Lecture 5
The Big Picture/Language Modeling

Michael Picheny, Stanley F. Chen, Ellen Eide
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA
{picheny,stanchen,eeide}@us.ibm.com

October 6, 2005

Administrivia

Outline

- The Big Picture (review)
  - The story so far, plus situating Lab 1 and Lab 2
- language modeling
  - why?
  - how? (grammars, n-grams)
  - parameter estimation (smoothing)
  - evaluation

Pattern Classification

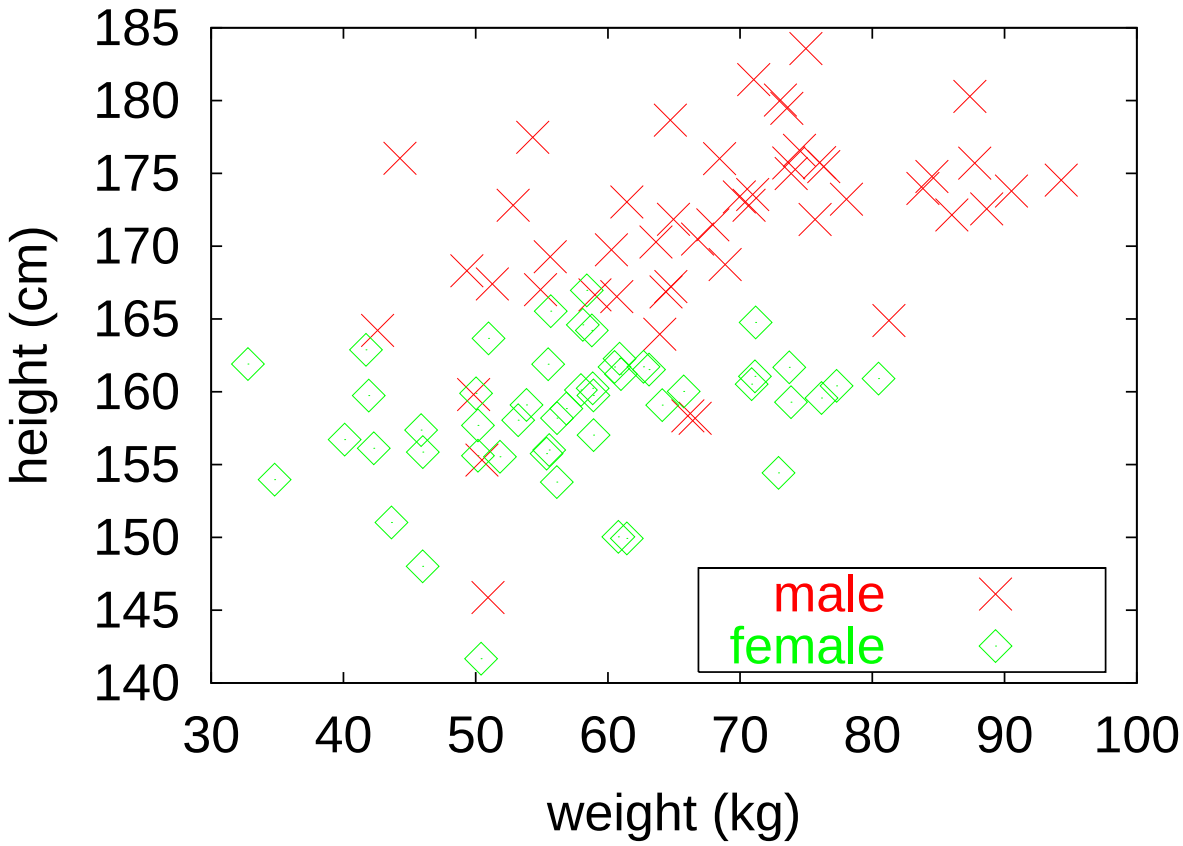
- given training samples $(\mathbf{x}_1, \omega_1), \ldots, (\mathbf{x}_N, \omega_N)$
  - feature vector $\mathbf{x} = (x_1, \ldots, x_T)$
  - class labels ω
- guess class label of unseen feature vectors $\mathbf{x}$ (*i.e.*, from the *test* set)
- example: guess gender of person given height+weight
  - feature vector $\mathbf{x} = (\text{height}, \text{weight})$
  - class labels $\omega \in \{\text{M}, \text{F}\}$

Speech Recognition is Pattern Classification

- *e.g.*, isolated digit recognition
  - feature vector $\mathbf{x} = (x_1, \ldots, x_T)$ is acoustic waveform
    - T=20000 if 1 sec sample, 20kHz sampling rate
  - class labels $\omega \in \{\text{ONE}, \text{TWO}, \ldots, \text{NINE}, \text{ZERO}, \text{OH}\}$

Nearest Neighbor Classification

- to classify a test vector $\mathbf{x}$
  - find nearest sample $\mathbf{x}_i$ in training set
  - select associated class ω_i



Nearest Neighbor Classification

Let's write down some equations

- for each class ω, we have a *model* M_ω
- to classify a test vector $\mathbf{x}_{\text{test}}$, find nearest model M_ω

$$\text{class}(\mathbf{x}_{\text{test}}) = \arg\min_{\omega} \text{ distance}(\mathbf{x}_{\text{test}}, M_\omega)$$

- nearest neighbor classification
  - let $M_\omega = \{\text{training samples } \mathbf{x} \text{ labeled with class } \omega\}$
  - $\text{distance}(\mathbf{x}_{\text{test}}, M_\omega) = \min_{\mathbf{x} \in M_\omega} \text{distance}(\mathbf{x}_{\text{test}}, \mathbf{x})$
  - *e.g.*, $\text{distance}(\mathbf{x}_{\text{test}}, \mathbf{x})$ = Euclidean distance

Speech Recognition as Pattern Classification

Problem 1: acoustic waveform is not good representation for nearest neighbor classification

- consider two equal-length acoustic waveforms

$$\begin{aligned} \mathbf{x} &= (x_1, \ldots, x_{20000}) \\ \mathbf{x}' &= (x'_1, \ldots, x'_{20000}) \end{aligned}$$

