Lecture 3: Pattern Classification

1. The problem of classification
2. Linear and nonlinear classifiers
3. Probabilistic classification
4. Gaussians, mixtures and EM
5. Methodological remarks

Dan Ellis <dpwe@ee.columbia.edu>
http://www.ee.columbia.edu/~dpwe/e6820/

Columbia University Dept. of Electrical Engineering
Spring 2003
Pattern classification

- Classification means: finding categorical (discrete) labels for real-world (continuous) observations

- Why?
  - information extraction
  - conditional processing
  - detect exceptions
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
  - calculate (weighted) error rate

![Diagram showing Pols vowel formants: "u" (x) vs. "o" (o)]
Classification system parts

Sensor

signal

Pre-processing/segmentation

segment

Feature extraction

feature vector

Classification

class

Post-processing

- 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 practice**
  - representations make important aspects explicit
  - remove irrelevant information

- **Feature design incorporates ‘domain knowledge’**
  - more data → less need for ‘cleverness’?

- **Smaller ‘feature space’ (fewer dimensions)**
  → simpler models (fewer parameters)
  → less training data needed
  → faster training
Minimum distance classification

- Find closest match (nearest neighbor)
  \[ \text{min}[D(x, y_{\omega, i})] \]
  - choice of distance metric \( D(x, y) \) is important

- Find representative match (class prototypes)
  \[ \text{min}[D(x, z_{\omega})] \]
  - class data \( \{ y_{\omega, i} \} \) → class model \( z_{\omega} \)

Pols vowel formants: "u" (x), "o" (o), "a" (+)
2 Linear and nonlinear classifiers

- Minimum Euclidean distance is equivalent to a linear discriminant function:
  - examples \{ y_{\omega,i} \} \rightarrow \text{template } z_{\omega} = \text{mean}\{ y_{\omega,i} \}
  - observed feature vector \( x = [x_1 \ x_2 \ ...]^T \)
  - Euclidean distance metric
    \[
    D[x, y] = \sqrt{(x - y)^T (x - y)}
    \]
  - minimum distance rule:
    class \( i = \arg\min_D D[x, z_i] \)
    \[
    = \arg\min_D D^2[x, z_i] \]
  - i.e. choose class \( O \) over \( U \) if
    \[
    D^2[x, z_O] < D^2[x, z_U] \quad ...
    \]
Linear classifier (cont’d)

- Decision rule: Choose class O over U if:

\[ D^2[x, z_O] < D^2[x, z_U] \]

\[ (x - z_O)^T (x - z_O) < (x - z_U)^T (x - z_U) \]

\[ x^T x + z_O^T z_O - 2x^T z_O < x^T x + z_U^T z_U - 2x^T z_U \]

\[ 2x^T (z_O - z_U) > z_O^T z_O - z_U^T z_U \]

\[ (x - z_U)^T (z_O - z_U) > \frac{1}{2} (z_O - z_U)^T (z_O - z_U) \]

- i.e. distance from \( z_U \) to \( x \)

is less than half the distance to \( z_O \) ...
Linear classification boundaries

• Min-distance divides normal to class centers:

\[
\frac{1}{2} (z_O - z_U)^T (z_O - z_U) / |z_O - z_U| \\
(x - z_U)^T (z_O - z_U) / |z_O - z_U|
\]

• Scaling axes changes boundary:

Squeezed horizontally

Squeezed vertically
**Decision boundaries**

- **Linear functions give linear boundaries**
  - Planes and hyperplanes as dimensions increase

- **Linear boundaries can become curves under feature transformations**
  - E.g. a linear discriminant in $x' = [x_1^2 \ x_2^2 \ ...]^T$

- **What about more complex boundaries?**
  - I.e. if a linear discriminant is $\sum w_i x_i$
    how about a nonlinear function?
  - $F[\sum w_i x_i]$ changes only threshold space, but
  $\sum_j F[\sum_i w_{ij} x_i]$ offers more flexibility, and
  $F[\sum_j w_{jk} \cdot F[\sum_i w_{ij} x_i]]$ has even more ...
Neural networks

- **Sums** over **nonlinear** functions of sums → large range of decision surfaces

- e.g. Multi-layer perceptron (MLP) with 1 hidden layer:

\[ y_k = F\left[ \sum_j w_{jk} \cdot F\left[ \sum_i w_{ij} x_i \right] \right] \]

- **Problem is finding the weights** \( w_{ij} \) ...
  
