SecretSanta: flexible pipelines for automated prediction of secreted proteins

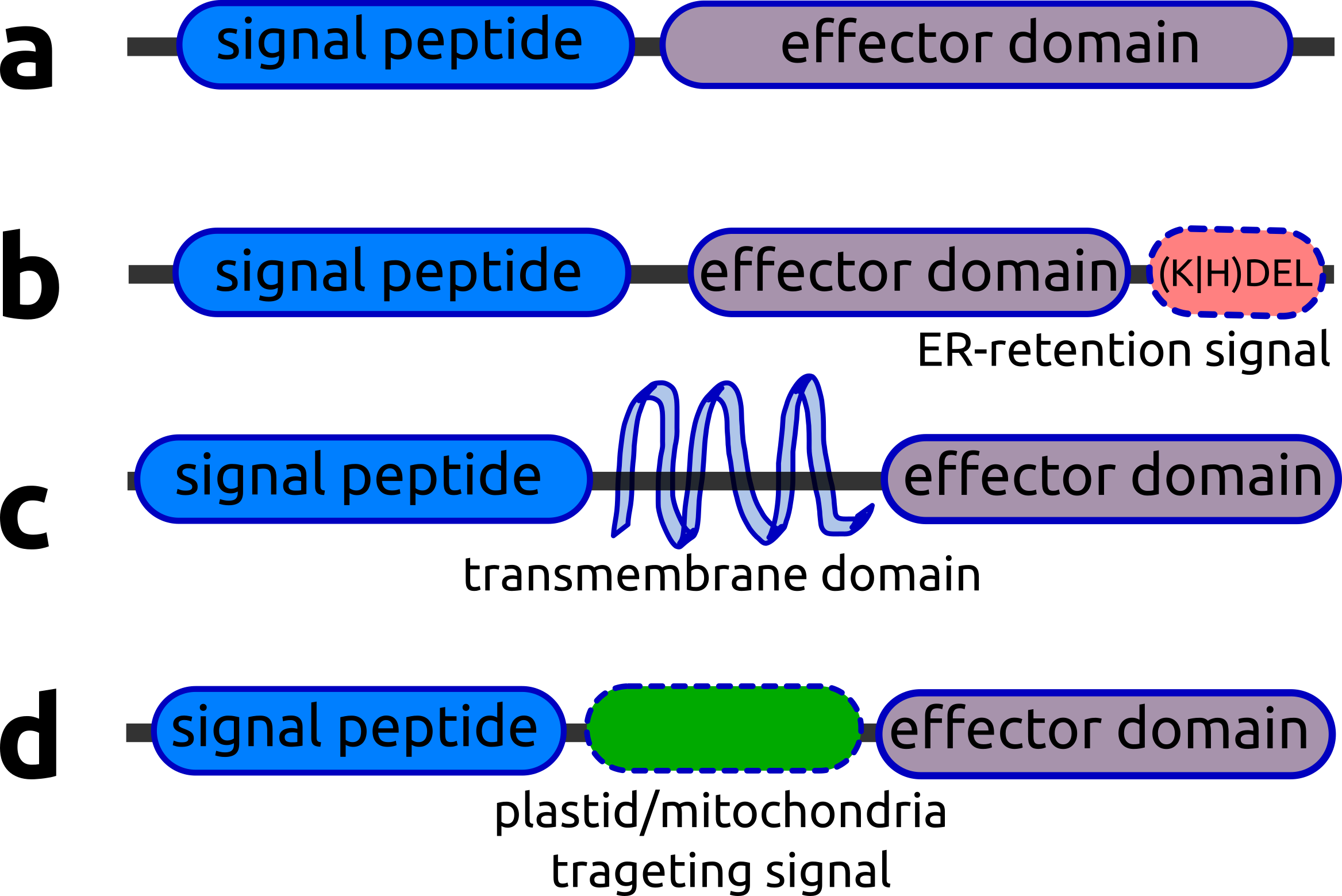
Anna Gogleva

2017-09-12

## 1. Background

**SecretSanta** provides an R interface aiding integrative prediction of extracellular proteins secreted via classical secretion pathway, i.e. requiring presence of a signal peptide.