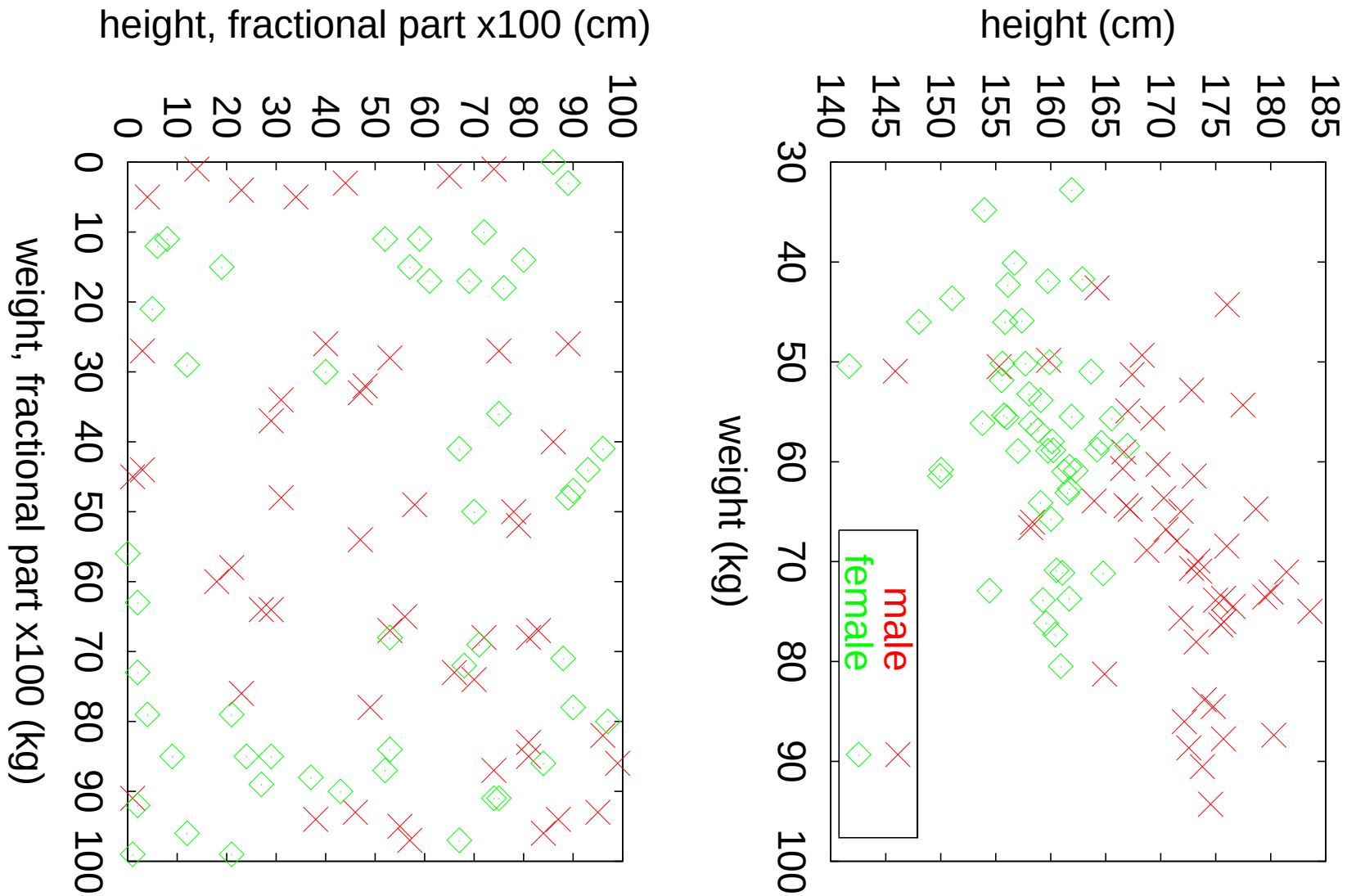
- do we believe that distance$(\mathbf{x}, \mathbf{x}')$ ...
  - will be smaller if class$(\mathbf{x}) =$ class$(\mathbf{x}')$ ...
  - than if class$(\mathbf{x}) \neq$ class$(\mathbf{x}')$?
- Lab 1: windowing only; computing error rate with DTW
  - windows are constant-length acoustic waveforms
  - performance is near random

Speech Recognition as Pattern Classification

Finding a transformation F for samples: $F(\mathbf{x}_{\text{raw}}) \Rightarrow \mathbf{x}_{\text{new}}$

- want samples $\mathbf{x}_{\text{new}}$ in same class to be close to each other
- want samples $\mathbf{x}_{\text{new}}$ in different classes to be far apart
- $\Rightarrow$ signal processing/front end processing

Feature Transformation



What Did We Learn in Lab 1?

- a smoothed frequency-domain representation works well
  - FFT computes energy at each frequency (fine-grained)
  - mel-binning does coarse-grained grouping (93.6% accuracy)
  - log+DCT, Hamming window help
  - not completely unlike human auditory system
- accuracy on “PCM” (pulse code modulation): 89.1%
  - DTW is powerful; speaker-dependent only?
- DTW is OK for speaker-dependent small-voc isolated word ASR
  - $>$95% accuracy in clean conditions
- data reduction
  - instead of $(x_1, \ldots, x_{20000})$ for 1 sec sample
  - $(\vec{x}_1, \ldots, \vec{x}_{100})$, dim$(\vec{x}_t) = 12$

Speech Recognition as Pattern Classification

Problem 2: acoustic waveforms have different lengths

- *e.g.*, how to compute:

$$\text{distance}\{(\vec{x}_1, \ldots, \vec{x}_{100}), (\vec{x}'_1, \ldots, \vec{x}'_{150})\}$$

- answer: dynamic time warping (asymmetric)
  - introduce concept of *alignment* $A = a_1 \cdots a_T$
  - map each test frame t to a template frame a_t

