Code

Exercise Set 1

Exercise Set 1

See Moodle for general instructions. For each of the problem please give an estimate of how long the problem took to solve; we will use this information to tune the next exercise sets.

Problem 1

Learning objective: probability distributions, familiarity with tools, idea of sampling from distribution.

Consider sampling real numbers from a normal distribution with mean $\mu \in \mathbb{R}$ and unit variance, $N(\mu, 1)$. One way to do the sampling is the Metropolis-Hastings (MH) algorithm which works as follows.

Let $P(x)$ be the probability of x, in the case of this problem given by $P(x) = p(x \mid \mu) = \exp(-(x-\mu)^2/2)/\sqrt{2\pi}$. The MH algorithm works as follows, given parameter $\epsilon^2 > 0$ and random initial point x_0 and t initialised to $t \leftarrow 1$.

1. Sample a *proposal* x' from a normal distribution with mean x_{t-1} and variance of ϵ^2.
2. Let $r = \min(1, P(x')/P(x_{t-1}))$ or, equivalently, $\log r = \min(0, \log P(x') - \log P(x_{t-1}))$ [see Problem 4!].
3. Sample a random number r' from uniform distribution in $[0, 1]$.
4. If $r' \leq r$ let $x_t \leftarrow x'$ (accept the proposal), otherwise let $x_t \leftarrow x_{t-1}$ (reject the proposal).
5. Increment t by one $t \leftarrow t + 1$ and iterate.

You will end up with a sequance of numbers $x_1, x_2, \ldots$, called Markov chain Monte Carlo (MCMC) **chain**, that (by some theory that we will not cover in the course) are from the distribution given by $P(x)$, at least after you have iterated long enough.

I'll be using Stan in this course which samples probabilistic models. This problem introduces you to the principles of MCMC sampling. The advantage of MH algorithm is that we can in principle obtain samples from any distribution for which we can write down the (log-)probability.

Tasks

a. Use numpy's sampling function (or R's equivalent) to sample 100 real numbers from $N(\mu, 1)$ when $\mu = 2$. I.e., do the sampling without MH.
b. Then use the MH algorithm, as described above, to sample 100 real numbers. Use $\mu = 2$, $\epsilon = 0.5$, and $x_0 = 4.2$.
c. Plot the sampled numbers as a function of t and histogram of sampled values for both cases above. How do the results compare? What happens if you increase the number of samples to 1000? Also try $x_0 = 42$.

Problem 2

Learning objective: familiarity with command line tools

Unix command line tools are quite useful. Download the csv-file of Problem 3 below and do the following operations with the unix command line tools.

Hint: Useful commands: head, tail, unique. You can find the instructions in any unix system by man command (e.g., "man head"), or by using Google.

Tasks

a. Show the first and last lines by using head and tail.
b. Count the rows by using wc.
c. Change the file from csv (comma-separated values) format to tsv (tab-separated values) format by using sed.
d. Separate the class variable (column “event”) by using awk and list the unique classes.

Problem 3

Objective: familiarity with tools, basic description of the data set

This exercise relates to a data set about new particle formation (NPF) of which you will do your term project. A version of the data set can be be found in the file npf_train_full.csv. It contains a number of variables measured on 704 different days (rows of the file) at the Hyytiälä forestry research station. The variables are daily means and standard deviations of various measurements between sunrise and sunset, except for wind direction (variables "HYY_META.WD*“) for which we have computed mean of cosine and sine of the wind vectors, respectively. You can find the explanation of the variables at https://avaa.tdata.fi/web/smart/smear (https://avaa.tdata.fi/web/smart/smear) or via https://wiki.helsinki.fi/x/XYiKDg (https://wiki.helsinki.fi/x/XYiKDg). On each day there can be a NPF event. The type of NPF event, if occurred, is given in column”event“, with value”nonevent" meaning no NPF event took place and “Ia”, “Ib”, and “II” being different NPF event types. For more information see, e.g., Hyvönen et al. (2005), https://hal.archives-ouvertes.fr/hal-00301731 (https://hal.archives-ouvertes.fr/hal-00301731), or Joutsensaari et al. (2018), https://doi.org/10.5194/acp-18-9597-2018 (https://doi.org/10.5194/acp-18-9597-2018). CS is the condensation sink in the units of s^{-1}, for definition see Kulmala et al. (2012), https://doi.org/10.1038/nprot.2012.091 (https://doi.org/10.1038/nprot.2012.091).

