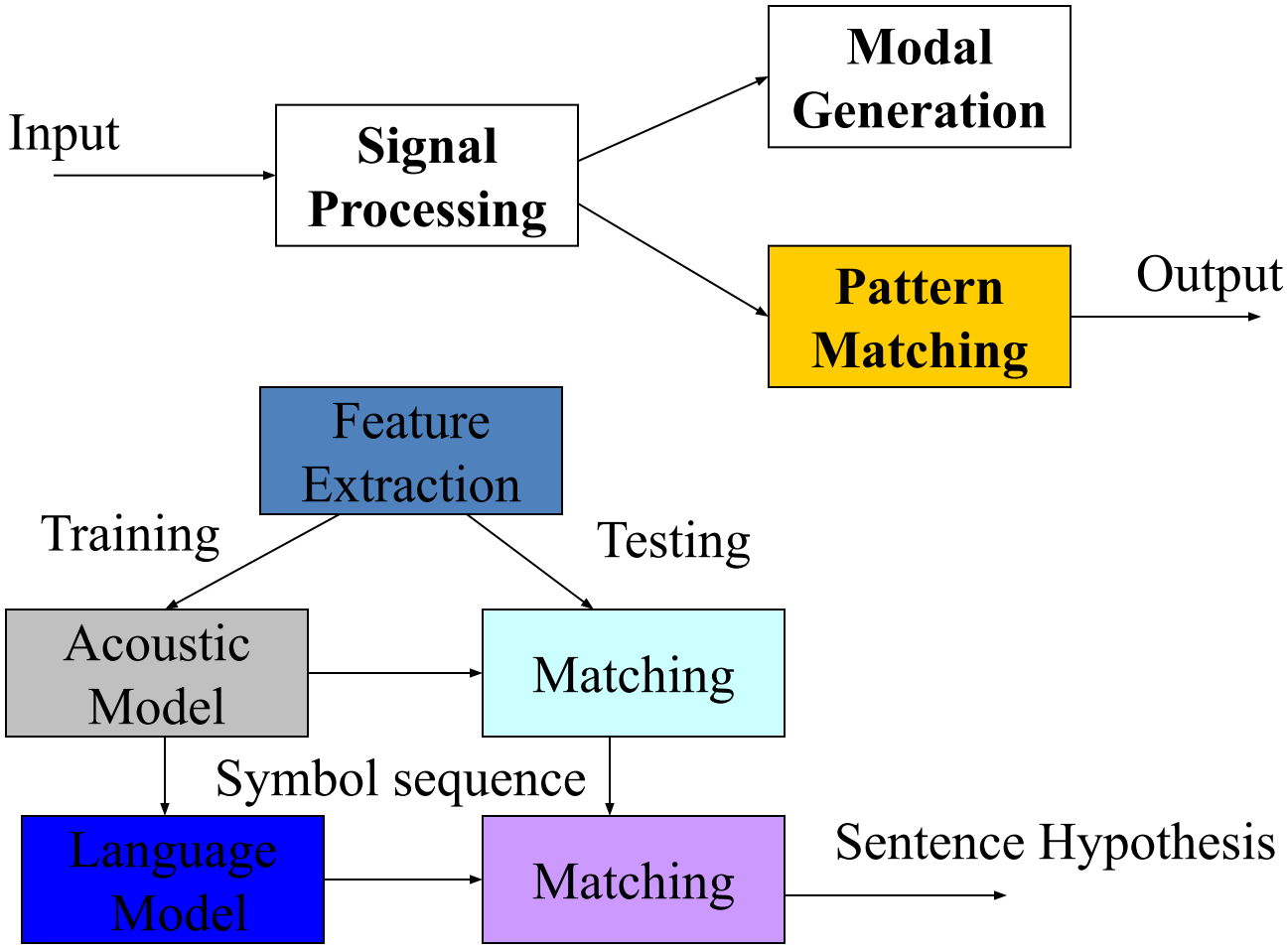
Automatic Speech Recognition (ASR) -

- **ASR**
  - incoming acoustic waveform ->sequence of linguistic words

Speech is the most natural means of communication for humans and usually learned to a high degree of proficiency at a very early age.

- ASR Applications:
  - Natural human-machine communication
  - Interactive problem solving
  - Telecommunications
    - Cost reduction (replace human operators)
    - Revenue generation (new services)
  - Hands-free operation
  - Dictation & automatic transcription
  - Aids for handicapped
  - Automatic language translation

Speech Recognition



The Speech Recognition Problem

- If we know the signal patterns that represent every spoken word beforehand, we could try to identify the words whose patterns best match the input
  - word patterns are never reproducible exactly
  - How do we represent these signal patterns?
  - Given this uncertainty, how do we compare the input to known patterns?
- Speech recognition is the study of these problems

Why is Speech Recognition Difficult?

- Tremendous range of variability in speech, even though the message may be constant:
  - Human physiology: squeaky voice *vs* deep voice
  - Speaking style: clear, spontaneous, slurred or sloppy
  - Speaking rate: fast or slow speech Speaking rate can change within a single sentence
  - Emotional state: happy, sad, etc.
  - Emphasis: stressed speech *vs.* unstressed speech
  - Accents, dialects, foreign words
  - Environmental or background noise
  - *Even the same person never speaks exactly the same way twice*
- In addition:
  - Large vocabulary and infinite language
  - Absence of word boundary markers in continuous speech
  - Inherent ambiguities: “I scream” or “Ice cream”?

Technological Challenges

- Representations of spoken words are inexact
  - Even the same person never says a given sentence exactly the same way twice
  - No representation can capture the infinite range of variations
  - Yet, humans have apparently no difficulty
    - They *adapt* to new situations effortlessly
  - The problem is understanding and representing what is *invariant*
- Pattern matching is necessarily inexact
  - Given the above, there will always be mismatches in pattern matching, and hence misrecognitions
    - Even humans are not perfect!
  - Finding *optimal* pattern matching algorithms, and hence minimizing misrecognitions, is another challenge

Technological Challenges (contd.)

- As target vocabulary size increases, complexity increases
  - Computational resource requirements increase
    - Memory size to store patterns
    - Computational cost of matching
  - Most important, the degree of confusability between words increases
    - More and more words begin sounding alike
    - Requires finer and finer models (patterns)
    - Further aggravates the computational cost problem

Speech recognition techniques

- Template based approach
- Acoustic phonetic or knowledge based approach
- Statistical approach
- Learning based approach
- Artificial Intelligence (AI) approach
- Manner Based Lexically Driven approach

Frequency Domain Parameters

- Filter Bank Analysis
- Short-term spectral analysis
- CC
- Formant Parameters
- MFCC, Delta MFCC, Delta-Delta MFCC

Time Domain Parameters

- LPC
- Shape Parameters

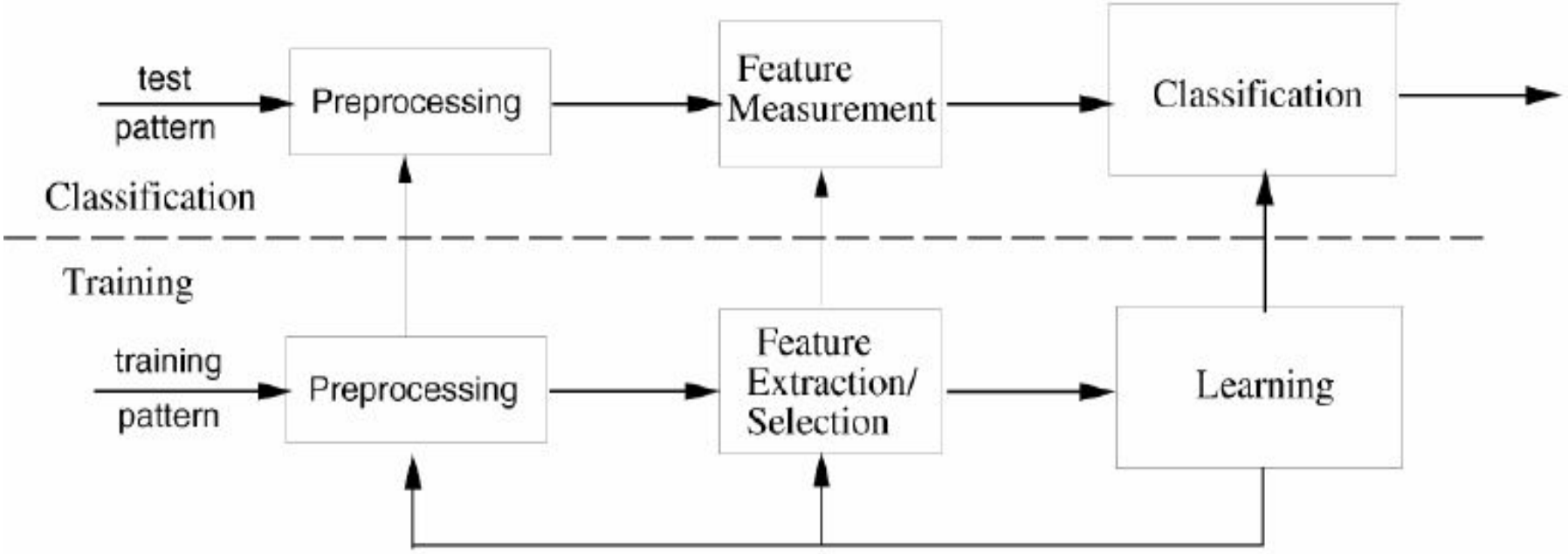
Perceptual Linear Prediction (PLP):

- Combines the LPC and filter-bank approaches by fitting an all-pole model to the set of energies produced by a perceptually motivated filter bank and then computing the cepstrum from the model parameters.

Pattern Recognition

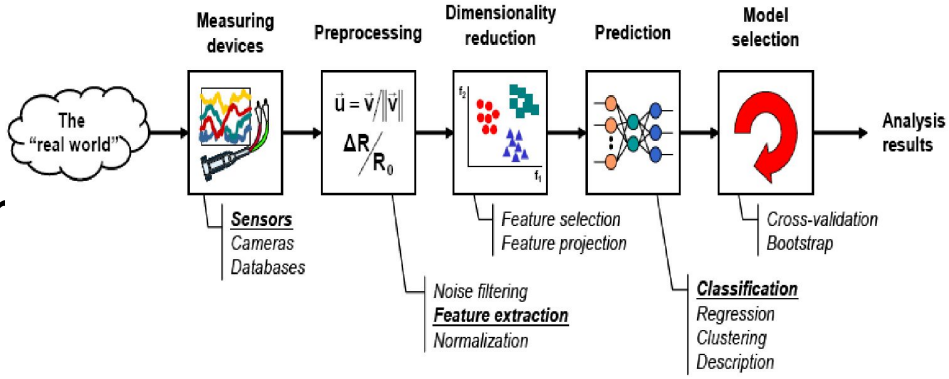
- **What is Pattern Recognition?**
  - "The assignment of a physical object or event to one of several pre-specified categories" -- Duda & Hart
- A **pattern** is an object, process or event that can be given a name.
- A **pattern class** (or category) is a set of patterns sharing common attributes and usually originating from the same source.
- During **recognition** (or **classification**) given objects are assigned to prescribed classes.
- A **classifier** is a machine which performs classification
- **Approaches**:
  - **Statistical PR:** based on underlying statistical model of patterns and pattern classes.
  - **Structural (or syntactic) PR:** pattern classes represented by means of formal structures as grammars, automata, strings, etc.

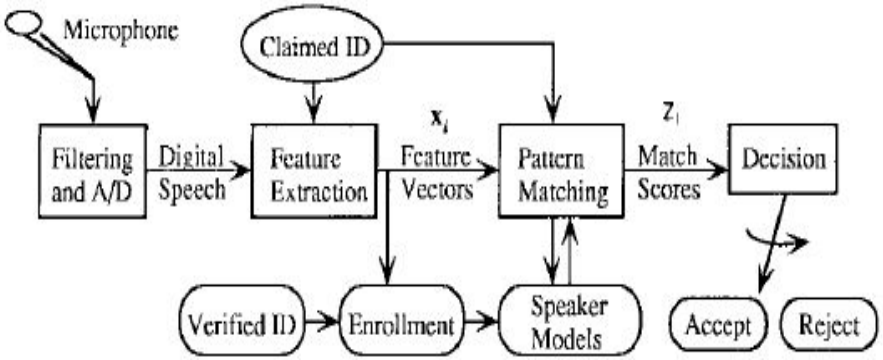
Pattern Recognition



Components of PR System

- **A basic pattern classification system contains**
  - **Data Acquisition**
  - **Feature Extraction mechanism (manual or automated)**
  - **Pattern Matching**
  - Making an Accept/Reject Decision (Classification algorithm)
  - **A set of examples (training set) already classified or described**
    - **Speaker Reference Models**





Speech recognition techniques

- Template based approach
- Acoustic phonetic or knowledge based approach
- Statistics based approach
- Learning based approach
- Artificial Intelligence (AI) approach
- Manner Based Lexically Driven approach

ASR: Pattern Matching Methods

- **Vector Quantization (VQ)**
  - Code book Models are used
  - A VQ codebook is designed by standard clustering procedures for each enrolled speaker using his training data, usually based upon reading a specific text.
  - The pattern match score is the distance between an input vector and the minimum distance code word in the VQ codebook C
- **Dynamic Time Warping (DTW)**
  - Template Models are used
  - A text-dependent template model is a sequence of templates (X1 ..XN) that must be matched to an input sequence (Y1 ..YM).
- **Hidden Markov Models (HMM)**
  - Statistical Models are used
  - Using a stochastic model, the pattern-matching problem can be formulated as measuring the likelihood of an observation (a feature vector of a collection of vectors from the unknown speaker) given the speaker model

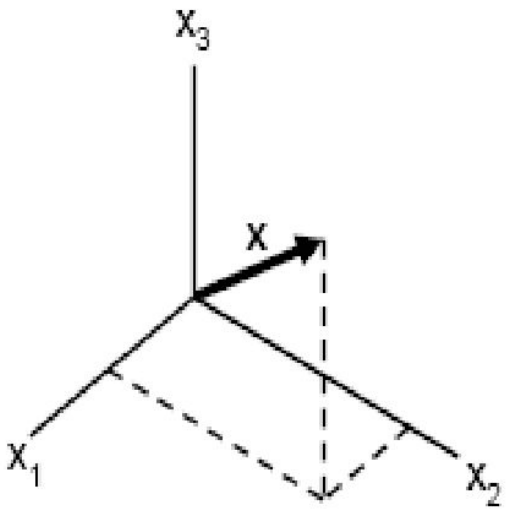
PR System: Features and Patterns

- **Feature**
  - Feature is any distinctive aspect, quality or characteristic
    - Features may be symbolic (i.e., color) or numeric (i.e., height)
  - Definitions
    - The combination of d features is represented as a d-dimensional column vector called a **feature vector**
    - The d-dimensional space defined by the feature vector is called the **feature space**
    - Objects are represented as points in feature space. This representation is called a **scatter plot**
- **Pattern**
  - Pattern is a composite of traits or features characteristic of an individual
  - In classification tasks, a pattern is a pair of variables $\{x,\omega\}$ where
    - x is a collection of observations or features (feature vector)
    - ω is the concept behind the observation (label)

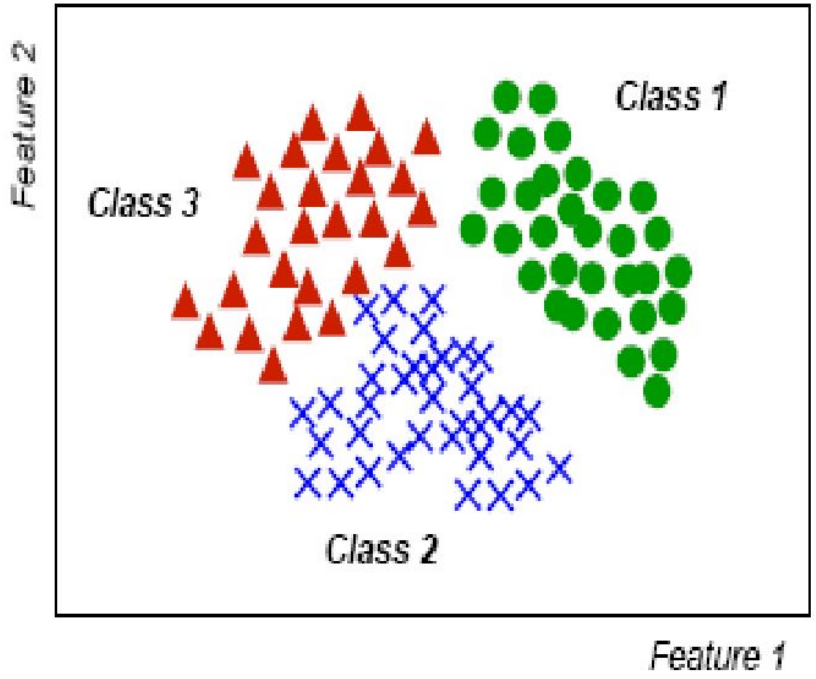
PR System: Features and Patterns

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \\ x_d \end{bmatrix}$$

Feature vector



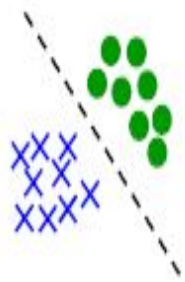
Feature space (3D)



Scatter plot (2D)

PR System: Features and Patterns

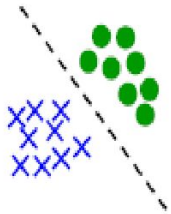
- **What makes a "good" feature vector?**
  - The quality of a feature vector is related to its ability to discriminate examples from different classes
    - Examples from the same class should have similar feature values
    - Examples from different classes have different feature values



