Supervised Learning

1. Introduction to supervised learning

Mark Herbster

University College London
Department of Computer Science

September 29, 2014

Acknowledgments and References

Thanks

Thanks to Massi Pontil for the course notes with additions by John-Shawe Taylor. These in turn were inherited notes from Fernando Perez-Cruz, Iain Murray and Ed Snelson of the Gatsby Unit at UCL.

Useful References

- *Pattern Recognition and Machine Learning*, Bishop, Christopher M., Springer (2006)
- *An Introduction to Support Vector Machines*, Shawe-Taylor J. and Cristianini N., Cambridge University Press (2000)
- *Kernel Methods for Pattern Analysis*, Shawe-Taylor.J, and Cristianini N., Cambridge University Press (2004)

Course information

1. When: Mondays, 14:00–17:00
2. Course webpage:
   `http://www.cs.ucl.ac.uk/staff/M.Herbster/GI01/`
3. Office: 8.03, CS Building, Malet Place
4. Questions : `sl-support@cs.ucl.ac.uk`

Assessment

1. Homework (25%) and Exam (75%)
2. 2 homework assignments
   (deliver them on-time, penalty otherwise)
3. To pass the course, you must obtain 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.Herbster/GI01`
- Reference book
  - Hastie, Tibshirani, & Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2001
- 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, & Gÿorfi; 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)

Provisional course outline

- (Week 1) 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);
- (Week 2) Statistical Learning Theory
- (Week 3) Regularisation, Kernels
- (Week 4) Lab on Regression and Kernels
- (Week 5) Support Vector Machines

Provisional course outline

- (Week 6) Sparsity Methods
- (Week 7) Proximal Methods
- (Week 8) Multi-task Learning
- (Week 9) Semi-supervised learning
- (Week 10) Online learning

Today's plan

- Supervised learning problem
- Why and when learning?
- 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

$$\boldsymbol{x} \longrightarrow \boxed{f} \longrightarrow y$$

- **Example 1:** (people detection) given an image we wish to say if it depicts a person or not. The output is one of 2 possible categories
- **Example 2:** (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 $\boldsymbol{x}$ representing pixel intensity/color

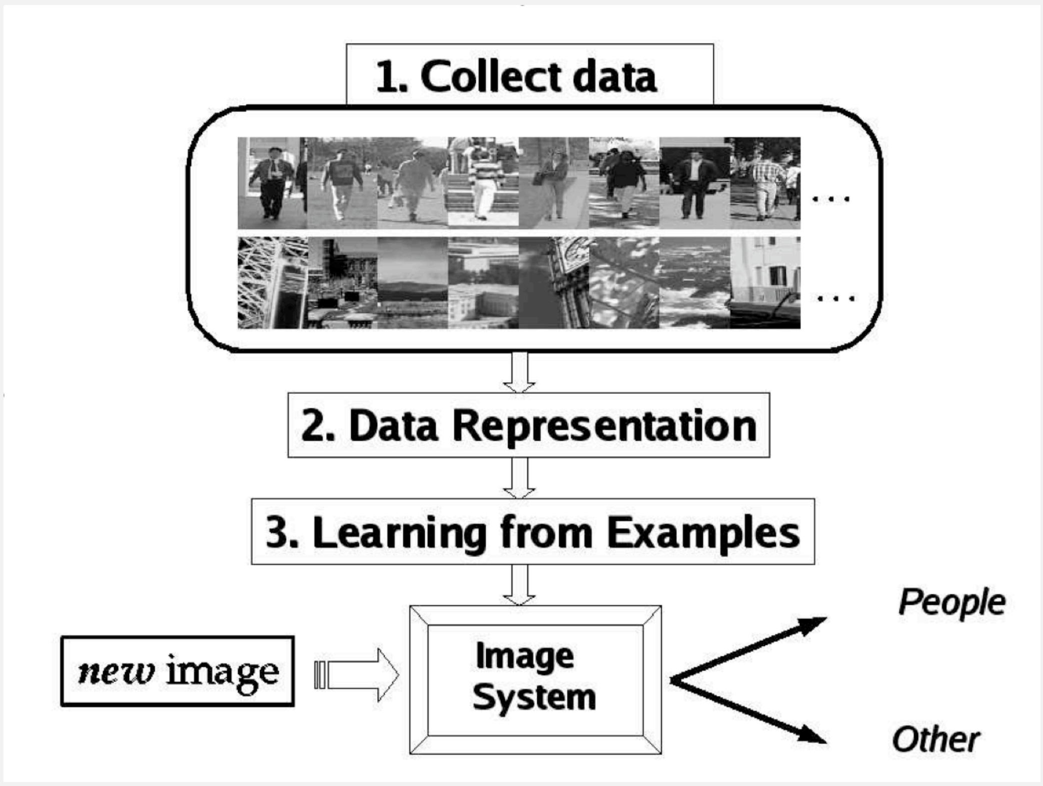
Why and when learning?

- The aim of learning is to develop software systems able to perform particular tasks such as people detection.
- Standard software engineering approach would be to specify the problem, develop an algorithm to compute the solution, and then implement efficiently.
- Problem with this approach is developing the algorithm:
  - No known criterion for distinguishing the images;
  - In many cases humans have no difficulty;
  - Typically problem is to specify the problem in logical terms.