Secretome prediction often involves multiple steps. Typically it starts with prediction of short signal peptides at the N-terminus end of a protein (**Figure 1, a**). Next, it is crucial to ensure the absence of motifs and domains preventing the protein from being secreted despite the presence of the signal peptide. These sequences include transmembrane domains, short ER lumen retention signals, mitochondria and plastid targeting signals (**Figure 1, b-d**). The ultimate aim of a secretome prediction pipeline is to pick secreted proteins shown in **Figure 1, a** and filer out those shown in **Figure 1, b-d**.



**Figure 1.** Characteristic motifs, domains and their arrangements, helping to distinguish extracellular proteins from proteins retained inside the cell.

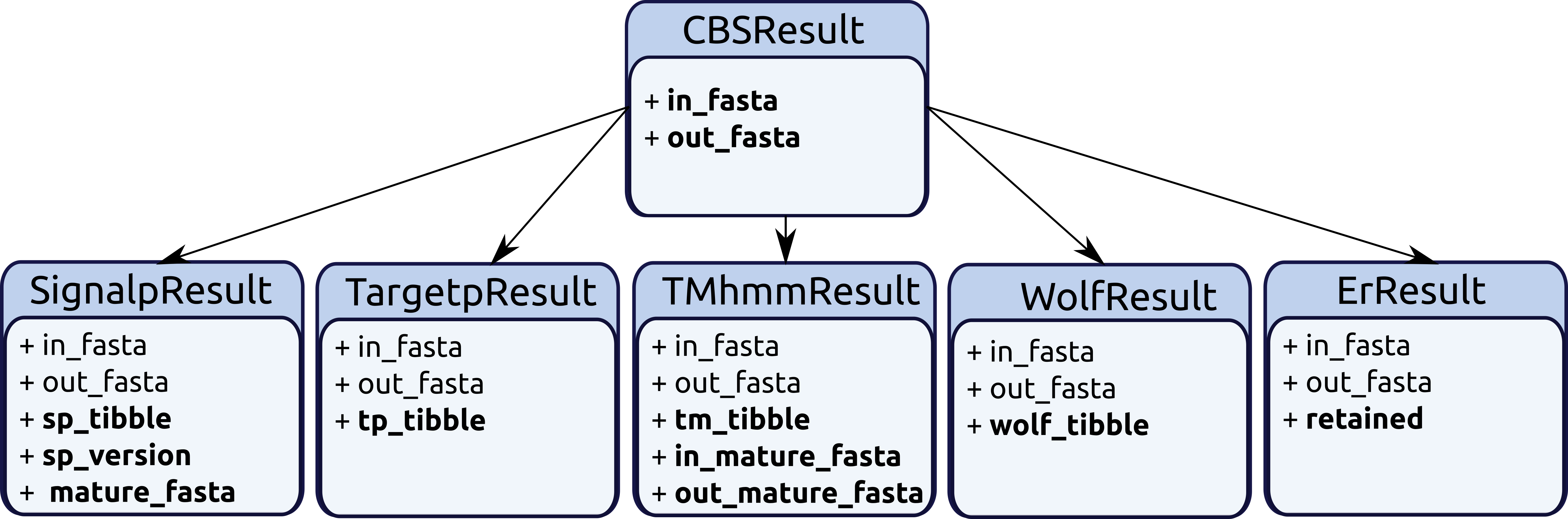
A number of excellent command line tools and web-interfaces exist to perform prediction of individual motifs and domains ([signalp](http://www.cbs.dtu.dk/services/SignalP/), [targetp](http://www.cbs.dtu.dk/services/TargetP/), [TMHMM](http://www.cbs.dtu.dk/services/TMHMM/), [WolfPsort](https://github.com/fmaguire/WoLFPSort)), however the interface allowing to combine the outputs in a single flexible workflow is lacking.

