Statistical Learning

Due in two weeks on Moodle (March 21 + tolerance)

Homework-01

General Instructions

- You can use *any* programming language you want, as long as your work is runnable/correct/readable. Two examples:
  - **In `R`**: it would be nice to upload a well-edited and working `R Markdown` file (`.rmd`) + its `html` output.
  - **In `Python`**: it would be nice to upload a well-edited and working `Jupyter notebook` (or similia).
- Remember our **policy on collaboration**:

*Collaboration on homework assignments with fellow students is **encouraged**.*
However, such collaboration should be clearly acknowledged, by listing the names of the students with whom you have had discussions concerning your solution.
*You may **not**, however, share written work or code after discussing a problem with others.*
*The solutions should be written by **you**.*

In case of `R`

If you go for `R`, to be sure that everything is working, start `RStudio` and create an empty project called `HW1`. Now open a new `R Markdown` file (`File > New File > R Markdown...`); set the output to `HTML mode`, press `OK` and then click on `Knit HTML`. This should produce a `html`. You can now start editing this file to produce your homework submission.

- For more info on `R Markdown`, check the support webpage: R Markdown from RStudio.
- For more info on how to write math formulas in LaTex: Wikibooks.

Exercice 1: Linear algebra is good...

...but with functions is better!

⇝ Your job ⇜

1. Read my review notes on *Linear Algebra* and study the **Functions as vectors** section (from page 9 on).
2. Depending on your background, things may not be totally clear at this point. Hence, make a post on our Forum where you provide: A) the top-3 take-home messages **you** got from this section, B) a question and/or a comment and/or a code snippet to tell me what is unclear/obscure/worth-further-explanation-from-me.
3. To show you got the basic idea(s) right, do the following:
   - Based on this material, understand and then briefly (but clearly!) explain to me, why and how this "*(orthogonal) series expansion*" idea is just another version of our "*be linear in transformed feature space*" mantra we implemented via polynomials in our very first supervised learning (toy) example.
   - Consider **Example 8** at page 15 and the related code snippets (page 12, 15 and 17; also linked here). Instead of the 6 *linear* approximations at page 17, recycle as much as possible to implement the corresponding 6 *non-linear* approximations (again to the Doppler function) described at the end of Remark 5 (page 14-15).
     Under this *greedy* policy, you reconstruct by considering, **not** the first J coefficients, but the J-largest ones.
   - [bonus4A+] Are you able to write a function (in `R`, `Python`, etc) to compare the L_2-*reconstruction error*

$$\mathrm{dist}_{\mathrm{L}_2}(m, m_J) = \big\|m - m_J\big\|_2 = \sqrt{\int \big(m(x) - m_J(x)\big)^2 \mathrm{d}x} \qquad \big(m(\cdot) \text{ denotes the Doppler function here}\big),$$

   obtained by the *linear* and *non-linear* approximations $m_J(\cdot)$ at different J? If yes, who's better for the Doppler?

Exercise 2: Polynomials are good...

...but smooth piecewise polynomials (a.k.a. splines) are better!

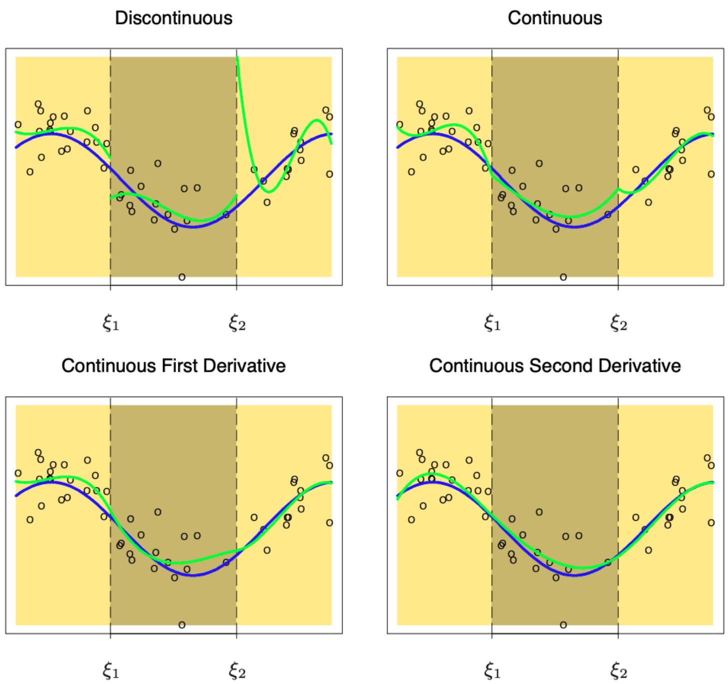
In our initial supervised toy example we tried to recover from samples a *non-linear* (in the original 1-dimensional feature-space) function using a linear model in a higher-dimensional transformed feature-space (polynomials of degree d) Now, polynomials are fine, but they are *global* objects (they span the entire real-line in principle) and may not be able to capture *local* structures of the target function.

RECIPE FOR A SOLUTION: 1. take a polynomial, 2. break it down is small (almost) free-to-move chunks, 3. shake a bit, 4. glue them back together adding some regularity constraint (continuity, differentiability, etc) as needed... a *spline* is born...