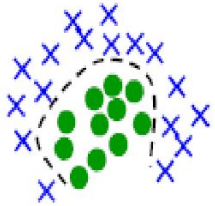
"Good" features

"Bad" features

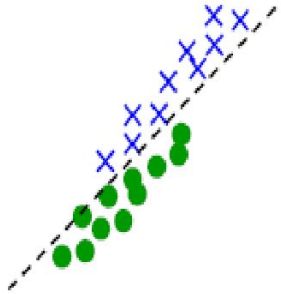
- **More feature properties**



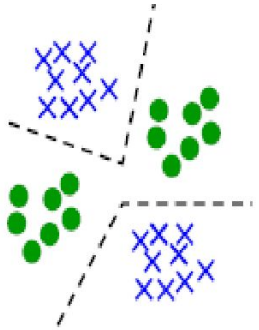
Linear separability



Non-linear separability



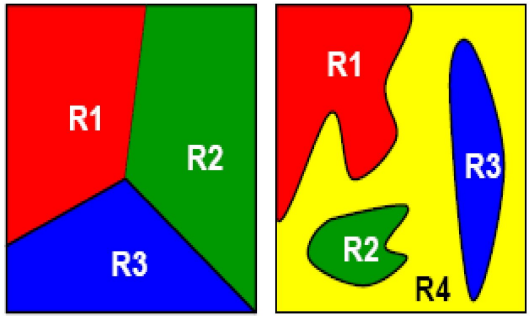
Highly correlated features

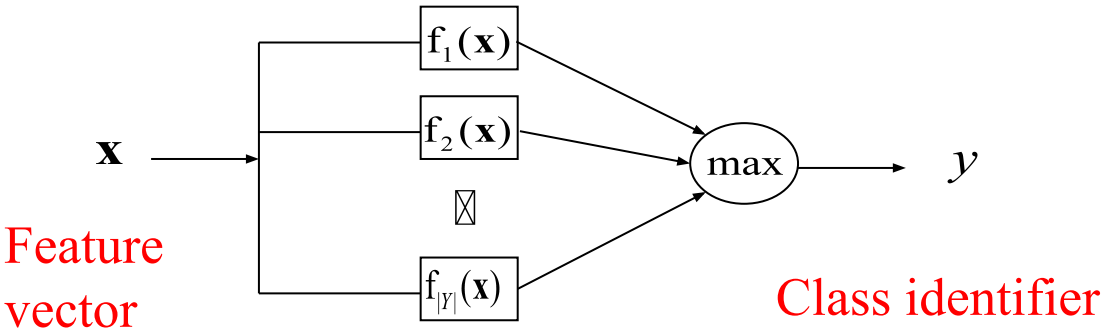


Multi-modal

PR System: Classifiers

- The task of a classifier is to partition feature space into class-labeled decision regions
  - Borders between decision regions are called **decision boundaries**
  - The classification of feature vector ***x*** consists of determining which decision region it belongs to, and assign ***x*** to this class
- A classifier can be represented as a set of discriminant functions
  - The classifier assigns a feature vector ***x*** to class ωt if fi(x)>fj(x); j‡ i

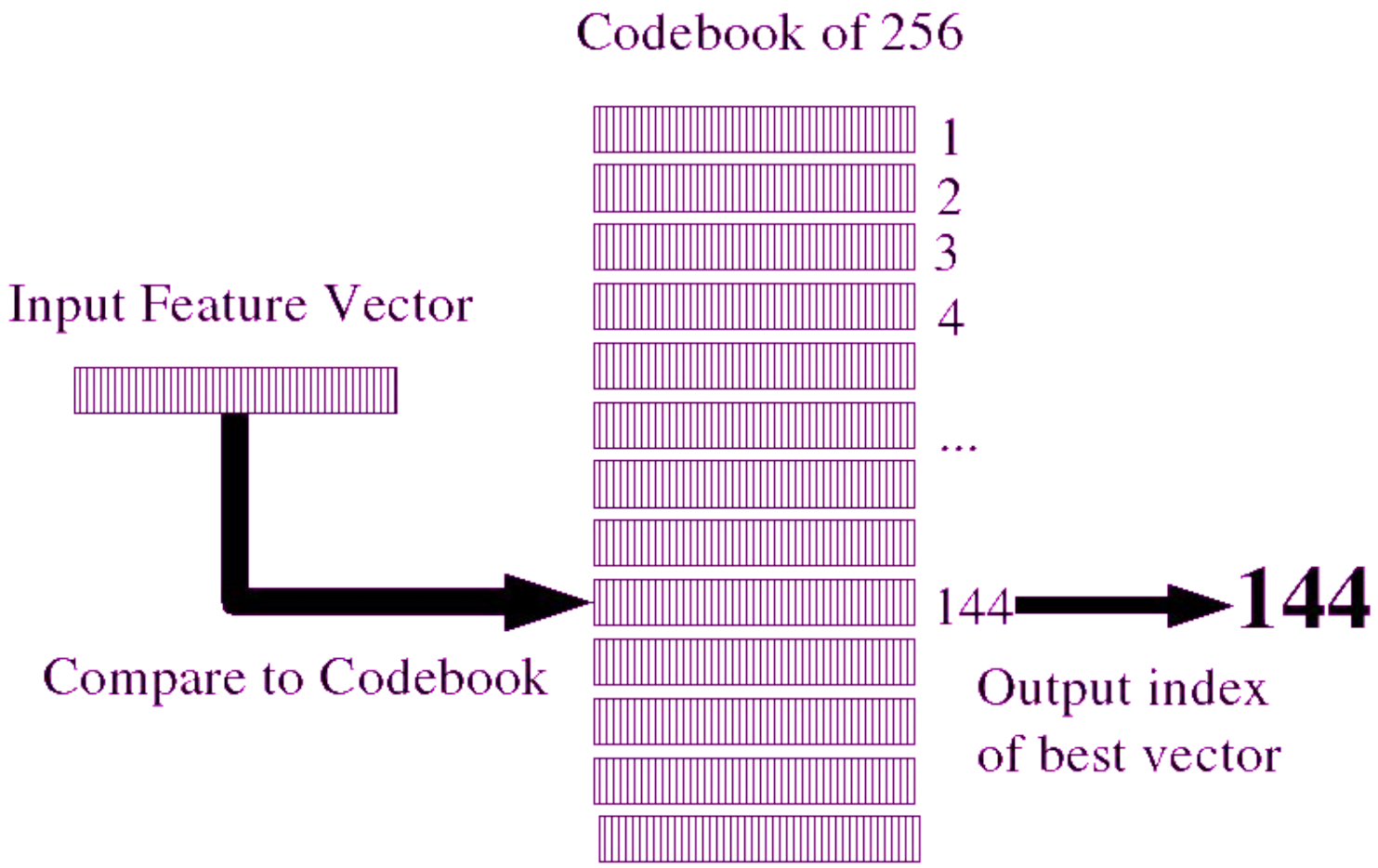




Vector Quantization

- Create a training set of feature vectors
- Cluster them into a small number of classes
- Represent each class by a discrete symbol
- We'll define a
  - Codebook, which lists for each symbol
  - A prototype vector, or codeword
- If we had 256 classes ('8-bit VQ'),
  - A codebook with 256 prototype vectors
  - Given an incoming feature vector, we compare it to each of the 256 prototype vectors
  - We pick whichever one is closest (by some 'distance metric')
  - And replace the input vector by the index of this prototype vector

VQ


Codebook of 256
1
2
3
4
...
144
144
Input Feature Vector
Compare to Codebook
Output index
of best vector

VQ requirements

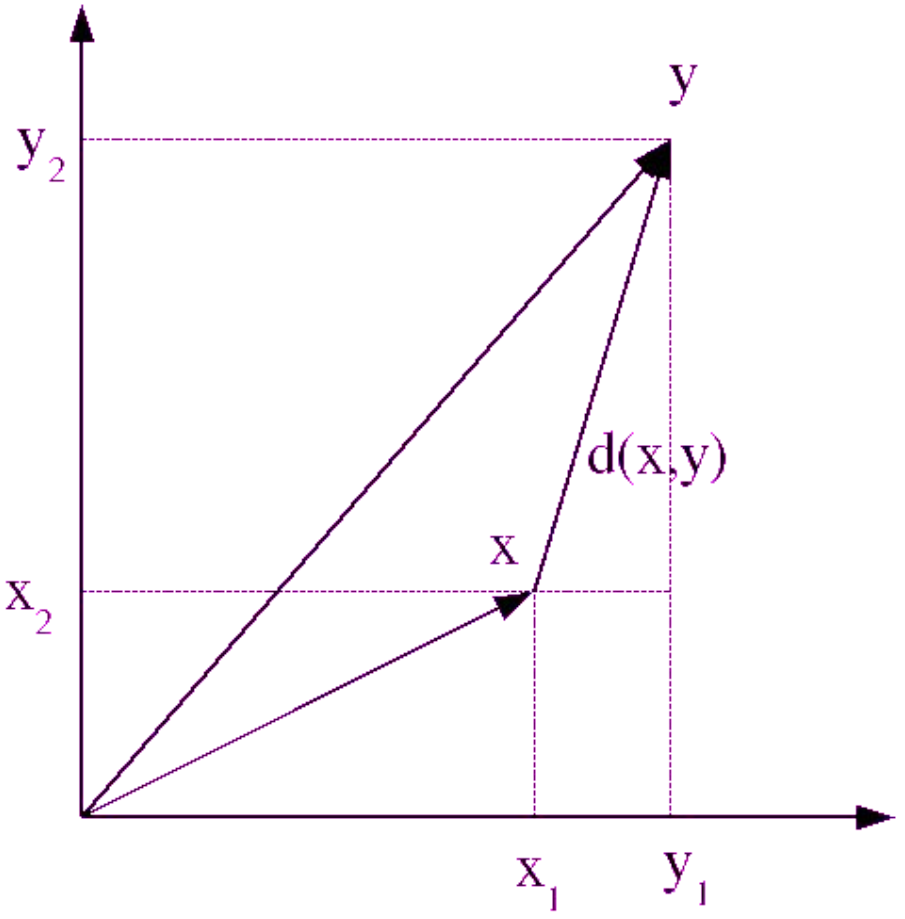
- A distance metric or distortion metric
  - Specifies how similar two vectors are
  - Used:
    - to build clusters
    - To find prototype vector for cluster
    - And to compare incoming vector to prototypes
- A clustering algorithm
  - K-means, etc.

Distance metrics

- Simplest:
  - (square of) Euclidean distance

$$d^2(x,y) = \sum_{i=1}^{D} (x_i - y_i)^2$$

  - Also called ‘sum-squared error’



Distance metrics

- More sophisticated:
  - (square of) Mahalanobis distance
  - Assume that each dimension of feature vector has variance σ^2

$$d^2(x,y) = \sum_{i=1}^{D} \frac{(x_i - y_i)^2}{\sigma_i^2}$$

  - Equation above assumes diagonal covariance matrix

Training a VQ system (generating codebook): K-means clustering

1. Initialization
   choose M vectors from L training vectors (typically $M=2^B$)
   as initial code words… random or max. distance.

2. Search:
   for each training vector, find the closest code word,
   assign this training vector to that cell

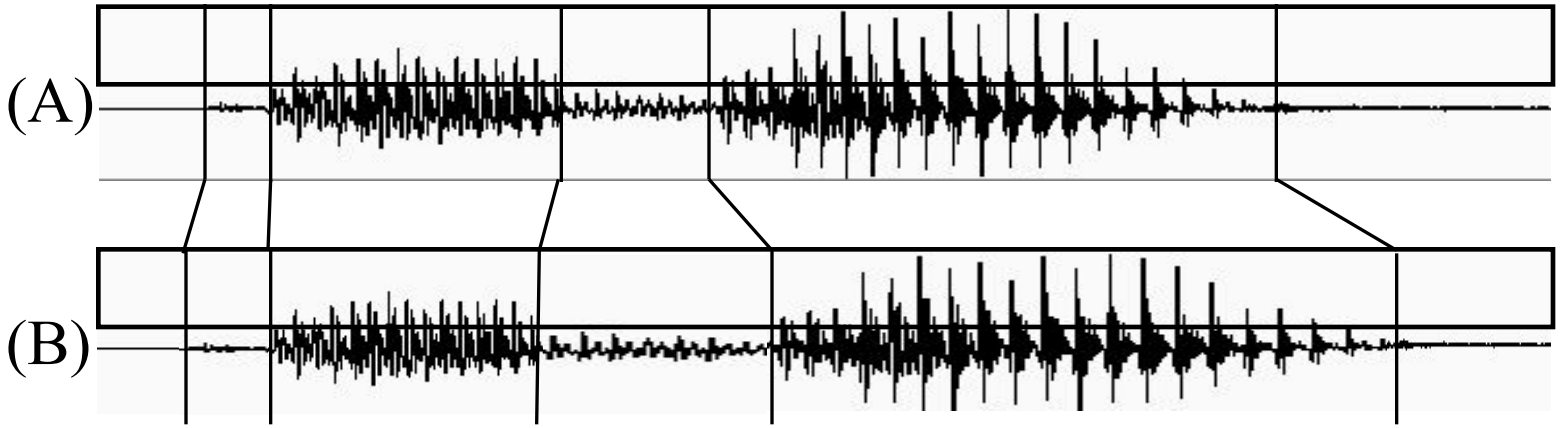
3. Centroid Update:
   for each cell, compute centroid of that cell. The
   new code word is the centroid.

4. Repeat (2)-(3) until average distance falls below threshold (or no change)

Slide from John-Paul Hosum, OHSU/OGI

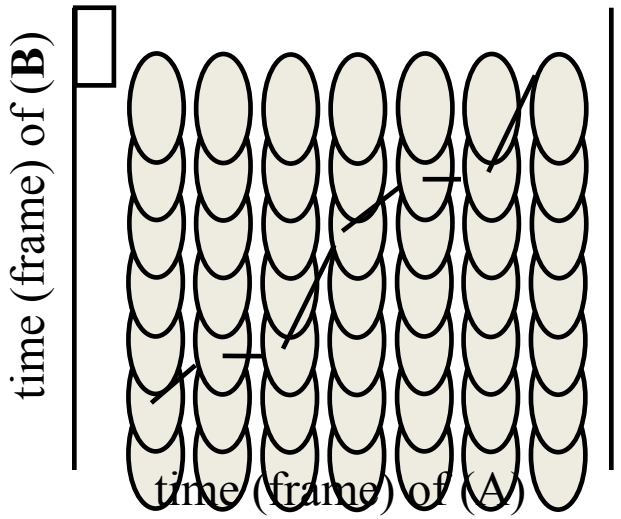
Dynamic Time Warping (DTW)

- Goal: Given two utterances, find “best” alignment between pairs of frames from each utterance.



The path through this matrix shows the best pairing of frames from utterance A with utterance B:

This path can be considered the best “warping” between A and B.



Dynamic Time Warping (DTW)

- Dynamic Time Warping
  - Requires measure of “distance” between 2 frames of speech, one frame from utterance A and one from utterance B.
  - Requires heuristics about allowable transitions from one frame in A to another frame in A (and likewise for B).
  - Uses inductive algorithm to find best warping.
  - Can get total “distortion score” for best warped path.
- Distance:
  - Measure of dissimilarity of two frames of speech
- Heuristics:
  - Constrain begin and end times to be (1,1) and (T,T)
  - Allow only monotonically increasing time
  - Don’t allow too many frames to be skipped
  - Can express in terms of “paths” with “slope weights”

Dynamic Time Warping (DTW)

- Does not require that both patterns have the same length
- We may refer to one speech pattern as the “input” and the other speech pattern as the “template”, and compare input with template.
- For speech, we divide speech signal into equally-spaced frames (e.g. 10 msec) and compute one set of features per frame. The local distance measure is the distance between features at a pair of frames (one from A, one from B).
- Local distance between frames called *d*. Global distortion from beginning of utterance until current pair of frames called *D*.
- DTW can also be applied to related speech problems, such as matching up two similar sequences of phonemes.

Template Matching: DTW

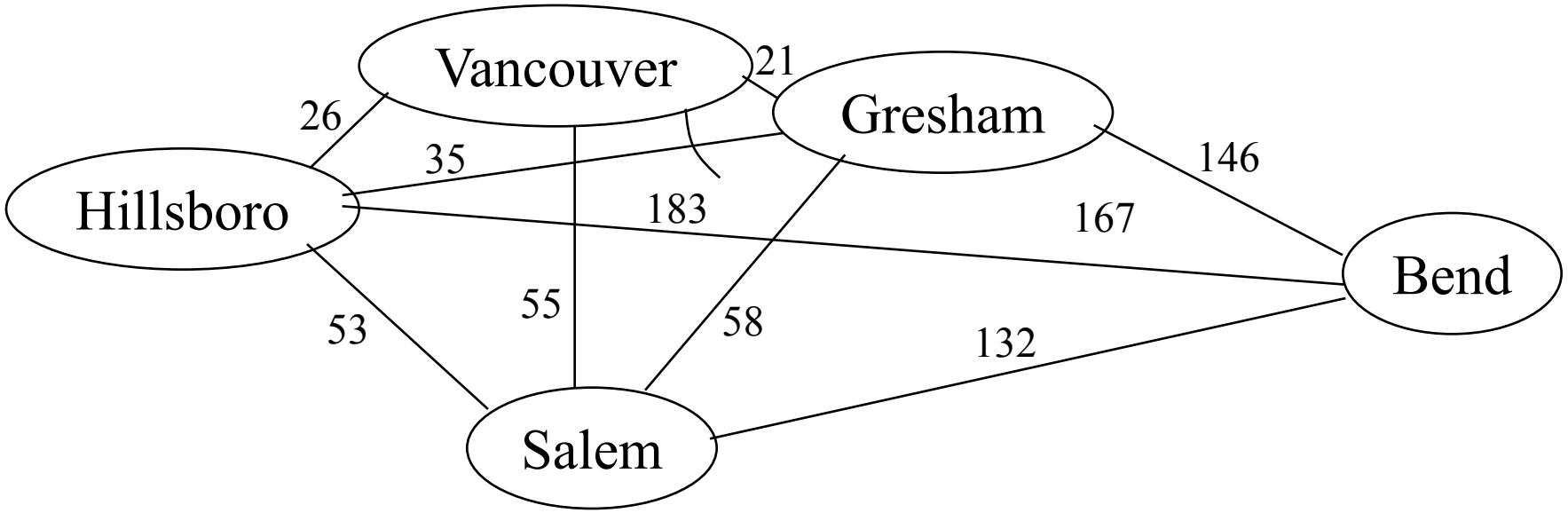
- Clearly, it is necessary to *dynamically* adjust the durations of various *segments* within the input and template speech
  - How to optimally (and automatically!) *align* the template to the input speech to obtain the best possible match?
  - Brute force? Try all possible segmentations and alignments?
    - Computationally far too expensive!
    - Besides, precise segment boundaries may not even be known
- An efficient algorithm for finding an optimal alignment exists:
  - *Dynamic Time Warping* or DTW
- DTW in speech recognition is essentially based on *dynamic programming* (DP)
  - DP is a well known optimization algorithm in Operations Research

Induction (Dynamic Programming)

- To find value *X* at step *t* in a process (*X*(*t*)), where *X*(*t*) can be computed from *X*(*t*-1):
  1. Compute *X*(1)
  2. For *m* = 2 to *t:*
     1. Use value from previous iteration (*X*(*m-1*)) to determine *X*(*m*)
  3. *X*(*t*) is last result from Step (2).
- For speech, *X*(*t*) will be the "best" value at time *t*, either in terms of "least distortion" or "highest probability".
- By showing that the best value at time *t* depends only on the previous values at time *t*-1, the best value for an entire utterance (the end of the signal, time *T*) can be computed.

