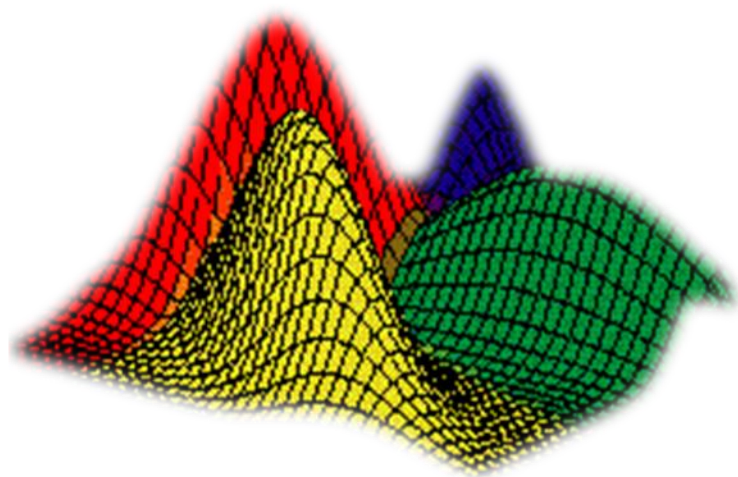
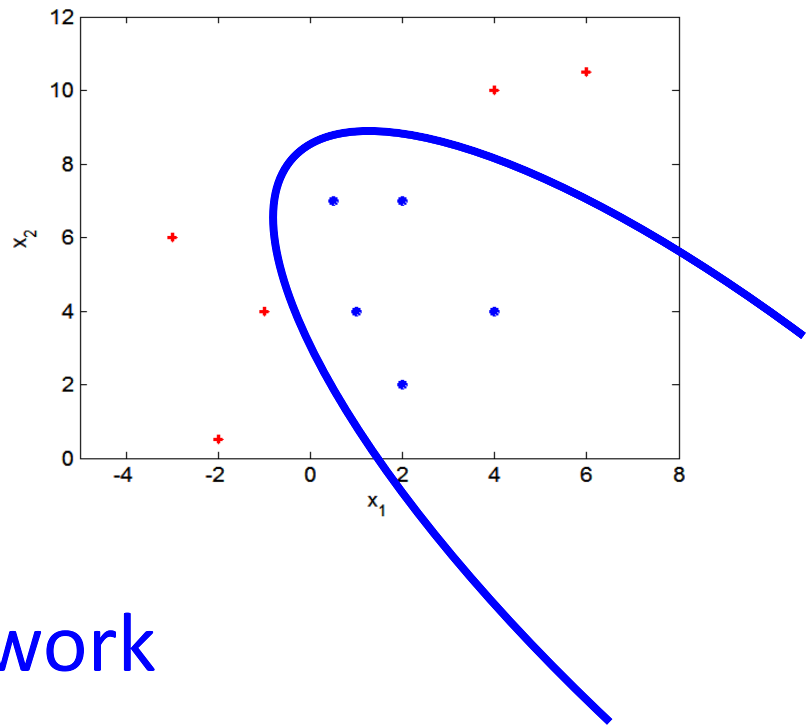


CSE 473
Pattern Recognition

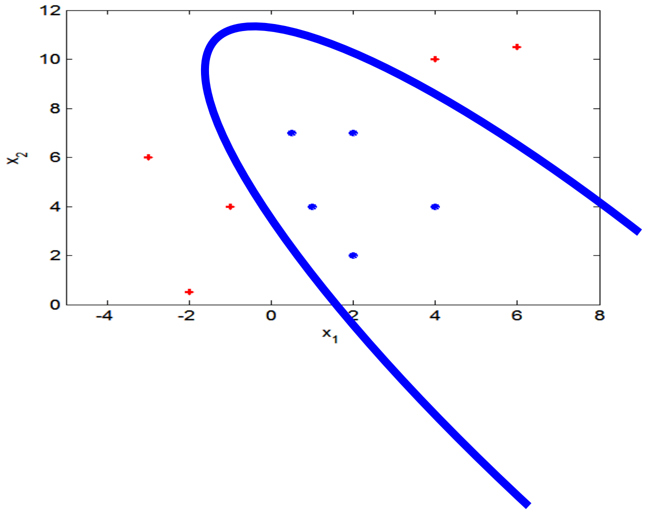
Non-linear Classifiers

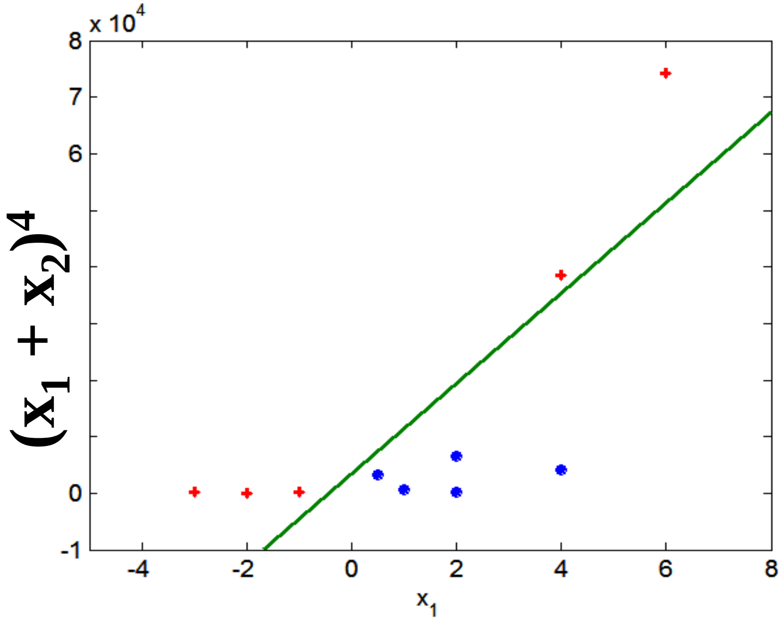


- Neural Network
- Decision Tree
- Non-linear SVM

Non-linear SVM

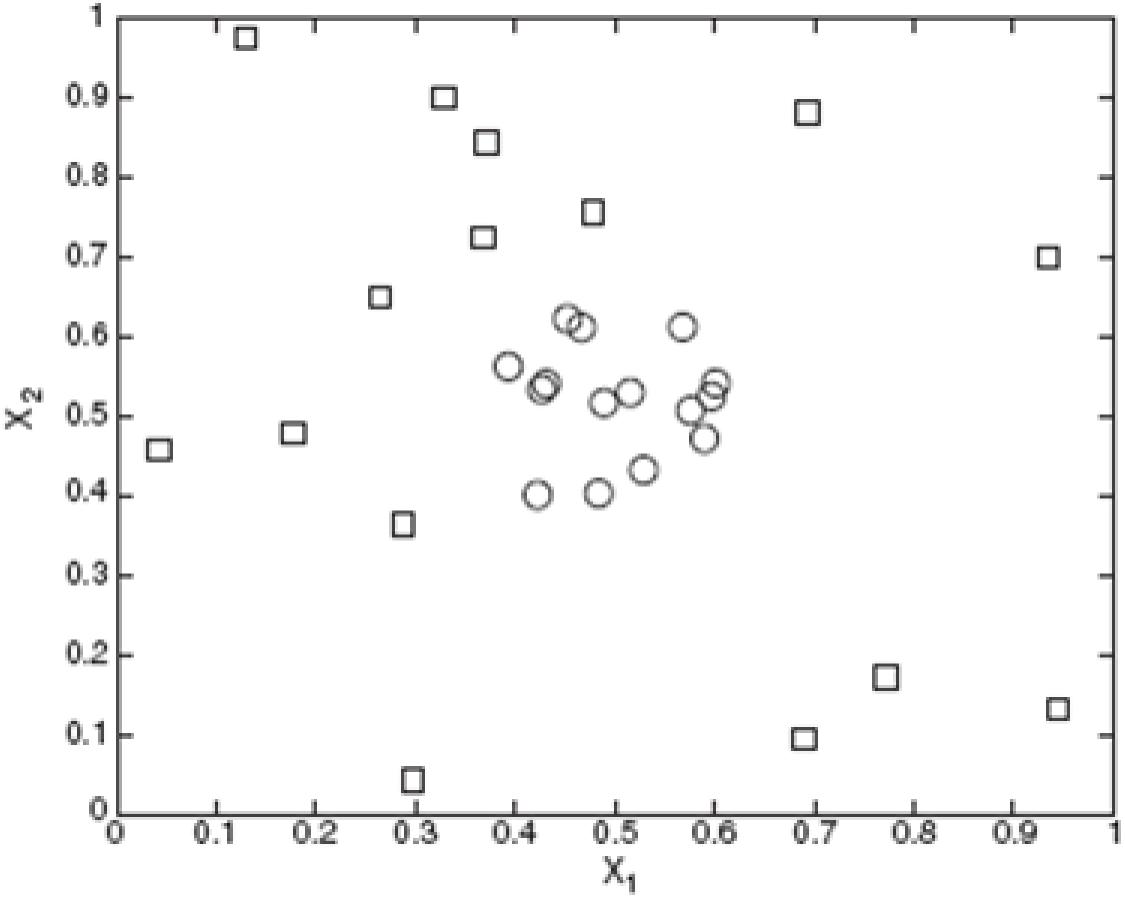
- Transform data into a different (possibly higher) dimensional space





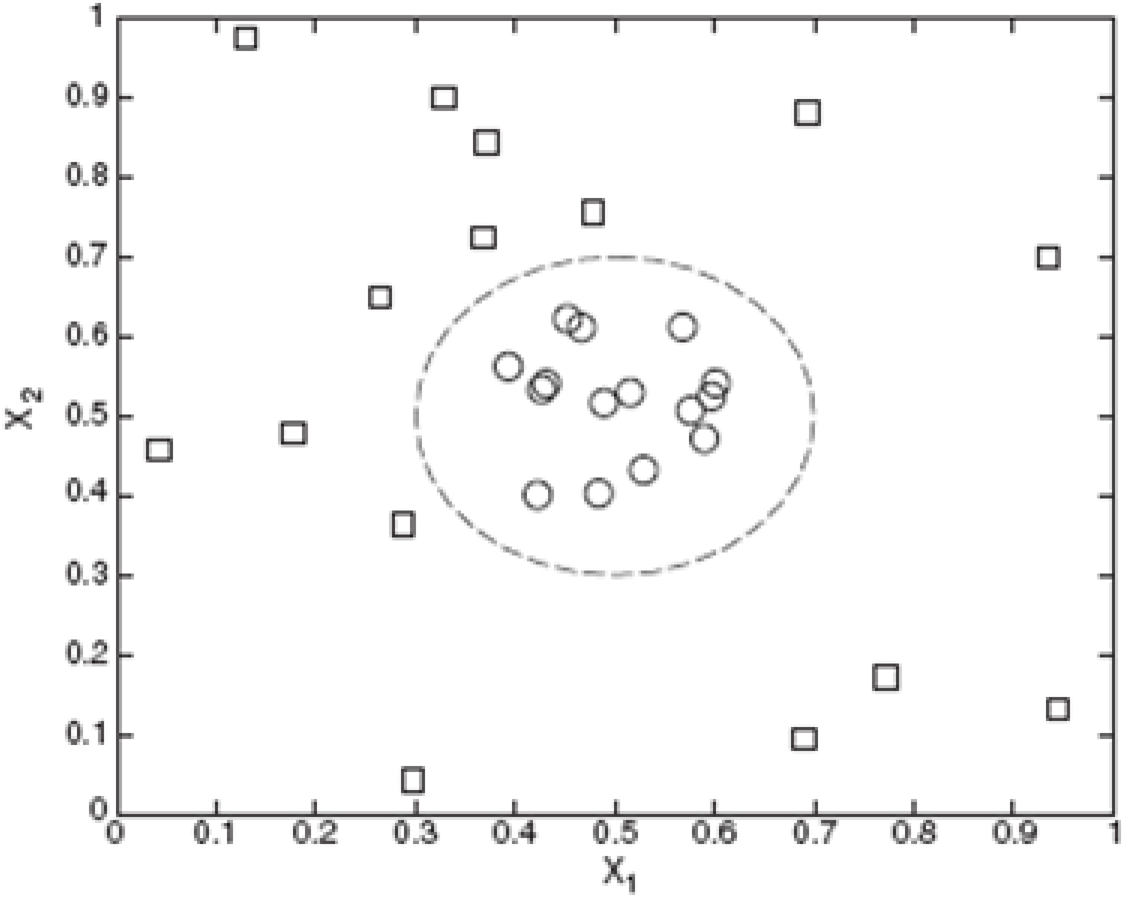
Non-linear SVM

- Another Example



Non-linear SVM

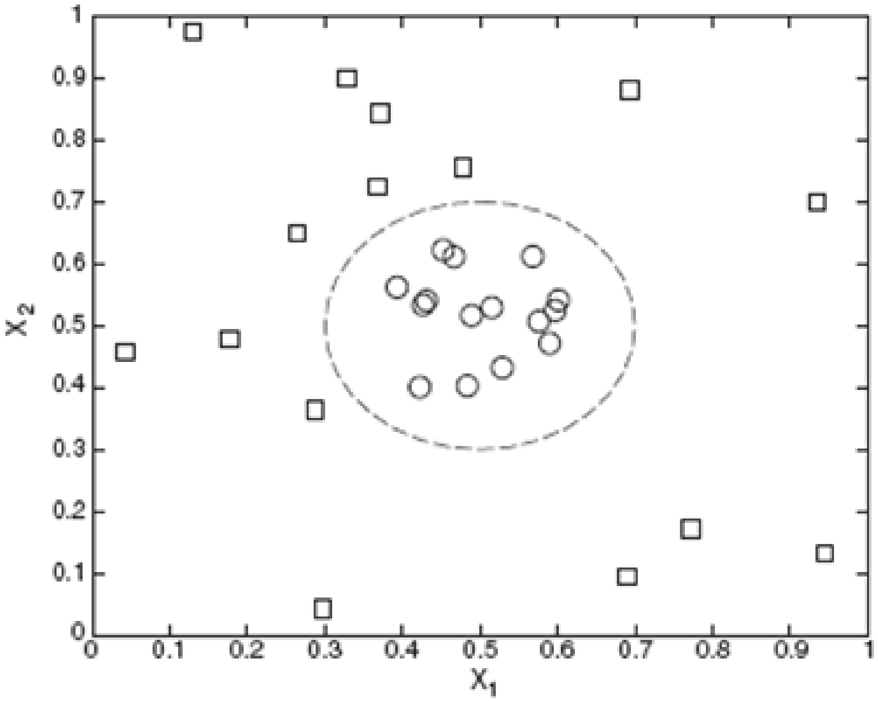
- Another Example



Non-linear SVM

- Decision boundary:

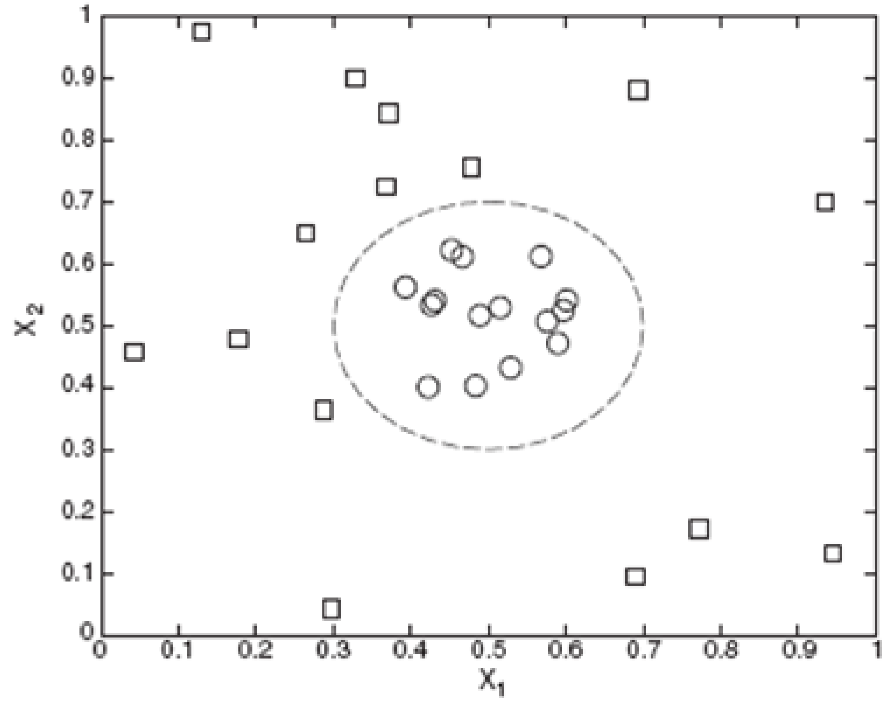
$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$



Non-linear SVM

- Decision boundary:

$$\sqrt{(x_1-0.5)^2+(x_2-0.5)^2}=0.2$$



- 2 classes are defined as

$$y(x_1,x_2)=\begin{cases}1, & \text{if } \sqrt{(x_1-0.5)^2+(x_2-0.5)^2}>0.2\\ -1, & \text{otherwise}\end{cases}$$

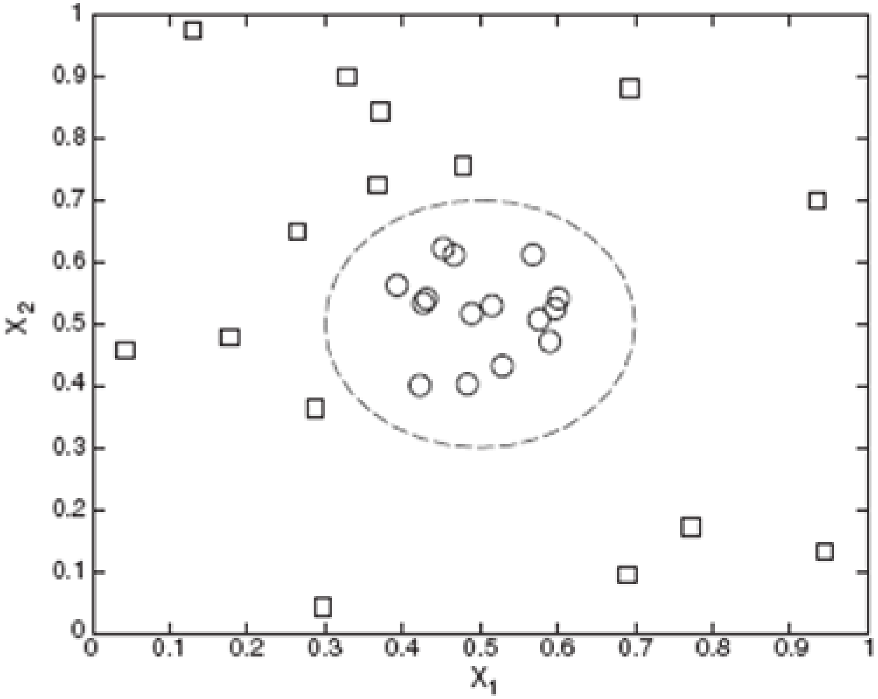
Non-linear SVM

- Decision boundary:

$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$

can be written as

$$x_1^2 - x_1 + x_2^2 - x_2 = -0.46$$



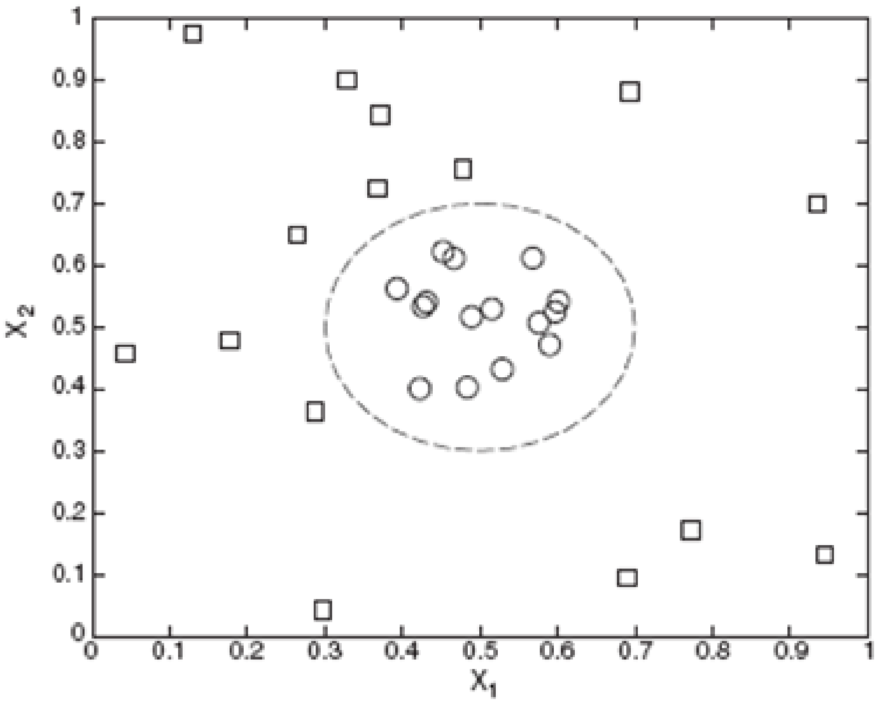
Non-linear SVM

- Decision boundary:

$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$

can be written as

$$\underbrace{x_1^2 - x_1}_{y_1} + \underbrace{x_2^2 - x_2}_{y_2} = -0.46$$



Non-linear SVM

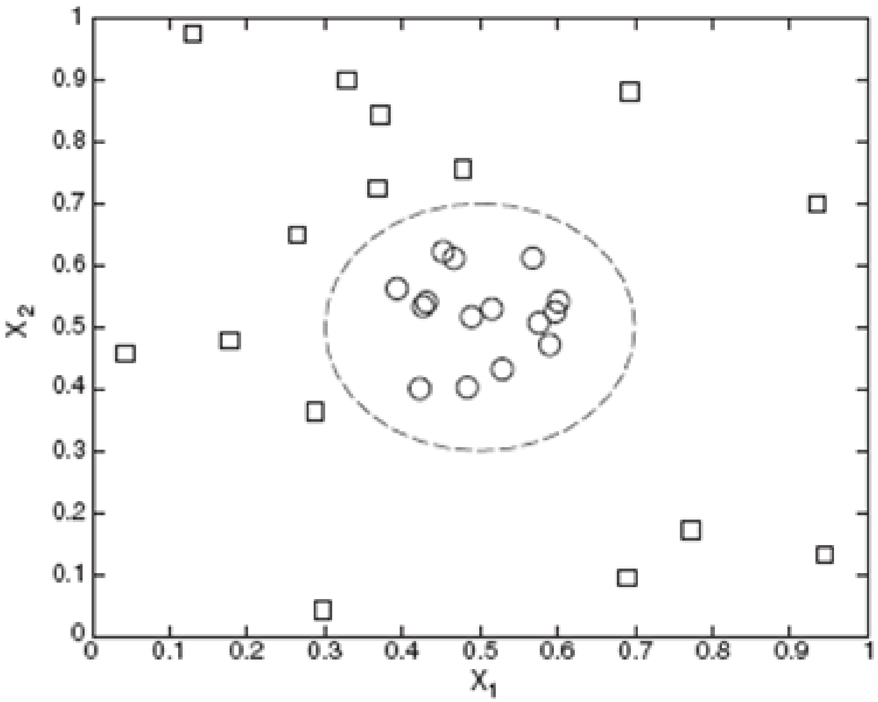
- Decision boundary:

$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$

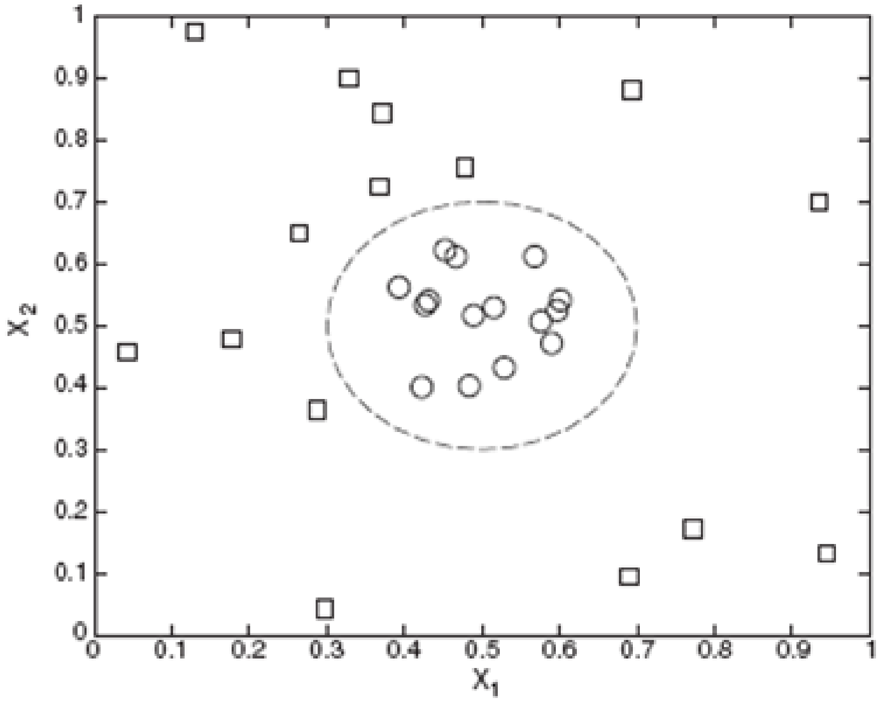
can be written as

$$\underbrace{x_1^2 - x_1}_{y_1} + \underbrace{x_2^2 - x_2}_{y_2} = -0.46$$

$$y_1 + y_2 = -0.46$$

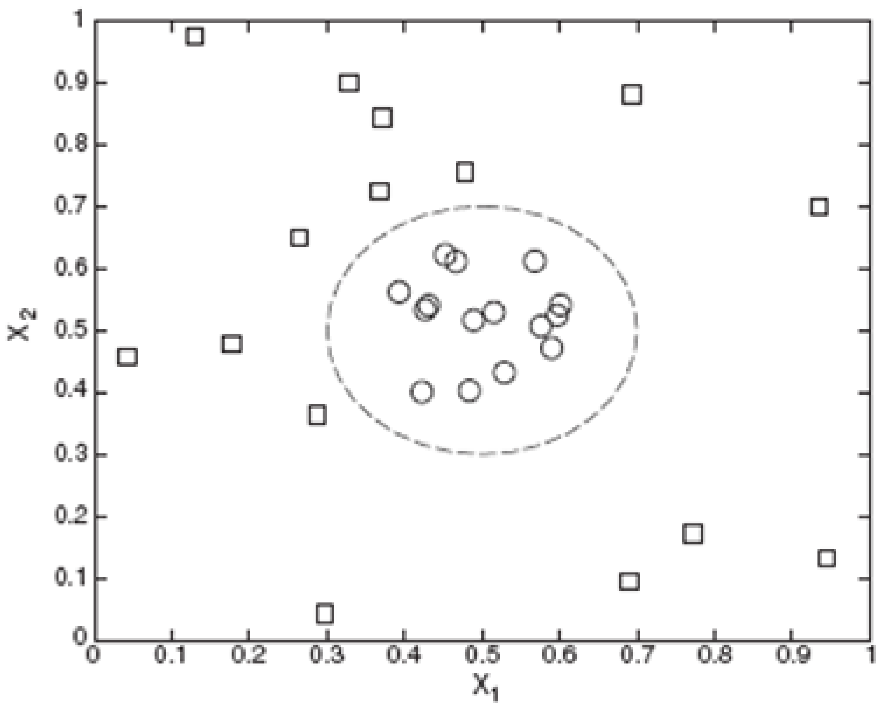


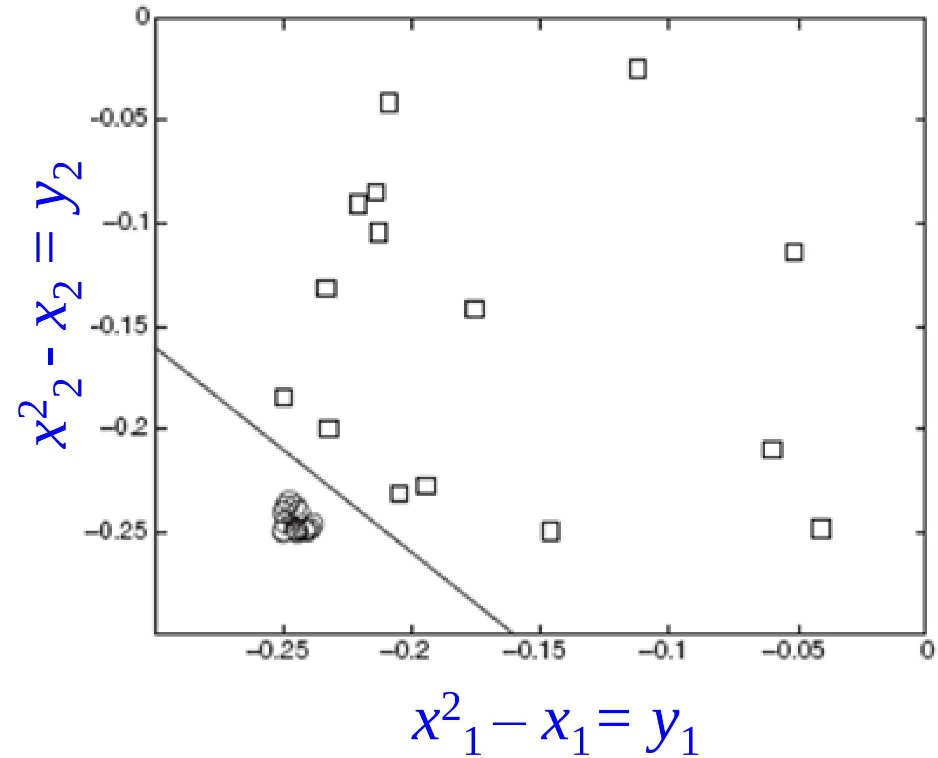
Non-linear SVM



$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$

Non-linear SVM





$$\sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2} = 0.2$$

$$x_1^2 - x_1 + x_2^2 - x_2 = -0.46$$

OR, $$y_1 + y_2 = -0.46$$

Non-linear SVM

- We need a transformation like this

$$\Phi : (x_1, x_2) \rightarrow (x_1^2 - x_1, x_2^2 - x_2)$$

Non-linear SVM

- We need a transformation like this

$$\Phi : (x_1, x_2) \rightarrow (x_1^2 - x_1, x_2^2 - x_2)$$

OR,

$$\Phi : (x_1, x_2) \rightarrow (y_1, y_2)$$

Non-linear SVM

- We need a transformation like this

$$\Phi : (x_1, x_2) \rightarrow (x_1^2 - x_1, x_2^2 - x_2)$$

OR,

$$\Phi : (x_1, x_2) \rightarrow (y_1, y_2)$$

OR,

$$\Phi : \mathbf{x} \rightarrow \mathbf{y}$$

Non-linear SVM

- We need a transformation like this

$$\Phi : (x_1, x_2) \rightarrow (x_1^2 - x_1, x_2^2 - x_2)$$

- OR, more generally:

$$\Phi : (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, 1)$$

Non-linear SVM

- With the transform

$$\Phi : (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, 1)$$

The equation of the classifier will be of the form:

$$w_5 x_1^2 + w_4 x_2^2 + w_3\sqrt{2}x_1 + w_2\sqrt{2}x_2 + w_1\sqrt{2}x_1x_2 + w_0 1 = 0$$

Non-linear SVM

- With the transform

$$\Phi : (x_1, x_2) \rightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, 1)$$

The equation of the classifier will be of the form:

$$w_5 x_1^2 + w_4 x_2^2 + w_3 \sqrt{2} x_1 + w_2 \sqrt{2} x_2 + w_1 \sqrt{2} x_1 x_2 + w_0 1 = 0$$

OR

$$w_5 y_5 + w_4 y_4 + w_3 y_3 \qquad + w_2 y_2 \qquad + w_1 y_1 \qquad + w_0 y_0 = 0$$

Non-linear SVM

Transformation:

$$\Phi:(x_1,x_2)\rightarrow(x_1^2,x_2^2,\sqrt{2}x_1,\sqrt{2}x_2,\sqrt{2}x_1x_2,1)\rightarrow(y_1,y_2,\cdots,y_5)$$

Classifier:

$$w_5x_1^2+w_4x_2^2+w_3\sqrt{2}x_1+w_2\sqrt{2}x_2w_1+\sqrt{2}x_1x_2+w_01=0$$

OR

$$w_5y_5+w_4y_4+w_3y_3 \qquad +w_2y_2 \qquad\qquad +w_1y_1 \qquad +w_0y_0=0$$

- The main idea: *linear-separability* increases as the feature dimension increases

Formulation of a Non-linear SVM

- With the new feature vectors $\Phi(\vec{x})$, replace all **x** with $\Phi(\vec{x})$ in linear SVM

Formulation of a Non-linear SVM

- With the new feature vectors $\Phi(\vec{x})$, replace all **x** with $\Phi(\vec{x})$ in linear SVM
  - Minimize $L(w) = \frac{\|\vec{w}\|^2}{2}$
  - Subject to $y_i(\vec{\mathrm{w}} \bullet \Phi(\vec{\mathrm{x}}_\mathrm{i}) + \mathrm{b}) \geq 1$
  - The Dual function is:

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j})$$

Formulation of a Non-linear SVM

- With the new feature vectors $\Phi(\vec{x})$, replace all **x** with $\Phi(\vec{x})$ in linear SVM
  - Minimize $L(w) = \frac{||\vec{w}||^2}{2}$
  - Subject to $y_i(\vec{\mathrm{w}} \bullet \Phi(\vec{\mathrm{x}}_{\mathrm{i}}) + \mathrm{b}) \geq 1$
  - The Dual function is:

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j})$$

$\Phi(\vec{x})$ is called a kernel function

Non-linear SVM

- Once we get the solution of λ's, find w and b using following equations:

$$\vec{w} = \sum_{i=1}^{N} \lambda_i y_i \Phi(\vec{x}_i)$$

$$\lambda_i \{ y_i (\sum_j \lambda_j y_j \Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i) + b) - 1 \} = 0$$

Non-linear SVM

- The new object **z** is classified as:

$$f(\vec{z}) = sign(\vec{w} \cdot \Phi(\vec{z}) + b)$$

Non-linear SVM

- The new object **z** is classified as:

$$f(\vec{z}) = sign(\vec{w} \cdot \Phi(\vec{z}) + b)$$

replacing $\vec{w} = \sum_{i=1}^{N} \lambda_i y_i \Phi(\vec{x}_i)$

Non-linear SVM

- The new object **z** is classified as:

$$f(\vec{z}) = sign(\vec{w} \cdot \Phi(\vec{z}) + b)$$

replacing $\vec{w} = \sum_{i=1}^{N} \lambda_i y_i \Phi(\vec{x}_i)$

$$= sign(\sum_i \lambda_i y_i \Phi(\vec{x}_i) \cdot \Phi(\vec{z}) + b)$$

Issues in Non-linear SVM

- The mapping function is often unclear
- Increase in dimensionality leads to high computation

Issues in Non-linear SVM

- The mapping function is often unclear
- Increase in dimensionality leads to high computation

Solution?

Issues in Non-linear SVM

- Note the equations:

$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j})$$

$$\lambda_i \{ y_i (\sum_j \lambda_j y_j \Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i) + b) - 1 \} = 0$$

$$f(\vec{z}) = sign(\sum_i \lambda_i y_i \Phi(\vec{x}_i) \cdot \Phi(\vec{z}) + b)$$

Issues in Non-linear SVM

- Note the equations:

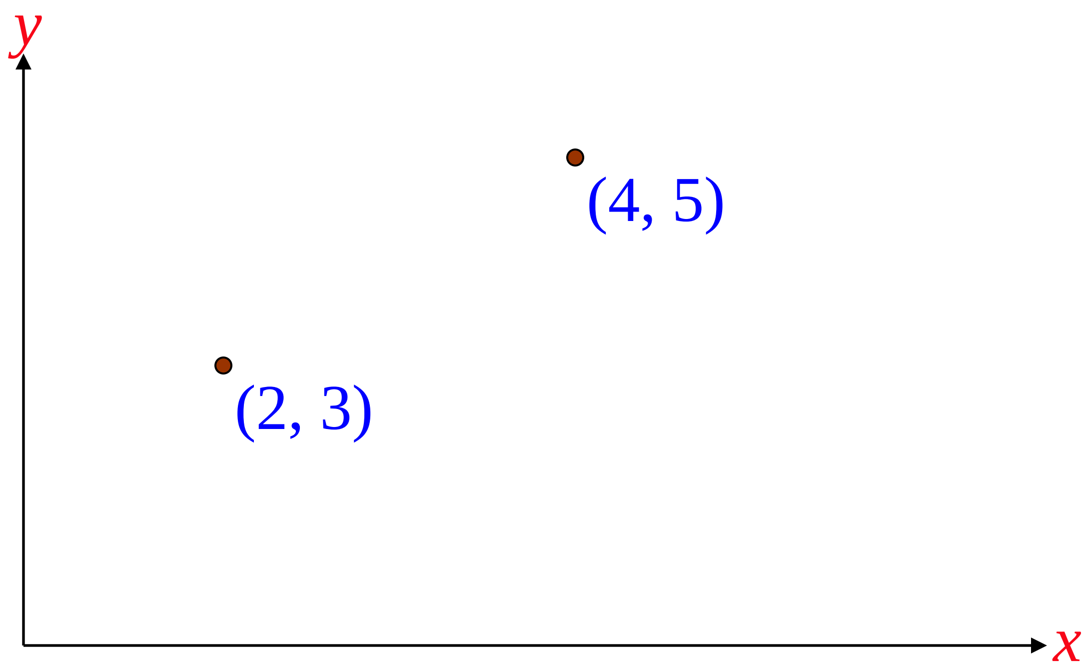
$$L_D = \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j})$$

$$\lambda_i \{ y_i (\sum_j \lambda_j y_j \Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i) + b) - 1 \} = 0$$

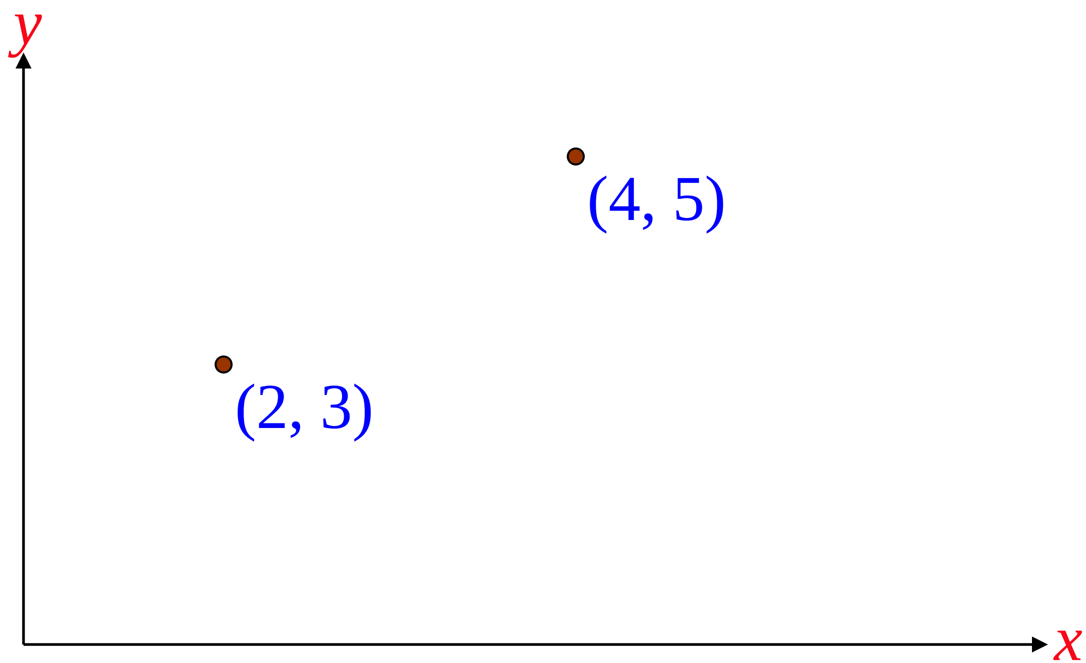
$$f(\vec{z}) = sign(\sum_i \lambda_i y_i \Phi(\vec{x}_i) \cdot \Phi(\vec{z}) + b)$$

- The dot product $\Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i)$ is a similarity measurement

Similarity/Distance Measurement



Similarity/Distance Measurement



$$\text{distance} = \left[(2-4)^2 + (3-5)^2\right]^{1/2}$$

$$\text{Cosine Simmilarity} = \frac{2.4 + 3.5}{\sqrt{(2^2+3^2)}\sqrt{(4^2+5^2)}}$$

Issues in Non-linear SVM

- We can calculate the term $\Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i)$ in the original space using Kernel trick

Issues in Non-linear SVM

- We can calculate the term $\Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i)$ in the original space using Kernel trick

Example:

Let $\vec{\mathbf{u}} = (u_1, u_2) \Rightarrow (u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, \sqrt{2}u_1 u_2, 1)$

Issues in Non-linear SVM

- We can calculate the term $\Phi(\vec{x}_j)\cdot\Phi(\vec{x}_i)$ in the original space using Kernel trick

Example:

$$\Phi(\vec{u})\cdot\Phi(\vec{v})$$

Issues in Non-linear SVM

- We can calculate the term $\Phi(\vec{x}_j)\cdot\Phi(\vec{x}_i)$ in the original space using Kernel trick

Example:

$$\Phi(\vec{u})\cdot\Phi(\vec{v}) = (u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, \sqrt{2}u_1u_2, 1) \cdot (v_1^2, v_2^2, \sqrt{2}v_1, \sqrt{2}v_2, \sqrt{2}v_1v_2, 1)$$

Issues in Non-linear SVM

- We can calculate the term $\Phi(\vec{x}_j)\cdot\Phi(\vec{x}_i)$ in the original space using Kernel trick

Example:

$$\Phi(\vec{u})\cdot\Phi(\vec{v}) = (u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, \sqrt{2}u_1u_2, 1)$$

$$\cdot(v_1^2, v_2^2, \sqrt{2}v_1, \sqrt{2}v_2, \sqrt{2}v_1v_2, 1)$$

$$= u_1^2v_1^2 + u_2^2v_2^2 + 2u_1v_1 + 2u_2v_2 + 2u_1v_1u_2v_2 + 1$$

Issues in Non-linear SVM

- We can calculate the term $\Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i)$ in the original space using Kernel trick