Alignments

- take minimum distance over all possible alignments A

$$\text{distance}(\mathbf{x}, \mathbf{x}') = \min_A \text{ distance}_A(\mathbf{x}, \mathbf{x}')$$

- distance given an alignment is ...
  - sum of distances between each pair of aligned frames

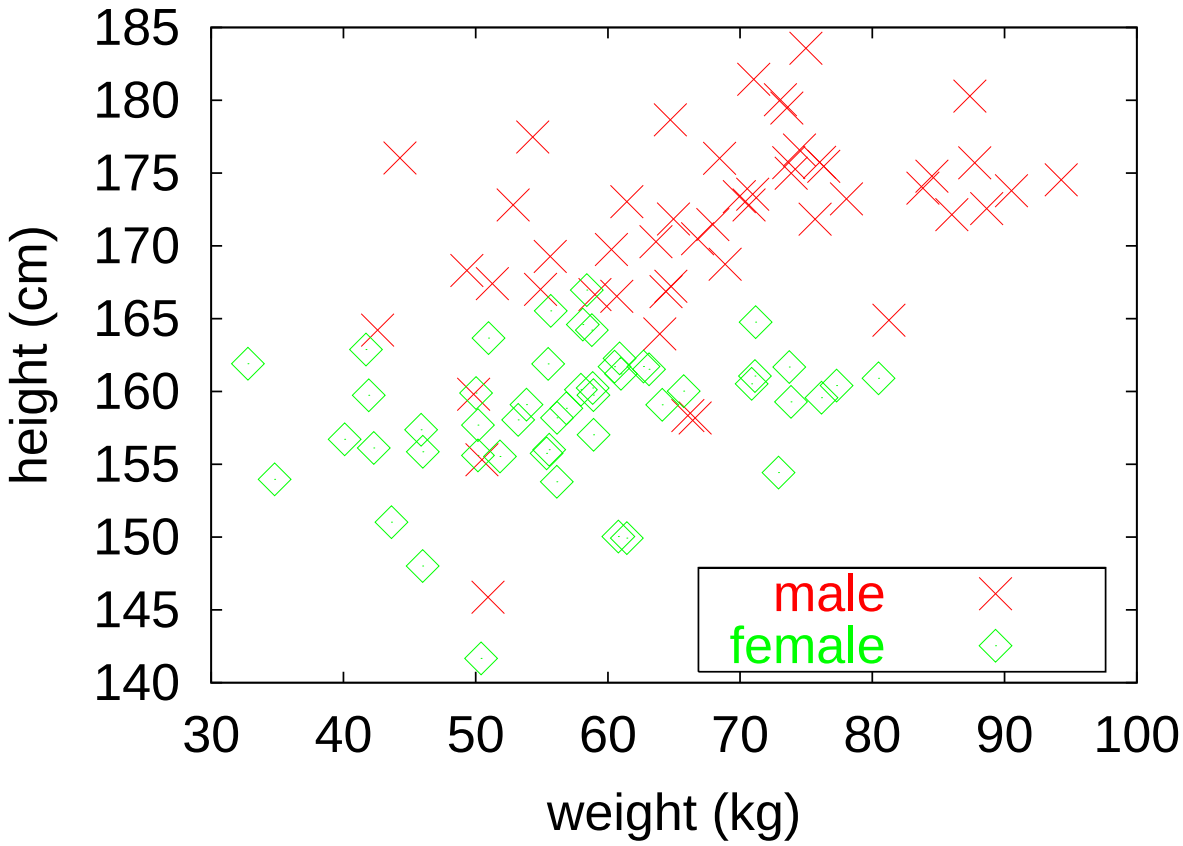
$$\text{distance}_A\{(\vec{x}_1, \ldots, \vec{x}_T), (\vec{x}'_1, \ldots, \vec{x}'_{T'})\} = \sum_{t=1}^{T} \text{distance}(\vec{x}_t, \vec{x}'_{a_t})$$

- *dynamic programming* computes $\min_A \text{distance}_A(\cdot, \cdot)$ efficiently
  - search exponential number of alignments in polynomial time

DTW as Nearest Neighbor Classification

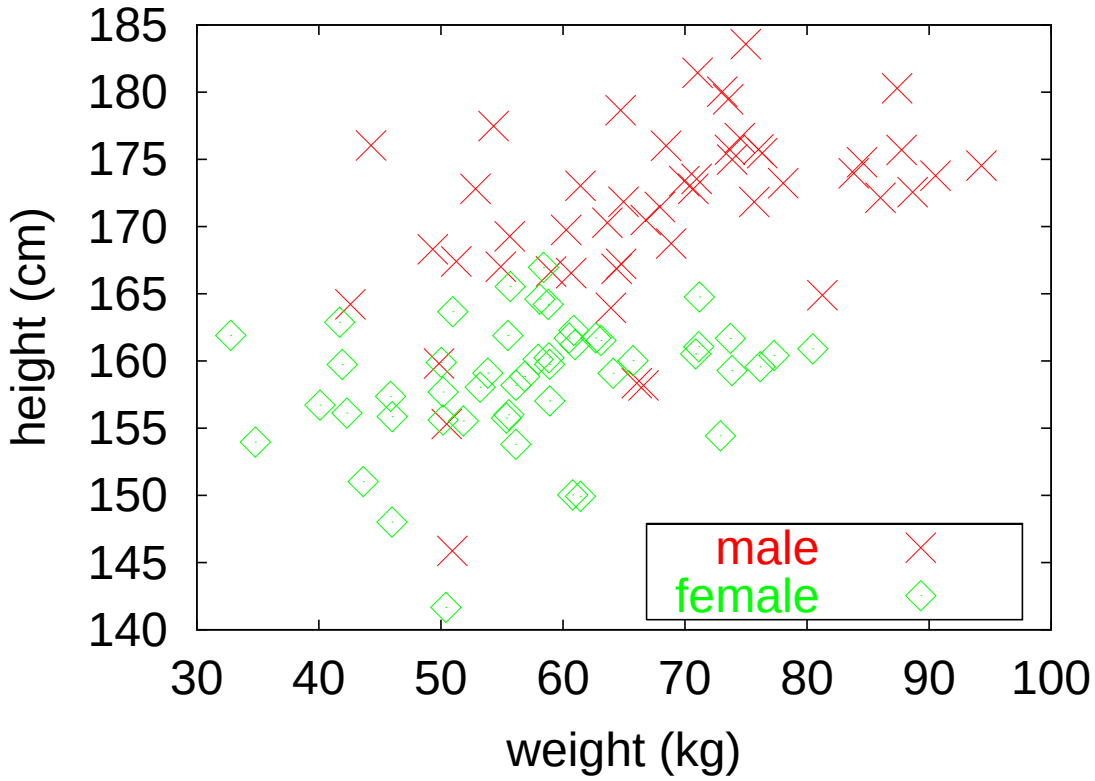
Lab 1: collect only one training sample $\mathbf{x}_\omega$ per class ω

