Processing of Biological Data

Prof. Dr. Volkhard Helms
Summer Semester 2017

Saarland University
Chair for Computational Biology

Exercise Sheet 3

Due: 13.06.2017 10:15

Submission

- You are advised to work in groups of two people. If necessary, we will suggest teammates.
- Submit your solutions on paper at the beginning of the lecture in the lecture hall or in Room 3.01, both E2 1. Alternatively you may send an email with a single PDF attachment to kerstin.reuter@bioinformatik.uni-saarland.de. Late submissions will not be considered. **In any case, hand in all source code via mail.**
- Do not forget to label your axes, append legends etc. when visualizing data. Numbers should be rounded in a reasonable way.
- Do not forget to mention your names/matriculation numbers.
- You are free to use any programming language to solve the problems. Nevertheless, Python programming language together with the Seaborn Python visualization library, SciPy, and Pandas are highly recommended. Thus, you are allowed to use libraries to solve the following problems.
- Don't worry about the length of this exercise sheet. The following pages contain a lot of information on suitable data structures, file formats, and reasonable strategies how to solve the problems.

Exercise 3.1: Artificial mutation data (30 points = 5+5+5+5+5+5)

In the first part of the assignment you will analyze artificial mutation data in terms of distributions, statistical significance, and effect size. This exercise serves as an introduction to understand the basic principles needed for Exercise 3.2. The data is provided in tab-separated files named `artificial1.txt`, `artificial2.txt`, and `artificial3.txt`. Each file consists of two columns with every column representing mutation densities (mutations per kilo–base [kb]) found in two different genomic regions (e.g. promoter and CDS – coding DNA sequence – region). All numbers are drawn from normal distributions.

(a) Implement a parser that takes an `artificial*.txt` file as input and returns two lists containing all values of the two columns, respectively. Thus, the lists represent the mutation densities for the two genomic regions, respectively.

(b) For every artificial dataset, visualize the two distributions via histograms and/or density plots, e.g. using the seaborn.distplot function. The data for the genomic regions should be summarized into one plot, hence resulting in a total of three plots (one for every artificial dataset), each illustrating the two distributions for the genomic regions, respectively.

(c) For every artificial dataset, report the mean, standard deviation, and median mutation densities of the two genomic regions. A table representation could be beneficial here.

(d) Next, to analyze the statistical significance between mutation densities of different genomic regions, a statistical significance test is necessary. For every artificial dataset, calculate the p–value between the mutation densities of the two genomic regions by applying the *Wilcoxon rank-sum test*, e.g. using the scipy.stats.ranksums function.

(e) *Cohen's d* is a simple measure to calculate the *effect size* in order to quantitatively measure the difference between two sample distributions. *Cohen's d* is defined as

$$d = \frac{(m_2 - m_1)}{\sqrt{\frac{s_1^2 + s_2^2}{2}}}$$

with m_1, m_2, s_1, and s_2 the mean m and standard deviation s of two samples, respectively. Effect sizes are separated into *small* ($0.2 \leq d < 0.5$), *medium* ($0.5 \leq d < 0.8$), and *large* ($d \geq 0.8$) difference. Implement a function that takes two samples and returns the *Cohen's d* value. This function should be applied to the mutation densities of the two genomic regions of every artificial dataset, respectively.

(f) Compare the three artificial datasets considering the visualized distributions, mean/std/median values, statistical significance, and *Cohen's d* effect size. Explain the different results between the three artificial datasets (high or low p–values/effect sizes). Also consider the number of samples within each dataset.

Exercise 3.2: Experimental mutation data (70 points = 15+15+20+20)

In the second part of the assignment you will analyze experimental mutation data from the popular *1000 Genomes Project*. The *1000 Genomes Project* reports more than 80 Mio. human SNPs and indels from more than 2,000 individuals across various populations.
A small fraction of these mutations (100,000 SNPs and indels) located on chromosome 1 can be found in the `Mutations_1000_genomes_bed_chr1.csv` file. Mutations are reported in tab-separated *.bed format:

chromosome	start position [bp]	end position [bp]	identifier	reference	alternative
chr1	11007	11008	rs575272151	C	G

with the genomic location (chromosome, start, and end position), a unique identifier, and the mutation itself, here a C/G SNP.
You will compare and analyze mutations in different genomic regions, namely the promoter region, transcript region, coding region, and intergenic region. Genomic information is provided in the tab-separated `genes_hg19_chr1.txt` file. The following columns are necessary (all other columns can be ignored):

Column name	Explanation
name	Reference ID
chrom	Chromosome
strand	+ or – strand
txStart	Transcript start [bp]
txEnd	Transcript end [bp]
cdsStart	Coding start [bp]
cdsEnd	Coding end [bp]
name2	Gene name

(a) First, the genomic information from the gene file `genes_hg19_chr1.txt` must be provided in suitable format for further analyses.

(1) Write a parser that takes the gene file as input and returns all necessary information saved as useful data structure, such as a Python dictionary with the gene names (`name2`) as keys and genomic information as values:
`{'SGIP1':{'chrom':'chr1','txStart':66999824,'txEnd':67210768,'cdsStart':67000041,'cdsEnd':67208778,'refid':'NM_032291'},'NECAP2':{...},...}`
The data structure should only contain genes that fulfill the following criteria:

- Located on the + `strand`
- `cdsStart` differs from `cdsEnd`
- Do not encode microRNAs (`name2` starts with 'MIR') or small nucleolar RNAs (`name2` starts with 'SNO')
- `name`, i.e. the RefSeq ID, starts with 'NM_'

If there are several entries for one gene name (`name2`), the longest transcript variant (*length* = `txEnd`–`txStart`) should be taken into account. If the transcript variants of a gene have the same length, take the one appearing first in the gene file.