Example:

$$\Phi(\vec{u}) \cdot \Phi(\vec{v}) = (u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, \sqrt{2}u_1u_2, 1)$$

$$\cdot (v_1^2, v_2^2, \sqrt{2}v_1, \sqrt{2}v_2, \sqrt{2}v_1v_2, 1)$$

$$= u_1^2v_1^2 + u_2^2v_2^2 + 2u_1v_1 + 2u_2v_2 + 2u_1v_1u_2v_2 + 1$$

$$= (u_1v_1 + u_2v_2 + 1)^2$$

Issues in Non-linear SVM

- We can calculate the term $\Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i)$ in the original space using Kernel trick

Example:

$$\Phi(\vec{u}) \cdot \Phi(\vec{v}) = (u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, \sqrt{2}u_1u_2, 1)$$

$$\cdot (v_1^2, v_2^2, \sqrt{2}v_1, \sqrt{2}v_2, \sqrt{2}v_1v_2, 1)$$

$$= u_1^2v_1^2 + u_2^2v_2^2 + 2u_1v_1 + 2u_2v_2 + 2u_1v_1u_2v_2 + 1$$

$$= (u_1v_1 + u_2v_2 + 1)^2$$

$$= \left((u_1, u_2).(v_1, v_2) + 1\right)^2$$

Issues in Non-linear SVM

- We can calculate the term $\Phi(\vec{x}_j) \cdot \Phi(\vec{x}_i)$ in the original space using Kernel trick

Example:

$$\Phi(\vec{u}) \cdot \Phi(\vec{v}) = (u_1^2, u_2^2, \sqrt{2}u_1, \sqrt{2}u_2, \sqrt{2}u_1u_2, 1)$$

$$\cdot (v_1^2, v_2^2, \sqrt{2}v_1, \sqrt{2}v_2, \sqrt{2}v_1v_2, 1)$$

$$= u_1^2v_1^2 + u_2^2v_2^2 + 2u_1v_1 + 2u_2v_2 + 2u_1v_1u_2v_2 + 1$$

$$= (u_1v_1 + u_2v_2 + 1)^2$$

$$= \left((u_1, u_2).(v_1, v_2) + 1\right)^2$$

$$= (\vec{u} \cdot \vec{v} + 1)^2$$

Issues in Non-linear SVM

- Kernel trick

$$\Phi(\vec{u}) \cdot \Phi(\vec{v}) = (\vec{u} \cdot \vec{v} + 1)^2$$

Issues in Non-linear SVM

- Kernel trick

$$\Phi(\vec{u}) \cdot \Phi(\vec{v}) = (\vec{u} \cdot \vec{v} + 1)^2$$

$$K(\vec{u}, \vec{v}) = (\vec{u} \cdot \vec{v} + 1)^2$$

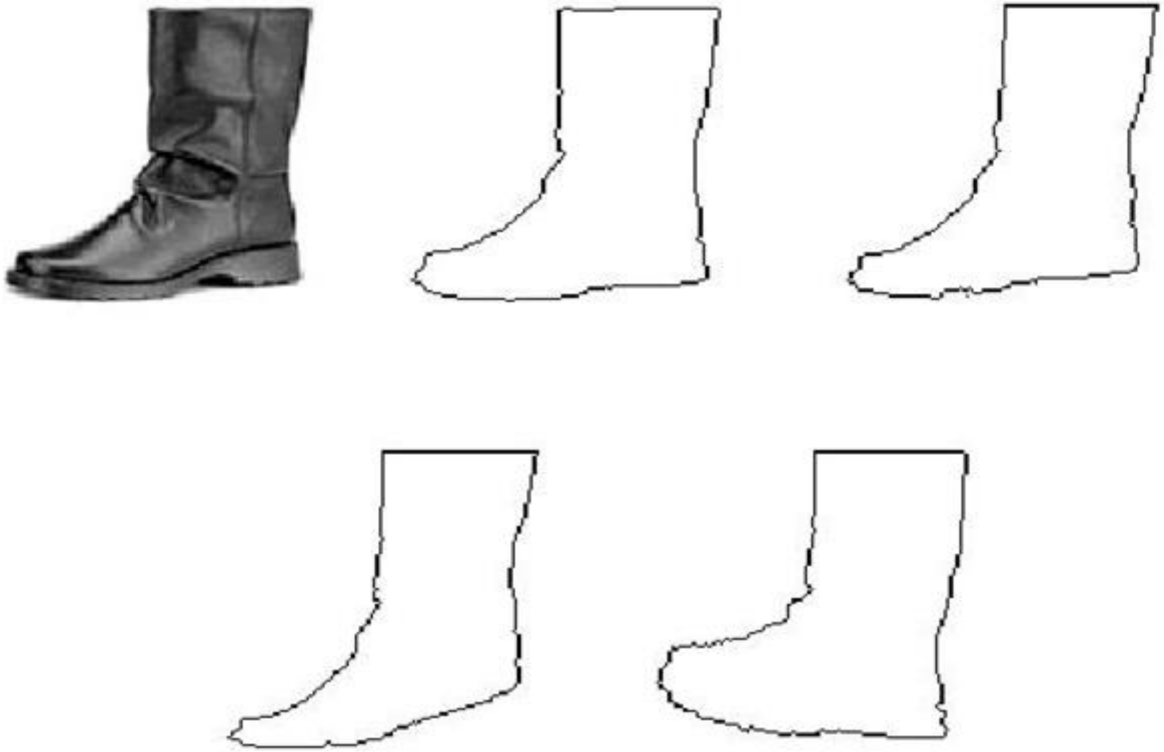
Issues in Non-linear SVM

- Some Kernel Functions are:

$$K(\vec{u},\vec{v}) = (\vec{u}\cdot\vec{v}+1)^{p}$$

$$K(\vec{u},\vec{v}) = e^{-\|\vec{u}-\vec{v}\|^{2}/(2\sigma^{2})}$$

Template Matching



Template Matching

- Typical Applications
  - Speech Recognition
  - Motion Estimation in Video Coding
  - Data Base Image Retrieval
  - Written Word Recognition
  - Bioinformatics

Template Matching

- The Goal:
  - Given a set of reference patterns known as TEMPLATES,
  - find the best match for unknown pattern
  - each class represented by a single typical pattern.

- requires an appropriate “measure” to quantify similarity or matching.

Template Matching

- The cost "measure":
  - deviations between the template and the test pattern.

Template Matching

- The cost "measure":
  - deviations between the template and the test pattern.
  - For example:
    - The word beauty may have been read as beeauty or beuty, etc., due to errors.
    - The same person may speak the same word differently.

Template Matching Methods

- Optimal path searching techniques
- Correlation
- Deformable models

TM using Optimal Path Searching

- Representation: Represent the template by a sequence of measurement vectors or string patterns

Template: $\underline{r}(1), \underline{r}(2), \ldots, \underline{r}(I)$

Test pattern: $\underline{t}(1), \underline{t}(2), \ldots, \underline{t}(J)$

TM using Optimal Path Searching

Template: $\underline{r}(1), \underline{r}(2), \ldots, \underline{r}(I)$

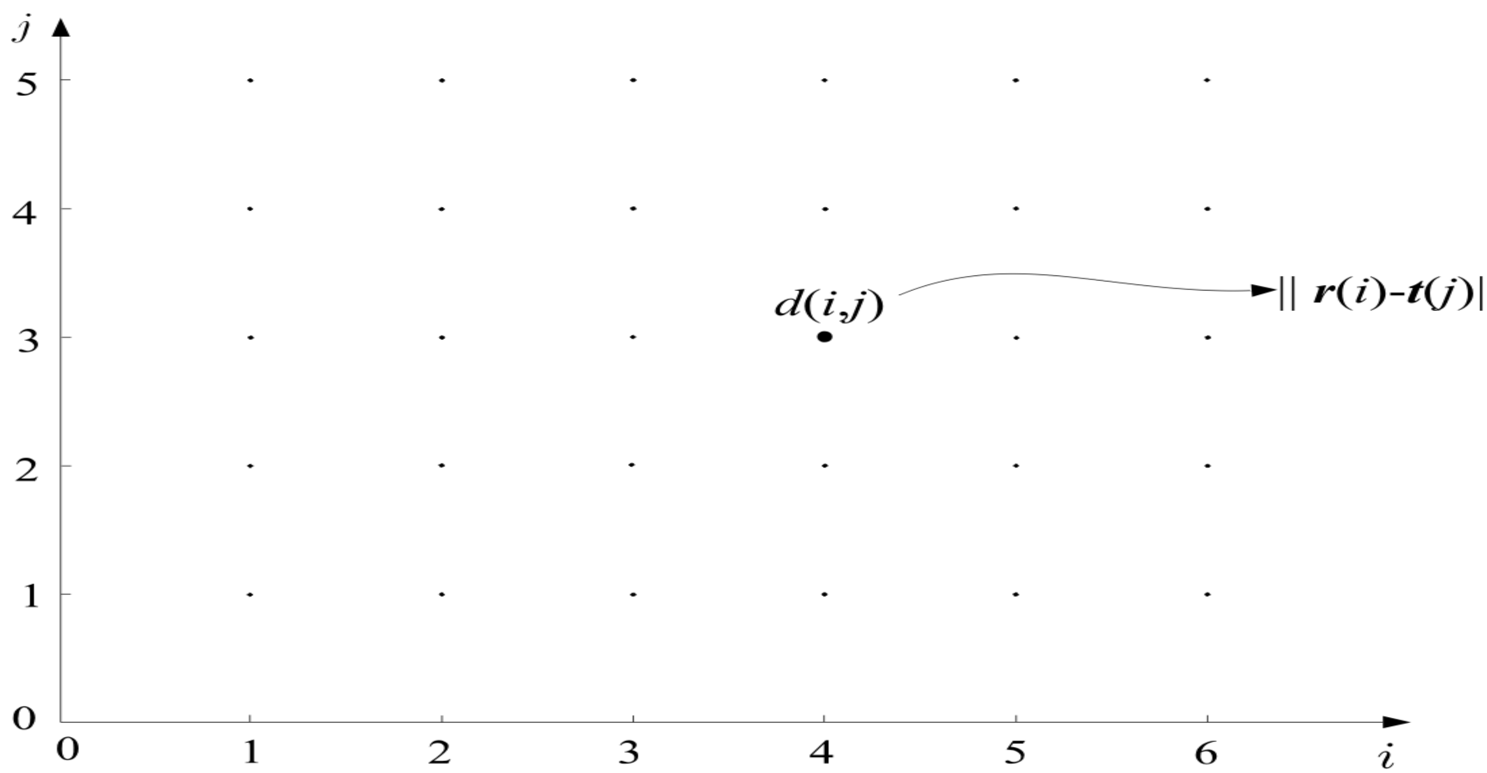
Test pattern: $\underline{t}(1), \underline{t}(2), \ldots, \underline{t}(J)$

- In general $I \neq J$

- We need to find an appropriate distance measure between test and reference patterns.

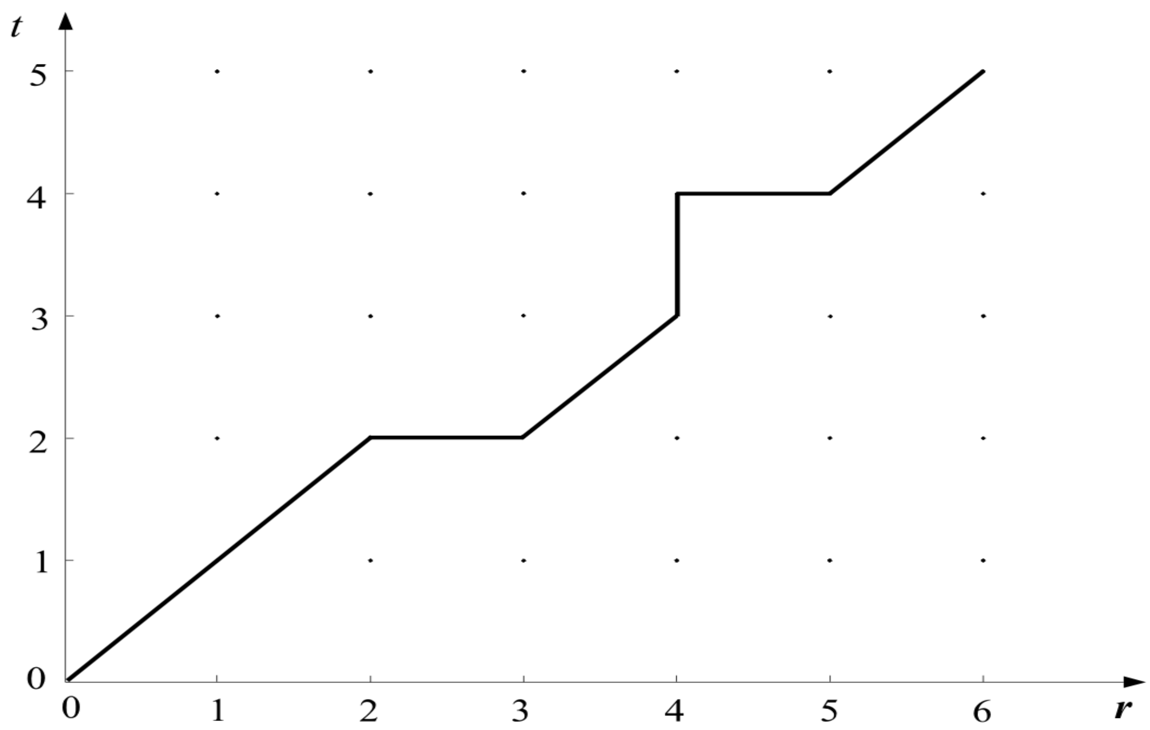
TM using Optimal Path Searching

- Form a grid with I points (template) in horizontal and J points (test) in vertical
- Each point (i,j) of the grid measures the distance between $\underline{r}(i)$ and $\underline{t}(j)$



