Semi-Supervised Learning

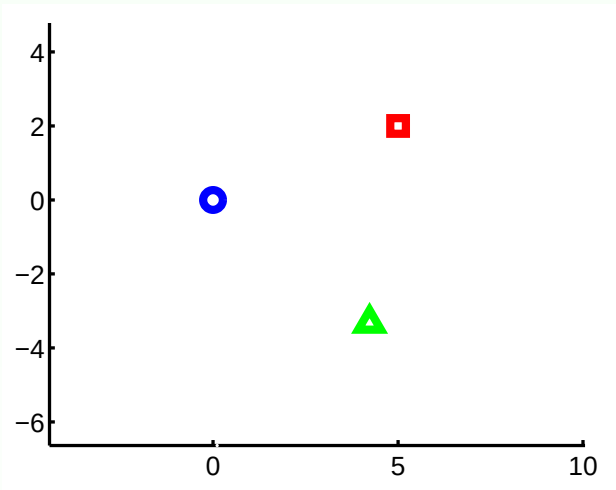
Zalán Bodó

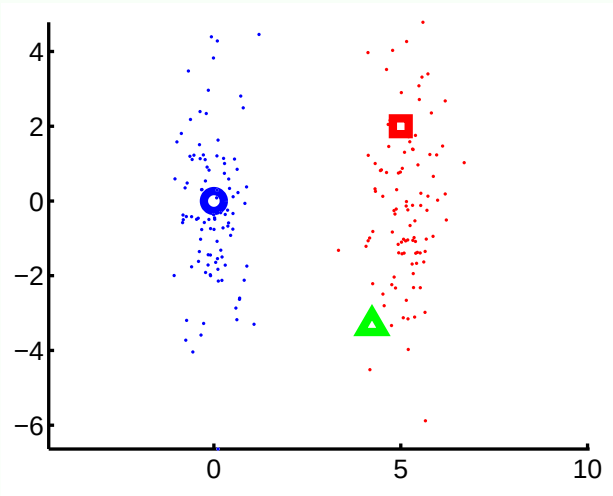
Babeș–Bolyai University
Faculty of Mathematics and Computer Science

Outline

Supervised and Semi-Supervised Learning

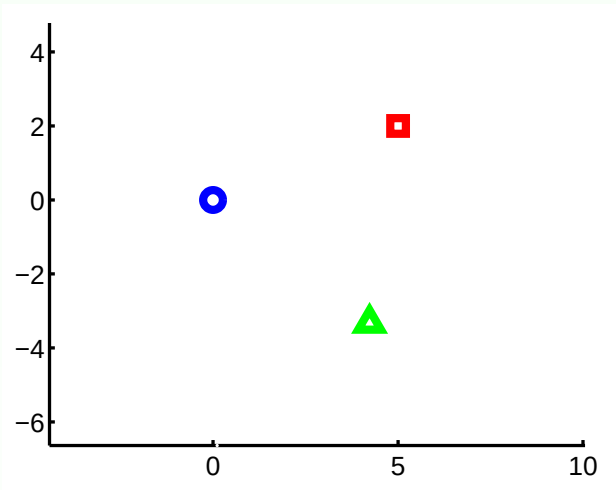
- supervised learning:
  $D = \{(\boldsymbol{x}_i, y_i) \,|\, \boldsymbol{x}_i \in X \subseteq \mathbb{R}^d, y_i \in \{-1, +1\}, i = 1, \ldots, \ell\}$;
  find $f : X \to \{-1, +1\}$ which agrees with D
- semi-supervised learning:
  $D = \{(\boldsymbol{x}_i, y_i) \,|\, i = 1, \ldots, \ell\} \cup \{\boldsymbol{x}_j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

