Spatial Statistics and Image Analysis

Lecture 5

Konstantinos Konstantinou

Mathematical sciences
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden

Lecture's content

Todays lecture will cover

- K-fold cross validation
- Supervised methods for image classification
    1. M-nearest neighbors
    2. Support vector machines
    3. Neural networks

Resubstitution error-rate estimate

How to estimate the error rate of missclasification?

- Resubstitution error-rate estimate (error of the training set)
    1. Too optimistic as the data used to estimate the parameters are used to evaluate the error rates.
    2. Overfitting when number of parameters is large.

Solution: Divide data into train and test set.
Problem: Might be wasteful if data are scarce.

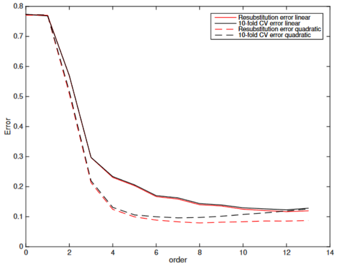
Cross-validation

- Use re-sampling methods (k- fold cross validation)
    1. Randomly divide the data into k approximate equal groups.
    2. Use all the data except those in group j to estimate the parameters.
    3. Use data in group j to evaluate the error rates.
    4. Repeat for all k groups and average the error rates.

How to choose k?
Currently k=5 or $k = 10$ is often recommended.
Special case when $k = n$, called leave-one-out cross-validation.

Example: Handwritten digits 1,...,9

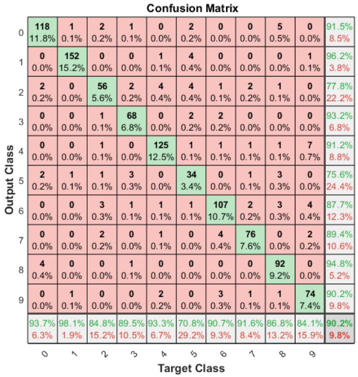
MNIST database: Images of handwritten digits 1,...,9
Aim: Classify the digits with using all the central moment features μ_{pq}, with $p + q \leq M$



Re-substitution and 10-fold CV error estimates for all $M \leq 13$.
Re-substitution error is lower than the CV error for both LDA and QDA.
For LDA minimum CV error is 12.3% for $M = 12$.
For QDA minimum CV error is 9.6% for $M = 7$.

Confusion matrix

A convenient way to illustrate the results of a discrimination analysis is the **confusion matrix**.
An example: MNIST data base, image moments.



Confusion Matrix

Output Class \ Target Class	0	1	2	3	4	5	6	7	8	9	
0	118 11.8%	1 0.1%	2 0.2%	1 0.1%	0 0.0%	2 0.2%	0 0.0%	0 0.0%	5 0.5%	0 0.0%	91.5% 8.5%
1	0 0.0%	152 15.2%	0 0.0%	0 0.0%	1 0.1%	4 0.4%	0 0.0%	0 0.0%	0 0.0%	1 0.1%	96.2% 3.8%
2	2 0.2%	0 0.0%	56 5.6%	2 0.2%	4 0.4%	4 0.4%	1 0.1%	2 0.2%	1 0.1%	0 0.0%	77.8% 22.2%
3	0 0.0%	0 0.0%	1 0.1%	68 6.8%	0 0.0%	2 0.2%	2 0.2%	0 0.0%	0 0.0%	0 0.0%	93.2% 6.8%
4	0 0.0%	0 0.0%	1 0.1%	0 0.0%	125 12.5%	1 0.1%	1 0.1%	1 0.1%	1 0.1%	7 0.7%	91.2% 8.8%
5	2 0.2%	1 0.1%	1 0.1%	3 0.3%	0 0.0%	34 3.4%	0 0.0%	1 0.1%	3 0.3%	0 0.0%	75.6% 24.4%
6	0 0.0%	0 0.0%	3 0.3%	1 0.1%	1 0.1%	1 0.1%	107 10.7%	2 0.2%	3 0.3%	4 0.4%	87.7% 12.3%
7	0 0.0%	0 0.0%	2 0.2%	0 0.0%	1 0.1%	0 0.0%	4 0.4%	76 7.6%	0 0.0%	2 0.2%	89.4% 10.6%
8	4 0.4%	0 0.0%	0 0.0%	1 0.1%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	92 9.2%	0 0.0%	94.8% 5.2%
9	0 0.0%	1 0.1%	0 0.0%	0 0.0%	2 0.2%	0 0.0%	3 0.3%	1 0.1%	1 0.1%	74 7.4%	90.2% 9.8%
	93.7% 6.3%	98.1% 1.9%	84.8% 15.2%	89.5% 10.5%	93.3% 6.7%	70.8% 29.2%	90.7% 9.3%	91.6% 8.4%	86.8% 13.2%	84.1% 15.9%	90.2% 9.8%