TM using Optimal Path Searching

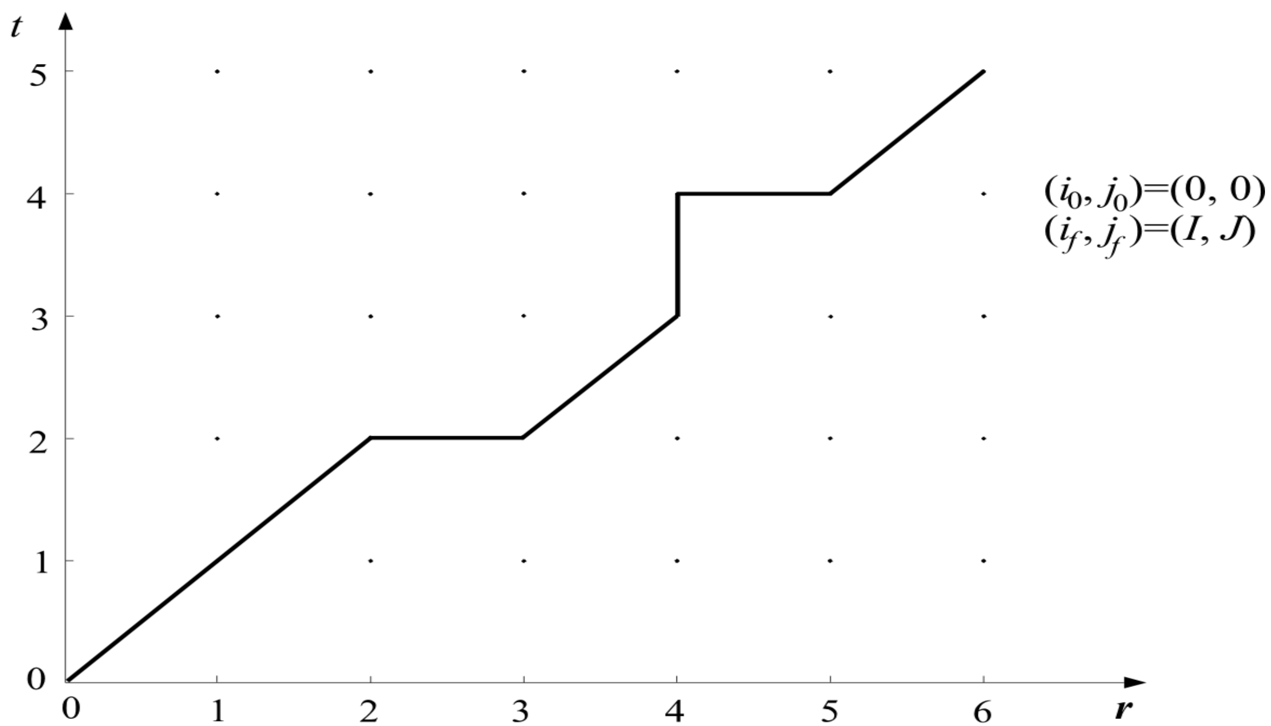
– Path: A path through the grid, from an initial node (i_0, j_0) to a final one (i_f, j_f), is an ordered set of nodes $(i_0, j_0), (i_1, j_1), (i_2, j_2) \ldots (i_k, j_k) \ldots (i_f, j_f)$



TM using Optimal Path Searching

– Path: A path is complete path if:

$$(i_0, j_0) = (0, 0), (i_1, j_1), (i_2, j_2), \ldots, (i_f, j_f) = (I, J)$$

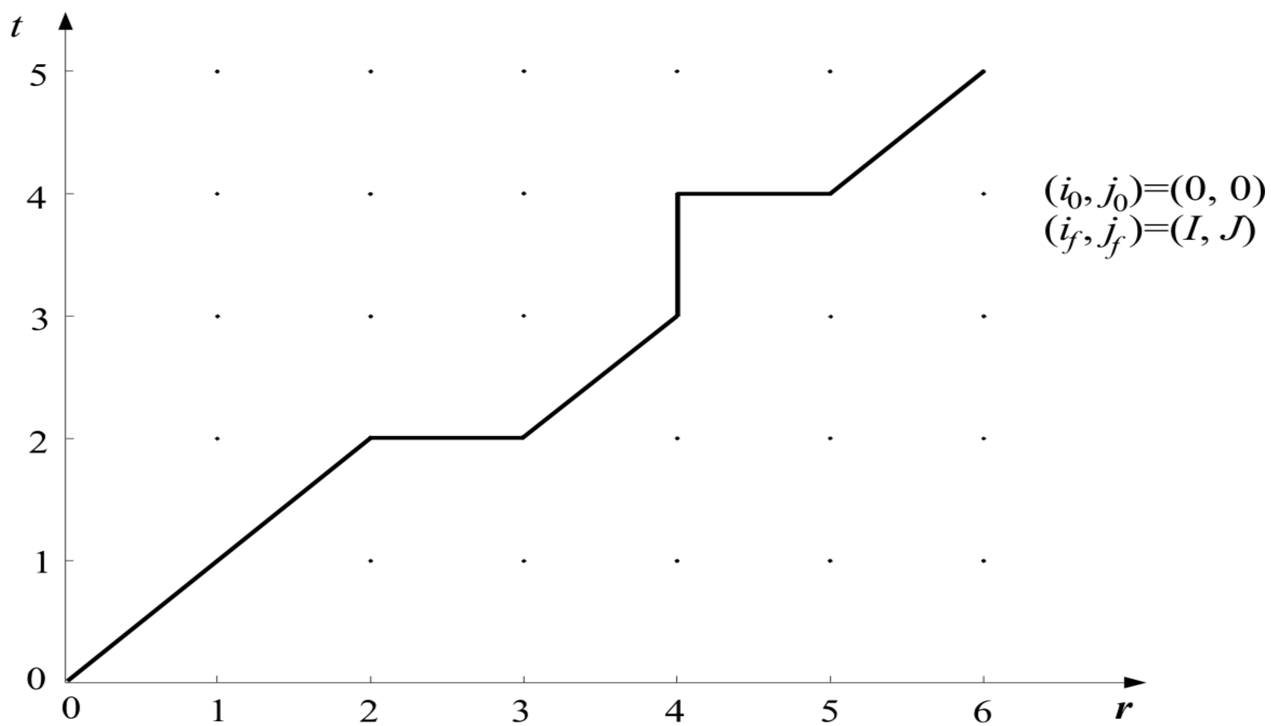


TM using Optimal Path Searching

– Each path is associated with a cost

$$D = \sum_{k=0}^{K-1} d(i_k, j_k)$$

where K is the number of nodes across the path



TM using Optimal Path Searching

- Let the cost up to node (i_k, j_k) be $D(i_k, j_k)$
- By convention
  - $D(0, 0)=0$
  - $d(0,0)=0$

TM using Optimal Path Searching

- The equation

$$D = \sum_{k=0}^{K-1} d(i_k, j_k)$$

assumes that each node has been associated with some cost

TM using Optimal Path Searching

- The equation

$$D = \sum_{k=0}^{K-1} d(i_k, j_k)$$

assumes that each node has been associated with some cost

- However, each transition (i_{k-1}, j_{k-1}) to (i_k, j_k) may also associate with a cost
- The new equation is:

$$D = \sum_{k} d(i_k, j_k | i_{k-1}, j_{k-1})$$

TM using Optimal Path Searching

$$D = \sum_{k} d(i_k, j_k | i_{k-1}, j_{k-1})$$

- Search for the path with the optimal cost $D_{opt.}$
- The matching cost between template $\underline{r}$ and test pattern $\underline{t}$ is $D_{opt.}$
- Costly operation
- Needs efficient computation

Bellman's Optimality Principle

- Optimal path:

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f)$$

Bellman's Optimality Principle

- Optimal path:

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f)$$

- Let *(i,j)* be an intermediate node, i.e.

$$(i_0, j_0) \rightarrow \ldots \rightarrow (i, j) \rightarrow \ldots \rightarrow (i_f, j_f)$$

Bellman's Optimality Principle

- Optimal path:

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f)$$

- Let *(i,j)* be an intermediate node, i.e.

$$(i_0, j_0) \rightarrow \ldots \rightarrow (i, j) \rightarrow \ldots \rightarrow (i_f, j_f)$$

Then, write the optimal path through (i, j)

$$(i_0, j_0) \xrightarrow[(i,j)]{opt} (i_f, j_f)$$

Bellman's Optimality Principle

- Bellman's Principle:

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f) \text{ can be obtained as}$$

$$(i_0, j_0) \xrightarrow{opt} (i, j) \oplus (i, j) \xrightarrow{opt} (i_f, j_f)$$

- meaning: The overall optimal path from (i_0,j_0) to (i_f,j_f) through (i,j) is the concatenation of the optimal paths from (i_0,j_0) to (i,j) and from (i,j) to (i_f,j_f)

Bellman's Optimality Principle

- Bellman's Principle:

$$(i_0, j_0) \xrightarrow{opt} (i_f, j_f) \Leftrightarrow (i_0, j_0) \xrightarrow{opt} (i, j) \oplus (i, j) \xrightarrow{opt} (i_f, j_f)$$

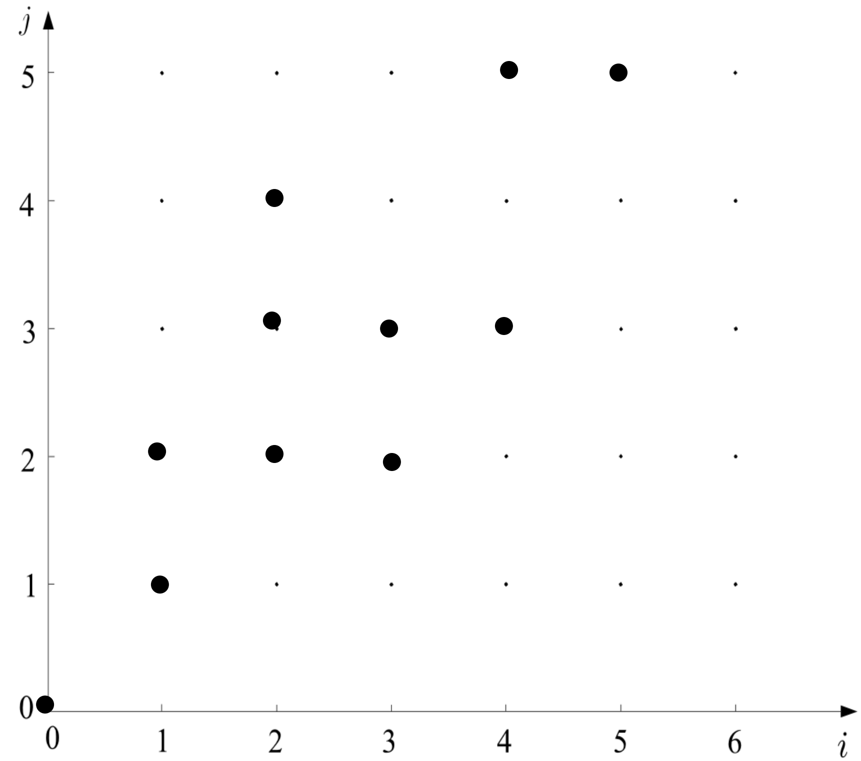
- Let $D_{opt.}(i_{k\text{-}1}, j_{k\text{-}1})$ is the optimal path to reach $(i_{k\text{-}1}, j_{k\text{-}1})$ from (i_0, j_0), then Bellman's principle is stated as:

$$D_{opt}(i_k, j_k) = opt\{D_{opt}(i_{k-1}, j_{k-1}) + d(i_k, j_k \mid i_{k-1}, j_{k-1})\}$$

Bellman's Optimality Principle

$$D_{opt}(i_k, j_k) = opt\{D_{opt}(i_{k-1}, j_{k-1}) + d(i_k, j_k \mid i_{k-1}, j_{k-1})\}$$

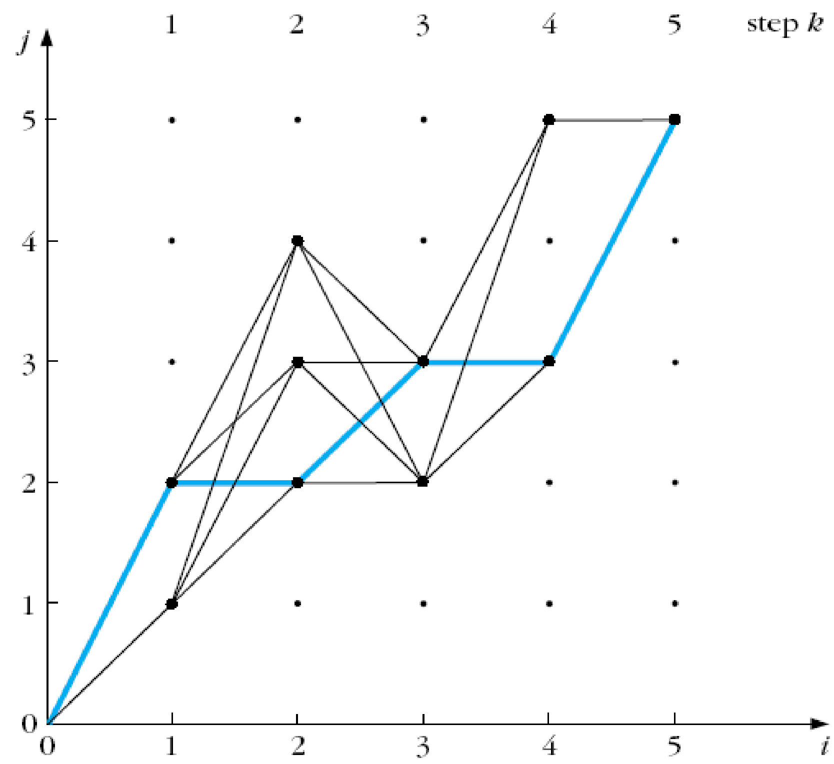
- We don't need to search the whole space to find the optimal path
- Global and local constraints may be imposed to reduce the search space



Bellman's Optimality Principle

$$D_{opt}(i_k, j_k) = opt\{D_{opt}(i_{k-1}, j_{k-1}) + d(i_k, j_k \mid i_{k-1}, j_{k-1})\}$$

- We don't need to search the whole space to find the optimal path
- Global and local constraints may be imposed to reduce the search space



Application of TM in Text Matching: The Edit Distance

- The Edit distance
  - It is used for matching written words. Applications:
    - Automatic Editing
    - Text Retrieval

Application of TM in Text Matching: The Edit Distance

- The Edit distance
  - It is used for matching written words. Applications:
    - Automatic Editing
    - Text Retrieval
  - The measure to be adopted for matching, must take into account:
    - Wrongly identified symbols
      e.g. “befuty” instead of “beauty”
    - Insertion errors, e.g. “bearuty”
    - Deletion errors, e.g. “beuty”

The Edit Distance

- Edit distance: **Minimal** total number of changes, C, insertions I and deletions R, required to change pattern A into pattern B,

$$D(A,B)=\min_{j}[C(j)+I(j)+R(j)]$$

where j runs over All possible variations of symbols, in order to convert $A \longrightarrow B$

