Kernel Methods and Semi-Supervised Learning

Zalán Bodó

Faculty of Mathematics and Computer Science,
Babeș–Bolyai University, Cluj-Napoca/Kolozsvár/Klausenburg

Derby, June 2016

Contents

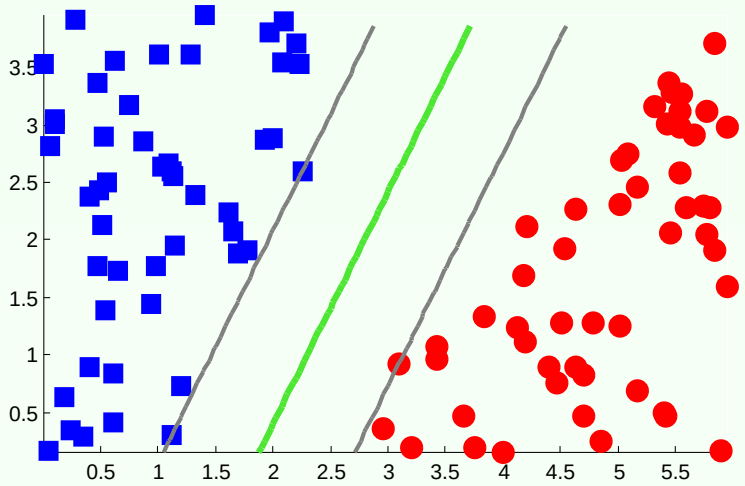
The aims of the short course

- to get an overview of kernel methods
- to get an overview of semi-supervised classification

- **kernel methods**: change the kernel $\Rightarrow$ obtain a *new* algorithm
- **semi-supervised learning**: use a large unlabeled dataset
- **semi-supervised kernels**: use a supervised algorithm with a semi-supervised kernel to obtain a semi-supervised learning method
- no need to develop semi-supervised methods
- use a supervised algorithm with a good – and replaceable – semi-supervised kernel

Machine learning. Supervised learning

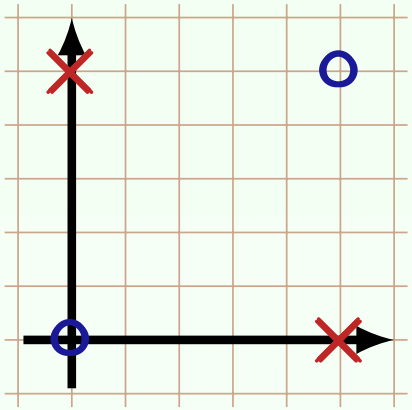
- machine learning (ML): subdomain of AI, modeling the most important brain activities: classification, differentiation, prediction
- **supervised learning**: examples with teaching instructions
  - *example*: learn to differentiate between relevant and spam emails
- **unsupervised learning**: no teaching instructions; group similar points into clusters
  - *example*: find products similar to a given one
- **semi-supervised learning**: halfway between supervised and unsupervised learning
  - *example*: learn to classify handwritten digits based on a few labeled and a large set of unlabeled examples

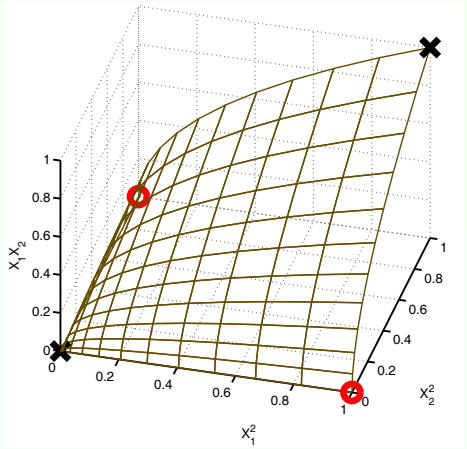


- training data: $\mathcal{D}_{train} = \{(\boldsymbol{x}_i, y_i) \mid i = 1, \ldots, N\}$, $\boldsymbol{x}_i \in X$, $y_i \in Y$
- X is the input space, $\boldsymbol{x}_i$'s are vectors
- Y is the label set, y_i's are scalars
- goal: find $\hat{f} : X \to Y$ to approximate f given by $\mathcal{D}_{train}$
  - if $|Y| = 2 \Rightarrow$ binary classification
  - if $|Y| > 2 \Rightarrow$ multi-class classification
  - if $f : X \to Y \Rightarrow$ single-label classification
  - if $f : X \to 2^Y \Rightarrow$ multi-label classification

Kernel methods

- James Mercer (1909): *any continuous symmetric, positive semi-definite kernel function can be expressed as a dot product in a high-dimensional space*
- M. Aizerman, E. Braverman, and L. Rozonoer – 1964
- Boser, Guyon, and Vapnik – 1992 (Support Vector Machine)
- linear algorithm $\rightarrow$ non-linear algorithm
- ϕ: feature mapping
- **kernels**: $k(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle$, $\phi : X \rightarrow \mathcal{H}$, $X \subseteq \mathbb{R}^n$ i.e. cosine of the angle enclosed by the vectors in a high-dimensional space
- covers all geometric constructions that can be formulated in terms of angles, lengths and distances
- any positive definite kernel is a dot product in another space; any mapping ϕ to a dot product space defines a positive definite kernel by $\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle$





- we want to solve the XOR-problem (shown above) by a linear separator; data: $(0,0),(1,1),(1,0),(0,1)$
- easy to observe: this cannot be done in the initial space
- lets use the following mapping: $\phi(\boldsymbol{x}) = \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}x_1x_2 \end{bmatrix}'$
- then the dot product:
  $\phi(\boldsymbol{x})'\phi(\boldsymbol{z}) = x_1^2z_1^2 + 2x_1z_1x_2z_2 + x_2^2z_2^2 = (\boldsymbol{x}'\boldsymbol{z})^2 = k(\boldsymbol{x},\boldsymbol{z})$
- it is called the second order homogeneous polynomial kernel

- we work in/with Hilbert spaces
- what is a Hilbert space?

