GI01/4C55: Supervised Learning

1. Introduction

October 3, 2005

Massimiliano Pontil

Course information

1. When: Mondays, 14:00–17:00
   Where: Room 105, 24 Gordon Square

2. Course webpage:
   http://www.cs.ucl.ac.uk/staff/M.Pontil/courses/index-SL05.htm

3. Mailing list: gi01@cs.ucl.ac.uk

4. Office: 8.05, CS Building, Malet Place
Assessment

1. Homework (40%) and Exam (60%)

2. 5 homework assignments
   (deliver them on-time, penalty otherwise)

3. To pass the course, you must obtain at least 40% on the homework component and an average of at least 50% when the homework and exam components are weighted together

Material

• Lecture notes
  – http://www.cs.ucl.ac.uk/staff/M.Pontil/courses/index-SL05.htm

• Reference book

• Additional material (see webpage for more info)
  – General: Duda, Hart & Stork; Mitchell; Bishop; ...
  – Bayesian methods in ML: Mackay; ...
  – Kernel methods: Shawe-Taylor & Cristianini; Schölkopft & Smola; ...
  – Learning theory: Devroye, Lugosi, & Gyorfi; Vapnik; ...
Prerequisites

- Calculus (real-valued functions, limits, derivatives, Taylor series, integrals,...)
- Elements of probability theory (random variables, expectation, variance, conditional probabilities, Bayes rule,...)
- Fundamentals of linear algebra (vectors, angles, matrices, eigenvectors/eigenvalues,...),
- A bit of optimization theory (convex functions, Lagrange multipliers)

Course outline

- (Weeks 1,2) Key concepts (probabilistic formulation of learning from examples, error functionals; loss function, Bayes rule, learning algorithm, overfitting and underfitting, model selection, cross validation); Some basic learning algorithms (linear regression, k-NN, linear discriminant analysis, logistic regression); Discriminative vs. Generative approach
- (Week 3) Hypothesis spaces (Linear models, kernels, neural nets, trees), Optimization algorithms
- (Week 4) Elements of Learning Theory (PAC-learning, Complexity of hypothesis space, VC-bounds)
- (Week 5) Regularization, Kernels
Course outline (cont.)

- (Week 6) Support vector machines, Gaussian processes
- (Week 7) Decision trees, Boosting
- (Week 8) Online learning
- (Week 9) Semi-supervised learning
- (Week 10) Multi-task learning

Acknowledgments: Some lectures use material from last year course. Thanks to Fernando Perez-Cruz, Iain Murray and Ed Snelson of the Gatsby Unit at UCL for preparing this!

Today plan

- Supervised learning problem
- Regression and classification
- Two learning algorithms: least squares and $k$-NN
- Probabilistic model, error functional, optimal solutions
- Hypothesis space, overfitting and underfitting
- Choice of the learning algorithm (Model selection)
Supervised learning problem

Given a set of **input/output** pairs (**training set**) we wish to compute the functional relationship between the input and the output

\[ x \rightarrow f \rightarrow y \]

- **Ex1** (people detection) given an image we wish to say if it depicts a person or not
  The output is one of 2 possible categories

- **Ex2** (pose estimation) we wish to predict the pose of a face image
  The output is a continuous number (here a real number describing the face rotation angle)

In both problems the input is a high dimensional vector \( x \) representing pixel intensity/color
People detection example (cont.)

Data are sparse! Risk for overfitting!

Notation

- $\mathcal{X}$: input space (e.g., $\mathcal{X} \subseteq \mathbb{R}^d$), with elements $x, x', x_i, \ldots$
- $\mathcal{Y}$: output space, with elements $y, y', y_i, \ldots$
  - Classification (qualitative output) $\mathcal{Y} = \{c_1, \ldots, c_k\}$
    - Binary classification: $k = 2$
  - Regression (quantitative output) $\mathcal{Y} \subseteq \mathbb{R}$
- $S = \{(x_i, y_i)\}_{i=1}^m$: training set (set of I/O examples)

Supervised learning problem: compute a function which “best describes” I/O relationship
Learning algorithm

Training set: \( S = \{ (x_i, y_i)_{i=1}^m \} \subseteq \mathcal{X} \times \mathcal{Y} \)

A **learning algorithm** is a mapping \( S \mapsto f_S \)

A new input \( x \) is predicted as \( f_S(x) \)

In the course we mainly deal with deterministic algorithms but we'll also comment on some randomized ones