- variation: keep many/all training samples



Long Live Nearest Neighbor?

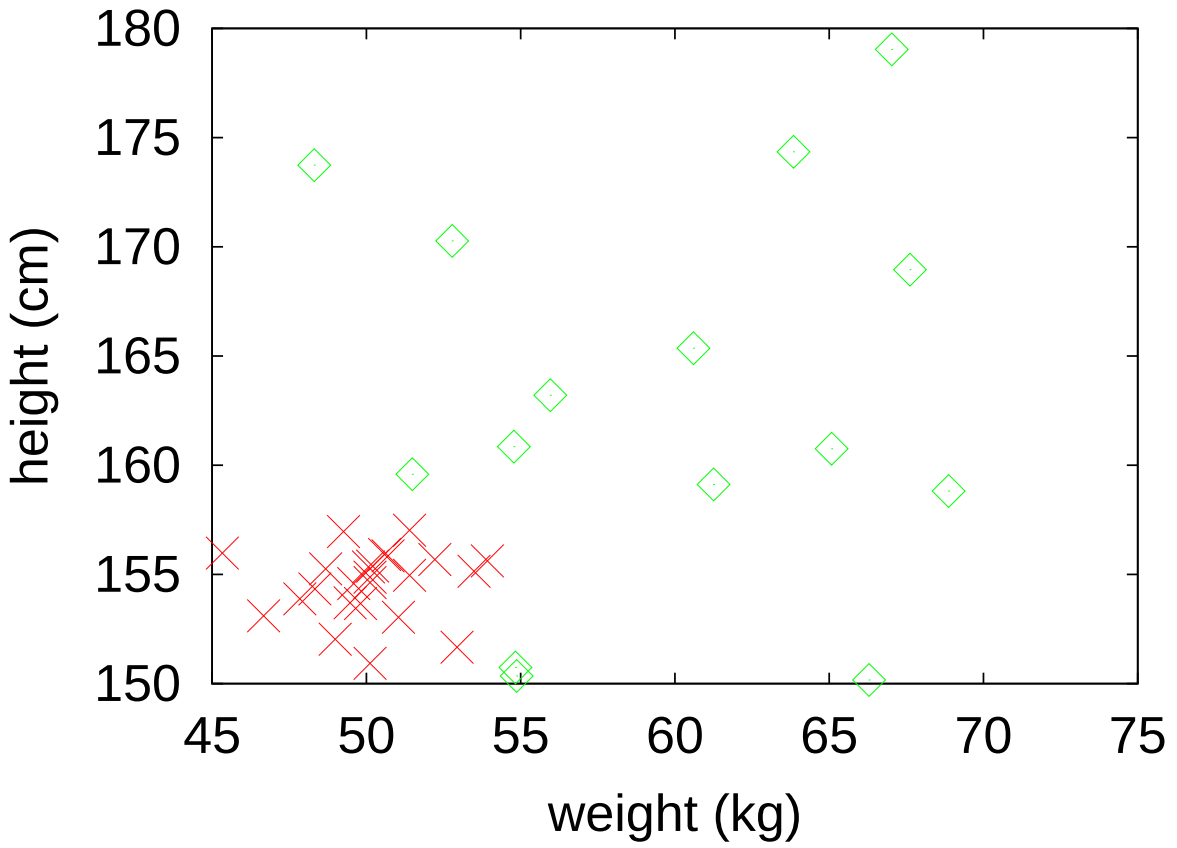
Issue 1: outliers



- k-nearest neighbors?
  - find $k > 1$ nearest training samples; pick most frequent class

Long Live k-Nearest Neighbors?