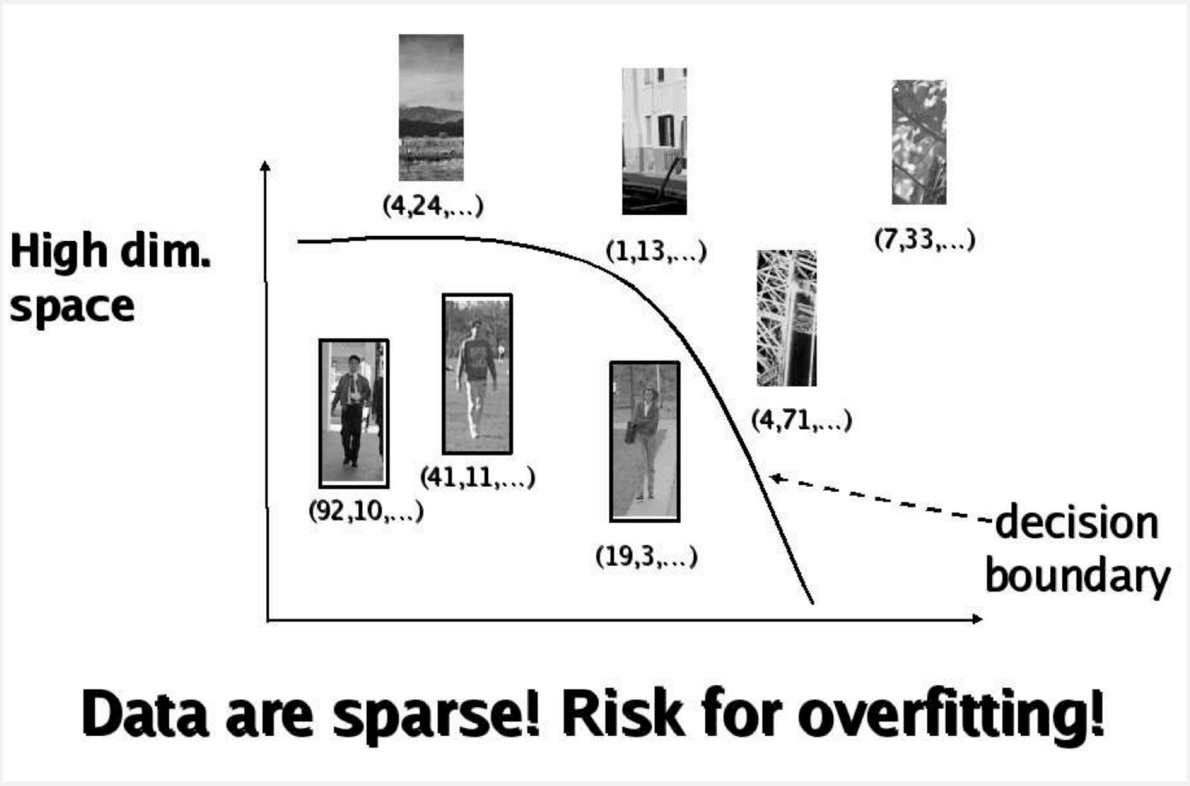
Learning approach

- Learning attempts to infer the algorithm for a set of (labelled) examples in much the same way that children learn by being shown a set of examples (eg sports/non sports car).
- Attempts to isolate underlying structure from a set of examples. Approach should be
  - stable: finds something that is not chance part of set of examples
  - efficient: infers solution in time polynomial in the size of the data
  - robust: should not be too sensitive to mislabelled/noisy examples

People Detection Example



People detection example (cont.)



Supervised Learning Model

- Goal: Given training data (pattern,target) pairs

$$S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$$

infer a function f_S such that

$$f_S(\boldsymbol{x}_i) \approx y_i$$

for the future data

$$S' = \{(\boldsymbol{x}_{m+1}, y_{m+1}), (\boldsymbol{x}_{m+2}, y_{m+2}), \ldots\}.$$

- Classification : $y \in \{-1, +1\}$; Regression : $y \in \mathbb{R}$
- $\mathcal{X}$: input space (eg, $\mathcal{X} \subseteq \mathbb{R}^d$), with elements $\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{x}_i, \ldots$
- $\mathcal{Y}$: output space, with elements $y, y', y_i, \ldots$

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

Learning algorithm

- Training set: $S = \{(\boldsymbol{x}_i, y_i)_{i=1}^m\} \subseteq \mathcal{X} \times \mathcal{Y}$
- A **learning algorithm** is a mapping $S \mapsto f_S$
- A new input $\boldsymbol{x}$ is predicted as $f_S(\boldsymbol{x})$

Example Algorithms

- Linear Regression
- Neural Networks
- Decision Trees
- Support Vector Machines

- 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 neighbours

Some Important Questions

- How is the data **collected**? (need assumptions!)
- How do we **represent** the inputs? (may require preprocessing step)
- How **accurate** is $f_{\mathcal{S}}$ 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, $f_{\mathcal{S}}$ and $g_{\mathcal{S}}$ 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