  *(training)*
Back-propagation training

- Find $w_{ij}$ to minimize e.g. $E = \sum_n |o(x_n) - t_n|^2$
  for training set of patterns \{x_n\} and desired (target) outputs \{t_n\}

- Differentiate error with respect to weights:

  $$o_k = F[\sum_j w_{jk} \cdot h_j]$$

  $$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial \Sigma} \cdot \frac{\partial \Sigma}{\partial w_{jk}} (\sum_j w_{jk} h_j)$$

  $$= 2(o - t) \cdot F'[\Sigma] \cdot h_j$$

  $$w_{jk} = w_{jk} - \alpha \cdot \frac{\partial E}{\partial w_{jk}}$$

- i.e. gradient descent with learning rate $\alpha$
Neural net example

- 2 input units (normalized F1, F2)
- 5 hidden units, 3 output units (“U”, “O”, “A”)

![Neural network diagram]

- Sigmoid nonlinearity:

\[
F[x] = \frac{1}{1 + e^{-x}} \quad \Rightarrow \quad \frac{dF}{dx} = F(1 - F)
\]
Neural net training

2:5:3 net: MS error by training epoch

Contours @ 10 iterations
Contours @ 100 iterations

example...
3 Statistical Interpretation

- Observations are random variables whose distribution depends on the class:

  ![](image)

  - Source distributions $p(x|\omega_i)$
    - reflect variability in feature
    - reflect noise in observation
    - generally have to be estimated from data (rather than known in advance)
Random Variables review

- Random vars have joint distributions (pdf’s):

- marginal \( p(x) = \int p(x, y)dy \)

- covariance \( \Sigma = E[(\mathbf{o} - \mu)(\mathbf{o} - \mu)^T] = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{bmatrix} \)
Conditional probability

- Knowing one value in a joint distribution constrains the remainder:

\[ p(x, y) = p(x|y) \cdot p(y) \]

\[ \Rightarrow p(y|x) = \frac{p(x|y) \cdot p(y)}{p(x)} \]

- Bayes’ rule: can reverse conditioning given priors/marginal
  - either term can be discrete or continuous
Priors and posteriors

- Bayesian inference can be interpreted as updating prior beliefs with new information, \( x \):

\[
P r(\omega_i) \cdot \frac{p(x|\omega_i)}{\sum_j p(x|\omega_j) \cdot Pr(\omega_j)} = Pr(\omega_i|x)
\]

‘Evidence’ = \( p(x) \)

- Posterior is prior scaled by likelihood & normalized by evidence (so \( \sum \text{(posteriors)} = 1 \))

- Objection: priors are often unknown
  - but omitting them amounts to assuming they are all equal
Bayesian (MAP) classifier

- Minimize the probability of error by choosing *maximum a posteriori* (MAP) class:
  \[ \hat{\omega} = \underset{\omega_i}{\text{argmax}} \ Pr(\omega_i \mid x) \]

- Intuitively right - choose most probable class in light of the observation

- Formally, expected probability of error,
  \[ E[Pr(err)] = \int p(x) \cdot Pr(err \mid x) dx \]
  \[ = \sum_i \int_{X_{\hat{\omega}_i}} p(x)(1 - Pr(\omega_i \mid x)) dx \]

where \( X_{\hat{\omega}_i} \) is the region of \( X \) where \( \omega_i \) is chosen