In MATLAB: plotconfusion(targets,outputs)

M-nearest neighbor classifier

Suppose we have a train set (x_i, y_i) where y_i denote the K classes. For a new observation x, the neighborhood of x, N_x is defined as the M points in the train set that are closest to x.
The probability that x is from class m is given by

$$P(Y = m \mid X = x) = \frac{1}{M} \sum_{i \in N_x} \mathbb{I}(Y_i = m)$$

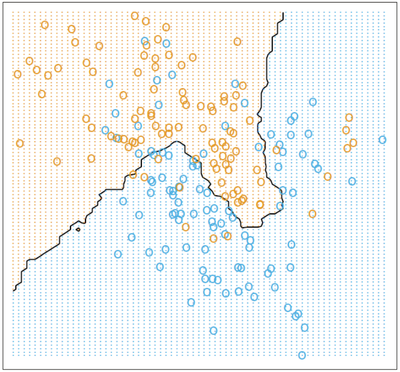
Thus the rule is very simple:

1. Look at your M nearest neighbors
2. The class with the largest number of neighbors wins!

Example M=15

Example of the boundary created by a 15-Nearest neighbor classifier.
A new observation x in the blue area will be classified as the "blue" class and in the orange area as the "orange" class.
Changing the number of neighbors will change the decision boundary.
How to choose M? Use cross validation

15-Nearest Neighbor Classifier



Support vector machines

Support vector machines(SVM) can be though as generalisations of linear discrimination.
Suppose we have two classes and a feature vector x.
Recall in LDA we choose class 1 over class 2 when $f(x) > 0$ where

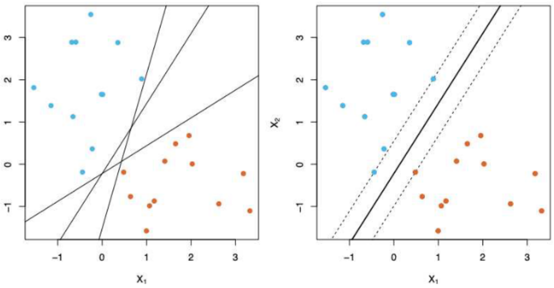
$$f(x) = (\mu_1 - \mu_2)^T C^{-1}(x - \frac{1}{2}(\mu_1 + \mu_2)) - ln\frac{\pi_2}{\pi_1}$$

More generally we choose class 1 when $f(x) > 0$ with

$$f(x) = \beta_0 + x^T \beta$$

SVM(cont.)

How to choose β_0 and β?
All three lines separate the data perfectly.
Basic idea of SVM: Find the optimal hyperplane that maximizes the distance from the nearest data points on each side.



SVM(cont.)

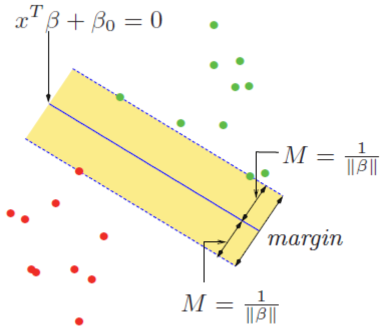
- Case 1: The problem is separable i.e there exist a hyperplane that can separate the data into different classes.
- We have pairs of inputs and labels (x_i, y_i), $i = 1, ..., N$, where y_i are labeled either 1 or -1 for the classes 1 and 2 respectively. Then, the SVM algorithm is equivalent to solving the optimization problem

$$\min_{\beta,\beta_0} || \beta ||$$

$$\text{Subject to} \qquad y_i(x_i^T \beta + \beta_0) \geq 1, \quad i = 1, ..., N$$

SVM

Recall that the shortest distance from a point x_0 to a hyperplane $x^T\beta + \beta_0 = 0$ is given by $\frac{|x_0^T\beta+\beta_0|}{||\beta||}$. This is equal to $\frac{1}{||\beta||}$ in our case.