**SecretSanta** package attempts to bridge this gap. It provides wrapper functions around existing command line tools for prediction of signal peptides and protein subcellular localization. The wrappers are designed to work together by producing standardized output as an instance of CBSResult superclass.

Objects of CBSResult class contain in\_fasta slot with the initial submitted sequences and out\_fasta slot with positive candidates after the method application. Each particular method then complements this simple structure with relevant slots. For example signlp function outputs results organised in SignalpResult objects. Apart from in\_fasta and out\_fasta slots inherited from CBSResult class, SignalpResult objects contain three additional slots, relevant for **signalp** method:

* sp\_tibble - parsed **signalp** tabular output for positive candidates;
* mature\_fasta - mature sequence for the candidate secreted proteins, i.e sequences with cleaved N-terminal signal peptides;
* sp\_version - version of **signalp** used to generate this object.

The detailed organisation of class structure is shown in the **Figure 2**.



**Figure 2.** SecretSanta class structure.

This allows to pipe results between individual predictors easily to create flexible custom pipelines and also compare predictions between similar methods. For instance, **targetp** and **WoLFPsort** for subcellular localization and multiple versions of **signalp** for signal peptide prediction.

To speed-up processing of large input fasta files initial steps of the pipeline are automatically run as an embarrassingly parallel process when the number of input sequences exceeds a certain limit.

Taken together **SecretSanta** provides tools aiding automation of multi-step secretome prediction pipelines that can be applied to large protein sets to facilitate comparison of secretomes across multiple species.

To avoid confusion, here and further, names of external command line tools will be shown in **bold**, the corresponding R functions from the **SecretSanta** package will have standard code highlighting, for example:

* **signalp** - CBS signalp and its associated derivative files;
* signalp - **SecretSanta** wrap function around **signalp**;

Majority of the **SecretSanta** functions could be run in 2 modes:

* starter - in the case you initiate secretome pipeline with the above function;
* piper - for downstream/intermediate steps, so that the function expects an output from another wrapper function.

## 2. Installation of external dependencies

For prediction of secreted proteins SecretSanta relies on a set of existing command line tools. Please install them and configure according to the listed instructions.

### 2.1 Download and configure external dependencies

#### Tools for prediction of signal peptides and cleavage sites:

* **signalp-2.0**
  + This version can run under IRIX, IRIX64, Linux, OSF1, SunOS.
  + Download stand alone signalp2.0 <http://www.cbs.dtu.dk/cgi-bin/sw_request?signalp+2.0>
  + Unpack the archive
* tar -zxvf signalp-2.0.Linux.tar.Z  
  cd signalp-2.0
  + Edit "General settings" at the top of the **signalp** file. Set value of 'SIGNALP' variable to be path to your **signalp-2.0** directory. Other variables usually do not require changes. We will not use plotting functions from signalp, so gnuplot, ppmtogif and ghostview are not required. For more details please check signalp-2.0.readme.
  + Since, we want to be able to run different versions of **signalp**, including the legacy ones, it is important to be able to discriminate between them. R is oblivious to shell aliases, so we will simply rename the **siganlp** script:
* mv signalp siganlp2
* **signalp-3.0**
  + This version will run on the most common UNIX platforms.
  + Download stand alone signalp3.0 <http://www.cbs.dtu.dk/cgi-bin/sw_request?signalp+3.0>
  + Unpack the archive
* tar -zxvf signalp-3.0.Linux.tar.Z  
  cd signalp-3.0
  + Similar to **signalp-2.0**, edit "General settings" at the top of the signalp file. Set value of 'SIGNALP' variable to be path to your **signalp-3.0** directory. Other variables usually do not require changes. For more details please check signalp-3.0.readme.
  + Rename **signalp** script to avoid further confusion between the versions:
* mv signalp signalp3
* **signalp-4.1** - the most recent version
  + This version can run under Windows, OS X (Macintosh) and Linux.
  + Download stand alone signalp4.0 <http://www.cbs.dtu.dk/cgi-bin/sw_request?signalp+4.0>
  + Unpack the archive
* tar -zxvf signalp-4.1.Linux.tar.Z  
  cd signalp-4.1
  + Edit "General settings" at the top of the **signalp** file. Set values for 'SIGNALP' and 'outputDir' variables. For more details please check signalp-4.1.readme.
  + Rename **signalp** script to avoid further confusion between the versions:
* mv signalp signalp4

