1 Installing and running Kingfisher in linux (shell server)

The most efficient pattern discovery programs are usually developed with C or C++ (sometimes with gnu extensions) in linux and making them work in other operating systems is not guaranteed. If you don't have linux in your own computer, you can use Aalto's linux shell services from home (see `https://www.aalto.fi/en/services/linux-shell-servers-at-aalto`) or e.g., install linux in VirtualBox. Here are brief instructions how to install and run the Kingfisher program in linux. In the analysis, you will need to add your own feature extraction program to the pipeline (a simple shell script will help, but is not necessary).

1. Log into one of the linux servers (any server is fine in this task, since it will be light computation), see `https://www.aalto.fi/en/services/linux-shell-servers-at-aalto`. Skip this step, if you run linux in your own computer.

2. Load package from the course page or `https://mycourses.aalto.fi/pluginfile.php/2000134/course/section/233748/kingfisher1_2c.tar.gz`. In the server, the command is
   `wget https://mycourses.aalto.fi/pluginfile.php/2000134/course/section/233748/kingfisher1_2c.tar.gz`

3. Unpack the programme, command
   `tar xzf kingfisher1_2c.tar.gz`

4. Compile it by command `make`. (The current version has been tested with gcc 9.4.0.)
   `make`

5. Load Mushroom data for testing from the course page or `http://fimi.uantwerpen.be/data/mushroom.dat`. The Mushroom data is already in the transaction form with numeric codes, so no preprocessing is needed. This is only for practising. In the server, the command is
   `wget http://fimi.uantwerpen.be/data/mushroom.dat`

6. Run a test:
   `./kingfisher -i mushroom.dat -k120 -M-100`

   The -i option defines input file (give the path if it is not located in the same directory) and -k option tells the last item number (or an upper-bound for it – numbering can begin from 0 or 1). Since the measure

type is not defined, the program uses by default measure $\ln(p_F)$. The initial threshold is set to -100 with option -M. Since the number of rules and the rule type (positive, negative or both) are not defined, the program uses the default values, and outputs max 100 positive association rules having $\ln(p_F) \leq -100$.

The program outputs following statistics for all rules: fr (frequency), cf (confidence = precision), gamma (lift), delta (leverage) and M (the chosen goodness measure).

7. You can write the result into a file by directing the output or using option -o. If the rule arrows are inconvenient, you can change the output format with option -p2.

8. With Mushroom, there will be no computational problems, even if you search for the top-1000 rules and used far too loose initial threshold -1:
`./kingfisher -i ../../../DATA/mushroom.dat -k120 -M-1 -q1000`
However, if the data is demanding (execution takes too long), consider these:

   - Search less rules, e.g., top-10 (option -q10) or just top-1. This will also give an idea what is the magnitude of $\ln(p)$-values (helps in the following).
   - Use a stricter initial threshold for your measure (smaller for $\ln(p)$, larger for mutual information). This can speed the first levels a lot! However, finding a good threshold requires trial and error (either the program is slow or you don't find anything – but it is also possible there are no strong associations!).
   - Restrict the complexity of rules with option -l (e.g., -l4 creates rules using 2–4 attributes). This can also help, if the data is very small and multiple hypothesis testing would be a problem.
   - Use minimum frequency threshold (option -m) in addition to the goodness measure. However, use this only as a last resort, since this means that you may miss some rare but significant discoveries.

9. You can see all options by command `./kingfisher`. The usage is described in more detail in README of the package.

   - In the current task, you might benefit from certain constraints (option -b followed by the constraint file name) that filter out certain uninteresting or redundant rule. For example, since worms, insects, larvae, and snails belong to invertebrates, rules involving

them (e.g., *eats_larvae*→ *eats_invertebrates*) are not interesting. Similarly, you may want to forbid rules involving biotopes both in the condition and consequence parts.
If there are only a few constraints, you can generate the file manually, but for multiple constraints, a generator script is recommended. E.g., if constraints (using numerical codes) are in file bconstr.txt.codes, run
`./kingfisher/version2021/kingfisher -i birdstrans.txt.codes -k152 -M-5 -o birdrules.txt -b bconstr.txt.codes`

2 Converting names to codes and vice versa

Checking the correctness of data and interpreting the rules is easier, if the attribute names are informative strings, like "genderssimilar", "manyeggs", or "eatsworms", instead of numerical codes. Therefore, transformation between names and numerical codes is needed. For this purpose you can use namescodes.c program that transforms labels to codes and codes to labels.