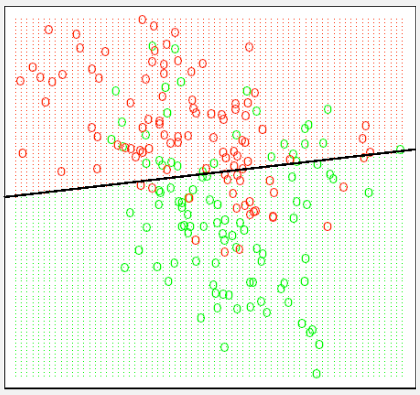
More Examples / Applications

- 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: $\mathcal{X} = \mathbb{R}^2$, $\boldsymbol{x} = (x_1, x_2)$ and $\mathcal{Y} = \{green, red\}$
Our first learning algorithm computes a **linear function** (linear regression), $\boldsymbol{w}^\top \boldsymbol{x} + b$ and classifies an input $\boldsymbol{x}$ as

$$f(\boldsymbol{x}) = \begin{cases} \texttt{red} & \boldsymbol{w}^\top \boldsymbol{x} + b > 0 \\ \texttt{green} & \boldsymbol{w}^\top \boldsymbol{x} + b \leq 0 \end{cases}$$



Linear Regression (Least Squares)

- Emerged in response to problems in Astronomy and Navigation
- Motivated by the need to combine multiple noisy measurements
- Method first described by Gauss in 1794

A Simple Problem – 1

Given the data set

$$S = \{((1,1),3),((2,3),7)\}$$

Then with the new input $\boldsymbol{x}_3 = (4,2)$

how should we predict y_3?

Why? What Assumptions?

A Simple Problem – 2

Model as a system of equations

$$\begin{aligned} w_1 + w_2 &= 3 \\ 2w_1 + 3w_2 &= 7 \end{aligned}$$

or more directly as

$$X\boldsymbol{w} = \mathbf{y}$$

with

$$X = \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}; \qquad \mathbf{y} = \begin{pmatrix} 3 \\ 7 \end{pmatrix}$$

A Simple Problem – 3

Solving in matlab

$$\boldsymbol{w} = X^{-1}\mathbf{y}$$

```
>> y = [3 ; 7]
y = 3
    7
>> X = [1,1 ; 2, 3]
X = 1      1
    2      3
>> XI = X^(-1)
XI = 3     -1
    -2      1
>> w= XI * y
w = 2
    1
>> w= X \ y %% More efficient than calculating inverse use in practice see help mldivide
w = 2
    1
```

We now have the linear predictor

$$\hat{y} = \boldsymbol{w} \cdot \boldsymbol{x}$$

Thus predict $\hat{y}_3 = \boldsymbol{w} \cdot \boldsymbol{x}_3 = w_1 x_{3,1} + w_2 x_{3,2} = 4 \times 2 + 1 \times 2 = 10.$

A Simple Problem – 4

What if?

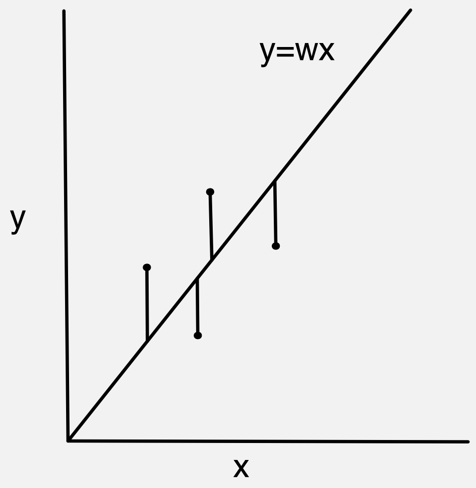
- Overdetermined:

$$S = \{((1,1),3),((2,3),7),((2,1),3)\}$$

- Underdetermined:

$$S = \{((1,1,2),3),((2,4,3),7)\}$$

Minimize square error – 1



Find a linear predictor $\hat{y} = \boldsymbol{w} \cdot \boldsymbol{x}$ to minimize the square error over the data $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$ thus

$$\text{Minimize: } \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{m} (y_i - \boldsymbol{w} \cdot \boldsymbol{x}_i)^2$$

Minimize square error – 2

Thus given,

$$\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$$

with $\boldsymbol{x} \in \mathbb{R}^n$ we may represent the pattern and target vectors with the matrices

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,n} \\ x_{2,1} & x_{2,2} & \ldots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \ldots & x_{m,n} \end{pmatrix} \text{ and } \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

MSE in Matrix Notation

Thus in matrix notation the *empirical* mean (square) error of the linear predictor $\hat{y} = \boldsymbol{w} \cdot \boldsymbol{x}$ on the data sequence $\mathcal{S}$ is

$$\begin{aligned} \mathcal{E}\text{emp}(\mathcal{S}, \boldsymbol{w}) &= \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \\ &= \frac{1}{m} \sum_{i=1}^{m} (y_i - \boldsymbol{w} \cdot \boldsymbol{x}_i)^2 \\ &= \frac{1}{m} \sum_{i=1}^{m} (y_i - \sum_{j=1}^{n} w_j x_{i,j})^2 \\ &= \frac{1}{m} (X\boldsymbol{w} - \mathbf{y})^\top (X\boldsymbol{w} - \mathbf{y}) \end{aligned}$$

