Speech Recognition – Intro and DTW

Jan Černocký {cernocky|ihubeika}@fit.vutbr.cz

FIT BUT Brno

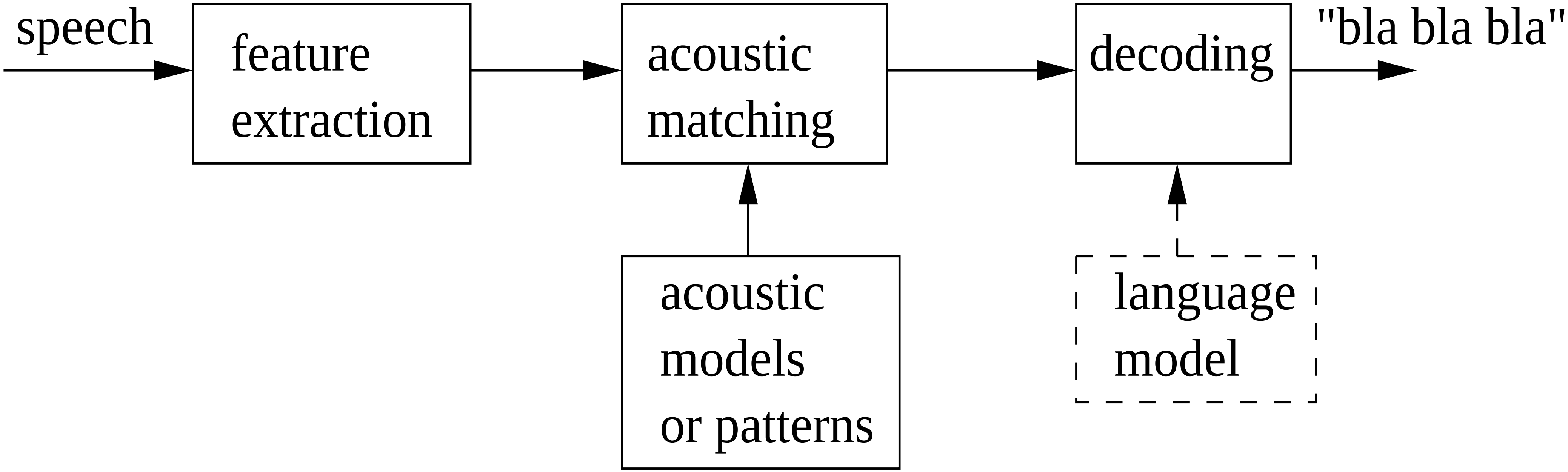
Speech Recognition

Goal: given an unseen speech signal estimate what was said

Classification:

- Isolated words – cell phone voice control. Need a voice activity detector or push-to-talk.
- Continuous words (constrained vocabulary) – e.g. figures in a telephone number or credit card number. The recognition is usually conducted by a network or a simple grammar.
- Large vocabulary continuous speech recognition LVCSR – hardest task. Requires information on acoustics but also the structure of the language (language model) and pronunciation dictionary. Works with smaller units than words (60 thousand words cannot be learned. . .) – phonemes, context dependent phonemes.

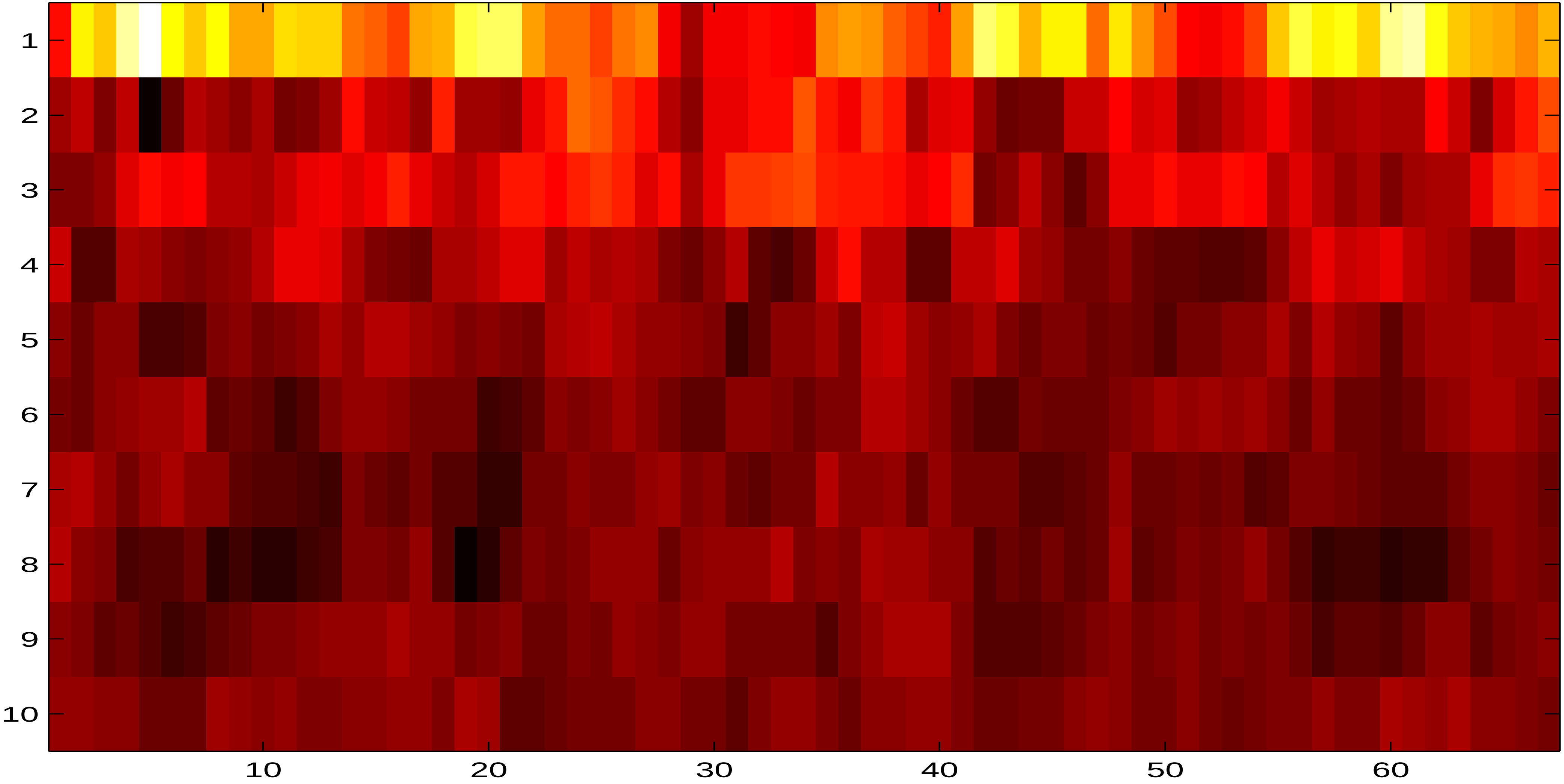
Structure of a recognizer



Parameterization

- Data size reduction.
- Discarding the components we are not interested in (pitch, mean value, phase)
- Usually based on spectral analysis (Mel-frequency cepstral coefficients) or LPC analysis (LPC cepstrum).
- Framing (quasi-stationarity)
- Parameters have to be convenient for the recognizer (uncorrelated parameters)
- See the lecture on parameterization !