#### Tools for prediction of protein subcellular localization:

* **taretp-1.1**
  + **tatgetp-1.1** will run on the most common UNIX platforms
  + Download stand alone targetp <http://www.cbs.dtu.dk/cgi-bin/nph-sw_request?targetp>
  + Unpack the archive:
* tar -zxvf targetp-1.1b.Linux.tar.Z   
  cd targetp-1.1
  + Edit the paragraph labeled "GENERAL SETTINGS, customize" at the top of the **targetp** file. Set values for 'TARGETP' and 'TMP' variables. Ensure, that the path to **targetp** does not exceed 60 characters, otherwise **targetp-1.1** might fail.
* **WoLFPsort**
  + Clone WoLFPsort
* git clone https://github.com/fmaguire/WoLFPSort.git  
  cd WoLFPSort
  + Copy the binaries from the appropriate platform specific binary directory ./bin/binByPlatform/binary-? to ./bin/
  + For more details please check the INSTALL file.
  + The most important script we need **runWolfPsortSummary** has a bulky name, we will rename it to simply **wolfpsort** for the future convenience:
* mv runWolfPsortSummary wolfpsort

#### Tools for prediction of transmembrane domains

* **tmhmm-2.0**
  + **tmhmm-2.0** will run on the most common UNIX platforms
  + Download stand alone tmhmm (<http://www.cbs.dtu.dk/cgi-bin/nph-sw_request?tmhmm>)
  + Unpack the archive:
* tar -zxvf tmhmm-2.0c.Linux.tar.gz  
  cd tmhmm-2.0c
  + Set correct path for Perl 5.x in the first line of bin/tmhmm and bin/tmhmmformat.pl scripts.
  + For more details please check the README file.

### 2.2 Organise access to the external dependecies

Two options are possible:

#### Option A: all the external dependencies are accessible from any location.

This requires modification of $PATH environment variable. To make the change permanent, edit .profile:

Open ./profile:

gedit ~/.profile

Add a line with all the path exports. In this example all the dependencies are installed in the my\_tool directory:

export PATH=  
"/home/my\_tools/signalp-4.1:\  
/home/my\_tools/signalp-2.0:\  
/home/my\_tools/signalp-3.0:\  
/home/my\_tools/targetp-1.1:\  
/home/tmhmm-2.0c/bin:\  
/home/my\_tools/WoLFPSort/bin:\  
$PATH"

Reload /.profile:

. ~/.profile

Reboot, to make changes visible to R, alternatively - restart the R session.

#### Option B: paths to external dependencies are supplied in a separate file:

This option should be used when $PATH variable can not be modified. Create a 2-column space-separated text file with listed paths for all the external dependencies.

**Table1**: Sample paths to external dependencies

|  |  |
| --- | --- |
| tool | path |
| signalp4 | /home/tools/signalp-4.1/signalp |
| signalp3 | /home/tools/signalp-3.0/signalp |
| signalp2 | /home/tools/signalp-2.0/signalp |
| targetp | /home/tools/targetp-1.1/targetp |
| TMHMM | /home/tools/tmhmm-2.0c/bin/tmhmm |
| WoLFPSORT | /home/tools/WoLFPSort-master/bin/wolfpsort |

For external tool names please use simple character strings without version numbers. The only exception here are different versions of **signalp**, which require version number in one-digit format in the tool column to be distinguishable.

### 2.3 Check that all the required dependencies are installed correctly

library(SecretSanta)

manage\_paths function will run small test jobs and check that all the external dependencies are functional.

