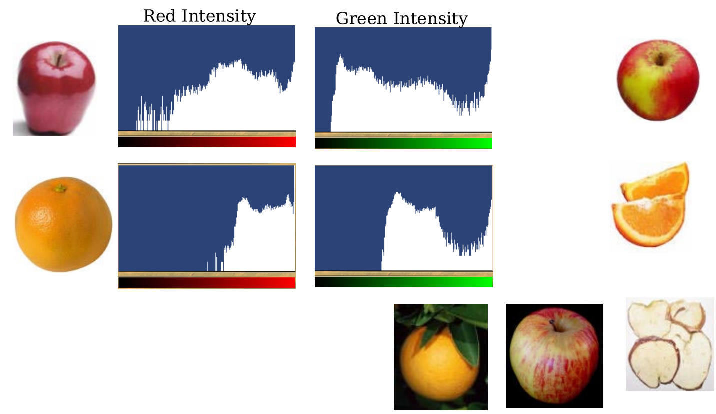
Statistical Pattern Recognition

15-486/782: Artificial Neural Networks
David S. Touretzky

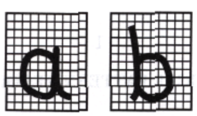
Fall 2006

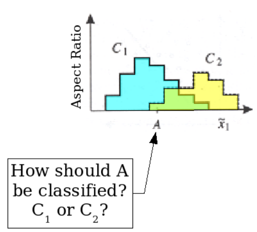
Reading: Bishop chapter 1

Telling “Apples From Oranges”



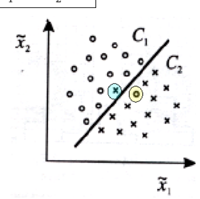
Telling A's from B's





- Single features, such as the character aspect ratio (height/width), may not be adequate to accurately discriminate classes.

- Using two features can give better accuracy. (Still not perfect.)



Feature-Based Pattern Recognition

A “feature” is some quantity computed from the raw input values describing the instance.

Goal: learn the most likely class for each combination of feature values.

Use only a small number of features?

- Cheap and fast to implement.
- Small parameter space: easy to train.
- But accuracy may be poor: can't discriminate well.

How Many Features to Use?

Lots of features?

- In theory, should discriminate well.
- But expensive to implement (slow).
- "Curse of dimensionality": need lots of training examples to cover the feature space.

Choose a few maximally informative features:

- Best strategy.
- But it may be hard to find good features.
- Pre-processing the input data can help.

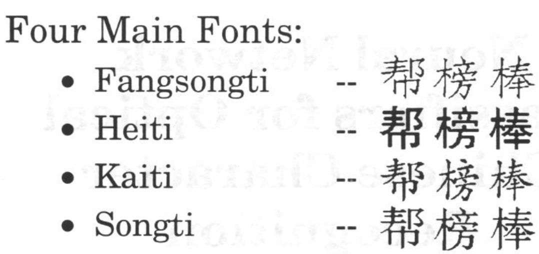
5

Pre-Processing

- Replace raw input data with a lower-dimensional vector of specialized features.
- Features should capture important structure in the input while eliminating irrelevant detail.
- Examples:
  - edge detection for character recognition.
  - formant detection for speech recognition
- Construction of good feature sets is an art.

6

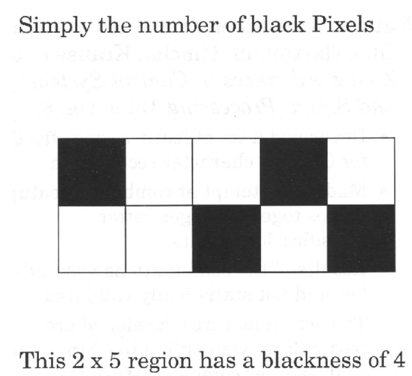
Example: Chinese OCR



Let's look at a feature set from Suchenwirth et al. (1989) *Advances in Control Systems and Signal Processing*, vol. 8.