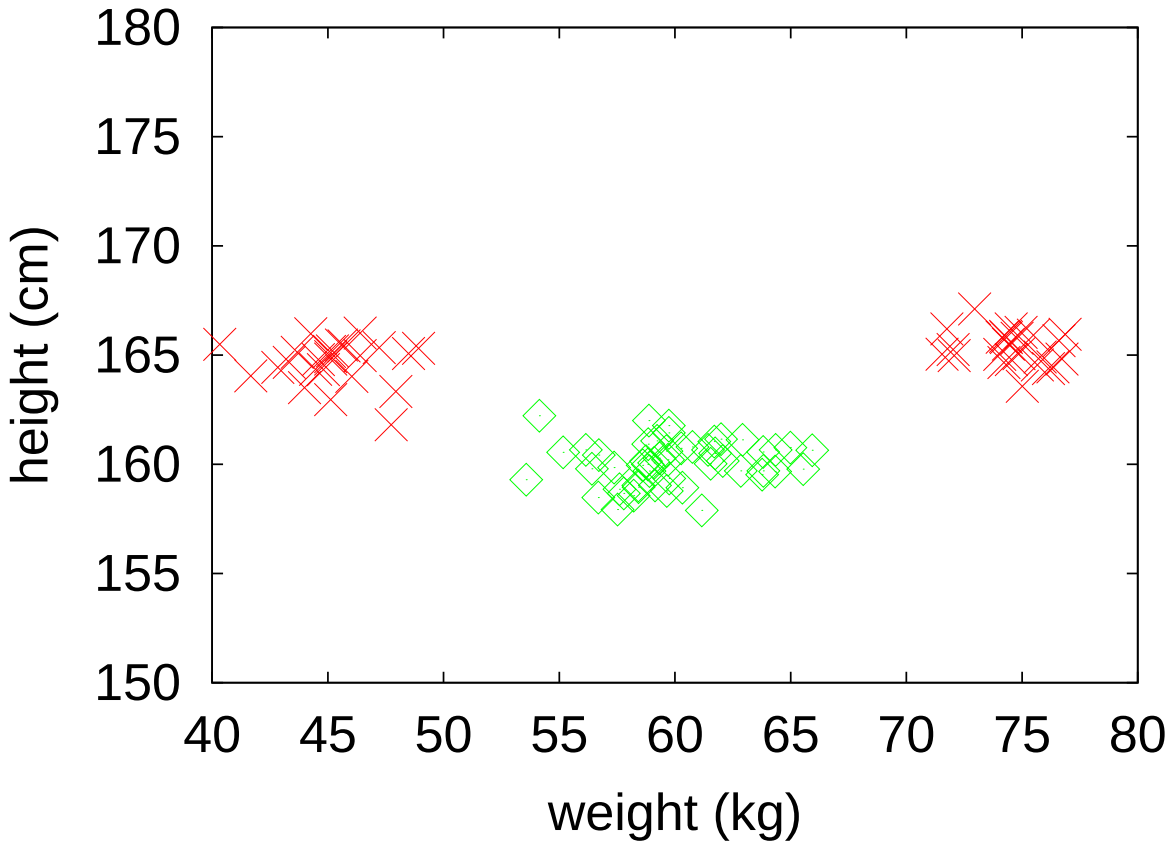
Issue 2: different sample densities



- averaging all samples in a class to find “canonical” sample?

Long Live Canonical Samples?

Issue 3: bimodal distributions



- what if we try to compute several “canonical” samples?

Is There a Better Way?

- using “distance” is too restrictive?
- more generally, want some sort of “score”: $\text{score}_\omega(\mathbf{x})$
  - reflects how strongly point $\mathbf{x}$ belongs to class ω
- to classify a sample $\mathbf{x}$, pick class with highest score

Is There a Better Way?

- is there a framework for assigning scores that . . .
  - can handle arbitrarily weird sample distributions?
    - outliers, varying sample densities, bimodal, etc.
  - provides guidance on how to find models that perform well at classification?
  - provides tools for finding these models efficiently?
- $\Rightarrow$ probabilistic modeling!

Probability 101

- probabilities $P(x)$ are nonnegative scores that sum to 1

$$\sum_x P(x) = 1 \qquad P(x) \geq 0$$

  - semantics: $P(x) \sim$ frequency of event x
- joint probabilities: $P(x, y)$

$$\sum_{x,y} P(x, y) = 1 \qquad P(x, y) \geq 0$$

Probability 101

- marginal probabilities

$$P(x) = \sum_y P(x, y) \qquad P(y) = \sum_x P(x, y)$$

  - probability that I trip is equal to probability I trip and fall plus the probability I trip and don't fall

- conditional probabilities

$$P(x, y) = P(x)P(y|x) = P(y)P(x|y)$$

  - probability that I trip and fall is equal to probability that I trip times the probability that I fall given that I trip

Probabilistic Modeling as Nearest Neighbor

- for each class ω, we have a *model* $P_\omega(\cdot)$
  - $P_\omega(\mathbf{x})$ is a probability distribution over samples $\mathbf{x}$
  - proportional to frequency that samples from class ω are located at/around $\mathbf{x}$
- to classify a test vector $\mathbf{x}_{\text{test}}$, find nearest model $P_\omega(\cdot)$

$$\text{class}(\mathbf{x}_{\text{test}}) = \arg\min_{\omega} \text{ distance}(\mathbf{x}_{\text{test}}, P_\omega(\cdot))$$

- define distance as negative log probability
  - $\text{distance}(\mathbf{x}_{\text{test}}, P_\omega(\cdot)) \equiv -\log P_\omega(\mathbf{x}_{\text{test}})$

What Does Probabilistic Modeling Give Us?

- if each of our class probability distributions $P_\omega(\mathbf{x})$ are (perfectly) accurate . . .
  - we can perform classification optimally!
- *e.g.*, two-class classification $\omega \in \{\mathrm{M}, \mathrm{F}\}$ (equally frequent)
  - choose M if $P_M(\mathbf{x}) > P_F(\mathbf{x})$
  - choose F if $P_F(\mathbf{x}) > P_M(\mathbf{x})$
  - this is the best you can do!

What Does Probabilistic Modeling Give Us?

- consider a simple (1-D) probability distribution (Gaussian)
  - $P_\omega(\mathbf{x}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$
  - parameters μ, σ
  - assume distribution of samples from class ω is truly Gaussian
- maximum likelihood estimate $(\mu_{\text{MLE}}, \sigma_{\text{MLE}})$
  - choose parameter values that maximize likelihood/probability of training data $(\mathbf{x}_1, \omega_1), \ldots, (\mathbf{x}_N, \omega_N)$

$$(\mu_{\text{MLE}}, \sigma_{\text{MLE}}) = \arg\max_{(\mu,\sigma)} \prod_{i=1}^{N} P_{\omega_i}(\mathbf{x}_i)$$

What Does Probabilistic Modeling Give Us?

- in the presence of infinite training samples
  - maximum likelihood estimation (MLE) approaches the “true” parameter estimates
  - for most models, MLE is asymptotically consistent, unbiased, and efficient
- maximum likelihood estimation is easy for a wide class of models
  - count and normalize

Long Live Probabilistic Modeling!

