Using NicheNet with OmnipathR

Denes Turei Alberto Valdeolivas Julio Saez-Rodriguez

Abstract

NicheNet is a method to infer ligand activities from transcriptomics data. It relies on prior knowledge which includes signaling networks and transcriptomics data from ligand perturbation experiments. OmnipathR provides a close integration with NicheNet: it provides methods to build the networks directly from the original sources in a format suitable for NicheNet in a highly customizable way. OmnipathR also provides a workflow template which connects OmnipathR and NicheNet methods from building the prior knowledge up until the inference of ligand activities. With this pipeline the only thing users need to provide is processed transcriptomics data.

Contents

[1] Institute for Computational Biomedicine, Heidelberg University

1 Status

The integration between NicheNet and OmniPath is a fresh development, as of May 2021 we have already tested the basic functionality. To use the features described here you will need the latest git version of OmnipathR, which we update often as we test and improve the elements of the pipeline. Bug reports, questions and suggestions are welcome, please open an issue on github.

2 Introduction

The OmnipathR package contains methods to build NichNet[1] format networks from OmniPath[2] data and from the resources used in the original NicheNet study. With a few exceptions, almost all resources used originally by NicheNet are available by OmnipathR both in their raw format or after minimal preprocessing, and in the format suitable for NicheNet. However, the networks built by OmnipathR are not completely identical with the ones used by NicheNet due to the complexity of processing such a large amount of data. The advantage of building these networks with OmnipathR is the transparent and reproducible process, as the origin of any data record can be easily traced back to its original source. The data used in the NicheNet publication is deposited at Zenodo. The other part of the prior knowledge used by NicheNet is a collection of expression data from ligand perturbation experiments. In OmnipathR This is taken directly from the Zenodo repository.

Apart from building the prior knowledge, OmnipathR provides a number of glue methods which connect elements of the NicheNet workflow, ultimately making it possible to run everything with a single function call, starting from the processed transcriptomics data and going until the predicted ligand activities. To implement such a workflow we followed the analysis from the case study in the OmniPath paper and the vignettes from NicheNet. The workflow consists of thin wrappers around some of the NicheNet methods. Parameters for these methods can be provided from the top level call.

The transcriptomics data is the main input for the pipeline which is specific for the study. The transcriptomics data can be processed by standard tools, such as DESeq[3] for bulk and Seurat[4] for single-cell transcriptomics. For the latter more guidance is available in the NicheNet vignettes. The pipeline presented here only requires the list of genes expressed in the transmitter and receiver cells (in case of autocrine signaling these are the same); and a list with genes of interest. Genes of interest might come from the investigation subject or gene set enrichment analysis (see an example here).

3 Run the workflow by a single call

The `nichenet_main` function executes all the workflow elements, hence it can be used to run the complete workflow. Here we show it as an example, and in the next sections we take a closer look at the individual steps. The first four arguments comes from the study data and objective, as discussed above. All parameters of the workflow can be overridden here, for example, for the `mlrbo_optimization` we override the `ncores` argument, and exclude CPDB from the signaling network and set the confidence threshold for EVEX.

```
library(OmnipathR)
library(nichenetr)

nichenet_results <- nichenet_main(
    expressed_genes_transmitter = expressed_genes_transmitter,
    expressed_genes_receiver = expressed_genes_receiver,
    genes_of_interest = genes_of_interest,
    background_genes = background_genes,
    signaling_network = list(
        cpdb = NULL,
        evex = list(min_confidence = 1.0)
    ),
```

```
    gr_network = list(only_omnipath = TRUE),
    n_top_ligands = 20,
    mlrmbo_optimization_param = list(ncores = 4)
)
```

4 Only model building

Without transcriptomics data, it is possible to build a NicheNet model which later can be used for the ligand activity prediction. Hence if the first four arguments above are `NULL` the pipeline will run only the model building. Still, in this tutorial we wrap all blocks into `eval = FALSE` because the model optimization is computationally intensive, it might take hours to run. To create the NicheNet model it's enough to call `nichenet_main` without arguments, all further arguments are optional override of package defaults.

```
nichenet_model <- nichenet_main()
```