Note: Kingfisher assumes that **the field separator is space**. Therefore, you cannot use spaces in your attribute names.

When you have extracted binary features from the bird data, store them in the transactional form (using space as separator) into some file, e.g., birdstrans.txt. Here are instructions how you can use namescodes to convert your data into numerical codes and the resulting rules with numerical codes back to your own feature names.

1. Load the namescodes package from the course homepage or `https://mycourses.aalto.fi/pluginfile.php/2000134/course/section/233748/namescodes.tar.gz`

2. Unpack and compile namescodes as you did for kingfisher.

3. Transform labels to numerical codes:
   `./namescodes -nbirdstrans.txt -tbirdtable.txt -L`
   The attributes and codes are listed in birdtable.txt (you can check it to see how many attributes you have).

4. Run kingfisher with data birdstrans.txt.codes using a suitable threshold (so that you find, e.g., 100-300 rules) and save results to birdrules.txt

5. Transform numerical codes back to label names: ./namescodes -cbirdrules.txt -tbirdtable.txt The result is birdrules.txt.names

6. Look at birdrules.txt.names and analyze the rules. Hint: You can use command egrep to pick up rules related to certain attributes, like `egrep "long-billed" birdrules.txt.names`.

Note: If you generated a constraint file using labels, you need to convert it to numerical codes, too, but you need to use the same translation table. E.g., if the file with labels is bconstr.txt, run

```
namescodes -n bconstr.txt -tbirdtable.txt
```

3 Making a shell script

Pattern discovery is an iterative process and you will repeat the same steps multiple times, after inventing new features and adjusting your feature extraction program and program parameters. Therefore, it is handy to make a simple shell script that runs everything. Here are brief instructions:

1. Decide name for your shell script, e.g., runbirds.sh, and open it in a text editor.

2. List commands in the file. Here we will use text files for midphase results so that you can check their correctness if needed. E.g., let's suppose the feature extraction is done with an awk program featex.awk. Then the script would be

   ```
   awk -f featex.awk birdsv2.csv > birdstrans.txt
   ./namescodes -n birdstrans.txt -tbirdtable.txt -L
   ./kingfisher -i birdstrans.txt.codes -k152 -M-5 -q300 -obirdrules.txt
   ./namescodes -c birdrules.txt -tbirdtable.txt
   ```

   (Just set your own parameter values!)

3. Give execution rights to the script:
   `chmod u+x runbirds.sh`

4. Run it:
   `./runbirds.sh`

5. Look at the results in birdrules.txt.names

Note: If you use a constraint file, you need to transform it every time, the attributes change (new features are extracted). However, you don't need to generate the constraints again, if the listed attribute names have not changed. For example, if you have constraint file bconstr.txt containing the following lines:
eats_insects eats_invertebrates
eats_worms eats_invertebrates
eats_snails eats_invertebrates
eats_larvae eats_invertebrates
eats_invertebrates eats_insects eats_worms eats_snails eats_larvae

and you have modified featex.awk so that these attribute names remain untouched, the shell script would be

```
awk -f featex.awk birdsv2.csv > birdstrans.txt
./namescodes -n birdstrans.txt -tbirdtable.txt -L
./namescodes -n bconstr.txt -tbirdtable.txt
./kingfisher -i birdstrans.txt.codes -k152 -M-5 -o birdrules.txt
-b bconstr.txt.codes
./namescodes -c birdrules.txt -tbirdtable.txt
```