(2) From your final data structure, report the *RefSeq* gene IDs (`name`) for the genes *NECAP2* and *LEPR*.

(b) The overall goal is to analyze mutations located in different genomic regions. Therefore, mutations must be assigned to their respective location. This is accomplished by overlapping the genomic information (chromosome, start, end) of the mutations with the regions of interest. An easy–to–use function that intersects two genomic coordinate files is intersectBed provided by the BEDTools suite.

(1) First, the *.csv files containing the genomic information (chromosome, start, end, RefSeq ID, gene name) of the genomic elements must be generated. We consider the following elements:

- promoter region: Defined from −2000 bp to +1000 bp around `txStart`.
- transcript region: Defined from `txStart` to `txEnd`.
- coding region: Defined from `cdsStart` to `cdsEnd`.
- intergenic region: The region between two genes. For simplicity, defined from the `txEnd` up to `txEnd`+10000 bp.

Implement a function that takes the data structure from Exercise 3.2 (a) as input and generates four *.csv files – one for every genomic region defined above – that represents the genomic coordinates for every gene. The files should be named in the following way: `genes_hg19_chr1_*.csv` with the * representing the genomic region, i.e $* \in$ {`"promo"`, `"tx"`, `"cds"`, `"intergenic"`}. The final tab–separated *.csv file should adhere the following *.bed format:

chr1	14925493	15441131	NM_201628	KAZN
chr1	15578280	15723909	NM_052929	FHAD1
chr1	15874836	15894672	NM_001287811	DNAJC16

For example: assume a gene *gene_ABC* located on chromosome 1 has the following `txStart`: 5000. An entry, i.e. a row in the final `genes_hg19_chr1_promo.csv` promoter file should look as follows:

chr1	3000	6000	NM_...	gene_ABC

(2) For every of the four *.bed gene files generated above, apply the intersectBed function (or implement a function yourself) to the mutation and genomic information, e.g.:

```
intersectBed -a Mutations_1000_genomes_bed_chr1.csv
-b genes_hg19_chr1_promo.csv -wa -wb > intersect_result_promo.csv
```

This step generates four `intersect_result_*.csv files` – one for every genomic region defined above – containing information on the location of mutations within genomic regions, e.g the file `intersect_result_promo.csv`
provides the information that a G/A SNP is located in the promoter region of gene *ISG15*.

chrom	start	end	id	ref	alt	chrom	start	end	id	name
chr1	949234	949235	rs2465124	G	A	chr1	946846	949846	NM_005101	ISG15
chr1	949271	949272	rs568580969	C	G	chr1	946846	949846	NM_005101	ISG15
...	...	...	...	...	...	...	...	...	...	...

(c) For the final mutation analysis, the information on mutation in genomic elements for every gene should be provided in a suitable data structure such as a pandas.DataFrame. Based on the results/files obtained by applying `intersectBed`, every entry of the final data frame should thereby contain the element length in kb (`kb_...`), number of mutations (`nbr_muts_...`), and the mutation density defined as the number of mutations per kb (`den_muts_...`) for every gene, e.g.

gene_name	kb_promo	nbr_muts_promo	den_muts_promo	kb_tx	nbr_muts_tx	den_muts_tx	...
FAM213B	3.0	20	6.67	4.72	45	9.53	...
ESPN	3.0	19	6.33	36.16	302	8.35	...
...	...	...	...	...	...	...	...

...	kb_cds	nbr_muts_cds	den_muts_cds	kb_intergenic	nbr_muts_intergenic	den_muts_intergenic
...	2.63	21	7.98	10.0	131	13.1
...	35.19	293	8.33	10.0	103	10.3
...	...	...	...	...	...	...

Thus, write a function that parses all four `intersectBed` files and returns this Table (or a similar data structure if you are not using Python). Only report genes that have a mutation in at least one of the genomic elements. In other words, column rows with zero entries should be discarded.

(d) Finally, the distributions of mutations in the four different genomic elements should be analyzed.

(1) Generate boxplots, e.g. using seaborn.boxplot function, that display the mutation density (`den_muts_...`) for every of the four genomic regions. All four boxplots should be summarized into one plot. Column values can be easily retrieved from a pandas.DataFrame `df` by `list(df["den_muts_promo"])`.

(2) Similar to the boxplots, visualize the mutation density distribution as histograms and/or density plots, e.g. using the seaborn.distplot function (compare with Exercise 3.1).

(3) Report the mean, standard deviation, and median mutation densities of all genomic regions. A table representation could be beneficial here.

(4) Analyze the statistical significance between mutation densities of the intergenic region vs. the mutation densities of all other genomic regions by applying the *Wilcoxon rank-sum test*, e.g. using the scipy.stats.ranksums function. Why is it suitable to compare the distributions of these genomic regions to intergenic regions? This should result in a total of three p–values.

(5) Calculate the *effect size* to quantitatively measure the difference between the mutation densities of intergenic region vs. the mutation densities of all other genomic regions by calculating *Cohen's d* values. This should result in a total of three d values.

(6) Interpret your results. Why are p-values statistically significant/not statistically significant? What about the *Cohen's d* values? From a biological point of view, which genomic region(s) should exhibit the largest mutation densities? Which the lowest? Justify your answer. Did the analysis results match your expectations? When interpreting your data, also consider the given mutation/genomic data set as well as the definition of the genomic regions. For instance, the coding regions consist of introns and exons as well. The definition of the intergenic regions is kind of arbitrary.