.. but \( Pr(\omega_i \mid x) \) is largest in that region,
so \( Pr(err) \) is minimized.
Practical implementation

- **Optimal classifier** is \( \hat{\omega} = \arg\max_{\omega} Pr(\omega_i|x) \)

  but we don’t know \( Pr(\omega_i|x) \)

- Can model directly e.g. train a neural net to map from inputs \( x \) to a set of outputs \( Pr(\omega_i) \)
  - a **discriminative** model

- Often easier to model conditional distributions \( p(x|\omega_i) \) then use Bayes’ rule to find MAP class

![Diagram](diagram.png)

Labeled training examples \( \{x_n, \omega_{x_n}\} \) → Sort according to class → Estimate conditional pdf for class \( \omega_1 \) → ... → \( p(x|\omega_1) \)
Likelihood models

- Given models for each distribution $p(x | \omega_i)$, the search for $\hat{\omega} = \arg\max_{\omega_i} Pr(\omega_i | x)$ becomes $\arg\max_{\omega_i} \frac{p(x | \omega_i) \cdot Pr(\omega_i)}{\sum_j p(x | \omega_j) \cdot Pr(\omega_j)}$

  but denominator $= p(x)$ is the same over all $\omega_i$

  hence $\hat{\omega} = \arg\max_{\omega_i} p(x | \omega_i) \cdot Pr(\omega_i)$

  or even $\arg\max_{\omega_i} \left[ \log p(x | \omega_i) + \log Pr(\omega_i) \right]$

- Choose parameters to maximize data likelihood, $p(x_{train} | \omega_i)$
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 a given feature set regardless of classification scheme
Gaussian models

- Easiest way to model distributions is via parametric model
  - assume known form, estimate a few parameters

- Gaussian model is simple & useful:

\[
p(x|\omega_i) = \frac{1}{\sqrt{2\pi\sigma_i}} \cdot \exp\left[-\frac{1}{2}\left(\frac{x - \mu_i}{\sigma_i}\right)^2\right]
\]

normalization to make it sum to 1

- Parameters mean \(\mu_i\) and variance \(\sigma_i^2\) \(\rightarrow\) fit

\[
\begin{array}{c}
P_M \\
P_M \\
e^{1/2} \\
mu_i \\
\sigma_i \\
\end{array} \rightarrow \text{x}
\]
Gaussians in \(d\) dimensions:

\[
p(x | \omega_i) = \frac{1}{(\sqrt{2\pi})^d |\Sigma_i|^{1/2}} \cdot \exp\left[ -\frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) \right]
\]

- Described by \(d\) dimensional mean vector \(\mu_i\) and \(d \times d\) covariance matrix \(\Sigma_i\)

- Classify by maximizing log likelihood i.e.

\[
\arg\max_{\omega_i} \left[ -\frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) - \frac{1}{2} \log |\Sigma_i| + \log Pr(\omega_i) \right]
\]
Multivariate Gaussian covariance

- Eigen analysis of inverse of covariance gives
  \[ \Sigma_i^{-1} = (SV)^T (SV) \text{ with diagonal } S = \Lambda^{1/2} \]

→ exponent is \[ -\frac{1}{2} (x - \mu_i)^T (SV)^T (SV)(x - \mu_i) \]

i.e. \((SV)\) transforms \(x\) into uncorrelated, normalized-variance space

- \((x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) = \text{Mahalanobis distance}^2\)
  - Euclidean distance in normalized space

example...
Gaussian Mixture models (GMMs)

- Single Gaussians cannot model
  - distributions with multiple modes
  - distributions with nonlinear correlation

- What about a weighted sum?

\[ p(x) \approx \sum_k c_k p(x|m_k) \]

where \( \{c_k\} \) is a set of weights and

\( \{p(x|m_k)\} \) is a set of Gaussian components

- can fit anything given enough components

- Interpretation: each observation is generated by one of the Gaussians, chosen at random

with priors \( c_k = Pr(m_k) \)
Gaussian mixtures (2)

