Kernel Methods and Semi-Supervised Learning

Zalán Bodó

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

Szeged, March 2012
Contents

The aims of the short course
   Top 10 data mining algorithms

Machine learning. Supervised learning

Kernel methods
   Kernels
   Centroid method
   kNN
   SVM
   k-means

Semi-supervised learning
   Assumptions in SSL
   Classes of SSL
   Self-training
   Data graphs
   Label propagation
   Graph-based distances
   Semi-supervised kNN and k-means
   Cluster kernels
   Laplacian SVM
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 ⇒ 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
Top 10 data mining algorithms – of all times

1. C4.5
2. k-Means ←
3. SVM ←
4. Apriori
5. EM
6. PageRank (←)
7. AdaBoost
8. kNN ←
9. Naive Bayes
10. CART

- article: http://www.cs.uvm.edu/~icdm/algorithms/10Algorithms-08.pdf
- slides with results: http://www.cs.uvm.edu/~icdm/algorithms/ICDM06-Panel.pdf
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: $D_{\text{train}} = \{(x_i, y_i) \mid i = 1, \ldots, N\}$, $x_i \in X$, $y_i \in Y$
- $X$ is the input space, $x_i$’s are vectors
- $Y$ is the label set, $y_i$’s are scalars
- goal: find $\hat{f} : X \rightarrow Y$ to approximate $f$ given by $D_{\text{train}}$
  - if $|Y| = 2 \Rightarrow$ binary classification
  - if $|Y| > 2 \Rightarrow$ multi-class classification
  - if $f : X \rightarrow Y \Rightarrow$ single-label classification
  - if $f : X \rightarrow 2^Y \Rightarrow$ 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
- linear algorithm → non-linear algorithm
- $\phi$: feature mapping
- kernels: $k(x, z) = \langle \phi(x), \phi(z) \rangle$, $\phi : X \rightarrow \mathcal{H}$, $X \subseteq \mathbb{R}^n$ 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 $\phi$ to a dot product space defines a positive definite kernel by $\langle \phi(x), \phi(z) \rangle$
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: 
\[
\phi(x) = \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}x_1x_2 \end{bmatrix}^T
\]
then the dot product: 
\[
\phi(x)'\phi(z) = x_1^2z_1^2 + 2x_1z_1x_2z_2 + x_2^2z_2^2 = (x'z)^2 = k(x, z)
\]
it is called the second order homogeneous polynomial kernel
Kernel Methods and Semi-Supervised Learning

Zalán Bodó

The aims of the short course
Top 10 data mining algorithms
Machine learning. Supervised learning
Kernel methods
Kernels
Centroid method
kNN
SVM
k-means
Semi-supervised learning
Assumptions in SSL
Classes of SSL
Self-training
Data graphs
Label propagation
Graph-based distances
Semi-supervised kNN and k-means
Cluster kernels
Laplacian SVM

General purpose kernels

- Linear (dot product):
  \[ k(x, z) = \langle x, z \rangle \]

- Polynomial:
  \[ k(x, z) = (a \langle x, z \rangle + b)^c \]

- RBF (Gaussian):
  \[ k(x, z) = \exp \left( -\frac{||x - z||^2}{2\sigma^2} \right) \]

- Sigmoid:
  \[ k(x, z) = \tanh(a \langle x, z \rangle + r) \]
General purpose kernels

- linear (dot product):
  \[ k(x, z) = \langle x, z \rangle \]

- polynomial:
  \[ k(x, z) = (a \langle x, z \rangle + b)^c \]

- RBF (Gaussian):
  \[ k(x, z) = \exp \left( - \frac{\|x - z\|^2}{2\sigma^2} \right) \]

- sigmoid:
  \[ k(x, z) = \tanh(a \langle x, z \rangle + r) \]
**General purpose kernels**

- **linear (dot product):**
  \[ k(x, z) = \langle x, z \rangle \]