SVM with soft margin

- Case 2: There exist no hyperplane that can separate the data.
- Idea: allow misclassifications but add a loss term.

To define the hyperplane we solve

$$\min_{\beta,\beta_0} \|\ \beta\ \|^2 + C \sum_{i=i}^{N} max\{0, 1 - y_i(\beta_0 + x_i^T \beta)\}$$

where C is a regularization parameter that controls the trade-off between maximizing the margin and minimizing the errors.

The term $max\{0, 1 - y_i(\beta_0 + x_i^T \beta)\}$ is the Hinge loss function.

The loss function penalizes points within the margin or points in the wrong side of the boundary

SVM with Kernel functions.

For a new vector x to be classified one can write the classifier in the form

$$f(x) = \beta_0 + x^T\beta = \beta_0 + \sum_{i=1}^{N} a_i x^T x_i^T$$

where x_i are observations in the train set and $a_i \in \mathbb{R}$.
Idea: It is easier to separate the data when you map them in a higher dimensional space.
However it can be hard and impractical to define such mappings and computationally expensive to implement.

Kernels

It turns out we don't need to transform the data but only compute the inner products in the transformed space with a Kernel function. Examples of Kernel functions:

Kernel Function		
Linear Kernel	$\mathcal{K}(x_i, x_j) = x_i \cdot x_j$	
Radial Basis Function Kernel	$\mathcal{K}(x_i, x_j) = e^{-\gamma \|\|x_i - x_j\|\|^2}$	$\gamma > 0$
Polynomial Kernel	$\mathcal{K}(x_i, x_j) = (x_i \cdot x_j + 1)^d$	$d \in \mathbb{N}$

Table: Popular Kernel Functions

Neural networks for image classification

- Assume we have an image x with pixels $x_1, ..., x_N$, which can belong into two classes.
- We assume that the probabilities of belonging to the two classes

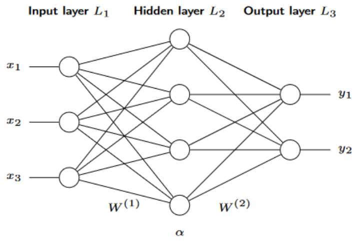
$$y_1 = P(Class = 1 \mid x) = f(x;\theta) \tag{1}$$

$$y_2 = P(Class = 2 \mid x) = 1 - f(x;\theta) \tag{2}$$

- The idea of neural nets is to approximate $f(x)$ as a sequence of "simple" non-linear functions.

Single layer network

- Let's look at a single-layer model first.



- f(x) is in the form

$$f(x) = g^{(2)}(W^{(2)}g^{(1)}(W^{(1)}x + b^{(1)}) + b^{(2)})$$

  where W and b are weights and biases parameters that need to be estimated.
- As y_1, y_2 are probabilities, we use the softmax function $g_k^{(2)}(x_1, x_2) = \frac{e^{x_k}}{e^{x_1}+e^{x_2}}$, k=1,2.

Feed forward network

- This is a feed-forward network since information only flows forward in the network.
- The nodes in the hidden layer are called neurons.
- Input data: x_1, x_2, x_3,
- Output: probabilities for classes 1 and 2.
- $g^{(1)}$ is an activation function. Common activation functions for the internal layers are:
  - Rectified Linear Unit (RELU): $g(x) = max(0, x)$
  - Sigmoid function: $g(x) = \frac{1}{1+e^{-x}}$
  - $g(x) = tanh(x)$

Backpropagation

The networks needs to be trained i.e find set of parameters $\theta = (W, b)$ that minimize a loss function.
For classification we often use the cross-entropy loss function

$$L(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} log f_k(x_i)$$

1. Initialize W and b
2. For input x given the parameters W and b calculate $f(x)$
3. Update the parameters W, b using an optimization method, using gradient descent the updates are :

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \gamma \frac{\partial L}{\partial w_{ij}^{(l)}}$$

$$b_p^{(l)} \leftarrow b_p^{(l)} - \gamma \frac{\partial L}{\partial b_p^{(l)}}$$

4. Go back to step 2 and repeat until convergence.

The gradient of L can be calculated using the chain rule.
The γ parameter is called the learning rate.