Induction (Dynamic Programming)

- **"Greedy Algorithm":**
  - **Make a *locally-optimum* choice going forward at each step, hoping (but not guaranteeing) that the *globally-optimum* will be found at the last step.**
- ***Travelling Salesman Problem*:**
  - **Given a number of cities, what is the shortest route that visits each city exactly once and then returns to the starting city?**

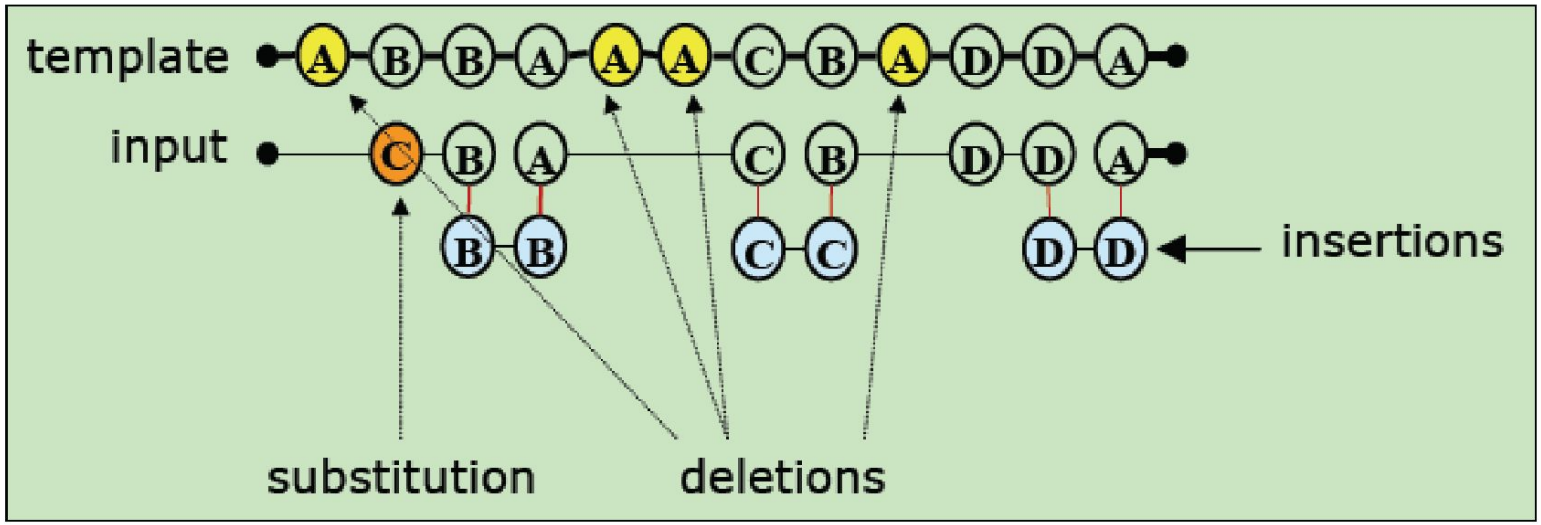


DP: String Matching Example

- First, consider a similar problem, but visually easier to grasp: Aligning two text strings to find how similar they are
  - *E.g.* spelling errors detected by google:
    - Type query string "RaviShanker" into google
    - Google responds: Did you mean "Ravi*Shankar*"?
    - How does it know this?
- "RaviShankar" is its template (among many others), and "RaviShanker" is the input
  - By measuring the dissimilarity between the two, it could guess that perhaps the template was the intended input

String Matching Example

- The example contained just a substitution error (*e* for *a*)
- In general, there can be three types of string errors:
  - ***Substitution***: a template letter has been changed in the input
  - ***Insertion***: a spurious letter is introduced in the input
  - ***Deletion***: a template letter is missing from the input
- Hypothetical example:
  - Template : ABBAAACBADDA
  - Input : CBBBACCCBDDDDA

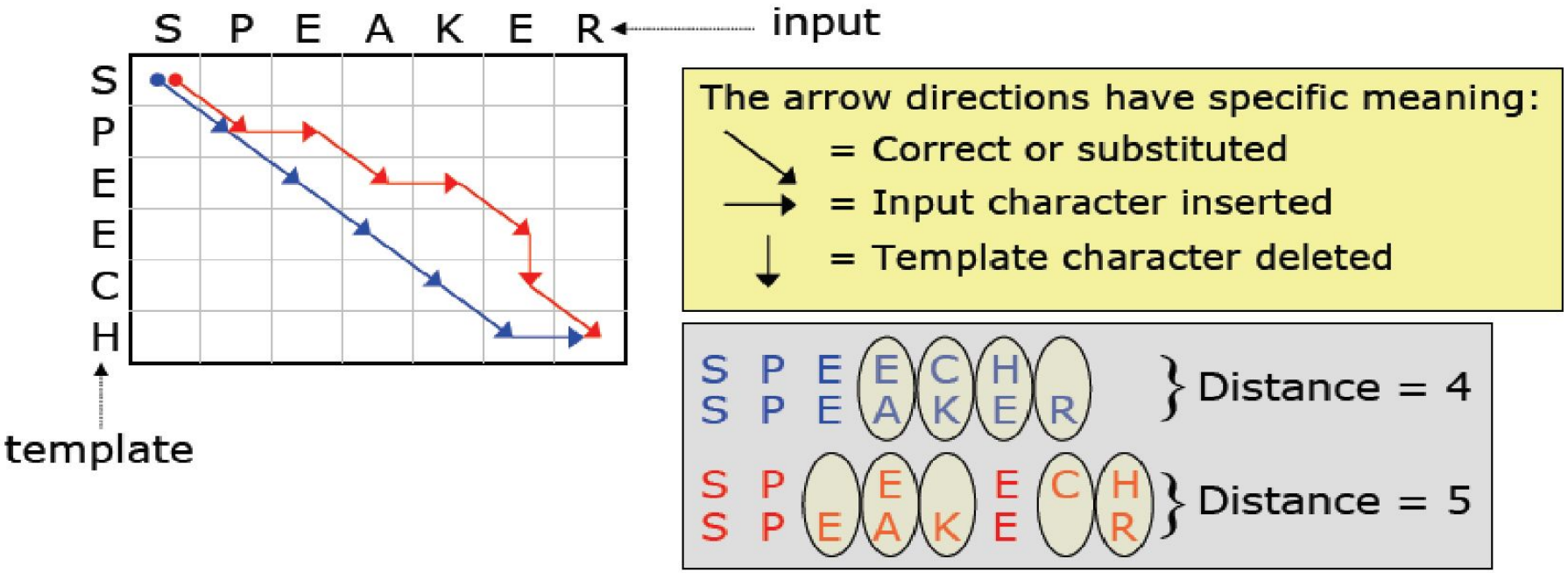


String Matching Problem

- Given two arbitrary strings, find the *minimum edit distance* between the two:
  - Edit distance = the *minimum number of editing operations* needed to convert one into the other
  - Editing operations: substitutions, insertions, deletions
  - Often, the distance is also called *cost*
- This minimum distance is a measure of the *dissimilarity* between the two strings
  - Also called the *Levenshtein distance*

String Edit Distance Computation

- Measuring edit distance is best visualized as a 2-D diagram of the template being *aligned* or *warped* to best match the input
  - Two possible alignments of template to input are shown, in blue and red
- Wanted: *MINIMUM* edit distance: *i.e. inherent* dissimilarity
  - Over *all possible paths* from corner to corner



Minimum String Edit Distance

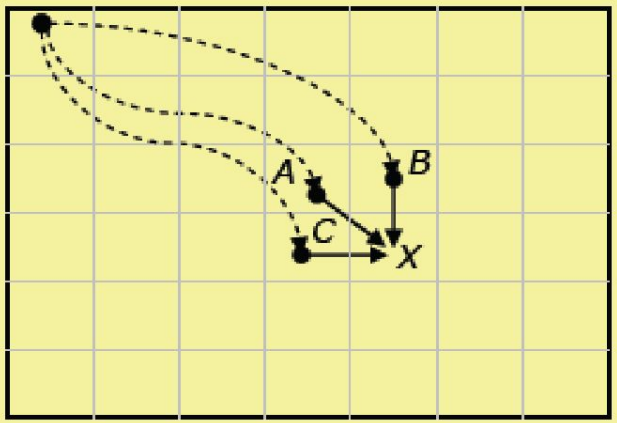
- This is an example of a *search* problem, since we need to search among all possible paths for the best one
- First possibility: Brute force search
  - Exhaustive search through all possible paths from top-left to bottom-right, and choose path with minimum cost
  - But, computationally intractable; exponentially many paths!
- A path is a connected sequence made up of the three types of arrows: diagonal, vertical and horizontal steps
- Solution**:** *Dynamic Programming* (DP)
  - Find optimal (minimum cost) path by utilizing (re-using) optimal sub-paths
  - *Crucial to virtually all major speech recognition systems*

Minimum String Edit Distance: DP

- *Central idea*: formulate optimal path to any intermediate point *X* in the matrix *in terms of optimal paths of all its immediate predecessors*
  - Let M_X= Min. path cost from origin to any pt. *X* in matrix
  - Say, *A*, *B* and *C* are all of *X*'s predecessors
  - Assume M_A, M_B and M_C are known (shown by dotted lines)
- Hence, start from the origin, and compute min. path cost for every matrix entry, proceeding from top-left to bottom right corner
  - Min. edit distance between the two strings = value at bottom right corner

Then, $M_X = \min (M_A+AX, M_B+BX, M_C+CX)$

- AX = edit distance for diagonal transition = 0 if the aligned letters are same, 1 if not)
- BX = edit distance for vertical transition = 1 (deletion)
- CX = edit distance for horizontal transition = 1 (insertion)

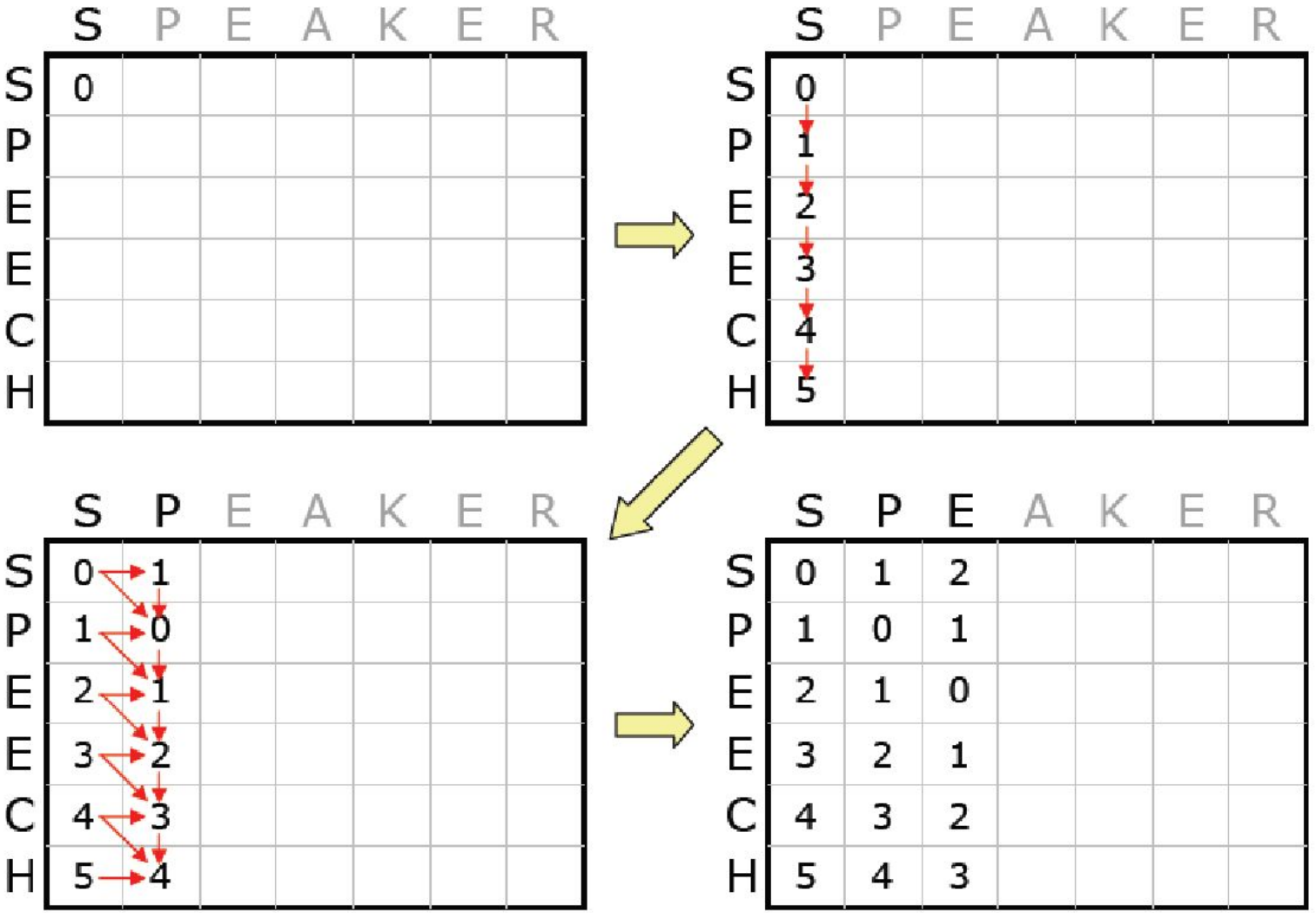


Minimum String Edit Distance: DP

- Although the matrix entries can be filled in any order (for any entry, we only need to ensure that all its predecessors are evaluated), it helps to proceed methodically, once column (*i.e.* one input character) at a time:
  - Consider each input character, one at a time
  - Fill out min. edit distance for that entire column before moving on to next input character
  - Forces us to examine every unit of input (in this case, every character) one at a time
    - Allows each input character to be processed *as it becomes available*

DP Example

First, initialize top left corner, aligning the first letters



DP Example

		S	P	E	A	K	E	R
	S	0	1	2	3			
	P	1	0	1	2			
	E	2	1	0	1			
	E	3	2	1	1			
	C	4	3	2	2			
	H	5	4	3	3			

		S	P	E	A	K	E	R
	S	0	1	2	3	4		
	P	1	0	1	2	3		
	E	2	1	0	1	2		
	E	3	2	1	1	2		
	C	4	3	2	2	2		
	H	5	4	3	3	3		

		S	P	E	A	K	E	R
	S	0	1	2	3	4	5	
	P	1	0	1	2	3	4	
	E	2	1	0	1	2	3	
	E	3	2	1	1	2	2	
	C	4	3	2	2	2	3	
	H	5	4	3	3	3	3	

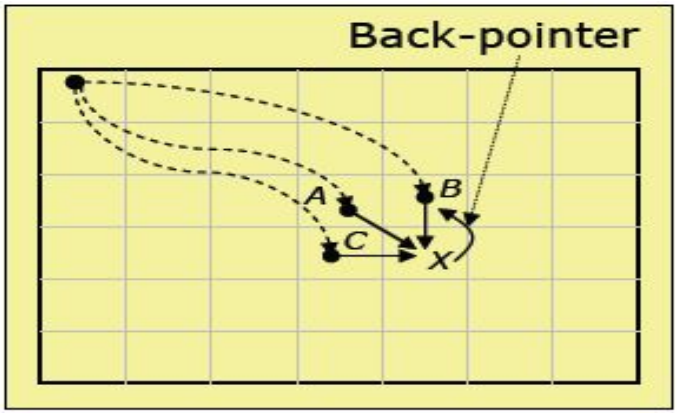
		S	P	E	A	K	E	R
	S	0	1	2	3	4	5	6
	P	1	0	1	2	3	4	5
	E	2	1	0	1	2	3	4
	E	3	2	1	1	2	2	3
	C	4	3	2	2	2	3	3
	H	5	4	3	3	3	3	4

Min. edit distance (SPEECH, SPEAKER) = 4

- One possible min. distance alignment is shown in blue

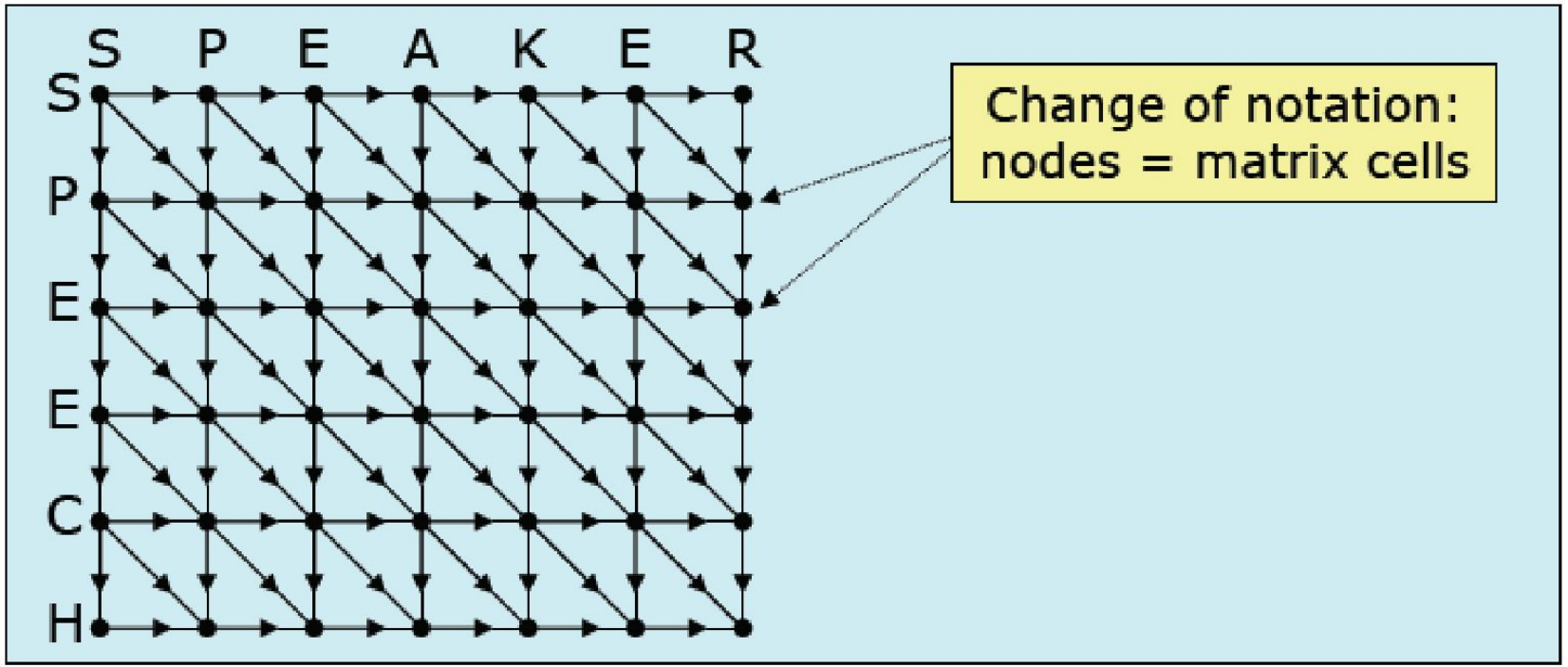
DP: Finding the Best Alignment

- The algorithm so far only finds the *cost*, not the alignment itself
- How do we find the actual path that minimizes edit distance?
  - There may be multiple such paths, any one path will suffice
- To determine the alignment, we modify the algorithm as follows
- Whenever a cell *X* is filled in, we maintain a *back-pointer* from *X* to its predecessor cell that led to the best score for *X*
  - Recall M_X= min (M_A+*AX*, M_B+*BX*, M_C+*CX*)
  - So, if M_B+*BX* happens to be the minimum we create a back-pointer *X->B*
  - If there are ties, break them arbitrarily
- Thus, every cell has a single back-pointer
- At the end, we *trace back* from the final cell to the origin, using the back-pointers, to obtain the best alignment