- **polynomial:**
  \[ k(x, z) = (a \langle x, z \rangle + b)^c \]

- **RBF (Gaussian):**
  \[ k(x, z) = \exp \left( - \frac{\|x - z\|^2}{2\sigma^2} \right) \]

- **sigmoid:**
  \[ k(x, z) = \tanh(a \langle x, z \rangle + r) \]
General purpose kernels

- linear (dot product):
  \[ k(x, z) = \langle x, z \rangle \]

- polynomial:
  \[ k(x, z) = (a \langle x, z \rangle + b)^c \]

- RBF (Gaussian):
  \[ k(x, z) = \exp \left( -\frac{\|x - z\|^2}{2\sigma^2} \right) \]

- sigmoid:
  \[ k(x, z) = \tanh(a \langle x, z \rangle + r) \]
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 $\tilde{k}$. 
Distances ↔ kernels

1. kernel → (squared) distance:

\[ D^{(2)} = \text{diag}(K)1_{NN} + 1_{NN}\text{diag}(K) - 2K \]

2. distance → kernel:

\[ -\frac{1}{2}JD^{(2)}J = -\frac{1}{2}J\text{diag}(K)1_{NN}J - \frac{1}{2}J1_{NN}\text{diag}(K)'J + JKJ \]

\[ = JKJ \]

where \( J = I - \frac{1}{N}11' = I - \frac{1}{N}1_{NN} \) is the centering matrix (since \( 1_{NN}J = 0 \))
Centroid method

Outline:

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

\[ m_+ = \left| \{ x_i | y_i = +1 \} \right|, \quad m_- = \left| \{ x_i | y_i = -1 \} \right| \]

centers:

\[ c_+ = \frac{1}{m_+} \sum_{\{ i | y_i = +1 \}} x_i \]

\[ c_- = \frac{1}{m_-} \sum_{\{ i | y_i = -1 \}} x_i \]

Let \( w := c_+ - c_- \) and \( c := (c_+ + c_-)/2 \); label will be determined by \( \langle w, x - c \rangle \)

\[ y = \text{sgn} \langle x - c, w \rangle \]

\[ = \text{sgn} \left( \langle c_+, x \rangle - \langle c_-, x \rangle + b \right), \]

where \( b := \left( \| c_- \|^2 - \| c_+ \|^2 \right) / 2 \)
Substituting the corresponding expressions for $c_+$ and $c_-$ we obtain

\[
y = \text{sgn} \left( \frac{1}{m_+} \sum_{\{i|y_i=+1\}} \langle x, x_i \rangle - \frac{1}{m_-} \sum_{\{i|y_i=-1\}} \langle x, x_i \rangle + b \right) = \text{sgn} \left( \frac{1}{m_+} \sum_{\{i|y_i=+1\}} k(x, x_i) - \frac{1}{m_-} \sum_{\{i|y_i=-1\}} k(x, x_i) + b \right),
\]

where

\[
b := \frac{1}{2} \left( \frac{1}{m_-^2} \sum_{\{(i,j)|y_i=y_j=-1\}} k(x_i, x_j) - \frac{1}{m_+^2} \sum_{\{(i,j)|y_i=y_j=+1\}} k(x_i, x_j) \right)
\]

and $k(\cdot, \cdot)$ is an arbitrary kernel function.
For **multi-class** classification:

\[ c_i = \frac{1}{|C_i|} \sum_{x_j \in C_i} x_i \]

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

\[ \|x - z\|^2 = x'x - 2x'z + z'z = k(x, x) - 2k(x, z) + k(z, z) \]

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

\[ \|x - c_i\|^2 = \|x - \frac{1}{|C_i|} \sum_{x_j \in C_i} x_i\|^2 \]

\[ = x'x - \frac{2}{|C_i|} \sum_{x_j \in C_i} x'x_j + \frac{1}{|C_i|^2} \sum_{x_j \in C_i, x_k \in C_i} x'_j x'_k \]

\[ = k(x, x) - \frac{2}{|C_i|} \sum_{x_j \in C_i} k(x, x_j) + \frac{1}{|C_i|^2} \sum_{x_j \in C_i, x_k \in C_i} k(x_j, x_k) \]
k-Nearest Neighbors