Today: we describe two simple learning algorithms:
linear regression and \( k \)-nearest neighbors

Some important questions

- How is the data **collected**? (need assumptions!)
- How do we **represent** the inputs? (may require preprocessing step)
- How **accurate** is \( f_S \) on new data (study of **generalization error**) / How do we **evaluate performance** of the learning algorithm on unseen data?
- How “**complex”** is a learning task? (computational complexity, sample complexity)
- Given two different learning algorithms, \( f_S \) and \( f'_S \) which one should we choose? (**model selection** problem)
Some difficulties/aspects of the learning process

- New inputs **differ** from the ones in the training set (look up tables do not work!)
- Inputs are measured with **noise**
- Output is **not deterministically** obtained by the input
- Input is **high dimensional** but some components/variables may be irrelevant
- Whenever **prior knowledge** is available it should be used

More examples / applications

- Optical digit recognition (useful for identifying the numbers in a ZIP code from a digitalized image) (Computer Vision)
- Predicting house prices based on sq. feet, number of rooms, distance from central London,... (Marketing)
- Estimate amount of glucose in the blood of a diabetic person (Medicine)
- Detect spam emails (Information retrieval)
- Predict protein functions / structures (Bioinformatics)
- Speaker identification / sound recognition (Speech recognition)
Binary classification: an example

We describe two basic learning algorithms/models for classification which can be easily adapted to regression as well. We choose: $\mathcal{X} = \mathbb{R}^2$, $x = (x_1, x_2)$ and $\mathcal{Y} = \{green, red\}$

Our first learning algorithm computes a **linear function** (perceptron), $w^T x + b$ and classifies an input $x$ as

$$f(x) = \begin{cases} 
red & w^T x + b > \frac{1}{2} \\
green & w^T x + b \leq \frac{1}{2} 
\end{cases}$$

Least squares

How do we compute the parameters $w$ and $b$?

We do not lose generality if we set $b = 0$ (we can change the data representation as $x^T \rightarrow (x^T, 1)$ and $w^T \rightarrow (w^T, b)$)

We find $w$ by minimizing the residual sum of squares on the data

$$R(w) = \sum_{i=1}^{m} (y_i - w^T x_i)^2$$

To compute the minimum we need to solve the system of equations

$$\nabla R(w) = 0 \quad \text{recall that : } \nabla = \left( \frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2} \right)^T$$
Normal equations

\[ R(w) = \sum_{i=1}^{m} (y_i - w^\top x_i)^2 \]

Note that:
\[ \frac{\partial R}{\partial w_k} = 2 \sum_{i=1}^{m} (w^\top x_i - y_i) \frac{\partial (w^\top x_i)}{\partial w_k} = 2 \sum_{i=1}^{m} (w^\top x_i - y_i) x_{ik} \]

Hence, to find \( w = (w_1, w_2)^\top \) we need to solve the linear system of equations
\[ \sum_{i=1}^{m} (x_{ik} x_1 w_1 + x_{ik} x_2 w_2) = \sum_{i=1}^{m} x_{ik} y_i, \quad k = 1, 2 \]

Normal equations (cont.)

In vector notations:
\[ \sum_{i=1}^{m} x_i x_i^\top w = \sum_{i=1}^{m} x_i y_i \]

In matrix notation:
\[ X^\top Xw = X^\top y \]

where
\[ X^\top = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{d1} & \cdots & x_{dm} \end{bmatrix} \equiv [x_1, \cdots, x_m], \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \]

Note: here \( d = 2 \) but, clearly, the result holds for any \( d \in \mathbb{N} \)
Least square solution

\[ X^\top Xw = X^\top y \]

For the time being we will assume that the matrix \( X^\top X \) is invertible, so we conclude that

\[ w = (X^\top X)^{-1}X^\top y \]

Otherwise, the solution may not be unique... (we will deal with the case that \( X^\top X \) is not invertible later)

Going back to \textit{“b”}

Substituting \( x^\top \) by \( (x^\top, 1) \) and \( w^\top \) by \( (w^\top, b) \), the above system of equations can be expressed in matrix form as (exercise):

\[
\begin{align*}
(X^\top X)w + X^\top 1b &= X^\top y \\
1^\top Xw + mb &= 1^\top y
\end{align*}
\]

that is

\[
\begin{bmatrix}
X^\top X & X^\top 1 \\
1^\top X & m
\end{bmatrix}
\begin{bmatrix}
w \\
b
\end{bmatrix} =
\begin{bmatrix}
X^\top y \\
1^\top y
\end{bmatrix}
\]

where \( 1 = (1, 1, \ldots, 1)^\top \), \( m \times 1 \) vector of \textit{“ones”}
A different approach: $k$–nearest neighbors