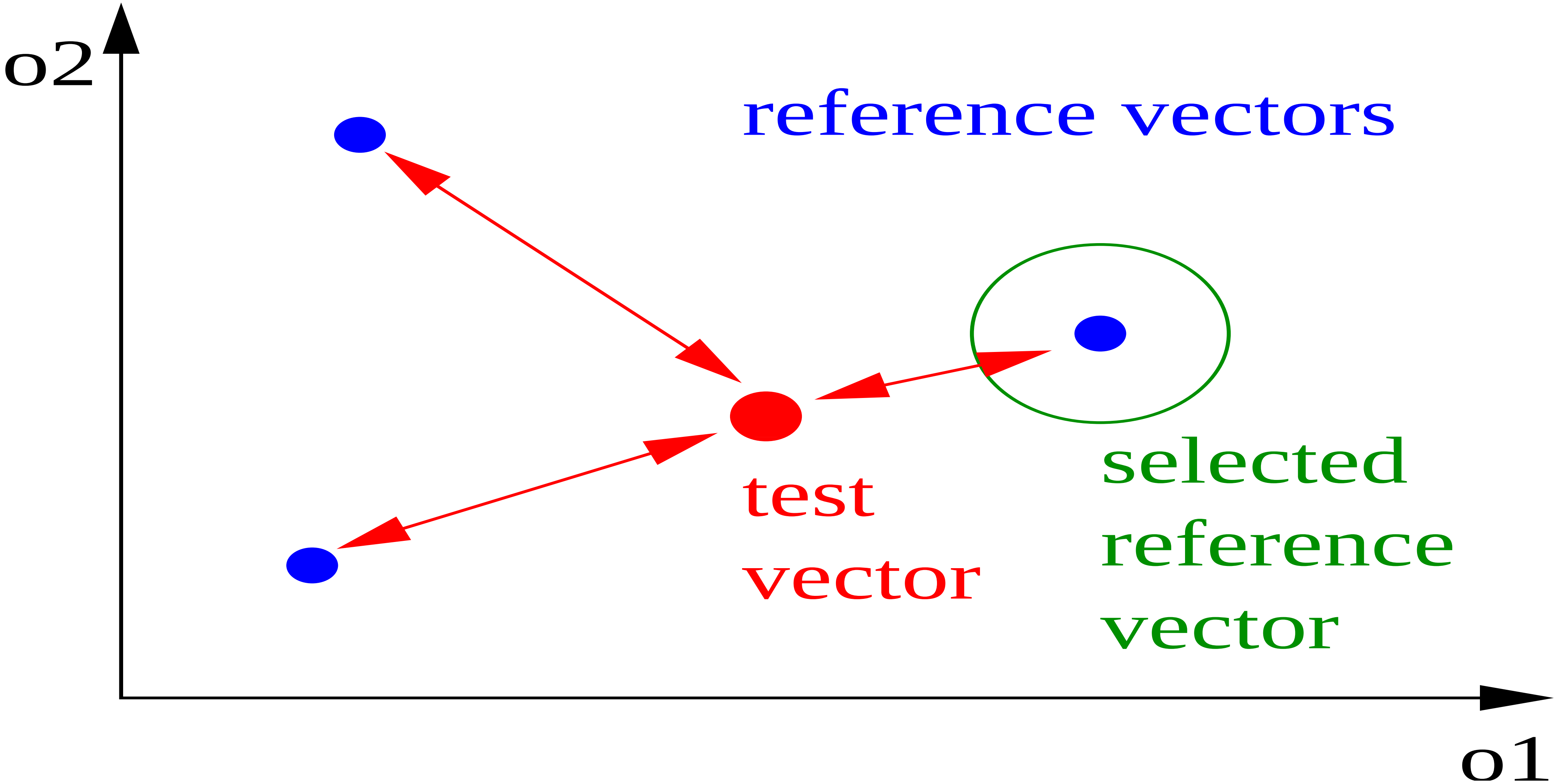
The result of parameterization is a sequence of vectors: $\mathbf{O} = [\mathbf{o}(1), \mathbf{o}(2), \ldots, \mathbf{o}(T)]$

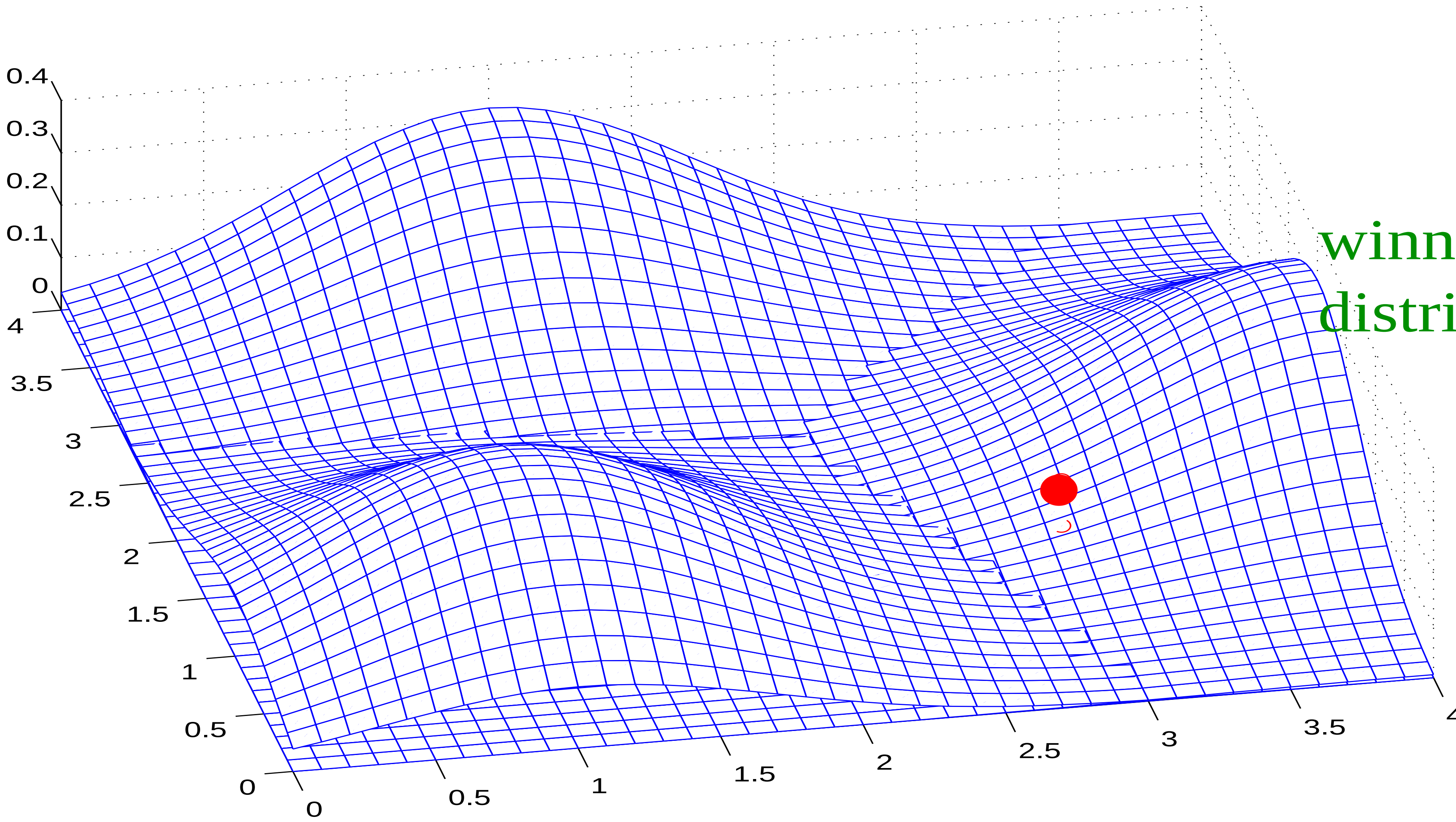


Acoustic matching — variability everywhere !!!

parameter space - a human never says one thing twice in exactly te same way
$\Rightarrow$ parameter vectors are **always different**. Methods working on text fail $\Rightarrow$ How to do it ?

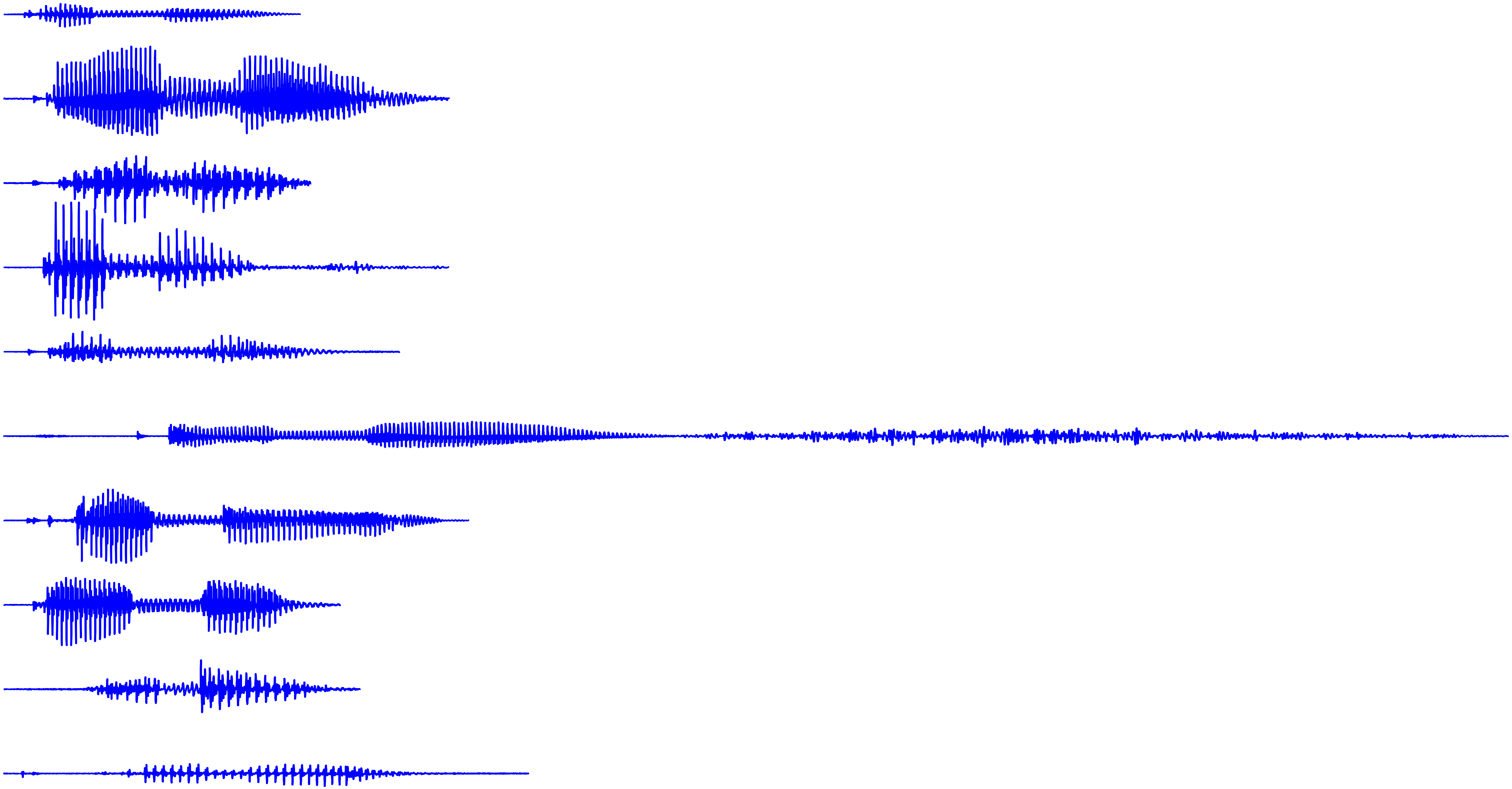
1. Calculation of distance between two vectors.
2. Statistical modeling.


