Image Analysis - Lecture 9

Classification

Introduction

- Contents
- Basic machine learning concepts and applications
- Linear discriminants and Support vector machines

Classification

- Optimal statistical classification
- Probability
- Nearest neighbor classification

Dimensionality reduction

- Principal component analysis

Other classifiers

- Artificial neural networks

Some classification problems

- OCR: Given an image, segment the characters and classify them (a,b,c, etc).
- Cell analysis: Given an image, segment the cells and classify them (white blood cell, red blood cells, etc)
- Grain analysis: Given an image of grain, segment the grains and classify them (wheat, rice, stone, rotten, etc)
- Handwriting recognition: Given handwritten data, segment the text in individual characters and classify them (a,b,c, etc),
- Diagnostic support: Given medical data (images, journal data) classify patient (healthy, sickness1, sickness2, etc)
- Face detection: Given an image, classify it as (face, not face)

Common strategy

All of these classification problems have in common:

- segmentation
- extract features
- data - **x**
- assign data to a number of classes

One would like to determine a class for every possible feature vector.

Feature vector

Here we will assume that the features are represented as a column vector

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

One would like to compare the feature vector **x** with those that one usually gets with a number of classes $\omega_1, \ldots, \omega_M$ where M denotes the number of classes.

Typical system: Image - filtering - segmentation - features - classification

Example: Plant identification

By measuring the length x_1 and width x_2 of the leaf of a plant, one would like to classify the plant in three classes

- Iris virginica - ω_1
- Iris versicolor - ω_2
- Iris setosa - ω_3.

How should one construct a function f that given a features vector $\mathbf{x}$ determines, which class it belongs to?

How can one determine if a recognition function f is good?

Are there any limits to the performance that one can get?

Classification

Idea: Construct M functions $d_i : \mathbf{x} \to \mathbb{R}$. (Think of these as distance or error functions)

Classify an element $\mathbf{x}$ as class i if

$$d_i(\mathbf{x}) < d_j(\mathbf{x}), \forall j \neq i$$

The borders between different classes ω_i and ω_j is given by

$$d_i(\mathbf{x}) = d_j(\mathbf{x})$$

Construction of error functions

How can the functions d_i be constructed?

- Linear discriminants, support vector machines
- Optimal statistical classification
- Nearest Neighbor classification
- Artificial Neural Networks

Optimal statistical classification

Assume that one can calculate the probability that the vector **x** comes from class ω_j:

$$P(\omega_j|\mathbf{x})$$

If our method classifies **x** as class ω_j when it really is ω_i we obtain a loss L_{ij}.

Discussion: What is the reason for having different losses for L_{12} and L_{21}?

The mean loss when classifying **x** as ω_j is then

$$r_j(\mathbf{x}) = \sum_{k=1}^{M} L_{kj} p(\omega_k|\mathbf{x})$$

Bayesian classification

Using Bayes rule

$$p(a|b) = \frac{p(a)p(b|a)}{p(b)}$$

one may rewrite r_j as

$$r_j(\mathbf{x}) = \frac{1}{p(\mathbf{x})} \sum_{k=1}^{M} L_{kj} p(\mathbf{x}|\omega_k) p(\omega_k)$$

Bayes classification (cont.)

Since $p(\mathbf{x})$ is positive and common for all r_j it does not affect the comparison of different r_j.

Bayes classification:
Classify $\mathbf{x}$ as ω_i if

$$r_i(\mathbf{x}) < r_j(\mathbf{x}) \quad \text{for all} \quad j \neq i \ .$$

Choice of L_{ij}.

If nothing else is known about the application it is common to choose

$$L_{ij} = \begin{cases} 0, & i = j \\ 1, & i \neq j \end{cases}$$

For this loss one may use

$$d_j(\mathbf{x}) = p(\mathbf{x}|\omega_j)p(\omega_j)$$

as decision function.

Example with discrete variables

$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ has prob $\frac{1}{4}$; $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ has prob $\frac{1}{4}$; $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ has prob $\frac{1}{2}$

Assume that there is 10 percent chance for errors in a pixel.
Assume independent errors.

What is the optimal classification of $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$?