7

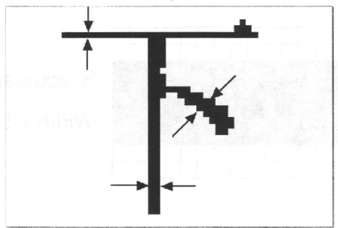
Blackness



8

Stroke Width

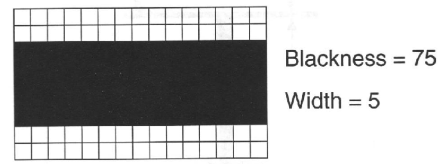
Average width of strokes, examining 4 main directions



9

Total Stroke Length

Divide the blackness by the average stroke width to obtain an estimate for the total stroke length

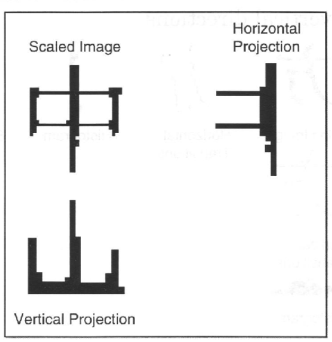


$$\frac{\text{Blackness}}{\text{Width}} = 15 \text{ (the length)}$$

10

Horizontal & Vertical Projection Profiles

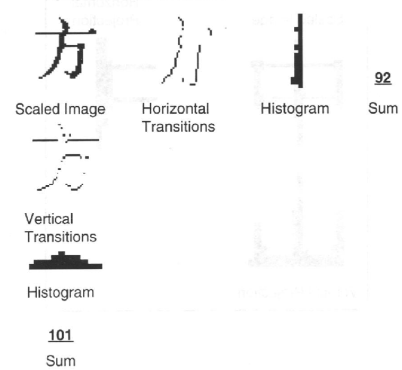
Sum of black pixels along the horizontal and vertical directions



11

Horizontal & Vertical Transitions

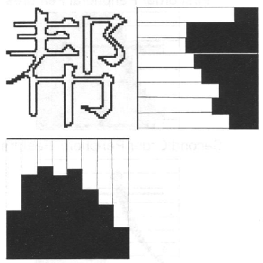
Sum of the number of black to white transitions in both horizontal and vertical directions



12

Stroke Density

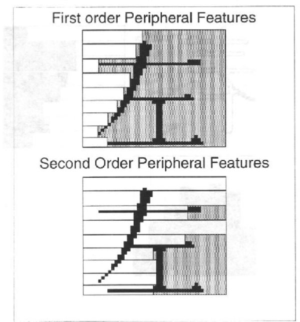
Sub-divide the image into regions, for each of the 4 major directions. For each region, sum the number of background to foreground transitions



13

Peripheral Features

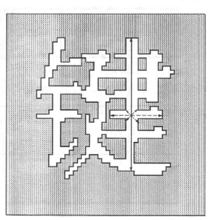
Distance from edge of image to the first (or second) black transition for a sub-divided image (done in all four NSEW directions)



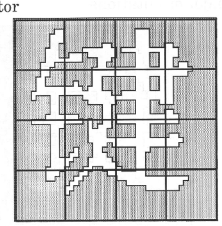
14

Local Direction Contributivity

Start by measuring the length of the line passing through a foreground pixel and ending once it reaches the background. Do this using four lines in the four major orientations.



