc...

  The above expansion is intractable!

  e.g. Up to 2nd moments, 10,000 dimension:

  Dim of feature space: ${}_{10000}C_1 + {}_{10000}C_2 = 50{,}005{,}000$

Exercise:

$x \in \mathbb{R},\ 0 < x < 1$

$$\phi(x) = (1, x, x^2, \cdots)$$

Calculate $\phi(x)^T \phi(y)$

Solution:

$x \in \mathbb{R},\ 0 < x < 1$

$$\phi(x) = (1, x, x^2, \cdots)$$

$$K(x, y) = \phi(x)^T \phi(y) = \sum_{i=0}^{\infty} (xy)^i = \frac{1}{1 - xy}$$

Moreover...

d=1

$$\phi(u).\phi(v) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} . \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2 = u.v$$

d=2

$$\phi(u).\phi(v) = \begin{pmatrix} u_1^2 \\ u_1 u_2 \\ u_2 u_1 \\ u_2^2 \end{pmatrix} . \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2 v_1 \\ v_2^2 \end{pmatrix} = u_1^2 v_1^2 + 2 u_1 v_1 u_2 v_2 + u_2^2 v_2^2$$
$$= (u_1 v_1 + u_2 v_2)^2$$
$$= (u.v)^2$$

For any *d (we will skip proof):*

$$\phi(u).\phi(v) = (u.v)^d$$

Polynomials of degree **exactly** *d*

Summarizing

- Consider high dimensional feature vectors that make the problem from non separable to separable

$$(x, y) \rightarrow (x, y, x^2 + y^2)$$

- The optimization problem can be written in terms of inner products of these vectors

- The inner products correspond to kernel functions $\phi(u).\phi(v) = (u.v)^d$

Gaussian kernel

$$K(x_i, x_j) = \exp\left(-\frac{1}{2\sigma^2}||x_i - x_j||^2\right) \qquad \sigma = 1/\sqrt{2}$$

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

$$= \exp\left(-x_i^\top x_i\right) \exp\left(-x_j^\top x_j\right) \exp(2x_i^\top x_j) \qquad \exp(z) = \sum_{n=0}^{\infty} \frac{z^n}{n!}$$

$$= \exp\left(-x_i^\top x_i\right) \exp\left(-x_j^\top x_j\right) \sum_{n=0}^{\infty} \frac{2^n (x_i^\top x_j)^n}{n!}$$

order-n polynomial kernel* $\Phi^n(x)$

$$\Phi^{\text{rbf}} = \exp(-x^\top x)\left[\Phi^1(x)^\top, \Phi^2(x)^\top, \ldots, \Phi^\infty(x)^\top\right]^\top$$

Fourier Kernels

$$
\begin{aligned}
K(x-y) &= \int_{R^d} p(w) exp(-jw^T(x-y)) \\
&= \int_{R^d} p(w) exp(-jw^T x).exp(jw^T y)) \\
&= \int_{R^d} p(w)(cos(w^T x) - jsin(w^T x)).(cos(w^T y) + jsin(w^T y)) \\
&= \int_{R^d} p(w)(cos(w^T x)cos(w^T y) + sin(w^T x)sin(w^T y)) \\
&= E_p\left[< (cos(w^T x), sin(w^T x)), (cos(w^T y), sin(w^T y)) >\right]
\end{aligned}
$$

We can also approximate the kernel

Consider this alternative definition of the random map:

$$z_{\omega_r}(\mathbf{x}) = \begin{bmatrix} \cos(\boldsymbol{\omega}_r^\top \mathbf{x}) \\ \sin(\boldsymbol{\omega}_r^\top \mathbf{x}) \end{bmatrix}.$$

Draw $R' = R/2$ samples

$$\boldsymbol{\omega}_r \sim p(\boldsymbol{\omega}).$$

Then

$$\begin{aligned}
\frac{1}{R'} \sum_{r=1}^{R'} z_{\omega_r}(\mathbf{x})^\top z_{\omega_r}(\mathbf{y}) &= \frac{2}{R} \sum_{r=1}^{R/2} \left(\begin{bmatrix} \cos(\boldsymbol{\omega}_r^\top \mathbf{x}) \\ \sin(\boldsymbol{\omega}_r^\top \mathbf{x}) \end{bmatrix}^\top \begin{bmatrix} \cos(\boldsymbol{\omega}_r^\top \mathbf{y}) \\ \sin(\boldsymbol{\omega}_r^\top \mathbf{y}) \end{bmatrix} \right) \\
&= \frac{2}{R} \sum_{r=1}^{R/2} \cos(\boldsymbol{\omega}_r^\top \mathbf{x}) \cos(\boldsymbol{\omega}_r^\top \mathbf{y}) + \sin(\boldsymbol{\omega}_r^\top \mathbf{x}) \sin(\boldsymbol{\omega}_r^\top \mathbf{y}) \\
&\overset{\star}{=} \frac{2}{R} \sum_{r=1}^{R/2} \cos(\boldsymbol{\omega}_r^\top \mathbf{x} - \boldsymbol{\omega}_r^\top \mathbf{y}) \\
&\approx \mathbb{E}_\omega[\cos(\boldsymbol{\omega}^\top (\mathbf{x} - \mathbf{y}))] \\
&= k(\mathbf{x}, \mathbf{y}).
\end{aligned}$$

From a famous paper: https://papers.nips.cc/paper_files/paper/2007/hash/013a006f03dbc5392effeb8f18fda755-Abstract.html

So let’s forget the features and start from the kernel

What defines a valid kernel function?

Preamble: Positive definite matrices

In mathematics, a symmetric matrix M with real entries is **positive-definite** if the real number $z^{\mathrm{T}} M z$ is positive for every nonzero real column vector z, where z^{T} is the transpose of z.[1] More generally, a Hermitian matrix (that is, a complex matrix equal to its conjugate transpose) is **positive-definite** if the real number $z^* M z$ is positive for every nonzero complex column vector z, where z^* denotes the conjugate transpose of z.

What defines a valid kernel function?

Let $\boldsymbol{z}$ be an arbitrary vector. Then,

$$\begin{aligned}\boldsymbol{z}^T\boldsymbol{K}\boldsymbol{z} &= \sum_{i=1}^{n}\sum_{j=1}^{n} z_i K_{ij} z_j = \sum_{i=1}^{n}\sum_{j=1}^{n} z_i \Phi(\boldsymbol{x}_i)^T \Phi(\boldsymbol{x}_j) z_j \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} z_i \left(\sum_{k=1}^{n} [\Phi(\boldsymbol{x}_i)]_k [\Phi(\boldsymbol{x}_j)]_k\right) z_j \overset{(a)}{=} \sum_{k=1}^{n}\left(\sum_{i=1}^{n} [\Phi(\boldsymbol{x}_i)]_k z_i\right)^2 \geq 0,\end{aligned}$$

where $[\Phi(\boldsymbol{x}_i)]_k$ denotes the k-th element of the vector $\Phi(\boldsymbol{x}_i)$. The result shows that if K is a valid kernel, then the corresponding matrix $\boldsymbol{K}$ must be symmetric positive semi-definite. More generally, this result is necessary and sufficient for K to be a valid kernel. That is, K is a valid kernel if and only if $\boldsymbol{K}$ is symmetric positive semi-definite. Kernels satisfying this condition is called **Mercer Kernel**.

Positive definite symmetric

Mercer kernel

Theorem (Mercer): Let K: $\mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ be given. Then for K to be a valid (Mercer) kernel, it is necessary and sufficient that for any $\{x_i, \ldots, x_m\}$, $(m < \infty)$, the corresponding kernel matrix is symmetric positive semi-denite.

Why do we care in practice ? Representer theorem

(Representer Theorem): *The optimal solution to any vector valued function learning problem of the form.*

$$f^* = \text{argmin}_{f \in \mathcal{H}_K} L(f(\mathbf{x}_1) \dots f(\mathbf{x}_l)) + \lambda \|f\|^2_{\mathcal{H}_K}$$

is a sum of matrix-vector products of the form

$$f^*(\mathbf{x}) = \sum_{i=1}^{l} K(\mathbf{x}, \mathbf{x}_i)\boldsymbol{\alpha}_i$$

where $\boldsymbol{\alpha}_i \in \mathbb{R}^n, \quad i = 1 \dots l$, *L is an arbitrary loss function (which can also be an indicator function encoding arbitrary constraints) and* $\lambda > 0$ *is a regularization parameter.*

We reduced an infinite dimensional non-linear optimization to a linear finite one!!!

Reproducing property!

Proof…

Suppose we project f onto the subspace:

$$\text{span}\{k(x_i, \cdot)\colon\ 1 \leq i \leq n\}$$

obtaining f_s (the component along the subspace) and $f_\perp$ (the component perpendicular to the subspace). We have:

$$f = f_s + f_\perp \Rightarrow ||f||^2 = ||f_s||^2 + ||f_\perp||^2 \geq ||f_s||^2$$

Since Ω is non-decreasing,

$$\Omega(||f||_H^2) \geq \Omega(||f_s||_H^2)$$

...Proof

implying that $\Omega(\cdots)$ is minimized if f lies in the subspace. Furthermore, since the kernel k has the reproducing property, we have:

$$f(x_i) = \langle f, k(x_i, \cdot)\rangle = \langle f_s, k(x_i, \cdot)\rangle + \langle f_\perp, k(x_i, \cdot)\rangle = \langle f_s, k(x_i, \cdot)\rangle = f_s(x_i)$$

Implying that:

$$L(f(x_1), \ldots, f(x_n)) = L(f_s(x_1), \ldots, f_s(x_n))$$

Hence, $L(\cdots)$ depends only on the component of f lying in the subspace: $\text{span}\{k(x_i, \cdot)\colon 1 \leq i \leq n\}$, and $\Omega(\cdots)$ is minimized if f lies in that subspace. Hence, $J(f)$ is minimized if f lies in that subspace, and we can express the minimizer as:

$$f^*(\cdot) = \sum_{i=1}^{n} \alpha_i k(x_i, \cdot)$$

Note that if $\Omega(\cdot)$ is strictly non-decreasing, then $||f_\perp||$ must necessarily be zero for f to be the minimizer of $J(f)$, implying that f^* must necessarily lie in the subspace: $\text{span}\{k(x_i, \cdot)\colon 1 \leq i \leq n\}$.

□

Different kernels different regularizations

- The norm depends on the kernel

  Changing the kernel we change the regularization!!!

Since $f = \sum_{j=1}^{n} c_j K_{x_j}$, then

$$\begin{aligned}
\|f\|_{\mathcal{H}}^2 &= \langle f, f \rangle_{\mathcal{H}} \\
&= \langle \sum_{i=1}^{n} c_i K_{x_i}, \sum_{j=1}^{n} c_j K_{x_j} \rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \langle K_{x_i}, K_{x_j} \rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(x_i, x_j) = c^t \mathbf{K} c
\end{aligned}$$

$$\|f\|_{\mathcal{H}} = c^T K c = c^T U^T \Lambda U c = \tilde{c}^T \Lambda \tilde{c}$$

Examples of different Function space regularization (different norms)

Gaussian kernel

$$K(x, y) = e^{-\frac{(x-y)^2}{2\sigma^2}}$$

corresponds to:

$$\varphi(t) = e^{-\frac{t^2}{2\sigma^2}}$$

$$\hat{\varphi}(\omega) = e^{-\frac{\sigma^2\omega^2}{2}}$$

and

$$\mathcal{H} = \left\{ f : \int \left| \hat{f}(\omega) \right|^2 e^{\frac{\sigma^2\omega^2}{2}} d\omega < \infty \right\}.$$

In particular, all functions in $\mathcal{H}$ are infinitely differentiable with all derivatives in L^2.

Laplace kernel

$$K(x,y) = \frac{1}{2} e^{-\gamma|x-y|}$$

corresponds to:

$$\varphi(to) = \frac{1}{2} e^{-\gamma|t|}$$
$$\hat{\varphi}(\omega) = \frac{\gamma}{\gamma^2 + \omega^2}$$

and

$$\mathcal{H} = \left\{ f : \int \left| \hat{f}(\omega) \right|^2 \frac{\left(\gamma^2 + \omega^2\right)}{\gamma} d\omega < \infty \right\},$$

the set of functions L^2 differentiable with derivatives in L^2 (Sobolev norm).

Low-frequency filter

$$K(x,y) = \frac{\sin(\Omega(x-y))}{\pi(x-y)}$$

corresponds to:

$$\varphi(t) = \frac{\sin(\Omega t)}{\pi t}$$
$$\hat{\varphi}(\omega) = 1_{[-\Omega,\Omega]}(\omega)$$

and

$$\mathcal{H} = \left\{ f : \int_{|\omega|>\Omega} \left| \hat{f}(\omega) \right|^2 d\omega = 0 \right\},$$

the set of functions whose spectrum is included in $[-\Omega, \Omega]$.

Again:

- We moved from a finite space of features to an infinite one: this makes the problem linearly separable
- We reduced the optimization problem solution to the estimation of a finite number of parameters

Class 4

- A zoo of kernels: linear, polynomial, gaussian, Fourier.
- Kernel regression.
- Kernel SVM

A zoo of kernels: linear, polynomial, gaussian

Name	Kernel Function (implicit dot product)	Feature Space (explicit dot product)
Linear	$K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z}$	Same as original input space
Polynomial (v1)	$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z})^d$	All polynomials **of** degree d
Polynomial (v2)	$K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + 1)^d$	All polynomials **up to** degree d
Gaussian	$K(\mathbf{x}, \mathbf{z}) = \exp(-\frac{\|\|\mathbf{x} - \mathbf{z}\|\|_2^2}{2\sigma^2})$	Infinite dimensional space
Hyperbolic Tangent (Sigmoid) Kernel	$K(\mathbf{x}, \mathbf{z}) = \tanh(\alpha\mathbf{x}^T\mathbf{z} + c)$	(With SVM, this is equivalent to a 2-layer neural network)

Polynomial kernel (1)

For $x, z \in [0,1]$ and $0 < \alpha < 1$

$$k(x,z) = \frac{1}{1-\alpha^2 xz}$$

Proof

$$\frac{1}{1-\alpha^2 xz} = \sum_{s=0}^{\infty} (\alpha^2 xz)^s = \Phi(x)^\top \Phi(z)$$

with

$$\Phi(x) = (1, \alpha x, \alpha^2 x^2, \alpha^3 x^3, \ldots)$$

Polynomial kernel (2)

- Inhomogeneous:

$$k(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T \mathbf{x}')^d$$

- E.g. $(d = 2)$

$$\begin{aligned} k(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}) &= (1 + x_1 x_1' + x_2 x_2')^2 \\ &= 1 + (x_1 x_1')^2 + (x_2 x_2')^2 + 2 x_1 x_1' + 2 x_2 x_2' + 2 x_1 x_1' x_2 x_2' \\ &= \underbrace{\begin{pmatrix} 1 & \sqrt{2} x_1 & \sqrt{2} x_2 & x_1^2 & \sqrt{2} x_1 x_2 & x_2^2 \end{pmatrix}^T}_{\Phi(\mathbf{x})^T} \underbrace{\begin{pmatrix} 1 \\ \sqrt{2} x_1' \\ \sqrt{2} x_2' \\ x_1'^2 \\ \sqrt{2} x_1' x_2' \\ x_2'^2 \end{pmatrix}}_{\Phi(\mathbf{x}')} \end{aligned}$$

Radial kernel features

Let us consider the RBF kernel again for points in $\mathbb{R}^2$. Then, the kernel can be written as

$$k(x, y) = \exp(-\|x - y\|^2) = \exp(-(x_1 - y_1)^2 - (x_2 - y_2)^2)$$

$$= \exp(-x_1^2 + 2x_1y_1 - y_1^2 - x_2^2 + 2x_2y_2 - y_2^2)$$

$$= \exp(-\|x\|^2) \exp(-\|y\|^2) \exp(2x^T y)$$

(assuming gamma = 1). Using the taylor series you can write this as,

$$k(x, y) = \exp(-\|x\|^2) \exp(-\|y\|^2) \sum_{n=0}^{\infty} \frac{(2x^T y)^n}{n!}$$

Now, if we were to come up with a feature map Φ just like we did for the polynomial kernel, you would realize that the feature map would map every point in our $\mathbb{R}^2$ to an infinite vector. Thus, RBF implicitly maps every point to an infinite dimensional space.

Intuition behind radial kernel

(Representer Theorem): *The optimal solution to any vector valued function learning problem of the form.*

$$f^* = \text{argmin}_{f \in \mathcal{H}_K} L(f(\mathbf{x}_1) \dots f(\mathbf{x}_l)) + \lambda \|f\|^2_{\mathcal{H}_K}$$

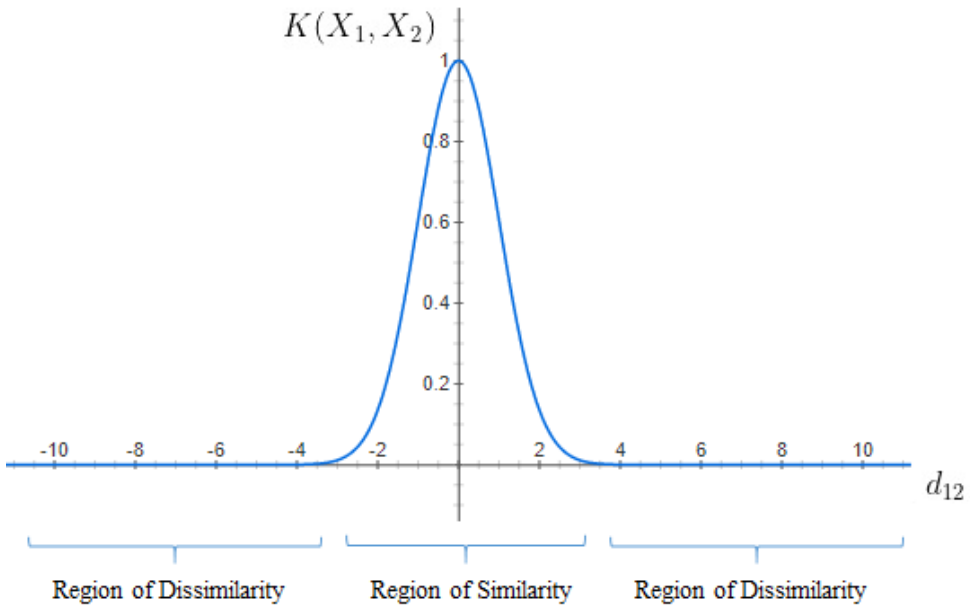
is a sum of matrix-vector products of the form

