T-61.5040

Learning Models and Methods, (5 cr) L

Spring 2007

Lecturer: **Petteri Pajunen**

Course Assistant: **Ville Viitaniemi**

LECTURE 2: 24.1.2007

CURSE OF DIMENSIONALITY
OVERFITTING

- NFL theorems $\Longrightarrow$ no information, no learning
- simple, correct model $\Longrightarrow$ easy to solve with enough data
- what about learning methods in practice?
- perhaps some information and lots of data leads to good results?

Weak Assumptions Lead to Local Learning

- lets examine what happens when a learning method is expected to contain "weak" information
- for example, it can be argued that a learning method such as a neural network prefers solutions that are regular in some sense
- but with a complex enough neural network, one can approximate closely almost any solution
- sounds plausible: there is some information (prefer well-behaving solutions) but not too much (many solutions can be closely approximated)

- lets examine how well this approach can be expected to work
- example: regression by kernel density estimation
- predictor of y as a function of x can be solved using the *joint distribution* $p(x, y)$ (exercise)
- the solution minimizes the MSE $\mathrm{E}(|y - \hat{y}|^2)$

- how to estimate $p(x, y)$ from data (x_i, y_i), $i = 1, \ldots, n$
- one way to estimate it is to use a *kernel density estimator*
- a nonnegative localized function with integral $1/n$ is located at each (x_i, y_i)
- the sum of these is the estimate of $p(x, y)$

- replacing $p(x, y)$ by its estimate will give a solution $\hat{y}$ as a function of x and training data
- exact result computed in exercises
- $\hat{y}$ will be a weighted sum of $y_1, \ldots, y_n$
- the weights are large when x_i is close to x and small when x_i is far from x

- the above solution has the property *x is close to x_i $\Longrightarrow$ y is close to y_i*
- methods such as neural networks in general have this property
- for example, a large MLP network is a nonlinear parametric function
- it does not vary very rapidly when input x changes, but is often flexible enough to provide a good fit to training points

- this property seems exactly what we wanted, but it is useful only near the training points
- *local learning* is essentially what was achieved
- i.e. the training data cannot be generalized except in a very small neighbourhood around the training points
- methods such as neural networks do predict outputs arbitrarily far from training points, but in general these predictions are not reliable

- lets examine when local learning works
- demo: *local.R*
- the correct function is x^3, and two high-degree polynomials are fitted to the data
- the polynomials don't vary extremely rapidly and can fit the correct solution exactly
- when 50 points are used, the result is quite good

- with 10 points used for learning, the solutions differ from x^3 a lot
- the only common feature is that the polynomials pass through, or close to the training points
- local learning effectively makes the prediction based on the closest point(s) in the set of observed data
- this is useful only when predicting *near enough* to a training point

- lets consider this geometrically
- training inputs $x_1, \ldots, x_n$ are in a d-dimensional space
- we want to predict the output at some point x in the same space
- local learning can be used if x is close to some x_i

- relevant quantities: n (number of training points), d (dimension of x)
- small d, large n: local learning may be useful
- note that the training data must cover well enough the area where predictions are needed

- when d is even moderately large, unexpected things happen
- this phenomenon is called the Curse of Dimensionality
- CoD is a geometric concept which describes certain properties of high-dimensional spaces
- some examples are given to illustrate this

High-Dimensional Data

- lets examine a d-dimensional vector space (or a bounded subset of it), and n points in it
- for local learning to work, the points must be close enough to each other
- curse of dimensionality $\implies$ n must be *very large* if d is even moderately large

Uniform density over data space

- consider $[0, 1]^d$, the d-dimensional hypercube
- cover it with points having distance 0.1 to the closest neighbour
- if $d = 1$, then ten points achieves this
- but if $d = 10$, we need about 10^{10} points for the same distance
- ten is not a very high dimension in many learning problems!

Most points are close to the sides

- consider $[0,1]^d$, and $[0.1,0.9]^d$ inside of it
- smaller cube seems relatively large: each edge has length 0.8 versus 1 in the larger cube
- volume of $[0,1]^d$ is one: the volume of $[0.1,0.9]^d$ is $(0.9-0.1)^d = (0.8)^d$
- for $d = 20$, this volume is approximately 0.01
- a set can look large in all coordinates separately, but be small in the whole data space

Points are far away from each other

- for n points uniformly distributed in $[0,1]^d$, the expected L_∞ distance of point number 1 to its nearest neighbour is

$$D(d,n) \approx \left(\frac{1}{n}\right)^{1/d}$$

- when d grows but n is fixed, this distance converges to 1 (exact result in exercises)
- *almost all points are near the side, and far from each other*

Points tend to be mutually orthogonal

- generate n points from $N(0,\sigma^2 I_d)$
- compute the inner product of $n-1$ points with the first point x
- compare the inner products with $\|x\|^2$
- result: when d is high, the inner products are small compared to $\|x\|^2$
- demo: `prohist.R`

- CoD $\implies$ local learning is not very useful in high-d spaces
- learning from data requires more information than vague regularity
- note that depending on the problem, seemingly weak information may actually be a strong assumption