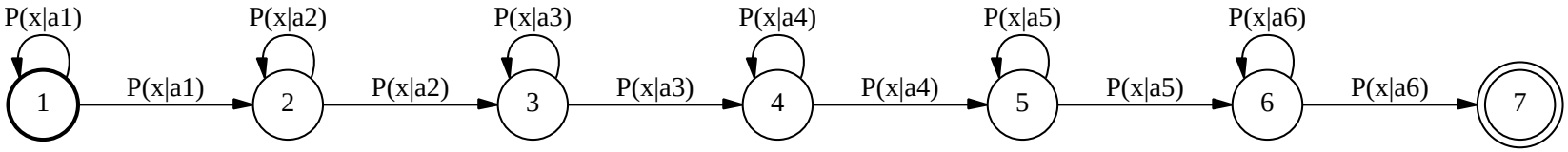
- if we choose form of $P_\omega(\mathbf{x})$ correctly and have lots of data . . .
  - we win!
- ASR moved to probabilistic modeling in mid-70's
- in ASR, no serious challenges to this framework since then
- by and large, probabilistic modeling . . .
  - does as well or better than all other techniques . . .
  - on all classification problems
  - (not everyone agrees)

What's the Catch?

- guarantees only hold if
  - know form of probability distribution that data comes from
    - MLE tells us how to find parameters only if we know the form of the model
  - can actually find MLE
    - in training HMM's w/ FB, can only find local optimum
    - MLE solution for GMM's is pathological
  - infinite training set
    - "there's no data like more data"
- these are the challenges!

From DTW to Probabilistic Modeling

- HMM's are natural way to express DTW as probabilistic model
- replace template for each word with an HMM
  - each frame in template corresponds to state in HMM



- alignment $A = a_1 \cdots a_T$
  - DTW: map each test frame t to template frame a_t
  - HMM: map each test frame t to arc a_t in HMM
    - each alignment corresponds to path through HMM
- can design HMM such that (see Holmes, Sec. 9.13, p. 155)

$$\text{distance}^{\text{DTW}}(\mathbf{x}_{\text{test}}, \mathbf{x}_\omega) \approx -\log P_\omega^{\text{HMM}}(\mathbf{x}_{\text{test}})$$

DTW *vs.* HMM's

DTW	HMM
acoustic template	HMM
frame in template	state in HMM
DTW alignment	path through HMM
move cost	transition (log)prob
distance between frames	output (log)prob
DTW search	Forward/Viterbi algorithm

Key Points for HMM's

- an HMM describes a probability distribution over observation sequences $\mathbf{x}$ (*i.e.*, acoustic feature vectors): $P(\mathbf{x})$
  - separate HMM for each word ω, describing likely acoustic feature vector sequences for that word: $P_\omega(\mathbf{x})$
- a path through the HMM
  - alignment $A = a_1 \cdots a_T$ between arcs a_t and observations (frames) x_t
- to calculate probability $P_\omega(\mathbf{x})$, sum (or max) probability for each alignment A
  - Forward algorithm: $P_\omega(\mathbf{x}) = \sum_A P_\omega(\mathbf{x}, A)$
  - Viterbi algorithm: $P_\omega(\mathbf{x}) \approx \max_A P_\omega(\mathbf{x}, A)$

Key Points for HMM's

- probability of alignment and observations

$$P_\omega(\mathbf{x}, A) = P_\omega(A) P_\omega(\mathbf{x}|A)$$

  - alignment probability $P_\omega(A)$: product of transition probabilities along path

$$P_\omega(A) = \prod_{t=1}^{T} P_{\text{trans}}(a_t)$$

  - observation probability $P_\omega(\mathbf{x}|A)$: product of observation probabilities along path

$$P_\omega(\mathbf{x}|A) = \prod_{t=1}^{T} P_{\text{obs}}(\vec{x}_t|a_t)$$

Key Points for HMM's

- each observation probability modeled as a Gaussian
  - diagonal covariance

$$P_{\text{obs}}(\vec{x}_t|a_t) = \prod_{\text{dim } d}^{D} \frac{1}{\sqrt{2\pi}\sigma_{a_t,d}} \exp\left[-\frac{1}{2}\left(\frac{x_{t,d}-\mu_{a_t,d}}{\sigma_{a_t,d}}\right)^2\right]$$

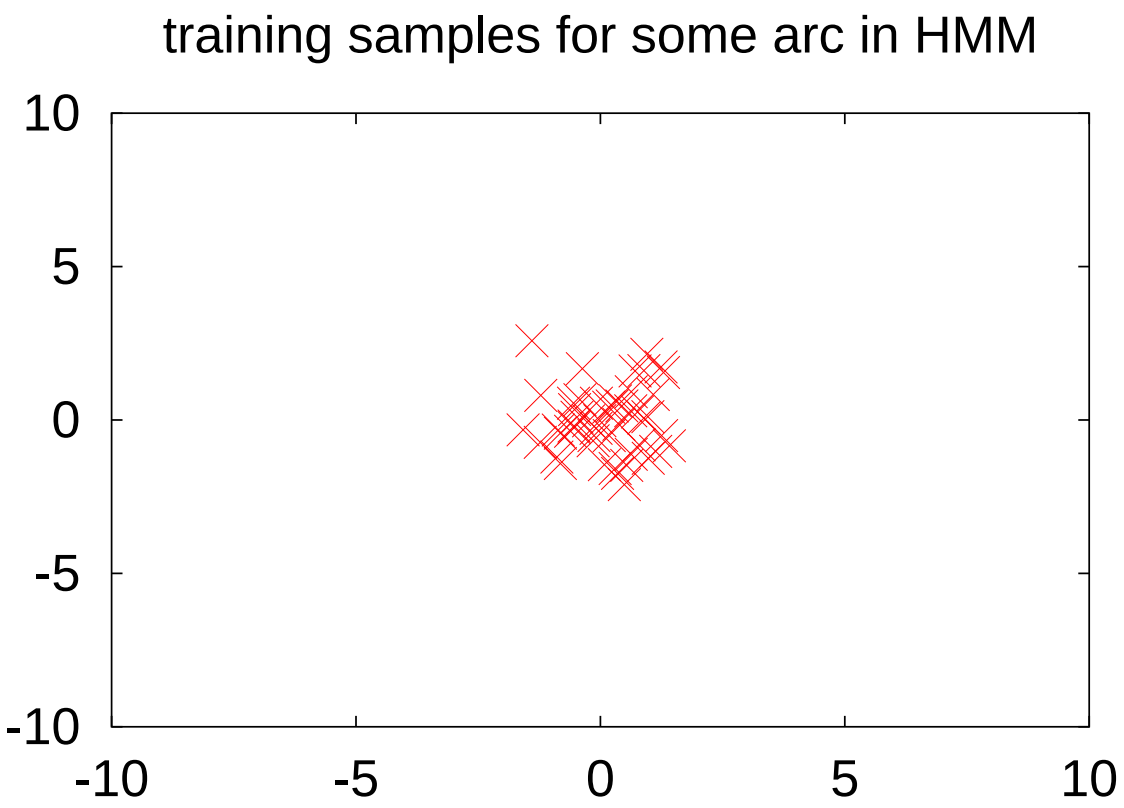
$$= \prod_{\text{dim } d}^{D} \mathcal{N}(x_{t,d};\mu_{a_t,d},\sigma_{a_t,d})$$