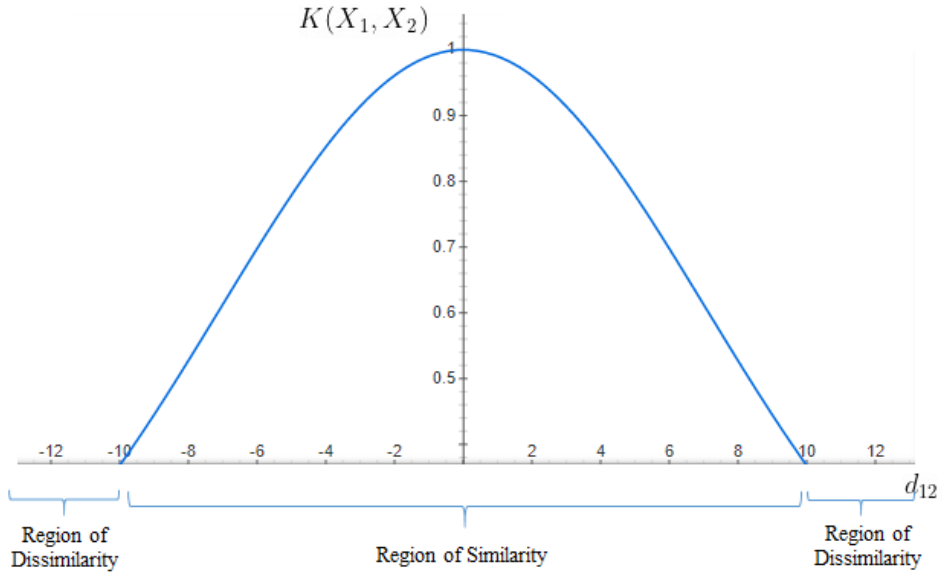
$$f^*(\mathbf{x}) = \sum_{i=1}^{l} K(\mathbf{x}, \mathbf{x}_i)\boldsymbol{\alpha}_i$$

where $\boldsymbol{\alpha}_i \in \mathbb{R}^n, \quad i = 1 \dots l$, *L is an arbitrary loss function (which can also be an indicator function encoding arbitrary constraints) and* $\lambda > 0$ *is a regularization parameter.*

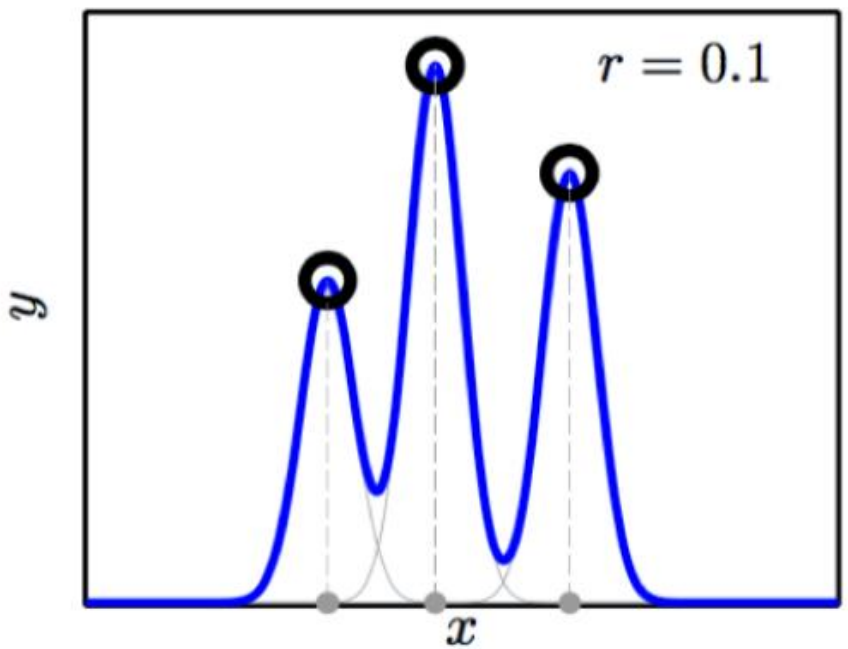
$$K(X_1, X_2) = e^{-\frac{\|X_1 - X_2\|_2^2}{r^2}}$$

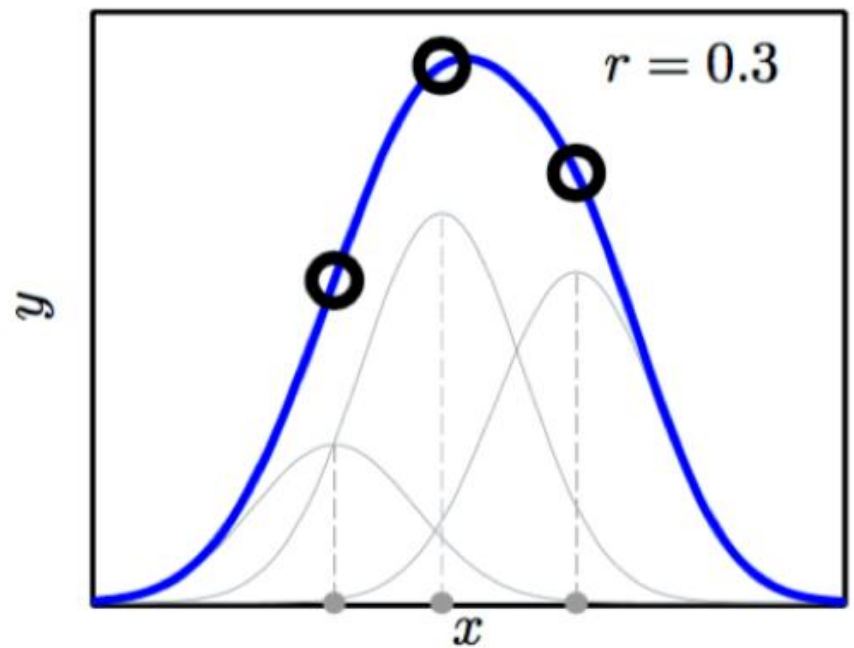
Gaussian kernel is a measure of data similarity





$$f(X) = \sum_i \alpha_i e^{-\frac{\|X - X_i\|_2^2}{r^2}}$$





Fourier kernels

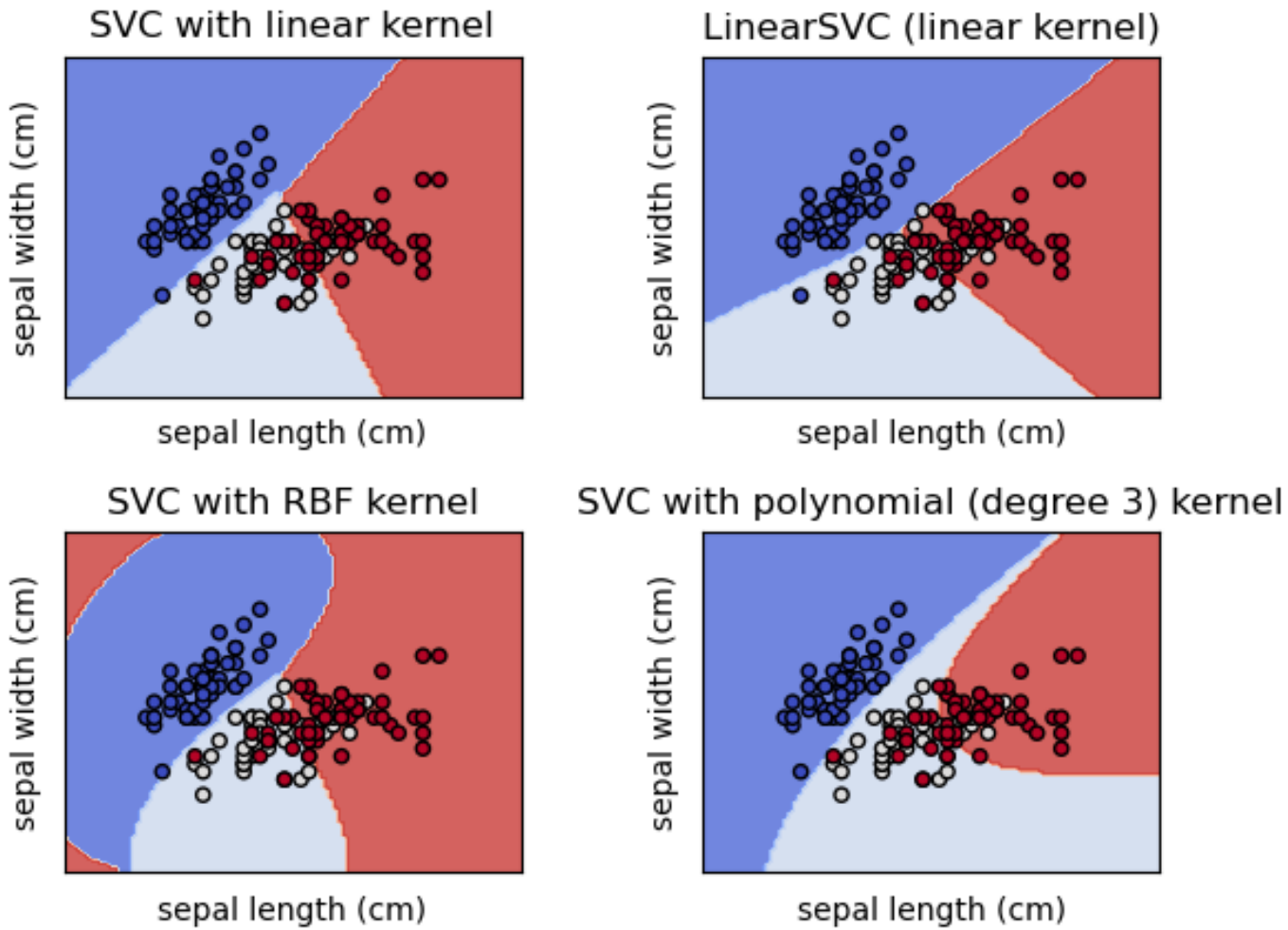
$$
\begin{aligned}
k(x,y) &= \int_{\mathbb{R}^p} \exp\left(i\omega^\top (x-y)\right) d\Lambda(\omega) \\
&= 2\int_{\mathbb{R}^p} \left\{\cos\left(\omega^\top x\right)\cos\left(\omega^\top y\right) + \sin\left(\omega^\top x\right)\sin\left(\omega^\top y\right)\right\} d\Lambda(\omega)
\end{aligned}
$$

- Sample m frequencies $\Omega = \{\omega_j\}_{j=1}^m \sim \Lambda$ and use a Monte Carlo estimator of the kernel function instead [Rahimi & Recht, 2007]:

$$
\begin{aligned}
\hat{k}(x,y) &= \frac{2}{m}\sum_{j=1}^{m} \left\{\cos\left(\omega_j^\top x\right)\cos\left(\omega_j^\top y\right) + \sin\left(\omega_j^\top x\right)\sin\left(\omega_j^\top y\right)\right\} \\
&= \langle \xi_\Omega(x), \xi_\Omega(y)\rangle_{\mathbb{R}^{2m}},
\end{aligned}
$$

with an explicit set of features $\xi_\Omega : x \mapsto \sqrt{\frac{2}{m}}\left[\cos\left(\omega_1^\top x\right), \sin\left(\omega_1^\top x\right), \ldots\right]^\top$.

Which kernel should one use?



Making kernels

Kernels can be :

- Summed together
  - On the same space $k(x, y) = k_1(x, y) + k_2(x, y)$
  - On the tensor space $k(\mathbf{x}, \mathbf{y}) = k_1(x_1, y_1) + k_2(x_2, y_2)$
- Multiplied together
  - On the same space $k(x, y) = k_1(x, y) \times k_2(x, y)$
  - On the tensor space $k(\mathbf{x}, \mathbf{y}) = k_1(x_1, y_1) \times k_2(x_2, y_2)$
- Composed with a function
  - $k(x, y) = k_1(f(x), f(y))$

All these operations will preserve the positive definiteness.

Example: sum of kernels is a kernel

- The sum of two kernels corresponds to the *concatenation* of their respective feature spaces:

$$
\begin{aligned}
(k_1 + k_2)(x, x') &= k_1(x, x') + k_2(x, x') \\
&= \Phi_1(x)^T \Phi_1(x') + \Phi_2(x)^T \Phi_2(x') \\
&= (\Phi_1(x)\ \Phi_2(x)) \begin{pmatrix} \Phi_1(x') \\ \Phi_2(x') \end{pmatrix}
\end{aligned}
$$

Summary: Kernelize a machine learning algorithm

(In practice) an algorithm can be kernelized in 2 steps:

1. Prove that the solution lies in the span of the training points (i.e. $\mathbf{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i$ for some α_i)
2. Rewrite the algorithm and the classifier so that all training or testing inputs $\mathbf{x}_i$ are only accessed in inner-products with other inputs, e.g. $\mathbf{x}_i^\top \mathbf{x}_j$.
3. Define a kernel function and substitute $\mathbf{k}(\mathbf{x}_i, \mathbf{x}_j)$ for $\mathbf{x}_i^\top \mathbf{x}_j$.

Let’s see a few examples

Kernel Ridge Regression (KRR)

- Let us now consider a RKHS $\mathcal{H}$, associated to a p.d. kernel K on $\mathcal{X}$.
- KRR is obtained by regularizing the MSE criterion by the RKHS norm:

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - f\left(\mathbf{x}_i\right)\right)^2 + \lambda \| f \|_{\mathcal{H}}^2$$

- *1st effect = prevent overfitting by penalizing non-smooth functions.*
- By the representer theorem, any solution of (2) can be expanded as

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K\left(\mathbf{x}_i, \mathbf{x}\right).$$

- *2nd effect = simplifying the solution.*

Solving KRR

- Let $\mathbf{y} = (y_1, \ldots, y_n)^\top \in \mathbb{R}^n$
- Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^\top \in \mathbb{R}^n$
- Let $\mathbf{K}$ be the $n \times n$ Gram matrix: $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
- We can then write:

$$\left(\hat{f}(\mathbf{x}_1), \ldots, \hat{f}(\mathbf{x}_n)\right)^\top = \mathbf{K}\boldsymbol{\alpha}$$

- The following holds as usual:

$$\| \hat{f} \|_{\mathcal{H}}^2 = \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha}$$

- The KRR problem (2) is therefore equivalent to:

$$\arg\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \frac{1}{n} (\mathbf{K}\boldsymbol{\alpha} - \mathbf{y})^\top (\mathbf{K}\boldsymbol{\alpha} - \mathbf{y}) + \lambda \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha}$$

Solving KRR

$$\arg\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \frac{1}{n}\left(\mathbf{K}\boldsymbol{\alpha}-\mathbf{y}\right)^\top\left(\mathbf{K}\boldsymbol{\alpha}-\mathbf{y}\right)+\lambda\boldsymbol{\alpha}^\top\mathbf{K}\boldsymbol{\alpha}$$

- This is a convex and differentiable function of $\boldsymbol{\alpha}$. Its minimum can therefore be found by setting the gradient in $\boldsymbol{\alpha}$ to zero:

$$\begin{aligned} 0 &= \frac{2}{n}\mathbf{K}\left(\mathbf{K}\boldsymbol{\alpha}-\mathbf{y}\right)+2\lambda\mathbf{K}\boldsymbol{\alpha} \\ &= \mathbf{K}\left[\left(\mathbf{K}+\lambda n\mathbf{I}\right)\boldsymbol{\alpha}-\mathbf{y}\right] \end{aligned}$$

- For $\lambda > 0$, $\mathbf{K}+\lambda n\mathbf{I}$ is invertible (because $\mathbf{K}$ is positive semidefinite) so one solution is to take:

$$\boldsymbol{\alpha} = \left(\mathbf{K}+\lambda n\mathbf{I}\right)^{-1}\mathbf{y}.$$

Unbalanced dataset

Weighted regression

- Given weights $W_1, \ldots, W_n \in \mathbb{R}$, a variant of ridge regression is to weight differently the error at different points:

$$\arg\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} W_i \left(y_i - f\left(\mathbf{x}_i\right)\right)^2 + \lambda \| f \|_{\mathcal{H}}^2$$

- By the representer theorem the solution is $f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K\left(\mathbf{x}_i, \mathbf{x}\right)$ where $\boldsymbol{\alpha}$ solves, with $\mathbf{W} = \operatorname{diag}\left(W_1, \ldots, W_n\right)$:

$$\arg\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \frac{1}{n} \left(\mathbf{K}\boldsymbol{\alpha} - \mathbf{y}\right)^\top \mathbf{W} \left(\mathbf{K}\boldsymbol{\alpha} - \mathbf{y}\right) + \lambda \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha}$$

Weighted regression

- Setting the gradient to zero gives

$$\begin{aligned} 0 &= \frac{2}{n}\left(\mathbf{KWK}\boldsymbol{\alpha} - \mathbf{KWy}\right) + 2\lambda\mathbf{K}\boldsymbol{\alpha} \\ &= \frac{2}{n}\mathbf{KW}^{\frac{1}{2}}\left[\left(\mathbf{W}^{\frac{1}{2}}\mathbf{KW}^{\frac{1}{2}} + n\lambda\mathbf{I}\right)\mathbf{W}^{-\frac{1}{2}}\boldsymbol{\alpha} - \mathbf{W}^{\frac{1}{2}}\mathbf{y}\right] \end{aligned}$$

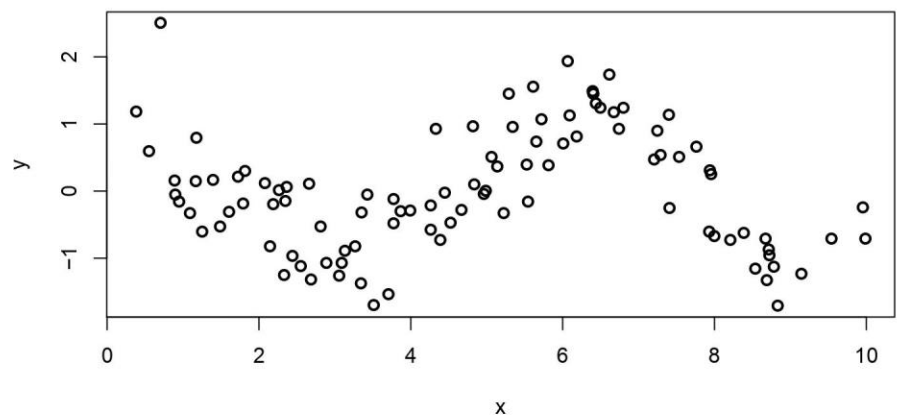
- A solution is therefore given by

$$\left(\mathbf{W}^{\frac{1}{2}}\mathbf{KW}^{\frac{1}{2}} + n\lambda\mathbf{I}\right)\mathbf{W}^{-\frac{1}{2}}\boldsymbol{\alpha} - \mathbf{W}^{\frac{1}{2}}\mathbf{y} = 0$$

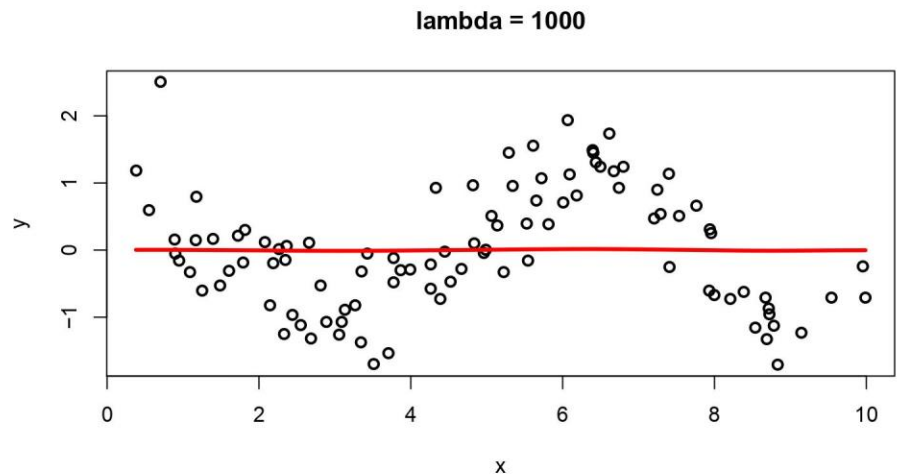
therefore

$$\boldsymbol{\alpha} = \mathbf{W}^{\frac{1}{2}}\left(\mathbf{W}^{\frac{1}{2}}\mathbf{KW}^{\frac{1}{2}} + n\lambda\mathbf{I}\right)^{-1}\mathbf{W}^{\frac{1}{2}}\mathbf{Y}$$

