Image Analysis - Lecture 5

Machine Learning 1 - Classification

Kalle Åström

13 September 2016

Introduction
- Contents

Classification
- Optimal statistical classification
- Probability
- Nearest Neigbour classification

Other classifiers
- Artificial Neural Networks
- Convolutional Neural Networks - Deep Learning
- Support Vector Machines

Dimensionality reduction
- Principal component analysis

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)

All of these classification problems have in common:

- data - **x** (after segmentation, extract features)
- A number of classes

One would like to determine a class for every possible feature vector.
Here we will assume that the features are represented as a column vector, i.e. $\mathbf{x} \in \mathbb{R}^n$,

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

One would like to compare the feature vector **x** with those that one usually gets with a number of classes. Let y denote the class index, i.e. the classes are $y \in \omega_y = \{1, \ldots, M\}$ 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 - $y = 1$
- Iris versicolor - $y = 2$
- Iris setosa - $y = 3$.

How should one construct a function $f : \mathbb{R}^n \to \omega_y$ that given a features vector $\mathbf{x}$ determines, which class y 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

- Optimal statistical classification
- Nearest Neighbour classification
- Support Vector Machines
- Artificial Neural Networks

Assume that one feature vector $\mathbf{x}$ and class y are drawn from a joint probability distribution. If one can calculate the probability that the class is $y = j$ given the measurements $\mathbf{x}$, i.e. the so called **posterior probability**.

$$P(y = j|\mathbf{x})$$

The **maximum a posteriori classifier** is obtained as selecting the class j that maximizes the posterior probability, i.e.

$$j = \text{argmax}_k \, P(y = k|\mathbf{x}).$$

It is often easier to model and estimate the **likelihood** $P(\mathbf{x}|y = j)$ and to model the **prior** $p(y = j)$. The a posteriori probabilites can then be calculated using the Bayes rule,

$$p(y = j|\mathbf{x}) = \frac{p(\mathbf{x}|y = j)p(y = j)}{p(\mathbf{x})}.$$

Sometimes there is asymetric loss. Introduce a loss function L so that if our method classifies $\mathbf{x}$ as class j when it really is 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 $\mathbf{x}$ as ω_j is then

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

Using Bayes rule

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

one may rewrite r_j as

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

Choose j as to minimize loss, i.e. $y = f(\mathbf{x}) = \text{argmin}_j\, r_j(\mathbf{x})$. This gives the **statistically optimal** classifier.

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 $y = i$ if $r_i(\mathbf{x}) < r_j(\mathbf{x})$ for all $j \neq i$.

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

$$p(\mathbf{x}|y = j)p(y = j)$$

to decide which the optimal classification is.

Example with discrete variables

Assume three classes $\omega_y = \{1, 2, 3\}$.

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

The expected loss was

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

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

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

Minimize expected loss by maximizing

$$1 - r_j(\mathbf{x}) = p(y=j|\mathbf{x}) = \frac{p(\mathbf{x}|y=j)p(y=j)}{p(\mathbf{x})}.$$

We might as well find the j that maximize

$$p(\mathbf{x}|y=j)p(y=j).$$

Minimize expected loss by maximizing

$$1 - r_j(\mathbf{x}) = p(y = j|\mathbf{x}) = \frac{p(\mathbf{x}|y = j)p(y = j)}{p(\mathbf{x})}.$$

We might as well find the j that maximize

$$p(\mathbf{x}|y = j)p(y = j).$$

But once we have calculated $p(\mathbf{x}|y = j)p(y = j)$ it is straightforward to calculate

$$p(\mathbf{x}) = \sum_j (\mathbf{x}|y = j)p(y = j),$$

which after division gives

$$p(y = j|\mathbf{x}) = \frac{p(\mathbf{x}|y = j)p(y = j)}{p(\mathbf{x})},$$

which are probabilities that sum to 1.

First calculate

$$p(\mathbf{x}|y=j)p(y=j)$$

for each j.

$$p(\mathbf{x}|y=1)p(y=1) = 0.9^3 0.1^1 * 0.25 \approx 0.018$$

$$p(\mathbf{x}|y=2)p(y=2) = 0.9^1 0.1^3 * 0.25 \approx 0.0002$$

$$p(\mathbf{x}|y=3)p(y=3) = 0.9^2 0.1^2 * 0.50 \approx 0.004$$

The first image is most probable. But now it is easy to calculate

$$p(\mathbf{x}) = \sum_j (\mathbf{x}|y=j)p(y=j) = 0.018 + 0.0002 + 0004 = 0.0222$$

So we might as well go ahead and calculate the probabilites

$$p(y = j|\mathbf{x}) = \frac{p(\mathbf{x}|y = j)p(y = j)}{p(\mathbf{x})}$$

for each j.

$$p(y = 1|\mathbf{x}) = \frac{p(\mathbf{x}|y = 1)p(y = 1)}{p(\mathbf{x})} = \frac{0.018}{0.022} = 0.81$$

$$p(y = 2|\mathbf{x}) = \frac{p(\mathbf{x}|y = 2)p(y = 2)}{p(\mathbf{x})} = \frac{2.2510^{-4}}{0.022} = 0.01$$

$$p(y = 3|\mathbf{x}) = \frac{p(\mathbf{x}|y = 3)p(y = 3)}{p(\mathbf{x})} = \frac{0.004}{0.022} = 0.18$$

Note that the probabilites sum to 1. The probability that the image is from class $y = 1$ is 0.81.

