EE E6820: Speech & Audio Processing & Recognition

Lecture 3: Machine learning, classification, and generative models

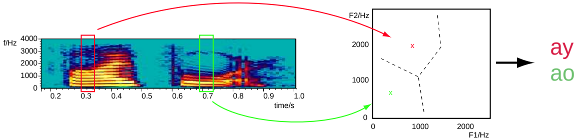
Michael Mandel <mim@ee.columbia.edu>

Columbia University Dept. of Electrical Engineering
http://www.ee.columbia.edu/∼dpwe/e6820

February 7, 2008

Outline

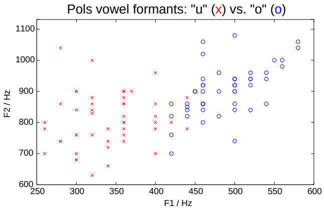
Classification and generative models



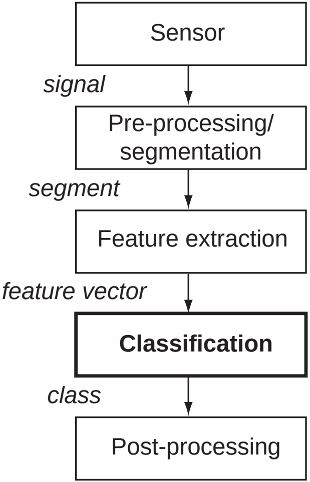
- Classification
  - discriminative models
  - discrete, categorical random variable of interest
  - fixed set of categories
- Generative models
  - descriptive models
  - continuous or discrete random variable(s) of interest
  - can estimate parameters
  - Bayes' rule makes them useful for classification

Building a classifier

- Define classes/attributes
  - could state explicit rules
  - better to define through ‘training’ examples
- Define feature space
- Define decision algorithm
  - set parameters from examples
- Measure performance
  - calculated (weighted) error rate



Classification system parts



- STFT
- Locate vowels

- Formant extraction

- Context constraints
- Costs/risk

Feature extraction

- Right features are critical
  - waveform vs formants vs cepstra
  - invariance under irrelevant modifications
- Theoretically equivalent features may act very differently in a particular classifier
  - representations make important aspects explicit
  - remove irrelevant information
- Feature design incorporates 'domain knowledge'
  - although more data $\Rightarrow$ less need for 'cleverness'
- Smaller 'feature space' (fewer dimensions)

  $\rightarrow$ simpler models (fewer parameters)

  $\rightarrow$ less training data needed

  $\rightarrow$ faster training

[inverting MFCCs]

Optimal classification

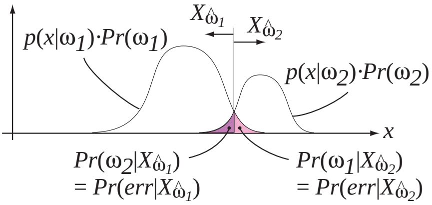
- Minimize probability of error with Bayes optimal decision

$$
\begin{aligned}
\hat{\theta} &= \operatorname*{argmax}_{\theta_i} p(\theta_i \mid x) \\
p(\text{error}) &= \int p(\text{error} \mid x) p(x)\, dx \\
&= \sum_i \int_{\Lambda_i} (1 - p(\theta_i \mid x)) p(x)\, dx
\end{aligned}
$$

  - where Λ_i is the region of x where θ_i is chosen

  ... but $p(\theta_i \mid x)$ is largest in that region
  - so $p(\text{error})$ is minimized

Sources of error



- Suboptimal threshold / regions (bias error)
  - use a Bayes classifier
- Incorrect distributions (model error)
  - better distribution models / more training data
- Misleading features ('Bayes error')
  - irreducible for given feature set
  - regardless of classification scheme

Two roads to classification

Optimal classifier is

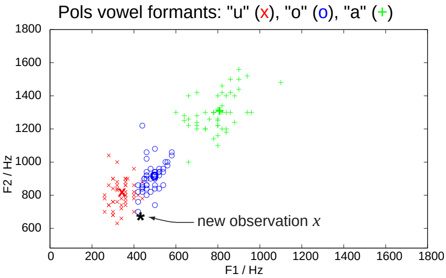
$$\hat{\theta} = \operatorname*{argmax}_{\theta_i} p(\theta_i \mid x)$$

but we don't know $p(\theta_i \mid x)$

- Can model distribution directly

  e.g. Nearest neighbor, SVM, AdaBoost, neural net
    - maps from inputs x to outputs θ_i
    - a discriminative model
- Often easier to model data likelihood $p(x \mid \theta_i)$
    - use Bayes' rule to convert to $p(\theta_i \mid x)$
    - a generative (descriptive) model