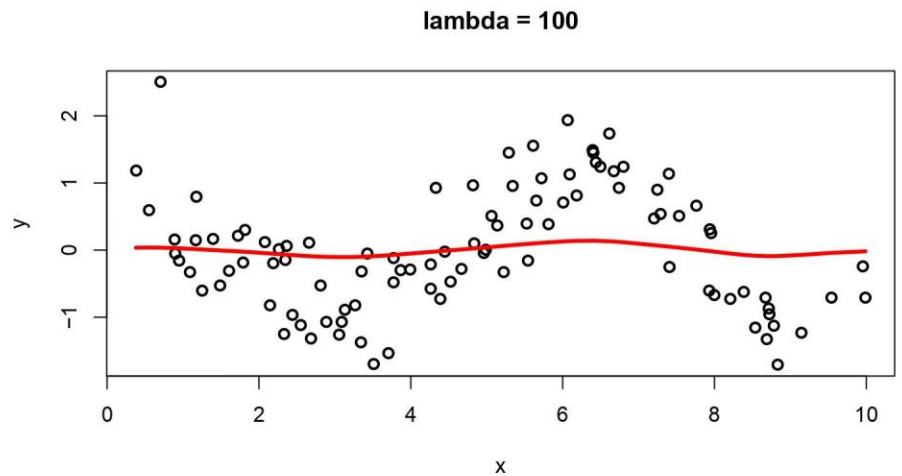
Example (KRR with Gaussian RBF kernel)



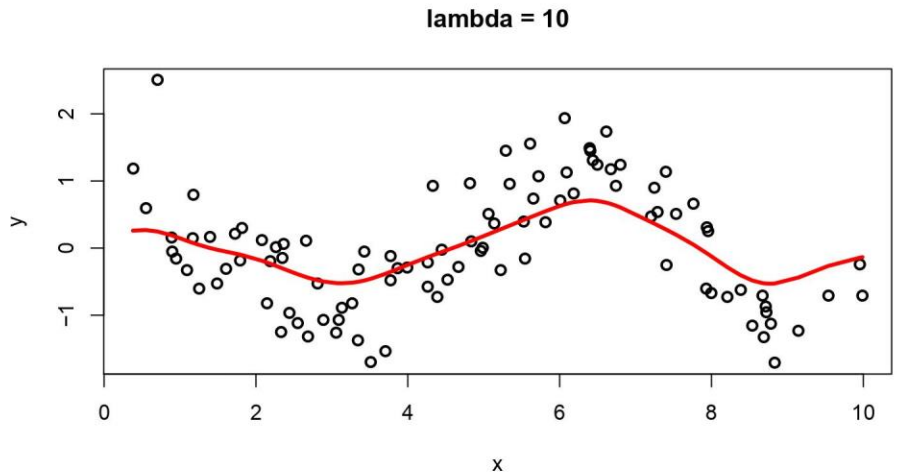
Example (KRR with Gaussian RBF kernel)



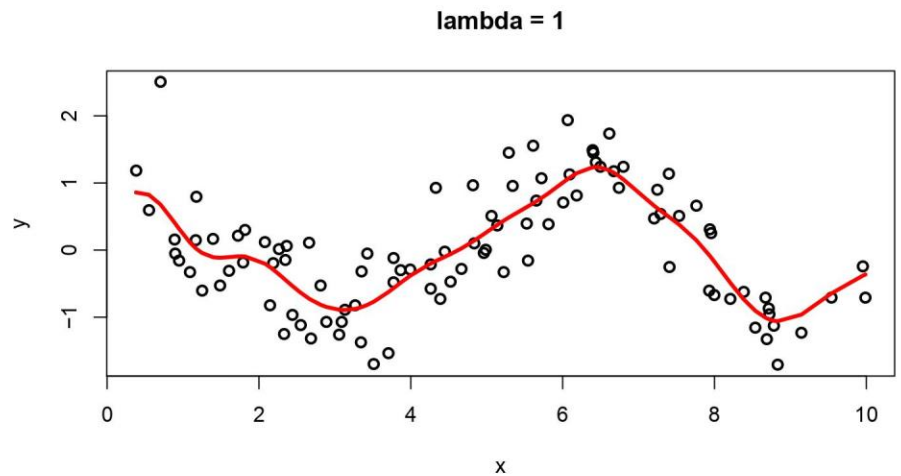
Example (KRR with Gaussian RBF kernel)



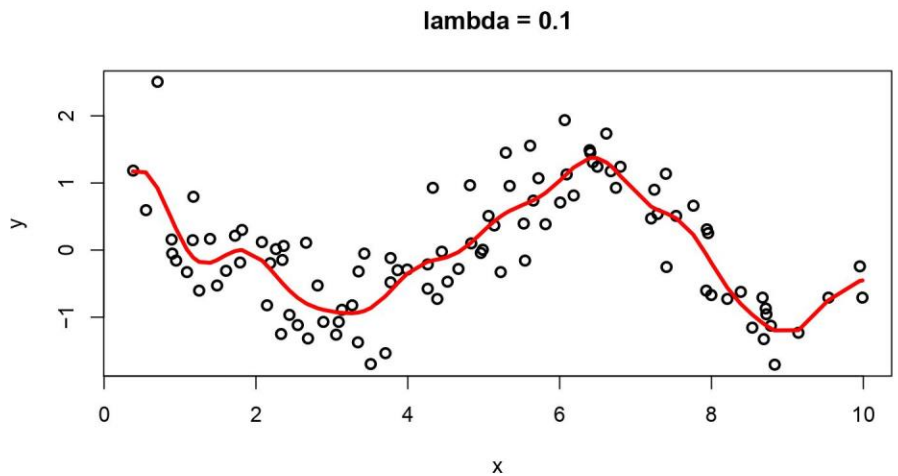
Example (KRR with Gaussian RBF kernel)



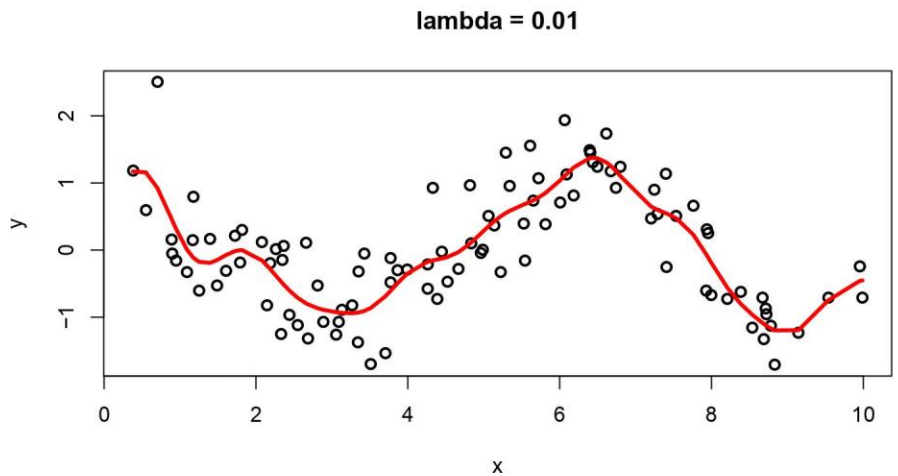
Example (KRR with Gaussian RBF kernel)



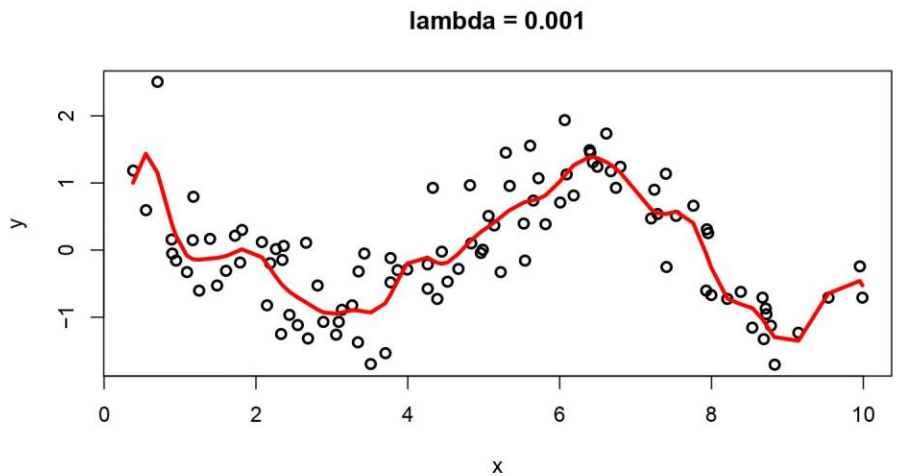
Example (KRR with Gaussian RBF kernel)



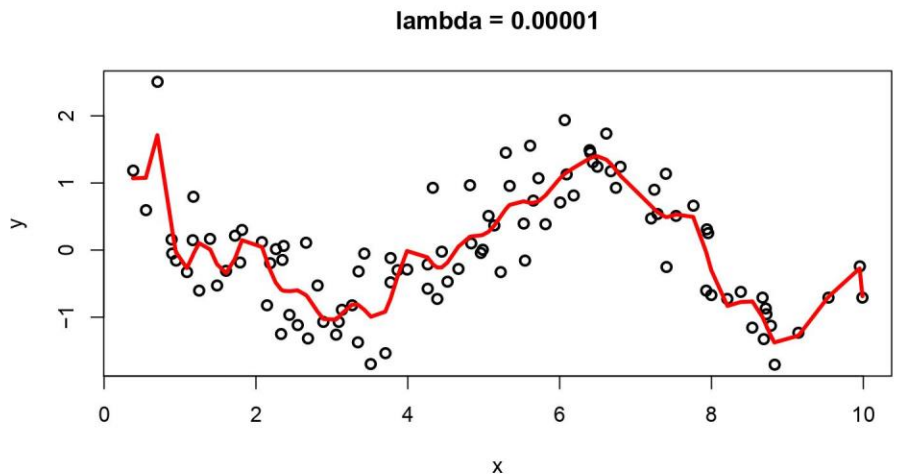
Example (KRR with Gaussian RBF kernel)



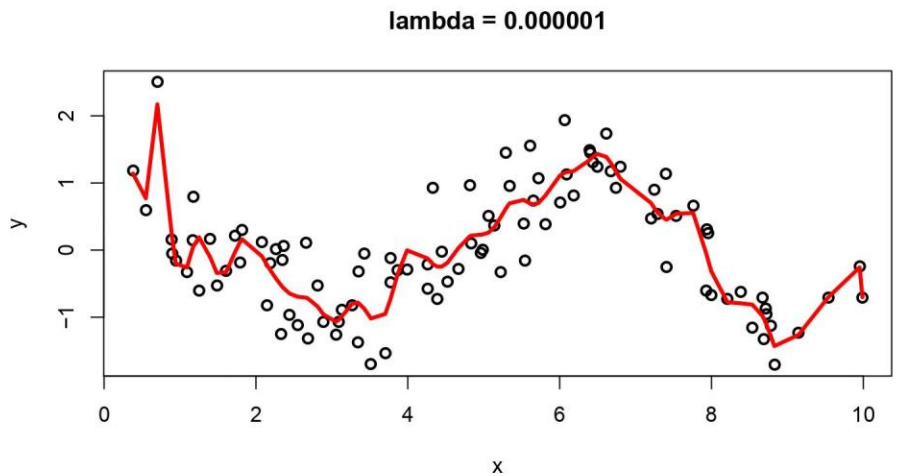
Example (KRR with Gaussian RBF kernel)



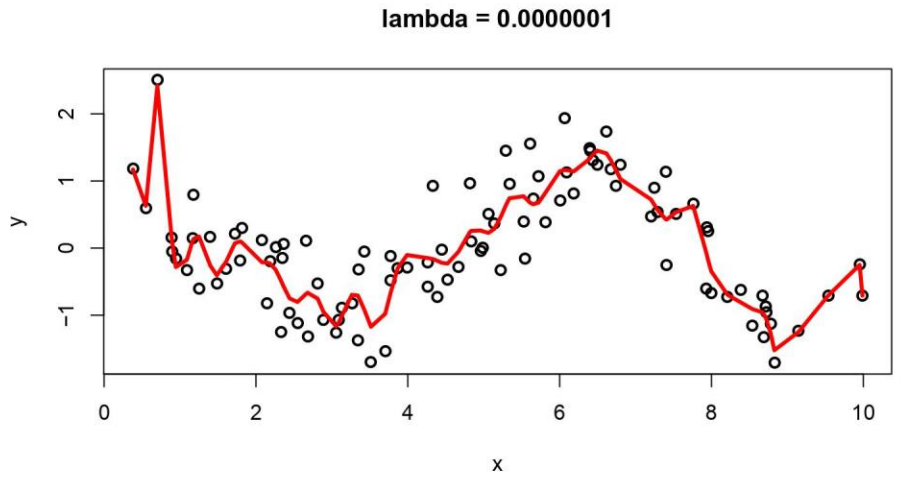
Example (KRR with Gaussian RBF kernel)



Example (KRR with Gaussian RBF kernel)



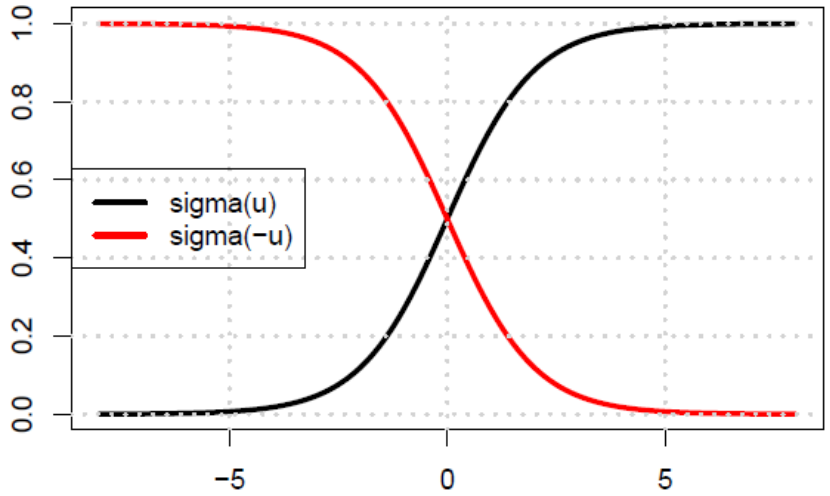
Example (KRR with Gaussian RBF kernel)



What can you conclude on the role of lambda?

Kernel SVM

$$\forall \mathbf{y} \in \{-1, 1\}, \quad p(y \mid f(\mathbf{x})) = \frac{1}{1 + e^{-yf(\mathbf{x})}} = \sigma(yf(\mathbf{x}))$$



- The logistic loss is the negative conditional likelihood:

$$\ell_{logistic}(f(\mathbf{x}), y) = -\ln p(y \mid f(\mathbf{x})) = \ln\left(1 + e^{-yf(\mathbf{x})}\right)$$

Solving Kernel SVM

- By the representer theorem, any solution of KLR can be expanded as

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x})$$

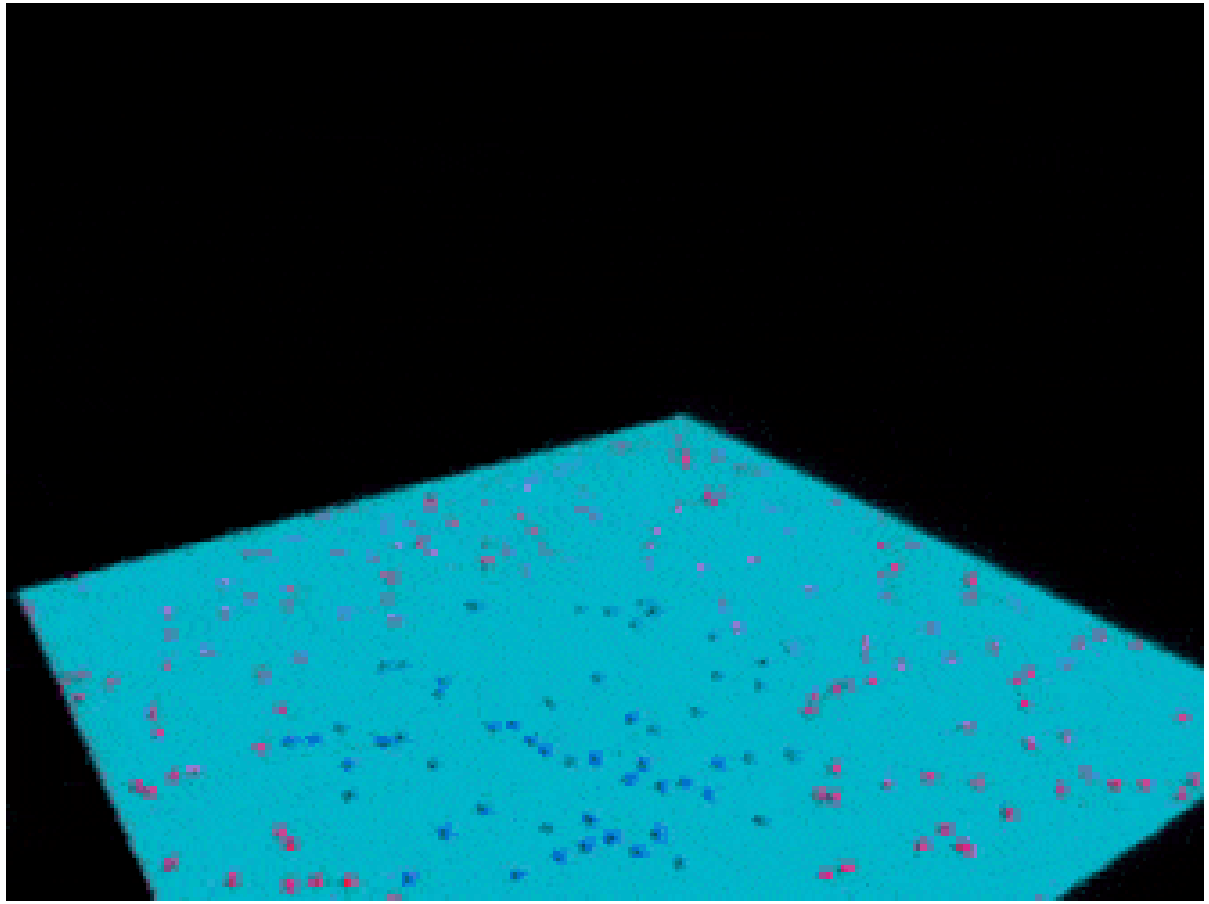
and as always we have:

$$\left(\hat{f}(\mathbf{x}_1), \ldots, \hat{f}(\mathbf{x}_n)\right)^\top = \mathbf{K}\boldsymbol{\alpha} \quad \text{and} \quad \| \hat{f} \|_{\mathcal{H}}^2 = \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha}$$