Def. Inner product

An inner product on a complex linear space X is a map

$$\langle \cdot, \cdot \rangle : X \times X \to \mathbb{C}$$

s.t. $\forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in X$ and $\lambda, \mu \in \mathbb{C}$:

(a) $\langle \boldsymbol{x}, \lambda \boldsymbol{y} + \mu \boldsymbol{z} \rangle = \lambda \langle \boldsymbol{x}, \boldsymbol{y} \rangle + \mu \langle \boldsymbol{x}, \boldsymbol{z} \rangle$ (linear in the 2nd argument)

(b) $\langle \boldsymbol{y}, \boldsymbol{x} \rangle = \overline{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}$ (Hermitian symmetric)

(c) $\langle \boldsymbol{x}, \boldsymbol{x} \rangle \geq 0$ (non-negative)

(d) $\langle \boldsymbol{x}, \boldsymbol{x} \rangle = 0$ iff $\boldsymbol{x} = \boldsymbol{0}$ (positive definite)

Def. Inner product space, pre-Hilbert space

A linear space with an inner product is called an inner product space or pre-Hilbert space.

Def. Hilbert space

A Hilbert space is an inner product space that is also a complete metric space with respect to the distance function induced by the inner product.

Norm is defined as

$$\|\boldsymbol{x}\| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}$$

Distance is defined as

$$d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\| = \sqrt{\langle \boldsymbol{x} - \boldsymbol{y}, \boldsymbol{x} - \boldsymbol{y} \rangle}$$

Example

Euclidean space: $\langle \boldsymbol{x}, \boldsymbol{z} \rangle = \sum_{i=1}^{d} x_i z_i, \quad \|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{d} x_i^2}$

General-purpose kernels

- linear (dot product):

$$k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}'\boldsymbol{z}$$

- polynomial:

$$k(\boldsymbol{x}, \boldsymbol{z}) = (a\boldsymbol{x}'\boldsymbol{z} + b)^c$$

- RBF (Gaussian):

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|_2^2}{2\sigma^2}\right) = \exp\left(-\gamma\|\boldsymbol{x} - \boldsymbol{z}\|_2^2\right)$$

General-purpose kernels

- linear (dot product):

$$k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}'\boldsymbol{z}$$

- polynomial:

$$k(\boldsymbol{x}, \boldsymbol{z}) = (a\boldsymbol{x}'\boldsymbol{z} + b)^c$$

- RBF (Gaussian):

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|_2^2}{2\sigma^2}\right) = \exp\left(-\gamma\|\boldsymbol{x} - \boldsymbol{z}\|_2^2\right)$$

General-purpose kernels

- linear (dot product):

$$k(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{x}'\boldsymbol{z}$$

- polynomial:

$$k(\boldsymbol{x}, \boldsymbol{z}) = (a\boldsymbol{x}'\boldsymbol{z} + b)^c$$

- RBF (Gaussian):

$$k(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|_2^2}{2\sigma^2}\right) = \exp\left(-\gamma\|\boldsymbol{x} - \boldsymbol{z}\|_2^2\right)$$

Kernel trick

Given an algorithm which is formulated in terms of a positive definite kernel k, one can construct an alternative algorithm by replacing k by another positive definite kernel $\widetilde{k}$.

Distances $\leftrightarrow$ kernels

1. kernel $\rightarrow$ (squared) distance:

$$\boldsymbol{D}^{(2)} = \mathrm{diag}(\boldsymbol{K})\mathbf{1}_{NN} + \mathbf{1}_{NN}\mathrm{diag}(\boldsymbol{K}) - 2\boldsymbol{K}$$

2. distance $\rightarrow$ kernel:

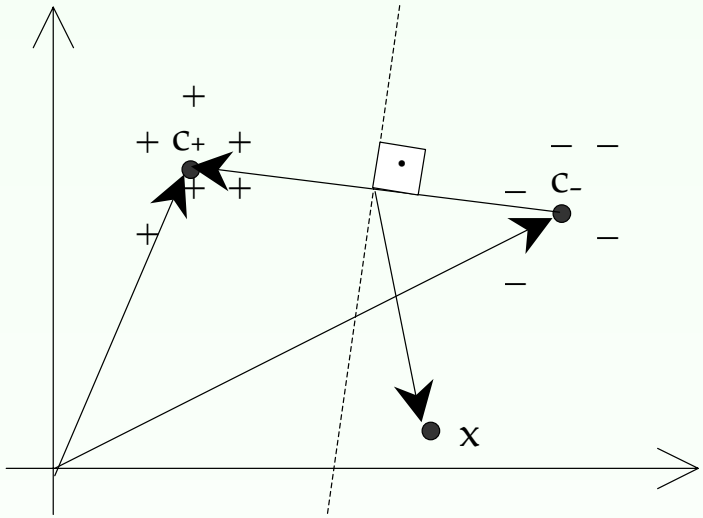
$$\begin{aligned} -\frac{1}{2}\boldsymbol{J}\boldsymbol{D}^{(2)}\boldsymbol{J} &= -\frac{1}{2}\boldsymbol{J}\mathrm{diag}(\boldsymbol{K})\mathbf{1}_{NN}\boldsymbol{J} - \frac{1}{2}\boldsymbol{J}\mathbf{1}_{NN}\mathrm{diag}(\boldsymbol{K})'\boldsymbol{J} + \boldsymbol{J}\boldsymbol{K}\boldsymbol{J} \\ &= \boldsymbol{J}\boldsymbol{K}\boldsymbol{J} \end{aligned}$$

