Bioinformatics III

Prof. Dr. Volkhard Helms
Daria Gaidar, Markus Hollander, Duy Nguyen, Thorsten Will
Summer Semester 2018

Saarland University
Chair for Computational Biology

Exercise Sheet 3

Due: May 4, 2018 13:15

Submit your solutions on paper, hand-written or printed at the *beginning* of the lecture or in building E2.1, Room 3.02. Alternatively you may send an email with a single PDF attachment to thorsten.will@bioinformatik.uni-saarland.de. Either way, send me your source code. Due to a bank holiday on May 1, the tutorials for Assignments 2 and 3 will be held together at May 8.

Bayesian classification and network communities

In this assignment you will implement a Naive Bayes classifier and the algorithm of *Radicchi et al.* to identify the communities of a given network. There will be no code templates for this assignment and you are free to use any programming language. The usage of libraries that solve core problems of the exercises (any graph or machine learning package) is prohibited, though.

Exercise 3.1: Naive Bayes classifier (50 points)

A naive Bayes classifier is a simple classifier based on the application of Bayes' theorem and the (naive) assumption of independence among all features.
You will implement and evaluate such a classifier on the basis of two artificial datasets that relate 100 discrete features (for simplicity) to a boolean outcome. In our case, imagine that the features describe properties derived from a pair of proteins (co-expression, genomic distance, etc.) that may tell us if they interact with each other and form a complex.

The following abbreviations will be used within the remaining exercise:
C : "the proteins form a complex", $\overline{C}$: "no interaction between the proteins",
S : state of all features regarding the pair, S_i : state of individual feature i.

(a) Given the states of the features, you want to infer if two proteins are likely to physically interact. In practice, log-likelihood ratios are used in binary classification:

$$\log \frac{P(C|S)}{P(\overline{C}|S)}.$$

Derive a term that uses observable probabilities such as $P(S_i|C)$ to calculate the log-likelihood ratio from training data. How does the actual classification work?

(b) Shortly discuss: What are the practical advantages of the logarithm and the likelihood ratio within this framework? State two reasons why this particular type of classifier may perform poorly on a real world dataset.

(c) Use the file 'training1.tsv' to build a model. This basically means to determine all necessary priors and likelihoods from part (a). The file layout is explained in 'README.txt'. Report $P(C)$ and $P(\overline{C})$ as well as the ten S_i (feature number, variant and log-ratio) with the highest absolute log-likelihood ratios. Examine and comment on the results of the training-phase. Which features seem to be the most helpful?

(d) Predict the ability to interact for the protein pairs in 'test1.tsv' with your previously trained model and report the accuracy of the classifier.

(e) At last, train a model using 'training2.tsv' and test it on 'test2.tsv'. Again, take a look at the measures asked for in parts (c) and (d). Why is the performance inferior? Could this have been expected?

Exercise 3.2: Network communities (50pts)

(a) **Edge-clustering coefficient**

The edge-clustering coefficient $\tilde{C}^{(3)}_{i,j}$ of a link between nodes i and j is defined as the ratio of the actual number of triangles $z^{(3)}_{i,j}$ to which the link between i and j contributes and the number of possible triangles, determined by the minimum of the degrees k_i and k_j of the two nodes i and j:

$$\tilde{C}^{(3)}_{i,j} = \frac{z^{(3)}_{i,j} + 1}{\min[k_i - 1, k_j - 1]}$$

If one of the nodes has a degree of 1, then $\tilde{C}^{(3)}_{i,j}$ is infinite. What is the maximal *finite* value that the edge-clustering coefficient can take? For which configuration does this occur? Give an example!

(b) **Determine communities**

To determine the communities of the supplied network given in `GoT.txt` (found in the additional materials), perform the following steps:

(1) **Decomposition of the network** (by programming)

As explained in the lecture, iteratively delete the links with the smallest $\tilde{C}^{(3)}_{i,j}$:

i. Read in the network file.
ii. Calculate the edge-clustering coefficient $\tilde{C}^{(3)}_{i,j}$ for each link.
iii. Find the link with the smallest $\tilde{C}^{(3)}_{i,j}$ and delete it from the network (or mark it). Store the link.
iv. Repeat from (ii) until there is no link left.