Nearest neighbor classification

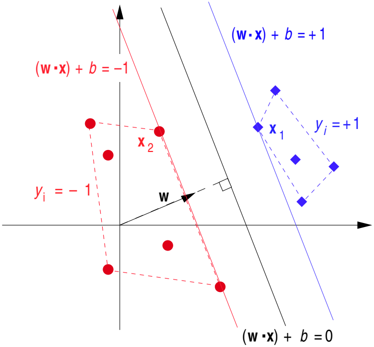


Find closest match (Nearest Neighbor)

- Naïve implementation takes $O(N)$ time for N training points
- As $N \to \infty$, error rate approaches twice the Bayes error rate
- With K summarized classes, takes $O(K)$ time
- Locality sensitive hashing gives approximate nearest neighbors in $O(dn^{1/c^2})$ time (Andoni and Indyk, 2006)

Support vector machines

- "Large margin" linear classifier for separable data
  - regularization of margin avoids over-fitting
  - can be adapted to non-separable data (C parameter)
  - made nonlinear using kernels $k(x_1, x_2) = \Phi(x_1) \cdot \Phi(x_2)$

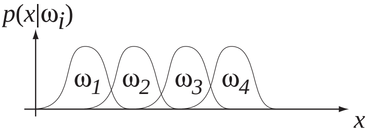


- Depends only on training points near the decision boundary, the support vectors
- Unique, optimal solution for given Φ and C

Outline

Generative models

- Describe the data using structured probabilistic models
- Observations are random variables whose distribution depends on model parameters
- Source distributions $p(x \mid \theta_i)$
  - reflect variability in features
  - reflect noise in observation
  - generally have to be estimated from data (rather than known in advance)



Generative models (2)

Three things to do with generative models

- Evaluate the probability of an observation, possibly under multiple parameter settings

$$p(x), \quad p(x \mid \theta_1), \quad p(x \mid \theta_2), \quad \ldots$$

- Estimate model parameters from observed data

$$\hat{\theta} = \operatorname*{argmin}_{\theta} C(\theta^*, \theta \mid x)$$

- Run the model forward to generate new data

$$\tilde{x} \sim p(x \mid \hat{\theta})$$

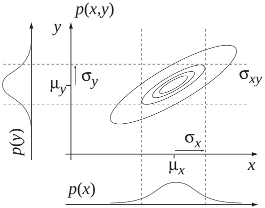
Random variables review

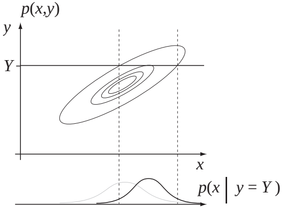
- Random variables have joint distributions, $p(x, y)$
- Marginal distribution of y

$$p(y) = \int p(x, y)\, dx$$

- Knowing one value in a joint distribution constrains the remainder
- Conditional distribution of x given y

$$p(x \mid y) \equiv \frac{p(x, y)}{p(y)} = \frac{p(x, y)}{\int p(x, y)\, dy}$$





Bayes' rule

$$p(x \mid y)p(y) = p(x, y) = p(y \mid x)p(x)$$
$$\therefore \quad p(y \mid x) = \frac{p(x \mid y)p(y)}{p(x)}$$

$\Rightarrow$ can reverse conditioning given priors/marginals

- terms can be discrete or continuous
- generalizes to more variables

$$p(x, y, z) = p(x \mid y, z)p(y, z) = p(x \mid y, z)p(y \mid z)p(z)$$

- allows conversion between joint, marginals, conditionals

Bayes' rule for generative models

- Run generative models backwards to compare them

$$\underset{\text{Posterior prob}}{p(\theta \mid x)} = \underset{\text{Evidence} = p(x)}{\overset{\text{Likelihood}}{\frac{p(x \mid \theta)}{\int p(x \mid \theta) p(\theta)}}} \cdot \underset{\text{Prior prob}}{p(\theta)}$$

- Posterior is the classification we're looking for
  - combination of prior belief in each class
  - with likelihood under our model
  - normalized by evidence (so $\int$ posteriors $= 1$)
- Objection: priors are often unknown