Continuous probability distributions

It is common to model the features as continuous variables. The multivariate normal distribution is one popular choice. It has a mean μ and a covariance matrix Σ and a probability density function

$$f_X(X = x|Y = j) = \frac{1}{\sqrt{(2\pi)^n|\Sigma_j|}} e^{-\frac{1}{2}(\mathbf{x}-\mu_j)^T\Sigma_j^{-1}(\mathbf{x}-\mu_j)}.$$

A so called 'plug-in' classifier consists of two steps:
1. Training. Estimate parameters, e.g. m_j, Σ_j from training data.
2. Classification. Assuming that the model is correct and assuming that the estimates are exact. Use Bayes formula to estimate $p(w_i|x)$ and classify according to minimum loss or maximum a posteriori.

$$p(Y = j|X = \mathbf{x}) = \frac{f_X(X{=}\mathbf{x}|Y{=}j)P(Y{=}j)}{f_X(X{=}\mathbf{x})}$$

Estimate probabilities using histograms

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 Neigbour classification

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

$$T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\},$$

where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{1, \ldots, m\}$.

During classification one compares the distance between the feature vector $\mathbf{x}$ with the ones in the training set $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$, i.e. solves

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

and put

$$y = k.$$

(k,l)-nearest neigbour classification

A generalisation of the above method is to find the k nearest neigbours 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'.

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 find 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.

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

ANN with two layers

It was common to use two layers

$$d_j(\mathbf{x}) = f(v_j^T \mathbf{y} + v_{j,0})$$

where

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

and then interpret the last layer d_j as probabilites after a so called softmax normalization

$$P(Y = j | X = \mathbf{x}) \approx p_j = \frac{e^{d_j}}{\sum_{k=1}^{m} e^{d_k}}.$$

By studying the training set $\mathbf{x}_i$ and comparing with the ground truth y_i on may try to optimize the parameters (v, w), so as to get p_{y_i} close to 1. e.g. by optimizing

$$(v_{opt}, w_{opt}) = \text{argmin}_{(v,w)} \sum_{i=1}^{N} -\ln p_{y_i}(\mathbf{x}_i).$$

The most common method is to use local search (steepest descent) to change the variables For neural network, using steepest descent is usually called *back-propagation*.

Convolutional Neural Networks

Recently so called **Deep learning** has revolutionized the field of computer vision. It follows basically the same idea as for Artificial Neural Network.
More on this at a later lecture.

Assume that we would like to classify features in two classes. Denote the ground truth for the two classes: 1 and -1, i.e. the training data is

$$T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\},$$

where $\mathbf{x}_i \in R^d$ and $y_i \in \{-1, 1\}$.
Assume for simplicity that it is possible to separate data with a hyperplane, i.e. it is possible to find a vector w and a number b such that $y_i(w^T\mathbf{x}_i + b) \geq 0, \quad \forall i$.
Note the clever use of y_i.
Note also that it is possible to scale (w, b) such that $y_i(w^T\mathbf{x}_i + b) \geq 1, \quad \forall i$.

Optimal separating hyperplane

$y_i(w^T \mathbf{x}_i + b) \geq 1, \quad \forall i.$
The distance between a point $\mathbf{x}_i$ to the hyperplane (w, b) is given by $|\frac{w^T \mathbf{x}_i + b}{|w|}|$ which is larger than $\frac{1}{|w|}$.
Maximising the minimal distance to the hyperplane is equivalent to minimising $|w|$ or

$$\min_{w,b} w^T w$$

with constraints $y_i(w^T \mathbf{x}_i + b) \geq 1, \quad \forall i.$
This is a so called convex optimization problem. Such problems have no extra local minima.

Classification

After finding the optimal hyperplane (w, b) new examples are classified according to
$y = 1$ if $(w^T \mathbf{x}_i + b) > 0$ and $y = -1$ otherwise.

Feature selection

In image analysis, there is often much information.
E.g: 1600 $\times$ 1200 colour image has 5 760 000 elements, i.e.
x is a 5 760 000 $\times$ 1 vector.
For practical reasons one has to select some 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).
Feature selection is difficult. Some ideas are:

- Try single features in x. How good are they?
- Boosting
- Branch and bound - a technique for global optimisation.
- Singular value decomposition.
- Intuition and knowledge about the problem

Principal component analysis

Assume that we have a number of feature vectors $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$, where every vector $\mathbf{x}_i \in R^n$.
Study the deviations from the mean $\mathbf{x}_i - \mu$, where

$$\mu = \frac{1}{N} \sum_i \mathbf{x}_i$$

An estimate of the covariance matrix for the data is

$$\Sigma = \frac{1}{(N-1)} \sum_i (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T$$

Assume that we would like to generate one new feature, e.g. using

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

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

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

Maximising

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

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

Additional features are obtained by taking the second largest eigenvector, etc.

Algorithm: If we want to project the features $\mathbf{x}$ of dimension d to a subspace v of dimension k so that as much of the variance is kept, then we should choose

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

where v_i is the eigenvectors of the k largest eigenvalues of the covariance matrix Σ of $\mathbf{x}$.

As a bonus, the features become independent

Example: Face basis

Masters thesis suggestion of the day: Smart Cities

Review - Lecture 5

- Machine Learning - overview and examples
- Classification
  - Statistically optimal classification
  - Nearest neigbour classification
  - Artificial Neural Networks
  - Support Vector Machines
- Dimensionality reduction
  - Principal Component Analysis