Statistical Learning

Due Sunday, April 10 on Moodle

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).

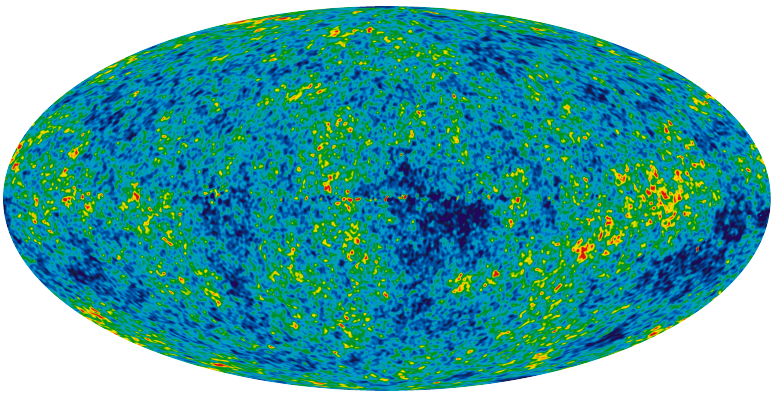
In case of `R`

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.

The Data

For this first homework you'll be working on the WMAP data described at page 45 (Example 4.4) of our purple book. We are talking about a snapshot of our universe in its *infancy*, something like 379,000 years after the Big Bang, nothing compared to the estimated age of our universe[1].



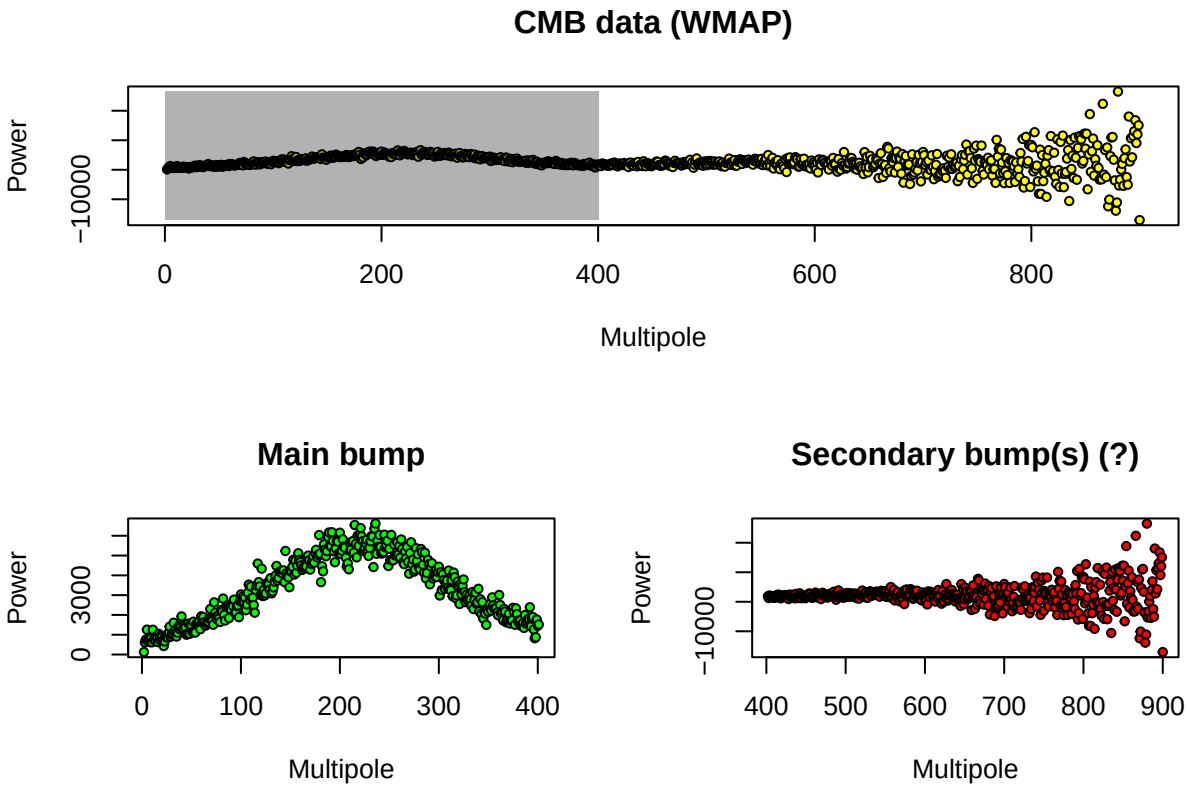
The map above was taken by the Wilkinson Microwave Anisotropy Probe (WMAP) and shows differences across the sky in the temperature of the cosmic microwave background (CMB), the radiant heat remaining from the Big Bang. The average temperature is 2.73 degrees above absolute zero but the temperature is not constant across the sky. The fluctuations in the temperature map provide information about the early universe. Indeed, as the universe expanded, there was a tug of war between the force of expansion and contraction due to gravity. This caused acoustic waves in the hot gas, which is why there are temperature fluctuations.

