The *crystalmeth* vignette

Christoph Geisenberger

2020-03-19

Loading data

Let's begin with loading the required packages.

```r
library(crystalmeth)
library(tidyverse)
```

Currently, crystalmeth only supports reading data directly from IDAT files. This is because some of the functionalities (such as creating copy-number plots) need access to raw fluorescence measurements. For the purpose of the tutorial let's assume we have a folder with the following files:

```
.
+-- temp
|   +-- input_data
|       +-- Screenshot 2020-01-04 at 18.20.12.png
|       +-- README.md
|       +-- GSM3319212_3999941101_R04C02_Grn.idat
|       +-- GSM3319210_3999941101_R02C02_Red.idat
|       +-- GSM3319211_3999941101_R03C02_Red.idat
|       +-- GSM3319210_3999941101_R02C02_Grn.idat
|       +-- GSM2309170_200134080018_R03C01_Red.idat
|       +-- GSM2309170_200134080018_R03C01_Grn.idat
```

Some of the files do not have the correct input format, and others are missing either the Red.idat or Grn.idat file. Let's use `scan_directory()` to find the cases with proper input data:

```r
input_dir <- "../temp/input_data/"
files <- scan_directory(dir = input_dir)

# get_cases() will extract the basenames with matching _Red.idat and _Grn.idat files
basenames <- files %>% get_cases()
basenames
#> [1] "GSM2309170_200134080018_R03C01" "GSM3319210_3999941101_R02C02"
```

As you can see, only the two complete cases are listed. It's time to set up our workflow. The next step will be to create new instances of the `ClassificationCase` class. The idea behind using a dedicated class is to keep all information pertaining to a sample in the same place. Furthermore, this design creates an opportunity to for easy parallelization, in case it should be necessary to process large numbers of samples.

```r
# Let's create a new Classification case object for one of the cases
case <- ClassificationCase$new(basename = basenames[1], path = input_dir)

# messages are suppressed in this code chunk
```

NB: Because every sample is self-contained within an object, certain preprocessing functions cannot be used. However, in our experience, these differences are negligible when it comes to the classification results.

Data processing

After creating the object, we will need to run multiple steps in order to produce data for a sample report. These are: 1. Reading data from IDAT files 2. Perform preprocessing 3. Impute missing data 4. Classify the sample 5. Estimate tumor purity 6. Create a copy-number profile

It is helpful to dig into some details here. There are currently multiple different algorithms to perform background normalization. While some of them are slightly better at removing unwanted technical variation, these differences are negligible when it comes to the classification results. The default setting is `minfi::minfi::preprocessIllumina()` which we recommend. In addition, some classifiers are not able to deal with missing data. This means that a single CpG without data can render the sample un-classifiable. Therefore, missing data is imputed. The current standard is to sample randomly from the data points which are *not* missing. In our experience, classification results are stable up to 20% (!) missing data. This measurement is therefore reported in the PDF.

> NB: The current implementation of crystalmeth only supports randomForest models. However, we are planning > to make a wider range of machine learning models available in future releases.

```r
load("../temp/NetID_v1.RData") # load net_id_v1 (randomForest classifier)

# this command will run the full workflow
case$run_workflow(rf_object = net_id_v1)

# messages are suppressed in this code chunk
```

Generage diagnostic report

As a last step, most clinical user probably want to generate reports for diagnostic cases. Crystalmeth offers a handy wrapper around `rmarkdown::render` to achieve this. Please note that a *template file* is needed for this purpose. While not part of the package, refer to the GitHub Readme for more information.

```r
output_directory = "../temp/reports/"
template_path = "../temp/netid_report.Rmd"

# note that render_report() returns the location(s) of the output files
#out_files <- render_report(case, template = template_path, out_dir = output_directory)
```