o2
reference vectors
selected
reference
vector
test
vector
o1

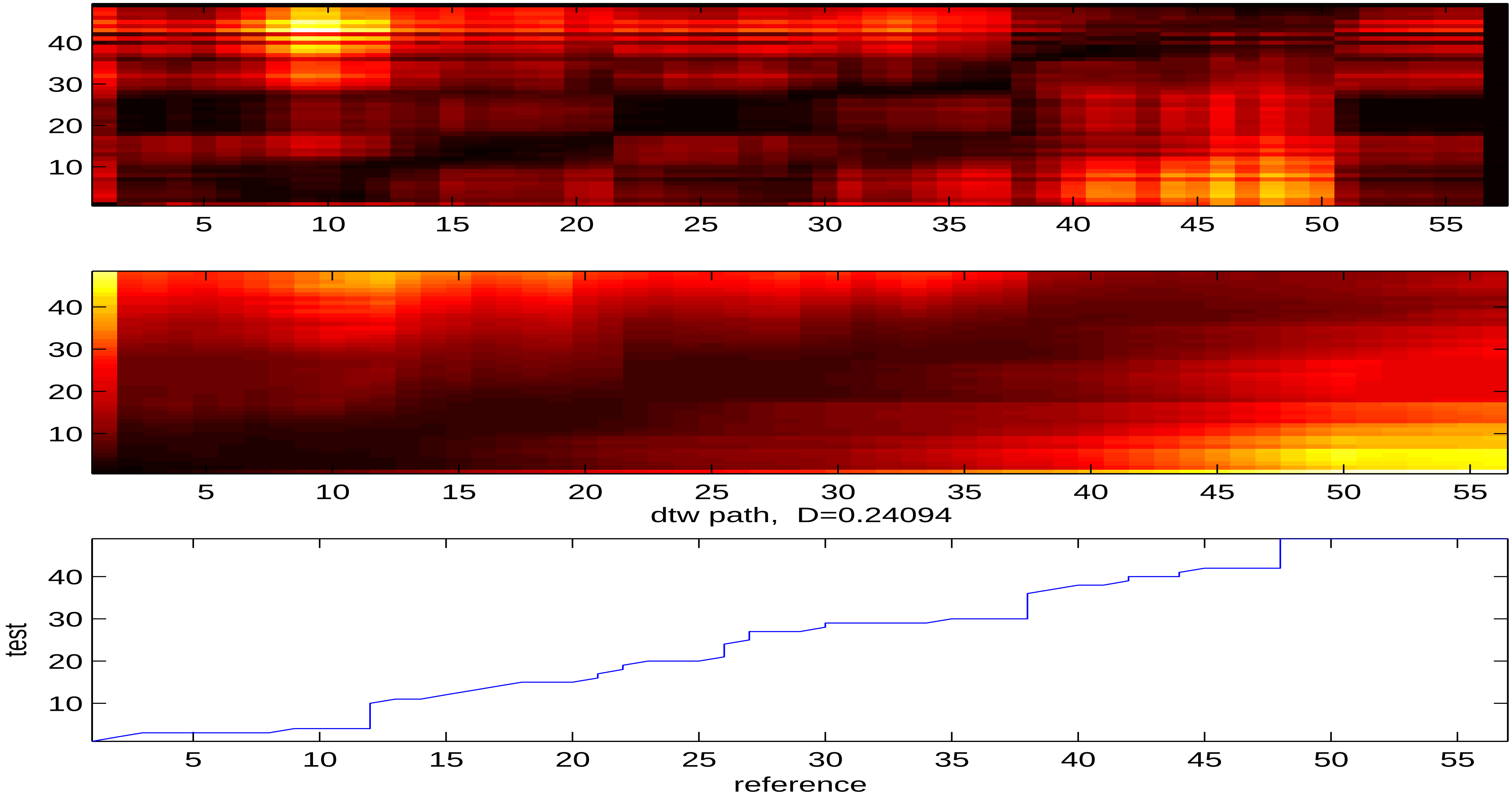


winning distribution

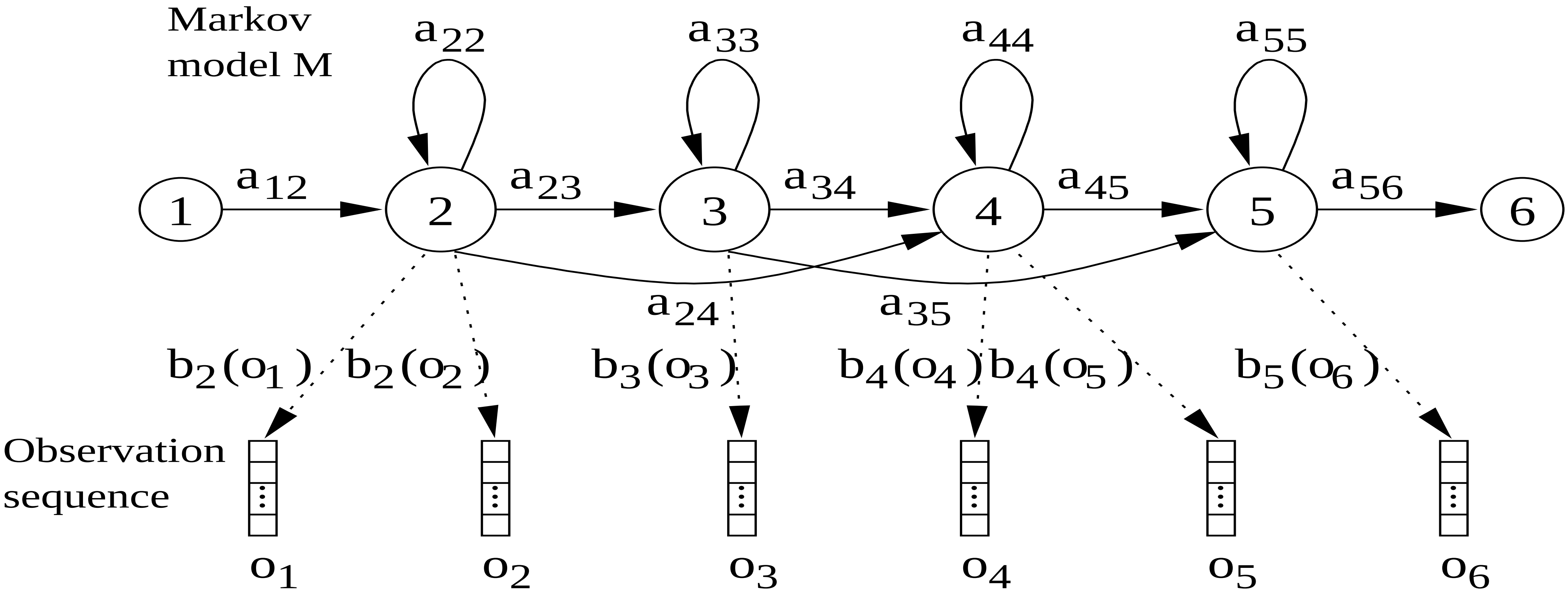
timing – people never say one thing with the same timing.