Finding the Best Alignment: DP Trellis

- The 2-D matrix, with all possible transitions filled in, is called the *search trellis*
- DP does *not* require that transitions be limited to the three types used in the example
- The primary requirement is that the optimal path be computable recursively, based on a node's predecessors optimal sub-paths



Search Trellis

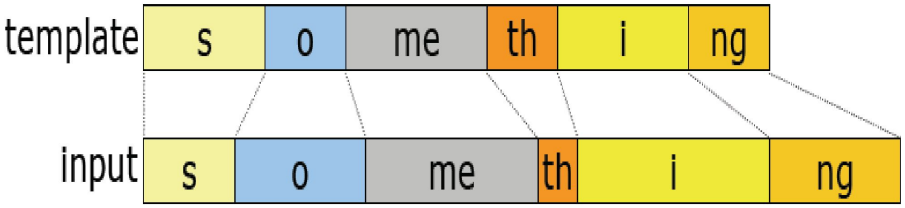
- *The search trellis is arguably one of the most crucial concepts in modern day decoders*!
- Just about any decoding problem is usually cast in terms of such a trellis
- It is then a matter of searching through the trellis for the best path
- The DP algorithm, at every node (cells of the DP matrix), maintains a *path cost* for the *best path* from the origin to that node
  - In string matching, this cost is the *substring* minimum edit distance
  - Path costs are computed by accumulating local node and edge costs (allowed transitions) according to the recursive formulation already seen (minimizing cost)
- One may also use a *similarity* measure, instead of *dissimilarity*
  - In this case DP algorithm should try to *maximize* the total path score

Applications of DP string matching

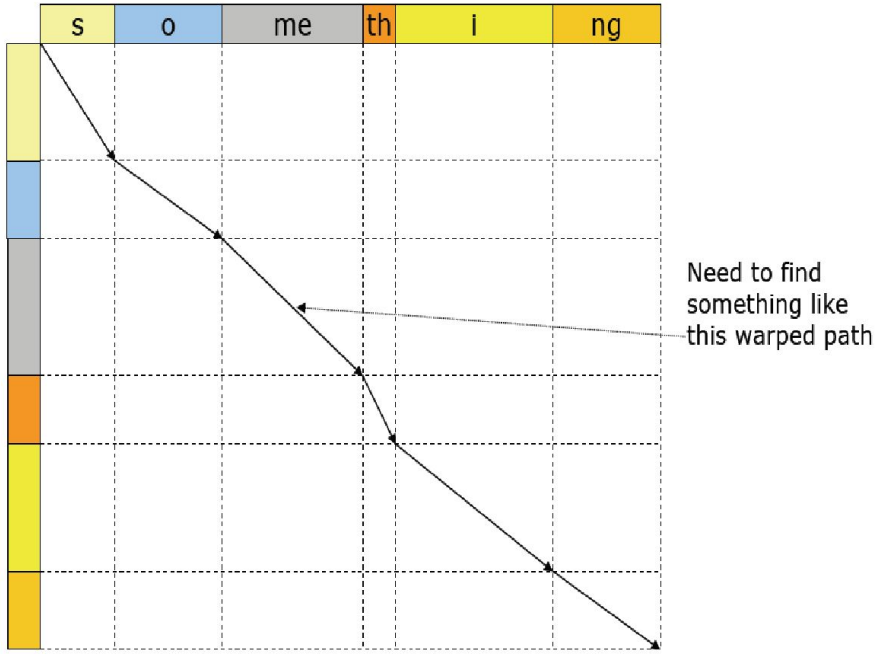
- How might google recognize “RaviShanker” as a mistyped version of “RaviShankar”?
- One hypothetical heuristic:
  - Google maintains a list of *highly popular* query strings
    - These are the templates; “RaviShankar” is one of them
- When a user types in a query, the string-matched to every template, using DP
- If a template matches exactly, there is no spelling error
- If a *single* template has one overall error (edit distance = 1), google can ask *Did you mean “…”*
- Otherwise, do nothing
  - Either multiple templates have edit distance = 1, or
  - Minimum edit distance > 1
- Here, we see the implicit introduction of a *confidence measure*
  - A decision based on the *value* of the min. edit distance

DTW: DP for Template Matching

- Template matching for speech: *dynamic time warping*
  - Inputs and templates are sequences of feature vectors instead of letters



- Intuitive understanding of why DP-like algorithm might work to find a best alignment of a template to the input:
  - We need to search for a path that finds the shown alignment:



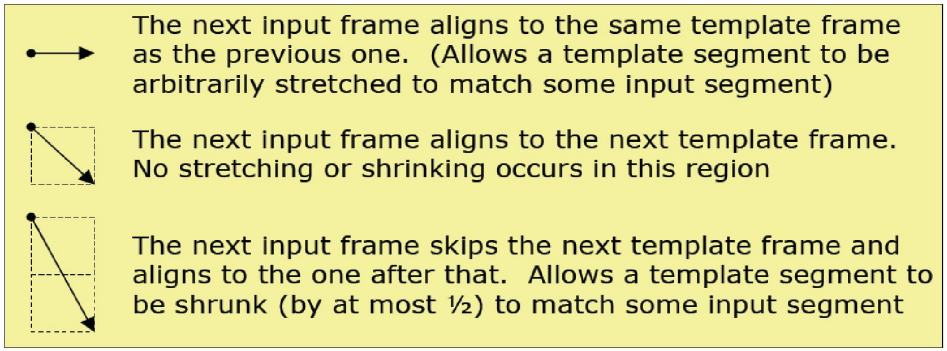
DTW: Adapting Concepts from DP

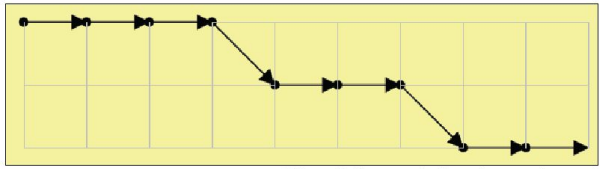
- Some concepts from string matching need to be adapted to this problem
  - What are the allowed set of transitions in the search trellis?
  - What are the edge and local node costs?

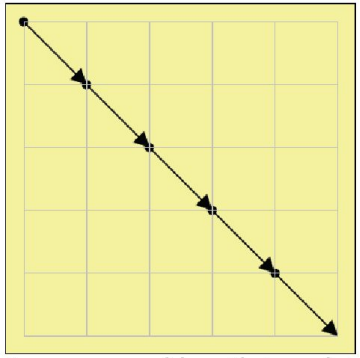
- Once these questions are answered, we can apply essentially the same DP algorithm to find a minimum cost match (path) through the search trellis

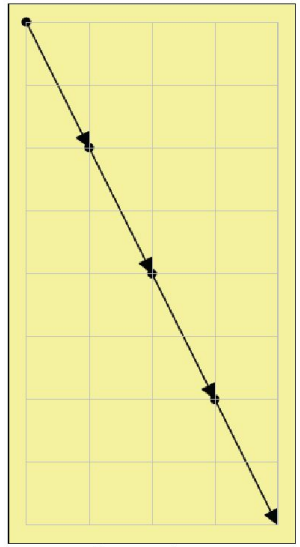
DTW: Determining Transitions

- The main modes of distortion are *stretching* and *shrinking* of speech segments, owing to different speaking rates
  - Of course, we do not know *a priori* what the distortion looks like
- Also, since speech signals are continuous valued instead of discrete, there is really no notion of insertion
  - Every input frame must be matched to *some* template frame
- For meaningful comparison of two different path costs, their lengths must be kept the same
  - So, every input frame is to be aligned to a template frame *exactly* once
- Typical transitions used in DTW for speech:




Short template, long input


Approx. equal length template, input

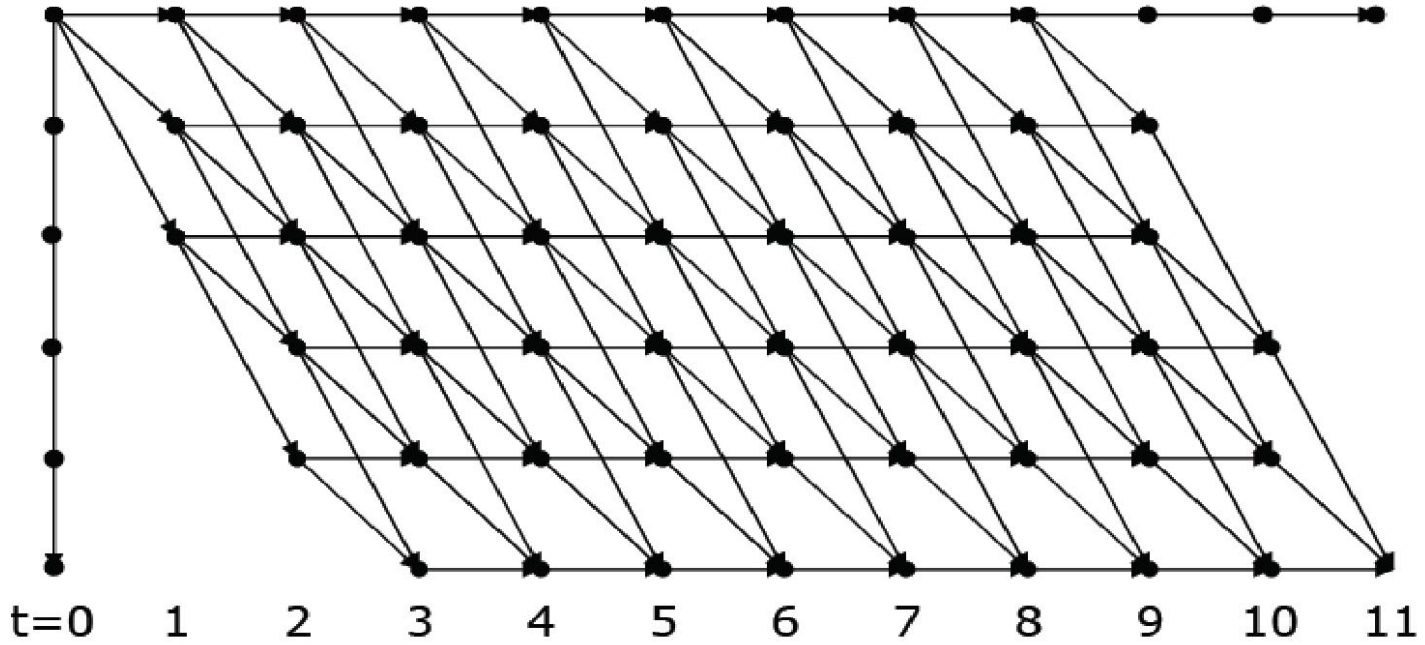

Long template, short input

DTW: Local Edge and Node Costs

- Typically, there are no edge costs; any edge can be taken with no cost
- Local node costs measure the dissimilarity or distance between the respective input and template frames
- Since the frame content is a multi-dimensional feature-vector, what dissimilarity measure can we use?
- A simple measure is *Euclidean distance*; *i.e.* geometrically how far one point is from the other in the multi-dimensional vector space
  - For two vectors X = (x1, x2, x3…xN), and Y = (y1, y2, y3…yN), the Euclidean distance between them is:
    - $\sqrt{\Sigma(xi-yi)^2}$, i= 1 .. N
  - Thus, if X and Y are the same point, the Euclidean distance = 0
  - The farther apart X and Y are, the greater the distance
- Other distance measure could also be used:
  - Manhattan metric or the L1 norm: $\Sigma|Ai–Bi|$
  - Weighted Minkowski norms: $(\Sigma wi|Ai–Bi|^n)^{1/n}$

DTW: Algorithm

- The transition structure and local edge and node costs are now defined
- The search trellis can be realized and the DP algorithm applied to search for the minimum cost path, as before
  - Example trellis using the transition types shown earlier:



DTW: Algorithm

- Let *Pi,j*= the best path cost from origin to node [*i,j*] (i.e., where *i*-th template frame aligns with *j*-th input frame)

- Let *Ci,j*= the local node cost of aligning template frame *i* to input frame *j* (Euclidean distance between the two vectors)

- Then, by the DP formulation:

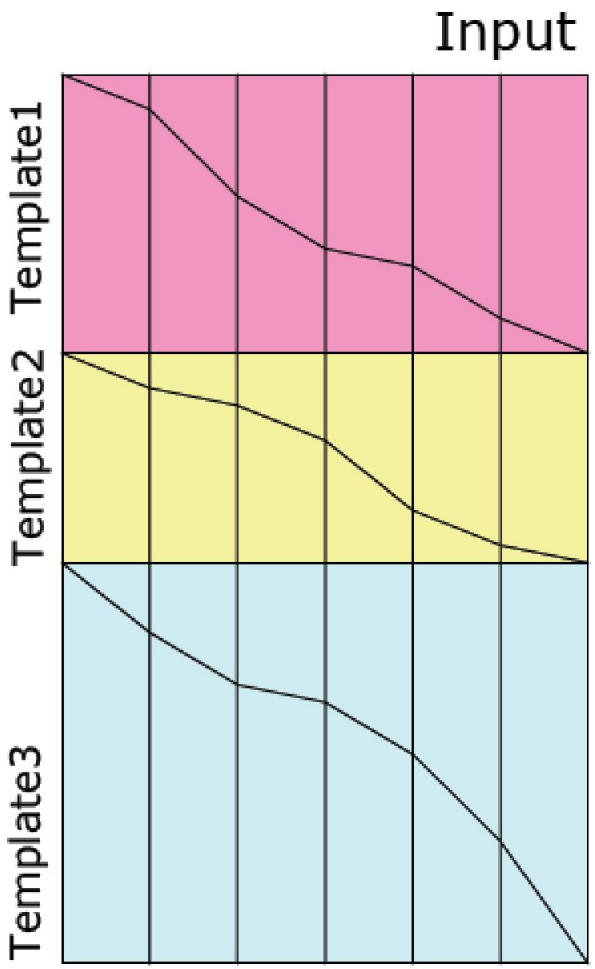
$$P_{i,j} = \min\left(P_{i,j-1} + C_{i,j},\ P_{i-1,j-1} + C_{i,j},\ P_{i-2,j-1} + C_{i,j}\right)$$
$$= \min\left(P_{i,j-1},\ P_{i-1,j-1},\ P_{i-2,j-1}\right) + C_{i,j}$$

IWR using DTW

- We now have a method for measuring the best match of a template to the input speech
- How can we apply this to perform isolated word recognition?
  - For each word in the vocabulary, pre-record a spoken example (its template)
  - For a given input utterance, measure its minimum distance to each template using DTW
  - Choose the template that delivers the smallest distance
- As easy as that!
  - Could implement this on a cell phone for dialing

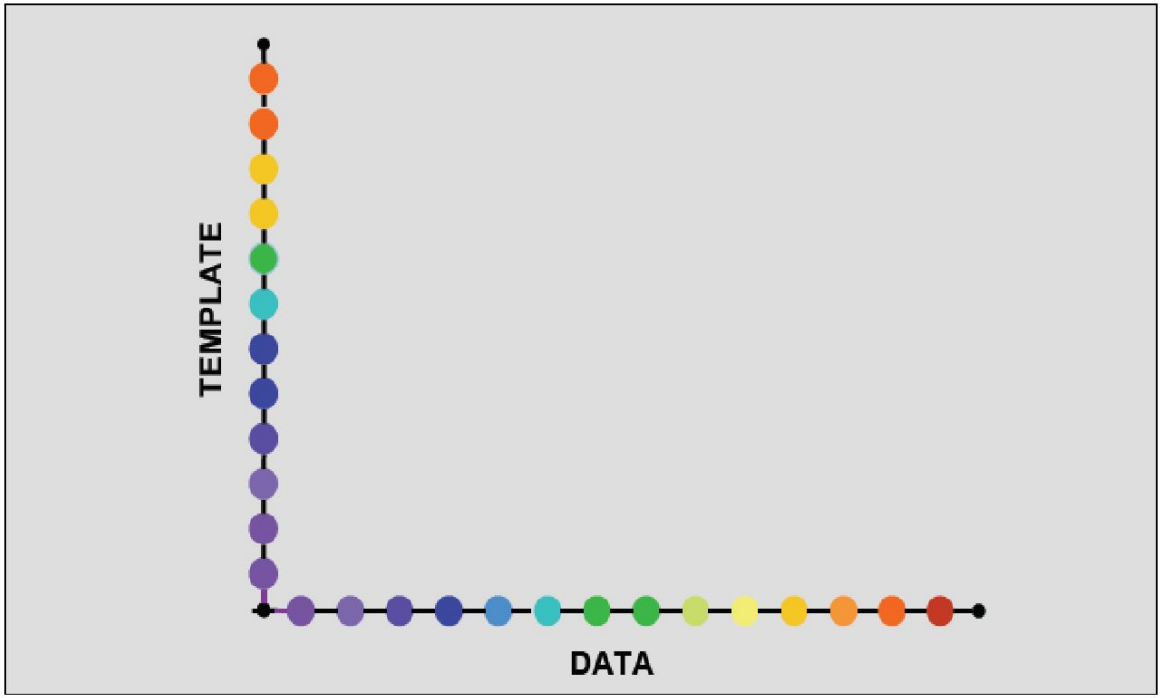
Time Synchronous Search

- Since input frames are processed sequentially, the input speech can be matched to all templates simultaneously
- The figure shows three such matches going on in parallel
- Essentially, every template match is started simultaneously and stepped through the input in lock-step fashion
  - Hence the term *time synchronous*
- Advantages
  - **No need to store the entire input for matching with successive templates**
  - **Matching can proceed as the input comes in**
  - **Enables *pruning* for computational efficiency**
  - **Other advantages in continuous speech recognition**