- or mixture of Gaussians

$$P_{\text{obs}}(\vec{x}_t|a_t) = \sum_{m=1}^{M} \lambda_{a_t,m} \prod_{\text{dim } d}^{D} \mathcal{N}(x_{t,d};\mu_{a_t,m,d},\sigma_{a_t,m,d})$$

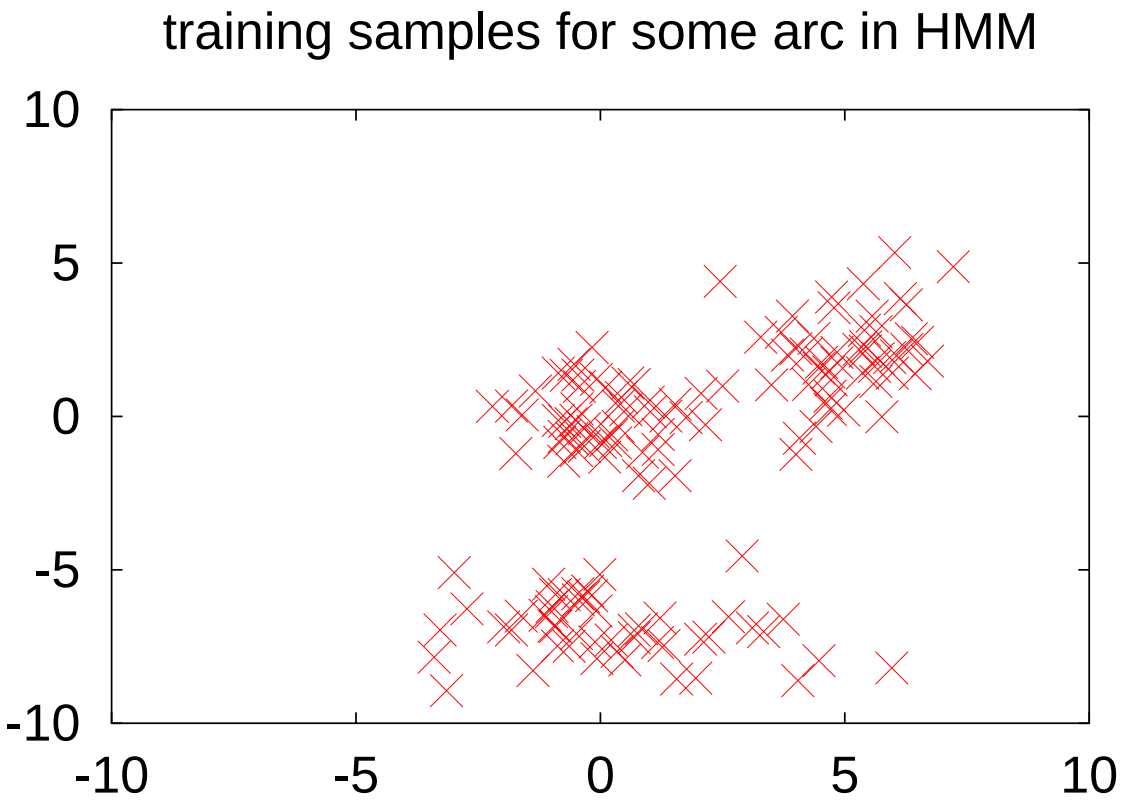
Why Gaussians and GMM's?

- a Gaussian is good at modeling a single cluster of samples



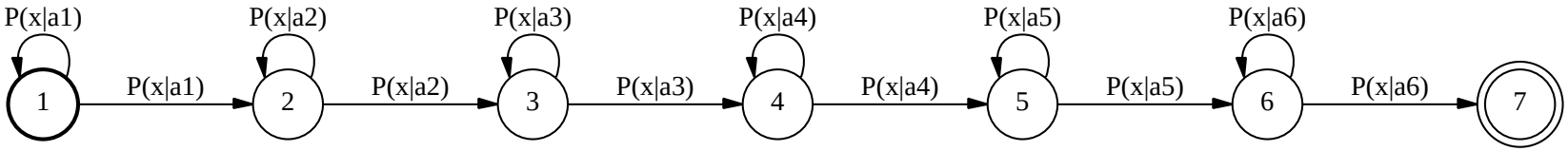
Why Gaussians and GMM's?