The strength of the temperature fluctuations $f(x)$ at each frequency (or multipole) x is called the **power spectrum** and this power spectrum can be used by cosmologists to answer cosmological questions. For example, the relative abundance of different constituents of the universe (such as baryons and dark matter) corresponds to **peaks** in the power spectrum. Through a complicated procedure (not described here), the temperature map can be reduced to the following scatterplot of power versus frequency:

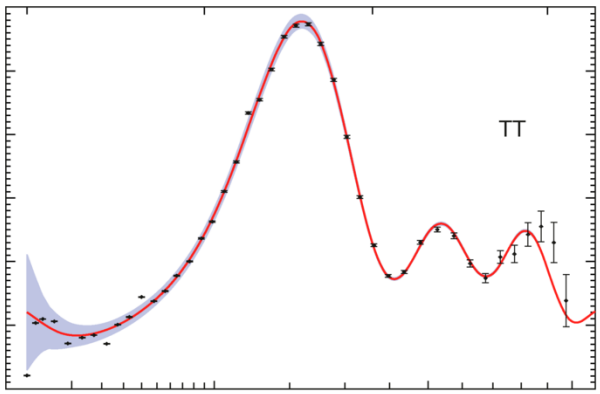
```
# Load the data
wmap <- read.csv("~/wmap.csv")
# Plot
```

[1] ...something like 14 *billion* years!

```
lay.mat = matrix(c(1,1,2,3), nrow = 2, byrow = T)
layout(lay.mat)
# All
plot(wmap$x, wmap$y, pch = 21, bg = "yellow", cex = .7,
     main = "CMB data (WMAP)", xlab = "Multipole", ylab = "Power")
polygon(c(0,400,400,0),
        c(min(wmap$y), min(wmap$y), max(wmap$y), max(wmap$y)),
        border = NA, col = rgb(0,0,0,.3))
# First bump
plot(wmap$x[1:400], wmap$y[1:400], pch = 21, bg = "green", cex = .7,
     main = "Main bump", xlab = "Multipole", ylab = "Power")
# Secondary bump(s)
plot(wmap$x[-(1:400)], wmap$y[-(1:400)], pch = 21, bg = "red", cex = .7,
     main = "Secondary bump(s) (?)", xlab = "Multipole", ylab = "Power")
```



The horizontal axis is the multipole moment, essentially the frequency of fluctuations in the temperature field of the CMB. The vertical axis is the power or strength of the fluctuations at each frequency. The top plot shows the full data set. The two bottom plots zoom in the first 400 and the next 500 data points. The 1^{st} peak, around $x \approx 200$, is obvious. But many "official" fit (see **Fig. 2** in that page) show also a 2^{nd} and 3^{rd} peak further to the right...



...are they really there? Does the data support (without further a priori info) these secondary features?