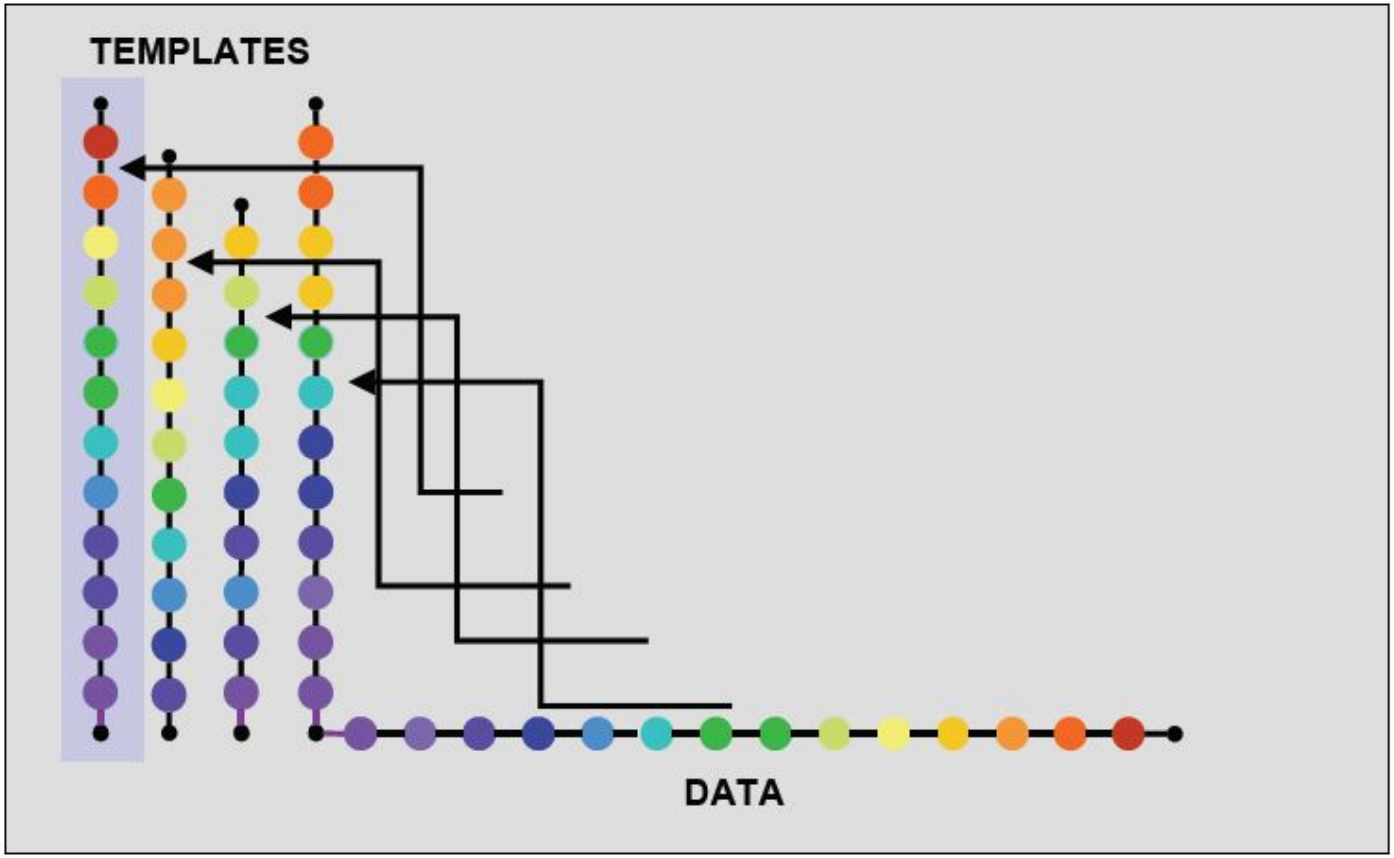
DTW using Single Template: Limitations

- As noted in the previous topic, a single template cannot capture all the variations in speech
- One alternative already suggested: use multiple templates for each word, and match the input against each one



DTW with Multiple Templates

- Each template warps differently to best match the input; the best matching template is selected



Problem with Multiple Templates

- Finding the best match requires the evaluation of many more templates (depending on the number)
  - This can be computationally expensive
    - Important for handheld devices, even for small-vocabulary applications
    - Think of battery life!
  - Need a method for reducing multiple templates into a single one

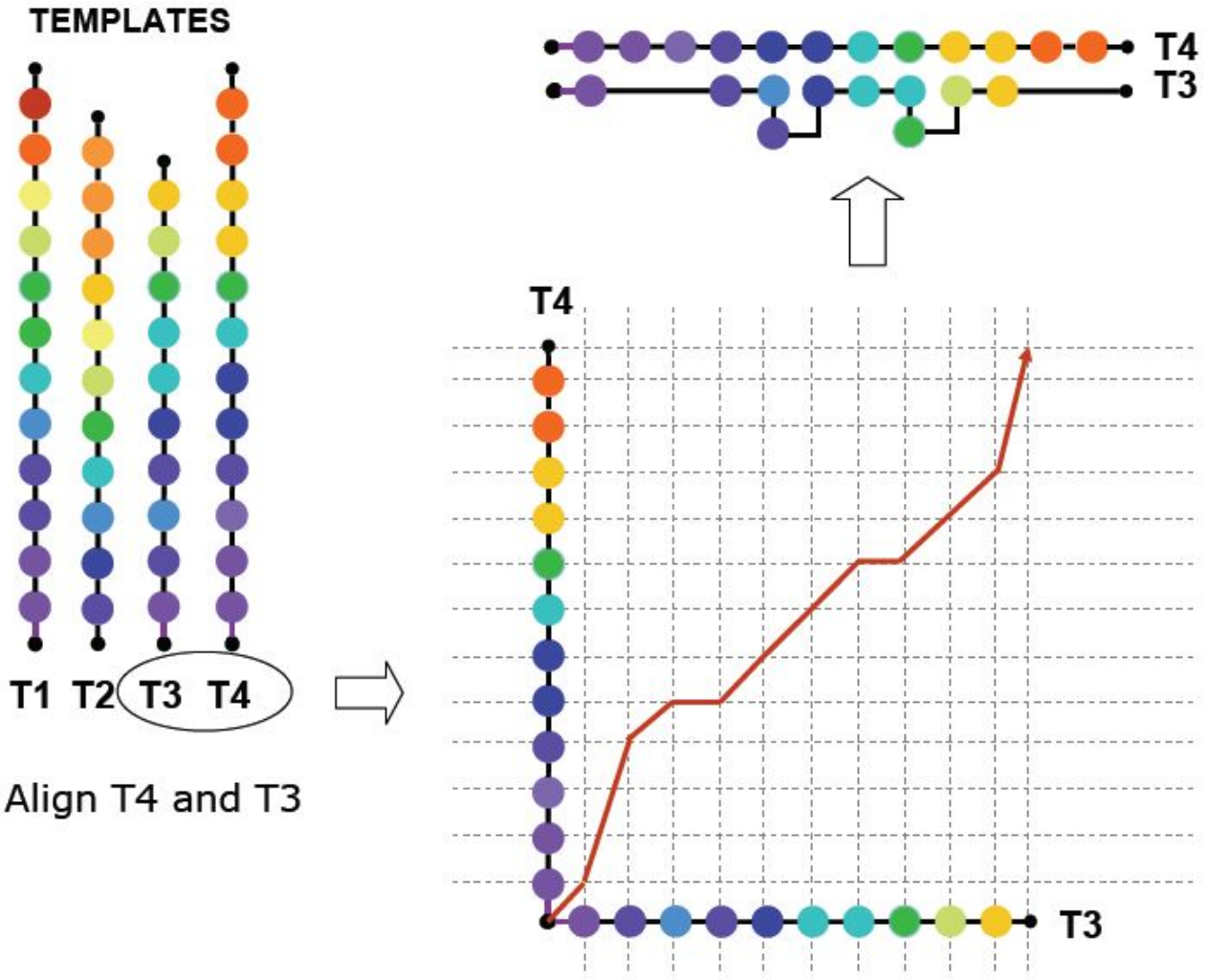
- Even multiple templates do not cover the *space* of possible variations
  - Need mechanism of generalizing from the templates to include data not seen before

- We can achieve both objectives by *averaging* all the templates for a given word

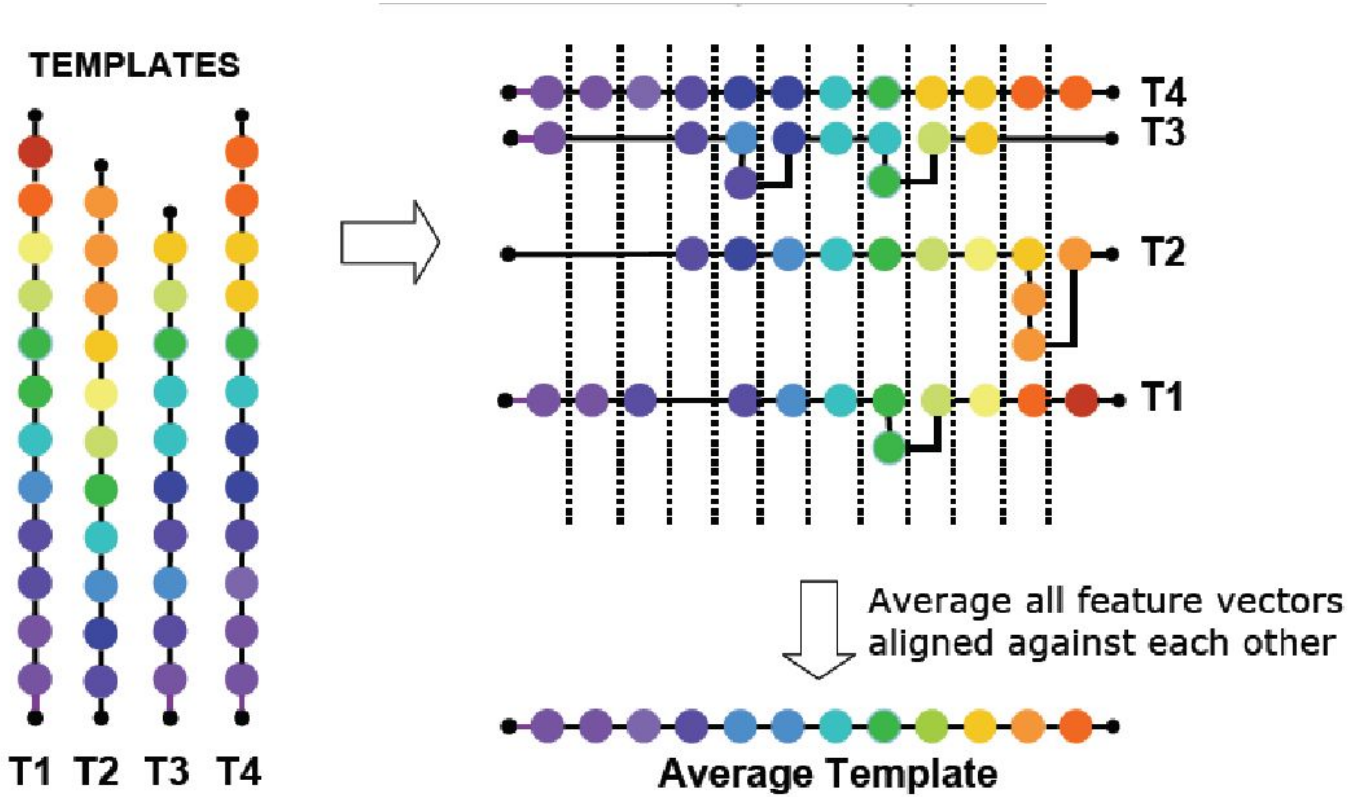
Template Averaging

- How can we average the templates when they're of different lengths?
  - Somehow need to normalize them to each other
- *Solution*: Apply DTW
  - Pick one template as a "master"
  - Align all other templates to it
    - *Note*: This requires not just finding the best cost, but the actual alignment between the template and input frame sequences, using the *back-pointers* described earlier
  - Use the alignments so generated to compute their average
- *Note*: Choosing a different *master* template will lead to a different average template
  - Which template to choose as the master?
    - No definitive answer exists
    - Only trial and error solutions exist

DTW with Multiple Templates



DTW with Multiple Templates



Align T4/T2 and T4/T1, similarly; then average all of them

Benefits of Template Averaging

- Obviously, we have eliminated the computational cost of having multiple templates for each word

- Using the averages of the aligned feature vectors *generalizes* from the samples
  - The average is representative of the templates, and more generally, assumed to be representative of future utterances of the word

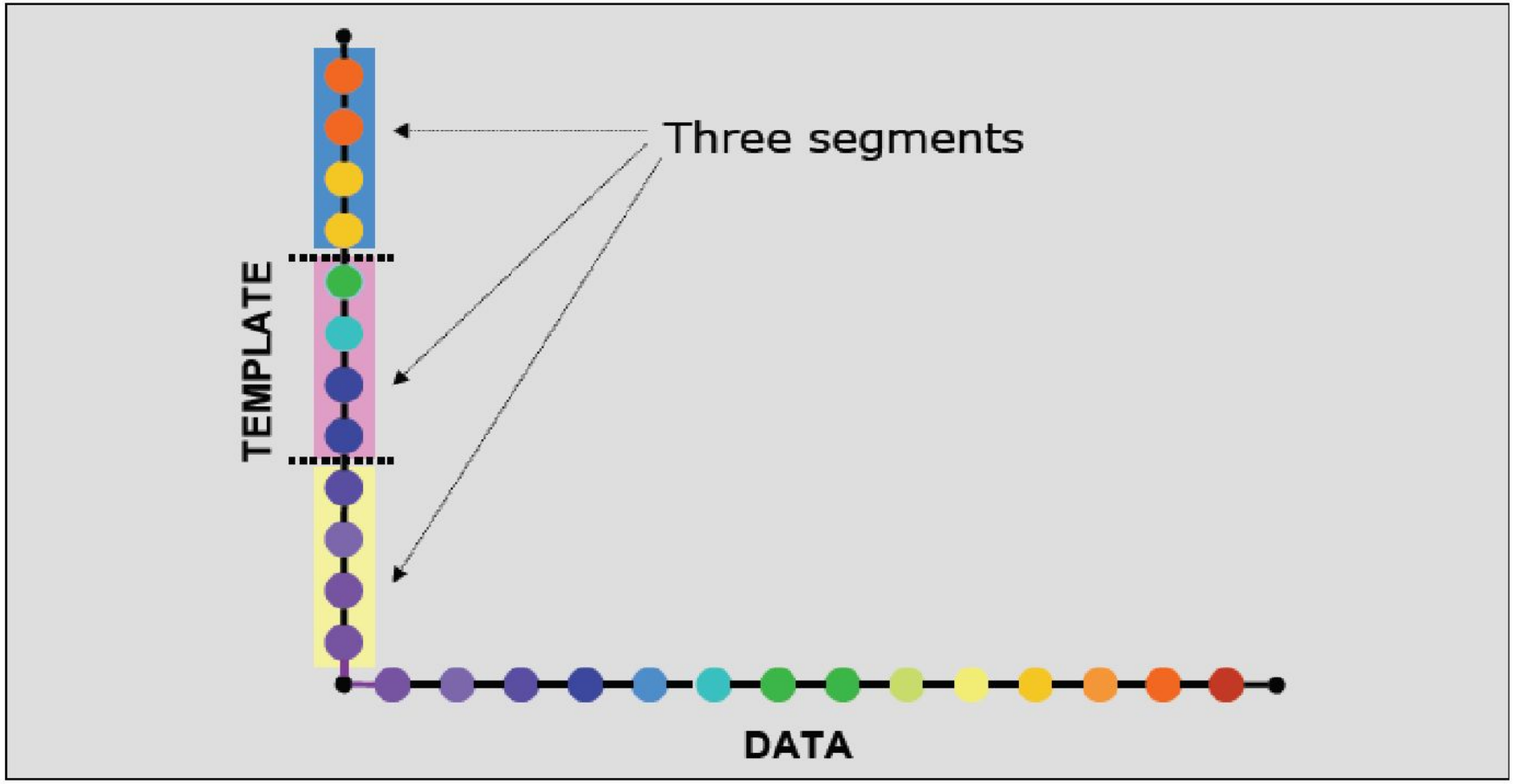
- The more the number of templates, the better the generalization

Template Size Reduction

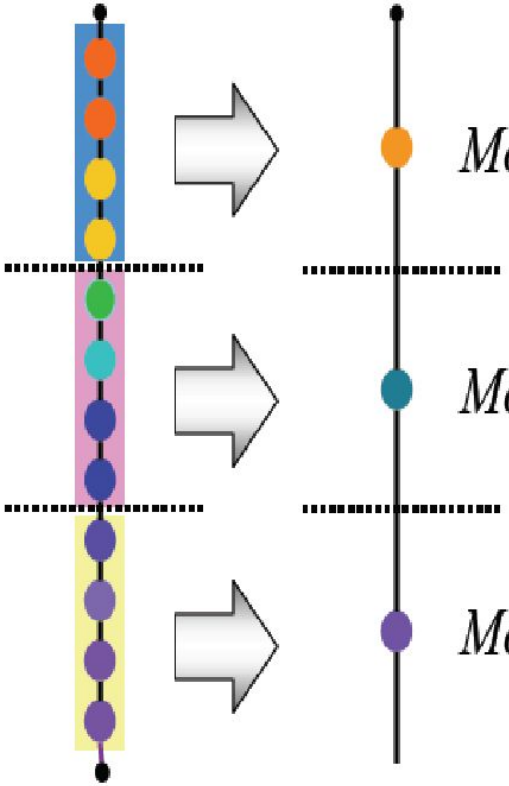
- Can we do better? Consider the template for “something”:
- Here, the template has been manually *segmented* into 6 segments, where each segment is a single phoneme
- Hence, the frames of speech that make up any single segment ought to be fairly alike
- If so, why not replace each segment by a *single* representative feature vector?
  - How? Again by averaging the frames within the segment
- This gives a reduction in the template *size* (memory size)

Single Template with Three Segments

- The feature vectors within each segment are assumed to be similar to each other.