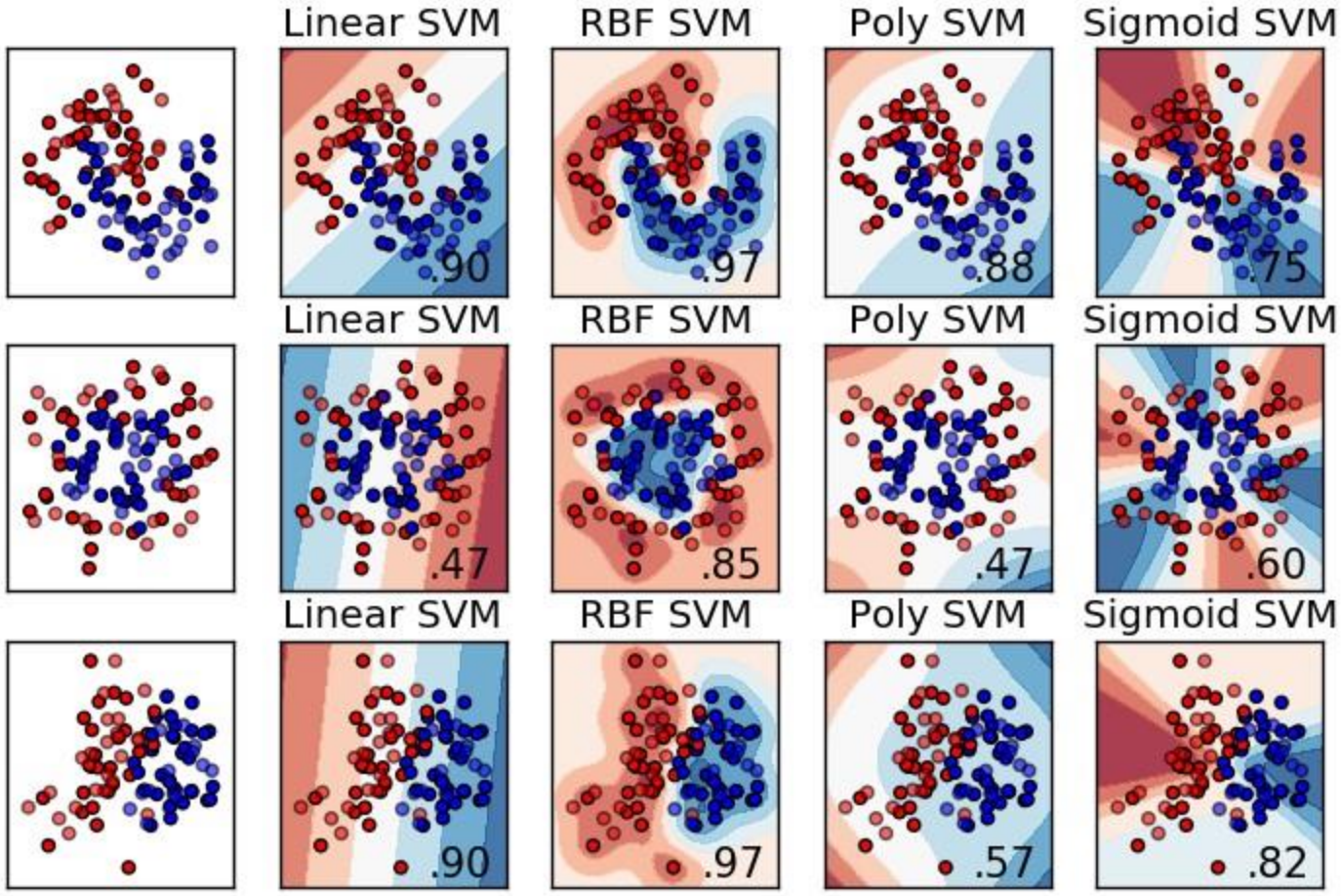
- To find $\boldsymbol{\alpha}$ we therefore need to solve:

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} \ln\left(1 + e^{-y_i [\mathbf{K}\boldsymbol{\alpha}]_i}\right) + \frac{\lambda}{2} \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha}$$

Kernel SVM

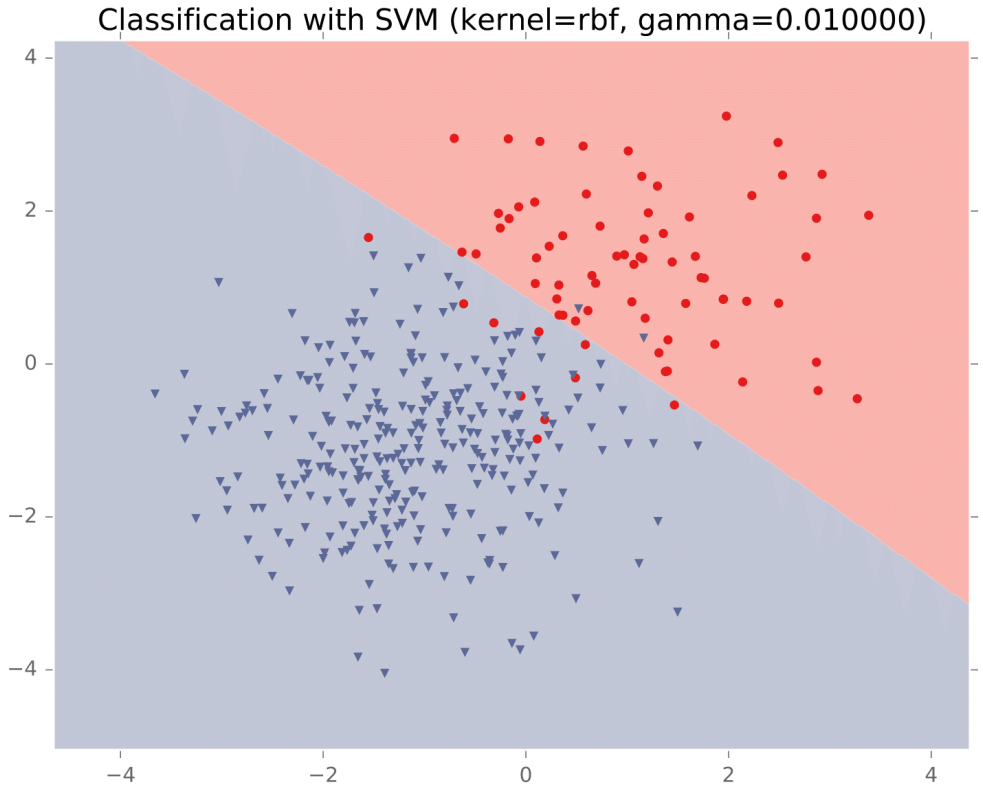


Kernel SVM



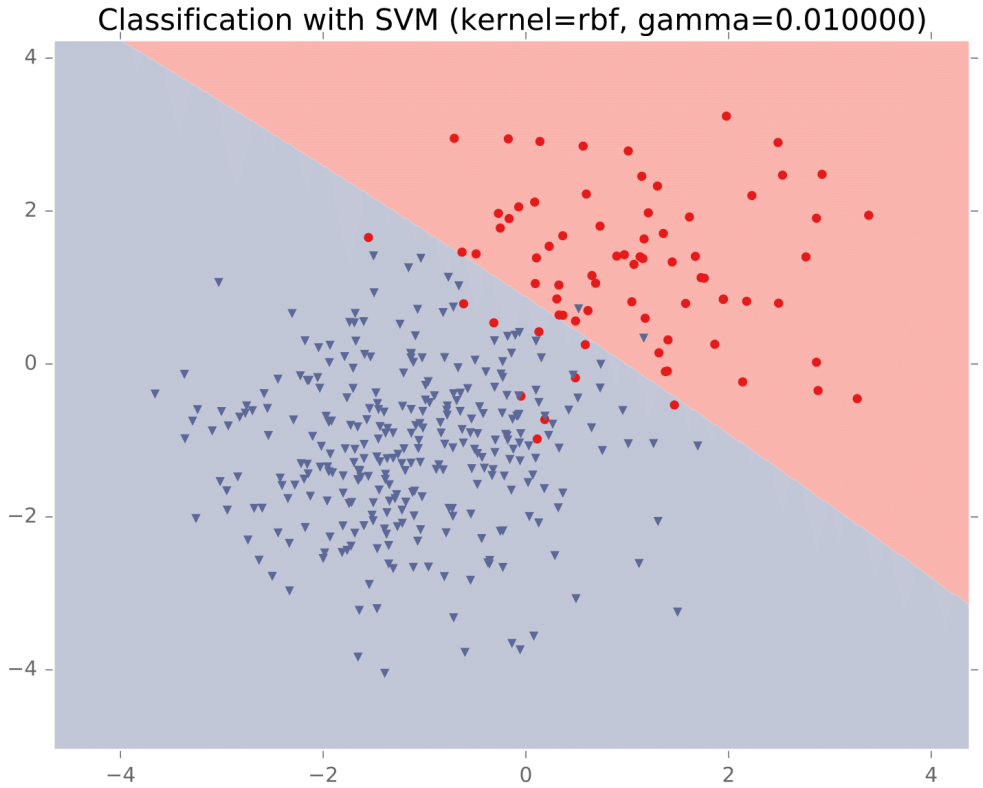
Different separators depending on the kernel choice!!!!

RBF Kernel Example



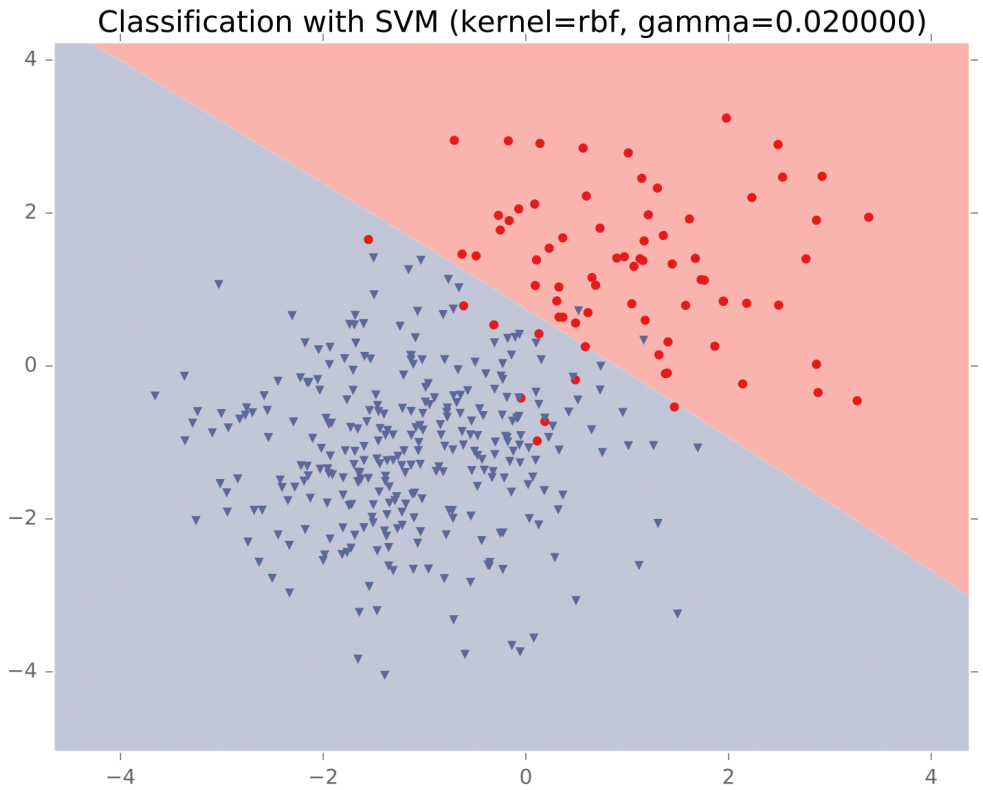
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



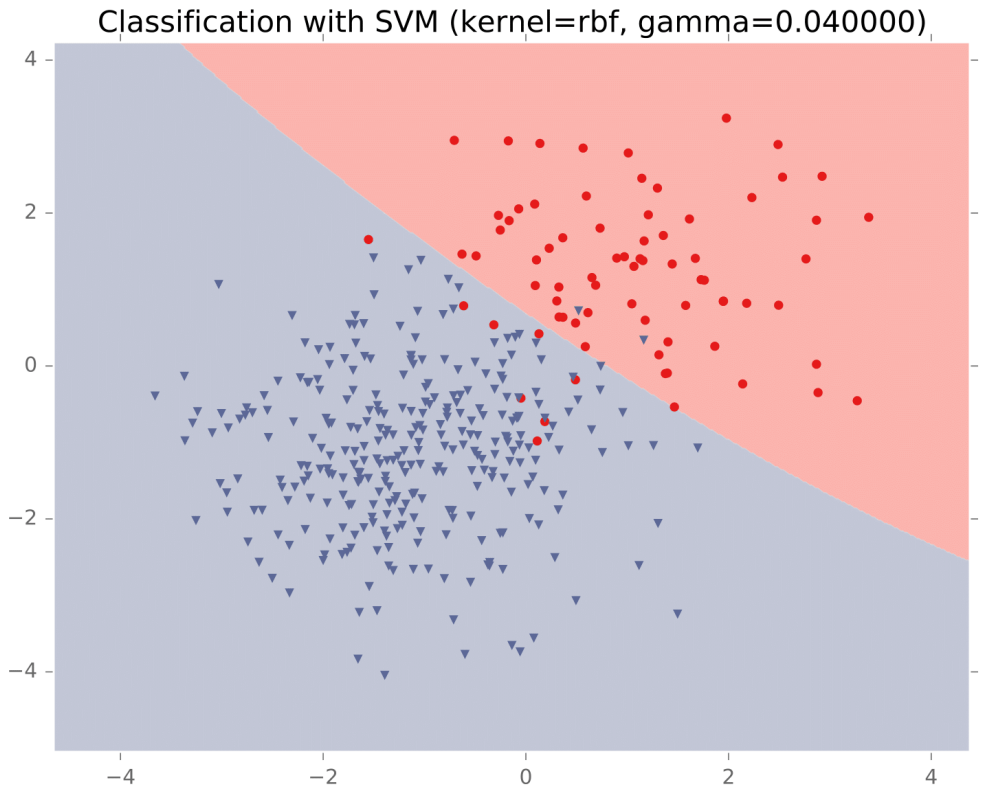
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



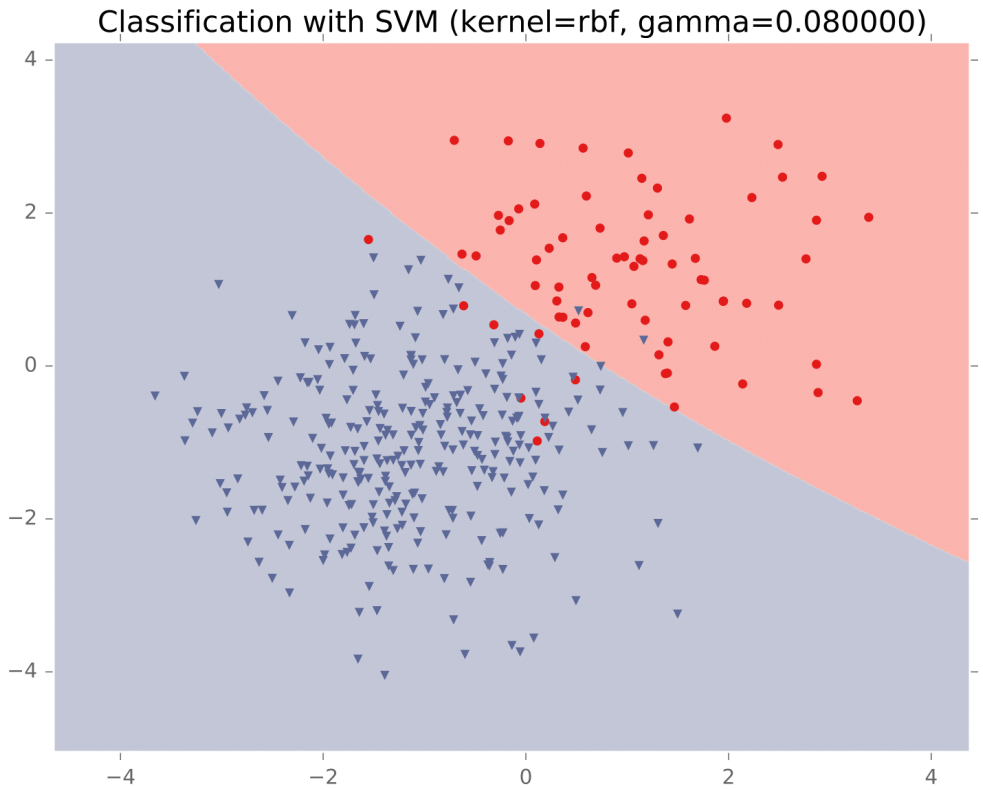
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



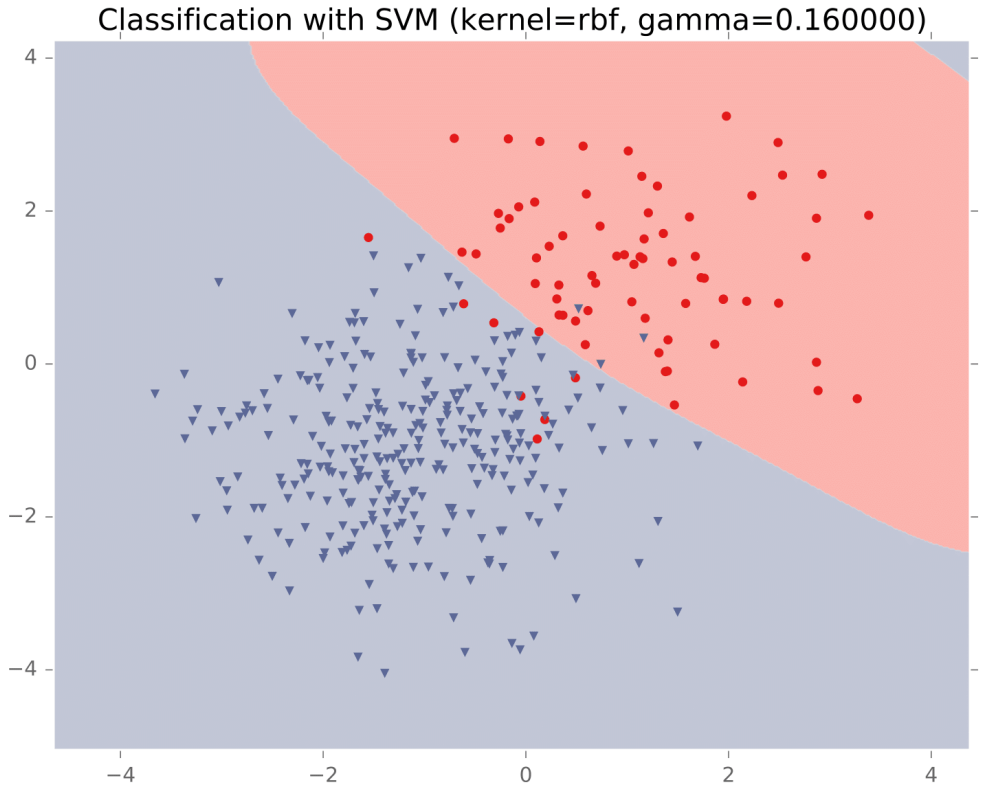
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