Solution

$$p(\omega_1|x) = \frac{p(x|\omega_1)p(\omega_1)}{p(x)} \sim p(x|\omega_1)p(\omega_1) = 0.9^3 0.1^1 * 0.25 \approx 0.018$$

$$p(\omega_2|x) = \frac{p(x|\omega_2)p(\omega_2)}{p(x)} \sim p(x|\omega_2)p(\omega_2) = 0.9^1 0.1^3 * 0.25 \approx 0.0002$$

$$p(\omega_3|x) = \frac{p(x|\omega_3)p(\omega_3)}{p(x)} \sim p(x|\omega_3)p(\omega_3) = 0.9^2 0.1^2 * 0.50 \approx 0.004$$

The first image is most probable!

The total probability

In this case it is possible to calculate the total probability

$$p(x) = p(x|\omega_1)p(\omega_1) + p(x|\omega_2)p(\omega_2) + p(x|\omega_3)p(\omega_3) = 0.022$$

This makes it possible to calculate the probabilities for each class

$$p(\omega_1|x) = \frac{p(x|\omega_1)p(\omega_1)}{p(x)} = \frac{0.018}{0.022} = 0.81$$

$$p(\omega_2|x) = \frac{p(x|\omega_2)p(\omega_2)}{p(x)} = \frac{2.2510^{-4}}{0.022} = 0.01$$

$$p(\omega_3|x) = \frac{p(x|\omega_3)p(\omega_3)}{p(x)} = \frac{0.004}{0.022} = 0.18$$

Continuous probability distributions

It is common to model the features as continuous variables.

Gaussian distributions are popular models

$$P(X|\omega_1) = N(m_1, \Sigma_1)$$

Standard classifier

A so called 'plug-in' classifier consists of two steps:

1. Training.
   Estimate parameters, e.g. m_i, Σ_i from training data.
2. Classification.
   Assuming that the model is correct and assuming that the estimates are exact. Use Bayes formula to estimate $p(\omega_i|x)$ and classify according to minimum loss or maximum a posteriori.

Histogram

If it is possible to discretise the variables and if there is a large amount of training data, one may estimate the frequency functions using histograms.

Nearest Neighbor classification

A simple (but surprisingly effective) method is to save the entire training set:

$$T = \{(x_1, y_1), \ldots, (x_N, y_N)\},$$

where $x_i \in R^d$ and $y_i \in \{1, \ldots, M\}$.

During classification one compares the distance between the feature vector x with the ones in the training set $(x_1, \ldots, x_n)$, i.e. solves

$$k = \text{argmin}_i \, d(x_i, x)$$

and put

$$y = y_k .$$

(k,l)-nearest neighbor classification

A generalization of the above method is to find the k nearest neighbors to the vector x.

Classify it the class y with the most representatives among these k if this number is above l.

Otherwise classify it as 'unknown'.

Choosing features

In image analysis, there is often much information.

E.g: 1600×1200 color image has $1600 * 1200 * 3 = 5\ 760\ 000$ elements, i.e.
x is a $5\ 760\ 000 \times 1$ vector.

For practical reasons one has to select a subset of these features.

This could be thought of a part of the classification problem, but is often too difficult to handle with general methods (of today).

Choosing features (cont.)

Choosing features is difficult. Some ideas are:

- Try single features in x. How good are they?
- Boosting
- Principal components analysis
- Intuition and knowledge about the problem

The two latter are used quite often.

Invariant properties

Sometimes there are great variations in the features vector **x** that depends on object position, lighting, etc.

The feature vectors of a class could then form large spread out clouds in $\mathbb{R}^n$, making it difficult to do good classification.