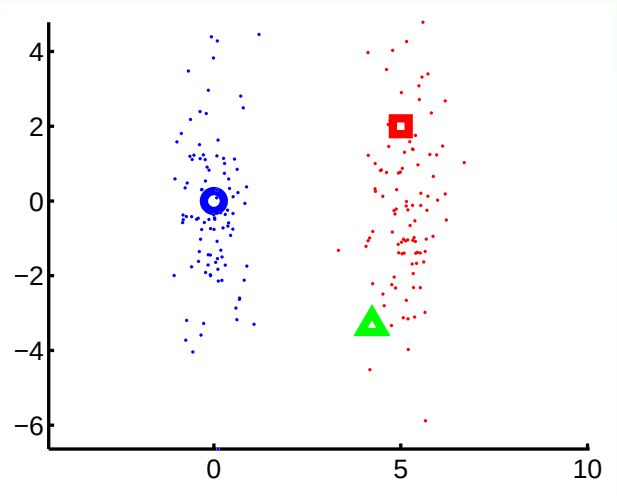




Supervised and Semi-Supervised Learning

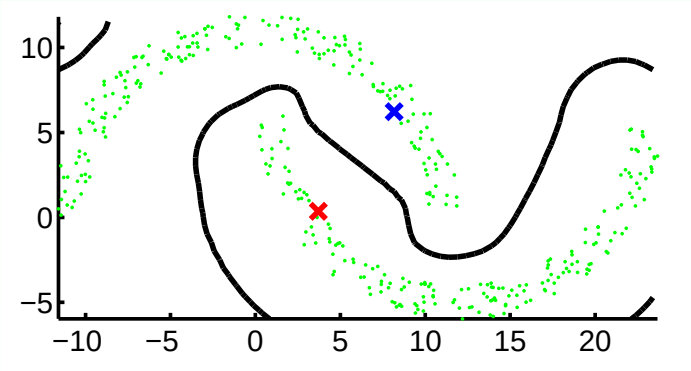
- supervised learning:
  $D = \{(\boldsymbol{x}_i, y_i) \,|\, \boldsymbol{x}_i \in X \subseteq \mathbb{R}^d, y_i \in \{-1, +1\}, i = 1, \ldots, \ell\}$;
  find $f : X \to \{-1, +1\}$ which agrees with D
- semi-supervised learning:
  $D = \{(\boldsymbol{x}_i, y_i) \,|\, i = 1, \ldots, \ell\} \cup \{\boldsymbol{x}_j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





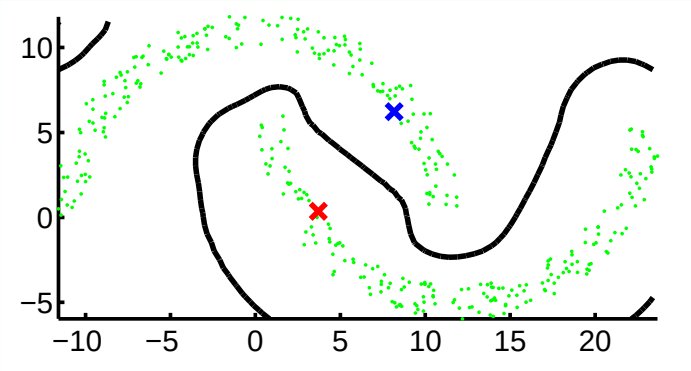
Assumptions in SSL

- **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



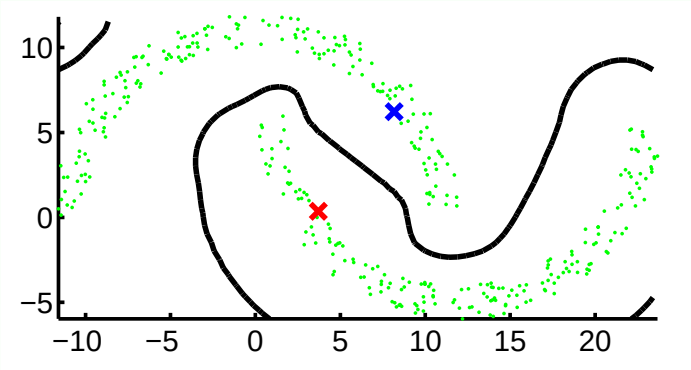
Assumptions in SSL

- **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



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 $P(\boldsymbol{x}|y)$ 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 $P(\boldsymbol{d}_i \mid c_j)$ 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 $P(\boldsymbol{d}_i \mid c_j)$ 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)

$$
\begin{aligned}
\min \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 \\
\text{s.t.} \quad & y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) \geq 1, \quad i = 1, \ldots, \ell \\
& y_j(\boldsymbol{w}'\boldsymbol{x}_j + b) \geq 1, \quad j = \ell + 1, \ldots, N \\
& y_j \in \{-1, +1\}, \quad j = \ell + 1, \ldots, N
\end{aligned}
$$