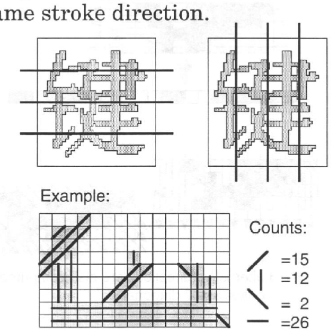
Split the image into segments and average the distances obtained □for each foreground pixel, once per direction. An (NxN) gridding yields a $4N^2$ feature vector



15

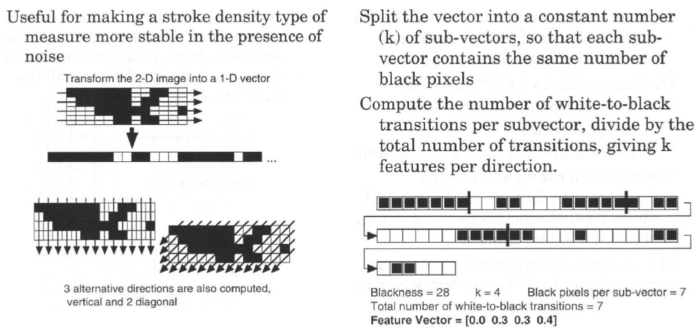
Stroke Proportion

Subdivide the image into four stripes for both horizontal and vertical directions. For each stripe, count the number of foreground pixels which are all the same stroke direction.



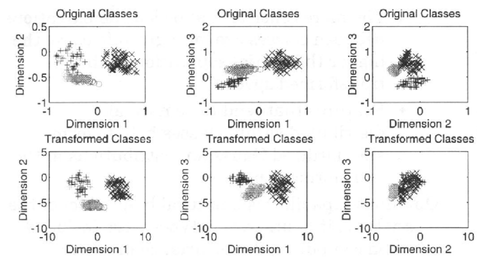
16

Black Jump Distribution in Balanced Subvectors



Reducing the Feature Space

- Original feature space was 405-element vector.
- Use Karhunen-Loeve transform (principal component analysis) to reduce to a 100-dimensional space. Reduce within-class variance; increase between-class.



- Feed the result to a neural network classifier.

The Curse of Dimensionality

Assume d features (dimensions), with M values per dimension.

Label each bin with its class.

Total number of bins = M^d.

Growth is exponential in d: bad news.

Can't label all bins in high-dimensional problems, like Chinese OCR (100 dimensional feature space.)

Too many bins ⇒ not enough training data.

Can't learn (or even write down) a separate class for every possible combination of features.

Classification Functions

Explicitly assigning a class to each point in the feature space is too expensive in high-dimensional spaces.

Instead, write a classification function to do it!

$$f : X^n \rightarrow C$$

What should this function look like?

- Could be linear (a perceptron)
- Higher order polynomial
- Something else (e.g., a neural network)

Classification via Regression

Classifiers map points in feature space X^n to discrete classes C_i.

Regression is a function approximation technique.

Suppose we want to approximate $F(\mathbf{x})$ by a function $f(\mathbf{x};\mathbf{w})$, where $\mathbf{w}$ is a vector of parameters.

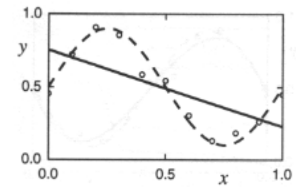
Regression problem: find $\mathbf{w}^*$ that minimizes the error of $f(\mathbf{x};\mathbf{w}^*)$ as an estimator of $F(\mathbf{x})$.

The LMS learning rule uses sum-squared error; it does linear regression.

The perceptron rule does not do regression, but it does search weight space to train a linear classifier.

Linear Regression

$$y = 0.5 + 0.4\sin(2\pi x) + \eta \qquad where\ \eta \in N(0, 0.05)$$



LMS fits a line (or plane, or hyperplane) to a dataset. The fit here is poor.

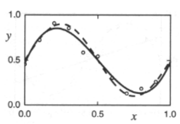
Why not fit a higher order polynomial?