- if you sum enough Gaussians, you can approximate any distribution arbitrarily well



Optimizing HMM's for Isolated Word Recognition

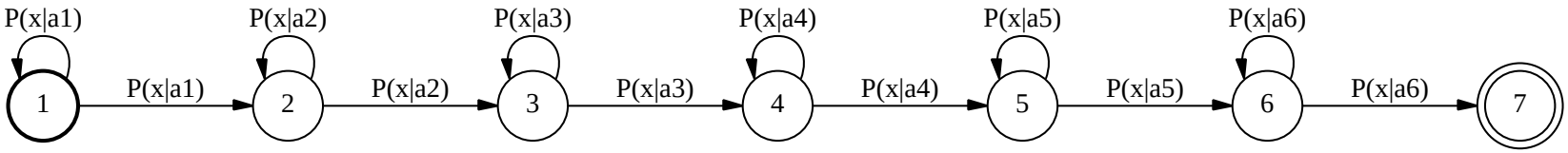
- each frame in DTW template for word ω corresponds to state in HMM?
- consecutive frames may be very similar to each other
  - no point having consecutive states in HMM with nearly identical output distributions



- how many states do we really need?
  - rule of thumb: three states per *phoneme*
  - one for start, middle, and end of phoneme

Word HMM's

- word TWO is composed of phonemes T UW
- two phonemes $\Rightarrow$ six HMM states



- convention: same output distribution for all arcs exiting a state
  - or place output distributions on states
- parameters: 12 transition probabilities, 6 output distributions

Isolated Word Recognition with HMM's

- training
  - for each class (word) ω, collect all training samples $\mathbf{x}$ with that label
  - to train HMM for word ω, run FB on these samples
- decoding a test sample $\mathbf{x}_{\text{test}}$
  - calculate (Viterbi or Forward) likelihood $P_\omega(\mathbf{x}_{\text{test}})$ of sample with each word model $P_\omega(\cdot)$, pick best

$$\text{class}(\mathbf{x}_{\text{test}}) = \arg\max_{\omega} \; P_\omega(\mathbf{x}_{\text{test}})$$

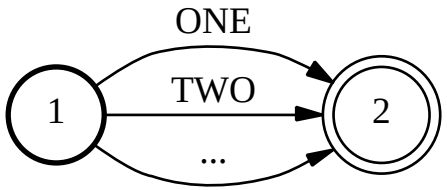
Control Flow for Training

For each iteration of Forward-Backward training

- initialize training counts to 0
- for each utterance $(\mathbf{x}_i, \omega_i)$ in training data
  - calculate acoustic feature vectors from $\mathbf{x}_i$ using front end
  - construct HMM representing word sequence ω_i
  - run FB algorithm to update training counts for all transitions present in HMM
- reestimate HMM parameters using training counts

Decoding with HMM's

- instead of separate HMM for each word ω
  - merge each word HMM into single big HMM



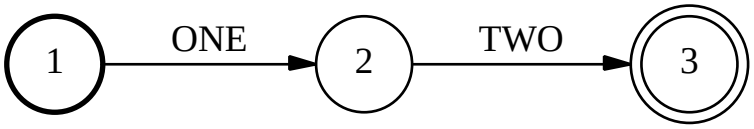
- use Viterbi algorithm (not Forward algorithm, why?)
  - in backtrace, collect all words along the way
- why one big graph?
  - when move to word sequences, can't have HMM per sequence
  - pruning during search
  - graph minimization

Continuous Speech Recognition with HMM's

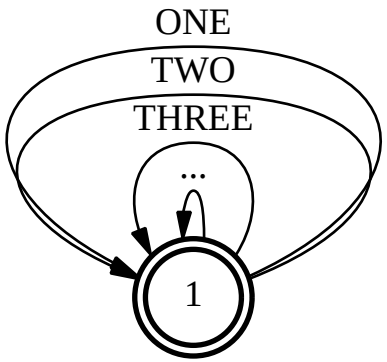
- *e.g.*, ASR for digit sequences of arbitrary length
  - set of classes ω for classification is infinite
- instead of separate HMM parameters for each digit string ω
  - have one HMM for each digit as before
  - glue together to make HMM for a digit string

Continuous Speech Recognition with HMM's

- training
  - *e.g.*, reference transcript: ONE TWO



  - for each training sample, update counts for each word HMM in transcript
- decoding
  - want HMM that represents all digit strings

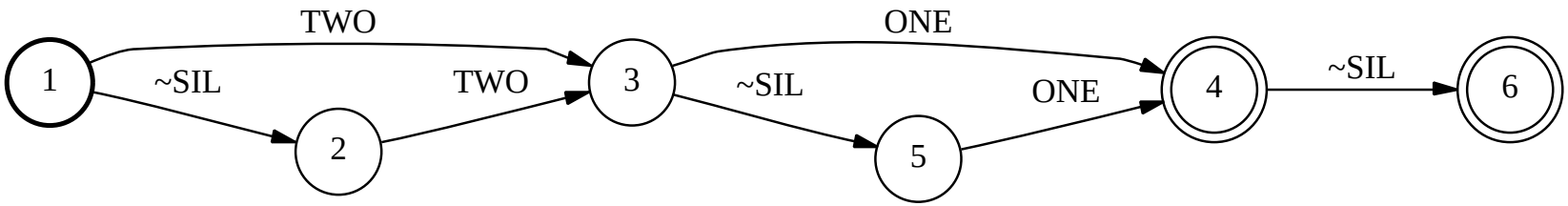


Silence is Golden

- what about all those pauses in speech?
  - at begin/end of utterance; in-between words
- add silence word $\sim$SIL to vocabulary
  - three states? (skip arcs?)
- allow optional silence at beginning/end of utterances and in-between words

Silence is Golden

- *e.g.*, HMM for training with reference script: ONE TWO



- in practice, often precompute best word sequence using some other model
  - *e.g.*, bootstrap models without using optional silences
- Lab 2: all these graphs are automatically constructed for you
  - silence is also used in isolated word training/decoding

Wreck a Nice Beach?

The story so far:

- to decode a test sample $\mathbf{x}_{\text{test}}$, find best word sequence ω

$$\text{class}(\mathbf{x}_{\text{test}}) = \arg\max_{\omega} \; P_{\omega}(\mathbf{x}_{\text{test}}) \equiv \arg\max_{\omega} \; P(\mathbf{x}_{\text{test}}|\omega)$$

  - find word sequence ω maximizing likelihood $P_{\omega}(\mathbf{x}_{\text{test}}|\omega)$
  - maximum likelihood classification
- well, you know:

  THIS IS OUR ROOM FOR A FOUR HOUR PERIOD .

  THIS IS HOUR ROOM FOUR A FOR OUR . PERIOD