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 $\boldsymbol{W}$; use graph Laplacian $\boldsymbol{L} - \boldsymbol{W}$
- **e.g.** Label Propagation
  1. Compute $\boldsymbol{Y}(t+1) = \boldsymbol{P}\,\boldsymbol{Y}(t)$.
  2. Reset the labeled data, $\boldsymbol{Y}_L(t+1) = \boldsymbol{Y}_L(0)$.
  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**

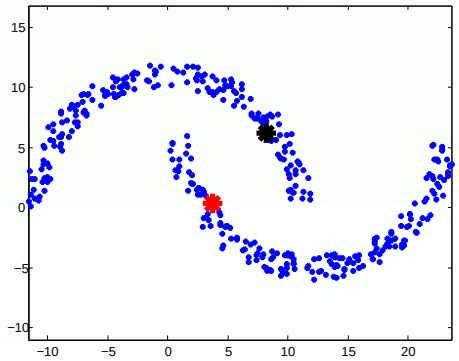
- find some structure in the whole data set; use the information provided by the labeled and unlabeled data set to create a new representation
  1. Build the new representation – new distance, dot-product or kernel – of the learning examples.
  2. Use a supervised learning method for obtaining the decision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

Datasets

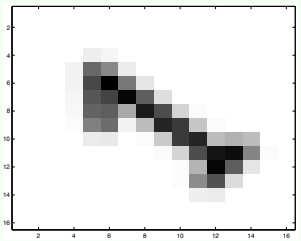
- book: Chapelle et al. *Semi-Supervised Learning*. MIT Press, 2006.
- link: `http://olivier.chapelle.cc/ssl-book/`
  - benchmark datasets: `http://olivier.chapelle.cc/ssl-book/benchmarks.html`
  - *Analysis of Benchmarks* (Ch. 21): `http://olivier.chapelle.cc/ssl-book/benchmarks.pdf`

Datasets we use:

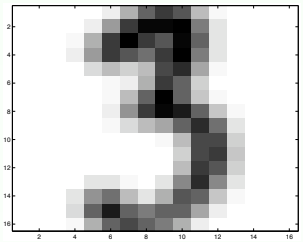
1. **Two moons** – two half moons with two labeled points; artificial dataset; $d = 2$

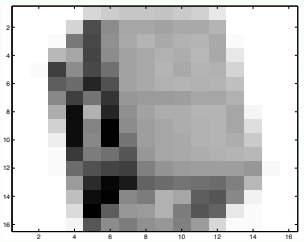


2. **Digit1** – artificial writings of the digit 1; $d = 241$, $N = 1500$; the classes were formed according to the tilt angle of the digit.



3. **USPS** (United States Postal Service) – digits 2 and 5 vs rest; $d = 241$, $N = 1500$
4. **COIL2** (Columbia Object Image Library) – 16×16 pixels images; $d = 241$, $N = 1500$





5. **Text** – derived from 20Newsgroups; category `comp.sys.ibm.pc.hardware` vs remaining `comp.*` categories; representation: *tfidf* weighting; $d = 11960$, $N = 1500$

Data set	Classes	Dimension	Points	Comment
Two moons	2	2	$2 + m$	artificial
USPS	2	241	1500	imbalanced
Digit1	2	241	1500	artificial
COIL2	2	241	1500	
Text	2	11 960	1500	sparse discrete

- binary labels
- two formats: text and Matlab
- 12 splits of each dataset:
  - there are 12 splits with 10 and 100 labeled data
  - the splits are defined by the labeled (and/or unlabeled) indices

Evaluation Metrics

- **accuracy**:

$$A = \sum_{i=1}^{t} \frac{I(f(x_i) == y_i)}{t}$$

- **error**:

$$E = \sum_{i=1}^{t} \frac{I(f(x_i)! = y_i)}{t} = 1 - A$$

1. training accuracy/error: $x_i \in D_L$
2. test accuracy/error: $x_i \in D_U$ (however the test set can be entirely different from D_U for non-transductive settings)

We will be interested in **test error**.