   where $\boldsymbol{J} = \boldsymbol{I} - \frac{1}{N}\mathbf{1}\mathbf{1}' = \boldsymbol{I} - \frac{1}{N}\mathbf{1}_{NN}$ is the centering matrix (since $\mathbf{1}_{NN}\boldsymbol{J} = \mathbf{0}$)

Effect of double centering: 0 mean (rows & columns)

Outline:

- learn: calculate the class centroids
- classify: label an unseen example as belonging to the class with closest centroid



Binary classification:

$$m_+ = |\{\boldsymbol{x}_i | y_i = +1\}|, m_- = |\{\boldsymbol{x}_i | y_i = -1\}|$$

centers:

$$\boldsymbol{c}_+ = \frac{1}{m_+} \sum_{\{i|y_i=+1\}} \boldsymbol{x}_i$$

$$\boldsymbol{c}_- = \frac{1}{m_-} \sum_{\{i|y_i=-1\}} \boldsymbol{x}_i$$

Let $\boldsymbol{w} := \boldsymbol{c}_+ - \boldsymbol{c}_-$ and $\boldsymbol{c} := (\boldsymbol{c}_+ + \boldsymbol{c}_-)/2$; label will be determined by $\boldsymbol{w}'(\boldsymbol{x} - \boldsymbol{c})$

$$\begin{aligned} y &= \operatorname{sgn}\left((\boldsymbol{x} - \boldsymbol{c})'\boldsymbol{w}\right) \\ &= \operatorname{sgn}\left(\boldsymbol{c}'_+\boldsymbol{x} - \boldsymbol{c}'_-\boldsymbol{x} + b\right), \end{aligned}$$

where $b := \left(\|\boldsymbol{c}_-\|^2 - \|\boldsymbol{c}_+\|^2\right)/2$

Substituting the corresponding expressions for $\boldsymbol{c}_+$ and $\boldsymbol{c}_-$ we obtain

$$\begin{aligned} y &= \operatorname{sgn}\left(\frac{1}{m_+}\sum_{\{i|y_i=+1\}} \boldsymbol{x}'\boldsymbol{x}_i - \frac{1}{m_-}\sum_{\{i|y_i=-1\}} \boldsymbol{x}'\boldsymbol{x}_i + b\right) \\ &= \operatorname{sgn}\left(\frac{1}{m_+}\sum_{\{i|y_i=+1\}} k(\boldsymbol{x},\boldsymbol{x}_i) - \frac{1}{m_-}\sum_{\{i|y_i=-1\}} k(\boldsymbol{x},\boldsymbol{x}_i) + b\right), \end{aligned}$$

where

$$b := \frac{1}{2}\left(\frac{1}{m_-^2}\sum_{\{(i,j)|y_i=y_j=-1\}} k(\boldsymbol{x}_i,\boldsymbol{x}_j) - \frac{1}{m_+^2}\sum_{\{(i,j)|y_i=y_j=+1\}} k(\boldsymbol{x}_i,\boldsymbol{x}_j)\right)$$

and $k(\cdot,\cdot)$ is an arbitrary kernel function.

For **multi-class** classification:

$$\boldsymbol{c}_i = \frac{1}{|C_i|} \sum_{\boldsymbol{x}_j \in C_i} \boldsymbol{x}_i$$

The Euclidean distance can be written in terms of the dot product:

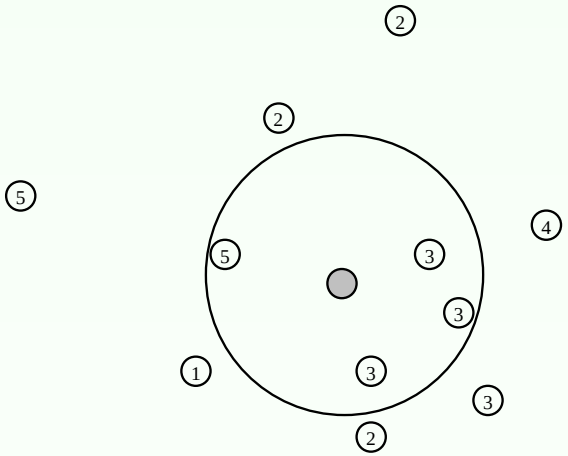
$$\|\boldsymbol{x} - \boldsymbol{z}\|^2 = \boldsymbol{x}'\boldsymbol{x} - 2\boldsymbol{x}'\boldsymbol{z} + \boldsymbol{z}'\boldsymbol{z} = k(\boldsymbol{x}, \boldsymbol{x}) - 2k(\boldsymbol{x}, \boldsymbol{z}) + k(\boldsymbol{z}, \boldsymbol{z})$$