Higher-Order Polynomials (Order M)

$$y(x) = w_0 + w_1 x + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Minimize sum-squared error over N training points x_i:

$$E = \frac{1}{2}\sum_{i=1}^{N}\left[y(x_i;\mathbf{w}) - t_i\right]^2$$



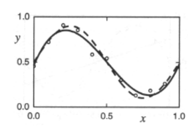
M=3: cubic polynomial provides a reasonably good fit.

$$\mathbf{w}^* = \langle w_0^*, w_1^*, w_2^*, w_3^* \rangle$$

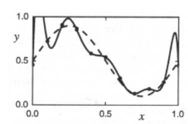
Generalization: Use a Test Set to Measure the RMS Error

$$RMS\ Error = \sqrt{\frac{1}{T}\sum_{i=1}^{T}\left[y(x_i;\mathbf{w}) - t_i\right]^2}$$

Independent of the number of test set points T.

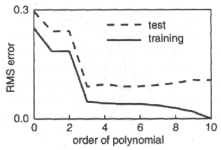


M=3 (cubic poly) gives reasonably low error on both training and test sets.



M=10 hits all 11 training points spot-on. But performance on the test set is poor. Why? Overfitting: we're fitting the noise.

Optimal Model



Test set performance is best for M=3 (cubic poly).

Higher order polynomials fit the training data better.

But performance on the test set can get worse, if the model is overfitting.

Generalization is usually what we care about.

Using Regression to Train a Binary Classifier

Let $F(\mathbf{x}) \in C \equiv \{0,\ 1\}$.

Find $\hat{\mathbf{w}}$ *that makes* $f(\mathbf{x};\hat{\mathbf{w}})$ *the best estimator of* $F(\mathbf{x})$.

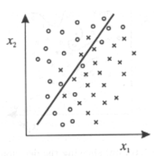
Turn an estimator into a classifier: map f(**x**;**w**) into C.

For binary classification problems, we can use a threshold function to do this.

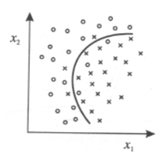
$$\textit{Regression:} \quad y = f(\mathbf{x};\mathbf{w})$$

$$\textit{Classification:} \quad y - f(\mathbf{x};\mathbf{w}) > 0$$

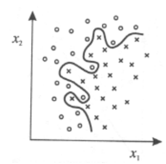
Training a Classifier



Linear classifier makes a lot of errors.



Quadratic classifier does pretty good job.



Higher-order polynomial gets all points right. But...?

Regularization Smooths Polynomials by Penalizing "Bendiness"

$$E = \frac{1}{2}\sum_i (y_i - t_i)^2$$

$$\Omega = \frac{1}{2}\int \left(\frac{d^2 y}{dx^2}\right)^2 dx$$

$$\tilde{E} = E + \nu\Omega$$

Constant ν *determined empirically.*

Quadratic Classifiers

Quadratic in n variables:

$$y = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j \leq i}^{n} w_i x_i x_j$$

Still a linear model: it's a linear function of the weights.

Think of the quadratic terms as just extra features, with weights w_{ij}.

Building a Quadratic Classifier

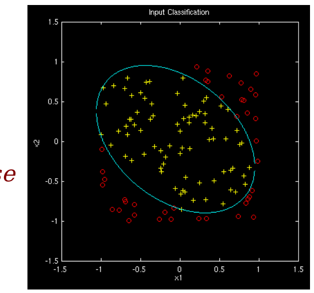
Assume 2D input space: x_1 *and* x_2

$$y = w_1 + w_2 x_1 + w_3 x_2 + w_4 x_1^2 + w_5 x_2^2 + w_6 x_1 x_2$$

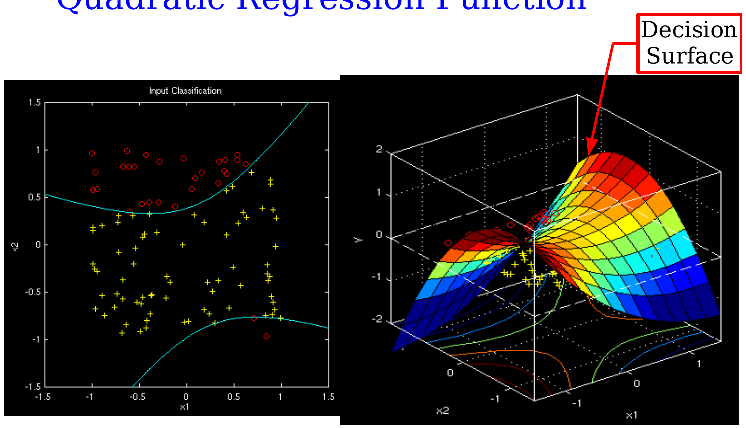
Decision boundary: $y > 0$

Training: LMS.