Give the links that you deleted from the network in (iii) by printing the names of the two nodes and their current edge-clustering coefficient in the order of their deletion. Of course, add the output to the PDF/sheet that you hand in. Implement this part as a script or class-based, there are no specifications you need to adjust to.

(2) **Build communities and the dendrogram** (by pen and paper)

There are two criteria for a community (see *Radicchi et al., 2004*):

i. In a *community in a strong sense* every single member of the subgraph V has more links to the inside of the community (k^{in}) than to the outside (k^{out}):

$$k_i^{\text{in}}(V) > k_i^{\text{out}}(V) \qquad \forall i \in V$$

ii. In a *community in a weak sense* the total number of links inside the subgraph V is bigger than to the outside:

$$\sum_{i \in V} k_i^{\text{in}} > \sum_{i \in V} k_i^{\text{out}}$$

Use the links deleted in (1) in reverse order, i.e., the link that was deleted last is now used first to construct the communities. To do so, take one link after the other and check if they have nodes in common with the already included links. During this composition stage you do not need to keep track of the links, but only of the nodes that belong to the same subgraph:

i. If the latest link is disjoint from the already processed links, then start a new subgraph (=list of nodes of this subgraph) from this one.
ii. If the latest link has a single node in common with one of the existing subgraphs, then add the other node of this link to that (list of the nodes of the) subgraph, too.
iii. If the two nodes of the latest link belong to two different subgraphs, then join the two subgraphs to form a single one from them. Highlight the two lists of nodes that are joined in this step.

Finally, when the last link is added, you should end up with a single graph that contains all nodes of the network and a listing of the subgraphs just before they were joined to form bigger ones.

To draw the dendrogram of the network, look at the above choice (iii), the joining of two groups: start from the individual nodes and every time that this happens, connect two subgraphs.

(c) **Visualization of the communities**

In Figure 1 a layout of the network is given. Use it to visually identify communities. Point out **two** communities that are disjunct. Specify for each of the communities whether the weak or the strong criterion applies.

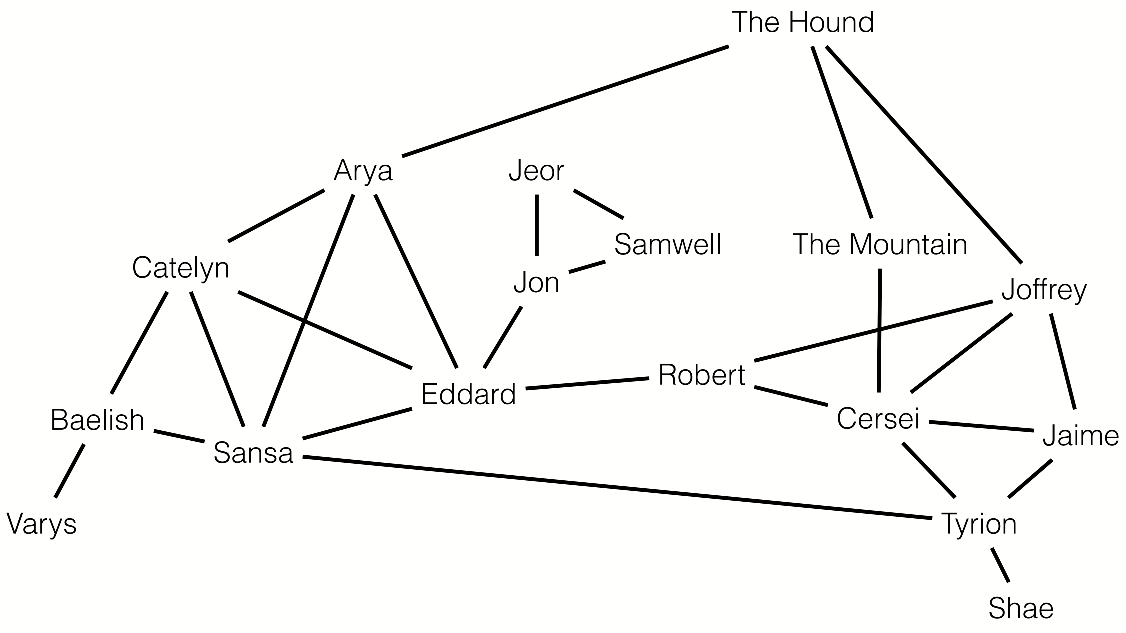

Figure 1: The network in “GoT.txt” visualized.

May the Fourth be with you!