Let $N(x; k)$ be the set of $k$ nearest training inputs to $x$ and $I_x = \{i : x_i \in N(x; k)\}$ the corresponding index set

$$f(x) = \begin{cases} 
\text{red} & \text{if } \frac{1}{k} \sum_{i \in I_x} y_i > \frac{1}{2} \\
\text{green} & \text{if } \frac{1}{k} \sum_{i \in I_x} y_i \leq \frac{1}{2}
\end{cases}$$

- Closeness is measured using a metric (e.g., Euclidean dist.)
- Local rule (compute local majority vote)
- Decision boundary is non-linear

Note: for regression we set $f(x) = \frac{1}{k} \sum_{i \in I_x} y_i$ (a “local mean”)

$k$–NN: the effect of $k$

- The smaller $k$ the more irregular the decision boundary

![Images](k=15.png) ![Images](k=1.png)

- How to choose $k$? later...
Linear regression vs. $k$-NN (informal)

- Global vs. local
- Linear vs. non-linear
- Bias / variance considerations:
  - LR relies heavily on linear assumption (may have large bias) $k$-NN does not
  - LR is stable (solution does not change much if data are perturbed) 1-NN isn’t!
- $k$-NN sensitive to input dimension $d$: if $d$ is high, the inputs tends to be far away from each other!

Supervised learning model (back to Q1)

We assume that the data is obtained by sampling i.i.d. from a fixed but unknown probability density $P(x, y)$

Expected error:

$$
\mathcal{E}(f) := \mathbb{E} \left[ (y - f(x))^2 \right] = \int (y - f(x))^2 dP(x, y)
$$

Our goal is to minimize $\mathcal{E}$

Optimal solution: $f^* := \text{argmin}_f \mathcal{E}(f)$

But: in order to compute $f^*$ we need to know $P$!

Note: for binary classification with $\mathcal{Y} = \{0, 1\}$ and $f : \mathcal{X} \to \mathcal{Y}$, $\mathcal{E}(f)$ counts the average number of mistakes of $f$ (aka expected misclassification error)
**Regression function...**

Let us compute the optimal solution $f^*$ for regression ($\mathcal{Y} = \mathbb{R}$) and binary classification ($\mathcal{Y} = \{0, 1\}$).

Using the decomposition $P(y, x) = P(y|x)P(x)$ we have

$$\mathcal{E}(f) = \int_X \left\{ \int_{\mathcal{Y}} (y - f(x))^2 dP(y|x) \right\} dP(x)$$

so we see that

- for regression, $f^*$ (called the regression function) is

$$f^*(x) = \arg\min_{c \in \mathbb{R}} \int_{\mathcal{Y}} (y - c)^2 dP(y|x) = (\text{exercise}) = \int_{\mathcal{Y}} ydP(y|x)$$

---

**... and Bayes classifier**

- for binary classification, $f^*$ (called the Bayes classifier) is

$$f^*(x) = \begin{cases} 1 & \text{if } P(1|x) > \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

To see this note, as before, that

$$f^*(x) = \arg\min_{c \in \{0, 1\}} \left\{ \int_{\mathcal{Y}} (y - c)^2 dP(y|x) = (1 - c)^2 P(1|x) + c^2 P(0|x) \right\}$$

and use

$$P(0|x) + P(1|x) = 1$$
**k-NN revised**

k-NN attempts to estimate/approximate $P(1|x)$ as $\frac{1}{k} \sum_{i \in I_x} y_i$

- Expectation is replaced by averaging over sample data

- Conditioning at $x$ is relaxed to conditioning on some region close to $x$

One can show that $\mathcal{E}(f_S) \to \mathcal{E}(f^*)$ as $m \to \infty$ provided that: $k(m) \to \infty$ and $\frac{k(m)}{m} \to 0$

Weakness: the approximation (rate of convergence) depends critically on the input dimension...

---

**Least squares revisited**

$P(x, y)$ is unknown $\Rightarrow$ cannot compute $f^* = \arg\min_f \mathcal{E}(f)$

We are only given a sample (training set) from $P$

**A natural approach:** we approximate the expected error $\mathcal{E}(f)$ by the empirical error

$$\mathcal{E}_{\text{emp}}(f) := \frac{1}{m} \sum_{i=1}^{m} (y_i - f(x_i))^2$$

- If we minimize $\mathcal{E}_{\text{emp}}$ over all possible functions, we can always find a function with zero empirical error!