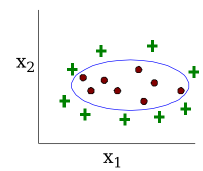
Shape of decision surface?
parabola, hyperbola, ellipse



Quadratic Regression Function



Plotting the Decision Boundary



$$w_1 + w_2 x_1 + w_3 x_2 + w_4 x_1^2 + w_5 x_2^2 + w_6 x_1 x_2 = 0$$

const. linear quadratic

$$w_5 x_2^2 + (w_3 + w_6 x_1) x_2 + (w_1 + w_2 x_1 + w_4 x_1^2) = 0$$

a b c

$$a x_2^2 + b x_2 + c = 0$$

$$x_2 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Note: up to 2 real roots.

What's Better Than a Polynomial Classifier?

Multilayer perceptrons!

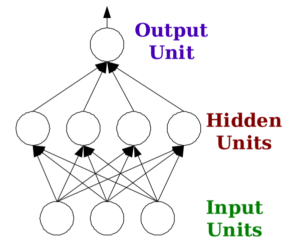
Polynomial classifier of order $k \leq d$ with d-dimensional input needs how many terms?

$$\sum_{i=0}^{k} \frac{d^i}{i!}$$ *this is exponential in k*

Neural nets (MLPs) can do the job with far fewer parameters.

But there's a price to pay: nonlinearity, local minima ...

Why MLPs Are Better

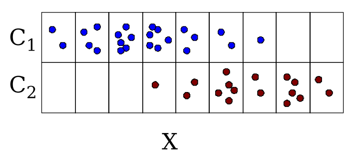


- Barron (1993): sum-squared error decreases as $O(1/M)$, where M = # of nonlinear hidden units.
- This is true independent of the number of inputs!

For polynomial approximators, error falls as $O(1/M^{2/d})$, where d is the dimensionality of the input. (Assumes linear combination of fixed polynomial terms.)

For large d, neural nets win big!

Basics of Probability



Classes: C_1 *and* C_2
Feature values: $X = \{x_1, \cdots, x_9\}$

Prior probability:	$P(C_k)$
Joint probability:	$P(C_k, X_l)$
Conditional probability:	$P(X_l \mid C_k)$
Posterior probability:	$P(C_k \mid X_l)$
Normalization const.	$P(X_l)$

Bayes Theorem

$$P(C_k, X_l) = P(C_k \mid X_l) \cdot P(X_l) = P(X_l \mid C_k) \cdot P(C_k)$$

Bayes Theorem:

$$P(C_k \mid X_l) = \frac{P(X_l \mid C_k) \cdot P(C_k)}{P(X_l)}$$

This will be on the midterm.

Why Use Bayes Rule?

- Tumor detection task:
  - 99% of samples are normal
  - 1% abonormal
- Training set: 50% normal, 50% abnormal
- Use training set to estimate $P(X_i|C_k)$
- Class priors: $P(C_1) = 0.99$, $P(C_2) = 0.01$
- Bayes' rule gives the correct posterior probability:

$$P(C_k|X_i) = \frac{P(X_i|C_k) \cdot P(C_k)}{P(X_i)}$$

Sample Probability Problem

1) One third of Americans believe Elvis is alive.

Elvis alive: 1/3		
Elvis dead: 2/3		

Sample Probability Problem

1) One third of Americans believe Elvis is alive.