MORE FORMALLY: any d^{th}-order spline $f(\cdot)$ is a **piecewise polynomial function** of degree d that is continuous and has continuous derivatives of orders $\{1, \ldots, k-1\}$ at the so called *knot points*. Specifically, how do we build a generic d^{th}-order spline $f(\cdot)$? We start from a bunch of points, say q, that we call *knots* $\xi_1 < \cdots < \xi_q$, and we then ask that...

1. ...$f(\cdot)$ is *some* polynomial of degree d on each of the intervals: $(-\infty, \xi_1], [\xi_1, \xi_2], [\xi_2, \xi_3], \ldots, [\xi_q, +\infty)$;
2. ...its j^{th} derivative $f^{(j)}(\cdot)$ is continuous at $\{\xi_1, \ldots, \xi_q\}$ for each $j \in \{0, 1, \ldots, k-1\}$.

The figure below from Chapter 5 of ELS illustrates the effects of enforcing continuity at the knots, across various orders of the derivative, for a cubic piecewise polynomial.



Splines have some amazing properties, and they have been a topic of interest among statisticians and mathematicians for a very long time (classic VS recent). **But**, given a set of points $\xi_1 < \xi_2 < \cdots < \xi_q$, is there a quick-and-dirty way to describe/generate the whole set of d^{th}-order spline functions over those q knots? The easiest one (**not the best!**), is to start from **truncated power functions** $\mathcal{G}_{d,q} = \{g_1(x), \ldots g_{d+1}(x), g_{(d+1)+1}(x), \ldots, g_{(d+1)+q}(x)\}$, defined as

$$\left\{g_1(x) = 1, g_2(x) = x, \ldots, g_{d+1}(x) = x^d\right\} \text{ and } \left\{g_{(d+1)+j}(x) = (x - \xi_j)_+^d\right\}_{j=1}^q \text{ where } (x)_+ = \max\{0, x\}.$$

Then, if $f(\cdot)$ is a d^{th}-order spline with knots $\{\xi_1, \ldots, \xi_q\}$ you can show it can be obtained as a *linear combinations* over $\mathcal{G}_{d,q}$

$$f(x) = \sum_{j=1}^{(d+1)+q} \beta_j \cdot g_j(x), \text{ for some set of coefficients } \boldsymbol{\beta} = \left[\beta_1, \ldots, \beta_{d+1}, \beta_{(d+1)+1}, \ldots \beta_{(d+1)+q}\right]^{\mathsf{T}}.$$

IDEA: let's perform regression on splines instead of polynomials! In other words, as in our initial toy example, given inputs $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ and responses $\boldsymbol{y} = \{y_1, \ldots, y_n\}$, consider fitting functions $f(\cdot)$ that are d^{th}-order splines with knots at some chosen locations, typically the first q quantiles of $\boldsymbol{x}$ $\rightsquigarrow$ this method is dubbed **regression splines** and it is different from another famous technique called *smoothing splines* (we will talk about it later as an example of *(Mercer) kernel method*).

REMARK: here you have many tuning parameters (the degree d and the number+position of knots) to be selected *predictively* via `Cp` or some flavor of cross-validation (sample-splitting, k-fold `CV`, `LOOCV`, `GCV`). Although there is a large literature on knot selection for regression splines via greedy methods like recursive partitioning, here we will go for an easy-to-implement option.

$\rightsquigarrow$ Your job $\leftsquigarrow$

1. First of all, as before, briefly explain to me why this idea is yet another manifestation of our "*be linear in transformed feature space*" mantra. Do you "perceive" any technical difference with the "*(orthogonal) series expansion*" point of view? Don't be shy, give this question at least a try!

2. Visualize/plot (some of) the elements of $\mathcal{G}_{d,q}$ with $d \in \{1, 3, 5\}$ and $q \in \{3, 5, 10\}$ equispaced knots in the interval $[0, 1]$.

3. Following our polynomial regression example, implement regression splines from scratch on the same data by taking:

   - $d = 3$ (i.e. consider cubic-splines only);
   - knots on q-quantiles of the input vector $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ with $q \in \{3, 5, 10\}$. In `R`, for example:

     ```
     kn = quantile(x, probs = seq(0, 1, length.out = q)
     ```

     You will choose the best between these three q-values via `Cp` and one type of `CV` of your choice.

   Of course, the crucial/boring point is to handmade the $n \times (d + 1) + q$ design matrix $\mathbb{X}$ having generic entry

   $$\mathbb{X}[i, j] = g_j(x_i) \ \text{ for } \ i \in \{1, \ldots, n\} \ \text{ and } \ j \in \big\{1, \ldots, (d + 1) + q\big\}.$$

   After that, you can just use least squares (as implemented in `lm()`, for example) to determine the optimal coefficients $\widehat{\boldsymbol{\beta}} = \big[\widehat{\beta}_1, \ldots, \widehat{\beta}_{d+1}, \widehat{\beta}_{(d+1)+1}, \ldots, \widehat{\beta}_{(d+1)+q}\big]^{\mathsf{T}}$, which then leaves us with the fitted *regression spline*

   $$\widehat{f}(x) = \sum_{j=1}^{(d+1)+q} \widehat{\beta}_j \, g_j(x).$$

4. Now, although we are making no real effort to get a stellar fit, try to (at least qualitatively) compare the best spline fit you got, with our original polynomial predictor.