MSE minimization. General Case

To compute the minimum we solve for

$$\nabla_{\boldsymbol{w}} \mathcal{E}_{\text{emp}}(\mathcal{S}, \boldsymbol{w}) = \mathbf{0}.$$

recalling that

$$\nabla_{\boldsymbol{w}} = \begin{pmatrix} \frac{\partial}{\partial w_1} \\ \vdots \\ \frac{\partial}{\partial w_n} \end{pmatrix}$$

Thus we need to solve,

$$\begin{aligned} \nabla_{\boldsymbol{w}} \left[(X\boldsymbol{w} - \mathbf{y})^T (X\boldsymbol{w} - \mathbf{y}) \right] &= \mathbf{0} \\ \left(\sum_{i=1}^{m} \frac{\partial}{\partial w_1} \left(\sum_{j=1}^{n} X_{ij} w_j - y_i \right)^2, \ldots, \sum_{i=1}^{m} \frac{\partial}{\partial w_n} \left(\sum_{j=1}^{n} X_{ij} w_j - y_i \right)^2 \right)^\top &= \mathbf{0} \end{aligned}$$

Normal equations

Consider the 2-d case ($n = 2$)

$$\mathcal{E}_{\text{emp}}(\mathcal{S}, \boldsymbol{w}) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2$$

Note that:

$$\frac{\partial \mathcal{E}_{\text{emp}}(\mathcal{S}, \boldsymbol{w})}{\partial w_k} = \frac{2}{m} \sum_{i=1}^{m} (\boldsymbol{w}^\top \boldsymbol{x}_i - y_i) \frac{\partial (\boldsymbol{w}^\top \boldsymbol{x}_i)}{\partial w_k} = \frac{2}{m} \sum_{i=1}^{m} (\boldsymbol{w}^\top \boldsymbol{x}_i - y_i) x_{ik}$$

Hence, to find $\boldsymbol{w} = (w_1, w_2)^\top$ we need to solve the *linear system* of equations

$$\sum_{i=1}^{m} (x_{ik} x_{i1} w_1 + x_{ik} x_{i2} w_2) = \sum_{i=1}^{m} x_{ik} y_i, \quad k = 1, 2$$

Normal equations (cont.)

In vector notations:

$$\sum_{i=1}^{m} \boldsymbol{x}_i \boldsymbol{x}_i^\top \boldsymbol{w} = \sum_{i=1}^{m} \boldsymbol{x}_i y_i$$

In matrix notation:

$$X^\top X \boldsymbol{w} = X^\top \mathbf{y}$$

where

$$X^\top = \begin{bmatrix} x_{11} & \cdots & x_{m1} \\ \vdots & \ddots & \vdots \\ x_{1n} & \cdots & x_{mn} \end{bmatrix} \equiv \left[\boldsymbol{x}_1, \cdots, \boldsymbol{x}_m\right], \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

Least square solution

$$X^\top X \boldsymbol{w} = X^\top \mathbf{y}$$

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

$$\boldsymbol{w} = (X^\top X)^{-1} X^\top \mathbf{y}$$

Otherwise, the solution may not be unique...

Comment:
In matlab use

$$w = X \backslash y$$

Going back to “b” (adding a bias term)

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

$$\begin{aligned}(\mathbf{X}^\top\mathbf{X})\boldsymbol{w} + \mathbf{X}^\top\mathbf{1}b &= \mathbf{X}^\top\mathbf{y}\\ \mathbf{1}^\top\mathbf{X}\boldsymbol{w} + mb &= \mathbf{1}^\top\mathbf{y}\end{aligned}$$

$$\text{that is} \qquad \begin{bmatrix}\mathbf{X}^\top\mathbf{X} & \mathbf{X}^\top\mathbf{1}\\ \mathbf{1}^\top\mathbf{X} & m\end{bmatrix}\begin{bmatrix}\boldsymbol{w}\\ b\end{bmatrix} = \begin{bmatrix}\mathbf{X}^\top\mathbf{y}\\ \mathbf{1}^\top\mathbf{y}\end{bmatrix}$$

where $\mathbf{1} = (1, 1, \ldots, 1)^\top$, $m \times 1$ vector of “ones”

MSE minimization. An easy example – 1

Suppose we are given the data

$$\mathcal{S} = \{(1, y_1), (1, y_2), \ldots, (1, y_m)\}$$

i.e. $\boldsymbol{x}_1 = \boldsymbol{x}_2 \ldots \boldsymbol{x}_m = 1$.

What is the interpretation of the $\boldsymbol{w}$ that minimizes $\mathcal{E}_{\text{emp}}(\mathcal{S}, \boldsymbol{w})$?

MSE minimization. An easy example – 2

Given (w is a scalar as the data are 1-d)

$$\mathcal{E}\text{emp}(\mathcal{S}, w) = \frac{1}{m}\sum_{i=1}^{m}(y_i - w)^2$$

to compute the minimum we solve for

$$\frac{\partial}{\partial w}\mathcal{E}\text{emp}(\mathcal{S}, w) = 0.$$

Solving for w we have

$$\frac{\partial}{\partial w}\frac{1}{m}\sum_{i=1}^{m}(y_i - w)^2 = -\frac{1}{m}2\sum_{i=1}^{m}(y_i - w) = 0,$$

hence $w = \frac{1}{m}\sum_{i=1}^{m} y_i$ is the minimizer.