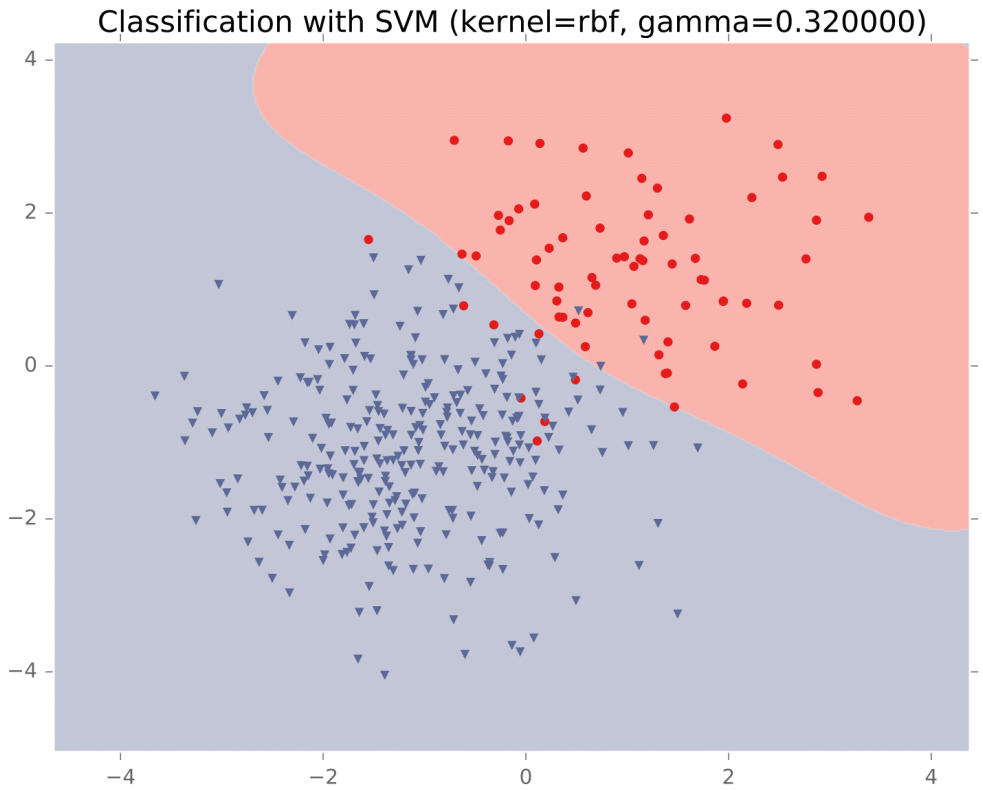
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



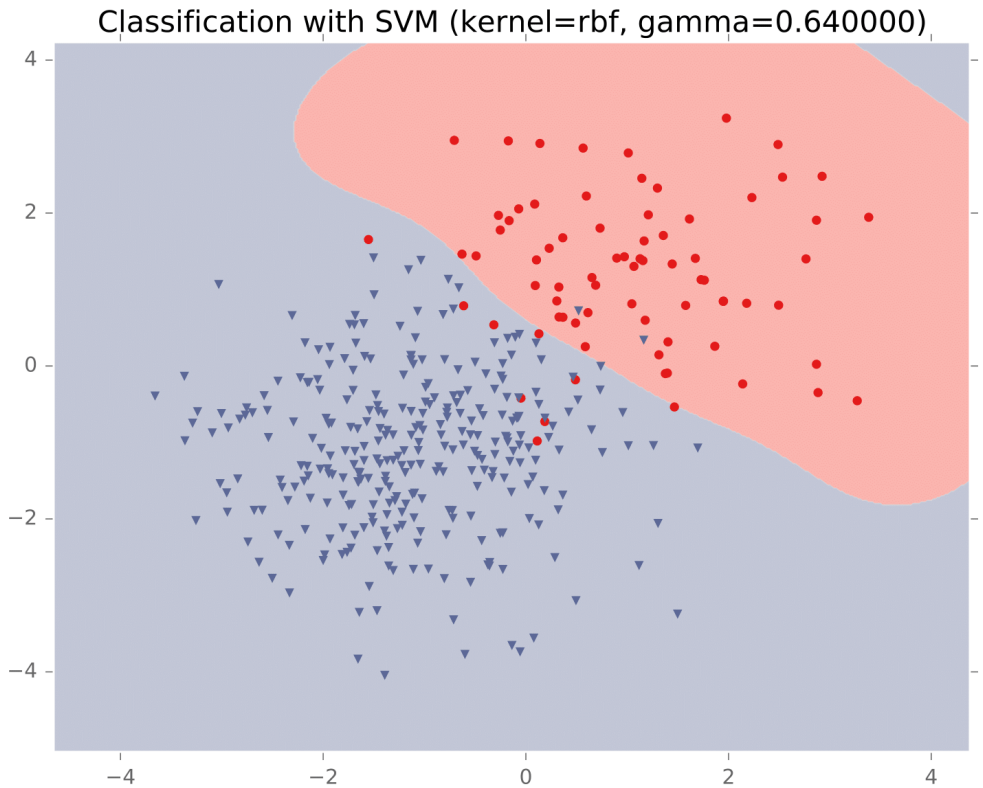
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



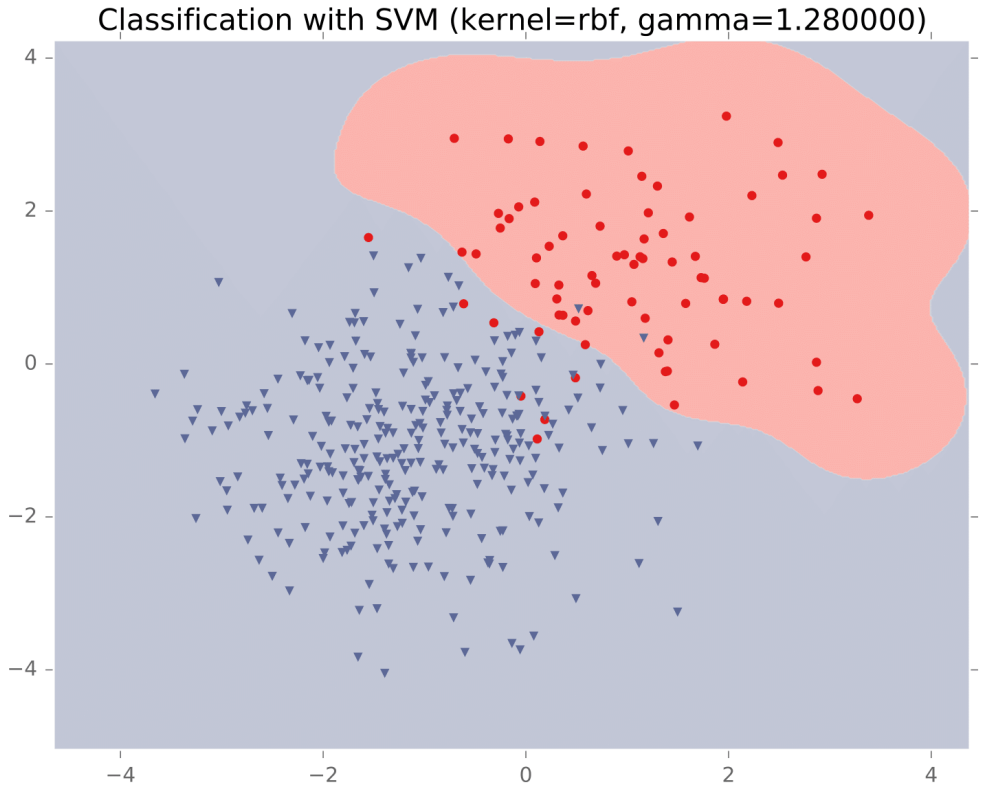
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



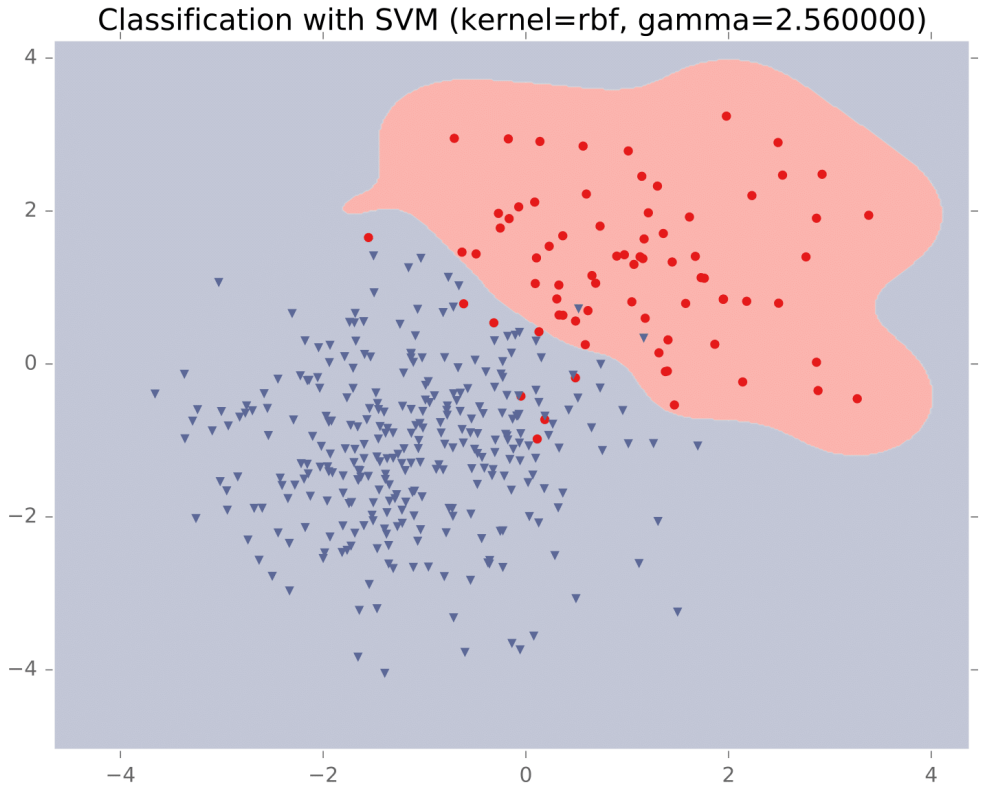
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



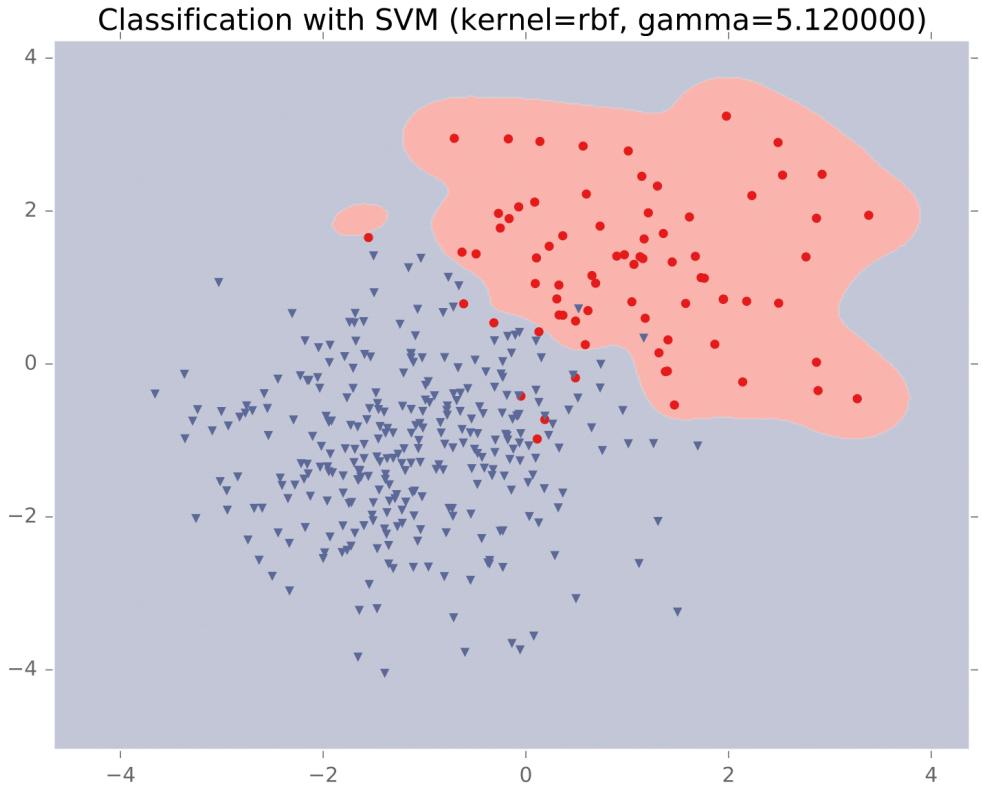
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



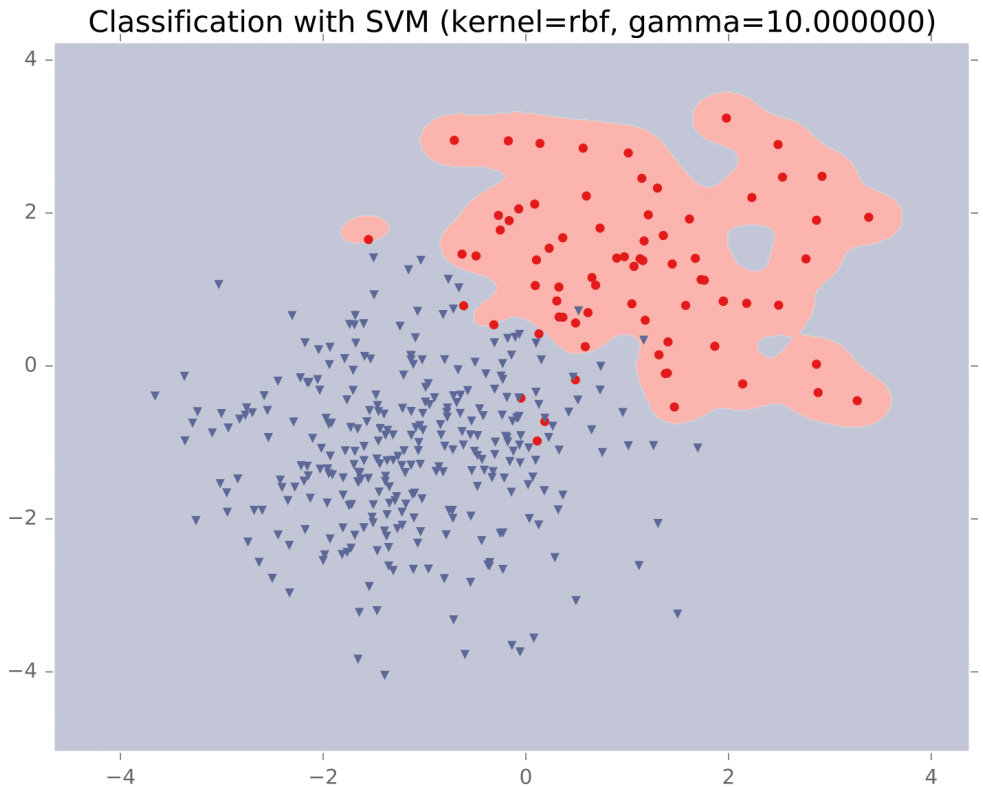
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example



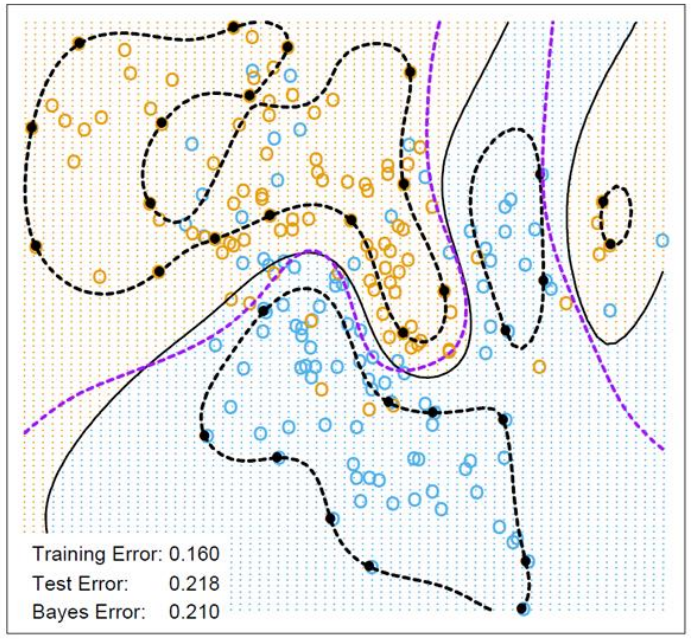
RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

RBF Kernel Example

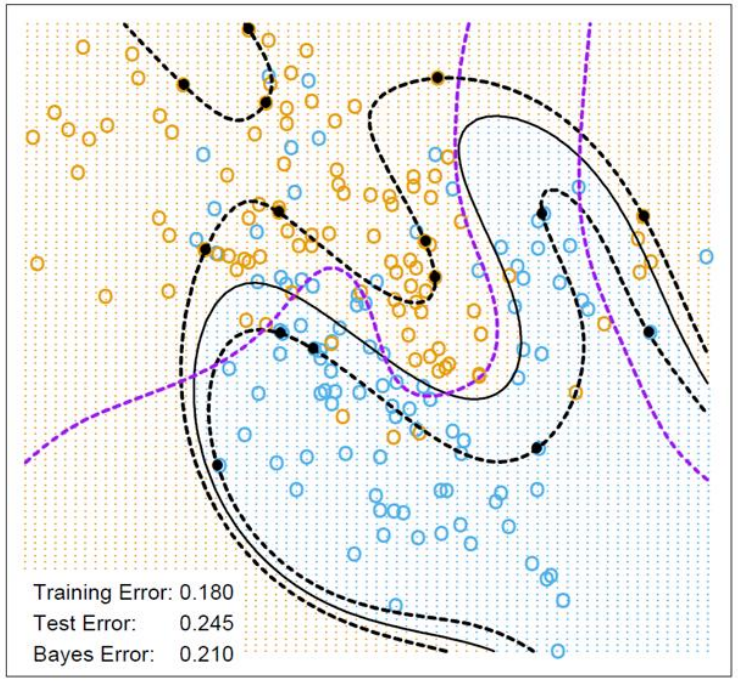


RBF Kernel: $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\gamma||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||_2^2)$

Gaussian Kernel SVM

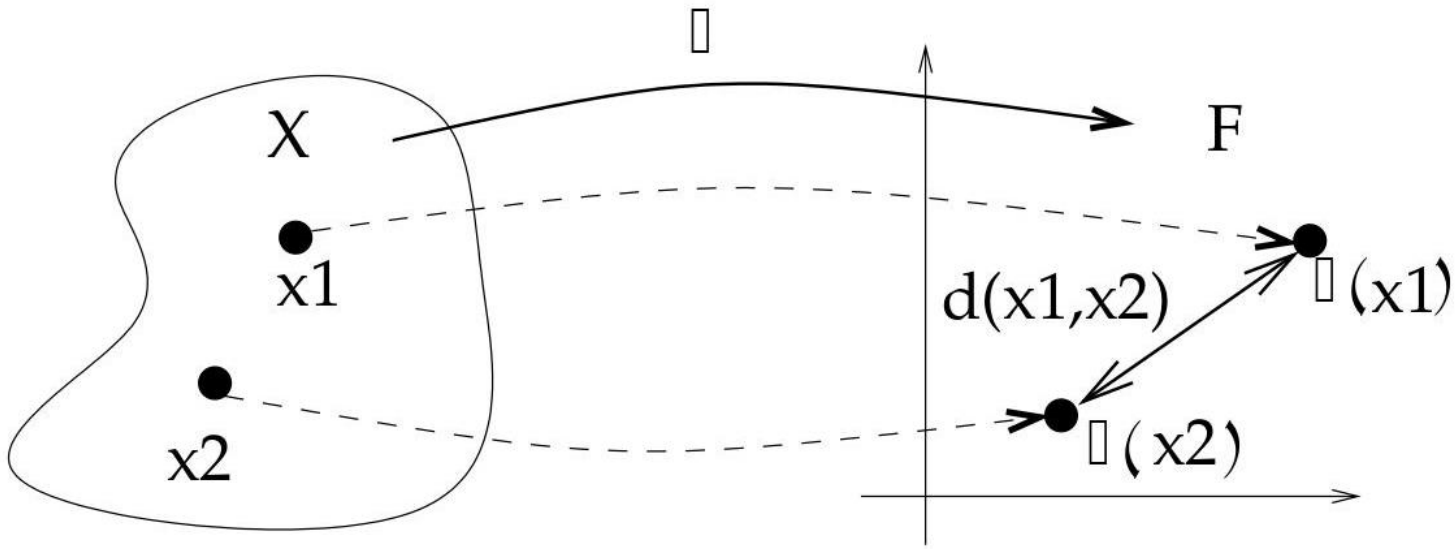


Degree-4 Polynomial Kernel SVM



Importance of choosing the kernel: how does the kernel choice influence the notion of distance?