Regularization

Neural networks in general have too many parameters and will overfit the data. Some of the solutions are

1. Early stopping: Stop the training when the validation set error start increasing .
2. Use a regularized loss function

$$\mathcal{L}(\theta) = L(\theta) + \lambda R(\theta)$$

for instance

$$R(\theta) = \sum_{i=1}^{|\theta|} \theta_i^2$$

shrink the parameters towards zero. The parameter λ needs to be chosen wisely ($\lambda = 0$ corresponds to no regularization and large value for λ might cause underfitting). Usually tuned using cross validation.

Convolutional neural networks

Let W and g be matrices. The convolution $W * g$ is defined by

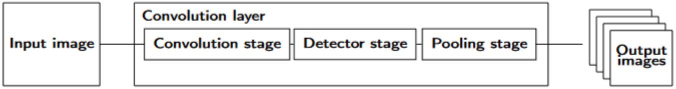
$$(W * g)_{ij} = \sum_k \sum_l w_{kl} g_{i-k,j-l}$$

- A fully connected network with an image as input will have too many parameters as we have a separate weight between each pixel and each hidden node.
- A Convolutional neural network assumes that the input data has a lattice structure. They are useful for analysis of images.
- A CNN is a method for image classification using filtered images as features, but where we do not need to specify features manually.
- A CNN consists of a special type of layers called convolution layers, which are based on filtering the image with a kernel.

Convolutional layer

A convolutional layer has three stages:

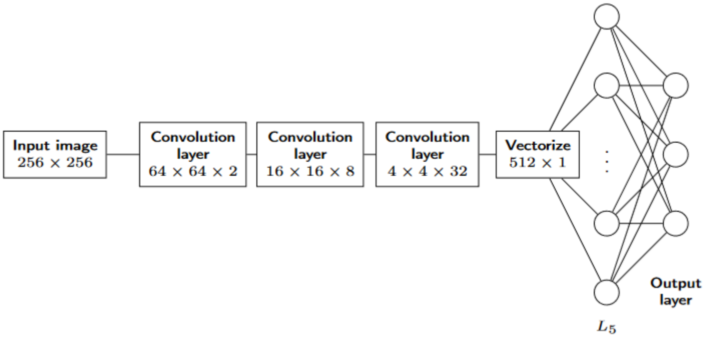
1. Convolution stage: Convolve each input image with f different linear filters, with kernels of size $q \times q$, producing f output images.
2. Detector stage: Apply a non-linear function to each image. Typically the RELU function.
3. Pooling stage: For each image, reduce each non-overlapping block of $r \times r$ pixels to one single value, by for example taking the largest value in the block.



Comments

- One could view the convolution stage as a regular layer where most of the weights are zero: A pixel in the output image only depends on the $q \times q$ nearest pixels in the input image.
- The different nodes share parameters, since we use the same convolution kernel across the entire image.
- As a result, a convolution layer has fq^2 parameters, where q is typically a small positive number.
- Since pooling reduces the image size, we can in the next stage use more filters without increasing the total number of nodes.
- Pooling makes the output less sensitive to small translations of the input.
- Another variant of pooling is to take the max across different learned features. This can make the output invariant to other things, such as rotations.

Example of a CNN



- The first layer has $f = 2$ filters, the second has $f = 4$ and the third $f = 4$ filters.
- Each pooling stage uses $r = 4$.
- The final hidden layer is usually a fully connected layer.

Comments

- Using CNNs for image classification re-popularized neural networks around 2010, and “Deep learning” was coined as a flashy name for using “deep” neural networks with more than one hidden layer.
- Transfer learning: Use a popular pre-trained model for a challenging image classification task as a starting point.
- For further details on neural networks, for example see:
    1. Computer age statistical inference by Efron and Hastie.
    2. deeplearningbook.org
    3. Matlab guides: Create simple deep learning network for classification
    4. A Gentle Introduction to Transfer Learning for Deep Learning