- average/expected error:

$$\overline{E} = \frac{1}{s} \sum_{i=1}^{s} E_i$$

- standard deviation:

$$\sigma(E) = \sqrt{\frac{1}{s-1} \sum_{i=1}^{s} (E_i - \overline{E})^2}$$

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.

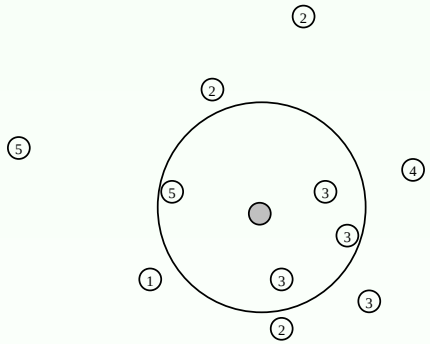
Final prediction: weighted majority vote among all the learners.

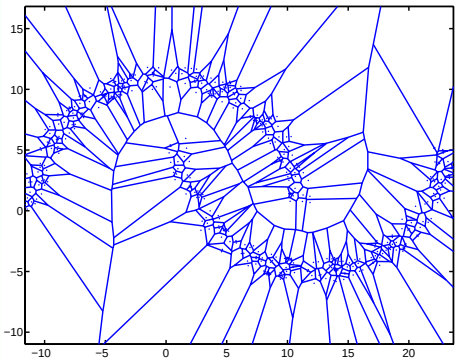
kNN

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

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

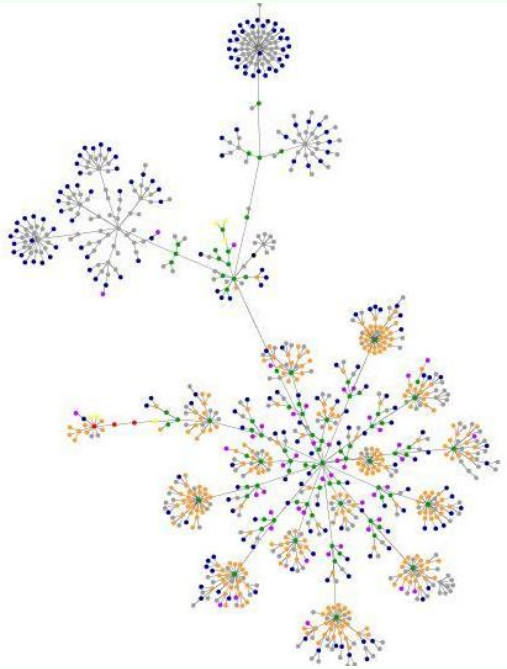
we use $\operatorname{sim}(\boldsymbol{z}, \boldsymbol{x}) = 1, \forall \boldsymbol{z}, \boldsymbol{x}$





Data graphs

- $G = (V, W)$ undirected weighted graph (usually)
- $V = X_L \cup X_U$ (training data)
- $W : V \times V \to \mathbb{R}$ (similarity function); $W(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathrm{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j) \quad (\stackrel{\text{def}}{=} W_{ij})$ for example:
  $\mathrm{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2}{\sigma^2}\right)$
- sparsification techniques: εNN, kNN graphs
- graph Laplacian: $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$, $\boldsymbol{D} = \boldsymbol{W} \cdot \boldsymbol{1}$
  - nice properties: e.g. positive semi-definite; has as many 0 eigenvalues as connected components contains; etc.