5 Installing packages and loading nichenetr

Before running the pipeline it's necessary to install `nichenetr` and its dependencies. The `nichenetr` package is available only from github, while the dependencies are distributed in standard repositories like CRAN.

```
require(devtools)
install_github('saeyslab/nichenetr')
```

This procedure should install the dependencies as well. Some more specific dependencies are `mlrMBO`, `ParallelMap`, `ParamHelpers` and `smoof`.

The `nichenetr` package is suggested by `OmnipathR` but not loaded and attached by default. This might cause issues, for example NicheNet requires access to some of its lazy loaded external data. To address this, you can either attach `nichenetr`:

```
library(nichenetr)
```

Or call a function which loads the datasets into the global environment, so NicheNet can access them:

```
nichenet_workarounds()
```

6 Testing the pipeline

With a fresh setup, as a first step it is recommended to test the pipeline, to make sure it is able to run in your particular environment, all packages are installed and loaded properly. The optimization process takes hours to run, while the test included in `OmnipathR` takes only around 10 minutes. This way you can avoid some errors to come up after running the analysis already for many hours. Although it's never guaranteed that you won't have errors in the pipeline as the outcome depends on the input data and certain conditions are not handled in `nichenetr`, `mlr` and `mlrMBO`. If you get an error during the optimization process, start it over a few times to see if it's consistent. Sometimes it helps also to restart your R session. I would recommend to try it around 10 times before suspecting a bug. See more details in the docs of `nichenet_test`.

The `nichenet_test` function builds a tiny network and runs the pipeline with a highly reduced number of parameters and iteration steps in the optimization. Also it inputs dummy gene sets as genes of interest and background genes.

```
require(OmnipathR)
nichenet_workarounds()
nichenet_test()
```

7 Workflow steps

7.1 Networks

NicheNet requires three types of interactions: signaling, ligand-receptor (lr) and gene regulatory (gr). All these are collected from various resources. In OmnipathR a top level function, `nichenet_networks` manages the build of all the three networks. It returns a list with three data frames:

```
networks <- nichenet_networks()
```

Its `only_omnipath` argument is a shortcut to build all networks by using only OmniPath data:

```
networks <- nichenet_networks(only_omnipath = TRUE)
```

To the network building functions of OmniPath many further arguments can be provided, at the end these are passed to `import_post_translational_interactions`, `import_intercell_network` and `import_transcriptional_interactions`, so we recommend to read the manual of these functions. As an example, one can restrict the signaling network to the resources SIGNOR and PhosphoSite, while for ligand-receptor use only ligands and enzymes on the transmitter side and receptors and transporters on the receiver side, while for gene regulatory network, use only the A confidence level of DoRothEA:

```
networks <- nichenet_networks(
    only_omnipath = TRUE,
    signaling_network = list(
        omnipath = list(
            resources = c('SIGNOR', 'PhosphoSite')
        )
    ),
    lr_network = list(
        omnipath = list(
            transmitter_param = list(parent = c('ligand', 'secreted_enzyme')),
            receiver_param = list(parent = c('receptor', 'transporter'))
        )
    ),
    gr_network = list(
        omnipath = list(
            resources = 'DoRothEA',
            dorothea_levels = 'A'
        )
    )
)
```

7.1.1 Signaling network

The function `nichenet_signaling_network` builds the signaling network from all resources with the default settings. Each argument of this function is a list of arguments for a single resource. If an argument is set to `NULL`, the resource will be omitted. In the example below we set custom confidence score thresholds for ConsensusPathDB and EVEX, while use no data from Pathway Commons. In this case, the rest of the resources will be loaded with their default parameters:

```
signaling_network <- nichenet_signaling_network(
    cpdb = list(
        complex_max_size = 1,
        min_score = .98
    ),
    evex = list(
        min_confidence = 2
```

```
    ),
    pathwaycommons = NULL
)
```