Part I: Polynomials are good...

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

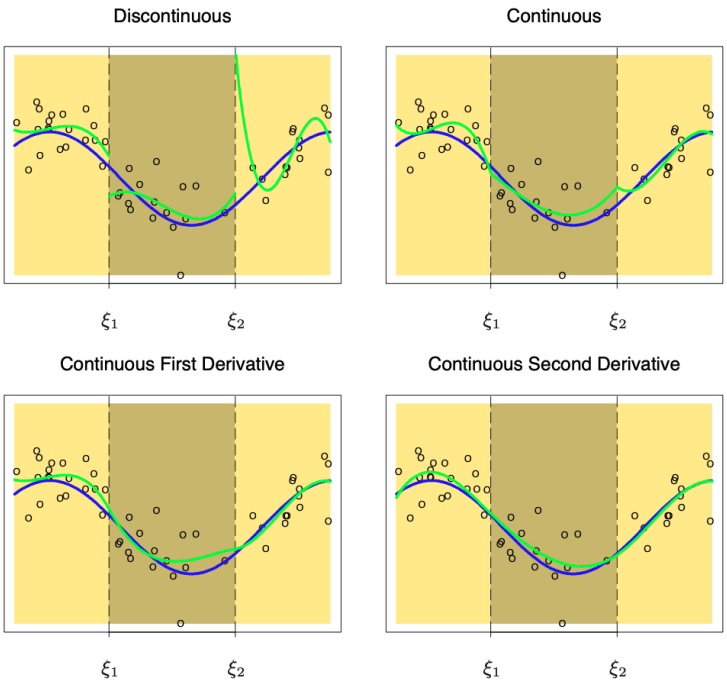
In our initial supervised toy example we tried to recover from samples a *non-linear* (in the original 1-dimensional feature-space) function using a linear model in a higher-dimensional transformed feature-space (polynomials of degree d) Now, polynomials are fine, but they are *global* objects (they 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 $\{1, \dots, d-1\}$ 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: $(-\infty, \xi_1], [\xi_1, \xi_2], [\xi_2, \xi_3], \dots, [\xi_q, +\infty)$;
2. ... its j^{th} derivative $f^{(j)}(\cdot)$ is continuous at $\{\xi_1, \dots, \xi_q\}$ for each $j \in \{0, 1, \dots, d-1\}$.

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** $\mathcal{G}_{d,q} = \{g_1(x), \dots g_{d+1}(x), g_{(d+1)+1}(x), \dots, g_{(d+1)+q}(x)\}$, defined as

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

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

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

IDEA: let's perform regression on splines instead of polynomials! In other words, as in our initial toy example, given inputs $\boldsymbol{x} = \{x_1, \dots, x_n\}$ and responses $\boldsymbol{y} = \{y_1, \dots, y_n\}$, 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.

⇝ Your job ⇜

1. First of all, 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 described in our "extended" Linear Algebra notes (from page 9)? Don't be shy, give this question at least a try!
2. Plot a few elements of $\mathcal{G}_{d,q}$ with $d \in \{1, 3\}$ and $q \in \{3, 10\}$ equispaced knots in the open interval $(0, 1)$.
3. Following our polynomial regression example, implement regression splines from scratch on the `wmap` data by considering:
   - $d \in \{1, 3\}$ (i.e. linear and cubic-splines only);
   - knots on q-equispaced locations (within the x-range) $\rightsquigarrow$ grid-search between a min value $q_{\texttt{min}}$ and a max value $q_{\texttt{max}}$ of your choosing.

   You will choose the best (d, q)-combination via `GCV` and any flavor of `CV` you like. 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, after having visualized the fit and although we are making no real effort to get a stellar fit, try to (at least qualitatively) compare the best spline fit you got so far with a `GCV`-tuned polynomial regression. Which one do you prefer? Explain.

Part II: To be parametric or not to be, this is (another) dilemma...