In case all the dependencies are available globally (see **Option A**):

check1 <- manage\_paths(in\_path = TRUE)  
## checking dependencies acessible via $PATH  
## signalp2 run completed  
## signalp3 run completed  
## signalp4 run completed  
## tmhmm run completed  
## targetp run completed  
## wolfpsort run completed

It is also possible to run just a single test for a specific dependency in question:

check1 <- manage\_paths(in\_path = TRUE,  
 test\_mode = 'signalp2')  
## checking dependencies acessible via $PATH  
## signalp2 run completed

In case we have paths organised in a separate file (see **Option B**):

check2 <- manage\_paths(  
 in\_path = FALSE,  
 path\_file = system.file("extdata",  
 "sample\_paths",  
 package = "SecretSanta")  
 )  
## All paths are valid  
## signalp2 run completed  
## signalp3 run completed  
## signalp4 run completed  
## tmhmm run completed  
## targetp run completed  
## wolfpsort run completed

Note, that manage\_paths is case insensitive and will convert all the tool names to the lower case.

## 3. Individual methods

### 3.1 Signalp

**signalp** is a software tool to predict classical signal peptides and cleavage sites (Henrik Nielsen and Heijne 1997) in eukaryotes and bacteria. The method combines prediction of signal peptides and cleavage cites based on a combination of artificial neural networks and hidden Markov models. The latter allows to distinguish between signal peptides and non-cleaved signal anchors(Henrik Nielsen and Heijne 1997).

Currently signalp function from SecretSanta provides an interface for the three recent versions of **signalp**. The reason behind this is that **signalp-4** and **signalp-4.1** are not sensitive enough to predict certain classes of secreted oomycete and fungal effectors (Sperschneider J 2015). Thus, providing access to the older versions allows greater flexibility for building versatile prediction pipelines for a wider taxonomic range of species.

* **signalp-2.0** (Nielsen and Krogh 1998)
* **signalp-3.0** (Dyrl�v Bendtsen and Brunak 2004)
* **signalp-4.0** (Thomas Nordahl Petersen and Nielsen. 2011)

signalp requires specifying organism argument; possible values include:

* *'euk'* - for eukaryotes;
* *'gram+'* - for gram-positive bacteria;
* *'gram-'* - for gram-negative bacteria.

To run signalp prediction, first read fasta file with amino acid sequences and store its contents in a separate variable:

aa <- readAAStringSet(system.file("extdata",  
 "sample\_prot\_100.fasta",  
 package = "SecretSanta"))

Initialize object of CBSResult class with aa as in\_fasta attribute.

inp <- CBSResult(in\_fasta = aa)

Since this is the first step in our secretome prediction pipeline, we will run signalp in a *starter* mode. Here we select version number 2 (**signalp-2.0**) and assume that signalp is accessible globally, so no path files should be specified. If **signalp** can not be accessed globally, this will result in an error message.

step1\_sp2 <- signalp(inp,  
 version = 2,  
 organism = 'euk',  
 run\_mode = "starter")  
## Version used... signalp2  
## running signalp locally...  
## 2 sequences to be truncated  
## Ok for single processing  
## Submitted sequences... 100  
## signalp < 4, calling parser for the output...  
## Candidate sequences with signal peptides... 8

The above code under the hood runs **signalp-2.0** prediction and outputs result as an instance of SignalpResult class, which belongs to a more generic CBSResult() superclass having just 2 slots: in\_fasta and out\_fasta.

class(step1\_sp2)

## [1] "SignalpResult"  
## attr(,"package")  
## [1] "SecretSanta"

slotNames(step1\_sp2)

## [1] "mature\_fasta" "sp\_version" "sp\_tibble" "in\_fasta"   
## [5] "out\_fasta"

SignalpResult object contains 5 slots:

* in\_fasta - original set of input amino acid sequences;
* sp\_tibble - parsed **signalp** tabular output for positive candidates;
* out\_fasta - full length amino acid sequences of positive candidates;
* mature\_fasta - mature sequence for the candidate secreted proteins, i.e sequences with cleaved N-terminal signal peptides;
* sp\_version - version of **signalp** used to generate this object.