The Edit Distance

- Edit distance: **Minimal** total number of changes, *C*, insertions I and deletions R, required to change pattern A into pattern B,

$$D(A,B) = \min_j [C(j) + I(j) + R(j)]$$

where j runs over All possible variations of symbols, in order to convert $A \longrightarrow B$

- *Example*: many ways to change **beuty** to **beauty**

The Edit Distance

- The optimal path search algorithm can be used, provided we know
  - Initial conditions
  - Search space
  - Allowable transitions
  - Distance measure

The Edit Distance

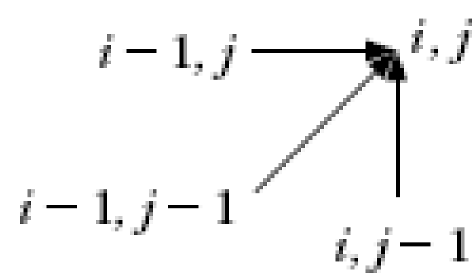
- Cost $D(0,0) = 0$,
- Complete path is searched
- Allowable predecessors and costs:

  - $(i-1, j-1) \rightarrow (i, j)$

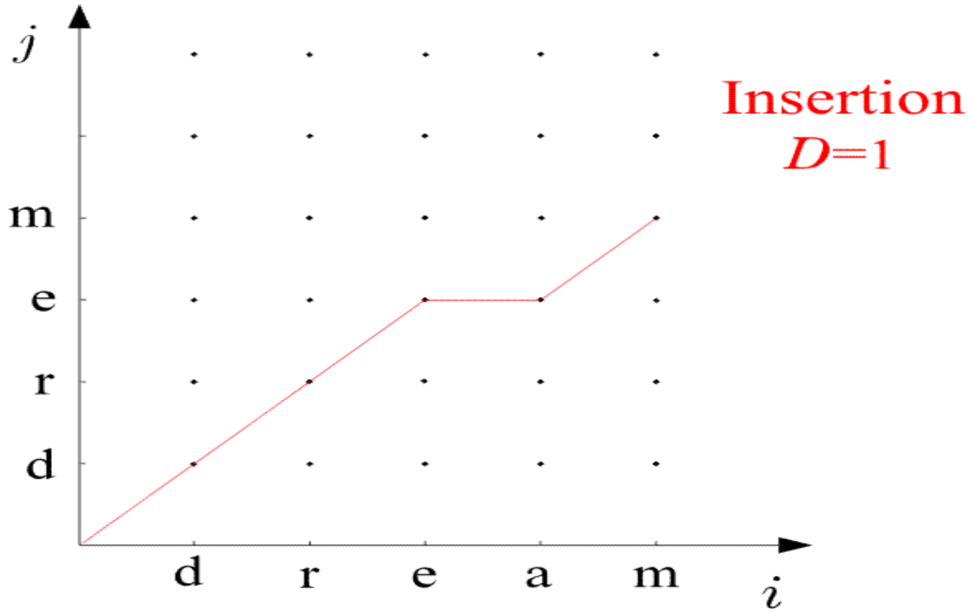
    $$d(i, j \mid i-1, j-1) = \begin{cases} 0, & \text{if } t(i) = r(j) \\ 1, & t(i) \neq r(j) \end{cases}$$

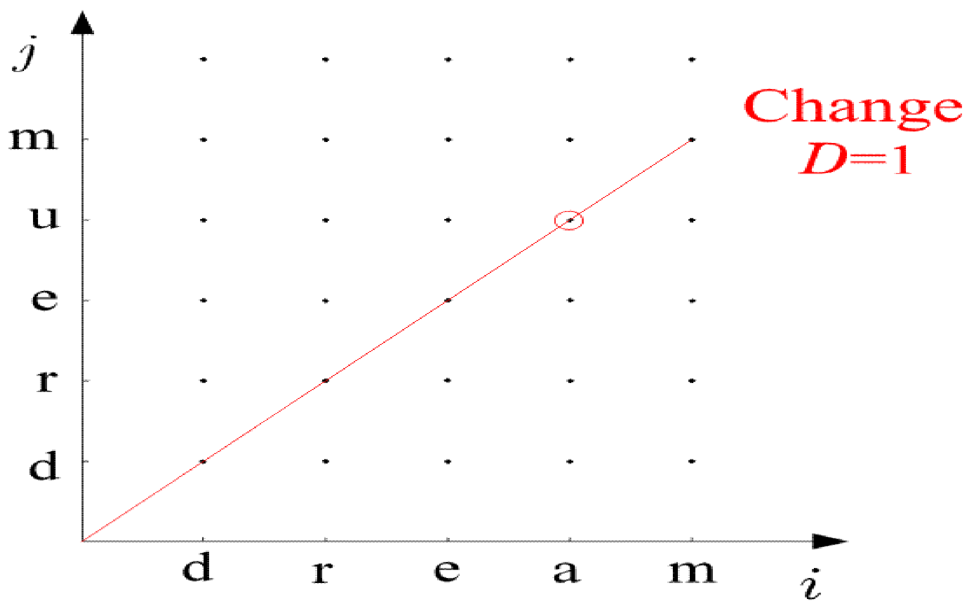
  - Horizontal $d(i, j \mid i-1, j) = 1$

  - Vertical $d(i, j \mid i, j-1) = 1$

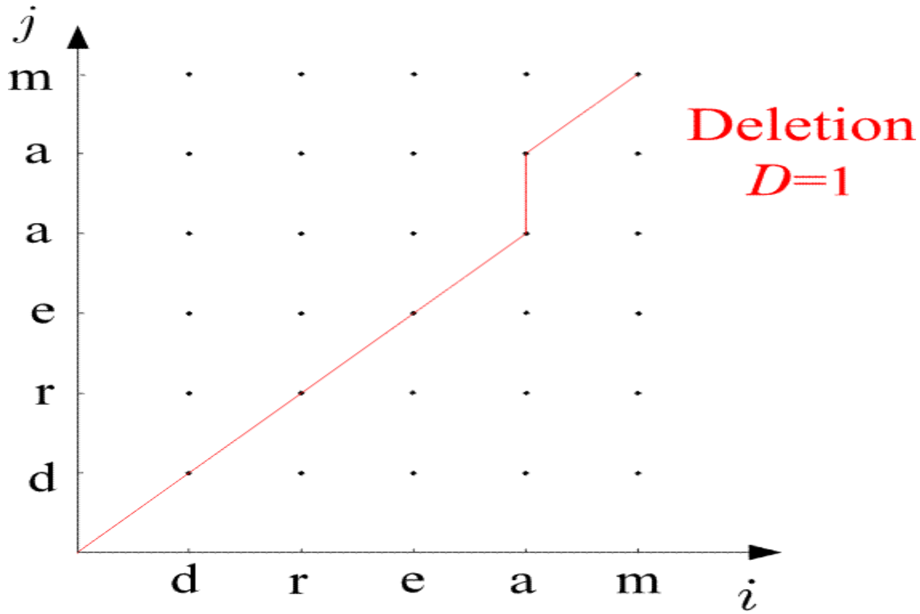


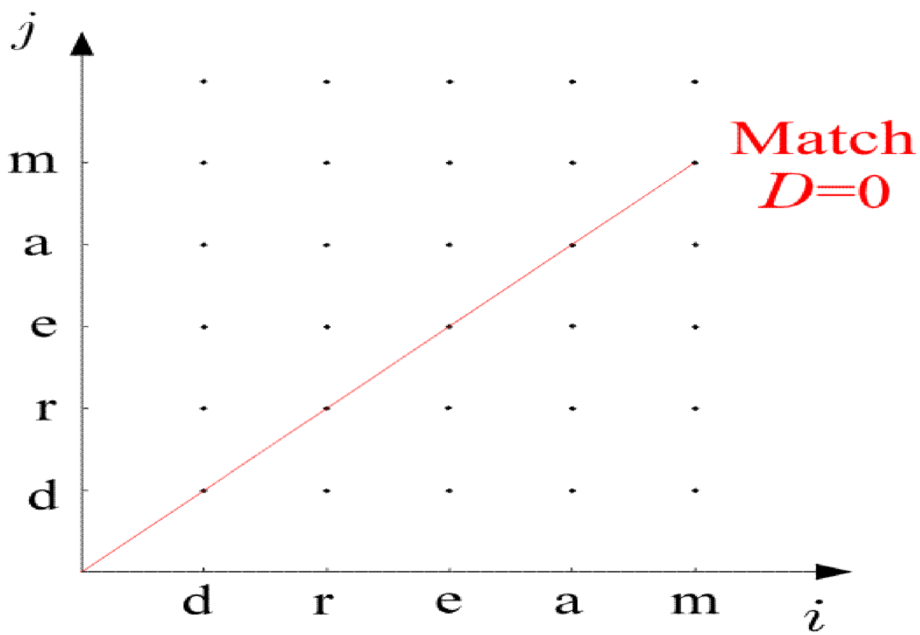
- Examples:





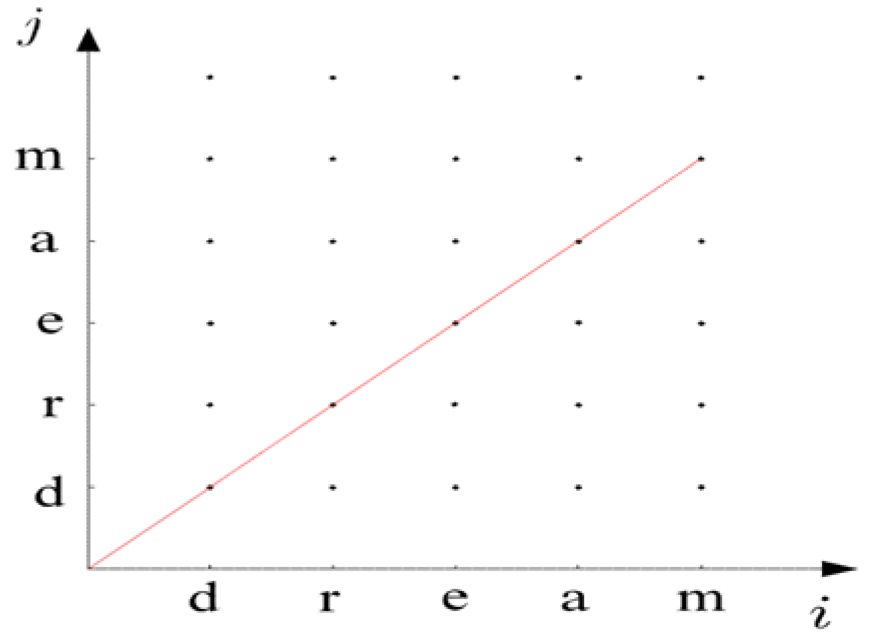
- Examples:





The Edit Distance

- The Algorithm
  - $D(0,0)=0$
  - For i=1, to I
    - $D(i,0)=D(i\text{-}1,0)+1$
  - END {FOR}
  - For j=1 to J
    - $D(0,j)=D(0,j\text{-}1)+1$
  - END{FOR}
  - For i=1 to I
    - For j=1, to J
      - $C_1=D(i\text{-}1,j\text{-}1)+d(i,j \mid i\text{-}1,j\text{-}1)$
      - $C_2=D(i\text{-}1,j)+1$
      - $C_3=D(i,j\text{-}1)+1$
      - $D(i,j)=min\ (C_1,C_2,C_3)$
    - END {FOR}
  - END {FOR}
  - $D(A,B)=D(I,J)$



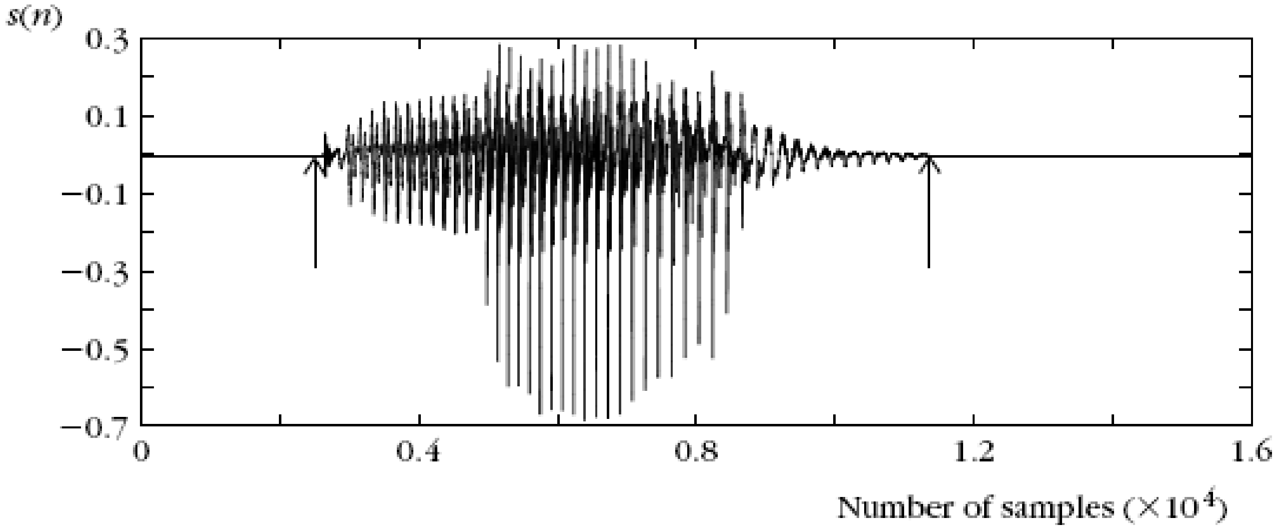
Application of TM in Speech Recognition

- A number of variations
  - Speaker Independent Speech Recognition
  - Speaker Dependent Speech Recognition
  - Continuous Speech Recognition
  - Isolated word recognition (IWR)