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

$$f(x) = \arg\max_{c=1,2,...,K} \sum_{z \in N_k(x), z \in D_{\text{train}}} \sim(z, x) \cdot \delta(c, f(z))$$

(we use $\sim(z, x) = 1, \forall z, x$)

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

(Kronecker delta)

to determine $N_k(x)$ the Euclidean distance is used

as before, we can rewrite the Euclidean distance using dot products:

$$\|x - z\|^2 = x'x - 2x'z + z'z = k(x, x) - 2k(x, z) + k(z, z)$$
Support Vector Machines

Outline:

▶ learn: find a separating hyperplane – with maximal margin – that separates the positive and negative examples

▶ classify: on one side of the hyperplane there are the positive points, on the other side the negative points

▶ large-margin separation: the probability of misclassification of an unseen example to be minimized

▶ this hyperplane is determined only by the points which lie on the marginal hyperplanes → Support Vector Machines (SVMs)

▶ setting: binary classification
Hyperplane:
\[ w'x_i + b \geq 1 \quad \forall x_i \text{ s.t. } y_i = 1 \]
\[ w'x_i + b \leq -1 \quad \forall x_i \text{ s.t. } y_i = -1 \]

Decision function:
\[ f(x) = \text{sgn}(w'x + b) \]

- margin of the separating hyperplane: \( 2/\|w\| \)
- optimization problem:
\[
\min F(w, b) = \frac{1}{2} \|w\|^2 \\
\text{subject to } y_i(w'x_i + b) \geq 1, \quad i = 1, \ldots, \ell
\]
Linearly non-separable case

- introduce slack variables; show the level of violation of linear separability

- new conditions:

\[ w'x_i + b \begin{cases} \geq 1 - \xi_i & \text{if } y_i = 1 \\ \leq -1 + \xi_i & \text{if } y_i = -1 \end{cases} \]

- new optimization problem:

\[
\min F(w, b, \xi) = \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{\ell} \xi_i \\
\text{s.t. } y_i(w'x_i + b) \geq 1 - \xi_i, \; \xi_i \geq 0, \; i = 1, \ldots, \ell
\]
Outline:
Theorem of the short course
Top 10 data mining algorithms
Machine learning
Supervised learning
Kernel methods
Kernels
Centroid method
kNN
SVM
k-means
Semi-supervised learning
Assumptions in SSL
Classes of SSL
Self-training
Data graphs
Label propagation
Graph-based distances
Semi-supervised kNN and k-means
Cluster kernels
Laplacian SVM

- 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:

\[ F = \sum_{i=1}^{K} \sum_{j=1}^{N} U_{ij} d(x_j, c_i) \]

where

- \( c_i, i = 1, \ldots, K \) are the cluster centers
- \( U \) is the membership matrix of size \( K \times N \) (\( K \) clusters, \( N \) points)
- \( U_{ij} \in \{0, 1\}; \ U_{ij} = 1 \) if \( x_j \) belongs to \( C_i \) (\( i \)-th cluster); otherwise 0
- \( d(\cdot, \cdot) \) is a distance, usually the squared Euclidean:
  \[ d(x, z) = \|x - z\|^2 \]
- \( F \) is minimized if the points are assigned to the closest cluster center, provided they are fixed:
  \[ U_{ij} = \begin{cases} 1, & \text{if } \|x_j - c_i\|^2 \leq \|x_j - c_k\|^2, \forall k \neq i \\ 0, & \text{otherwise} \end{cases} \]
- on the other side, if \( U \) is fixed, \( F \) is minimized by taking the centers of the groups:
  \[ c_i = \frac{1}{\sum_{j=1}^{N} U_{ij}} \sum_{k=1}^{N} U_{ik} x_k \]
k-Means