**Note:** Sometimes the (empirical) error is called (empirical) risk
Least squares revisited (cont.)

Proposed solution: we introduce a restrictive class of functions $\mathcal{H}$ called **hypothesis space**

We minimize $\mathcal{E}_{\text{emp}}$ within $\mathcal{H}$. That is, our learning algorithm is:

$$f_S = \arg \min_{f \in \mathcal{H}} \mathcal{E}_{\text{emp}}(f)$$

This approach is usually called **empirical error minimization**

Linear regression: $\mathcal{H} = \{ f(x) = w^T x + b : w \in \mathbb{R}^d, b \in \mathbb{R} \}$

31

Simplified scenario

Let us discuss the case that $\mathcal{X}$ is a finite set, $\mathcal{X} = \{ t^{(1)}, \ldots, t^{(N)} \} \subset \mathbb{R}^d$, and $\mathcal{Y} = \{ 0, 1 \}$. Moreover, assume that

- $P(x)$ is the uniform distribution
- the I/O relationship is deterministic: $P(y|x) = \delta(y, f^*(x))$

$$\mathcal{E}(f) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} (y - f(x))^2 P(x, y) = \frac{1}{N} \sum_{n=1}^{N} (f^*(t^{(n)}) - f(t^{(n)}))^2$$

$f^*$ is the Bayes solution and the Bayes error, $\mathcal{E}(f^*)$, is zero

**Note:** we may write $x_i = t^{(n(i))}$ where $n(i) \in \{1, \ldots, N\}$

32
Simplified scenario (cont)

\[ \mathcal{E}(f) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} (y - f(x))^2 P(t(n), y) = \frac{1}{N} \sum_{n=1}^{N} (f^*(t^n) - f(x))^2 \]

\[ \mathcal{E}_{\text{emp}}(f) = \sum_{i=1}^{m} (y_i - f(x_i))^2 \]

If hypothesis space \( \mathcal{H} \) is large, \( \mathcal{E}_{\text{emp}}(f) = 0 \) for many classifiers \( f \in \mathcal{H} \) but \( \mathcal{E}(f) \) may bounded away from zero! (risk for overfitting)

**Warning:** approximation \( \mathcal{E}_{\text{emp}}(f) \approx \mathcal{E}(f) \) may be too optimistic!

### Summary

- Data \( S \) sampled i.i.d from \( P \) (fixed but unknown)
- \( f^* \) is what we want, \( f_S \) is what we get
- Different approaches to attempt to estimate/approximate \( f^* \):
  - Minimize \( \mathcal{E}_{\text{emp}} \) in some restricted space of functions (eg, linear)
  - Compute local approximation of \( f^* \) (\( k \)-NN)
  - Estimate \( P \) and then use Bayes rule... (we’ll discuss this next week)
Perspectives

Theoretical / methodological aspects involved in supervised learning

- Function representation and approximation – to describe $H$
- Optimization/numerical methods – to compute $f_S$
- Probabilistic methods – to study generalization error of $f_S$

Additive noise model

Consider the regression problem. Assume that the output is computed as

$$y = f(x) + \epsilon$$

where $\epsilon$ is a zero mean r.v. Hence we can write

$$P(y, x) = P(y|x)P(x) = P_\epsilon(y - f(x))P(x)$$

where $\mathbb{E}[\epsilon] = 0$

Noise free model: $y$ deterministically computed from $x$ ($\epsilon \equiv 0$)
Additive noise model (cont.)

\[ y = f(x) + \epsilon \]

\[ P(y, x) = P(y|x)P(x) = P_\epsilon(y - f(x))P(x) \]

The training data is obtained, for \( i = 1, \ldots, m \) as following

- sample \( x_i \) from \( P_x \)
- sample \( \epsilon_i \) from \( P_\epsilon \)
- set \( y_i = f(x_i) + \epsilon_i \)

**Note:** \( P(x) \equiv P_x(x) \) (just use different notation when needed)

Additive noise model (cont.)

\[ P(y, x) = P(y|x)P_x(x) = P_\epsilon(y - f(x))P_x(x) \]

So, since \( \epsilon \) has zero mean, we have that

\[ f^*(x) := \int ydP(y|x) = \int ydP_\epsilon(y - f(x)) = \int (f(x) + \epsilon)dP_\epsilon(\epsilon) = f(x) \]