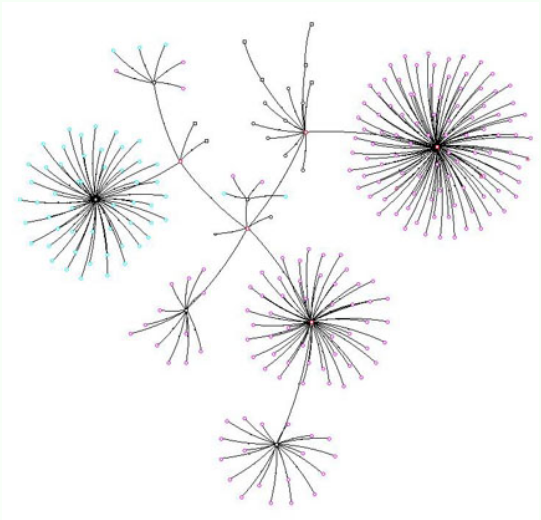


Label Propagation

- build data graph: $W_{ij} = \exp\left(-\frac{1}{2\sigma^2}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\right)$
- compute transition probabilities: $\boldsymbol{D} = \mathrm{diag}(\boldsymbol{W}\boldsymbol{1})$, $\boldsymbol{P} = \boldsymbol{D}^{-1}\boldsymbol{W}$
- vector of labels: $\boldsymbol{f} = [\boldsymbol{f}_L\ \boldsymbol{f}_U]'$

LP

1. $i = 0$; Initialize $\boldsymbol{f}_U^{(i)}$.
2. $\boldsymbol{f}^{(i+1)} = \boldsymbol{P}\boldsymbol{f}^{(i)}$.
3. Clamp the labeled data, $\boldsymbol{f}_L^{(i+1)} = \boldsymbol{f}_L$.
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 ($\boldsymbol{L}$ – graph Laplacian)

$$\boldsymbol{f}_U = -\boldsymbol{L}_{UU}^{-1} \cdot \boldsymbol{L}_{UL} \cdot \boldsymbol{f}_L$$

- minimizes

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

- LP is equivalent with searching for a minimum cut in the data graph

Transduction → induction

- fix the other labels and compute the new one
- we minimize:

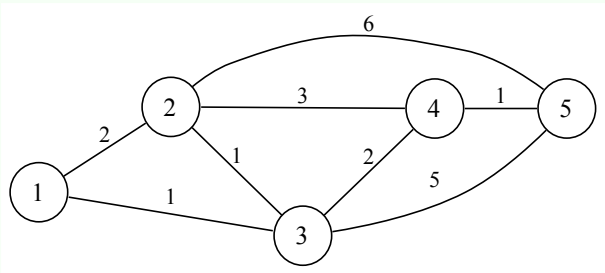
$$C + \sum_i W_{iy}(f_i - y)^2$$

- 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.



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 $D_{ij} > D_{ik} + D_{kj}$
   $D_{ij} = D_{ik} + D_{kj}$

Graph-based/Semi-Supervised kNN

1. Initial distance matrix $\Leftarrow$ kNN/εNN 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 using these graph distances.

Exercises

Ex 1.

Implement bootstrapping with kNN (in Matlab) and test the program with the *two-moons* dataset. Compute the test error to evaluate the system. Test for different values of k.

Confidence: $\frac{|\text{predicted label 1}|}{|\text{predicted label 2}|} \geq 4$ (= at least 80% of the neighbors are predicting the same label).

Convergence:

Ex 2.

Use the previous program to compare supervised kNN and semi-supervised kNN using the *two-moons* dataset. Use the same evaluation metrics as in the previous exercise.

Ex 3.

Use the previous program to compare supervised kNN and semi-supervised kNN using the *USPS* dataset with 100 labeled points.
For evaluation use error and use all the 12 splits to compute the average error (expectation) and standard deviation using 10 labeled points.

Ex 4.

Implement semi-supervised kNN and try it for the *two-moons* dataset.
Use different schemes (kNN, εNN) for building the graph.

Ex 5.

Implement Label Propagation (in Matlab) and test the program using the *two-moons* dataset. Try to use different similarity metrics (Gaussian, dot product etc.) and different graph construction methods.

Ex 6.

Try LP for the other datasets.

Ex 7.

Using the shortest path algorithm, implement the graph-based kNN algorithm and test it on several datasets.