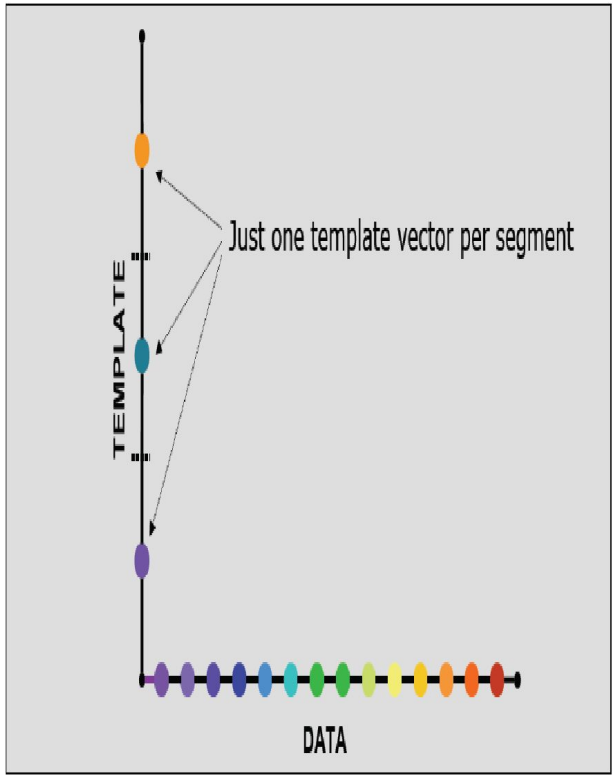
Averaging Each Template Segment



$$Model\,Vector = \frac{1}{N} \sum_{i \in segment} vector(i)$$

$$Model\,Vector = \frac{1}{N} \sum_{i \in segment} vector(i)$$

$$Model\,Vector = \frac{1}{N} \sum_{i \in segment} vector(i)$$



$$m_j = \frac{1}{N_j} \sum_{i \in segment(j)} x(i)$$

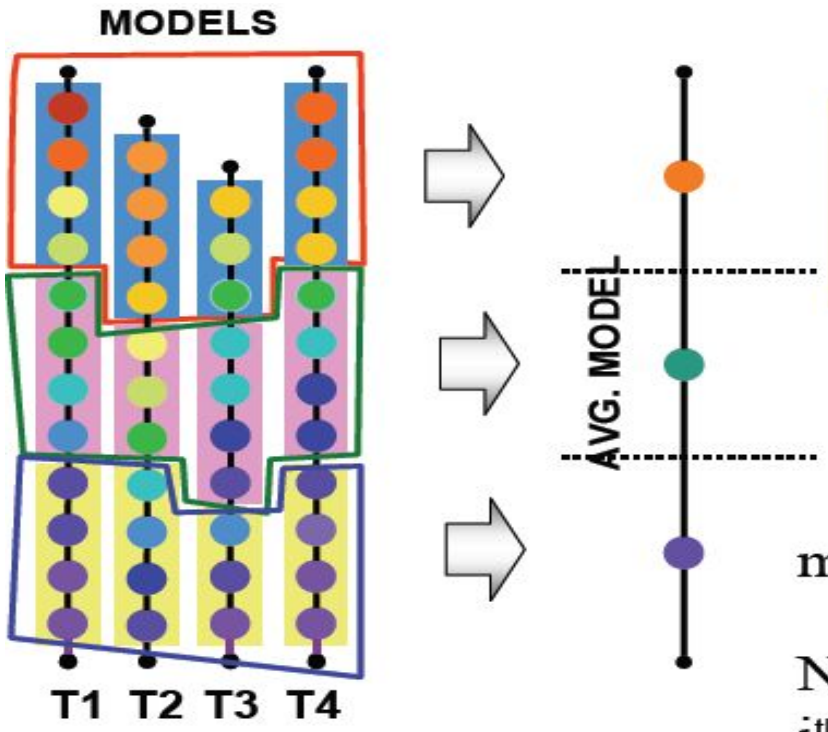
m_j is the model vector for the j^{th} segment

N_j is the number of vectors in the j^{th} segment

$x(i)$ is the i^{th} feature vector

DTW with Multiple Models

- Segment all templates
- Average each region into a single point



$$m_j = \frac{1}{\sum_k N_{k,j}} \sum_{i \in segment_k(j)} x_k(i)$$

$segment_k(j)$ is the j^{th} segment of the k^{th} training sequence

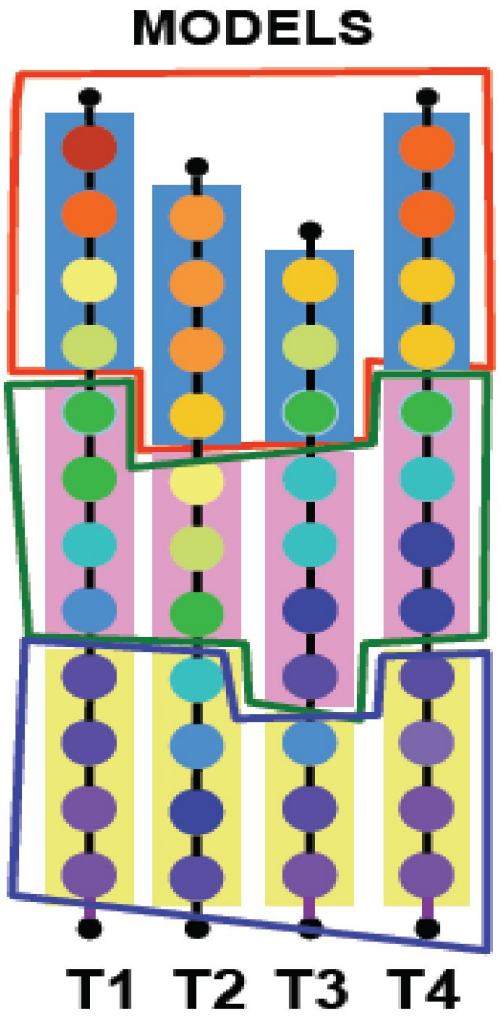
m_j is the model vector for the j^{th} segment

$N_{k,j}$ is the number of training vectors in the j^{th} segment of the k^{th} training sequence

$x_k(i)$ is the i^{th} vector of the k^{th} training sequence

DTW with Multiple Models

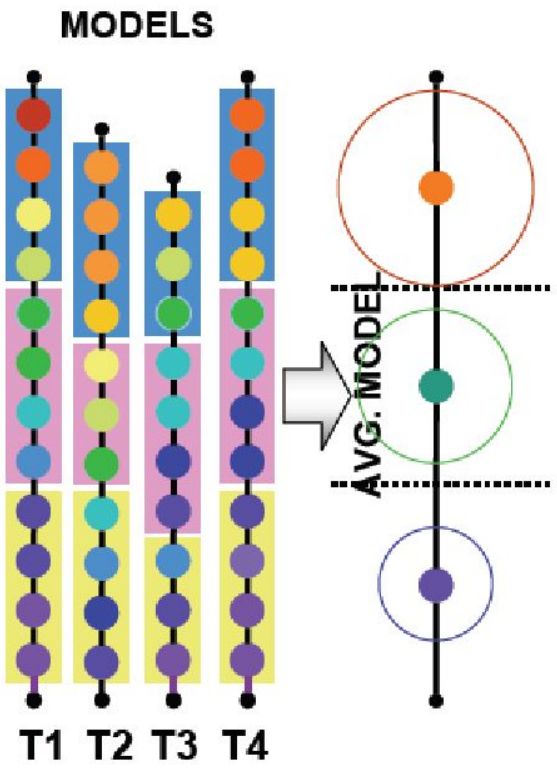
- The inherent variation between vectors is different for the different segments
  - E.g. the variation in the colors of the beads in the top segment is greater than that in the bottom segment
- Ideally we should account for the differences in variation in the segments
  - E.g. a vector in a test sequence may actually be more matched to the central segment, which permits greater variation, although it is closer, in a Euclidean sense, to the mean of the lower segment, which permits lesser variation



Segmental K-means

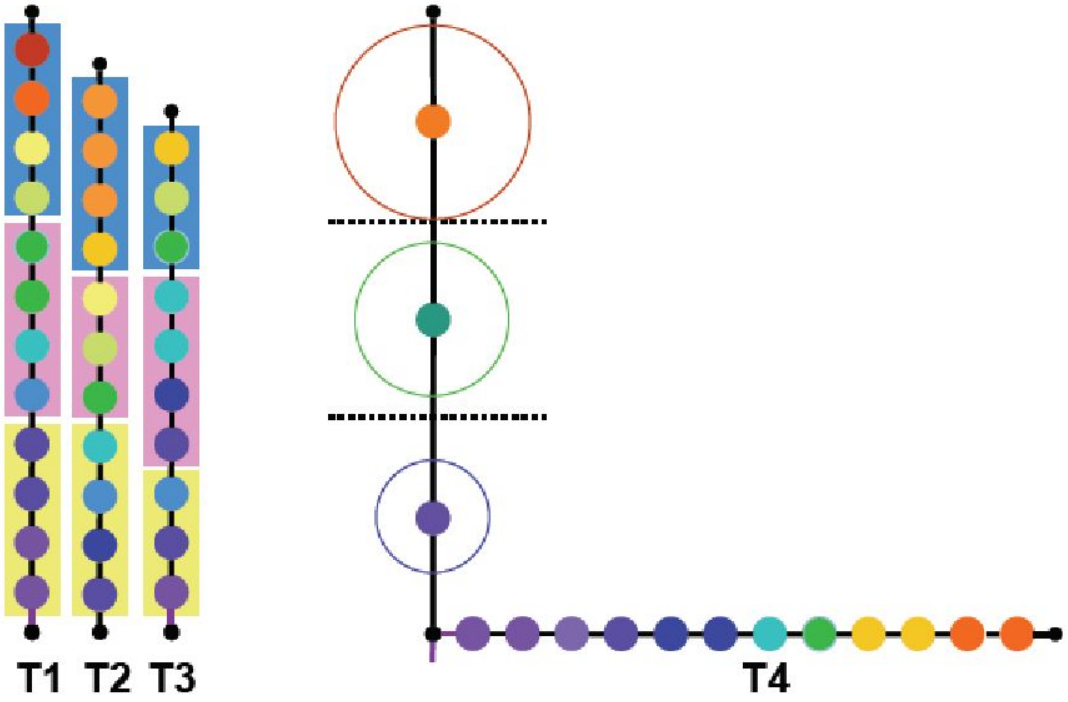
- Simple uniform segmentation of training instances is not the most effective method of grouping vectors in the training sequences
- A better segmentation strategy is to segment the training sequences such that the vectors within any segment are most alike
  - The total distance of vectors within each segment from the model vector for that segment is minimum
  - For a global optimum, the total distance of all vectors from the model for their respective segments must be minimum
- This segmentation must be estimated
- The segmental K-means procedure is an iterative procedure to estimate the optimal segmentation

Training with Multiple Vector Sequences



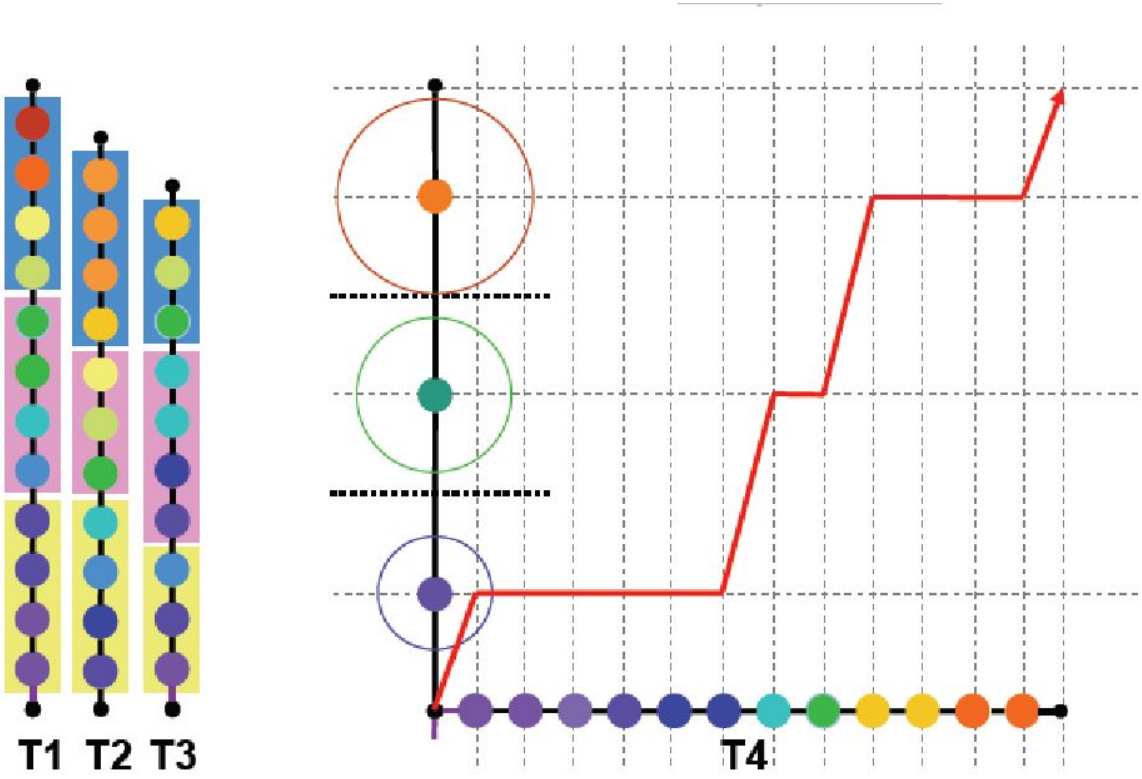
Initialize by uniform segmentation

Training with Multiple Vector Sequences



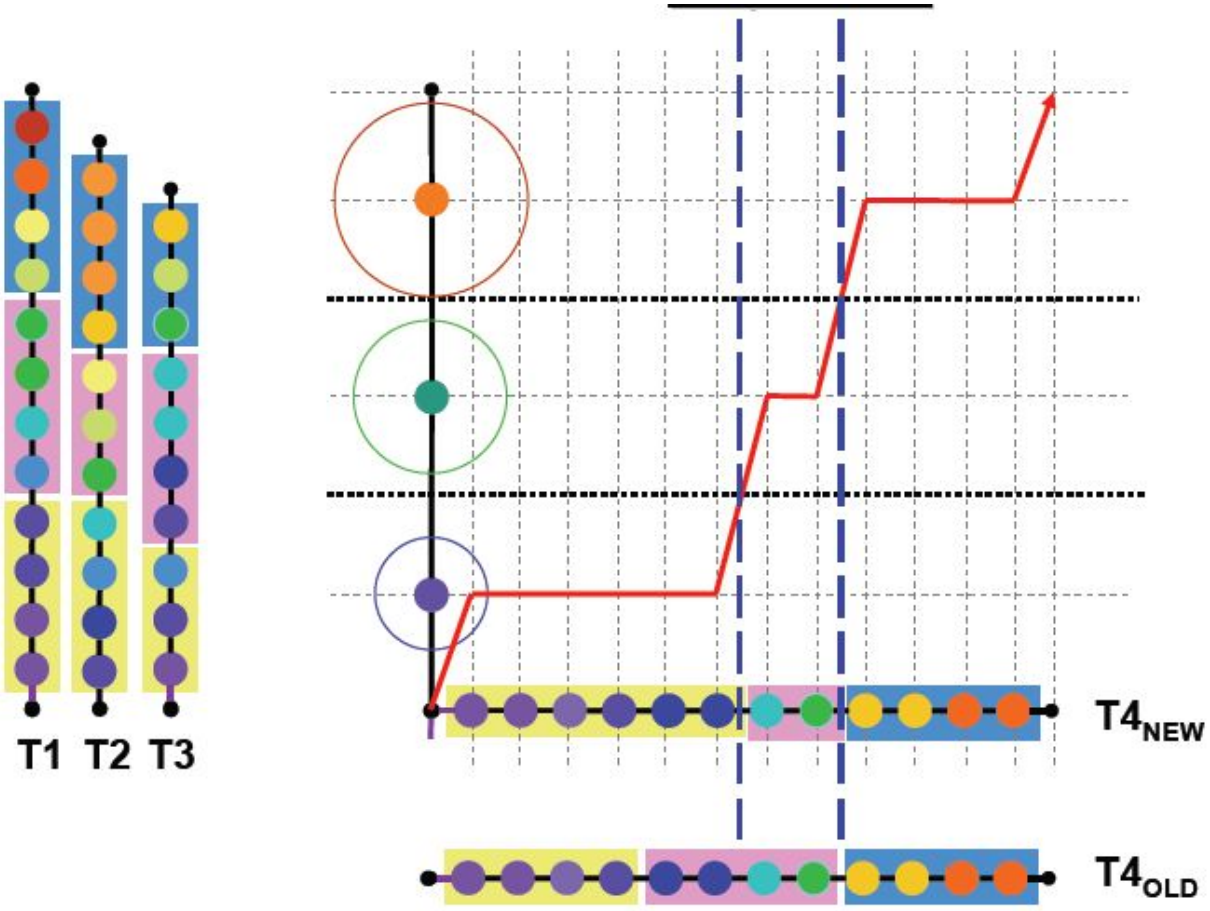
Initialize by uniform segmentation

Training with Multiple Vector Sequences

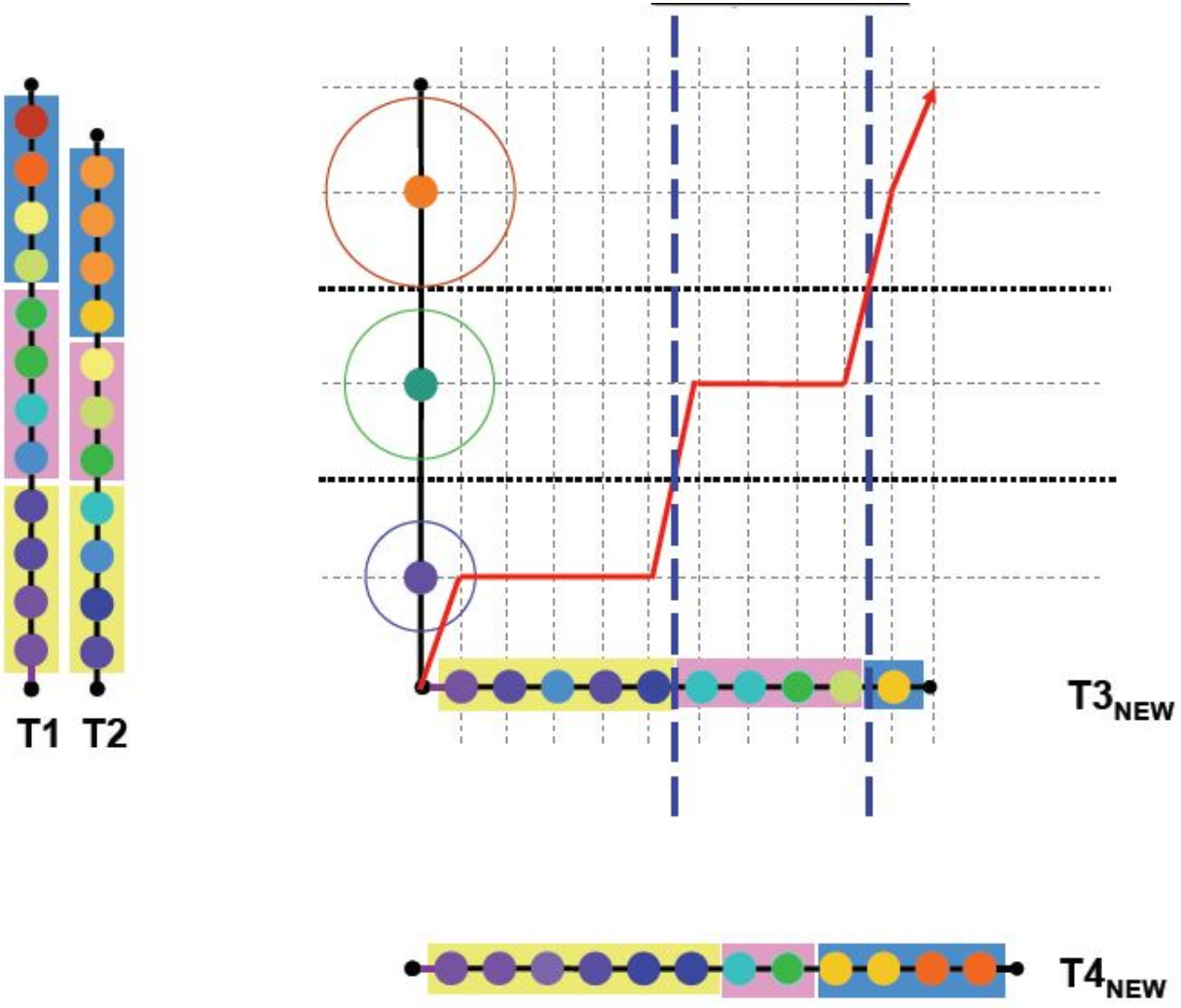


Initialize by uniform segmentation
Align each template to the averaged model to get new segmentations