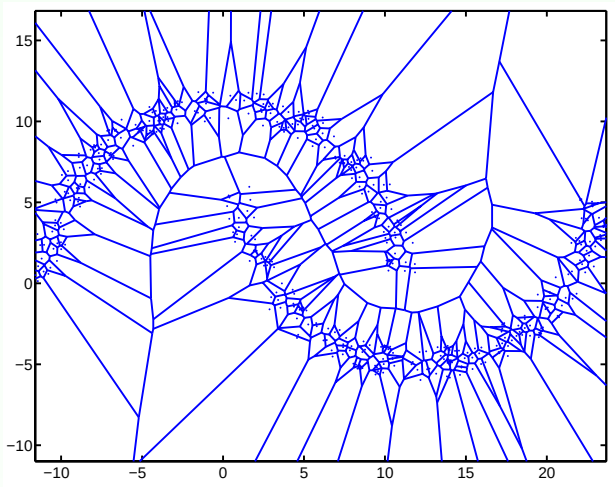
Since distances from the centroids are to be considered, we can calculate:

$$\begin{aligned}
\|\boldsymbol{x} - \boldsymbol{c}_i\|^2 &= \|\boldsymbol{x} - \frac{1}{|C_i|} \sum_{\boldsymbol{x}_j \in C_i} \boldsymbol{x}_i\|^2 \\
&= \boldsymbol{x}'\boldsymbol{x} - \frac{2}{|C_i|} \sum_{\boldsymbol{x}_j \in C_i} \boldsymbol{x}'\boldsymbol{x}_j + \frac{1}{|C_i|^2} \sum_{\boldsymbol{x}_j \in C_i, \boldsymbol{x}_k \in C_i} \boldsymbol{x}_j'\boldsymbol{x}_k \\
&= k(\boldsymbol{x}, \boldsymbol{x}) - \frac{2}{|C_i|} \sum_{\boldsymbol{x}_j \in C_i} k(\boldsymbol{x}, \boldsymbol{x}_j) + \frac{1}{|C_i|^2} \sum_{\boldsymbol{x}_j \in C_i, \boldsymbol{x}_k \in C_i} k(\boldsymbol{x}_j, \boldsymbol{x}_k)
\end{aligned}$$

Outline:

- *seemingly* no separate learning phase
- classify: find the k nearest neighbors of the example in question and label with the majority class





- assign the label which is in majority among the k-nearest neighbors

$$f(\boldsymbol{x}) = \underset{c=1,2,\ldots,K}{\operatorname{argmax}} \sum_{\boldsymbol{z} \in N_k(\boldsymbol{x}), \boldsymbol{x} \in \mathcal{D}_{train}} \operatorname{sim}(\boldsymbol{z}, \boldsymbol{x}) \cdot \delta(c, f(\boldsymbol{z}))$$

(we use $\operatorname{sim}(\boldsymbol{z}, \boldsymbol{x}) = 1, \forall \boldsymbol{z}, \boldsymbol{x}$)

$$\delta(a, b) = \begin{cases} 1, & a = b \\ 0, & \text{otherwise} \end{cases}$$

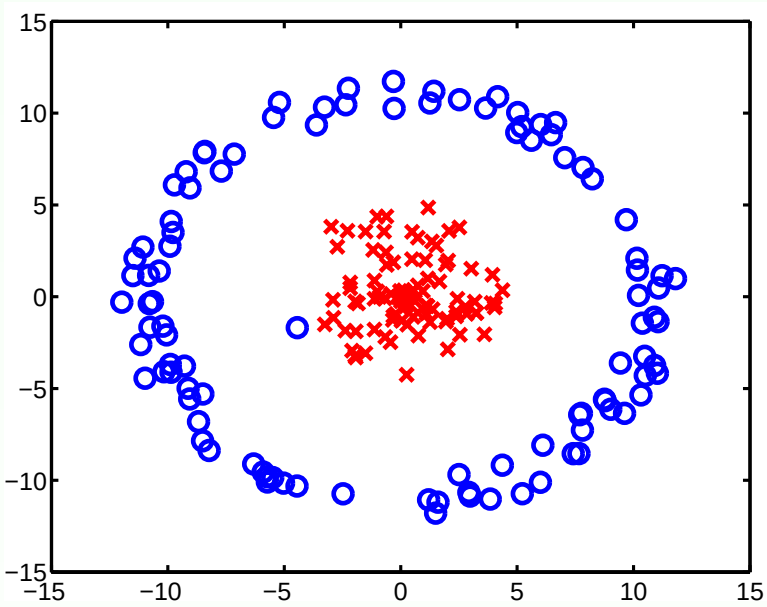
(Kronecker delta)

- to determine $N_k(\boldsymbol{x})$ the Euclidean distance is used
- as before, we can rewrite the Euclidean distance using dot products:

$$\|\boldsymbol{x} - \boldsymbol{z}\|^2 = \boldsymbol{x}'\boldsymbol{x} - 2\boldsymbol{x}'\boldsymbol{z} + \boldsymbol{z}'\boldsymbol{z} = k(\boldsymbol{x}, \boldsymbol{x}) - 2k(\boldsymbol{x}, \boldsymbol{z}) + k(\boldsymbol{z}, \boldsymbol{z})$$

Outline:

- clustering method; no supervised information is given
- goal: cluster data into K sets, such that similar points are grouped in the same set
- determine K cluster centers