Computing distances in feature space



$$d_K(\mathbf{x}_1, \mathbf{x}_2)^2 = \| \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2) \|^2_{\mathcal{H}}$$
$$= \langle \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2), \Phi(\mathbf{x}_1) - \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}}$$
$$= \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_1) \rangle_{\mathcal{H}} + \langle \Phi(\mathbf{x}_2), \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}} - 2 \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2) \rangle_{\mathcal{H}}$$

$$d_K(\mathbf{x}_1, \mathbf{x}_2)^2 = K(\mathbf{x}_1, \mathbf{x}_1) + K(\mathbf{x}_2, \mathbf{x}_2) - 2K(\mathbf{x}_1, \mathbf{x}_2)$$

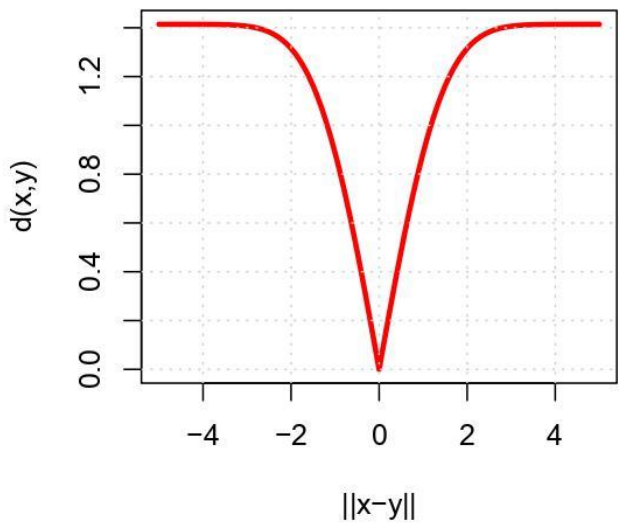
Distance for the Gaussian kernel

- The Gaussian kernel with bandwidth σ on $\mathbb{R}^d$ is:

$$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{\| \mathbf{x}-\mathbf{y} \|^2}{2\sigma^2}},$$

- $K(\mathbf{x}, \mathbf{x}) = 1 = \| \Phi(\mathbf{x}) \|^2_{\mathcal{H}}$, so all points are on the unit sphere in the feature space.
- The distance between the images of two points $\mathbf{x}$ and $\mathbf{y}$ in the feature space is given by:

$$d_K(\mathbf{x}, \mathbf{y}) = \sqrt{2\left[1 - e^{-\frac{\| \mathbf{x}-\mathbf{y} \|^2}{2\sigma^2}}\right]}$$



Distance set-point

Problem

- Let $\mathcal{S} = (\mathbf{x}_1, \cdots, \mathbf{x}_n)$ be a finite set of points in $\mathcal{X}$.
- How to define and compute the similarity between any point $\mathbf{x}$ in $\mathcal{X}$ and the set $\mathcal{S}$?

Where do you use this quantity?

Distance set-point

Problem

- Let $\mathcal{S} = (\mathbf{x}_1, \cdots, \mathbf{x}_n)$ be a finite set of points in $\mathcal{X}$.
- How to define and compute the similarity between any point $\mathbf{x}$ in $\mathcal{X}$ and the set $\mathcal{S}$?

Where do you use this quantity?

e.g. K-means

Distance set-point

Problem

- Let $\mathcal{S} = (\mathbf{x}_1, \cdots, \mathbf{x}_n)$ be a finite set of points in $\mathcal{X}$.
- How to define and compute the similarity between any point $\mathbf{x}$ in $\mathcal{X}$ and the set $\mathcal{S}$?

A solution:

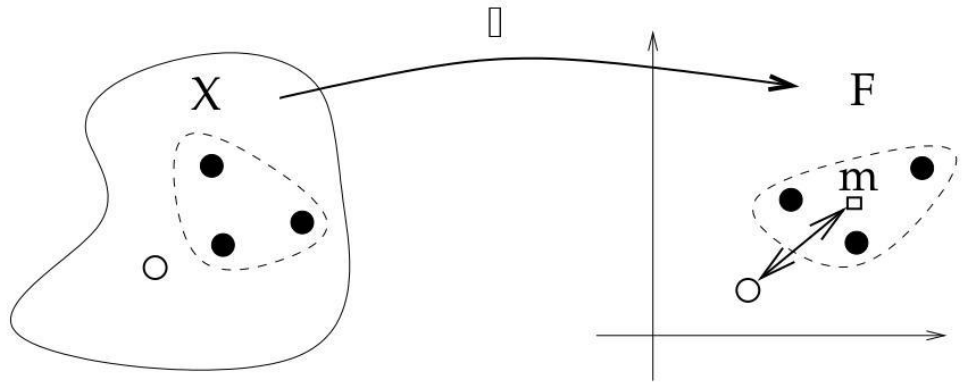
- Map all points to the feature space.
- Summarize $\mathcal{S}$ by the barycenter of the points:

$$\boldsymbol{\mu} := \frac{1}{n} \sum_{i=1}^{n} \Phi(\mathbf{x}_i) \ .$$

- Define the distance between $\mathbf{x}$ and $\mathcal{S}$ by:

$$d_K(\mathbf{x}, \mathcal{S}) := \| \Phi(\mathbf{x}) - \boldsymbol{\mu} \|_{\mathcal{H}} \ .$$

Computation



$$d_K(\mathbf{x}, \mathcal{S}) = \left\| \Phi(\mathbf{x}) - \frac{1}{n}\sum_{i=1}^{n} \Phi(\mathbf{x}_i) \right\|_{\mathcal{H}}$$

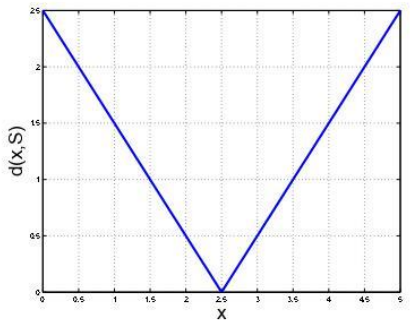
$$= \sqrt{K(\mathbf{x},\mathbf{x}) - \frac{2}{n}\sum_{i=1}^{n} K(\mathbf{x},\mathbf{x}_i) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} K(\mathbf{x}_i,\mathbf{x}_j)}.$$

Remark

The barycentre $\boldsymbol{\mu}$ only exists in the feature space in general: it does not necessarily have a pre-image $\mathbf{x}_{\boldsymbol{\mu}}$ such that $\Phi(\mathbf{x}_{\boldsymbol{\mu}}) = \boldsymbol{\mu}$.

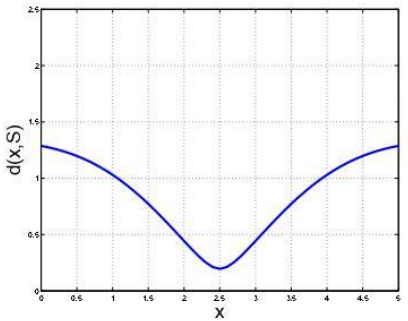
1D illustration

- $\mathcal{S} = \{2, 3\}$
- Plot $f(x) = d(x, \mathcal{S})$



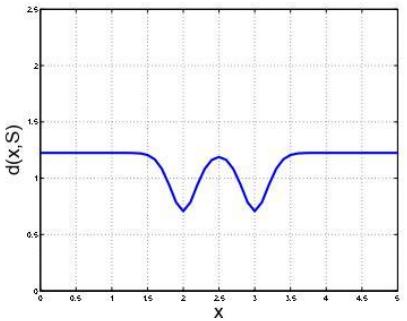
$K(x, y) = xy.$

(linear)



$K(x, y) = e^{-\frac{(x-y)^2}{2\sigma^2}}.$

with $\sigma = 1.$

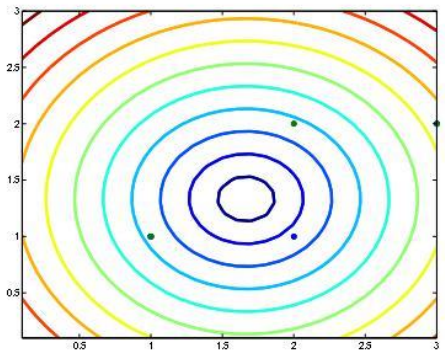


$K(x, y) = e^{-\frac{(x-y)^2}{2\sigma^2}}.$

with $\sigma = 0.2.$

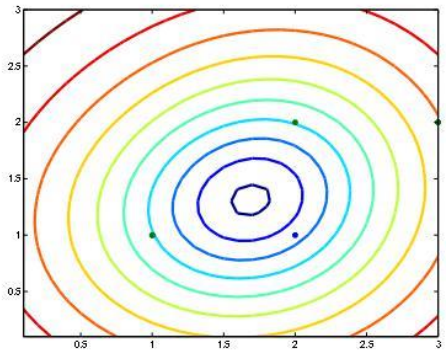
2D illustration

- $\mathcal{S} = \{(1,1)', (1,2)', (2,2)'\}$
- Plot $f(\mathbf{x}) = d(\mathbf{x}, \mathcal{S})$



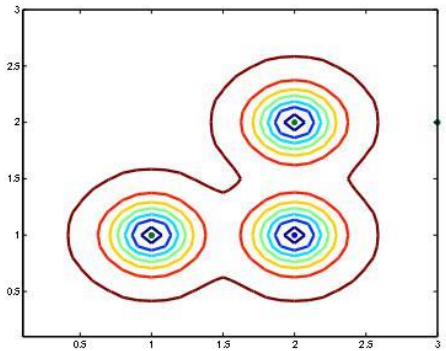
$K(\mathbf{x}, \mathbf{y}) = \mathbf{x}\mathbf{y}.$

(linear)



$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{(\mathbf{x}-\mathbf{y})^2}{2\sigma^2}}.$

with $\sigma = 1$.

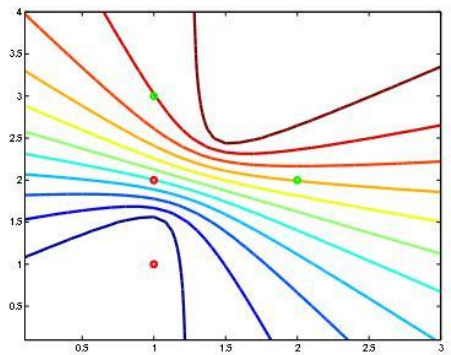


$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{(\mathbf{x}-\mathbf{y})^2}{2\sigma^2}}.$

with $\sigma = 0.2$.

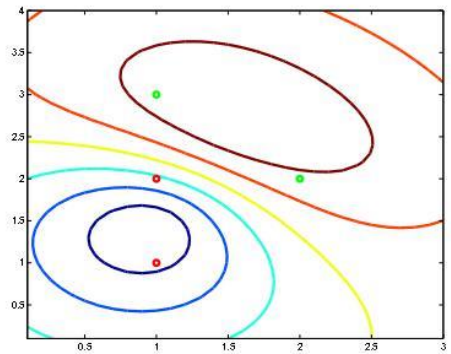
Basic application in discrimination

- $\mathcal{S}_1 = \{(1,1)', (1,2)'\}$ and $\mathcal{S}_2 = \{(1,3)', (2,2)'\}$
- Plot $f(\mathbf{x}) = d(\mathbf{x}, \mathcal{S}_1)^2 - d(\mathbf{x}, \mathcal{S}_2)^2$



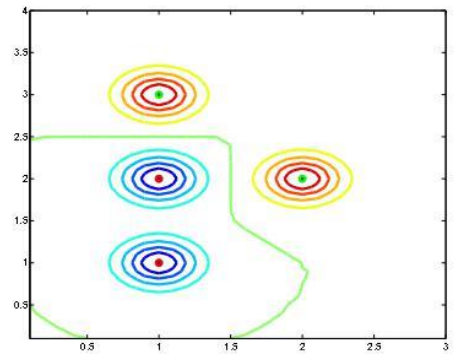
$K(\mathbf{x}, \mathbf{y}) = \mathbf{xy}.$

(linear)



$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{(\mathbf{x}-\mathbf{y})^2}{2\sigma^2}}.$

with $\sigma = 1$.



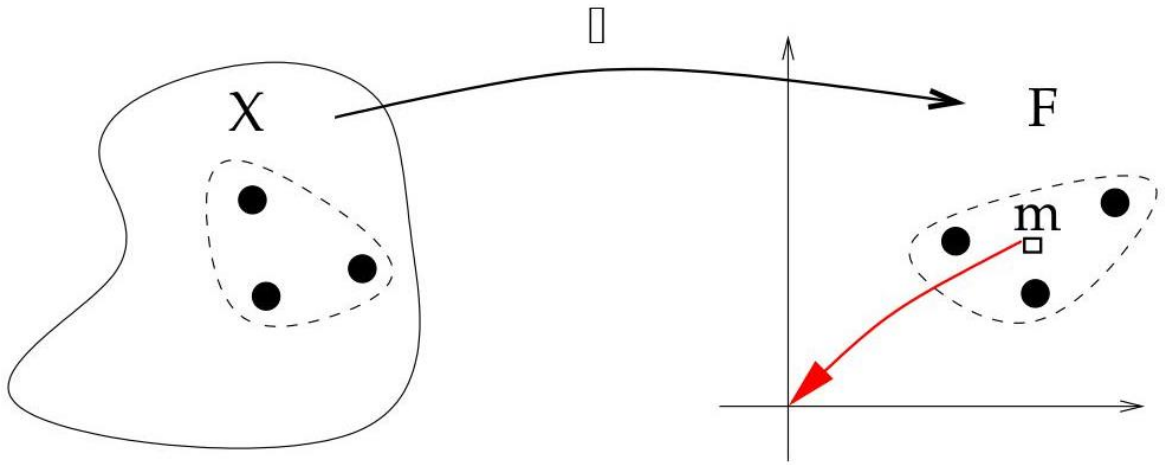
$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{(\mathbf{x}-\mathbf{y})^2}{2\sigma^2}}.$

with $\sigma = 0.2$.

Centering a feature in data space

Problem

- Let $\mathcal{S} = (\mathbf{x}_1, \cdots, \mathbf{x}_n)$ be a finite set of points in $\mathcal{X}$ endowed with a p.d. kernel K. Let $\mathbf{K}$ be their $n \times n$ Gram matrix: $[\mathbf{K}]_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$.
- Let $\boldsymbol{\mu} = 1/n \sum_{i=1}^{n} \Phi(\mathbf{x}_i)$ their barycenter, and $\mathbf{u}_i = \Phi(\mathbf{x}_i) - \boldsymbol{\mu}$ for $i = 1, \ldots, n$ be centered data in $\mathcal{H}$.
- How to compute the centered Gram matrix $[\mathbf{K}^c]_{i,j} = \langle \mathbf{u}_i, \mathbf{u}_j \rangle_{\mathcal{H}}$?



Class 5: more kernels applications

- Kernel PCA
- Kernel K-means

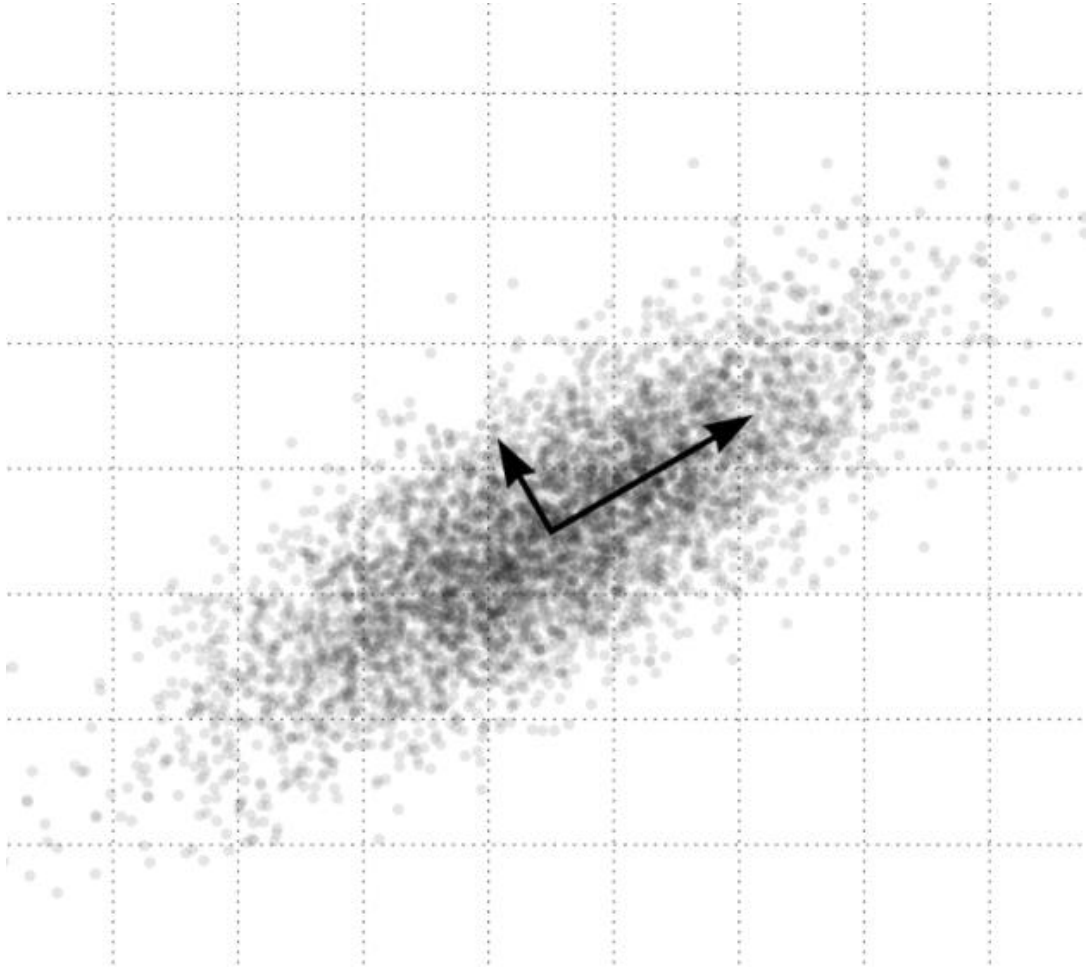
Kernel PCA

$$\mu = \frac{1}{N}\sum_i \phi(x_i) = 0$$

$$C_\phi = \frac{1}{N}\sum_i \phi(x_i)\phi^T(x_i)$$

$$C_\phi v = \lambda v$$

What is the problem of PCA?