timing n.1 - Dynamic time warping - path



timing n.2 - Hidden Markov models - state sequence



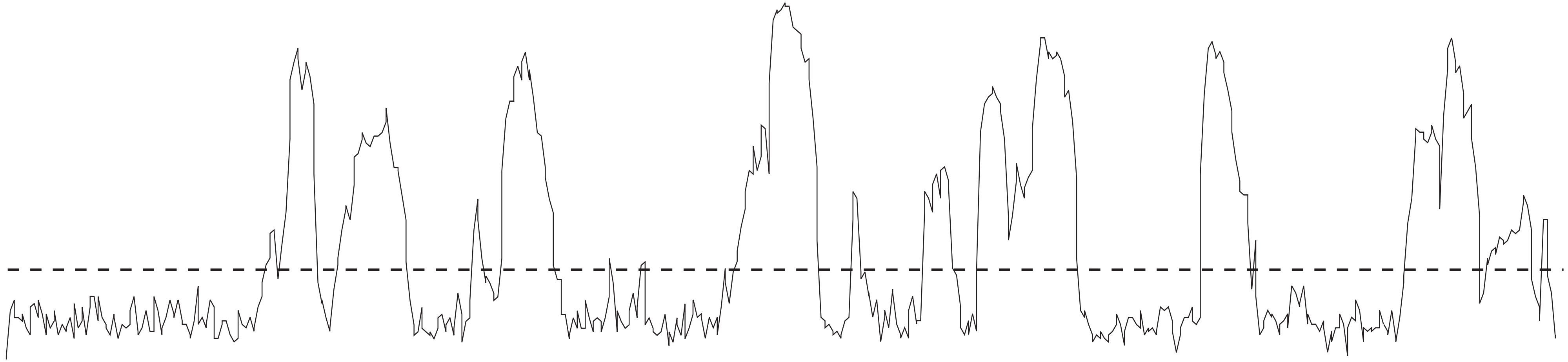
Decoding

- Isolated words: very simple (select the maximum probability or the minimum distance).
- LVCSR: very difficult (acoustic models (tri-phones), language model, pronunciation model) - Viterbi algorithm, $A^\star$-search, best-first decoding, finite state automata (!), search space narrowing (beam-search), etc.

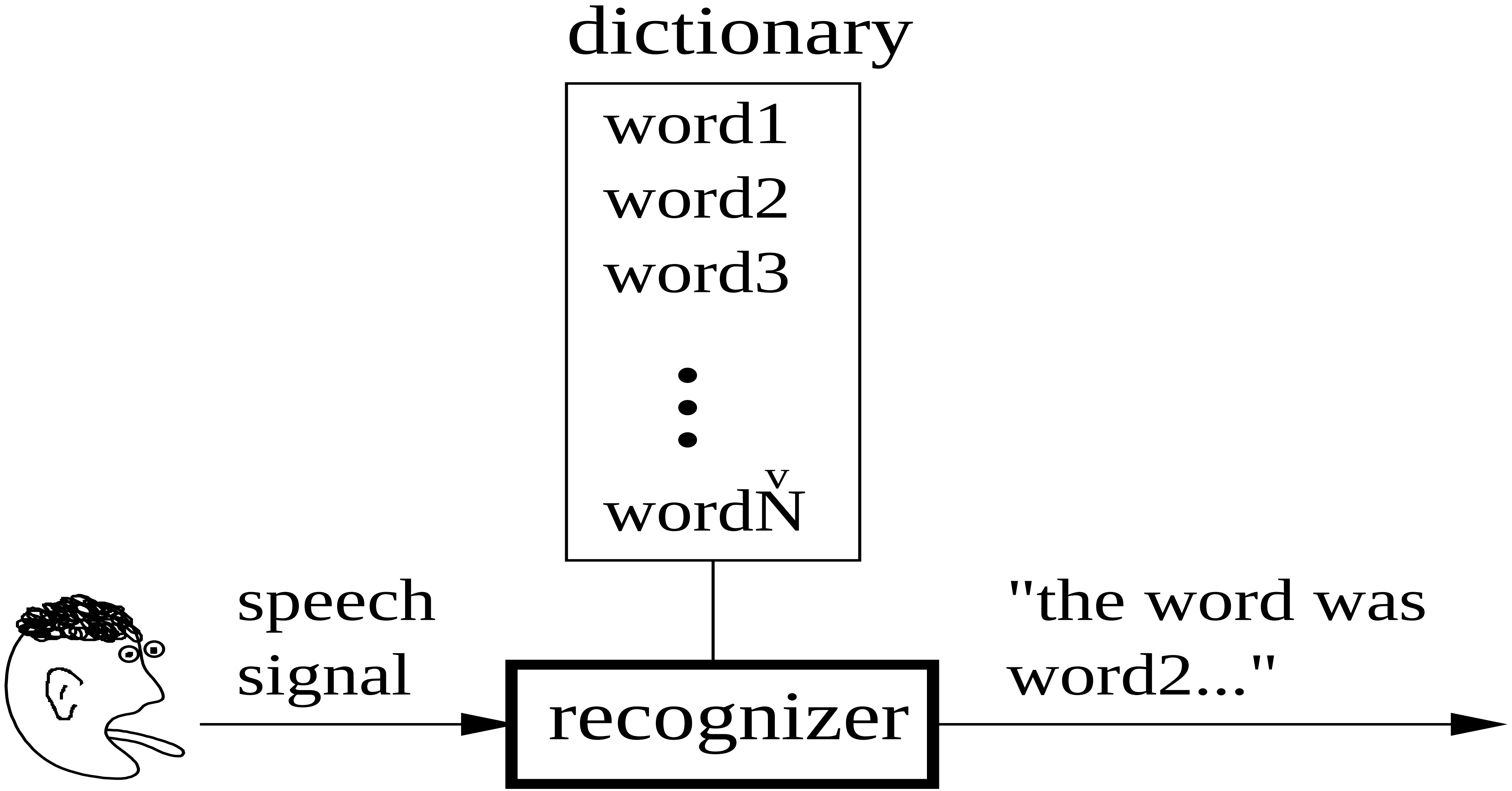
ISOLATED WORDS RECOGNITION USING DTW

Isolated words boundary detection:

- push-to-talk. . .
- speech activity detector – e.g. detector based on energy:



How does it work?:



- the dictionary contains **reference** parameter matrices for the words of interest:

$$\mathbf{R}_1 \ldots \mathbf{R}_N$$

- a **test** parameter matrix comes as an input to the recognizer $\mathbf{O}$
- the task is to determine which reference word corresponds to the test word.

If the words were represented by only one vector, it would be simple:

$$d(\mathbf{o}, \mathbf{r}_i) = \sqrt{\sum_{k=1}^{P} |o(k) - r_i(k)|^2}.$$

A minimum distance would be chosen.

Words are however represented by **more than one** vector (a sequence): The task is to determine the *distance* (*similarity*) of the reference vector of the length R:

$$\mathbf{R} = [\mathbf{r}(1), \ldots, \mathbf{r}(R)] \tag{1}$$

and the test sequence of the length T:

$$\mathbf{O} = [\mathbf{o}(1), \ldots, \mathbf{o}(T)] \tag{2}$$

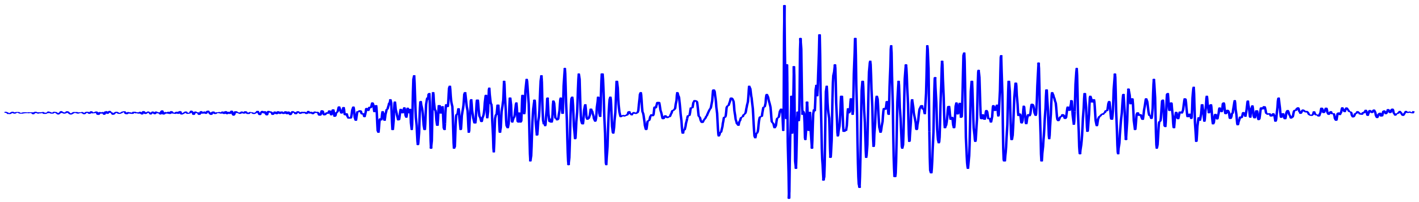
Calculating of distances between single vectors ? How do they correspond to each other? Words are **almost never represented by the sequence of the same length** $R \neq T$.