A common choice for the noise distribution \( P_\epsilon \) is a Gaussian:

\[ P_\epsilon(\epsilon) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{\epsilon^2}{2\sigma^2} \right) \]
Maximum likelihood

What is the probability of the data $S$?

\[
P(S) = \prod_{i=1}^{m} P(y_i, x_i) = \prod_{i=1}^{m} P(y_i|x_i)P(x_i)
\]

\[
= \prod_{i=1}^{m} P(x_i) \prod_{i=1}^{m} P(y_i|x_i) = A \prod_{i=1}^{m} P(y_i|x_i)
\]

where

\[
A = \prod_{i=1}^{m} P(x_i) = P(x_1, \ldots, x_m)
\]

We define the likelihood of $f$ as

\[
L(f; S) = P(S; f)
\]

Maximum likelihood (cont.)

Maximum likelihood principle: compute $f$ by maximizing $L(f; S)$

If we use a linear functions and additive Gaussian noise, we have

\[
L(w; S) = A \prod_{i=1}^{m} \left(2\pi\sigma^2\right)^{-\frac{1}{2}} \exp \left\{ \frac{-\left(y_i - w^\top x_i\right)^2}{2\sigma^2} \right\}
\]

In particular the log likelihood is (note: since the log function is strictly increasing maximization $L$ or $\log L$ is the same)

\[
\log L(w; S) = -\frac{1}{2\sigma^2} \sum_{i=1}^{m} \left(y_i - w^\top x_i\right)^2 + \text{const}
\]

Hence maximizing the likelihood is equivalent to least squares!
Supervised learning as function approximation

Given the training data $y_i = f^*(x_i) + \epsilon_i$, the goal is to compute an “approximation” of $f^*$ in some region of $\mathbb{R}^d$.

We look for an approximant of $f^*$ within a prescribed hypothesis space $\mathcal{H}$

- Unless prior knowledge is available on $f^*$ (eg, $f^*$ is linear) we cannot expect $f^* \in \mathcal{H}$
- Choosing $\mathcal{H}$ “very large” leads to overfitting! (we’ll see an example of this in a moment)

Polynomial fitting

As an example of hypothesis spaces of increasing “complexity” consider regression in one dimension

\[
H_0 = \left\{ f(x) = b : b \in \mathbb{R} \right\}
\]

\[
H_1 = \left\{ f(x) = ax + b : a, b \in \mathbb{R} \right\}
\]

\[
H_2 = \left\{ f(x) = a_1 x + a_2 x^2 + b : a_1, a_2, b \in \mathbb{R} \right\}
\]

\[\vdots\]

\[
H_n = \left\{ f(x) = \sum_{\ell=1}^{n} a_\ell x^\ell + b : a_1, \ldots, a_n, b \in \mathbb{R} \right\}
\]

Let us minimize the empirical error in $\mathcal{H}_r$ ($r =$ polynomial degree)
Polynomial fitting (simulation)

\[ r = 0, 1, 2, 3, 4, 5. \] As \( r \) increases the fit to the data improves (empirical error decreases)

Overfitting vs. underfitting

- Compare the empirical error (solid line) with expected error (dashed line)
  - \( r \) small: underfitting
  - \( r \) large: overfitting

- The larger \( r \) the lower the empirical error of \( f_5 \)
  \( \Rightarrow \) We cannot rely on the training error!
$k$-NN: the effect of $k$

- The smaller $k$ the more irregular the decision boundary and the smaller the empirical error

\[ k = 15 \quad k = 1 \]

$\frac{m}{k}$ large: overfitting

$\frac{m}{k}$ small: underfitting
Model selection

How to choose \( k \) in \( k \)-NN?

How to choose the degree \( r \) for polynomial regression?

The simplest approach is to use part of the training data (say 2/3) for training and the rest as a validation set.

Another approach is \( K \)-fold cross-validation: we split the data in \( K \) parts (of roughly equal sizes) and repeatedly train on \( K - 1 \) parts and test on the part “left out”

- We will discuss the model selection problem later in the course.

Other learning paradigms

Supervised learning is not the only learning setup!

- **Semi-supervised learning**: the ‘learning environment’ may give us access to many input examples but only few of them are labeled.

- **Active learning**: we are given many inputs and we can choose which ones to ask the label for.

- **Unsupervised learning**: We have only input examples. Here we may want to find data clusters, estimate the probability density of the data, find important features/variable (dimensionality reduction problem), detect anomalies, etc.