Currently the following signaling network resources are available by OmnipathR: OmniPath, Pathway Commons, Harmonizome (PhosphoSite, KEA and DEPOD), Vinayagam et al. 2011, ConsensusPathDB, EVEX and InWeb InBioMap. Let's take a closer look on one of the resource specific methods. The name of these functions all follow the same scheme, e.g. for EVEX it is: `nichenet_signaling_network_evex`. From most of the resources we just load and format the interactions, but a few of them accepts parameters, such as confidence score thresholds or switches to include or exclude certain kind of records:

7.1.2 Raw data from network resources

```
evex_signaling <- nichenet_signaling_network_evex(top_confidence = .9)
```

The network format of NicheNet is very minimalistic, it contains only the interacting partners and the resource. To investigate the data more in depth, we recommend to look at the original data. The methods retrieving data from resources all end by `_download` for foreign resources and start by `import_` for OmniPath. For example, to access the EVEX data as it is provided by the resource:

```
evex <- evex_download(remove_negatives = FALSE)
```

7.1.3 Ligand-receptor interactions and gene regulation

These networks work exactly the same way as the signaling network: they are built by the `nichenet_lr_network` and `nichenet_gr_network` functions. Currently the following ligand-receptor network resources are available in OmnipathR: OmniPath, Ramilowski et al. 2015, Guide to Pharmacology (IUPHAR/BPS); and in the gene regulatory network: OmniPath, Harmonizome, RegNetwork, HTRIdb, ReMap, EVEX, PathwayCommons and TRRUST.

7.1.3.1 The OmniPath ligand-receptor network The ligand-receptor interactions from OmniPath are special: these cover broader categories than only ligands and receptors. It includes also secreted enzymes, transporters, etc. Also the `import_intercell_network` function, responsible for creating the OmniPath LR network, by default returns the most complete network. However, this contains a considerable number of interactions which might be false positives in the context of intercellular communication. It is highly recommended to apply quality filtering on this network, using the `quality_filter_param` argument of the functions in the NicheNet pipeline. The default settings already ensure a certain level of filtering, read more about the options in the docs of `filter_intercell_network`. Passing the `high_confidence = TRUE` argument by `lr_network = list(omnipath = list(high_confidence = TRUE))` results a quite stringent filtering, it's better to find your preferred options and set them by `quality_filter_param`. An example how to pass quality options from the top of the pipeline:

```
nichenet_results <- nichenet_main(
    quality_filter_param = list(
        min_curation_effort = 1,
        consensus_percentile = 30
    )
)
```

7.1.4 Small network

Initially for testing purposes we included an option to create a small network (few thousands interactions instead of few millions), using only a high quality subset of OmniPath data. It might be interesting to try the analysis with this lower coverage but higher confidence network:

```
nichenet_results <- nichenet_main(small = TRUE)
```

The small network is not customizable, and it uses the `high_confidence` option in `import_intercell_network`. You can build similar smaller networks by using the general options instead. Another option `tiny` generates an even smaller network, however this involves a random subsetting, the outcome is not deterministic, this option is only for testing purposes.

7.2 Ligand perturbation experiments

The expression data from the collection of over a hundred of ligand perturbation experiments is available in the data deposited by NicheNet authors in Zenodo, in the `expression_settings.rds` file. The function below downloads and imports this data:

```
expression <- nichenet_expression_data()
```

If there is any ligand which is missing from the ligand-receptor network, probably we want to remove it:

```
lr_network <- nichenet_lr_network()
expression <- nichenet_remove_orphan_ligands(
    expression = expression,
    lr_network = lr_network
)
```