  ... but omitting them amounts to assuming they are all equal

Computing probabilities and estimating parameters

- Want probability of the observation under a model, $p(x)$
  - regardless of parameter settings
- Full Bayesian integral

$$p(x) = \int p(x \mid \theta) p(\theta) \, d\theta$$

- Difficult to compute in general, approximate as $p(x \mid \hat{\theta})$
  - Maximum likelihood (ML)

$$\hat{\theta} = \operatorname*{argmax}_{\theta} p(x \mid \theta)$$

  - Maximum *a posteriori* (MAP): ML + prior

$$\hat{\theta} = \operatorname*{argmax}_{\theta} p(\theta \mid x) = \operatorname*{argmax}_{\theta} p(x \mid \theta) p(\theta)$$

Model checking

- After estimating parameters, run the model forward
- Check that
  - model is rich enough to capture variation in data
  - parameters are estimated correctly
  - there aren't any bugs in your code
- Generate data from the model and compare it to observations

$$\tilde{x} \sim p(x \mid \theta)$$

  - are they similar under some statistics $T(x) : \mathbb{R}^d \mapsto \mathbb{R}$?
  - can you find the real data set in a group of synthetic data sets?
- Then go back and update your model accordingly
- Gelman et al. (2003, ch. 6)

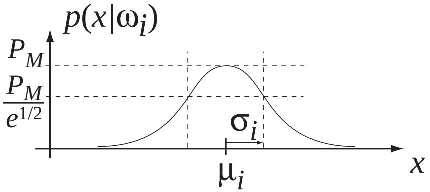
Outline

Gaussian models

- Easiest way to model distributions is via parametric model
  - assume known form, estimate a few parameters
- Gaussian model is simple and useful. In 1D

$$p(x \mid \theta_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left[-\frac{1}{2} \left(\frac{x - \mu_i}{\sigma_i} \right)^2 \right]$$

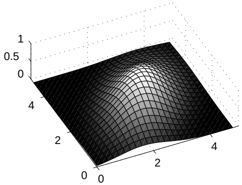
- Parameters mean μ_i and variance σ_i → fit

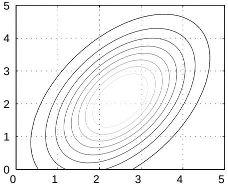


Gaussians in d dimensions

$$p(\mathbf{x} \mid \theta_i) = \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1}(\mathbf{x} - \mu_i)\right]$$

Described by a d-dimensional mean μ_i
and a $d \times d$ covariance matrix Σ_i





Gaussian mixture models

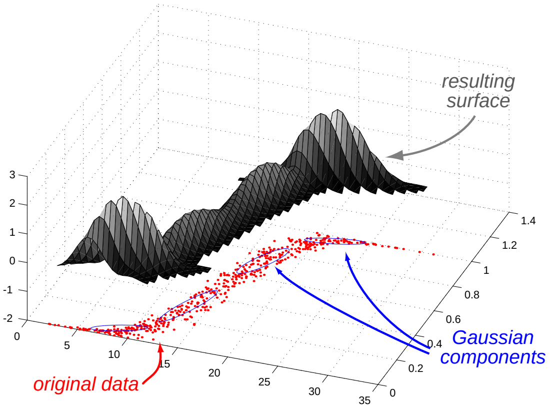
- Single Gaussians cannot model
  - distributions with multiple modes
  - distributions with nonlinear correlations
- What about a weighted sum?

$$p(x) \approx \sum_k c_k p(x \mid \theta_k)$$

  - where $\{c_k\}$ is a set of weights and $\{p(x \mid \theta_k)\}$ is a set of Gaussian components
  - can fit anything given enough components
- Interpretation: each observation is generated by one of the Gaussians, chosen with probability $c_k = p(\theta_k)$

Gaussian mixtures (2)

e.g. nonlinear correlation



Problem: finding c_k and θ_k parameters

- easy if we knew which θ_k generated each x

Expectation-maximization (EM)

- General procedure for estimating model parameters when some are unknown

  e.g. which GMM component generated a point

- Iteratively updated model parameters θ to maximize Q, the expected log-probability of observed data x and hidden data z

$$Q(\theta, \theta_t) = \int_z p(z \mid x, \theta_t) \log p(z, x \mid \theta)$$

  - E step: calculate $p(z \mid x, \theta_t)$ using θ_t
  - M step: find θ that maximizes Q using $p(z \mid x, \theta_t)$
  - can prove $p(x \mid \theta)$ non-decreasing
  - hence maximum likelihood model
  - local optimum—depends on initialization