Example

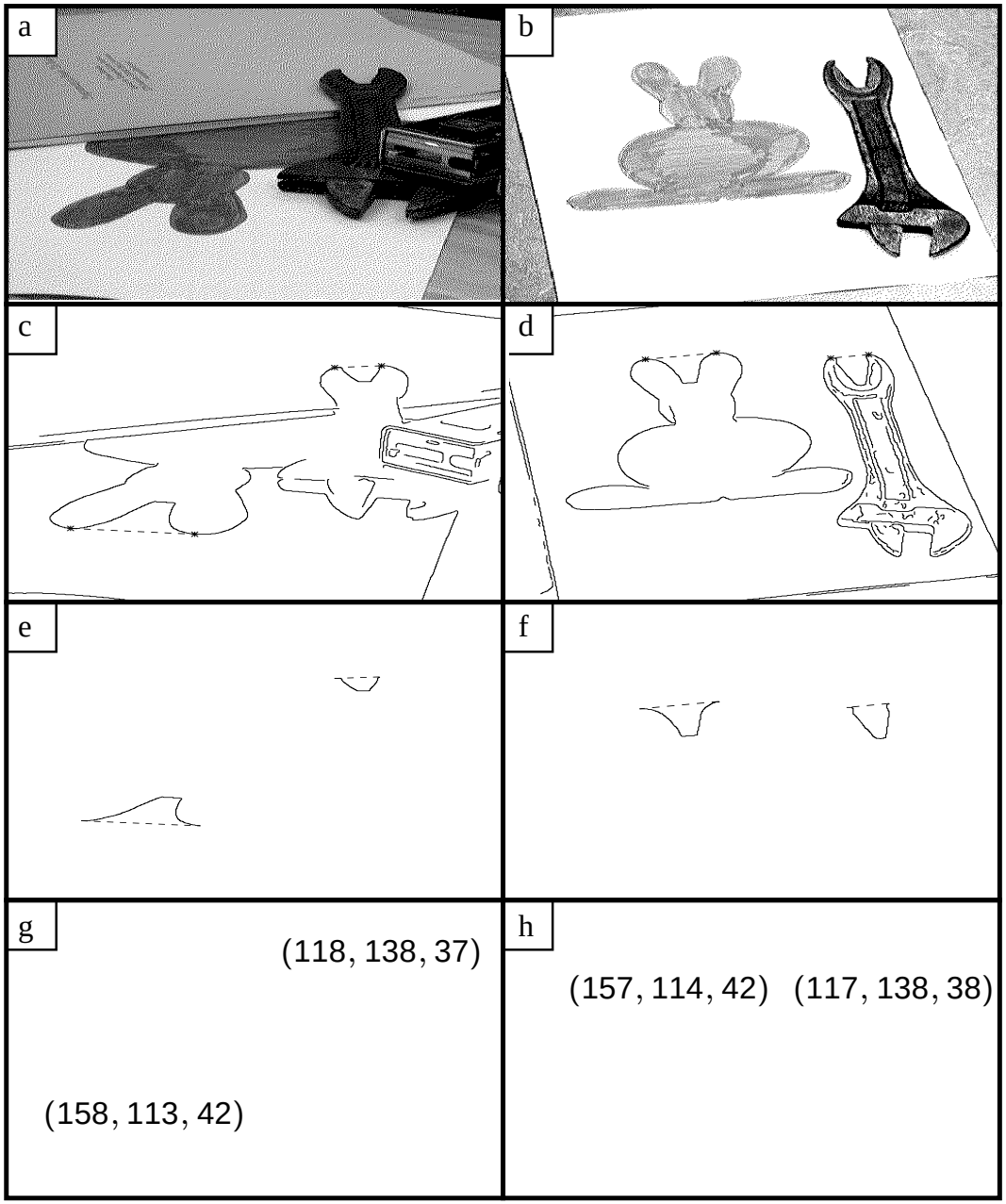
Assume that we measure the pixel coordinates for the left edge x_1 and the right edge x_2 of different objects. There are two classes (one for big objects and one for small objects). If we use $\mathbf{x} = (x_1, x_2)$ then different examples within a class will have very different feature vectors. However, if we use $\mathbf{x} = x_2 - x_1$ then the spread will become much less.

Invariant properties

Example

Assume that we would like to classify coins by studying their images. If we use intensities as features, then the feature vectors will vary depending on position and rotation of the coins.

The solution is often to find properties that are invariant (independent of) e.g. position, orientation, lighting etc.


a
b
c
d
e
f
g
(118, 138, 37)
(158, 113, 42)
h
(157, 114, 42)
(117, 138, 38)

Invariants

Definition

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ be a set of points and $\mathcal{G}$ a class of transformations (e.g. affine or projective). A function

$$f : (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \mapsto \mathbb{R}$$

is said to be an **invariant** under the transformation group $\mathcal{G}$ if

$$f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = f(T(\mathbf{x}_1), T(\mathbf{x}_2), \ldots, T(\mathbf{x}_n))$$

for every transformation $T \in \mathcal{G}$. ■

Invariants

In particular:

- $\mathcal{G}$=Euclidean transformations $\Rightarrow$ **Euclidean invariant** (rotation+translation)
- $\mathcal{G}$=similarity transformations $\Rightarrow$ **similarity invariant** (rotation+translation+global scale)
- $\mathcal{G}$=affine transformations $\Rightarrow$ **affine invariant**
- $\mathcal{G}$=projective transformations $\Rightarrow$ **projective invariant**

Examples of invariants

Translation and rotation

- Move the object so that the center is at the origin.
- Form a distance or intensity function that depends on angle.
- Calculate Fourier transforms and throw away the phase.
- or use distances between two points.