A different approach: $k-$nearest neighbours

Let $N(\boldsymbol{x}; k)$ be the set of k nearest training inputs to $\boldsymbol{x}$ and

$$I_{\boldsymbol{x}} = \{i : \boldsymbol{x}_i \in N(\boldsymbol{x}; k)\}$$

the corresponding index set

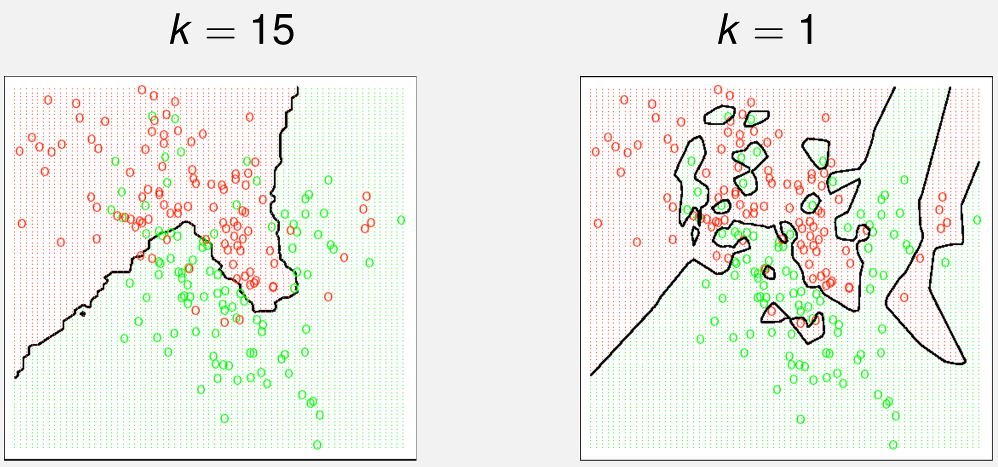
$$f(\boldsymbol{x}) = \begin{cases} \texttt{red} & \text{if } \frac{1}{k}\sum_{i \in I_{\boldsymbol{x}}} y_i > \frac{1}{2} \\ \texttt{green} & \text{if } \frac{1}{k}\sum_{i \in I_{\boldsymbol{x}}} y_i \leq \frac{1}{2} \end{cases}$$

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

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

$k-$NN: the effect of k

- The smaller k the more irregular the decision boundary



- 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!

Optimal Supervised Learning

Model: We assume that the data is obtained by sampling **i.i.d.** from a **fixed but unknown** probability density $P(\boldsymbol{x}, y)$
Expected error:

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

Our goal is to minimize $\mathcal{E}$
Optimal solution: $f^* := \mathrm{argmin}_f \mathcal{E}(f)$ (called *Bayes estimator*)

Problem A: 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 $\mathcal{Y} = \mathbb{R}$.
Using the decomposition $P(y, \boldsymbol{x}) = P(y|\boldsymbol{x})P(\boldsymbol{x})$ we have

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

So we see that f^* (called the regression function) is

$$f^*(\boldsymbol{x}) = \mathrm{argmin}_{c \in \mathbb{R}} \int_{\mathcal{Y}} (y - c)^2 dP(y|\boldsymbol{x}) = \int_{\mathcal{Y}} y dP(y|\boldsymbol{x})$$

Bayes classifier

The *Bayes classifier* (estimator) is the minimiser of the expected loss

- for C-class classification, f^* (called the Bayes classifier) is

$$f^*(\boldsymbol{x}) := \operatorname*{argmax}_{c \in \{1,\ldots,C\}} P(Y = c|\boldsymbol{x})$$

The *Bayes error rate* (optimal) is then

$$\int (1 - P(Y = f^*(\boldsymbol{x})|\boldsymbol{x}) dP(\boldsymbol{x})$$

Revisiting k-NN

k-NN attempts approximate $P(Y = c|\boldsymbol{x})$ as $\frac{|\{i : y_i = c, i \in I_x\}|}{k}$

- Expectation is replaced by averaging over sample data
- Conditioning at $\boldsymbol{x}$ is relaxed to conditioning on some region close to $\boldsymbol{x}$

As the number of samples goes to infinity ($m \to \infty$) 1-NN and k-NN become “good” estimators.

1-NN is near asymptotically optimal

Theorem

As the number samples goes to infinity the error rate is no more than twice the Bayes error rate.