Fitting GMMs with EM

- Want to find
  - parameters of the Gaussians $\theta_k = \{\mu_k, \Sigma_k\}$
  - weights/priors on Gaussians $c_k = p(\theta_k)$

  . . . that maximize likelihood of training data x
- If we could assign each x to a particular θ_k, estimation would be direct
- Hence treat mixture indices, z, as hidden
  - form $Q = \mathrm{E}\left[p(x, z \mid \theta)\right]$
  - differentiate wrt model parameters

  $\longrightarrow$ equations for μ_k, Σ_k, c_k to maximize Q

GMM EM updated equations

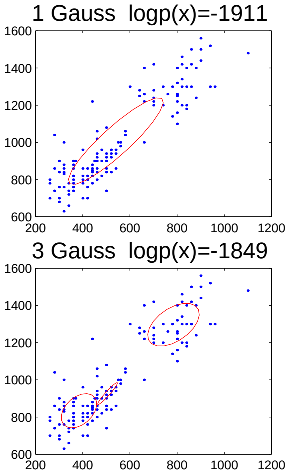
Parameters that maximize Q

$$
\begin{aligned}
\nu_{nk} &\equiv p(z_k \mid x_n, \theta_t) \\
\mu_k &= \frac{\sum_n \nu_{nk} x_n}{\sum_n \nu_{nk}} \\
\Sigma_k &= \frac{\sum_n \nu_{nk} (x_n - \mu_k)(x_n - \mu_k)^T}{\sum_n \nu_{nk}} \\
c_k &= \frac{1}{N} \sum_n \nu_{nk}
\end{aligned}
$$

- Each involves ν_{nk}, 'fuzzy membership' of x_n in Gaussian k
- Updated parameter is just sample average, weighted by fuzzy membership

GMM examples

Vowel data fit with different mixture counts



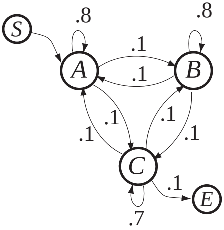
[Example. . .]

Outline

Markov models

- A (first order) Markov model is a finite-state system whose behavior depends only on the current state
- "The future is independent of the past, conditioned on the present"

e.g. generative Markov model

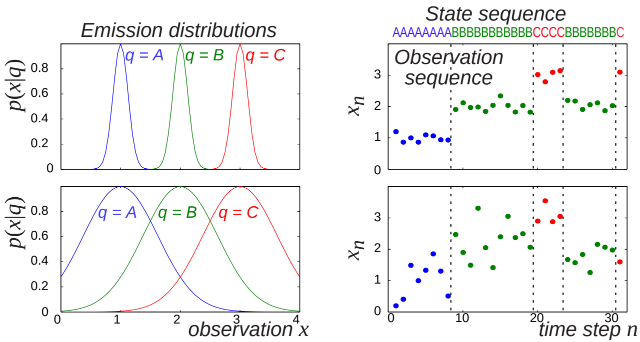


$p(q_{n+1} \mid q_n)$		q_{n+1}: S	A	B	C	E
	S	0	1	0	0	0
	A	0	.8	.1	.1	0
q_n	B	0	.1	.8	.1	0
	C	0	.1	.1	.7	.1
	E	0	0	0	0	1

S A A A A A A A A B B B B B B B B B C C C C B B B B B B C E

Hidden Markov models

- Markov model where state sequence $Q = \{q_n\}$ is not directly observable ('hidden')
- But, observations X *do* depend on Q
  - x_n is RV that depends only on current state $p(x_n \mid q_n)$



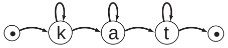
- can still tell *something* about state sequence...

(Generative) Markov models

HMM is specified by parameters Θ:

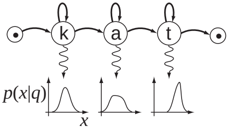
- states q^i

- transition probabilities a_{ij}



	k	a	t	•
•	1.0	0.0	0.0	0.0
k	0.9	0.1	0.0	0.0
a	0.0	0.9	0.1	0.0
t	0.0	0.0	0.9	0.1

- emission distributions $b_i(x)$



(+ initial state probabilities π_i)

$$a_{ij} \equiv p(q_n^j \mid q_{n-1}^i) \qquad b_i(x) \equiv p(x \mid q_i) \qquad \pi_i \equiv p(q_1^i)$$