Linear Alignment

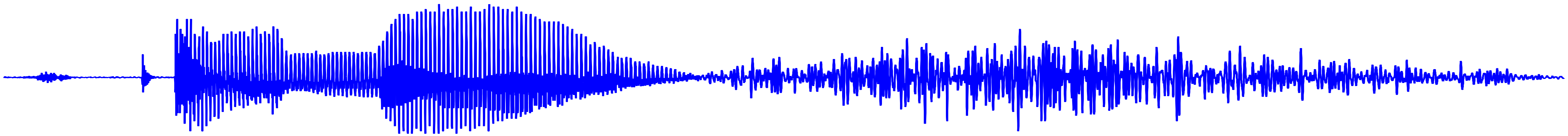
$$D(\mathbf{O}, \mathbf{R}) = \sum_{i=1}^{R} d[\mathbf{o}(w(i)), \mathbf{r}(i)] \quad (3)$$

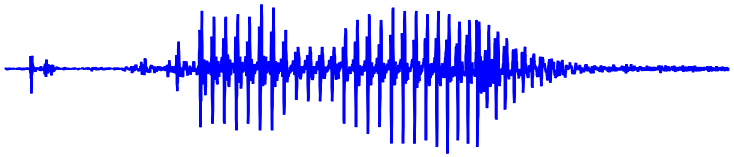
where $w(i)$ is defined so that the alignment is linear.

But this is not going to work in most cases... here though, still working:



. . . not working for this example (error of VAD):



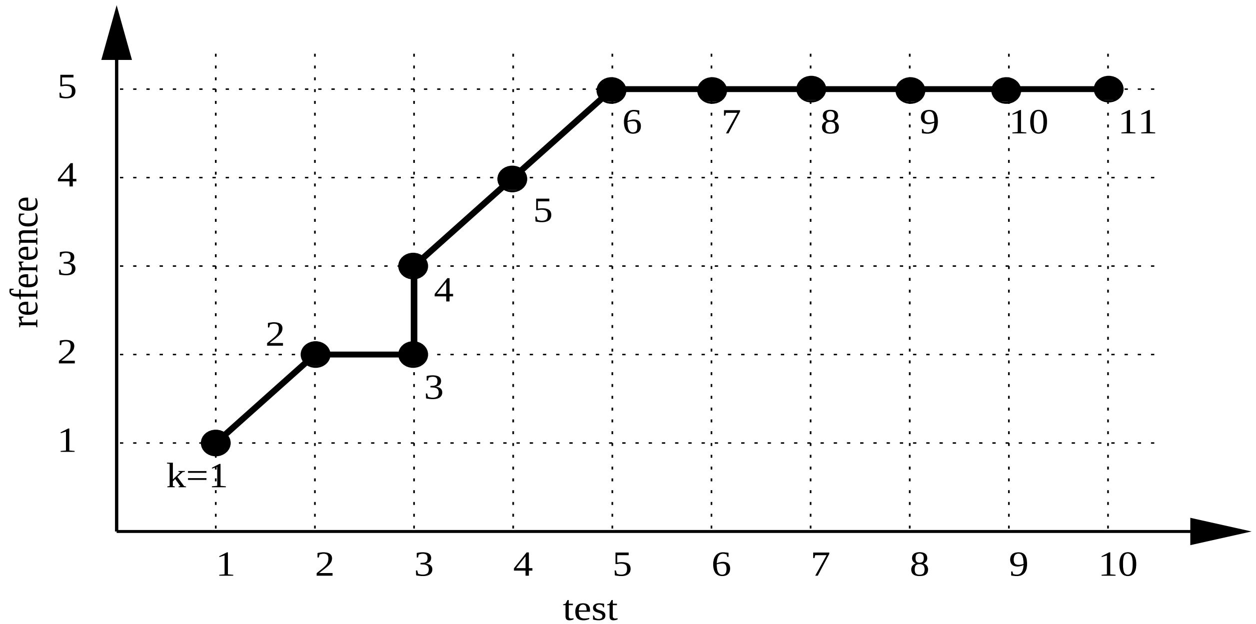


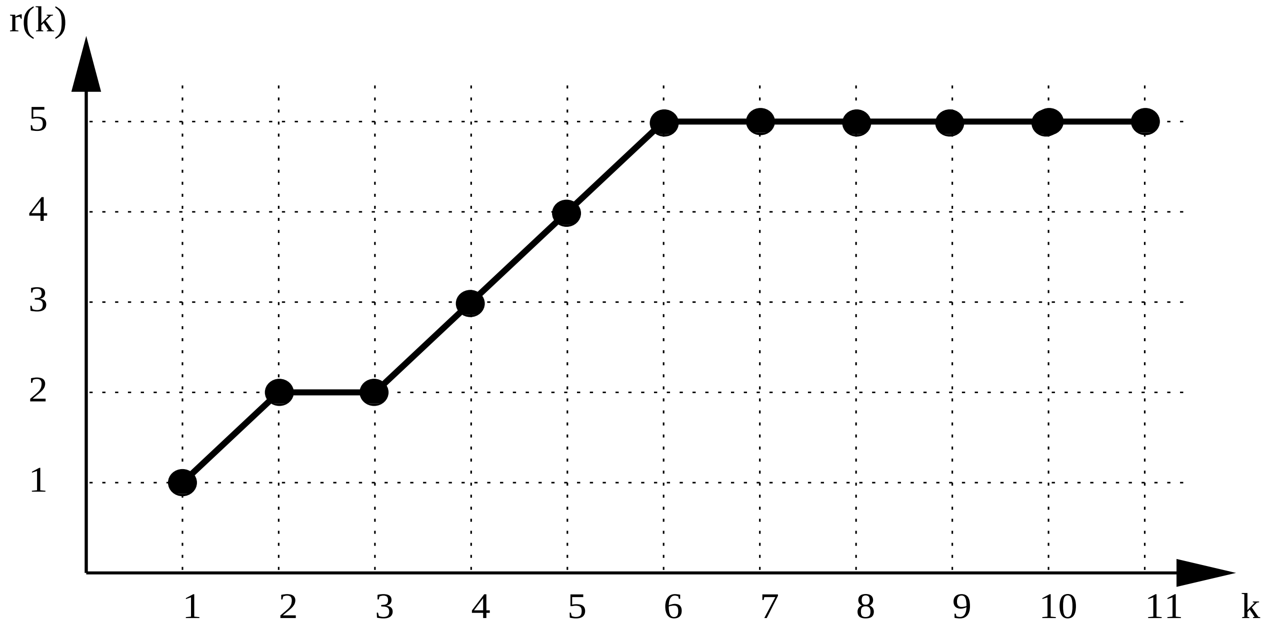
In the best case the alignment is *conducted* by the distances between single vectors
$\Rightarrow$ **Dynamic Time Warping (DTW)**.

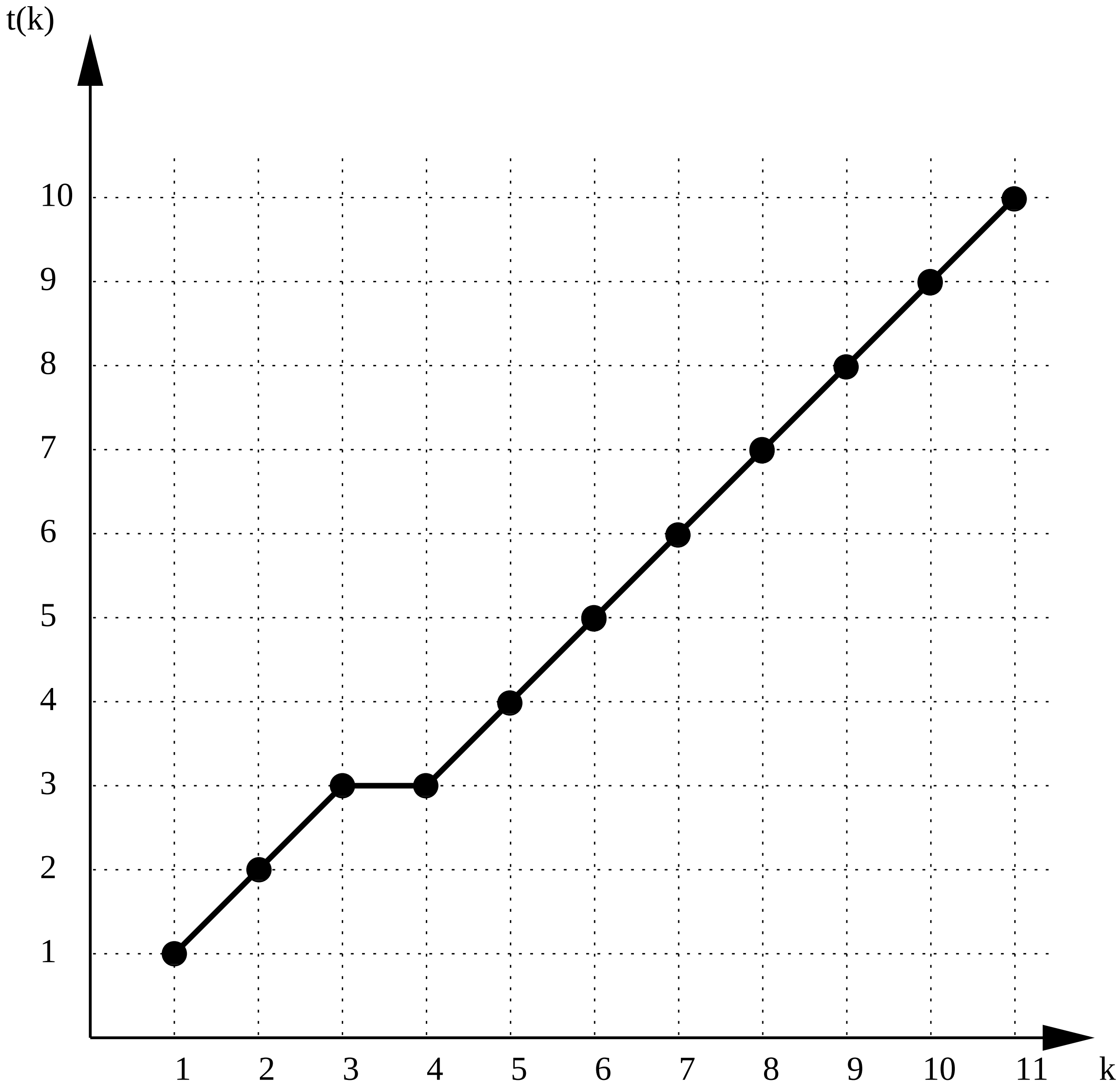
Define a time variable k and *two* transformation functions:

- $r(k)$ for the reference sequence.
- $t(k)$ for the test sequence.

The vector alignment can be represented by a **path**: The number of steps of the path is denoted as K. The reference is represented by the vertical axes and the test sequence is represented by the horizontal axes. According to the path, the functions $r(k)$ and $t(k)$ are estimated. The functions trace single sequences


reference
5
4
3
2
1
k=1
2
3
4
5
6
7
8
9
10
11
1
2
3
4
5
6
7
8
9
10
test


r(k)
5
4
3
2
1
1
2
3
4
5
6
7
8
9
10
11
k


t(k)
10
9
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
9
10
11
k

The path C is unambiguously given by its length K_C and the course of the functions $r_C(k)$ and $t_C(k)$. For this path the distance between the sequence $\mathbf{O}$ and $\mathbf{R}$ is given as:

$$D_C(\mathbf{O},\mathbf{R}) = \frac{\sum_{k=1}^{K_C} d[\mathbf{o}(t_C(k)),\mathbf{r}(r_C(k))]W_C(k)}{N_C} \quad (4)$$

where $d[\mathbf{o}(\cdot),\mathbf{r}(\cdot)]$ is length of the vectors, $W_C(k)$ is the weight corresponding to the k-th step and N_C is weight dependent normalization factor.

The distance between the sequences $\mathbf{O}$ and $\mathbf{R}$ is given as *minimum distance* over the set of all possible paths (all possible lengths, all possible courses):

$$D(\mathbf{O},\mathbf{R}) = \min_{\{C\}} D_C(\mathbf{O},\mathbf{R}). \quad (5)$$

We need to solve 3 things:

1. allowed courses of the functions $r(k)$ and $t(k)$. The path isn't allowed to take the opposite course, or skip frames, etc.
2. define normalization factors and the weighting function.
3. efficient and fast algorithm to calculate $D(\mathbf{O},\mathbf{R})$.

Path Constrain

1. **Start and terminal points**

$$\left.\begin{array}{l} r(1) = 1 \\ t(1) = 1 \end{array}\right\} \text{beginning} \qquad \left.\begin{array}{l} r(K) = R \\ t(K) = T \end{array}\right\} \text{end} \tag{6}$$

2. **Local correlation and local slope**

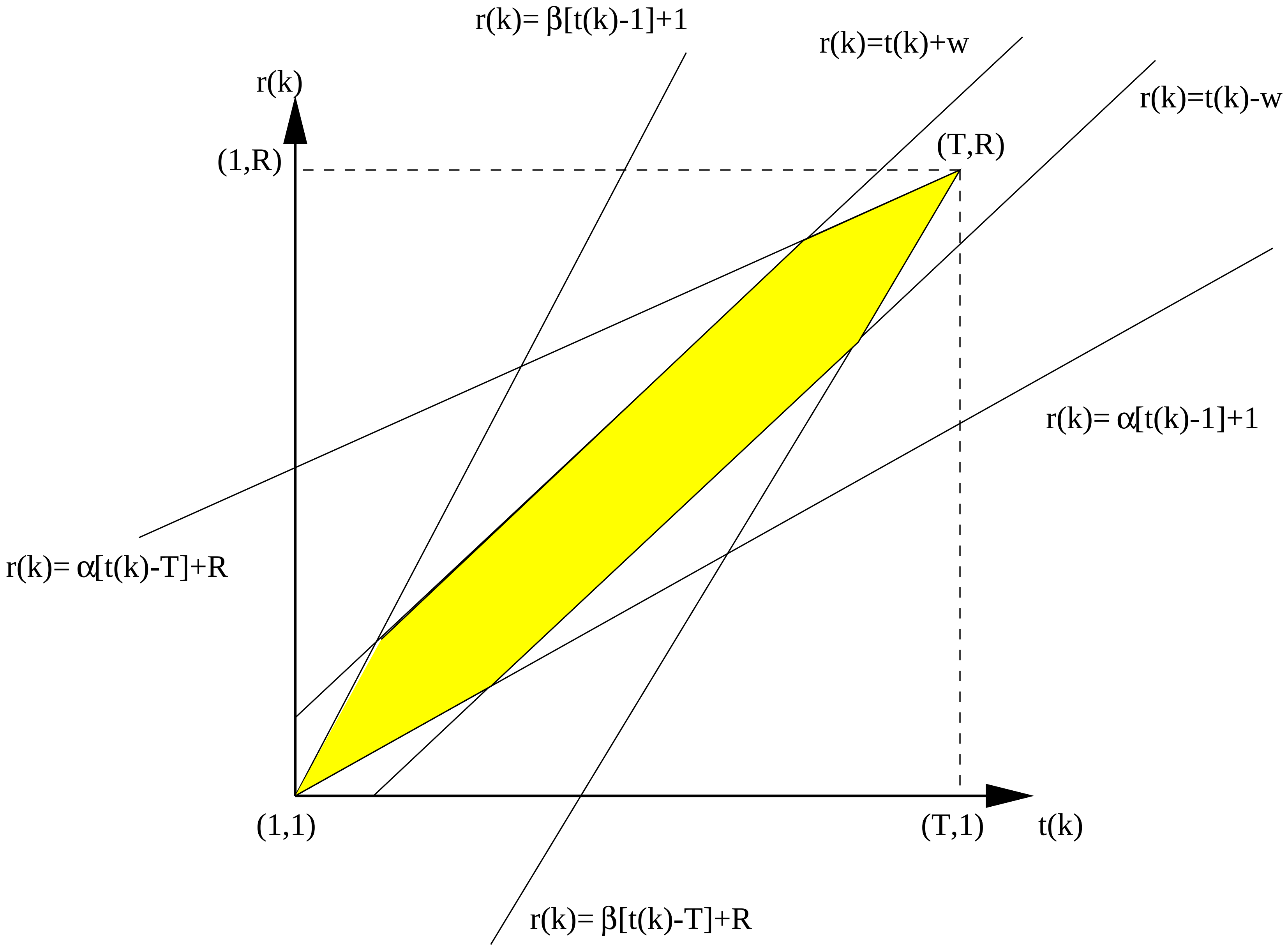
$$\begin{array}{c} 0 \leq r(k) - r(k-1) \leq R^\star \\ 0 \leq t(k) - t(k-1) \leq T^\star \end{array} \tag{7}$$

in pratise $R^\star, T^\star$=1,2,3.

- $R^\star, T^\star$=1: each vector should be considered at least once. $r(k) = r(k-1)$ denotes repeated use.
- $R^\star, T^\star > 1$: Vector(s) can be skipped.

3. **Global path restriction in DTW:** restriction using lines:

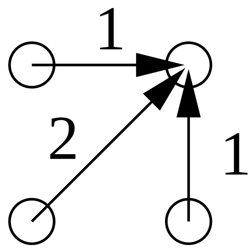
$$\begin{aligned} 1+\alpha[t(k)-1] \leq r(k) \leq 1+\beta[t(k)-1] \\ R+\beta[t(k)-T] \leq r(k) \leq R+\alpha[t(k)-T] \end{aligned} \quad (8)$$


r(k)= β[t(k)-1]+1
r(k)=t(k)+w
r(k)
r(k)=t(k)-w
(T,R)
(1,R)
r(k)= α[t(k)-1]+1
r(k)= α[t(k)-T]+R
(1,1)
(T,1)
t(k)
r(k)= β[t(k)-T]+R

Weighting Function Definition

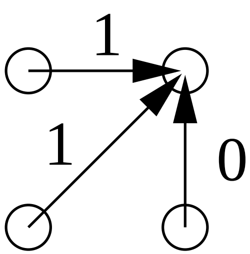
Weighting function $W(k)$ depends on local path progress. 4 types:

- **type a)** symmetric: $W_a(k) = [t(k) - t(k-1)] + [r(k) - r(k-1)]$.