- objective function:

$$F = \sum_{i=1}^{K} \sum_{j=1}^{N} U_{ij} d(\boldsymbol{x}_j, \boldsymbol{c}_i)$$

  where
  - $\boldsymbol{c}_i, i = 1, \ldots, K$ are the cluster centers
  - $\boldsymbol{U}$ is the membership matrix of size $K \times N$ (K clusters, N points)
  - $U_{ij} \in \{0, 1\}$; $U_{ij} = 1$ if $\boldsymbol{x}_j$ belongs to C_i (i-th cluster); otherwise 0
  - $d(\cdot, \cdot)$ is a distance, usually the squared Euclidean: $d(\boldsymbol{x}, \boldsymbol{z}) = \|\boldsymbol{x} - \boldsymbol{z}\|^2$
- F is minimized if the points are assigned to the closest cluster center, provided they are fixed:

$$U_{ij} = \begin{cases} 1, & \text{if } \|\boldsymbol{x}_j - \boldsymbol{c}_i\|^2 \leq \|\boldsymbol{x}_j - \boldsymbol{c}_k\|^2, \forall k \neq i \\ 0, & \text{otherwise} \end{cases}$$

- on the other side, if $\boldsymbol{U}$ is fixed, F is minimized by taking the centers of the groups:

$$\boldsymbol{c}_i = \frac{1}{\sum_{j=1}^{N} U_{ij}} \sum_{k=1}^{N} U_{ik} \boldsymbol{x}_k$$

k-Means

1. Initialize the centers.
2. Determine $\boldsymbol{U}$.
3. Compute cost function F.
4. If converged, then STOP.
   Else update centers using the new $\boldsymbol{U}$; GOTO step 2.

Kernel k-means

- k-means (and fuzzy c-means) are limited to (properly) find pairwise linearly separable clusters
- to obtain non-linear cluster boundaries we use kernels
- the only expression one needs to rewrite for the kernel-based variant of the algorithm is the distance of a point to a center:

$$\|\phi(\boldsymbol{x}) - \frac{1}{s_j}\sum_{k=1}^{N} U_{jk}\phi(\boldsymbol{x}_k)\|^2$$

$$= k(\boldsymbol{x},\boldsymbol{x}) + \frac{1}{s_j^2}\sum_{k,\ell=1}^{N} U_{jk}U_{j\ell}k(\boldsymbol{x}_k,\boldsymbol{x}_\ell) - \frac{2}{s_j}\sum_{k=1}^{N} U_{jk}k(\boldsymbol{x}_k,\boldsymbol{x})$$

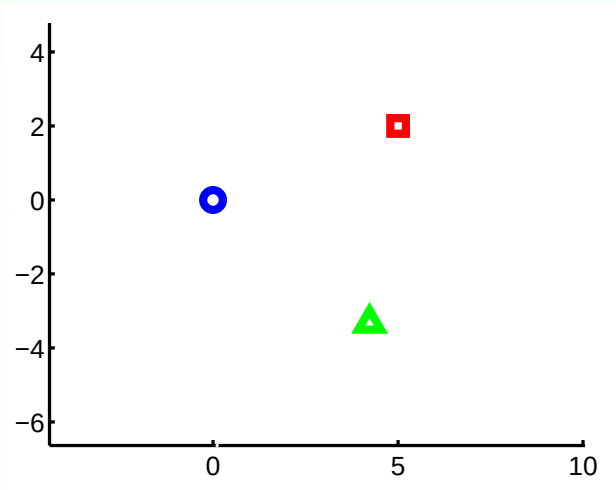
where s_j is the size of the j-th cluster

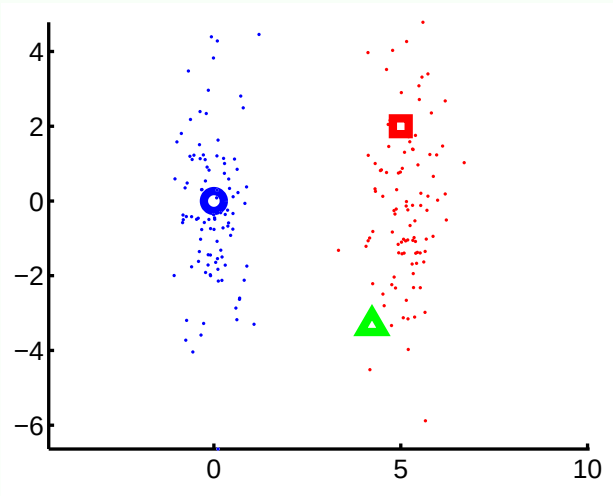
Kernel k-means

1. Generate a random $\boldsymbol{U}$ which satisfies the conditions, that is make a random assignments of the points.
2. Compute the cost function F.
3. If converged then STOP.
   Else update $\boldsymbol{U}$ using the update formula used in k-means;
   GOTO step 2.

Semi-supervised learning (SSL)

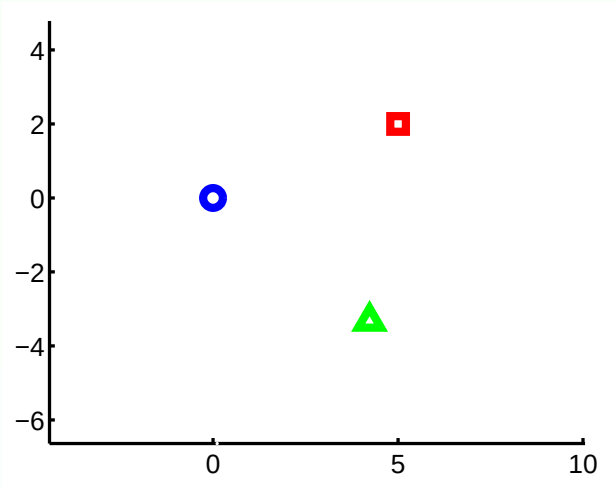
- **supervised** learning:
  $D = \{(\boldsymbol{x}_i, y_i) \,|\, \boldsymbol{x}_i \in X \subseteq \mathbb{R}^d, y_i \in \{-1, +1\}, i = 1, \ldots, \ell\}$;
  find $f : X \to \{-1, +1\}$ which agrees with D
- **semi-supervised** learning:
  $D = \{(\boldsymbol{x}_i, y_i) \,|\, i = 1, \ldots, \ell\} \cup \{\boldsymbol{x}_j \,|\, j = \ell + 1, \ldots, \ell + u\}$,
  $\ell \ll u$, $N = \ell + u$;
  - **inductive**: find $f : X \to \{-1, +1\}$ which agrees with D + use the information of D_U
  - **transductive**: find $f : D_U \to \{-1, +1\}$ by using $D = D_L \cup D_U$