Planar rotations.

- Form a distance or intensity function that depends on angle.
- Calculate Fourier transforms and throw away the phase.
- or use distances between two points.

Examples of invariants

Affine transformations

- 3 points on a line, (ratios of distances)
- 4 points in a plane, (affine coordinates, shape)
- planar curves, (affine normalization)

Projective transformations

- 4 points on a line (cross ratio)
- 5 points in a plane, (projective normalization)

Cross ratio

Definition

$$\frac{\xi_1'/\xi_1}{\xi_2'/\xi_2} = \frac{\eta_1'/\eta_1}{\eta_2'/\eta_2} = \frac{(x^4 - x^2)/(x^3 - x^2)}{(x^4 - x^1)/(x^3 - x^1)}$$

is called the **cross ratio**. ■

Theorem

The cross ratio is a projective invariant.

Principal component analysis

Assume that we have a number of feature vectors $(x_1, \ldots, x_n)$, where every vector $x_i \in R^d$.
Study the deviations from the mean $x_i - \mu$, where

$$\mu = \frac{1}{n} \sum_i x_i$$

An estimate of the covariance matrix for the data is

$$\Sigma = \frac{1}{(n-1)} \sum_i (x_i - \mu)(x_i - \mu)^T$$

(Bessel's correction: sample covariance relies on the difference between each observation and the sample mean, but the latter is correlated with each observation $\Rightarrow$ divide by $n-1$ not n)
Assume now that we project data on one dimension, e.g. using

$$v(x) = v^T(x - \mu).$$

Principal component analysis

A good feature $v(x)$ describes as much of the variance as possible. The variance of $v(x)$ is given by

$$V(v) = v^T \Sigma v$$

Maximizing

$$V(v) = v^T \Sigma v$$

with the constraint that $v^T v = 1$ is the same thing as finding the largest eigenvector.

Principal component analysis

Additional features are obtained by taking the second largest eigenvector, etc.
Algorithm: If we want to project the features x of dimension d to a subspace v of dimension k so that as much of the variance is kept, then we should choose

$$v(x) = \begin{pmatrix} v_1^T x \\ \vdots \\ v_k^T x \end{pmatrix}$$

where v_i is the eigenvectors of the k largest eigenvalues of the covariance matrix Σ of x.
As a bonus, the features become independent
Example: Face basis

Artificial neural networks

ANN=Artificial Neural Networks
Idea: Build a classifier function $d(\mathbf{x})$ using simple but general parts.

Each part is a model of a neuron. Each part has a number of parameters.

By changing the parameters, one may build a large class of functions.

Try to fin the parameters that makes the classification as good as possible.

Perceptron

Artificial neural networks are built on the so called *perceptron* which can be written:

$$d(\mathbf{x}) = f(w^T \mathbf{x} + w_0)$$

where f is a non-linear function, e.g.

- $$f(y) = \begin{cases} 0, & y < 0 \\ 1, & y \geq 0 \end{cases}$$
- $$f(y) = \frac{1}{1 + e^{-y/b}}$$
- $$f(y) = \frac{1}{\pi} \operatorname{atan}(y) + \frac{1}{2}$$

ANN with two layers

It is common to use two layers

$$d(\mathbf{x}) = f(v^T y + v_0)$$

where

$$y_i = f(w_i^T \mathbf{x} + w_{i,0})$$

Training

By studying the training set $\mathbf{x}_j$ and comparing with the ground truth $s_{i,j}$ on may try to optimize the parameters (v, w), i.e.

$$\min_{(v,w)} ||d_i(\mathbf{x}_j) - s_{i,j}||$$

The most common method is to use local search (steepest descent) to change the variables For ANN, this optimisation is often called *back-propagation*.

Review - Lecture 9

- Basic machine learning concepts and applications
- Linear discriminants and Support vector machines
- Statistically optimal classification
- Nearest neighbor classification
- Dimensionality reduction
- Invariants
- Artificial neural networks