Markov models for sequence recognition

- Independence of observations
  - observation x_n depends only on current state q_n

$$
\begin{aligned}
p(X \mid Q) &= p(x_1, x_2, \ldots x_N \mid q_1, q_2, \ldots q_N) \\
&= p(x_1 \mid q_1) p(x_2 \mid q_2) \cdots p(x_N \mid q_N) \\
&= \prod_{n=1}^{N} p(x_n \mid q_n) = \prod_{n=1}^{N} b_{q_n}(x_n)
\end{aligned}
$$

- Markov transitions
  - transition to next state q_{i+1} depends only on q_i

$$
\begin{aligned}
p(Q \mid M) &= p(q_1, q_2, \ldots \mid M) \\
&= p(q_N \mid q_{N-1} \ldots q_1) p(q_{N-1} \mid q_{N-2} \ldots q_1) p(q_2 \mid q_1) p(q_1) \\
&= p(q_N \mid q_{N-1}) p(q_{N-1} \mid q_{N-2}) p(q_2 \mid q_1) p(q_1) \\
&= p(q_1) \prod_{n=2}^{N} p(q_n \mid q_{n-1}) = \pi_{q_1} \prod_{n=2}^{N} a_{q_{n-1} q_n}
\end{aligned}
$$

Model-fit calculations

- From 'state-based modeling':

$$p(X \mid \Theta_j) = \sum_{\text{all } Q} p(X \mid Q, \Theta_j) p(Q \mid \Theta_j)$$

- For HMMs

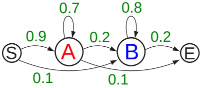
$$p(X \mid Q) = \prod_{n=1}^{N} b_{q_n}(x_n)$$

$$p(Q \mid M) = \pi_{q_1} \prod_{n=2}^{N} a_{q_{n-1} q_n}$$

- Hence, solve for $\hat{\Theta} = \operatorname{argmax}_{\Theta_j} p(\Theta_j \mid X)$
  - Using Bayes' rule to convert from $p(X \mid \Theta_j)$
- Sum over all Q???

Summing over all paths

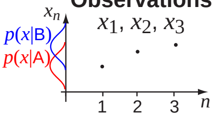
Model M_1



	S	A	B	E
S	•	0.9	0.1	•
A	•	0.7	0.2	0.1
B	•	•	0.8	0.2
E	•	•	•	1

Observations

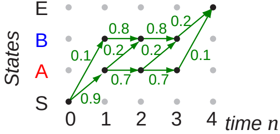
x_1, x_2, x_3



Observation likelihoods

$p(x\|q)$		x_1	x_2	x_3
q	A	2.5	0.2	0.1
	B	0.1	2.2	2.3

Paths



All possible 3-emission paths Q_k from S to E

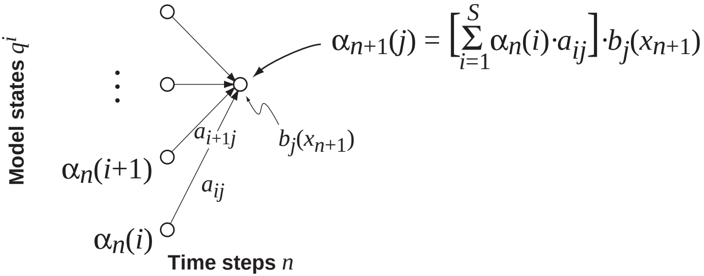
q_0	q_1	q_2	q_3	q_4	$p(Q \mid M) = \prod_n p(q_n \mid q_{n-1})$	$p(X \mid Q,M) = \prod_n p(x_n \mid q_n)$	$p(X,Q \mid M)$
S	A	A	A	E	.9 x .7 x .7 x .1 = **0.0441**	2.5 x 0.2 x 0.1 = 0.05	0.0022
S	A	A	B	E	.9 x .7 x .2 x .2 = 0.0252	2.5 x 0.2 x 2.3 = 1.15	0.0290
S	A	B	B	E	.9 x .2 x .8 x .2 = 0.0288	2.5 x 2.2 x 2.3 = 12.65	**0.3643**
S	B	B	B	E	.1 x .8 x .8 x .2 = 0.0128	0.1 x 2.2 x 2.3 = 0.506	0.0065
					Σ = 0.1109		$\boxed{\Sigma = p(X \mid M) = \mathbf{0.4020}}$

The 'forward recursion'