How do you solve it?

What if?

Kernel PCA

- Calculate the covariance matrix in the feats space:

$$C_F = \frac{1}{N} \sum_{i}^{N} \phi(x_i) \phi^T(x_i)$$

- We want to solve $C_F v = \lambda v$

Kernel PCA

Eigenvectors can be expressed as a linear combination of the features i.e.

$$v = \sum_{i=1}^{N} \alpha_i \phi(x_i)$$

- We have

$$C_F v = \lambda v = \frac{1}{N} \sum_{i}^{N} \phi(x_i) \phi^T(x_i) v$$

- Thus

$$v = \frac{1}{N\lambda} \sum_{i}^{N} \phi(x_i) \phi^T(x_i) v = \sum_{i}^{N} \phi(x_i) \frac{\beta_i}{N\lambda} = \sum_{i=1}^{N} \alpha_i \phi(x_i)$$

Kernel PCA

Let's multiply both sides by $\phi(x_i)$

- Expanding

$$\lambda\phi(x_k) = \phi(x_k)C_F$$

$$\lambda\sum_i \alpha_i\phi(x_k)\phi(x_i) = \frac{1}{N}\sum_i^N \alpha_i(\phi(x_k)\sum_j^N \phi(x_j)(\phi^T(x_j)\phi(x_i))$$

- Define $K_{ij} = \phi^T(x_j)\phi(x_i)$ we have

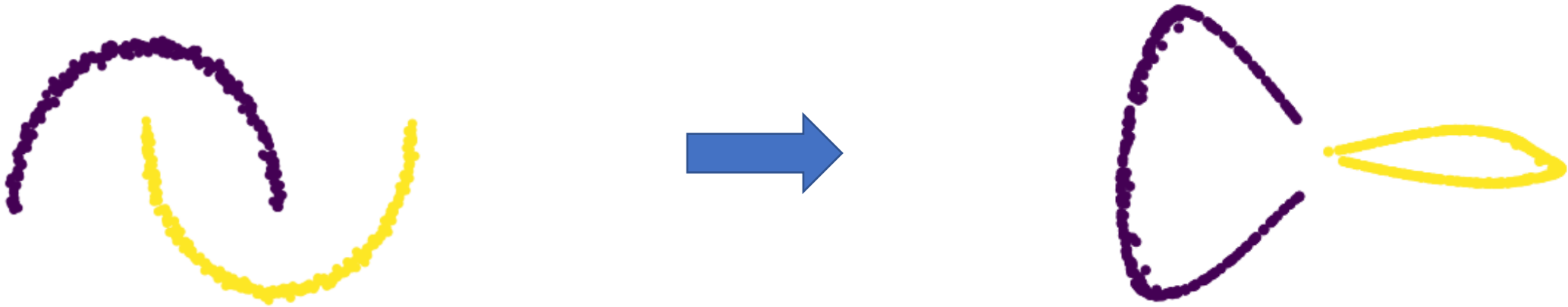
$$N\lambda K\alpha = K^2\alpha \quad \rightarrow \quad N\lambda\alpha = K\alpha$$

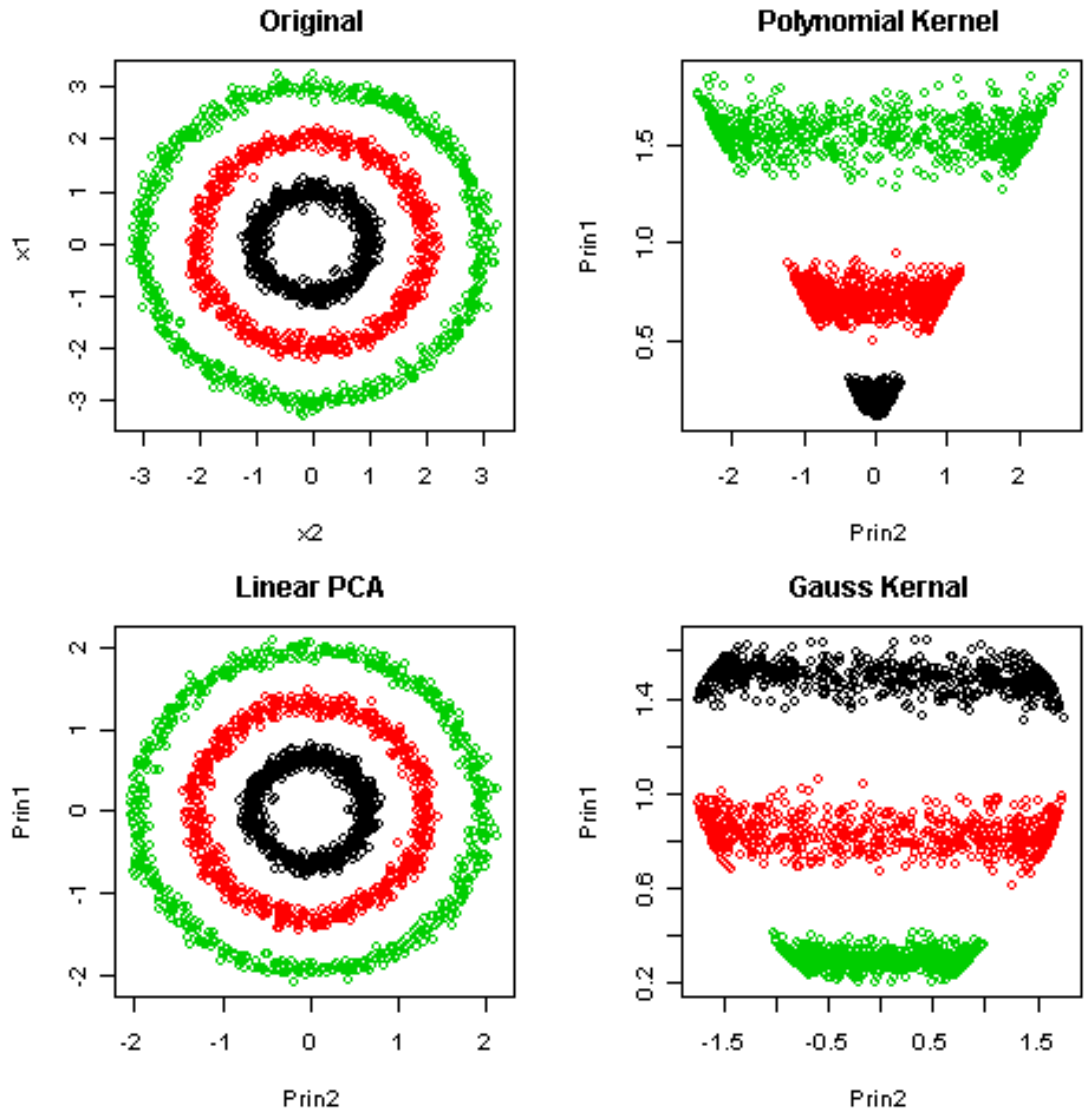
Kernel PCA, normalization

$$\hat{\phi}(x_i) = \phi(x_i) - \frac{1}{N}\sum_{j=1}^{N}\phi(x_k)$$

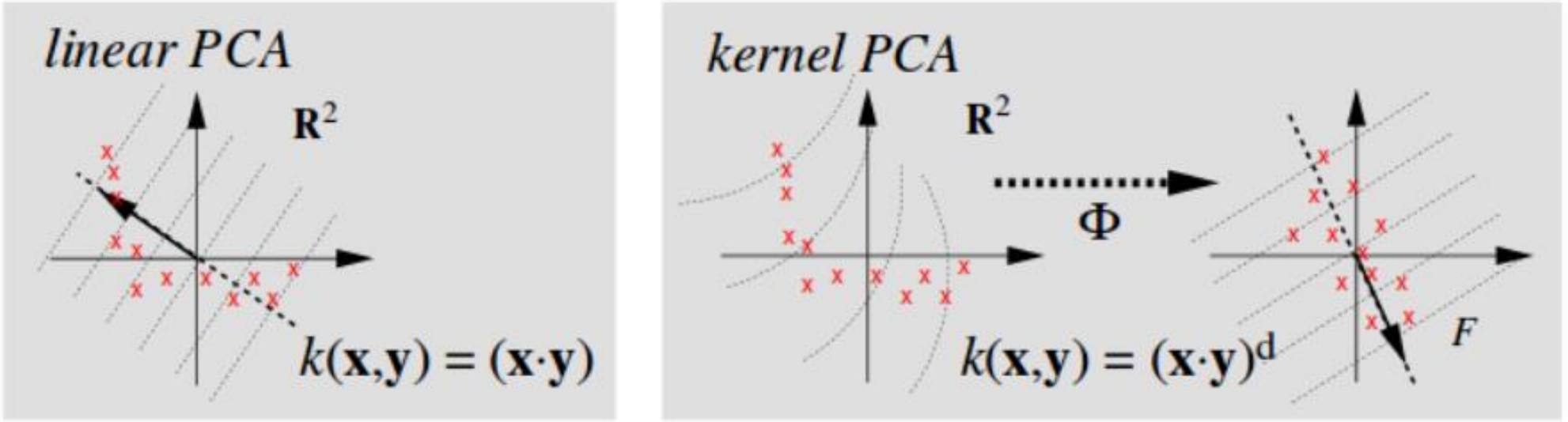
$$\begin{aligned}
\hat{K}_{ij} &= \Big(\phi(x_i) - \frac{1}{N}\sum_{k=1}^{N}\phi(x_k)\Big)\Big(\phi(x_j) - \frac{1}{N}\sum_{l=1}^{N}\phi(x_l)\Big) \\
&= K_{ij} - \frac{1}{N}\sum_{k=1}^{N}\phi^T(x_i)\phi(x_k) - \frac{1}{N}\sum_{l=1}^{N}\phi^T(x_j)\phi(x_l) + \frac{1}{N^2}\sum_{kl=1}^{N}\phi^T(x_k)\phi(x_l) \\
&= K_{ij} - \frac{1}{N}\sum_{k=1}^{N}K_{i,k} - \frac{1}{N}\sum_{k=1}^{N}K_{j,k} + \frac{1}{N^2}\sum_{k,l=1}^{N}K_{k,l}
\end{aligned}$$

Kernel PCA example



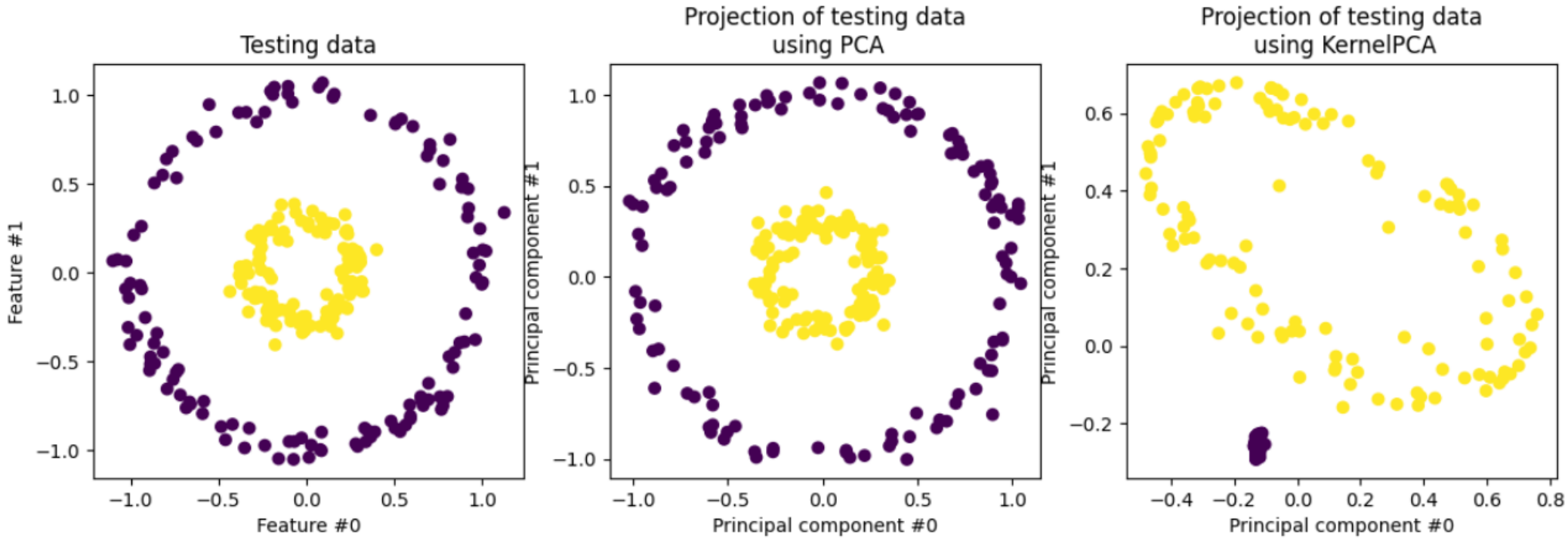


Kernel PCA example

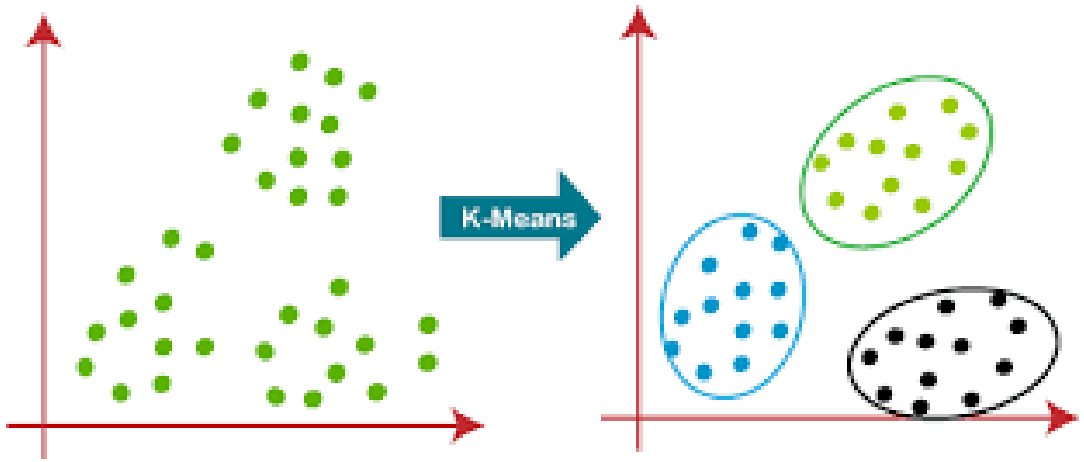


Both PCA and KPCA will reduce the dimensionality but KPCA will also find “non-linear directions”

Kernel PCA


Testing data
Feature #1
Feature #0
1.0
0.5
0.0
−0.5
−1.0
Projection of testing data
using PCA
Principal component #1
Principal component #0
Projection of testing data
using KernelPCA
0.6
0.4
0.2
0.0
−0.2
−0.4
0.8

k-means:recap



$$X = [x_1, x_2, x_3, \ldots, x_{100}]$$

$$\min_{C_1,C_2,C_3,\ldots,C_k} \sum_{i=1}^{k} \sum_{x \in C_i} ||x - u_i||^2$$

$$u_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

Rewriting and simplifying (1)

X1 belongs to the fist cluster (more ones)

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

- Matrix X is the input data points arranged as the columns, dimension MxN
- Matrix B is the cluster assignments of each data point, dimension NxK
- Matrix D is the number of data points assigned to each cluster, inverted and placed along the diagonal, dimension KxK

$$D = \text{diag}\left(1/|C_1|, \quad 1/|C_2|, \quad 1/|C_3|, \quad \cdots, \quad 1/C_k|\right)$$

$$D = \begin{bmatrix} 1/10 & 0 & 0 & 0 \\ 0 & 1/20 & 0 & 0 \\ 0 & 0 & 1/50 & 0 \\ 0 & 0 & 0 & 1/20 \end{bmatrix}$$

Rewriting and simplifying (2)

$$\mathrm{XBD} = \begin{bmatrix} u_1, & u_2, & u_3, & u_4 \end{bmatrix}$$

Matrix with the centroid of each cluster on the columns, dimension MxK

Multiplying the XBD matrix by B^T gives a matrix with the same dimension as X, where each column is the respective centroid of the cluster the data point is assigned to.

$$\mathrm{XBDB}^T = \begin{bmatrix} u_1, & u_3, & u_2, & \dots \end{bmatrix}$$

Matrix where each column corresponds to the cluster centroid of the assigned cluster. For example, if data point 1 and 2 are assigned to clusters 1 and 3, then the first and second column of the $XBDB^T$ matrix will be the cluster centroid of cluster 1 and cluster 3, respectively. And so this pattern continues for all the data points, yielding a matrix the same dimension as X (M by N).

Thus the problem can be compactly formulated as

$$\min_B ||X - XBDB^T||_F^2$$

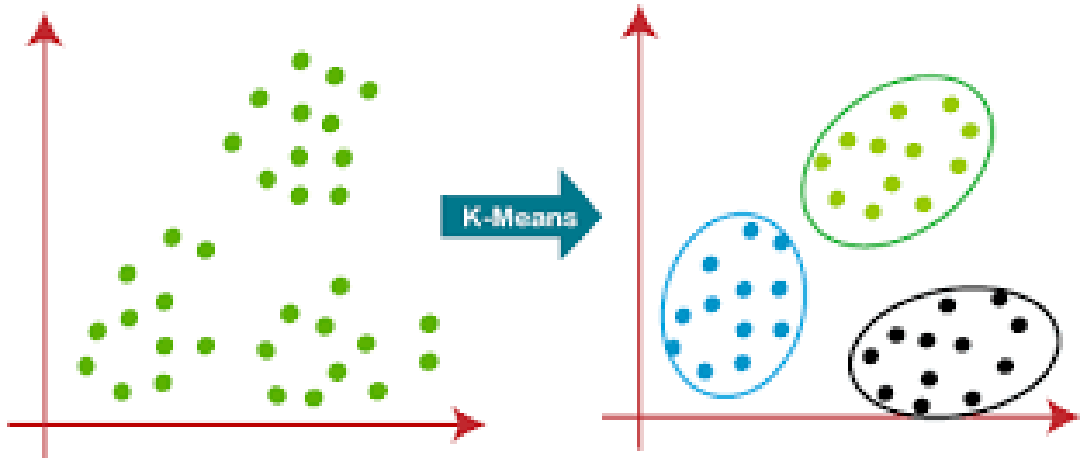
$$\min_{C_1,C_2,C_3,\ldots,C_k} \sum_{i=1}^{k} \sum_{x \in C_i} ||x - u_i||^2$$

equivalently

$$\max_B tr(X^T XBDB^T)$$

Do you see how to kernelize k-means?