1. Initialize the centers.
2. Determine $U$.
3. Compute cost function $F$.
4. If converged, then STOP.
   Else update centers using the new $U$; GOTO step 2.
Fuzzy c-means:
  
  - generalization of k-means
  - cluster membership values are $\in [0, 1]$
  - objective function:

$$F' = \sum_{i=1}^{K} \sum_{j=1}^{N} U_{ij}^p \|x_j - c_i\|^2$$

such that $U_{ij} \in [0, 1]$, $\sum_{i=1}^{K} U_{ij} = 1, \forall j, \ p \in [1, \infty)$

- if the centers are fixed, $F'$ is minimized by

$$U_{ij} = \frac{1}{\sum_{k=1}^{K} \left( \frac{\|x_j - c_i\|}{\|x_j - c_k\|} \right)^{N-1}}$$

- the values which minimize $F'$ for fixed $U$ are

$$c_i = \frac{1}{\sum_{j=1}^{N} \frac{U_{ij}^p}{\sum_{i=1}^{N} U_{ij}^p}} \sum_{j=1}^{N} U_{ij}^p x_j$$
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:

\[
\| \phi(x) - \frac{1}{s_j} \sum_{k=1}^{N} U_{jk} \phi(x_k) \|^2
\]

\[
= k(x, x) + \frac{1}{s_j^2} \sum_{k, \ell=1}^{N} U_{jk} U_{j\ell} k(x_k, x_\ell) - \frac{2}{s_j} \sum_{k=1}^{N} U_{jk} k(x_k, x)
\]

where \( s_j \) is the size of the \( j \)-th cluster
Kernel k-means

1. Generate a random $U$ which satisfies the conditions, that is make a random assignment of the points.

2. Compute the cost function $F$.

3. If converged then STOP. Else update $U$ using the update formula used in k-means ($=$ get new $U$); GOTO step 2.
Semi-supervised learning (SSL)

- **supervised** learning:
  \[ D = \{ (x_i, y_i) | x_i \in X \subseteq \mathbb{R}^d, y_i \in \{-1, +1\}, \; i = 1, \ldots, \ell \}; \]
  find \( f : X \rightarrow \{-1, +1\} \) which agrees with \( D \)

- **semi-supervised** learning:
  \[ D = \{ (x_i, y_i) | i = 1, \ldots, \ell \} \cup \{ x_j | j = \ell + 1, \ldots, \ell + u \}, \]
  \( \ell \ll u \), \( N = \ell + u \);
  - **inductive**: find \( f : X \rightarrow \{-1, +1\} \) which agrees with \( D \) +
    use the information of \( D_U \)
  - **transductive**: find \( f : D_U \rightarrow \{-1, +1\} \) by using \( D = D_L \cup D_U \)
Semi-supervised learning (SSL)

- **supervised** learning:
  \[ D = \{(x_i, y_i) \mid x_i \in X \subseteq \mathbb{R}^d, y_i \in \{-1, +1\}, i = 1, \ldots, \ell\}; \]
  find \( f : X \rightarrow \{-1, +1\} \) which agrees with \( D \)

- **semi-supervised** learning:
  \[ D = \{(x_i, y_i) \mid i = 1, \ldots, \ell\} \cup \{x_j \mid j = \ell + 1, \ldots, \ell + u\}, \]
  \( \ell \ll u, N = \ell + u; \)
  - **inductive**: find \( f : X \rightarrow \{-1, +1\} \) which agrees with \( D \) + use the information of \( D_U \)
  - **transductive**: find \( f : D_U \rightarrow \{-1, +1\} \) by using \( D = D_L \cup D_U \)
Assumptions in SSL

- **smoothness assumption**: If two points \( x_i \) and \( x_j \) 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 ⇒ smoothness assumption.
Assumptions in SSL