- Dynamic-programming-like technique to sum over all Q
- Define $\alpha_n(i)$ as the probability of getting to state q^i at time step n (by any path):

$$\alpha_n(i) = p(x_1, x_2, \ldots x_n, q_n = q^i) \equiv p(X_1^n, q_n^i)$$

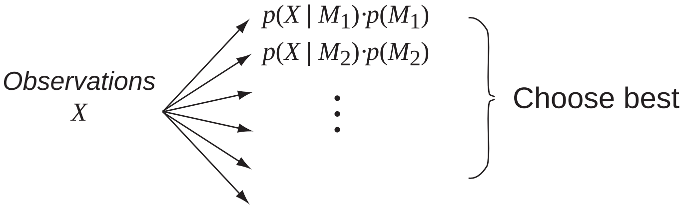
- $\alpha_{n+1}(j)$ can be calculated recursively:



Forward recursion (2)

- Initialize $\alpha_1(i) = \pi_i b_i(x_1)$
- Then total probability $p(X_1^N \mid \Theta) = \sum_{i=1}^{S} \alpha_N(i)$

$\longrightarrow$ Practical way to solve for $p(X \mid \Theta_j)$ and hence select the most probable model (recognition)



Optimal path

- May be interested in actual q_n assignments
  - which state was 'active' at each time frame

  e.g. phone labeling (for training?)
- Total probability is over *all* paths

  ... but can also solve for single *best* path, "Viterbi" state sequence
- Probability along best path to state q^j_{n+1}:

$$\hat{\alpha}_{n+1}(j) = \left[\max_i \{\hat{\alpha}_n(i) a_{ij}\}\right] b_j(x_{n+1})$$

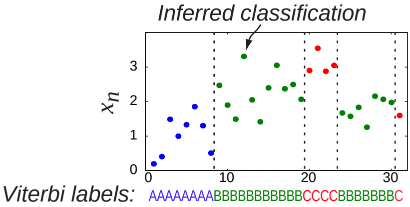
  - backtrack from final state to get best path
  - final probability is product only (no sum)

  $\rightarrow$ log-domain calculation is just summation
- Best path often dominates total probability

$$p(X \mid \Theta) \approx p(X, \hat{Q} \mid \Theta)$$

Interpreting the Viterbi path

- Viterbi path assigns each x_n to a state q^i
  - performing classification based on $b_i(x)$
  - ... at the same time applying transition constraints a_{ij}



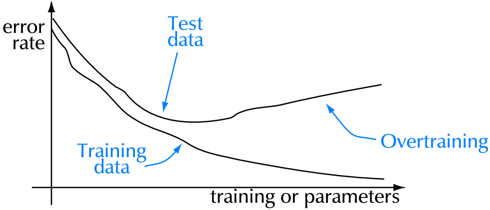
- Can be used for segmentation
  - train an HMM with 'garbage' and 'target' states
  - decode on new data to find 'targets', boundaries
- Can use for (heuristic) training

  e.g. forced alignment to bootstrap speech recognizer

  e.g. train classifiers based on labels...

Aside: Training and test data

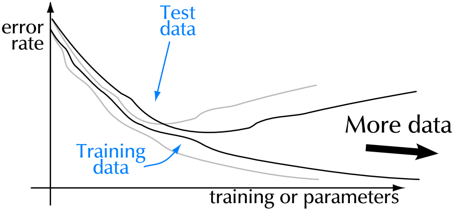
- A rich model can learn every training example (overtraining)



- But the goal is to classify new, unseen data
  - sometimes use ‘cross validation’ set to decide when to stop training
- For evaluation results to be meaningful:
  - don’t test with training data!
  - don’t train on test data (even indirectly. . .)

Aside (2): Model complexity

- More training data allows the use of larger models



- More model parameters create a better fit to the training data
  - more Gaussian mixture components
  - more HMM states
- For fixed training set size, there will be some optimal model size that avoids overtraining

Summary

- Classification is making discrete (hard) decisions
- Basis is comparison with known examples
  - explicitly or via a model
- Classification models
  - discriminative models, like SVMs, neural nets, boosters, directly learn posteriors $p(\theta_i \mid x)$
  - generative models, like Gaussians, GMMs, HMMs, model likelihoods $p(x \mid \theta)$
  - Bayes' rule lets us use generative models for classification
- EM allows parameter estimation even with some data missing

Parting thought

Is it wise to use generative models for discrimination or vice versa?

References

Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 459–468, Washington, DC, USA, 2006. IEEE Computer Society.

Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC, second edition, July 2003. ISBN 158488388X.

Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Jeff A. Bilmes. A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models, 1997.

Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, June 1998.