You can use accessor methods to get contents of individual slots:

getInfasta(step1\_sp2)

## A AAStringSet instance of length 100  
## width seq names   
## [1] 170 MQTDQRFKIKAPTATSTGLY...RPCAIEEHDIQVLQADNAN ALI\_PLTG\_54  
## [2] 230 MNVALVRERALHTMNRPGET...GNLVLYRICQMTPLLFCIC ALI\_PLTG\_92  
## [3] 454 MLYVKLSKCVFCAEEFPCLG...DRLTKRSHVIATTKKVSAP ALI\_PLTG\_68  
## [4] 239 MSLNYLAHTNMRRARENFLK...KTYIEYEAVENNTLNKTEG ALI\_PLTG\_98  
## [5] 207 MYDMKSDVDARVMNYFHLCN...VLIPELKFRVLFYDTKKDW ALI\_PLTG\_80  
## ... ... ...  
## [96] 210 MKLSSVCTIIFGLVFIDFNN...DGSVTTAAQKAAALVKANA ALI\_PLTG\_5  
## [97] 430 MKASYKETISPIPKSDQKAS...RRLMGLLKERAFFPLSEFG ALI\_PLTG\_78  
## [98] 284 MKERVCSVCDCFHRIDRMKF...KNLPQITPIEQFLACKMLK ALI\_PLTG\_34  
## [99] 137 MSEKATATMRFLRGGDINDN...AMMKKKKLRSFAEELQKMV ALI\_PLTG\_27  
## [100] 157 MTIHSLLGGVYETRAAAFDP...VGINTLVNDLAACEQPLNQ ALI\_PLTG\_43

Please note, that by default signlap truncates all the input sequences longer than 2000 a.a, in this case '\_truncated' is added to the original sequence ids to keep track of the changes. Alternative option is to discard long sequences completely before the analysis, to do so set truncate = FALSE when running signalp.

getSPtibble(step1\_sp2)

## # A tibble: 8 x 9  
## gene\_id Cmax Cpos Ymax Ypos Smax Spos Smean Prediction  
## <fctr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <fctr>  
## 1 ALI\_PLTG\_23 0.641 22 0.683 22 0.969 8 0.803 Signal peptide  
## 2 ALI\_PLTG\_83 0.393 34 0.506 20 0.956 13 0.846 Signal peptide  
## 3 ALI\_PLTG\_81 0.305 33 0.388 33 0.826 25 0.471 Signal peptide  
## 4 ALI\_PLTG\_38 0.572 20 0.696 20 0.994 9 0.954 Signal peptide  
## 5 ALI\_PLTG\_73 0.489 22 0.316 22 0.927 3 0.721 Signal peptide  
## 6 ALI\_PLTG\_59 0.210 17 0.288 19 0.948 1 0.666 Signal peptide  
## 7 ALI\_PLTG\_72 0.887 22 0.804 22 0.986 8 0.805 Signal peptide  
## 8 ALI\_PLTG\_5 1.000 25 0.687 25 0.906 3 0.637 Signal peptide

getOutfasta(step1\_sp2)

## A AAStringSet instance of length 8  
## width seq names   
## [1] 653 MKIVALVTFCIATLDSSIVFA...SEDPVYPLVKEYSDVVSKHP ALI\_PLTG\_23  
## [2] 164 MVSTSRVFALLLLPSPSARIF...SMVEHIKTTKRVIDEVQDHV ALI\_PLTG\_83  
## [3] 133 MNFLKSKKTSTQATLLVSPLI...MGGLMPNLHVDLENGIGNYP ALI\_PLTG\_81  
## [4] 495 MVVRVLLVVLALLAVGVQSKA...FAIVTIGASDGRVRYMNPPS ALI\_PLTG\_38  
## [5] 154 MYLKFAFMATVATAVVNIAIG...YAKLLLAYLNDFKRTSGLRS ALI\_PLTG\_73  
## [6] 125 MLMSTPAALMLRLAQSSLKPT...DQGAHVATSGTSNNVLLWHK ALI\_PLTG\_59  
## [7] 118 MRLKFSILIFGAVLLATTTNA...NIRYASMLQDFLNTYHRRGV ALI\_PLTG\_72  
## [8] 210 MKLSSVCTIIFGLVFIDFNNV...SDGSVTTAAQKAAALVKANA ALI\_PLTG\_5

getMatfasta(step1\_sp2)