2) Seven eights of these believers drive domestic cars.

	Domestic Car	Foreign Car
Elvis alive: 1/3	$\frac{1}{3} \cdot \frac{7}{8} = \frac{7}{24}$	
Elvis dead: 2/3		

Sample Probability Problem

1) One third of Americans believe Elvis is alive.

2) Seven eights of these believers drive domestic cars.

3) Three fourths of all the cars in the US today were manufactured domestically.

	Domestic Car 3/4	Foreign Car 1/4
Elvis alive: 1/3	$\frac{1}{3} \cdot \frac{7}{8} = \frac{7}{24}$	$\frac{1}{24}$
Elvis dead: 2/3	$\frac{11}{24}$	$\frac{5}{24}$

Sample Probability Problem

1) One third of Americans believe Elvis is alive.

2) Seven eights of these believers drive domestic cars.

3) Three fourths of all the cars in the US today were manufactured domestically.

The highway patrol stops a foreign-made car. What is the probability that the driver believes Elvis to be dead?

	Domestic Car 3/4	Foreign Car 1/4
Elvis alive: 1/3	$\frac{1}{3} \cdot \frac{7}{8} = \frac{7}{24}$	$\frac{1}{24}$
Elvis dead: 2/3	$\frac{11}{24}$	$\frac{5}{24}$

$$\begin{aligned} P(dead|foreign) &= \frac{P(dead,foreign)}{P(alive,foreign)+P(dead,foreign)} \\ &= \frac{5}{24} \,/\, \left(\frac{1}{24}+\frac{5}{24}\right) = \frac{5}{6} \end{aligned}$$

$$\begin{aligned} P(foreign|dead) &= \frac{P(dead|foreign) \cdot P(foreign)}{P(dead)} \\ &= \frac{\frac{5}{6} \cdot \frac{1}{4}}{\frac{2}{3}} = \frac{5}{16} \end{aligned}$$

Bayesian Classifiers

Put x in class C_k *if* $P(C_k|x) > P(C_j|x)$ *for all* $j \neq k$

Equivalently, by Bayes' Rule,
$P(x|C_k) \cdot P(C_k) > P(x|C_j) \cdot P(C_j)$ *for* $j \neq k$

Why is this the right thing to do?

Consider a two-class problem:

Class C_1 has decision region R_1

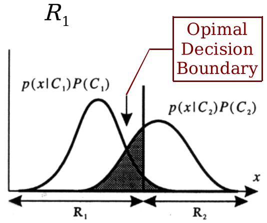
Class C_2 has decision region R_2

What is the probability of misclassifiying x?

Likelihood of Misclassification

$$\begin{aligned} P(error) &= P(x \in R_2, C_1) + P(x \in R_1, C_2) \\ &= P(x \in R_2|C_1) \cdot P(C_1) + P(x \in R_1|C_2) \cdot P(C_2) \\ &= \int_{R_2} p(x|C_1) \cdot P(C_1)\, dx + \int_{R_1} p(x|C_2) \cdot P(C_2)\, dx \end{aligned}$$

So if $p(x|C_1) \cdot P(C_1) > p(x|C_2) \cdot P(C_2)$, *we can reduce the error contribution by putting x in* R_1 *rather than* R_2.



Good News

A properly trained neural network

will approximate

the Bayesian posterior probabilities

$P(C_k \mid \mathbf{x})$

Discriminant Functions Do Pattern Classification

Define $y_k(x) \approx P(C_k|x)$ *(discriminant function)*
Could train a separate function approximator for each y_k

Class of x *is:* $\text{argmax}_k \; y_k(x)$
Decision boundary between C_j *and* C_k *is at:*

$$y_j(x) = y_k(x)$$

Special trick for two-class problems: define

$$y(x) = y_1(x) - y_2(x)$$

Assign x to C_1 *if* $y(x) > 0$.
One function discriminates two classes.