Proof Sketch

Observe that as the number samples goes to infinity, $m \to \infty$,

$$P(c|\boldsymbol{x}) \approx P(c|\boldsymbol{x}_{nn})$$

Thus the expected rate of 1-NN is

$$\sum_{c=1}^{C} P(c|\boldsymbol{x})[1 - P(c|\boldsymbol{x})]$$

We need to show

$$\sum_{c=1}^{C} P(c|\boldsymbol{x})[1 - P(c|\boldsymbol{x})] \leq 2[1 - \max_{c\in\{1,\ldots,C\}} P(c|\boldsymbol{x})]$$

Proof Sketch – continued

Proof Sketch – continued

Let $c^* = \text{argmax}_{c\in\{1,\ldots,C\}} P(c|\boldsymbol{x})$ and $p^* = P(c^*|\boldsymbol{x})$. Observe that

$$\begin{aligned}
\sum_{c=1}^{C} P(c|\boldsymbol{x})[1 - P(c|\boldsymbol{x})] &= \sum_{c\neq c^*}^{C} P(c|\boldsymbol{x})[1 - P(c|\boldsymbol{x})] + p^*(1-p^*) \\
&\leq (C-1)\frac{1-p^*}{C-1}[1 - \frac{1-p^*}{C-1}] + p^*(1-p^*) \\
&= (1-p^*)[1 - \frac{1-p^*}{C-1} + p^*]
\end{aligned}$$

Where the second line follows since the sum is maximised when all "$P(c|\boldsymbol{x})$" have the same value. And since $p^* < 1$ we are done. $\square$

k-NN is near asymptotically optimal

One can show that $\mathcal{E}(k - NN) \to \mathcal{E}(f^*)$ as $m \to \infty$ provided that:

1. $k(m) \to \infty$
2. $\frac{k(m)}{m} \to 0$

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

Reference
Cover & Hart : *Nearest Neighbor Pattern Classification*, 1967

Solving the “Problem A” (revisiting least squares)

$P(\boldsymbol{x}, y)$ is unknown $\Rightarrow$ cannot compute $f^* = \mathrm{argmin}_f \mathcal{E}(f)$
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

$$\mathcal{E}\mathrm{emp}(\mathcal{S}, f) = \frac{1}{m}\sum_{i=1}^{m}(y_i - f(\boldsymbol{x}_i))^2$$

Problem B: If we minimize $\mathcal{E}_{\mathrm{emp}}$ over all possible functions, we can always find a function with zero empirical error!
Why is this a problem?

Solving the “Problem B”

A Proposed solution: we introduce a restricted space of functions $\mathcal{H}$ called the **hypothesis space**

We minimize $\mathcal{E}\mathrm{emp}(\mathcal{S}, f)$ within $\mathcal{H}$. That is, our learning algorithm is:

$$f_S = \mathrm{argmin}_{f \in \mathcal{H}} \mathcal{E}\mathrm{emp}(\mathcal{S}, f)$$

This approach is usually called **empirical error (risk) minimization**

For example (Least Squares) :

$$\mathcal{H} = \{f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} : \boldsymbol{w} \in \mathbb{R}^n\}$$

Problem C: How do we choose a space $\mathcal{H}$ (discuss later)?

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 $\mathcal{E}_{\mathrm{emp}}$ in some restricted space of functions (eg, linear)
    - Compute local approximation of f^* (k-NN)
    - Estimate P and then use Bayes rule...

Perspectives

Theoretical / methodological aspects involved in supervised learning

- Function representation and approximation – to describe $\mathcal{H}$
- Optimization/numerical methods – to compute f_S
- Probabilistic methods – to study generalization error of f_S or to infer the likelihood of f_S

Additive noise model

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

$$y = f(\boldsymbol{x}) + \epsilon$$

where ϵ is a zero mean r.v. Hence we can write

$$P(y, \boldsymbol{x}) = P(y|\boldsymbol{x})P(\boldsymbol{x}) = P_\epsilon\big(y - f(\boldsymbol{x})\big)P(\boldsymbol{x})$$

where $\mathbf{E}[\epsilon] = 0$
Noise free model: y deterministically computed from $\boldsymbol{x}$ ($\epsilon \equiv 0$)

Additive noise model (cont.)

$$y = f(\boldsymbol{x}) + \epsilon$$

$$P(y, \boldsymbol{x}) = P(y|\boldsymbol{x})P(\boldsymbol{x}) = P_\epsilon\big(y - f(\boldsymbol{x})\big)P(\boldsymbol{x})$$

The training data is obtained, for $i = 1, \ldots, m$ as follows

- sample $\boldsymbol{x}_i$ from $P_{\boldsymbol{x}}$
- sample ϵ_i from P_ϵ
- set $y_i = f(\boldsymbol{x}_i) + \epsilon_i$

Note: $P(\boldsymbol{x}) \equiv P_x(\boldsymbol{x})$ (just use different notation when needed)

Additive noise model (cont.)

$$P(y, \boldsymbol{x}) = P(y|\boldsymbol{x})P_{\boldsymbol{x}}(\boldsymbol{x}) = P_\epsilon\big(y - f(\boldsymbol{x})\big)P_{\boldsymbol{x}}(\boldsymbol{x})$$

So, since ϵ has zero mean, we have that

$$f^*(\boldsymbol{x}) := \int y dP(y|\boldsymbol{x}) = \int y dP_\epsilon\big(y - f(\boldsymbol{x})\big) = \int (f(\boldsymbol{x}) + \epsilon) dP_\epsilon(\epsilon) = f(\boldsymbol{x})$$

A common choice for the noise distribution P_ϵ 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 given the underlying function is f?