## A AAStringSet instance of length 8  
## width seq names   
## [1] 632 AECTVDELTEISTIYSEAMTD...SEDPVYPLVKEYSDVVSKHP ALI\_PLTG\_23  
## [2] 131 ITTKLKQFNGTGFPAWGVQDK...SMVEHIKTTKRVIDEVQDHV ALI\_PLTG\_83  
## [3] 101 YGPPADVRPMVVRMKSDYRSY...MGGLMPNLHVDLENGIGNYP ALI\_PLTG\_81  
## [4] 476 KATRVRKSWTAYTSDEKEIYL...FAIVTIGASDGRVRYMNPPS ALI\_PLTG\_38  
## [5] 133 NVIDLPMPLTLESPNFVESAA...YAKLLLAYLNDFKRTSGLRS ALI\_PLTG\_73  
## [6] 109 SLKPTVGKAITVGGNHQIPTK...DQGAHVATSGTSNNVLLWHK ALI\_PLTG\_59  
## [7] 97 DDNSNKRLLRRQEKTDTTNGD...NIRYASMLQDFLNTYHRRGV ALI\_PLTG\_72  
## [8] 186 QCSDVHVVFARGSGEAAGLGI...SDGSVTTAAQKAAALVKANA ALI\_PLTG\_5

getSPversion(step1\_sp2)

## [1] 2

Next, imagine we would like to run all the **signalp** version on the same input for comparison. We can do this by simply changing version value:

step1\_sp3 <- signalp(inp,  
 version = 3,  
 organism = 'euk',  
 run\_mode = "starter")  
## Version used... signalp3  
## running signalp locally...  
## 2 sequences to be truncated  
## Ok for single processing  
## Submitted sequences... 100  
## signalp < 4, calling parser for the output...  
## Candidate sequences with signal peptides... 8  
step1\_sp4 <- signalp(inp,  
 version = 4,  
 organism = 'euk',  
 run\_mode = "starter")  
## Version used... signalp4  
## running signalp locally...  
## 2 sequences to be truncated  
## Ok for single processing  
## Submitted sequences... 100  
## Candidate sequences with signal peptides... 5

In this case signalp-4.1 version resulted in fewer candidates. Please note, that despite differences in the output format generated by multiple versions of **signalp**, signalp returns sp\_tibble in a standard format.

In order to pipe result from different versions of **signalp** just switch to the run\_mode = 'piper' for the second and other downstream steps. In this case signalp will run the analysis using contents of out\_fasta slot as an input. Say, we want to do the following piping: **signalp2** -> **signalp3** -> **signalp4**.

We will re-use step1\_sp2 object generated earlier in the starter mode for the **signalp2** -> **signalp3** piping operation:

step2\_sp3 <- signalp(step1\_sp2,  
 version = 3,  
 organism = 'euk',  
 run\_mode = "piper")  
## Version used... signalp3  
## running signalp locally...  
## Ok for single processing  
## Submitted sequences... 8  
## signalp < 4, calling parser for the output...  
## Candidate sequences with signal peptides... 8

Similar with the **signalp3** -> **signalp4** piping:

step3\_sp4 <- signalp(step2\_sp3,  
 version = 4,  
 organism = 'euk',  
 run\_mode = "piper")  
## Version used... signalp4  
## running signalp locally...  
## Ok for single processing  
## Submitted sequences... 8  
## Candidate sequences with signal peptides... 5

With the input fasta files containing more than 1000 sequences, signalp will automatically switch to parallel mode. It will split the input into smaller chunks and run prediction as an embarrassingly parallel process using specified number of CPUs (see cores argument). Execution time depends on the number if CPUs available and the input file size. With 48 CPUs it takes ~2 minutes to run signalp on the input fasta file with more than 40'000 sequences.