- e.g. nonlinear correlation:

- Problem: finding $c_k$ and $m_k$ parameters
  - easy if we knew which $m_k$ generated each $x$
Expectation-maximization (EM)

- General procedure for estimating a model when some parameters are unknown
  - e.g. which component generated a value in a Gaussian mixture model

- Procedure:
  Iteratively update fixed model parameters $\Theta$ to maximize $Q$=expected value of log-probability of known training data $x_{trn}$ and unknown parameters $u$:

$$Q(\Theta, \Theta_{old}) = \sum_u Pr(u | x_{trn}, \Theta_{old}) \log p(u, x_{trn} | \Theta)$$

  - iterative because $Pr(u | x_{trn}, \Theta_{old})$ changes each time we change $\Theta$
  - can prove this increases data likelihood, hence maximum-likelihood (ML) model
  - local optimum - depends on initialization
Fitting GMMs with EM

• Want to find component Gaussians

\[ m_k = \{\mu_k, \Sigma_k\} \] plus weights \( c_k = Pr(m_k) \)

to maximize likelihood of training data \( x_{trn} \)

• If we could assign each \( x \) to a particular \( m_k \), estimation would be direct

• Hence, treat \( k \)'s (mixture indices) as unknown, form \( Q = E[\log p(x, k | \Theta)] \), differentiate with respect to model parameters → equations for \( \mu_k, \Sigma_k \) and \( c_k \) to maximize \( Q \)...
GMM EM update equations:

- Solve for maximizing $Q$:
  
  $\mu_k = \frac{\sum_n p(k|x_n, \Theta) \cdot x_n}{\sum_n p(k|x_n, \Theta)}$

  $\Sigma_k = \frac{\sum_n p(k|x_n, \Theta)(x_n - \mu_k)(x_n - \mu_k)^T}{\sum_n p(k|x_n, \Theta)}$

  $c_k = \frac{1}{N} \sum_n p(k|x_n, \Theta)$

- Each involves $p(k|x_n, \Theta)$, ‘fuzzy membership’ of $x_n$ to Gaussian $k$

- Parameter is just sample average, weighted by ‘fuzzy membership’
GMM examples

- Vowel data fit with different mixture counts:

1 Gauss $\log p(x) = -1911$

2 Gauss $\log p(x) = -1864$

3 Gauss $\log p(x) = -1849$

4 Gauss $\log p(x) = -1840$

example...
Methodological Remarks:
1: Training and test data

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

- But, goal is to classify new, unseen data i.e. generalization
  - 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...)
Model complexity

- More model parameters $\rightarrow$ better fit
  - more Gaussian mixture components
  - more MLP hidden units

- More training data $\rightarrow$ can use larger models

- For best generalization (no overfitting), there will be some optimal model size:

![Graph showing word error rate (WER) vs. model size and training data](image-url)

- Optimal parameter/data ratio

Constant training time

Word Error Rate

More parameters

More training data

Optimal parameter/data ratio

WER for PLP12N-8k nets vs. net size & training data
Model combination

- No single model is always ‘best’
  *But* different models have different strengths

→ Benefits from *combining* several models e.g.

\[ Pr(\omega) = \sum_k w_k \cdot Pr(\omega | m_k) \]

where each \( Pr(\omega | m_k) \) comes from a different model / feature set

and \( w_k = Pr(m_k) \) estimates probability that model \( k \) is correct

- Should be able to incorporate into one model, but may be easier to combine in practice

- Trick is to have good estimates for \( w_k \):
  - static, from training data
  - dynamic, from test data...
Summary

• Classification is making hard decisions
• Basis is comparison with known examples
  - explicitly, or via a model
• Probabilistic interpretation for principled decisions
• Ways to build models:
  - neural nets can learn posteriors $Pr(ω|x)$
  - Gaussians can model pdfs $p(x|ω)$
• EM allows estimation of complex models

Parting thought:

• How do you know what’s going on?