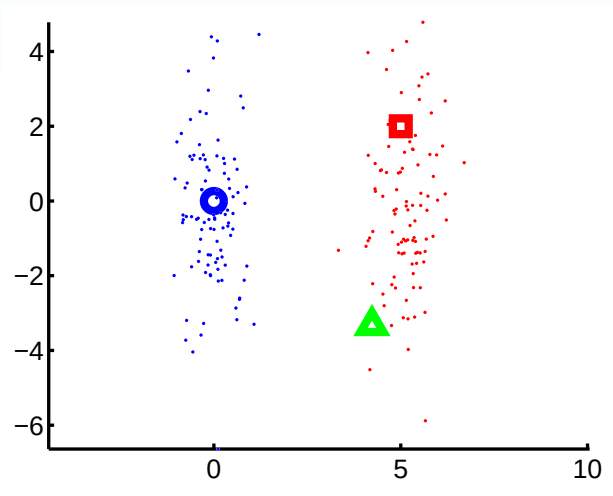




Semi-supervised learning (SSL)

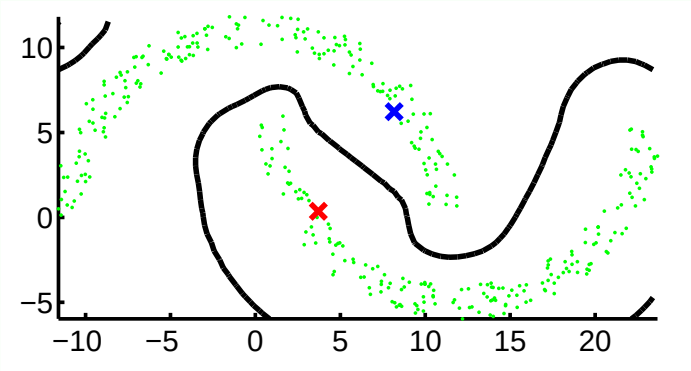
- **supervised** learning:
  $D = \{(\boldsymbol{x}_i, y_i) \,|\, \boldsymbol{x}_i \in X \subseteq \mathbb{R}^d, y_i \in \{-1, +1\}, i = 1, \ldots, \ell\}$;
  find $f : X \to \{-1, +1\}$ which agrees with D
- **semi-supervised** learning:
  $D = \{(\boldsymbol{x}_i, y_i) \,|\, i = 1, \ldots, \ell\} \cup \{\boldsymbol{x}_j \,|\, j = \ell + 1, \ldots, \ell + u\}$,
  $\ell \ll u$, $N = \ell + u$;
  - **inductive**: find $f : X \to \{-1, +1\}$ which agrees with D + use the information of D_U
  - **transductive**: find $f : D_U \to \{-1, +1\}$ by using $D = D_L \cup D_U$





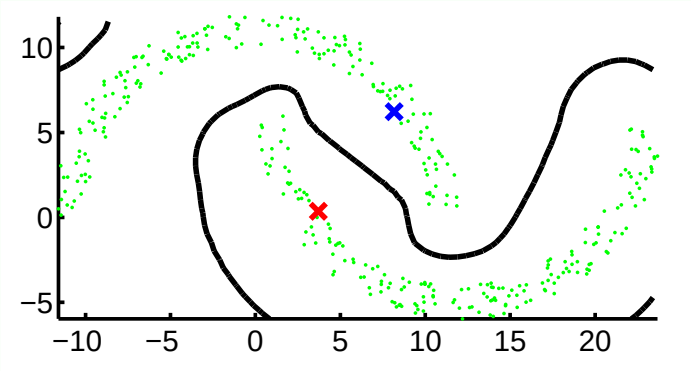
Assumptions in SSL

- **smoothness assumption**: *If two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in a high density region are close, then so should be the corresponding outputs y_i and y_j.*
- **cluster assumption**: *If two points are in the same cluster, they are likely to be of the same class.*
- **manifold assumption**: *The high dimensional data lie roughly on a low dimensional manifold.* – regarding dimensionality; but if manifold = approximation of the high-dimensional region $\Rightarrow$ smoothness assumption