The Theory

An important use of **nonparametric** regression is in testing whether a **parametric** regression model is well–specified or not.

Assume that our **parametric** regression model is given by

$$y_i = f(x_i \,|\, \boldsymbol{\theta}) + \epsilon_i \quad \text{where} \quad \epsilon_i \overset{\text{I.I.D.}}{\sim} \mathtt{N}(0, \sigma^2) \quad \forall\, i \in \{1, \ldots, n\}, \tag{1}$$

and $f(\cdot \,|\, \boldsymbol{\theta})$ is some (possibly **nonlinear**) function completely specified up to the parameter vector $\boldsymbol{\theta}$.
If, for example, $f(\cdot \,|\, \boldsymbol{\theta})$ describes a linear model, that is $f(x \,|\, \alpha, \beta) = \alpha + \beta\, x$ how can we check whether it is appropriate or not?
One approach would be to add specific departures from linearity, a quadratic term say, and seeing whether the coefficients that go with those terms are significantly non–zero, or whether the improvement in fit is relevant/significant.
But our flexible *nonparametric* regression, by its very definition, allows you to check for all kinds of possible departures, rather than just a particular one. Hence, the idea could go as follows:

- **IF** the parametric model is **correct**[2] it should predict better than – or at least as well as – the nonparametric one. Consequently we expect the following mean–squared error difference between the two models

  $$T = \mathtt{MSE}_{\mathtt{p}}\big(\boldsymbol{\theta}\big) - \mathtt{MSE}_{\mathtt{np}}\big(f\big)$$

  to be sufficiently small.

T is clearly our test statistics. As usual, we have no hope of working out analytically its distribution under the null $\mathtt{H}_0$. So, in the following, we will resort to a particular **parametric bootstrap**[3] strategy.

Here is the procedure in its generality:

0. Get the data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$.
1. Fit the **parametric** model to get an estimate $\widehat{\boldsymbol{\theta}}$. Let $\mathtt{MSE}_{\mathtt{p}}\big(\widehat{\boldsymbol{\theta}}\big)$ be the in–sample (or training) mean–squared error; that is

   $$\mathtt{MSE}_{\mathtt{p}}\big(\widehat{\boldsymbol{\theta}}\big) = \frac{1}{n}\sum_{i=1}^{n}\Big(y_i - f\big(x_i \,|\, \widehat{\boldsymbol{\theta}}\big)\Big)^2 = \mathtt{mean}\left(\mathtt{residuals}^2\right).$$

2. Fit your favorite **nonparametric** regression by, for example, cross–validation, getting curve $\widehat{f}(\cdot)$ and in–sample mean–squared error $\mathtt{MSE}_{\mathtt{np}}\big(\widehat{f}\big)$; that is

   $$\mathtt{MSE}_{\mathtt{np}}\big(\widehat{f}\big) = \frac{1}{n}\sum_{i=1}^{n}\Big(y_i - \widehat{f}(x_i)\Big)^2 = \mathtt{mean}\left(\mathtt{residuals}^2\right).$$

3. Calculate: $\widehat{t} = \mathtt{MSE}_{\mathtt{p}}\big(\widehat{\boldsymbol{\theta}}\big) - \mathtt{MSE}_{\mathtt{np}}\big(\widehat{f}\big)$.
4. Simulate from the parametric model in Equation (1) by taking $\boldsymbol{\theta} = \widehat{\boldsymbol{\theta}}$ to get fake data under an *approximate* null: $\mathcal{D}' = \{(x'_1, y'_1), \ldots, (x'_n, y'_n)\}$.
5. Fit the **parametric** model to the simulated data $\mathcal{D}'$ getting estimate $\widetilde{\boldsymbol{\theta}}$ and $\mathtt{MSE}_{\mathtt{p}}\big(\widetilde{\boldsymbol{\theta}}\big)$.
6. Fit the **nonparametric** model to the simulated data $\mathcal{D}'$ getting estimate $\widetilde{f}(\cdot)$ and $\mathtt{MSE}_{\mathtt{np}}\big(\widetilde{f}\big)$.
7. Calculate: $\widetilde{t} = \mathtt{MSE}_{\mathtt{p}}\big(\widetilde{\boldsymbol{\theta}}\big) - \mathtt{MSE}_{\mathtt{np}}\big(\widetilde{f}\big)$.
8. Repeat steps 4–7 B times to get an *estimate* of the distribution of the test statistics T **under the null**. Please notice that this step takes some time because each replication involves not just generating a new simulation sample, but also cross-validation to pick a bandwidth!
9. The p-value is then: $\frac{1}{B}\sum_{b=1}^{B}\mathbb{I}\big(\widetilde{t}_b > \widehat{t}\big) = \big\{\text{proportion of } \widetilde{t}\text{'s larger than } \widehat{t}\big\}$.