- compare regression with classification
- try to predict bits from observed inputs $x \in \mathbb{R}^d$
- this is two-class classification, or regression with binary outputs
- solving this as a regression problem seems impossible without lots of information about the problem

- but as a classification problem a somewhat regular discriminant function if often used
- such a discriminant function makes a strong but usually plausible assumption
- consider two clusters of training data, each containing data from one class only
- solving a classifier defines the classes of points arbitrarily far from training data
- an exercise problem considers $d = 1$ and a "linear" classifier

- so far:
  - NFL: no information $\implies$ no learning
  - weak assumptions $\implies$ local learning, not useful in high-d problems
  - strong assumptions $\implies$ learning is possible, if assumptions are correct

Overfitting

- lets examine what happens if a learning method works by selecting a solution out of a set $\{f(x|\theta)|\theta \in \Theta\}$
- assume the model $f(x|\theta)$ is correct, meaning that data is generated as $y = f(x|\theta_0) + n$ where n is an error term
- many learning methods use training data to compute an estimate $\hat{\theta}$
- the solution is then defined as $f(x|\hat{\theta})$

- estimation requires a *loss function* $L(y, f(x|\theta))$ to measure how good each θ is
- *expected loss (risk)* $R(\theta) = \mathrm{E}(L(y, f(x|\theta)))$
- *observed loss (empirical risk)* $R_{emp}(\theta) = \frac{1}{n}\sum_i L(y_i, f(x_i|\theta))$
- often $\hat{\theta}$ is found by minimizing $R_{emp}(\theta)$

- this loss-framework includes regression, classification, density estimation, ICA etc...
- if $\{f(x|\theta)|\theta \in \Theta\}$ is "small", then minimizing R_{emp} may work
- but many problems require complex models, where the set of solutions is not small
- large set leads to many values of θ minimizing R_{emp}
- called *overfitting*: can't tell solutions apart using only training data

- overfitting can happen even if $f(x|\theta_0)$ is the *correct solution* as defined above
- example: linear model

$$y = \mu x + \beta + n, \quad n \sim N(0,1)$$

- this is a standard linear regression model

- observe outputs y_0, y_1 at inputs $x = 0, x = 1$
- fit the model (find $\hat{\mu}$, $\hat{\beta}$) by minimizing the squared error

$$R_{emp}(\hat{\beta}, \hat{\mu}) = (y_0 - \hat{\beta})^2 + (y_1 - \hat{\mu} - \hat{\beta})^2$$

- the error is simply the sum of squared errors at $x = 0$ and $x = 1$

- predict output at $x = 2$ as $2\hat{\mu} + \hat{\beta}$
- for comparison, fit a *constant model* $y = \beta + n$
- correct prediction is $2\mu + \beta$: we can compare the two models against this
- depending on μ, sometimes the constant model gets a *smaller mean-square error!*
- demo: `overfitting.R`, and an exercise problem

- lets illustrate overfitting geometrically
- assume $\theta \in \mathbb{R}^d$, a d-dimensional space
- consider a training set of m points
- then

$$\underline{y} = (y_1, \ldots, y_m) \in \mathbb{R}^m$$

- each θ is mapped to

$$\underline{y}(\theta) = (f(x_1|\theta), f(x_2|\theta), \ldots, f(x_m|\theta))$$

- this defines a function $f' : \mathbb{R}^d \to \mathbb{R}^m$
- for simplicity, assume that this function is somewhat regular

- suppose $d \gg m$: there are lots of functions, but little data
- f' is a mapping from a higher-dimensional space to a lower-dimensional space
- example: $d = 3, m = 2$ ("flatten" a cube)
- many θ's map into a single $\underline{y}(\theta)$
- since good solution $= f'(\theta)$ close to $\underline{y}$, we find too many solutions

- if $d \ll m$, then the image of f' is a subspace of $\mathbb{R}^m$
- if $\underline{y}$ is not near this subspace, no solution is very good
- this is called *underfitting*
- if the simple model is correct, then underfitting is unlikely

- if $d \sim m$ (saying $d = m$ is too simple), the dimensions "match" and one can generally find a unique solution
- using the correct model is not enough if the learning method estimates a single value θ from training data
- having too few training points can prevent finding it

- lots of heuristic methods have been proposed to select the right size for $\{f(x|\theta)|\theta \in \Theta\}$
- e.g. Minimum Description Length, Structural Risk Minimization
- in most of these, a penalty proportional to model complexity is added to the loss

- other heuristics for avoiding overfitting are *validation* methods
- part of the data is lost, since it is used only in the validation step
- note that validation does not help if you strictly have no information about the learning problem (NFL again)

- the above discussion was about solving the learning problem by selecting a single value for θ (point estimation) by minimizing R_{emp}
- we will not solve the learning problem this way: intention was to demonstrate how it leads to overfitting
- however, many learning methods solve learning problems in the way described above

- point estimation, and the heuristics for avoiding overfitting, have some additional problems
  1. usually no uncertainty in θ: the exact value cannot generally be found using a finite training set
  2. overfitting avoidance leads to using the wrong model
  3. which heuristic to use? different solutions to exactly the same problem
- next week, Bayes approach is develop which will address these problems