$$\begin{aligned} P(S|f) &= \prod_{i=1}^{m} P(y_i, \boldsymbol{x}_i|f) = \prod_{i=1}^{m} P(y_i|\boldsymbol{x}_i, f)P(\boldsymbol{x}_i) \\ &= \prod_{i=1}^{m} P(\boldsymbol{x}_i) \prod_{i=1}^{m} P(y_i|\boldsymbol{x}_i, f) = A \prod_{i=1}^{m} P(y_i|\boldsymbol{x}_i, f) \end{aligned}$$

where

$$A = \prod_{i=1}^{m} P(\boldsymbol{x}_i) = P(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)$$

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 linear functions and additive Gaussian noise, we have

$$L(\boldsymbol{w}; S) = A \prod_{i=1}^{m} \left(2\pi\sigma^2\right)^{-\frac{1}{2}} \exp\left\{-\frac{(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2}{2\sigma^2}\right\}$$

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

$$\log L(\boldsymbol{w}; S) = -\frac{1}{2\sigma^2} \sum_{i=1}^{m} \left(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i\right)^2 + \mathit{const}$$

Hence maximizing the likelihood is equivalent to least squares!

Choosing a Hypothesis Space (returning to prob. “C”)

Given the training data $y_i = f^*(\boldsymbol{x}_i) + \epsilon_i$, the goal is to compute an “approximation” of f^*.

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)

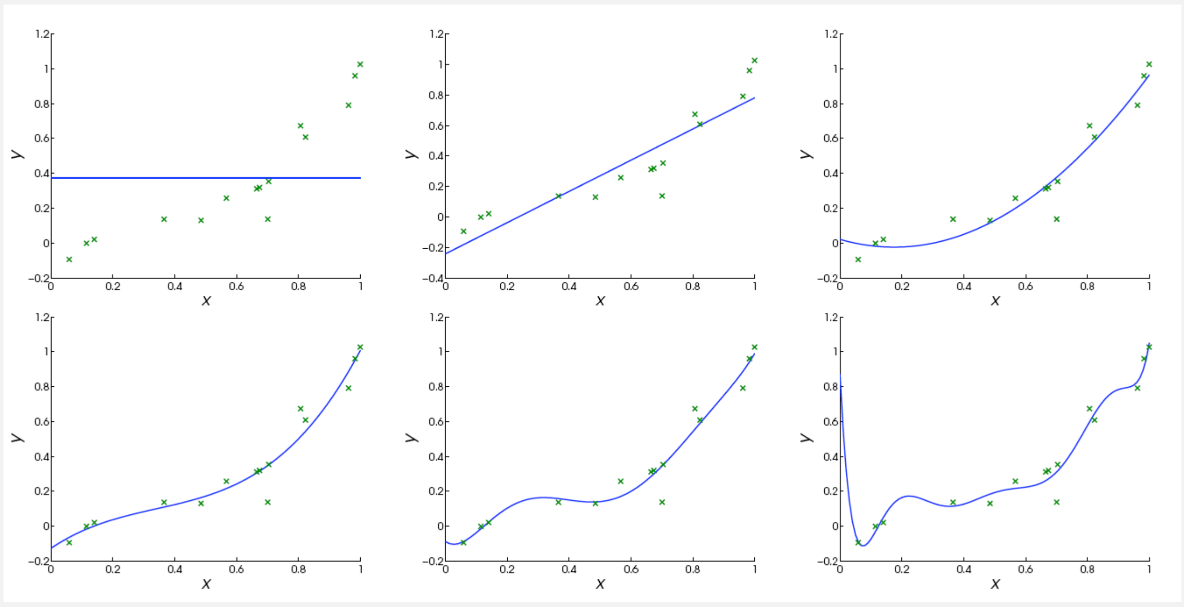
Polynomial fitting

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

$$
\begin{aligned}
H_0 &= \left\{ f(x) = b : b \in \mathbb{R} \right\} \\
H_1 &= \left\{ f(x) = ax + b : a, b \in \mathbb{R} \right\} \\
H_2 &= \left\{ f(x) = a_1 x + a_2 x^2 + b : a_1, a_2, b \in \mathbb{R} \right\} \\
&\vdots \\
H_n &= \left\{ f(x) = \sum_{\ell=1}^{n} a_\ell x^\ell + b : a_1, \ldots, a_n, b \in \mathbb{R} \right\}
\end{aligned}
$$

Consider minimizing the empirical error in $\mathcal{H}_r$ ($r =$ “polynomial degree”)

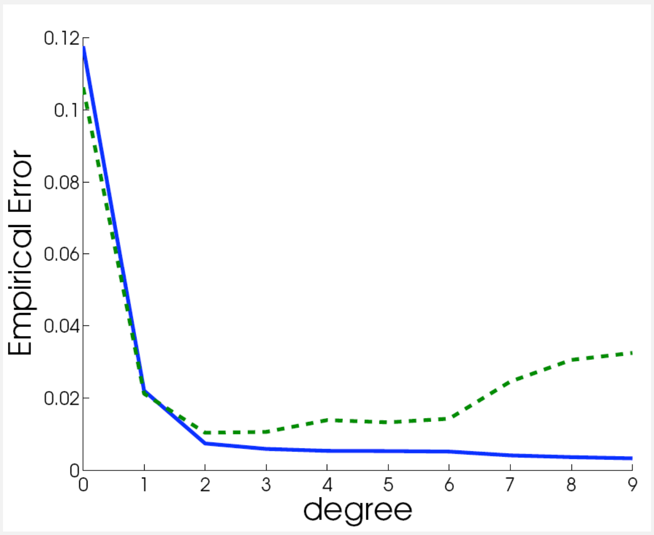
Polynomial fitting (simulation)