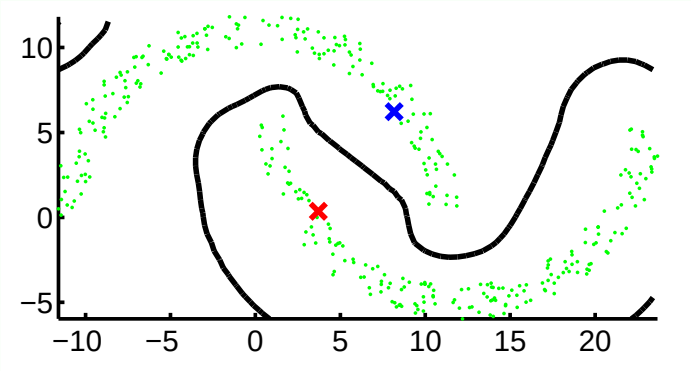
Assumptions in SSL

- **smoothness assumption**: *If two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in a high density region are close, then so should be the corresponding outputs y_i and y_j.*
- **cluster assumption**: *If two points are in the same cluster, they are likely to be of the same class.*
- **manifold assumption**: *The high dimensional data lie roughly on a low dimensional manifold.* – regarding dimensionality; but if manifold = approximation of the high-dimensional region $\Rightarrow$ smoothness assumption



Assumptions in SSL

- **smoothness assumption**: *If two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in a high density region are close, then so should be the corresponding outputs y_i and y_j.*
- **cluster assumption**: *If two points are in the same cluster, they are likely to be of the same class.*
- **manifold assumption**: *The high dimensional data lie roughly on a low dimensional manifold.* – regarding dimensionality; but if manifold = approximation of the high-dimensional region $\Rightarrow$ smoothness assumption



Simple self-training

Until convergence:

1. Train classifier with the labeled data.
2. Classify unlabeled data with the trained classifier.
3. Add the most confident unlabeled points to the training data.
4. GOTO step 1.

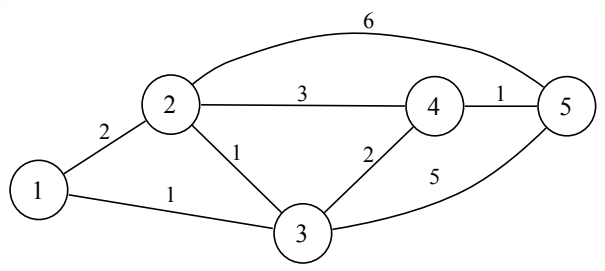
Committee-based learning

Until convergence:

1. Train separate classifiers on the same labeled data.
2. Make predictions on the unlabeled data.
3. Add the most agreed points to the training set.
4. GOTO step 1.

Final prediction: weighted majority vote among all the learners.

- if data lie on a manifold, graph-based distances are better
- **graph-based distances** = shortest path distances calculated using a known algorithm: Floyd–Warshall, Dijkstra, etc.



Floyd–Warshall algorithm

1. D_{ij} = weight of the edge; if no edge then ∞
2. for k=1:N
   for i=1:N
   for j=1:N
   if $D_{ij} > D_{ik} + D_{kj}$
   $D_{ij} = D_{ik} + D_{kj}$

Manifold learning

here: **semi-supervised = graph-based**

1. Initial distance matrix = sparsified (kNN/εNN) distance matrix
2. Compute the all-pair shortest path distance matrix—using for example the Floyd–Warshall algorithm:

$$D_{ij} = \text{shortest path value among all paths from } i \text{ to } j$$

3. Perform supervised learning using these graph distances.

Neighborhood kernel

- representing each as the average of its neighbors $\rightarrow$ neighborhood structure influences representation:

$$\phi_{\mathrm{nbd}}(\boldsymbol{x}) = \frac{1}{|N(\boldsymbol{x})|} \sum_{\boldsymbol{x}' \in N(\boldsymbol{x})} \phi_{\mathrm{b}}(\boldsymbol{x})$$

- the kernel:

$$k_{\mathrm{nbd}}(\boldsymbol{x}, \boldsymbol{z}) = \frac{\sum_{\boldsymbol{x}' \in N(\boldsymbol{x}), \boldsymbol{z}' \in N(\boldsymbol{z})} k_{\mathrm{b}}(\boldsymbol{x}', \boldsymbol{z}')}{|N(\boldsymbol{x})||N(\boldsymbol{z})|}$$

It is pd. because of its definition ($\phi_{\mathrm{nbd}}(\boldsymbol{x})$).

Bagged cluster kernel

- idea: **reweighting** the base kernel values by the probability that the points belong to the same cluster
- use k-means clustering: random initial cluster centers $\rightarrow$ affects the output of the algorithm

BCK

1. Run k-means t times, which results in $c_j(\boldsymbol{x}_i)$ cluster assignments, $j = 1, \ldots, t$, $i = 1, \ldots, N$, $c_\cdot(\cdot) \in \{1, \ldots, K\}$.
2. Construct the bagged kernel in the following way:

$$k_{\text{bag}}(\boldsymbol{x}, \boldsymbol{z}) = \frac{\sum_{j=1}^{t} [c_j(\boldsymbol{x}) = c_j(\boldsymbol{z})]}{t}$$

The final kernel:

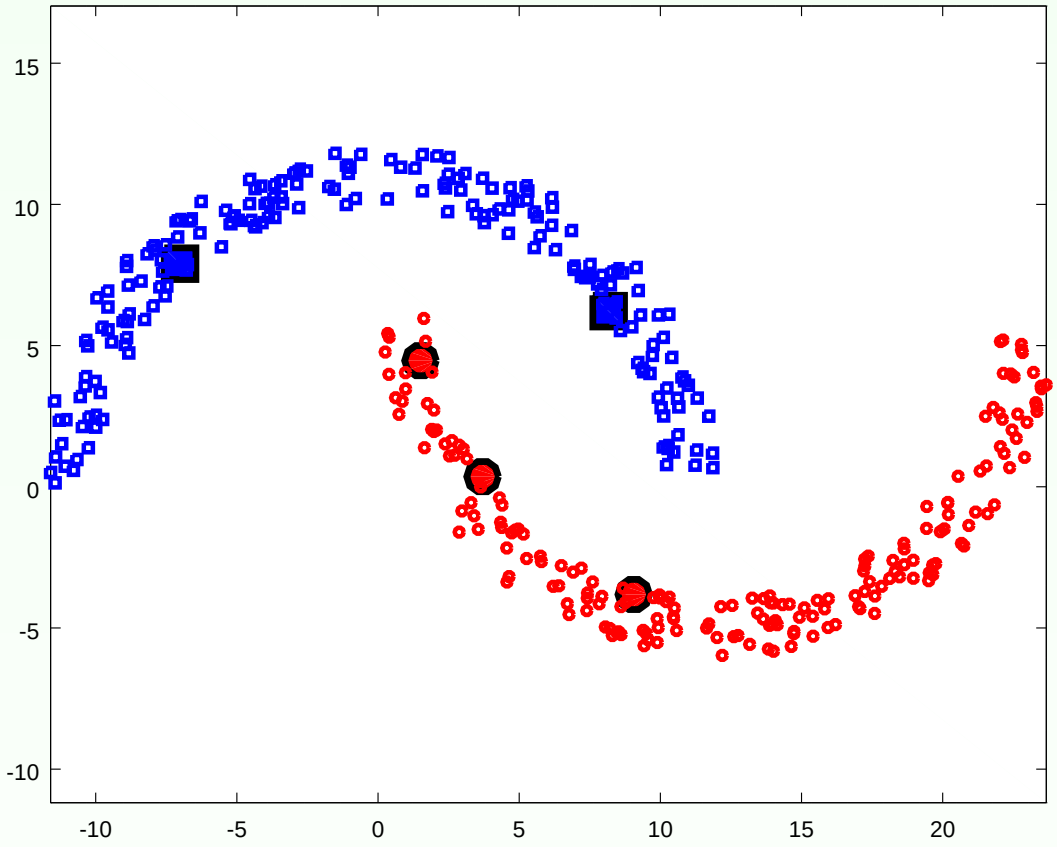
$$k(\boldsymbol{x}, \boldsymbol{z}) = k_{\text{b}}(\boldsymbol{x}, \boldsymbol{z}) \cdot k_{\text{bag}}(\boldsymbol{x}, \boldsymbol{z})$$

It is pd. because we can think of it as a feature mapping to the $t \cdot K$-dimensional space,

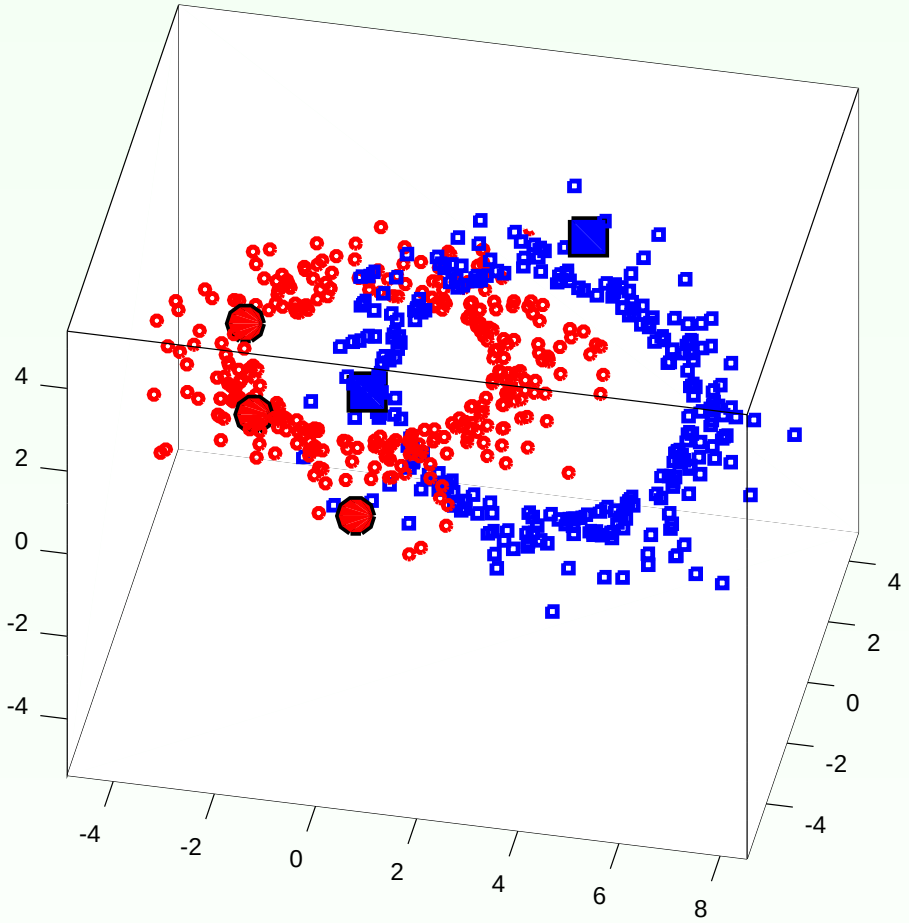
$$\phi_{\text{bag}}(\boldsymbol{x}) = \frac{1}{\sqrt{t}} \langle [c_j(\boldsymbol{x}) = q] \mid j = 1, \ldots, t, q \in \{1, 2, \ldots, K\} \rangle$$

Datasets

2moons



linked



concentric