The instructions below are in R, but you can do equivalent tasks with Python as well. See the 11 May lecture material (e.g., the file aktia.zip) for the corresponing Python operations. Before reading the data into Python or R, it can be viewed in Excel or a text editor.

Task a

Use the read.csv() function to read the data into R. Call the loaded data npf. Make sure that you have the directory set to the correct location for the data.

Hide

```
npf <- read.csv("npf_train_full.csv")
```

Task b

Look at the data using the fix() function. You should notice that the second column is the date and first column is the id. We don’t really want R to treat this as data. However, it may be handy to have the dates. Try the following commands:

Hide

```
rownames(npf) <- npf[,"date"]
fix(npf)
```

You should see that there is now a row.names column with the name of each id recorded. This means that R has given each row a name corresponding to the appropriate id. R will not try to perform calculations on the row names. However, we still need to eliminate the first column in the data where the names are stored. Try

Hide

```
npf <- npf[,c(-1,-2)]
#fix(npf)
```

Now you should see that the first data column is date. Note that another column labeled row.names now appears before the data column. However, this is not a data column but rather the name that R is giving to each row.

Task c

i. Use the summary() function to produce a numerical summary of the variables in the data set. We notice that the variable “partlybad” is always false and thereofore useless. We opt to remove it as well.

Hide

```
npf <- npf[,-2]
```

ii. Use the pairs() function to produce a scatterplot matrix of the first ten columns or variables of the data. Recall that you can reference the columns 2 to 11 of a matrix A using A[,2:11].

Hide

```
pairs(npf[,2:11])
```

iii. Use the plot() function to produce side-by-side boxplots of event vs. nonevent days.

Hide

```
boxplot(npf$HYY_META.RHIRGA84.mean ~ npf$event)
```

iv. Create a new qualitative variable, called isevent, which is “event” if there was a NPF event and nonevent" otherwise.

Hide

```
isevent <- rep("event",nrow(npf))
isevent[npf$event=="nonevent"] <- "nonevent"
isevent <- as.factor(isevent)
npf <- data.frame(npf,isevent)
```

Use the summary() function to see how many event days there are. Now use the plot() function to produce side-by-side boxplots of HYY_META.RHIRGA84.mean versus event.

Hide

```
boxplot(npf$CS.mean ~ npf$isevent)
```

v. Use the hist() function to produce some histograms with differing numbers of bins for a few of the quantitative variables. You may find the command par(mfrow=c(2,2)) useful: it will divide the print window into four regions so that four plots can be made simultaneously. Modifying the arguments to this function will divide the screen in other ways.

vi. Continue exploring the data, and provide a brief summary of what you discover.

Problem 4 (extra)

Learning objective: how to deal with probabilities, relevance of flaoting point arithmetics.

You can do this problem if you have time and interest. Otherwise, please just read through the material and remember it if you later have floating point problems.

In machine learning we often have to deal with very small (or large) numbers. Take, for example, the probability density of the normal distribution $p(x) = \exp(-x^2/2)/\sqrt{2\pi}$, which produces to very small numbers for any larger values of x. Multiplication, division, and sums can easily lead to under or overflows. We can mitigate the problem by representing the probabilities as logarithms and then doing the multiplication, division, and summation using logarithmic values. I.e., instead of a probability p_i, where $i \in [n] = \{1, \ldots, n\}$, we store into computer the logarithm of it, i.e., $l_i = \log p_i$. Denote the product by $p_P = \prod_{i=1}^{n} p_i$ and sum by $p_S = \sum_{i=1}^{n} p_i$. Denote $l_P = \log p_P$ and $l_S = \log p_S$. Show the following equalities are true:

- $l_P = \sum_{i=1}^{n} l_i$.
- $l_S = \max_{j \in [n]} l_j + \log \sum_{i=1}^{n} e^{l_i - \max_{j \in [n]} l_j}$.

Why are these numerically more stable way to compute probabilities?

Implement the operations as R or Python functions. Compute the following expressions by without the log presentation and by using the log presentatation:

- $p(100)/(p(100) + p(100.01))$.

You should notice that without the “log trick” you should obtain a NaN value because of floating points underflows, but with the log trick you should obtain a correct answer which in this case should be about 0.731. Operations like this could appear, e.g., in Naive Bayes classifier or other machine learning computations.