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

Overfitting vs. Underfitting

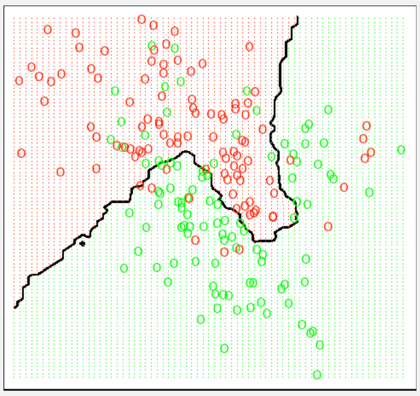
- Compare the empirical error (solid line) with expected error (dashed line)
    - r small: underfitting
    - r large: overfitting
- The larger r the lower the empirical error of f_S! $\Rightarrow$ We cannot rely on the training error!

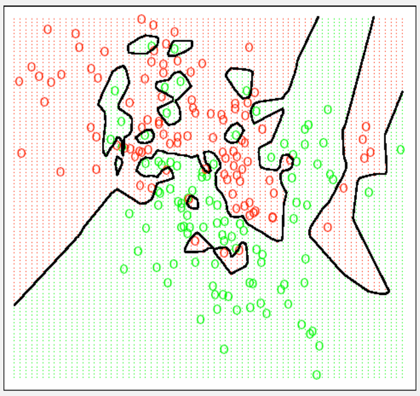


$k-$NN: the effect of k

- The smaller k the more irregular the decision boundary

$k = 15$ $k = 1$

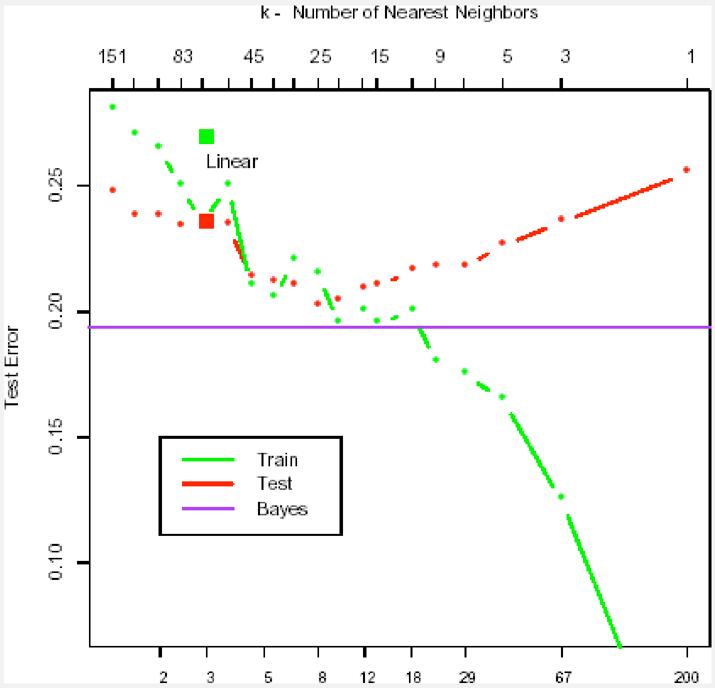




- How to choose k? later...

$k-$NN: the effect of k

$\frac{m}{k}$ large: overfitting versus $\frac{m}{k}$ small: underfitting



Model Selection

1. How to choose k in k-NN?
2. How to choose the degree r for polynomial regression?
3. The simplest approach is to use part of the training data (say 2/3) for training and the rest as a validation set for each $\mathcal{H}_r$
4. Another approach is $K-$fold cross-validation – see next slide
5. Then choose the “best” r relative to the error(s) on the validation set
6. We will return to model selection later in the course

Cross-validation

1. we split the data in K parts (of roughly equal sizes)
2. repeatedly train on $K-1$ parts and test on the part “left out”
3. average the errors of K “validation” sets to give so-called cross-validation error
4. smaller K is less expensive but poorer estimate as size of training set is smaller and random fluctuations larger

For a dataset of size m, m -fold cross-validation is referred to as leave-one-out (LOO) testing

Cross-validation comments

- Cross validation is good in “practice.”
- There are a variety of theoretical-based approaches (not covered today)
- Examples
  1. “Bayesian” model selection via the “evidence”
  2. Structural Risk Mininimization

Other learning paradigms

Supervised learning is not the only learning setup!

- **Online learning:** we observe the data sequentially and we make a prediction and update are learner after every datum
- **Active learning:** we are given many inputs and we can choose which ones to request a label
- **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.
- **Semi-supervised learning:** the ‘learning environment” may give us access to many input examples but only few of them are labeled