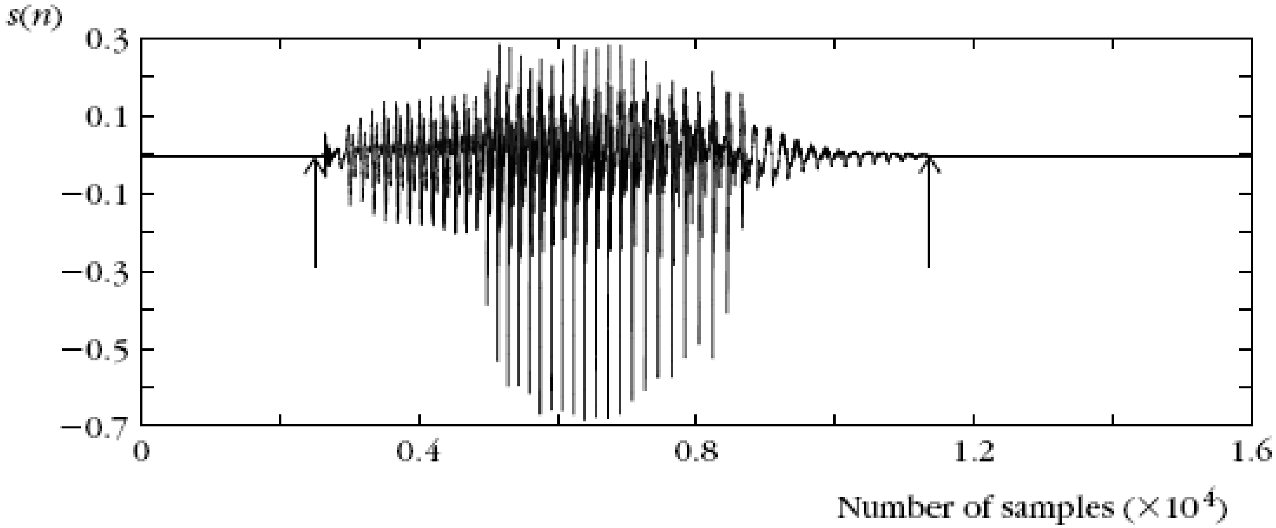
Application of TM in IWR

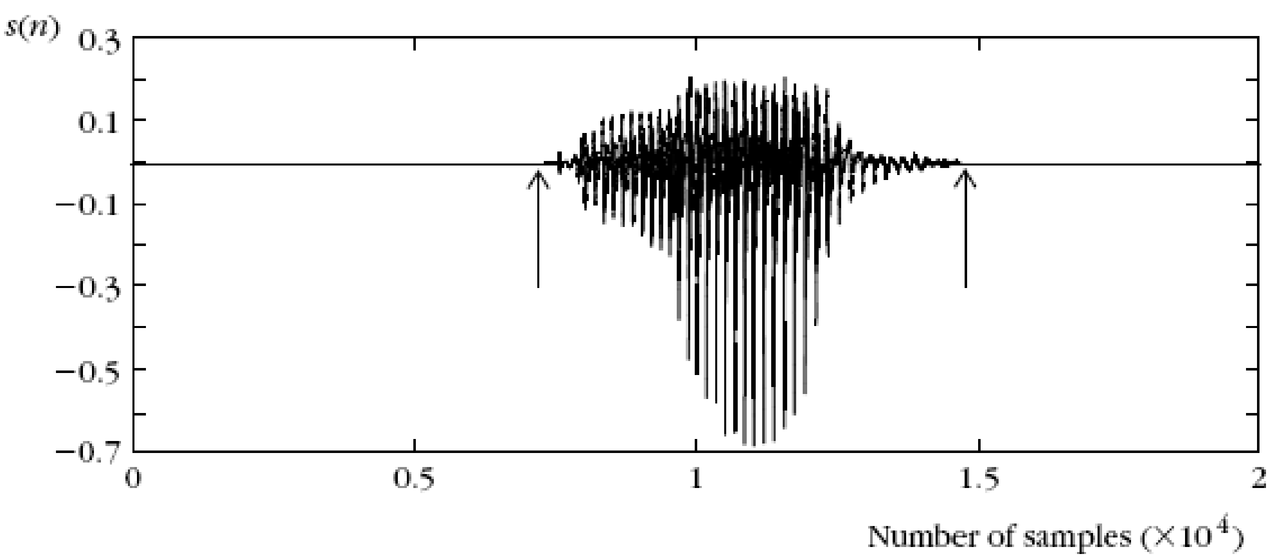
- The goal:
  - Given a number of known spoken words in a data base (reference patterns)
  - find the best match of an unknown spoken word (test pattern).
- Procedure:
  - compare the test word against reference words

Application of TM in IWR



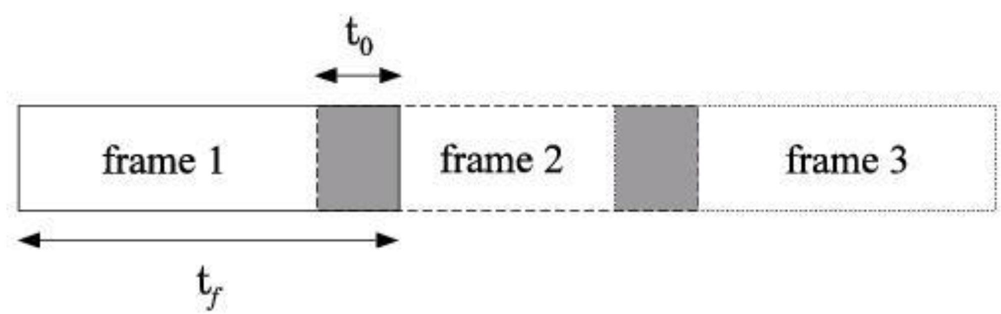
Application of TM in IWR





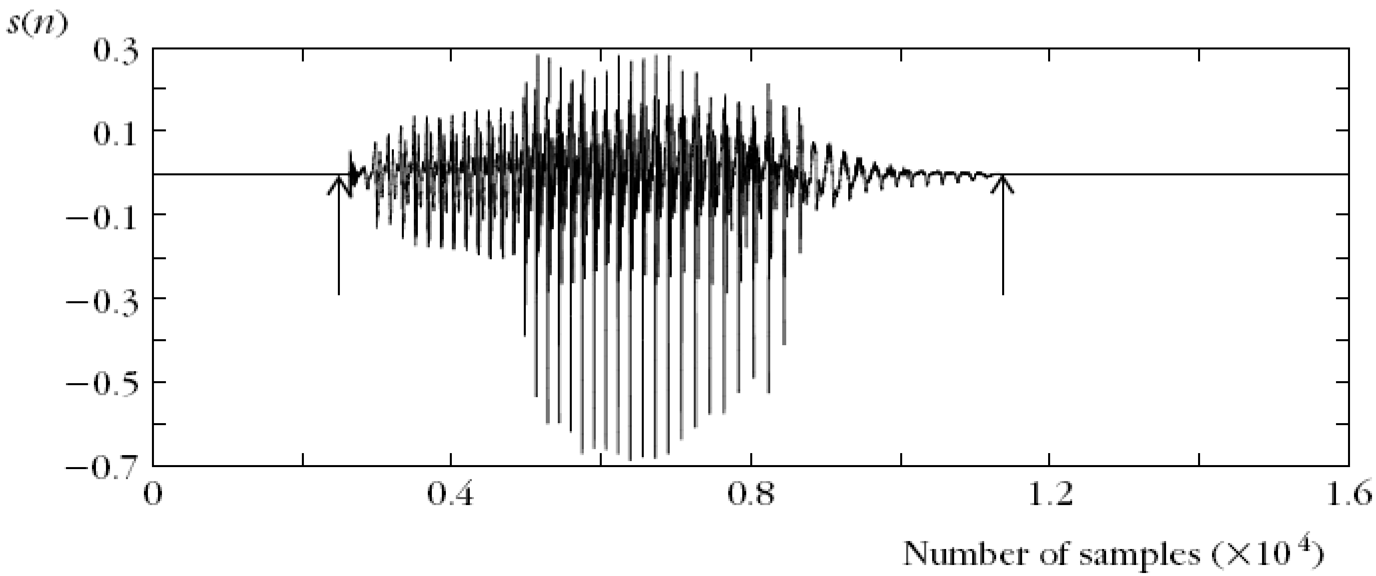
Application of TM in IWR

- The procedure:
  - Express the test and each of the reference patterns as sequences of feature vectors , $\underline{r}(i)$, $\underline{t}(j)$.
  - To this end, divide each of the speech segments in a number of successive frames.



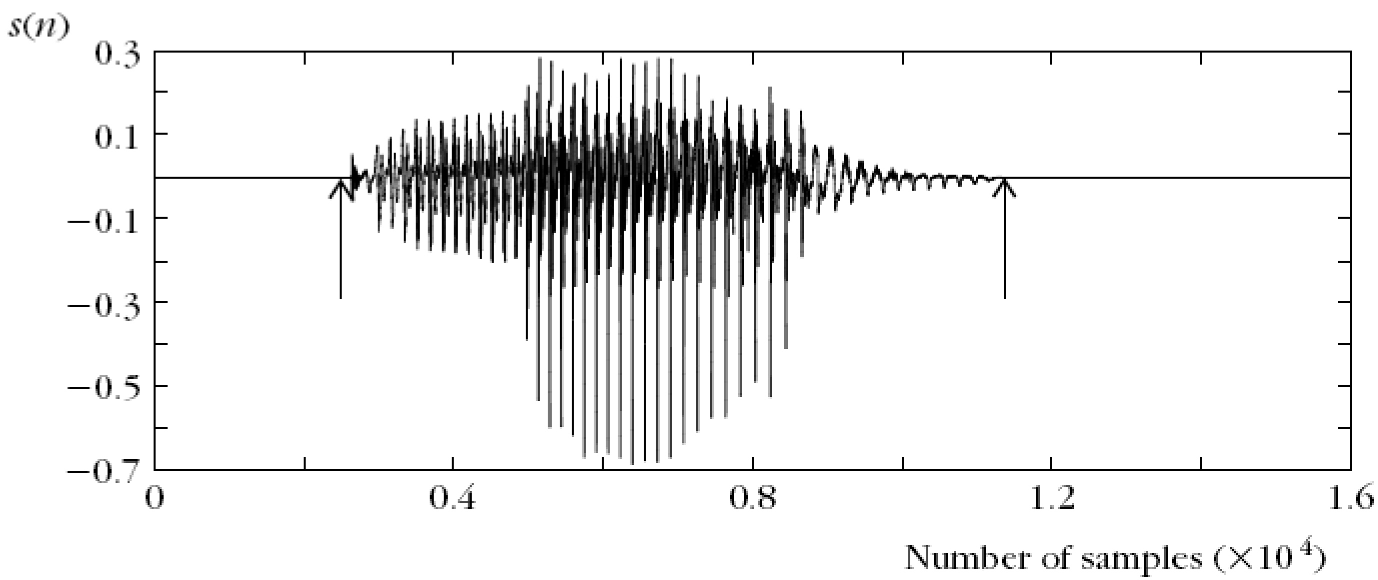
Application of TM in IWR

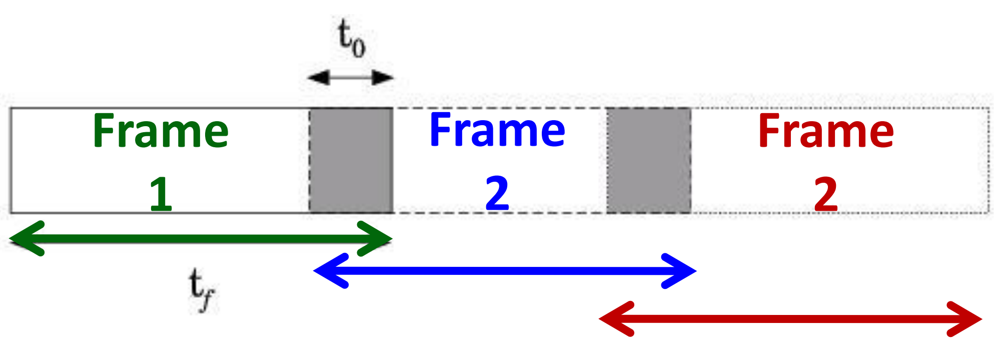
- The procedure:
  - Sample a speech segment from a microphone:



Application of TM in IWR

- The procedure:





$t_f = 512$

$t_0 = 100$

- each frame is represented by a vector of 512 samples

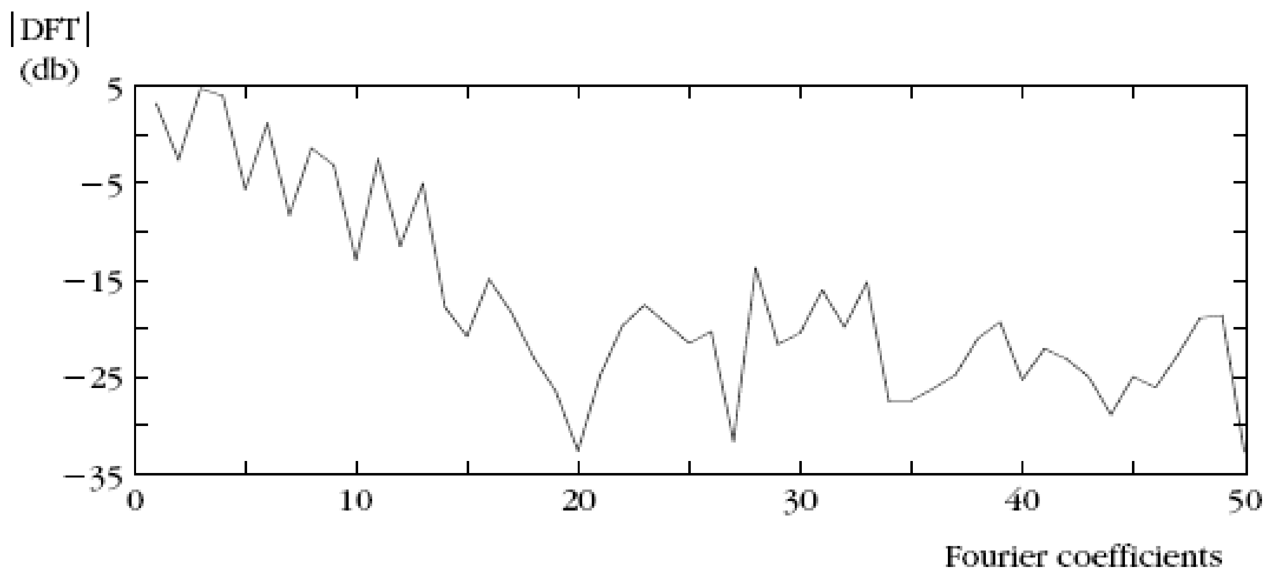
$$\underline{r}(i) = \begin{bmatrix} x_i(0) \\ x_i(1) \\ \dots \\ \dots \\ x_i(512) \end{bmatrix}, \ i = 1, \dots, I \qquad \underline{t}(j) = \begin{bmatrix} x_j(0) \\ x_j(1) \\ \dots \\ \dots \\ x_j(512) \end{bmatrix}, \ j = 1, \dots, J$$