7.3 Model optimization

In this step we optimize the parameters of the model and assign weights to the network resources, especially based on the ligand perturbation data. The `nichenet_optimization` function is a wrapper around `nichenetr::mlrmbl_optimization`. The arguments for this function, and the objective function can be overriden, for example it's a good idea to set the cores according to our machine's CPU count:

```
optimization_results <- nichenet_optimization(
    networks = networks,
    expression = expression,
    mlrmbo_optimization_param = list(ncores = 4)
)
```

This function takes very long, even hours to run. Before it returns the results, it saves them into an RDS file. This and the further RDS files are saved into the `nichenet_results` directory by default, it can be changed by the option `omnipath.nichenet_results_dir`.

```
options(omnipath.nichenet_results_dir = 'my/nichenet/dir')
nichenet_results_dir()
# [1] "my/nichenet/dir"
```

7.4 Model build

The next major part is to build a model which connects ligands to targets, i.e. which genes are affected in their expression by which of the ligands. Again, we use a wrapper around NicheNet functions: the `nichenet_build_model` calls `nichenetr::onstruct_weighted_networks` and `nichenetr::apply_hub_corrections`. If the argument `weights` is `FALSE`, the optimized resource weights are not used, all resources considered with the same weight. The results are saved automatically into an RDS.

```
nichenet_model <- nichenet_build_model(
    optimization_results = optimization_results,
    networks = networks,
)
```

7.5 Ligand-target matrix

Finally, we produce a ligand-target matrix, a sparse matrix with weighted connections from ligands to targets. The function `nichenet_ligand_target_matrix` is a wrapper around `nichenetr::construct_ligand_target_matrix`. As usual the results are exported to an RDS. This is the last step of the NicheNet model building process.

```
lt_matrix <- nichenet_ligand_target_matrix(
    nichenet_model$weighted_networks,
    networks$lr_network,
    nichenet_model$optimized_parameters
)
```

7.6 Ligand activities

In this step we ues the NicheNet model to infer ligand activities from our transcriptomics data. The last three arguments are all character vectors with gene symbols. For genes of interest, here we select 50 random genes from the targets in the ligand-target matrix. Also just for testing purposes, we consider all genes in the network to be expressed in both the transmitter and receiver cells.

```
genes_of_interest <- sample(rownames(ligand_target_matrix), 50)
background_genes <- setdiff(
    rownames(ligand_target_matrix),
    genes_of_interest
)
expressed_genes_transmitter <- union(
    unlist(purrr::map(networks, 'from')),
    unlist(purrr::map(networks, 'to'))
)
expressed_genes_receiver <- expressed_genes_transmitter

ligand_activities <- nichenet_ligand_activities(
    ligand_target_matrix = lt_matrix,
    lr_network = networks$lr_network,
    expressed_genes_transmitter = expressed_genes_transmitter,
    expressed_genes_receiver = expressed_genes_receiver,
    genes_of_interest = genes_of_interest
)
```

Once the ligand activities inferred, we can obtain a list of top ligand-target links. It's up to us how many of the top ranking ligands, and for each of these ligands, how many of their top targets we include in this table:

```
lt_links <- nichenet_ligand_target_links(
    ligand_activities = ligand_activities,
    ligand_target_matrix = lt_matrix,
    genes_of_interest = genes_of_interest,
    n_top_ligands = 20,
    n_top_targets = 100
)
```

8 Further steps

The NicheNet workflow in OmnipathR is implemented until this point. The results can be visualized by the methods included in `nichenetr`, can be analysed for enrichment of functional properties, or the signaling network connecting the ligands to targets can be reconstructed and analysed, as it's shown in the OmniPath case study.