► **smoothness assumption**: If two points \( x_i \) and \( x_j \) 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 \( x_i \) and \( x_j \) 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 ⇒ smoothness assumption
Classes of SSL

- Generative models
- Low-density separation
- Graph-based methods
- Change of representation
Classes of SSL

- Generative models
- Low-density separation
- Graph-based methods
- Change of representation

- model class conditional density $P(x|y)$ and class priors $P(y)$ and use the Bayes theorem for calculating posteriors, which are used for classification

- **e.g.** Naive Bayes + EM
  1. Train the classifier on the training examples; determine probabilities $P(d_i | c_j)$ and $P(c_j)$.
  2. Repeat while there is improvement:
     - **E-step**: Classify unlabeled examples using the trained classifier.
     - **M-step**: Re-estimate the classifier using the previously classified examples; recalculate $P(d_i | c_j)$ and $P(c_j)$. 
Classes of SSL

- Generative models
- Low-density separation
- Graph-based methods
- Change of representation

push away a decision boundary from labeled and unlabeled data; natural choice: use a large-margin classifier

**e.g.** Transductive SVM (TSVM)

\[
\begin{align*}
\min & \quad \frac{1}{2} \|w\|^2 \\
\text{s.t.} & \quad y_i (w' x_i + b) \geq 1, \quad i = 1, \ldots, \ell \\
& \quad y_j (w' x_j + b) \geq 1, \quad j = \ell + 1, \ldots, N \\
& \quad y_j \in \{-1, +1\}, \quad j = \ell + 1, \ldots, N
\end{align*}
\]
Classes of SSL

- Generative models
- Low-density separation
- Graph-based methods
- Change of representation

- use the graph of the labeled and unlabeled data, represented by weight matrix $W$; use graph Laplacian $L - W$

- e.g. Label Propagation
  1. Compute $Y(t + 1) = P Y(t)$.
  2. Reset the labeled data, $Y_L(t + 1) = Y_L(0)$.
  3. $t = t + 1$ and repeat the above steps until convergence.
Classes of SSL

- Generative models
- Low-density separation
- Graph-based methods
- Change of representation

- find some structure in the whole data set; use the information provided by the labeled and unlabeled data set to create a new representation
  1. Build the new representation – new distance, dot-product or kernel – of the learning examples.
  2. Use a supervised learning method for obtaining the decision function employing the new representation obtained in the previous step.

- e.g. PCA, KPCA, LLE, ISOMAP, etc.
Self-training/Bootstrapping

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.
Data graphs

- $G = (V, W)$ undirected weighted graph (usually)
- $V = X_L \cup X_U$ (training data)
- $W : V \times V \rightarrow \mathbb{R}$ (similarity function);
  $W(x_i, x_j) = \text{sim}(x_i, x_j) \quad (\text{def } W_{ij})$
  for example:
  $\text{sim}(x_i, x_j) = \exp \left( -\frac{||x_i-x_j||^2}{\sigma^2} \right)$
- sparsification techniques: $\varepsilon$NN, $k$NN graphs
- graph Laplacian: $L = D - W$, $D = \text{diag}(W \cdot 1)$
  - nice properties: e.g. positive semi-definite; has as many 0 eigenvalues as connected components contains; etc.

Data graphs

$G = (V, W)$ undirected weighted graph (usually)

$V = X_L \cup X_U$ (training data)

$W : V \times V \rightarrow \mathbb{R}$ (similarity function);

$W(x_i, x_j) = \text{sim}(x_i, x_j) \quad (\text{def } W_{ij})$

for example:

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

sparsification techniques: $\varepsilon$NN, $k$NN graphs

graph Laplacian: $L = D - W$, $D = \text{diag}(W \cdot 1)$

- nice properties: e.g. positive semi-definite; has as many 0 eigenvalues as connected components contains; etc.
Label propagation