- convert them to DFT

$$DFT(\underline{r}(i)) = DFT\left(\begin{bmatrix} x_i(0) \\ x_i(1) \\ \ldots \\ \ldots \\ x_i(512) \end{bmatrix}\right) = \begin{bmatrix} X_i(0) \\ X_i(1) \\ \ldots \\ \ldots \\ X_i(512) \end{bmatrix}$$

$$DFT(\underline{t}(j)) = DFT\left(\begin{bmatrix} x_i(0) \\ x_i(1) \\ \ldots \\ \ldots \\ x_i(512) \end{bmatrix}\right) = \begin{bmatrix} X_i(0) \\ X_i(1) \\ \ldots \\ \ldots \\ X_i(512) \end{bmatrix}$$

- convert them to DFT



- For each frame compute a feature vector. For example, the DFT coefficients and use, say, ℓ of those:

$$\underline{r}(i)=\begin{bmatrix} x_i(0) \\ x_i(1) \\ \dots \\ \dots \\ x_i(\ell-1) \end{bmatrix},\ i=1,\ \dots,\ I \qquad \underline{t}(j)=\begin{bmatrix} x_j(0) \\ x_j(1) \\ \dots \\ \dots \\ x_j(\ell-1) \end{bmatrix},\ j=1,\ \dots,\ J$$

- For each frame compute a feature vector. For example, the DFT coefficients and use, say, ℓ of those:

$$\underline{r}(i) = \begin{bmatrix} x_i(0) \\ x_i(1) \\ \dots \\ \dots \\ x_i(\ell-1) \end{bmatrix}, \; i = 1, \dots, I \qquad \underline{t}(j) = \begin{bmatrix} x_j(0) \\ x_j(1) \\ \dots \\ \dots \\ x_j(\ell-1) \end{bmatrix}, \; j = 1, \dots, J$$

- Choose a cost function associated with each node across a path, e.g., the Euclidean distance

$$\left\| \underline{r}(i_k) - \underline{t}(j_k) \right\| = d(i_k, j_k)$$

- For each frame compute a feature vector. For example, the DFT coefficients and use, say, ℓ of those:

$$\underline{r}(i) = \begin{bmatrix} x_i(0) \\ x_i(1) \\ \dots \\ \dots \\ x_i(\ell - 1) \end{bmatrix}, \; i = 1, \dots, I \qquad \underline{t}(j) = \begin{bmatrix} x_j(0) \\ x_j(1) \\ \dots \\ \dots \\ x_j(\ell - 1) \end{bmatrix}, \; j = 1, \dots, J$$

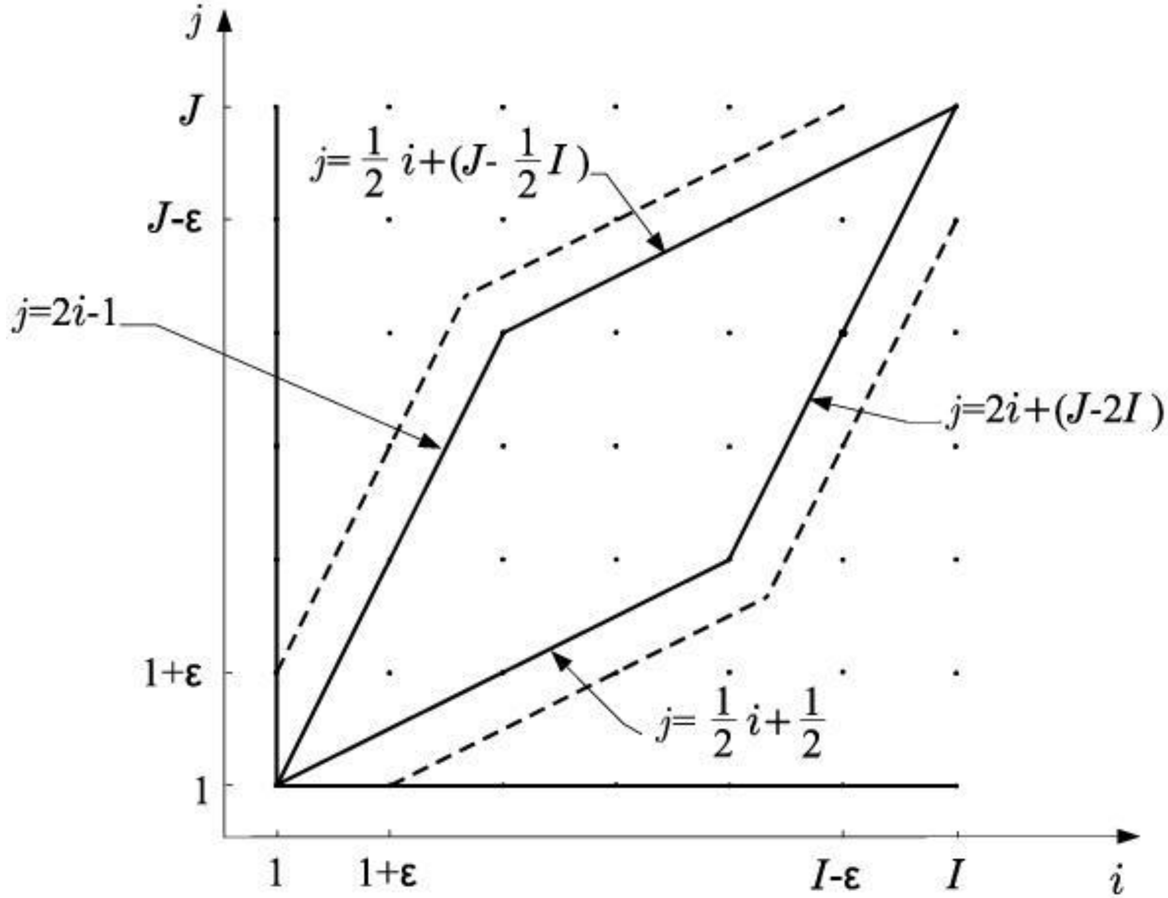
- Choose a cost function associated with each node across a path, e.g., the Euclidean distance

$$\left\| \underline{r}(i_k) - \underline{t}(j_k) \right\| = d(i_k, j_k)$$

- find the optimal path in the grid
- Match the test pattern to the reference pattern associated with the optimal path

- Prior to performing the math one has to choose:
  - end point constraints
  - global constraints
  - local constraints
  - distance

– Prior to performing the math one has to choose:
  - The global constraints: Defining the region of space within which the search for the optimal path will be performed.



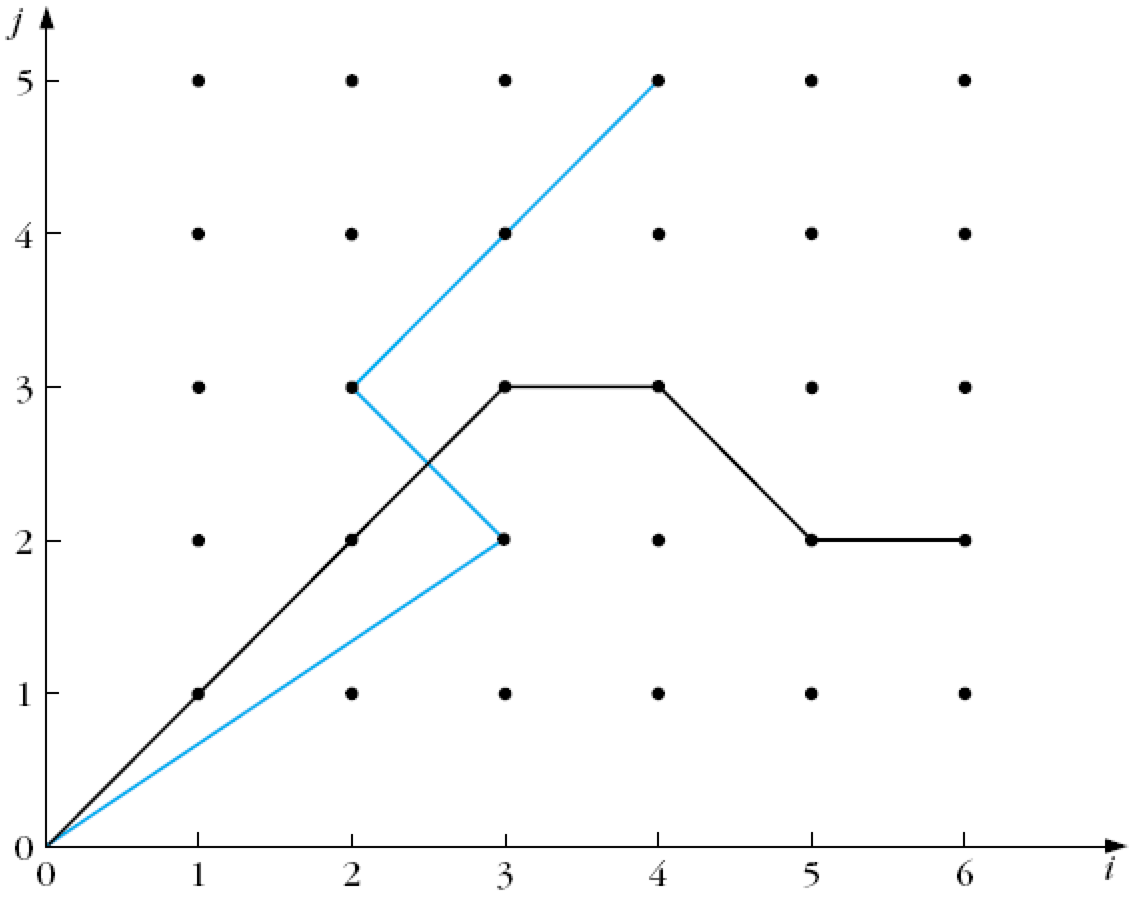
- The local constraints: monotonic path

$$i_{k-1} \leq i_k \quad \text{and} \quad j_{k-1} \leq j_k$$

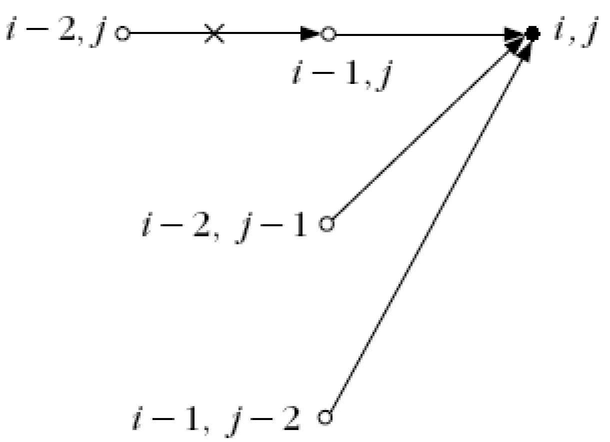
- The local constraints: monotonic path

$$i_{k-1} \leq i_k \quad \text{and} \quad j_{k-1} \leq j_k$$

- Non-monotonic path

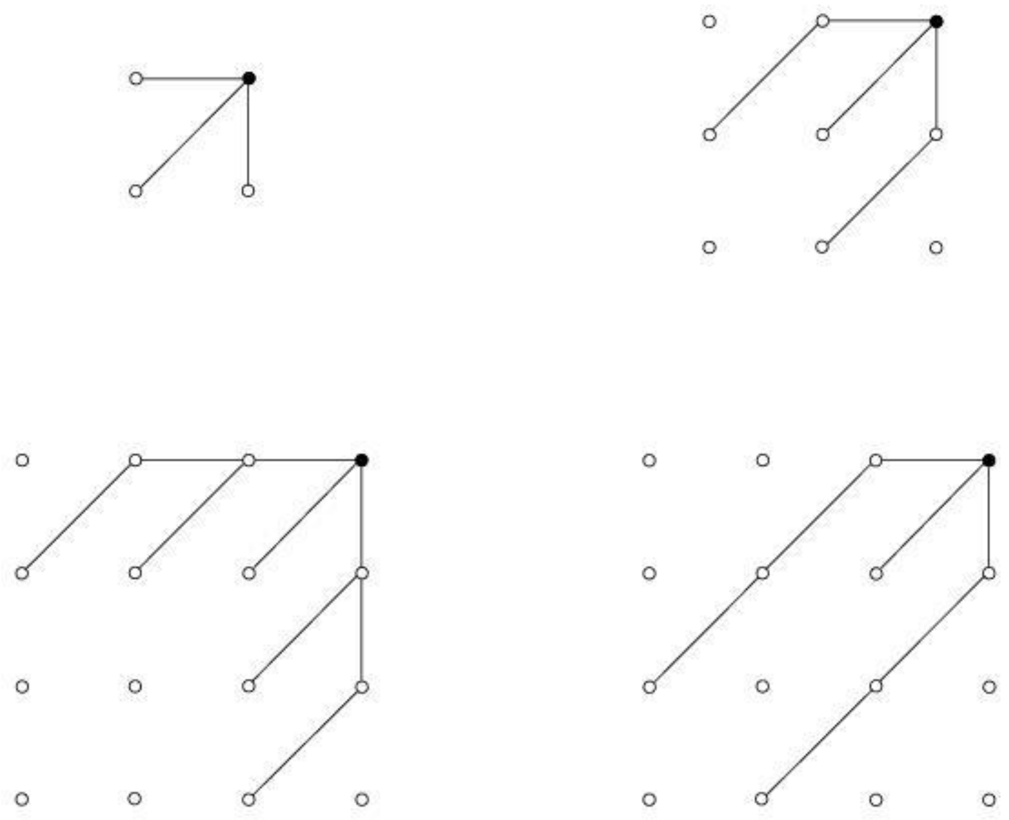


- The local constraints: Defining the type of transitions allowed between the nodes of the grid.



Itakura local constraints

- The local constraints: Defining the type of transitions allowed between the nodes of the grid.



Sakoe and Chiba local constraints

- cost function:
  - Euclidean distance
  - only node distance

$$d(i_k, j_k \mid i_{k-1}, j_{k-1}) = d(i_k, j_k)$$

$$= \left\| \underline{r}(i_k) - \underline{t}(j_k) \right\|$$