Session info

```
## R Under development (unstable) (2021-05-07 r80268)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Arch Linux
##
## Matrix products: default
## BLAS:   /usr/lib/libopenblasp-r0.3.14.so
## LAPACK: /usr/lib/liblapack.so.3.9.1
##
## locale:
##  [1] LC_CTYPE=en_GB.UTF-8       LC_NUMERIC=C               LC_TIME=en_GB.UTF-8        LC_COLLATE=en_
##  [8] LC_NAME=C                  LC_ADDRESS=C               LC_TELEPHONE=C             LC_MEASUREMENT=
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] rmarkdown_2.7     OmnipathR_2.99.19 testthat_3.0.2    devtools_2.4.1    usethis_2.0.1     setwid
##
## loaded via a namespace (and not attached):
##   [1] utf8_1.2.1            reticulate_1.20       tidyselect_1.1.1      htmlwidgets_1.5.3     grid_4
##  [10] ica_1.0-2             future_1.21.0         miniUI_0.1.1.1        withr_2.4.2           colorsp
##  [19] stats4_4.2.0          ROCR_1.0-11           tensor_1.5            listenv_0.8.0         polycl
##  [28] ipred_0.9-11          xfun_0.22             randomForest_4.6-14   R6_2.5.0              bitops
##  [37] nnet_7.3-16           gtable_0.3.0          globals_0.14.0        processx_3.5.2        goftes
##  [46] lazyeval_0.2.2        ModelMetrics_1.2.2.2  spatstat.geom_2.1-0   checkmate_2.0.0       yaml_2
##  [55] Hmisc_4.5-0           caret_6.0-86          DiagrammeR_1.0.6.1    tools_4.2.0           lava_1
##  [64] RColorBrewer_1.1-2    proxy_0.4-25          sessioninfo_1.1.1     ggridges_0.5.3        Rcpp_1
##  [73] purrr_0.3.4           ps_1.6.0              prettyunits_1.1.1     rpart_4.1-15          deldir
##  [82] SeuratObject_4.0.0    ggrepel_0.9.1         cluster_2.1.2         fs_1.5.0              magrit
##  [91] fitdistrplus_1.1-3    matrixStats_0.58.0    pkgload_1.2.1         evaluate_0.14         hms_1.
## [100] readxl_1.3.1          gridExtra_2.3         compiler_4.2.0        tibble_3.1.1          KernSm
## [109] Formula_1.2-4         tidyr_1.1.3           lubridate_1.7.10      MASS_7.3-54           rappdi
## [118] gower_0.2.2           igraph_1.2.6          pkgconfig_2.0.3       foreign_0.8-81        plotly
## [127] bslib_0.2.4           prodlim_2019.11.13    stringr_1.4.0         callr_3.7.0           digest
## [136] leiden_0.3.7          htmlTable_2.1.0       uwot_0.1.10           curl_4.3.1            shiny_
## [145] desc_1.3.0            viridisLite_0.4.0     fansi_0.4.2           pillar_1.6.0          lattic
## [154] glue_1.4.2            remotes_2.3.0         fdrtool_1.2.16        png_0.1-7             iterat
## [163] caTools_1.18.2        memoise_2.0.0         dplyr_1.0.6           irlba_2.3.3           e1071_
```

References

[1] R Browaeys, W Saelens and Y Saeys (2020) NicheNet: modeling intercellular communication by linking ligands to target genes. *Nat Methods* 17, 159–162

[2] D Turei, A Valdeolivas, L Gul, N Palacio-Escat, M Klein, O Ivanova, M Olbei, A Gabor, F Theis, D Modos, T Korcsmaros and J Saez-Rodriguez (2021) Integrated intra- and intercellular signaling knowledge for multicellular omics analysis. *Molecular Systems Biology* 17:e9923

[3] S Anders, W Huber (2010) Differential expression analysis for sequence count data. *Genome Biol* 11, R106

[4] Y Hao, S Hao, E Andersen-Nissen, WM Mauck, S Zheng, A Butler, MJ Lee, AJ Wilk, C Darby, M Zagar, P Hoffman, M Stoeckius, E Papalexi, EP Mimitou, J Jain, A Srivastava, T Stuart, LB Fleming, B Yeung, AJ Rogers, JM McElrath, CA Blish, R Gottardo, P Smibert and R Satija (2020) Integrated analysis of

multimodal single-cell data. *bioRxiv* 2020.10.12.335331