Alternatively



find C representative points in the data set to minimize sum of squared error

$$\min_{U} \sum_{k=1}^{C} \sum_{i=1}^{n} U_{ki} \| c_k - x_i \|_2^2$$

Euclidean distance

1 if x_i belongs to cluster k; 0 otherwise

Cluster centers

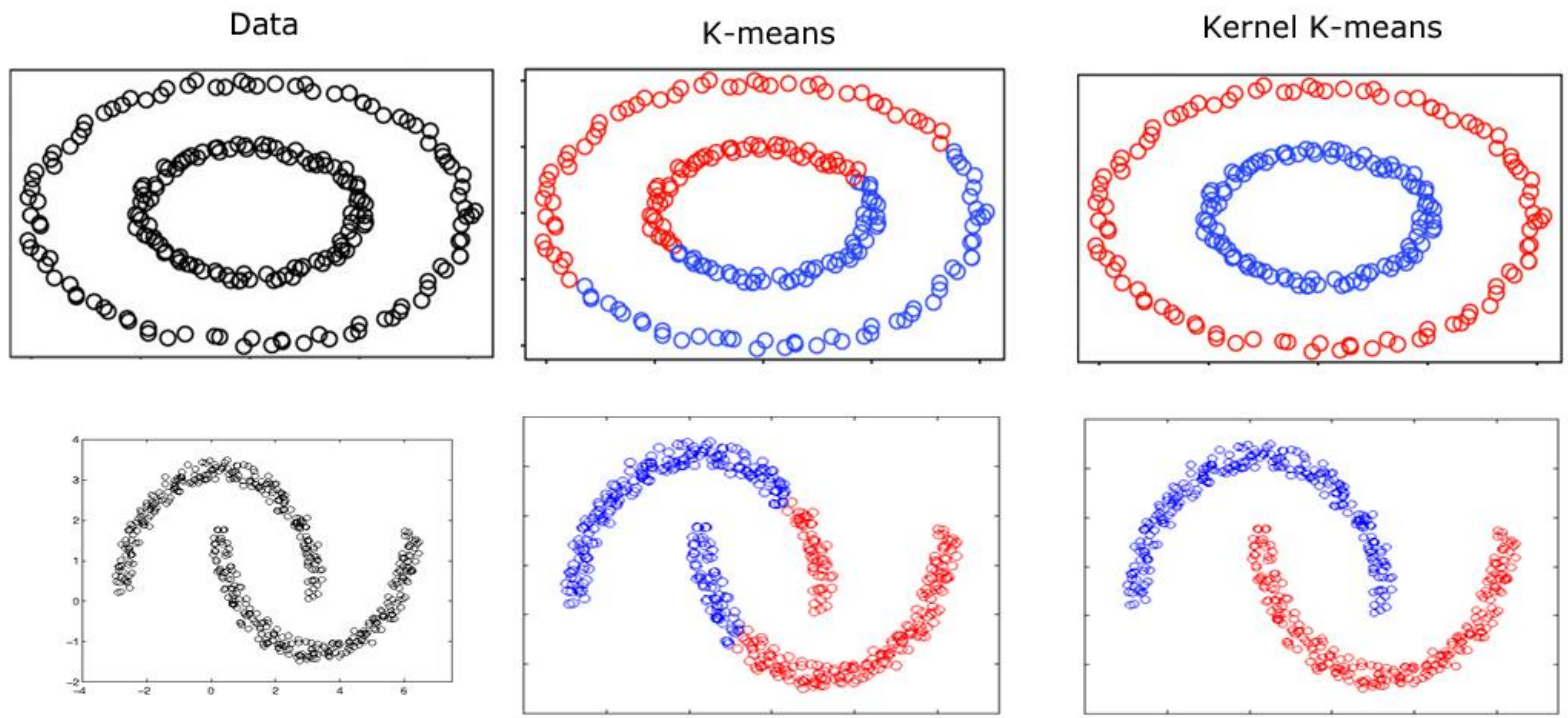
Kernel k-means

Finding C clusters in Hilbert (feature) space H to minimize sum of squared error

$$\min_{U} \sum_{k=1}^{C} \sum_{i=1}^{n} U_{ki} \| c_k - \varphi(x_i) \|_H^2$$

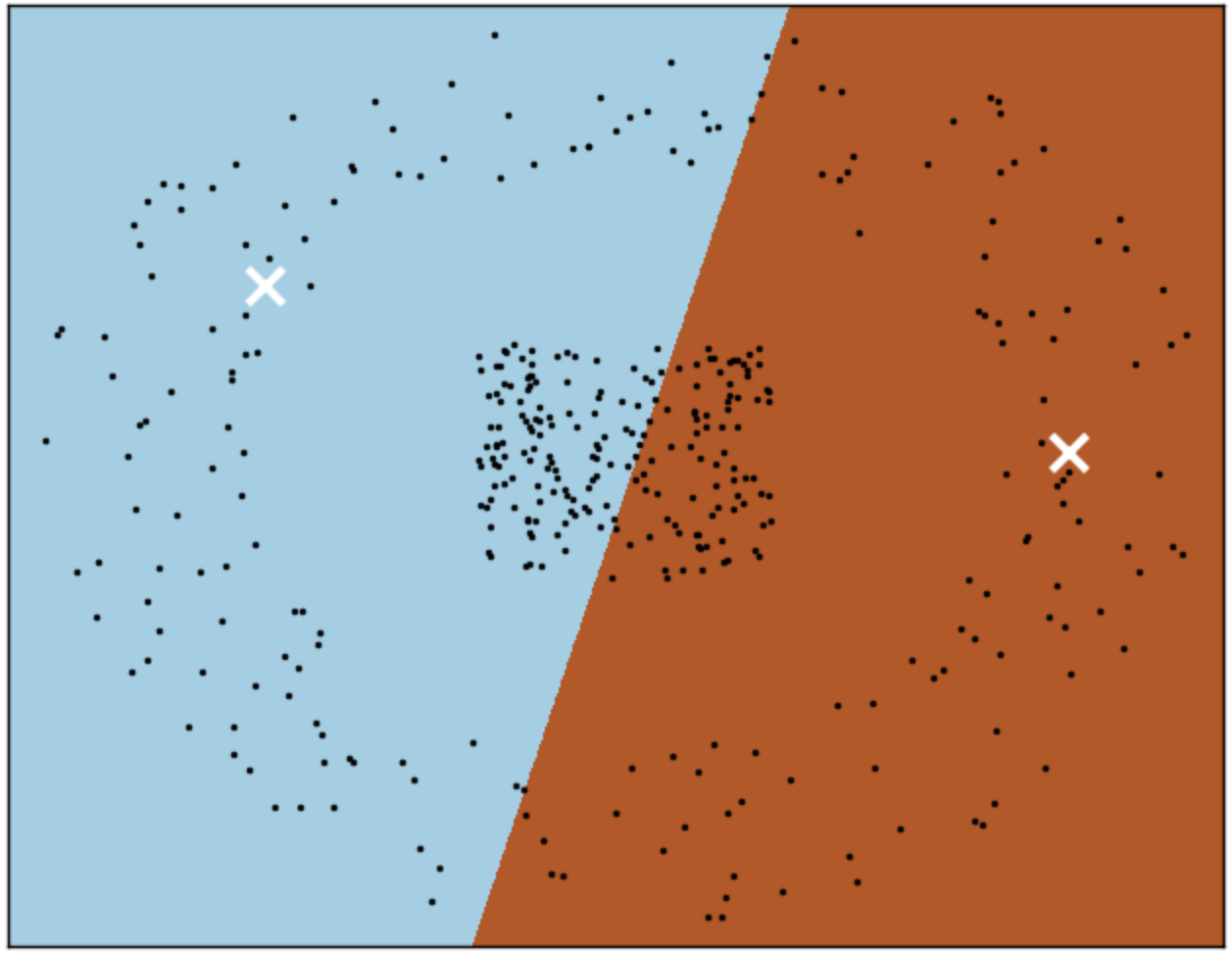
$$\min_{U} \sum_{i=1}^{n} \sum_{k=1}^{C} U_{ki} \left[\kappa(x_i, x_i) - \frac{2}{n_k} \sum_{j=1}^{n} U_{kj} \kappa(x_i, x_j) + \frac{1}{n_k^2} \sum_{j=1}^{n} \sum_{l=1}^{n} U_{kj} U_{kl} \kappa(x_j, x_l) \right]$$

K-means Vs Kernel K-means

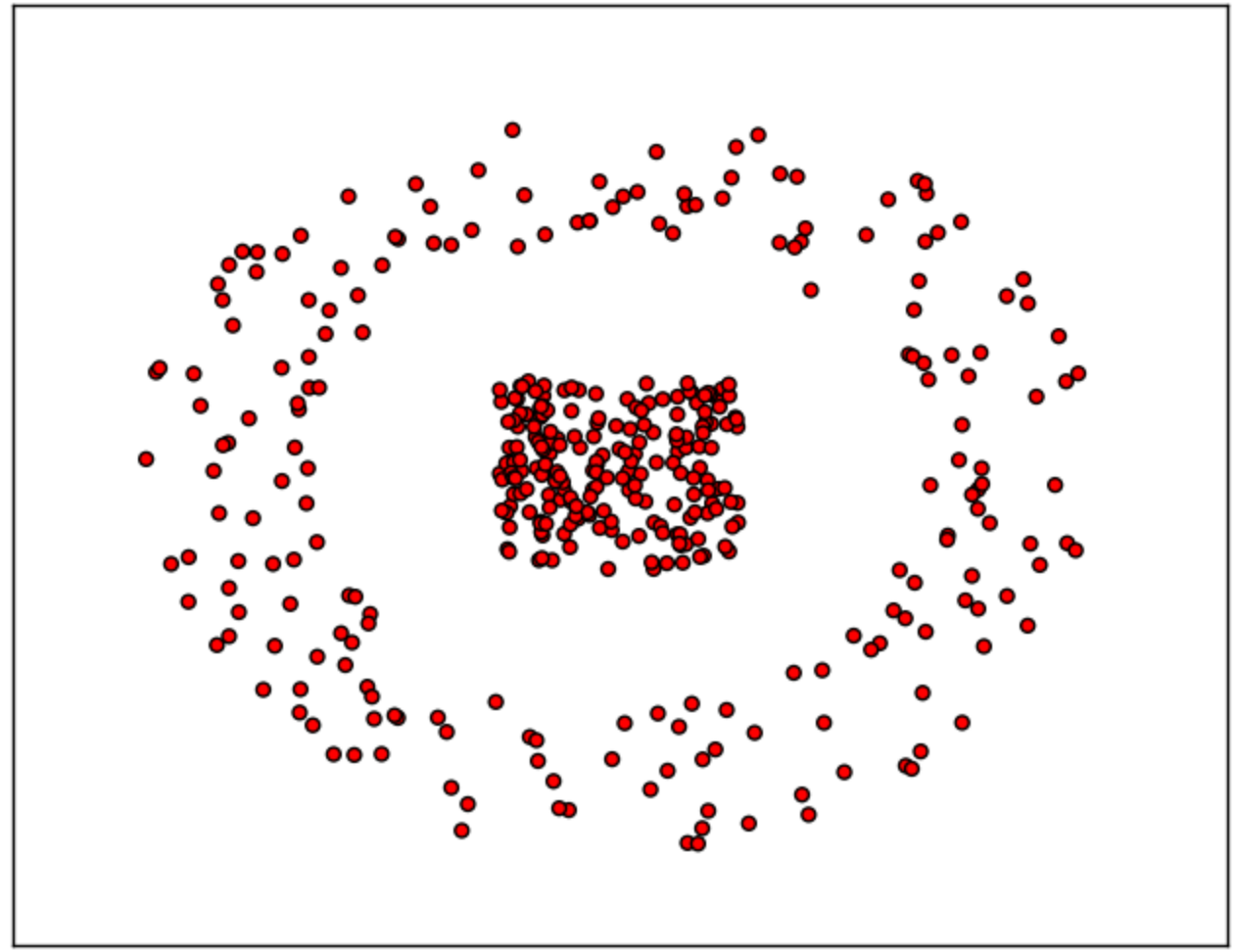


Kernel K-means is able to find “complex” clusters.

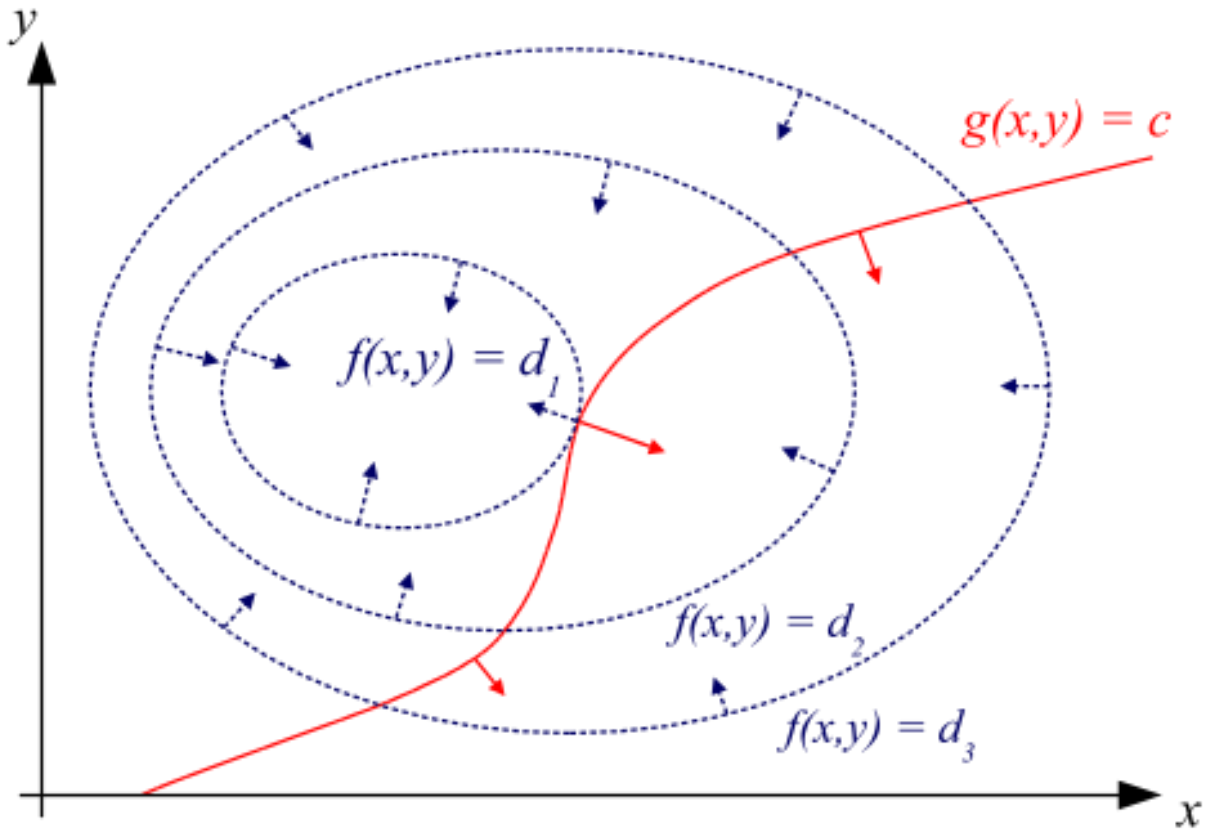
KMeans Iteration:0



Kernel KMeans Iteration:0

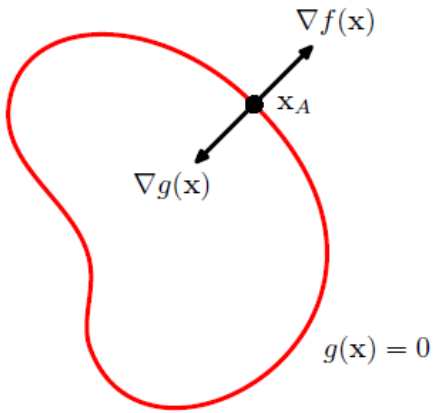


Lagrangian multipliers



$$\max \ f(x,y), \quad s.t. \quad g(x,y) = 0$$

Lagrange Multipliers



Consider the problem:

$$\max_{\boldsymbol{x}} f(\boldsymbol{x})$$
$$s.t. \quad g(\boldsymbol{x}) = 0$$

- Points on $g(\boldsymbol{x}) = 0$ must have $\nabla g(\boldsymbol{x})$ normal to surface
- A stationary point must have no change in f in the direction of the constraint surface, so $\nabla f(\boldsymbol{x})$ must also be normal to the surface.
  - So there must be some $\lambda \neq 0$ such that $\nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0$
- Define Lagrangian:

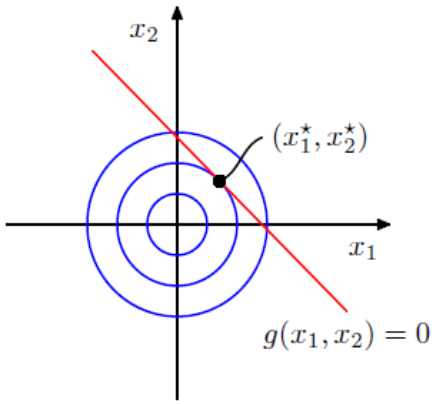
$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda g(\boldsymbol{x})$$

- Stationary points of $L(\boldsymbol{x}, \lambda)$ have $\nabla_x L(\boldsymbol{x}, \lambda) = \nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0$ and $\nabla_\lambda L(\boldsymbol{x}, \lambda) = g(\boldsymbol{x}) = 0$

This is because on the curves g is constant

This is because they are the same vector up to a multiplicative constant

Lagrange Multipliers Example



- Consider the problem

$$\max_{\boldsymbol{x}} f(x_1, x_2) = 1 - x_1^2 - x_2^2$$
$$s.t. \quad g(x_1, x_2) = x_1 + x_2 - 1 = 0$$

- Lagrangian:

$$L(\boldsymbol{x}, \lambda) = 1 - x_1^2 - x_2^2 + \lambda(x_1 + x_2 - 1)$$

- Stationary points require:

$$\begin{aligned} \partial L/\partial x_1 &= -2x_1 + \lambda = 0 \\ \partial L/\partial x_2 &= -2x_2 + \lambda = 0 \\ \partial L/\partial \lambda &= x_1 + x_2 - 1 = 0 \end{aligned}$$

- So stationary point is $(x_1^*, x_2^*) = (\frac{1}{2}, \frac{1}{2})$, $\lambda = 1$

Another example

Example 1. Consider the optimization

$$\begin{aligned} &\underset{x\in\mathbb{R}^n}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{x}_0\|^2, \\ &\text{subject to} \ \ \boldsymbol{A}\boldsymbol{x}=\boldsymbol{y}. \end{aligned} \tag{1.70}$$

The Lagrangian function of the problem is

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{\nu}) = \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{x}_0\|^2 - \boldsymbol{\nu}^T(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}).$$

The stationarity condition implies that $\nabla_x \mathcal{L}(\boldsymbol{x},\boldsymbol{\nu}) = \boldsymbol{0}$, which means that $2(\boldsymbol{x}-\boldsymbol{x}_0)-\boldsymbol{A}\boldsymbol{\nu} = \boldsymbol{0}$. Thus, we have $\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{A}^T\boldsymbol{\nu}$. Multiplying both sides by $\boldsymbol{A}$ yields $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\nu}$, and since the primal feasibility implies that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$, we can show that

$$\boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\nu} = \boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}_0 \quad \Rightarrow \quad \boldsymbol{\nu} = (\boldsymbol{A}\boldsymbol{A}^T)^{-1}(\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}_0).$$

Therefore, the solution is $\boldsymbol{x} = \boldsymbol{x}_0 + \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}(\boldsymbol{y}-\boldsymbol{A}\boldsymbol{x}_0)$

Further material