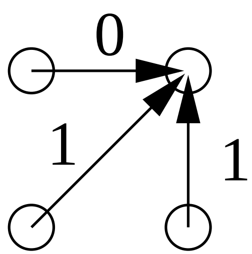


- **type b)** asymmetric:

b1) $W_{b1}(k) = t(k) - t(k-1)$



b2) $W_{b2} = r(k) - r(k-1)$



- **type c)** $W_c(k) = \min\{t(k) - t(k-1), r(k) - r(k-1)\}$
- **type d)** $W_d(k) = \max\{t(k) - t(k-1), r(k) - r(k-1)\}$

Normalization Factor

$$N = \sum_{k=1}^{K} W(k) \qquad (9)$$

For weighting function a) normalization factor is:

$$N_a = \sum_{k=1}^{K} [t(k) - t(k-1) + r(k) - r(k-1)] = t(K) - t(0) + r(K) - r(0) = T + R \qquad (10)$$

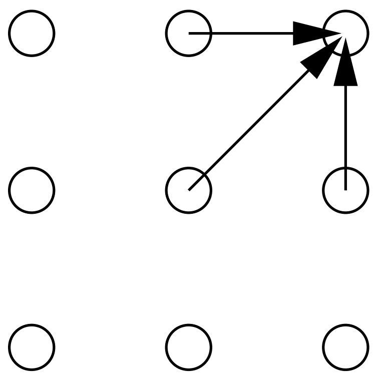
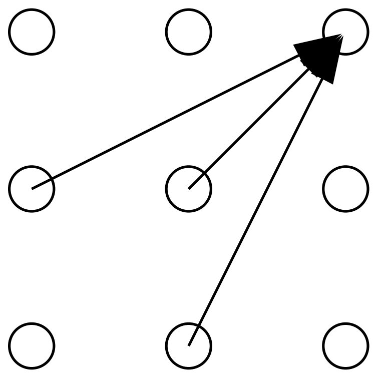
For weighting function b1) je normalization factor $N = T$.

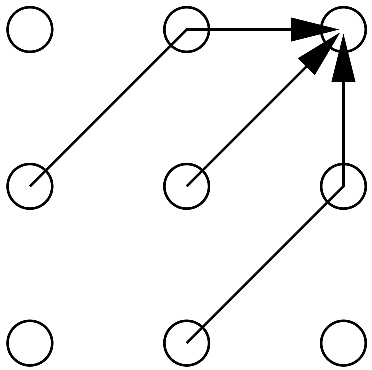
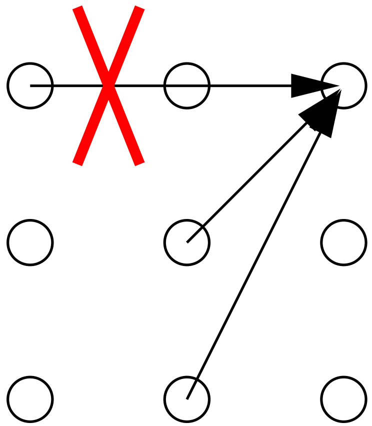
For weighting function b2) je normalization factor $N = R$.

For weighting function c), d) factor is strongly dependent on the path progress, better use constraint: $N = T$.

Local Restrictions of the Path

The table presents types of local restrictions and corresponding factors α and β. Meaning of $g(n, m)$ will be explained later.

Type		α	β	Type $w(k)$	$g(n,m)$
I.		0	∞	a	$\min \left\{ \begin{array}{l} g(n,m-1)+d(n,m) \\ g(n-1,m-1)+2d(n,m) \\ g(n-1,m)+d(n,m) \end{array} \right\}$
				d	$\min \left\{ \begin{array}{l} g(n,m-1)+d(n,m) \\ g(n-1,m-1)+d(n,m) \\ g(n-1,m)+d(n,m) \end{array} \right\}$
II.		$\frac{1}{2}$	2	a	$\min \left\{ \begin{array}{l} g(n-1,m-2)+3d(n,m) \\ g(n-1,m-1)+2d(n,m) \\ g(n-2,m-1)+3d(n,m) \end{array} \right\}$
				d	$\min \left\{ \begin{array}{l} g(n-1,m-2)+d(n,m) \\ g(n-1,m-1)+d(n,m) \\ g(n-2,m-1)+d(n,m) \end{array} \right\}$

III.		$\frac{1}{2}$	2	a	$\min \left\{ \begin{array}{l} g(n-1,m-2)+2d(n,m-1)+d(n,m) \\ g(n-1,m-1)+2d(n,m) \\ g(n-2,m-1)+2d(n-1,m)+d(n,m) \end{array} \right\}$
IV.		$\frac{1}{2}$	2	b1	$\min \left\{ \begin{array}{l} g(n-1,m)+kd(n,m) \\ g(n-1,m-1)+d(n,m) \\ g(n-1,m-2)+d(n,m) \end{array} \right\}$ where $k=1$ for $r(k-1) \neq r(k-2)$ $k=\infty$ for $r(k-1)=r(k-2)$

Efficient Calculation $D(\mathbf{O}, \mathbf{R})$

Minimum distance computation

$$D(\mathbf{O}, \mathbf{R}) = \min_{\{C\}} D_C(\mathbf{O}, \mathbf{R}). \qquad (11)$$

is simple, when normalization factor N_C is no function of path and we can write:

$$N_C = N \quad \text{for} \quad \forall C$$

$$D(\mathbf{O}, \mathbf{R}) = \frac{1}{N} \min_{\{C\}} \sum_{k=1}^{K_C} d[\mathbf{o}(t_C(k)), \mathbf{r}(r_C(k))] W_C(k) \qquad (12)$$

Procedure is the following:

1. the cell $\mathbf{d}$ of the size $T \times R$ contains distances between the reference and the test vector, all by all.
2. define cell $\mathbf{g}$ with *partial cumulated distance*. Compared to cell $\mathbf{d}$, $\mathbf{g}$ has zero row and zero column, that are initialized to:

$$g(0,0) = 0, \quad \text{a} \quad g(0, m \neq 0) = g(n \neq 0, 0) = \infty.$$

3. partial cumulated distance (for each point) is calculated as:

$$g(m,n) = \min_{\forall \text{predcesors}} \left[g(\text{predcesor}) + d(m,n)w(k)\right] \tag{13}$$

- predecessors are given by the restriction path table.
- weight $w(k)$ corresponds to the $[m,n]$ point pass (from the predecessor).
- relations for the partial cumulated distance are tabled

4. Final minimum normalized distance is thus given by :

$$D(\mathbf{O},\mathbf{R}) = \frac{1}{N} g(T,R) \tag{14}$$

Example

d

ref.

4	3	2
2	3	1
4	2	3
0	1	1

test

g

ref.

inf	10	9	7
inf	6	6	5
inf	4	3	5
inf	0	1	2
0	inf	inf	inf

test

Result:

- given distance $D = \frac{1}{3+4} 7 = 1$.
- we can "step the optimal path" backward (the path has 5 steps): $t(k) = [1\ 2\ 2\ 3\ 3]$, $r(k) = [1\ 1\ 2\ 3\ 4]$.

DTW based Recognizer

Recap: what do we want? Given a word $\mathbf{O}$ and a set of classes; we want to decide to which class, ω_r, the word belongs. We dispose of $\check{N}$ classes, representing words (e.g. "one", "two", "three", etc.).

Training – creating references or classes of patterns

during training, we dispose of data sequences from one or more speakers and we know two which class each of them belong.