- build data graph: $W_{ij} = \exp \left(-\frac{1}{2\sigma^2}||x_i - x_j||^2\right)$
- compute transition probabilities: $D = \text{diag}(W 1), P = D^{-1} W$
- vector of labels: $f = [f_L f_U]'$

LP

1. $i = 0$; Initialize $f_U^{(i)}$.
2. $f^{(i+1)} = Pf^{(i)}$.
3. Clamp the labeled data, $f_L^{(i+1)} = f_L$.
4. $i = i + 1$; Unless convergence go to step 2.
can be rewritten as

\[ f_i = \frac{1}{\sum_{j=1}^{N} W_{ij} \sum_{k=1}^{N} W_{ik} f_k} \]

exact solution (\( L \) – graph Laplacian)

\[ f_U = -L_{UU}^{-1} \cdot L_{UL} \cdot f_L \]

minimizes

\[ E(f) = \frac{1}{2} \sum_{i,j=1}^{N} W_{ij} (f_i - f_j)^2 = f' L f \]

LP is equivalent with searching for a minimum cut in the data graph
Transduction $\rightarrow$ induction

- fix the other labels and compute the new one
- we minimize:
  \[ C + \sum_i W_{iy}(f_i - y)^2 \]
- from setting the derivative to zero we obtain:
  \[ y = \frac{\sum_i W_{iy}f_i}{\sum_j W_{jy}} \]
Graph-based distances

- 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.

![Graph](image)

**Floyd–Warshall algorithm**

1. $D_{ij} = $ weight of the edge; if no edge then $\infty$
2. for $k=1:N$
   - for $i=1:N$
     - for $j=1:N$
       - if $D_{ij} > D_{ik} + D_{kj}$
         - $D_{ij} = D_{ik} + D_{kj}$
Semi-Supervised kNN and k-Means

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

1. Initial distance matrix $\Leftarrow$ kNN/$\varepsilon$NN 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 kNN/k-Means using these graph distances.
Cluster kernels

**Neighborhood kernel**

- representing each as the average of its neighbors → neighborhood structure influences representation:

\[
\phi_{\text{nbd}}(x) = \frac{1}{|N(x)|} \sum_{x' \in N(x)} \phi_b(x)
\]

- the kernel:

\[
k_{\text{nbd}}(x, z) = \frac{\sum_{x' \in N(x), z' \in N(z)} k_b(x', z')}{|N(x)||N(z)|}
\]
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 → affects the output of the algorithm

**BCK**

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

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

The final kernel:

$$k(x, z) = k_b(x, z) \cdot k_{\text{bag}}(x, z)$$
Laplacian SVM

- use SVM + semi-supervised (smoothness) assumption
- add the following member to the objective function:

$$\gamma_I \frac{1}{2} \sum_{i,j=1}^{N} w_{ij} (f_i - f_j)^2 = \gamma_I \mathbf{f}' L \mathbf{f}$$

where $L$ is the graph Laplacian, $\mathbf{f}$ is the $N \times 1$ vector containing the labels of the points;
$\gamma_I$ is a parameter determining how much the unlabeled points influence the result/kernel
- the optimization problem thus becomes:

$$\min_{\mathbf{w}, b} \quad \frac{1}{2} \| \mathbf{w} \|^2 + C \sum_{i=1}^{\ell} \xi_i + \gamma_I \mathbf{f}' L \mathbf{f}$$

s.t. $y_i (\mathbf{w}' \mathbf{x}_i + b) \geq 1 - \xi_i$
it can be shown that it is the same problem which is solved in the original SVM formulation, but now using the kernel

\[
\hat{k}(x, z) = k(x, z) - k'_x \left( \frac{1}{2\gamma} I + LK_{NN} \right)^{-1} Lk_z
\]

where \(\hat{k}(\cdot, \cdot)\) is the new semi-supervised kernel, while \(k(\cdot, \cdot)\) is the base kernel.
Thank you! Questions?