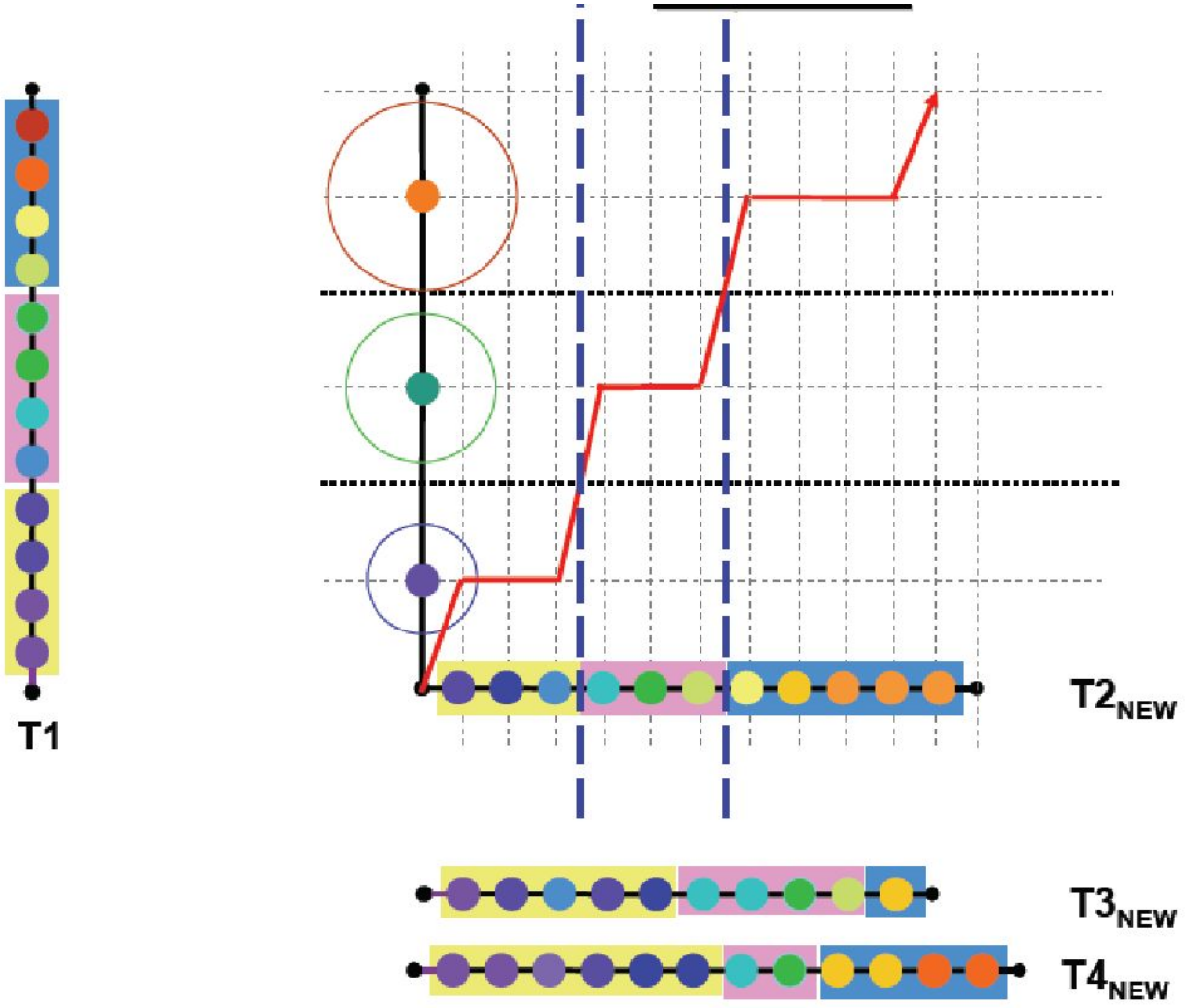
Training with Multiple Vector Sequences



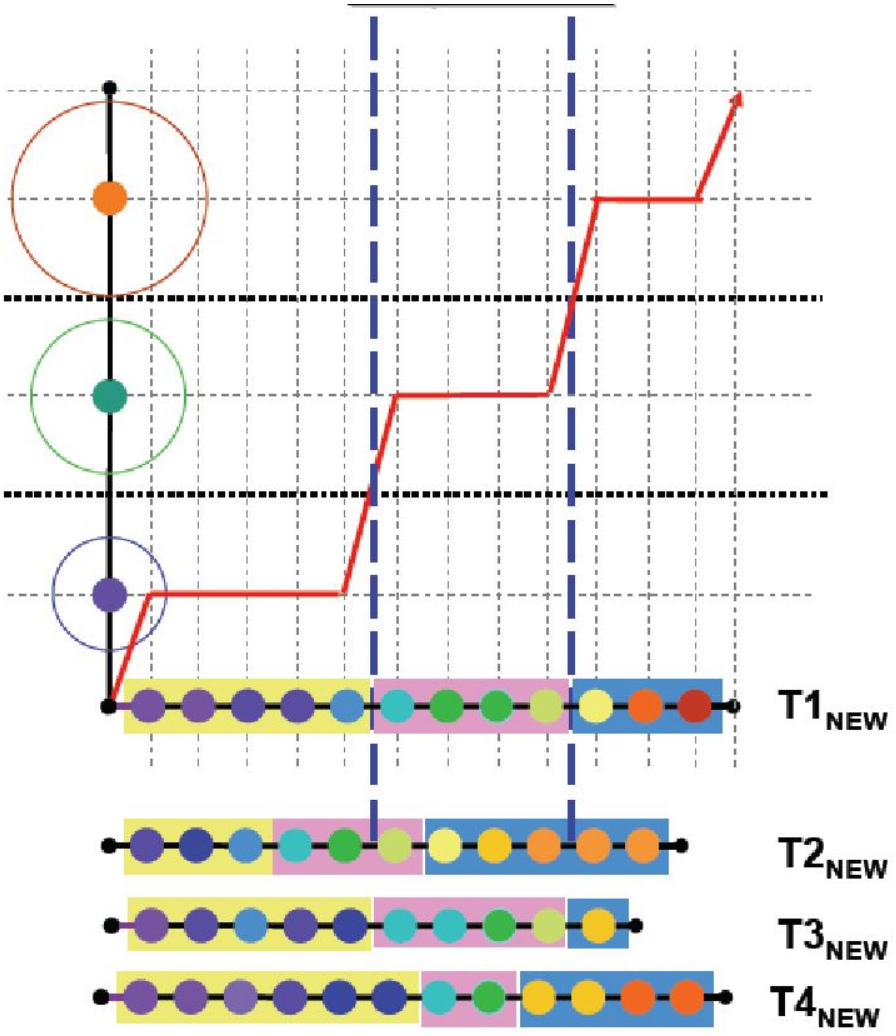
Training with Multiple Vector Sequences



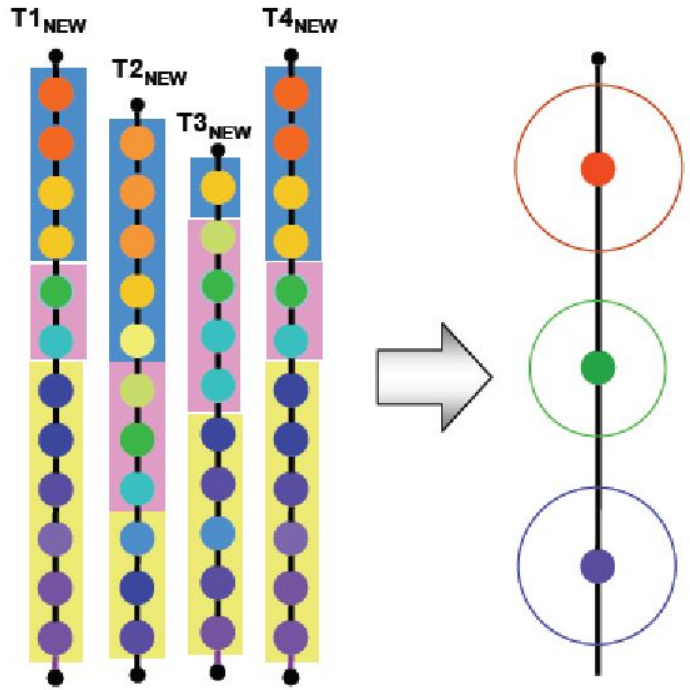
Training with Multiple Vector Sequences



Training with Multiple Vector Sequences

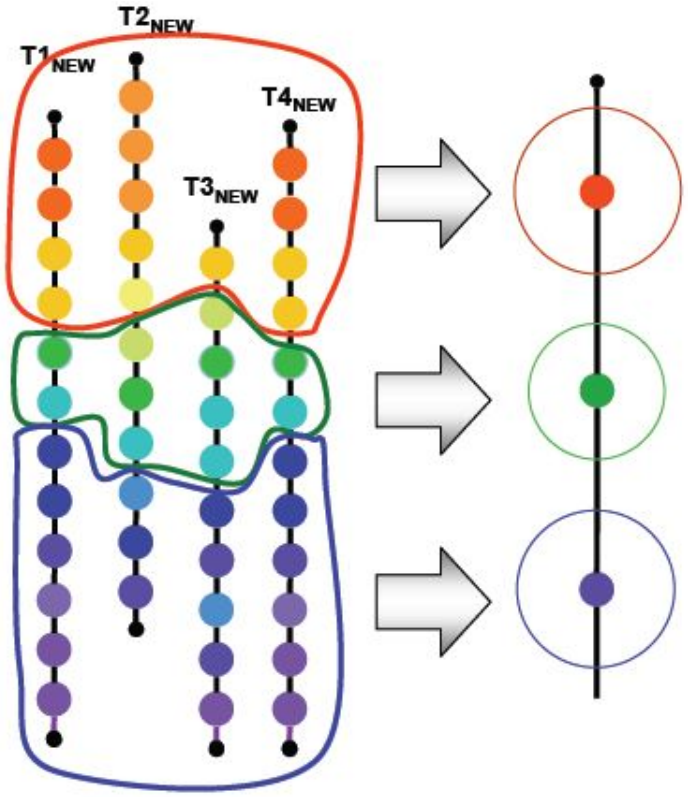


Training with Multiple Vector Sequences

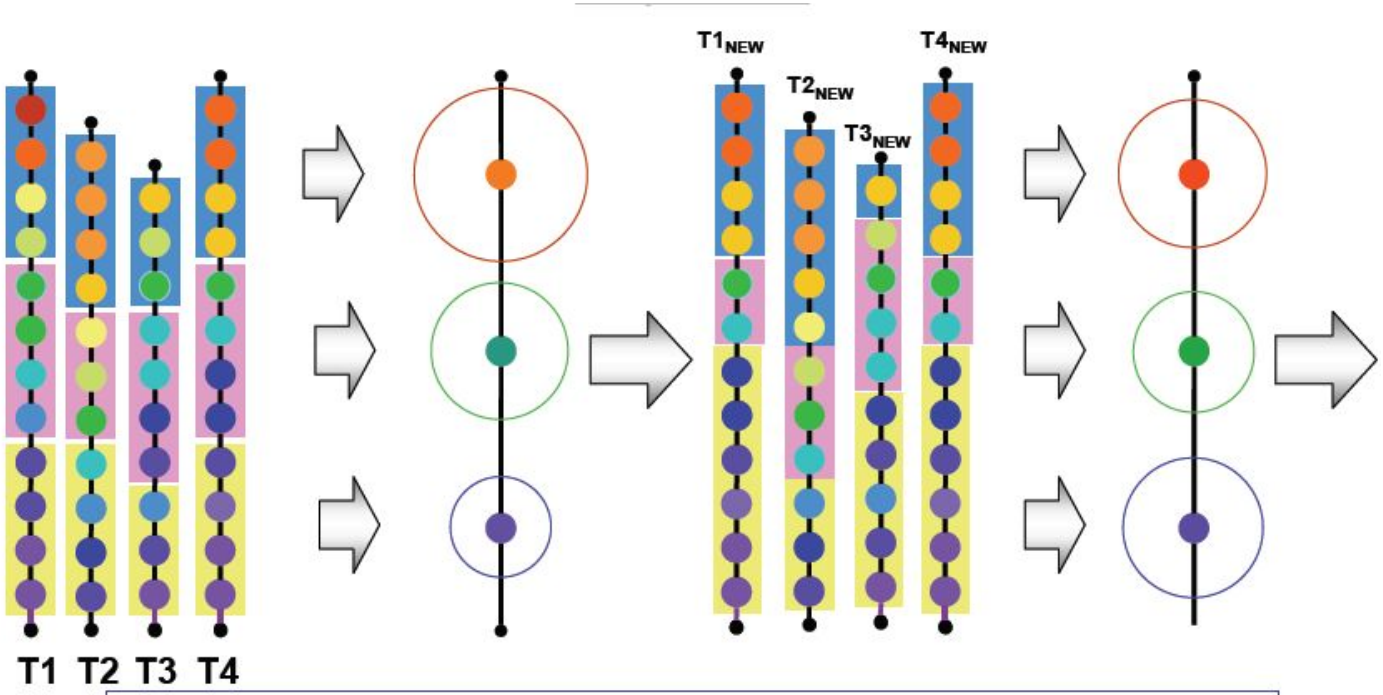


Initialize by uniform segmentation
Align each template to the averaged model to get new segmentations
Recompute the average model from new segmentations

Training with Multiple Vector Sequences



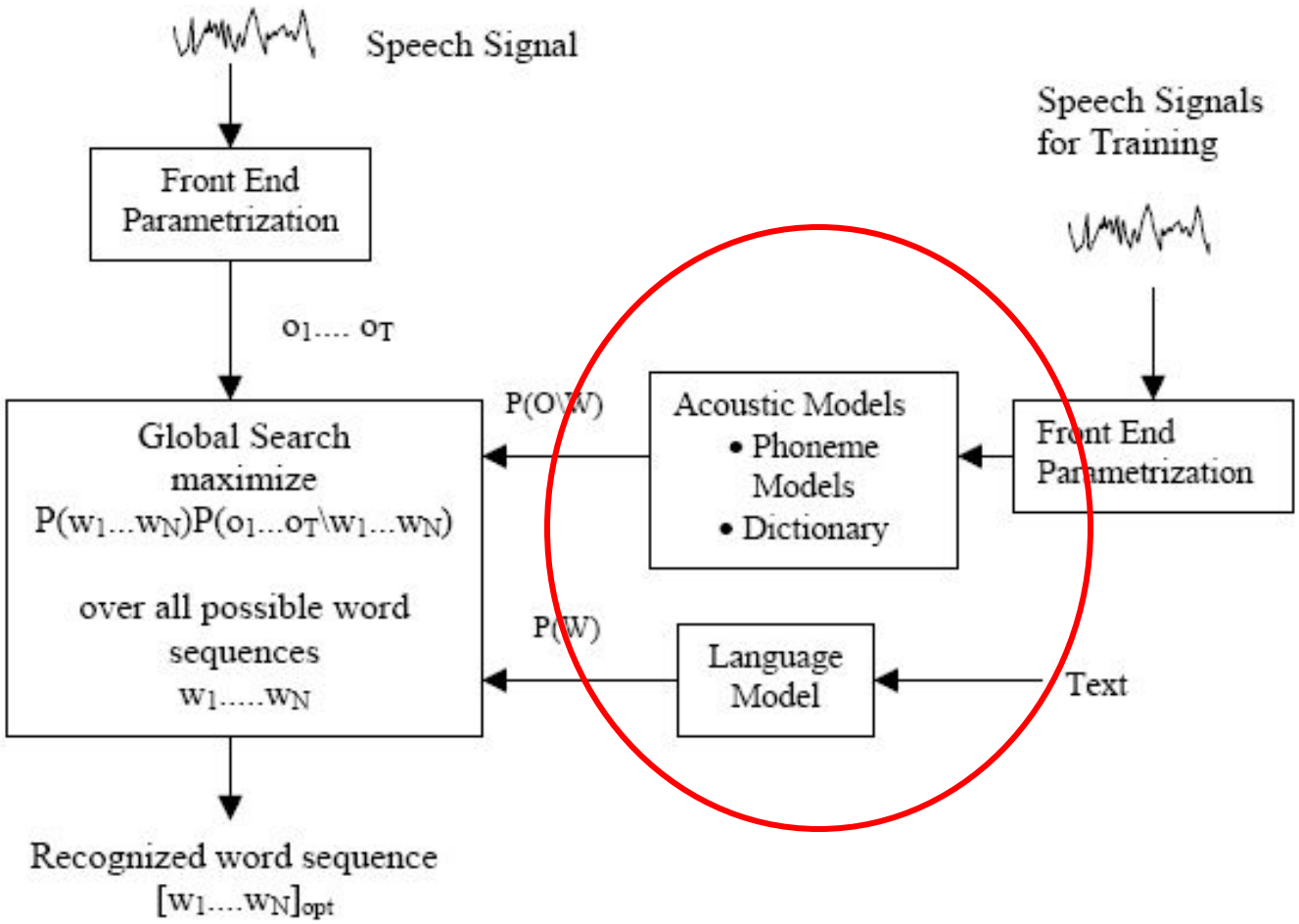
Training with Multiple Vector Sequences



The procedure can be continued until convergence

Convergence is achieved when the total best-alignment error for all training sequences does not change significantly with further refinement of the model