1. **simple:** each class ω_r is represented by one reference $\mathbf{R}_r$.
2. **advanced:** each class ω_r is represented by several references: $\mathbf{R}_{r,1} \dots \mathbf{R}_{r,\check{N}_r}$. These can be stored (in the vocabulary) as generated or normalized to have same length:

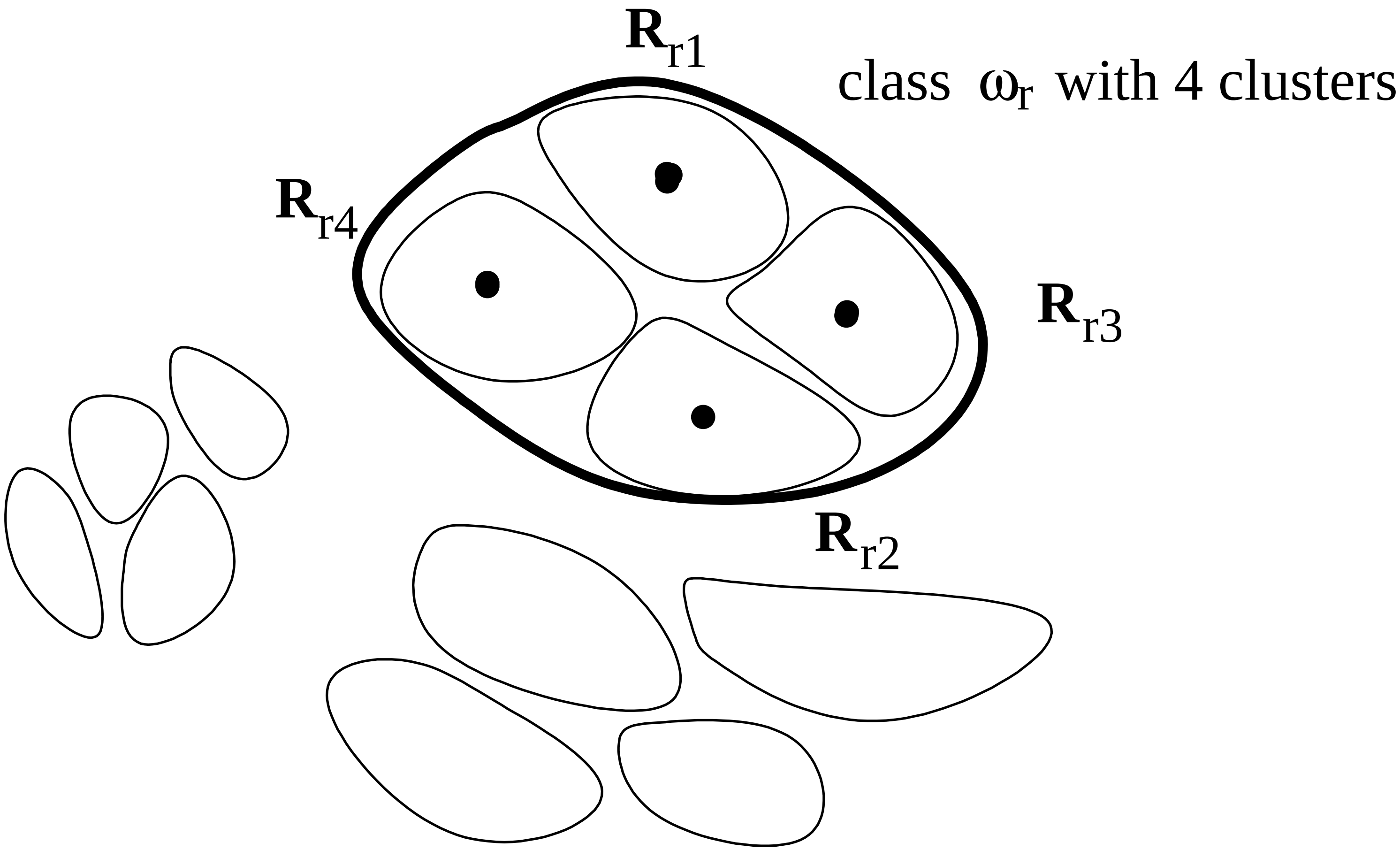
$$\bar{R} = \frac{1}{N} \sum_{r=1}^{N} \left[\frac{1}{\check{N}_r} \sum_{i=1}^{\check{N}_r} R_{r_i} \right], \quad (15)$$

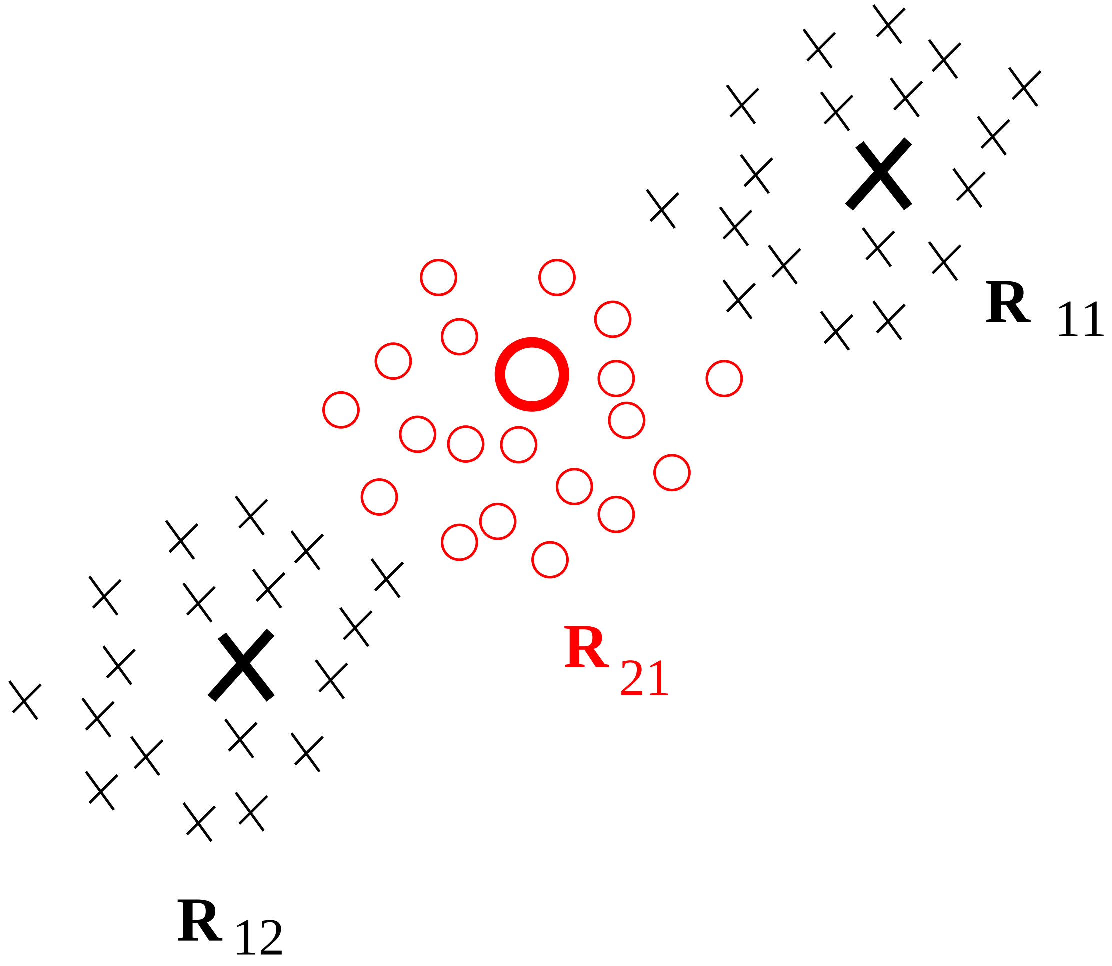
   where R_{r_i} is the length of the i-th sample of the class ω_r.
3. **average class pattern** ω_r:
   - linear averaging – average of the linearly aligned vectors. Danger: can result in nonsense pattern ...
   - dynamic averaging:

   (a) select a sample with appropriate length;

   (b) average samples aligned to this length using DTW.

4. training using *clustering*. Clusters are created to minimize within class variability and maximize across-class variability. There are several algorithms available, e.g.
   Mac Queen algorithm: align all the references to one cluster; the most distanced samples are split off the cluster thus forming new clusters; the data are realigned, etc..
   Clusters are represented by *centroids* $\mathbf{R}_{ri}$. Advantage over averaging is that classes can have more complicated structure.




R 11
R 21
R 12

Recognition (Classification)

If each class is represented by one reference, classification is easy:

$$\omega_r^\star = \arg\min_r D(\mathbf{O}, \mathbf{R}_r) \quad \text{pro} \quad r = 1, \ldots, N \tag{16}$$

When classes are presented each by several references, we can approach one of the two solutions:

1. **1-NN** nearest neighbor:

$$\omega_r^\star = \arg\min_{r,i} D(\mathbf{O}, \mathbf{R}_{r_i}) \quad \text{pro} \quad \begin{array}{l} r = 1, \ldots, N \\ i = 1, \ldots, N_r \end{array} \tag{17}$$

2. k-**NN** k nearest neighbors:

- for each class, calculate all distances $D(\mathbf{O}, \mathbf{R}_{ri})$ and sort them from the best to the worst:

$$D(\mathbf{O}, \mathbf{R}_{r_{(1)}}) \leq D(\mathbf{O}, \mathbf{R}_{r_{(2)}}) \leq \ldots \leq D(\mathbf{O}, \mathbf{R}_{r_{(N_r)}}) \quad (18$$

- sample $\mathbf{O}$ is assigned to the class ω_r according to the average distance of the k nearest neighbors:

$$\omega_r^{\star} = \arg\min_r \frac{1}{k} \sum_{i=1}^{k} D(\mathbf{O}, \mathbf{R}_{r_{(i)}}) \quad (19$$