[2] This is the null hypothesis $\mathtt{H}_0$: `the parametric model is correct`.
[3] See Section 3.3 page 31 of your book or slide 53 and after of my prerequisite notes

⇝ Your job ⇜

1. Focus your attention on the second part of the data by dropping the first 400 observations and saving the others in a dataframe called `wmap_sb`.

2. Consider a simple linear model $f(x \,|\, \boldsymbol{\theta}) = \theta_0 + \theta_1\, x$.
   Fit this model (in `R` use `lm()`) and call it `lin_fit`.
   Explore a little bit your fit by looking at summary statistics and diagnostic plot (in `R` use `summary(lin_fit)` and `plot(lin_fit)`).
   Plot the data-scatter and add the linear fit.
   Do you think this model is any good? Explain.
   Finally, evaluate the training mean-squared error and store this value into a variable called `MSEp_hat` (in `R`, use the function `residuals()` on `lin_fit` to get the residuals).

3. Fit and tune by using any method you like (`Cp`, or `LOOCV`, or `GCV`, etc) a spline regression model as in Part I to get `f_hat` and its in–sample mean–squared error to be stored in a variable called `MSEnp_hat`.
   Notice that, in `R`, the function `residuals()` can still be applied to extract the model residuals.
   Add the nonparametric model to the plot you made before, and *qualitatively* compare the parametric and the nonparametric fit you've got so far.

4. Calculate: `t_hat = MSEp_hat - MSEnp_hat`

5. `for(b in 1:B)` repeat the following four steps ($B \geqslant 500$):

   a. Simulate a new data set from the fitted parametric model `lin_fit` assuming homoskedastic Gaussian noise.
      The following function does the job, just pass the original `x` vector as second input.
      Explain what it does (i.e. why it makes sense), translate it in Python (if you need), and then use it!

```
# Inputs: linear model (lin_fit), x values at which to simulate (sim_x)
# Outputs: Data frame with columns x and y
sim_lm = function(lin_fit, sim_x) {
  n      = length(sim_x)
  sim_fr = data.frame(x = sim_x)
  sigma  = summary(lin_fit)$sigma
  y_sim  = predict(lin_fit, newdata = sim_fr)
  y_sim  = y_sim + rnorm(n, 0, sigma)            # Add noise
  sim_fr = data_frame(sim_fr, y = y_sim)         # Adds y column
  return(sim_fr)
}
```

   b. Fit the **parametric** model to the simulated data getting `MSEp_tilde[b]`

   c. Fit and tune (as done above) the **nonparametric** model to the simulated data getting `MSEnp_tilde[b]`

   d. Calculate: `t_tilde[b] = MSEp_tilde[b] - MSEnp_tilde[b]`

6. Once you get a (bootstrapped) p-value, what is the conclusion of your test? Are those secondary bumps there?
   Finally, look again at the original scatterplot, do you think that all the hypotheses behind the simulation scheme adopted (double-check above) are reasonable for the data at hand?
   More in general, how robust do you think your conclusions are w.r.t. them? Explain.