STATISTICAL ASR ARCHITECTURE

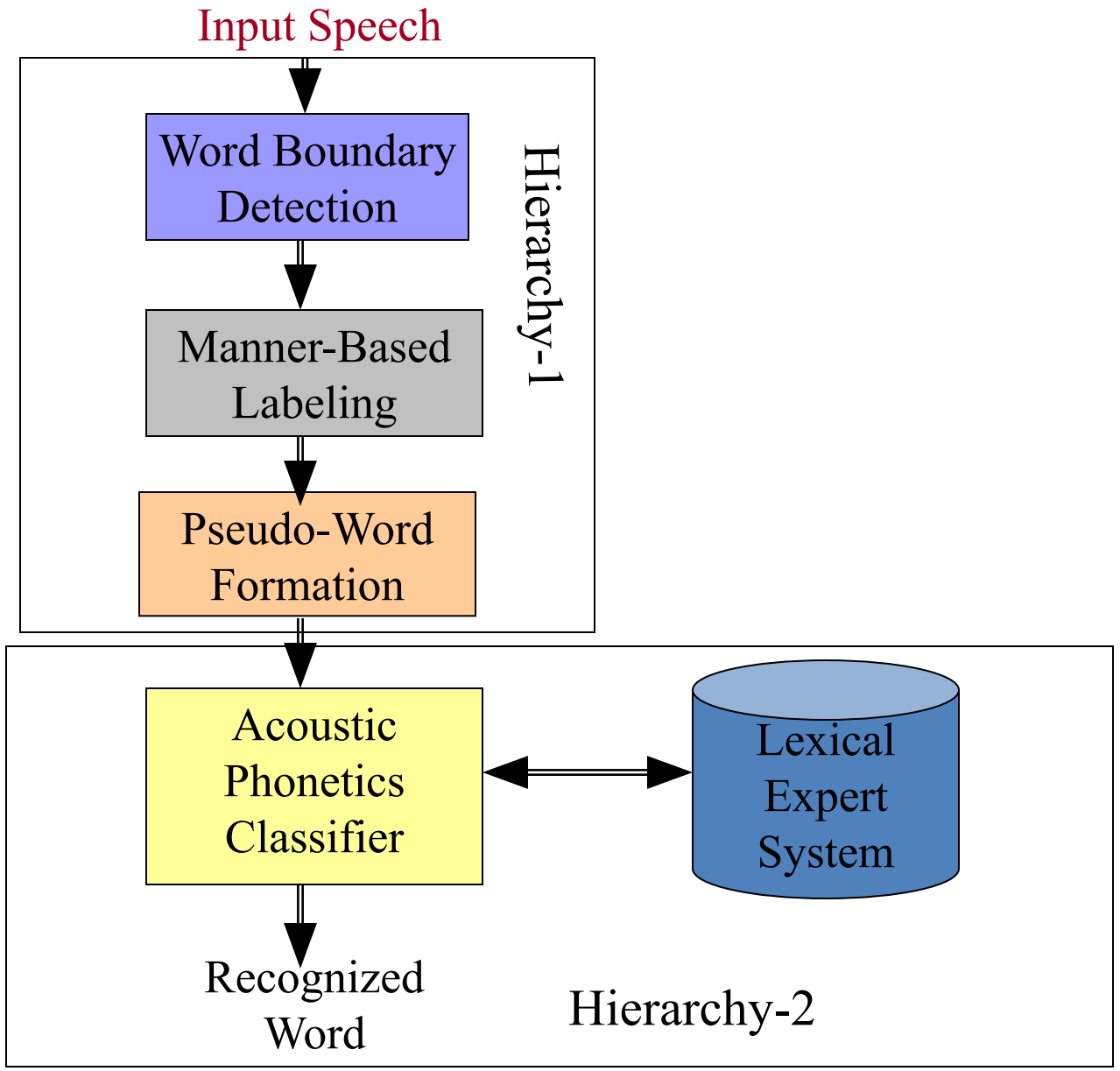


ASR: Machine vs. Human Performance

Corpus	Description	Vocabulary size	Human performance	Machine performance
Alphabet letters	Read alphabet letters	26	1.6%	5%
Resource Management	Read sentences	1000	0.1%	3.6%
North American Business News	Read sentences	5000-Unlimited	0.9%	7.2%
Switchboard	Spontaneous telephone conversations	2000-Unlimited	4%	40%
Switchboard wordspotting	Spontaneous telephone conversations	20 keywords	12.8%	31.3%

Word-error rate: Matched Condition

Lexically Driven Manner Based Speech Recognition Model



Limitation of Current ASR using stochastical Hidden Markov Models (HMMs)

- Current automatic speech recognition systems (ASRs) have pushed the possibilities of stochastical Hidden Markov Models (HMMs) to their limits but have not reached the performance of human listeners.
- HMMs need inconveniently large annotated corpora to train them for any new language or dialect they are applied to.

Solution Proposed in MLDB

✔ The proposed model is intended to overcome these limitations by a system that is based on an extremely efficient lexical coding and access model (MBLDM, *manner based lexically driven model*).

✔ The intention is the development of a multi-speaker, multi-language automatic speech recognition system, which, in a first implementation, can handle speaker-independent speech input for a reasonable vocabulary (±50,000 words) without prior training in Bangla continuous speech.

✔ Due to the construction of the system, additional languages or language dialects can be added by adding (word-) lexica and some language-specific parameters, without additional acoustic input.

MANNERS AND PLACE OF ARTICULATION OF PHONEME

- During the articulation the airstreams through the vocal tract must be obstructed in some way. The place where the obstruction takes place is called the place of articulation
- Manner of articulation is concerned with airflow ; the paths it take and the degree to which it is impeded by vocal tract constrictions.

The consonants are classified depending on the place of obstruction and manner of articulation.

ক */k/* Velar Un-aspirated unvoiced stop

Vowel sound may be specified in terms of the position of the tongue and the position of the lips.

ই, ঈ */I/* High front Un-rounded

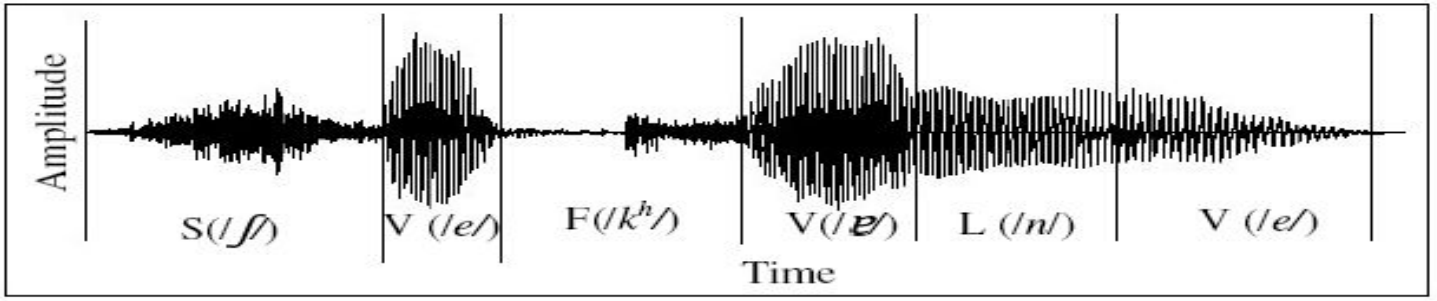
Final selected manner for pseudo words generation

Sl no.	Manner	Bangla phonemes for manner base labeling
1	S	/ʃ/, /s/
2	P	/k/, /t/, /ʈ/, /p/
3	F	/k^h/, /t^h/, /$ʈ^h$/, /p^h/, /ʧ/, /ʧʰ/
4	A	/g/, /d/, /ɖ/, /b/, /g^h/, /d^h/, /$ɖ^h$/, /b^h/, /ʤ/, /ʤʰ/
5	L	/l/, /m/, /n/
6	V	/ɔ/, /ɐ/, /i/, /u/, /e/, /o/, /æ/

Pseudo word

If the pronunciation of a word is represented by the selected manner symbols then the new representation of the word is called pseudo word

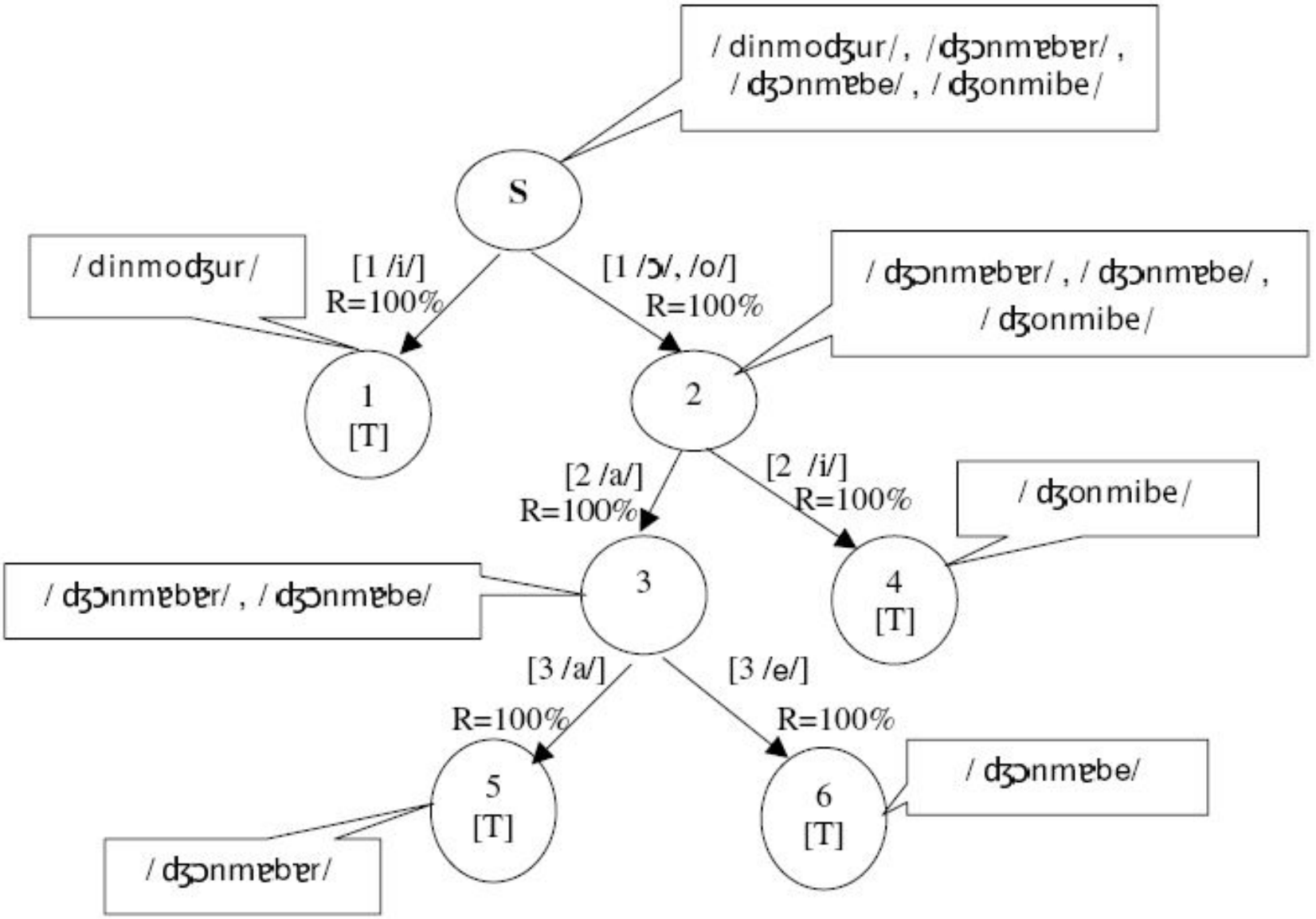
As for example if the word is /ʃekʰɐne/ then the corresponding pseudo word after manner based labeling is SVFVLV

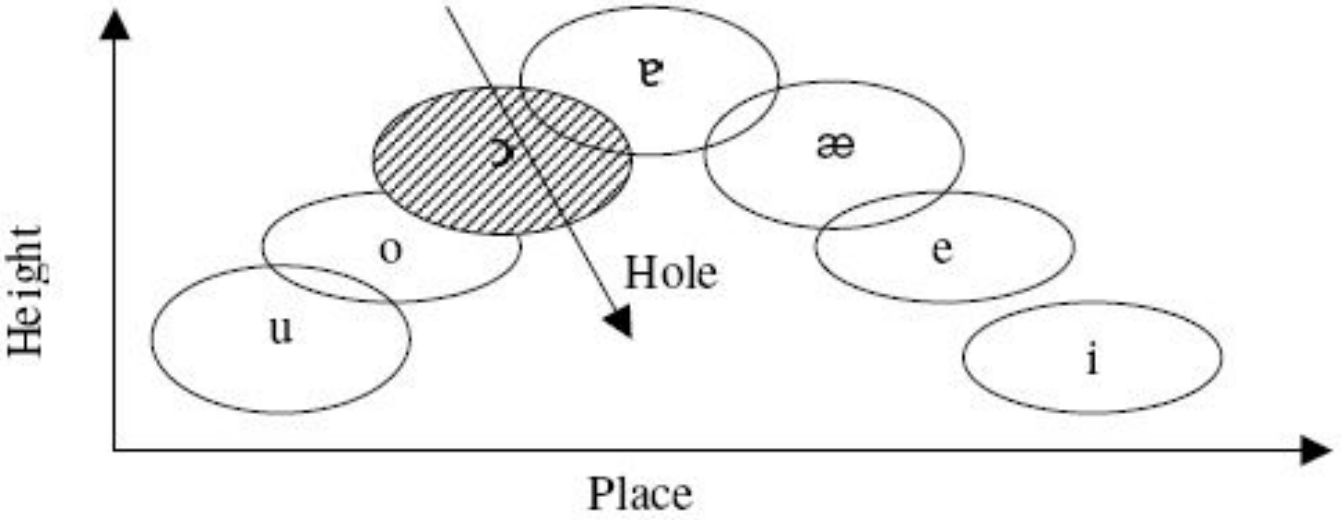


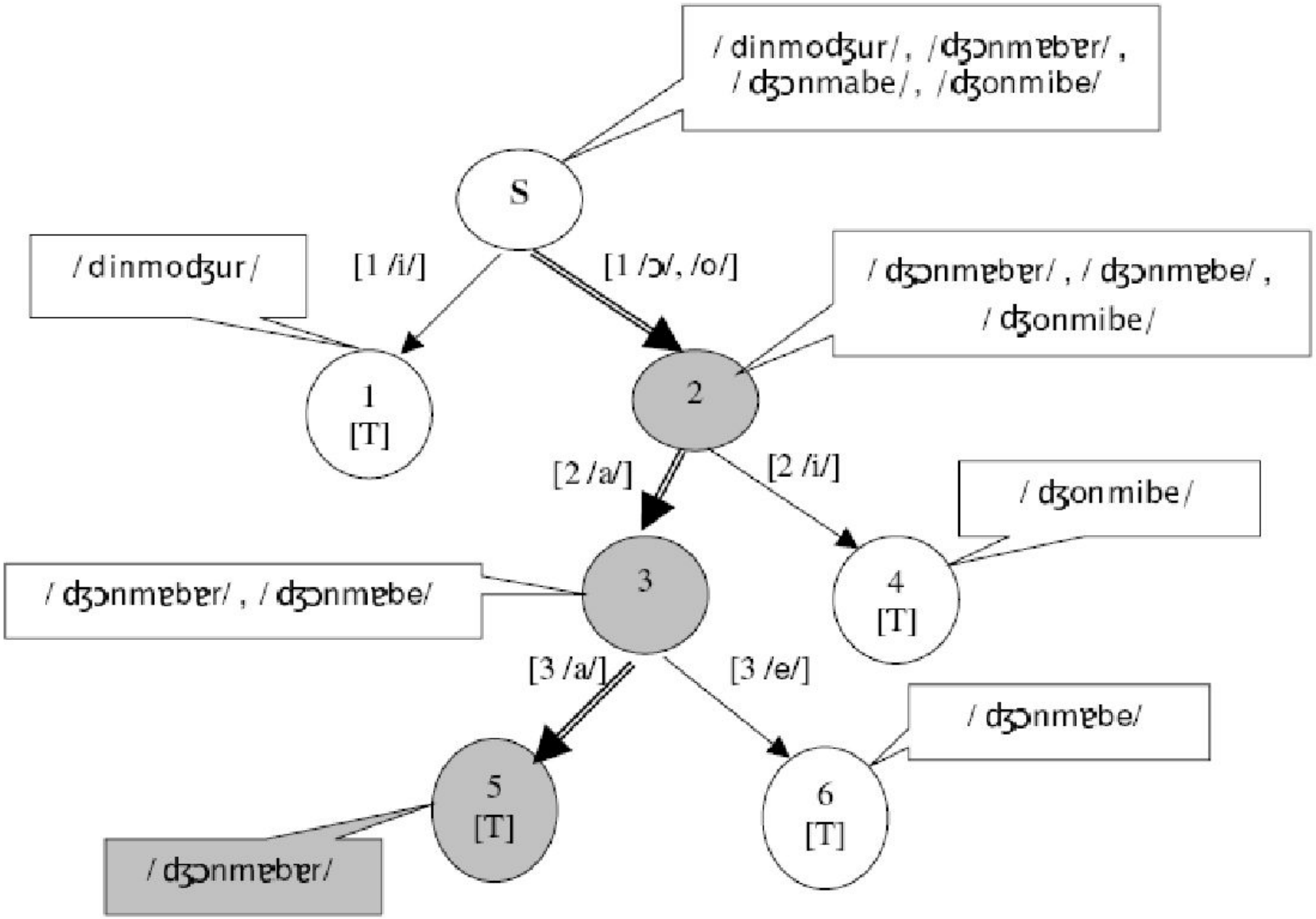
Cohorts

A group of words of same pseudo word representation forms a cohort. For Example cohort AVLLVAV consists of four Bangla words, /dinmodʒur/, /dʒɔnmɐbɐr/, /dʒɔnmɐbe/, /dʒonmibe/

দিনমজুর, জন্মাবার, জন্মাবে, জন্মিবে


/ dinmodʒur/ , /dʒɔnmɐbɐr/ , / dʒɔnmɐbe/ , / dʒonmibe/
S
/ dinmodʒur/
[1 /i/] R=100%
[1 /ɔ/, /o/] R=100%
/ dʒɔnmɐbɐr/ , / dʒɔnmɐbe/ , / dʒonmibe/
1 [T]
2
[2 /a/] R=100%
[2 /i/] R=100%
/ dʒonmibe/
/ dʒɔnmɐbɐr/ , / dʒɔnmɐbe/
3
4 [T]
[3 /a/] R=100%
[3 /e/] R=100%
/ dʒɔnmɐbe/
5 [T]
6 [T]
/ dʒɔnmɐbɐr/


ɐ
ɔ
æ
o
Hole
e
u
i
Height
Place


/ dinmodʒur/ , /dʒɔnmɐbɐr/ ,
/ dʒɔnmabe/ , /dʒonmibe/
S
/ dinmodʒur/
[1 /i/]
[1 /ɔ/, /o/]
/ dʒɔnmɐbɐr/ , / dʒɔnmɐbe/ ,
/ dʒonmibe/
1
[T]
2
[2 /a/]
[2 /i/]
/ dʒonmibe/
/ dʒɔnmɐbɐr/ , / dʒɔnmɐbe/
3
4
[T]
[3 /a/]
[3 /e/]
/ dʒɔnmɐbe/
5
[T]
6
[T]
/ dʒɔnmɐbɐr/

Thus the actual recognition rate =recognition rate of manner based labeling x over all recognition rate of the cohort =94*95.5 =90% (Using Manner Based Lexically Driven (MBLD) Model of Speech Recognition)