### 3.2 TMHMM

**TMHMM** predicts transmembrane a-helices and identifies integral membrane proteins based on HMMs [Krogh, Bjorn Larsson, and Sonnhammer (2001); Sonnhammer1998]. It is important to exclude proteins with transmembrane domains located after signal peptide, as they will be retained in the membrane. Since often **TMHMM** is not able to distinguish between N-terminal signal peptides and transmembrane domains, it is recommended to run tmhmmm on the mature sequences obtained after running signalp function.

tmhmm function can handle input objects of SignalpResult class with non-empty mature\_fasta slot.

Here we will use output of signalp function as an input for tmhmm:

tm <- tmhmm(step1\_sp2, TM = 0)  
## running TMHMM locally...  
## Submitted sequences... 8  
## Candidates with signal peptides and 0 TM domains in mature seq 8

Attempts to run tmhmm on the CBSResult object lacking mature\_fasta slot will produce an error:

tm2 <- tmhmm(inp, TM = 0)  
## Error in tmhmm(inp, TM = 0): input\_object does not belong to SignalpResult class

tmhmm outputs instances of TMhmmResult class, inheriting from CBSResult.

class(tm)

## [1] "TMhmmResult"  
## attr(,"package")  
## [1] "SecretSanta"

slotNames(tm)

## [1] "in\_mature\_fasta" "out\_mature\_fasta" "tm\_tibble"   
## [4] "in\_fasta" "out\_fasta"

TMhmmResult object contains 5 slots:

* in\_fasta and out\_fasta are common attributes of all classes inheriting from the CBSResult class;
* tm\_tibble - parsed **tmhmm** tabular output for candidates not having more transmembrane domains than specified by TM threshold;
* in\_mature\_fasta - mature sequences used as an input for tmhmm
* out\_mature\_fasta - outputted mature sequences not having more transmembrane domains than specified by TM threshold.

To get contents of individual slots you could use accessor functions:

getTMtibble(tm)

## # A tibble: 8 x 6  
## gene\_id length ExpAA First60 PredHel Topology  
## <chr> <dbl> <dbl> <dbl> <dbl> <chr>  
## 1 ALI\_PLTG\_23 632 0.00 0.00 0 o  
## 2 ALI\_PLTG\_83 131 1.44 0.02 0 o  
## 3 ALI\_PLTG\_81 101 0.00 0.00 0 o  
## 4 ALI\_PLTG\_38 476 0.13 0.09 0 o  
## 5 ALI\_PLTG\_73 133 0.00 0.00 0 o  
## 6 ALI\_PLTG\_59 109 0.00 0.00 0 o  
## 7 ALI\_PLTG\_72 97 0.00 0.00 0 i  
## 8 ALI\_PLTG\_5 186 1.29 1.16 0 o

getInMatfasta(tm)

## A AAStringSet instance of length 8  
## width seq names   
## [1] 632 AECTVDELTEISTIYSEAMTD...SEDPVYPLVKEYSDVVSKHP ALI\_PLTG\_23  
## [2] 131 ITTKLKQFNGTGFPAWGVQDK...SMVEHIKTTKRVIDEVQDHV ALI\_PLTG\_83  
## [3] 101 YGPPADVRPMVVRMKSDYRSY...MGGLMPNLHVDLENGIGNYP ALI\_PLTG\_81  
## [4] 476 KATRVRKSWTAYTSDEKEIYL...FAIVTIGASDGRVRYMNPPS ALI\_PLTG\_38  
## [5] 133 NVIDLPMPLTLESPNFVESAA...YAKLLLAYLNDFKRTSGLRS ALI\_PLTG\_73  
## [6] 109 SLKPTVGKAITVGGNHQIPTK...DQGAHVATSGTSNNVLLWHK ALI\_PLTG\_59  
## [7] 97 DDNSNKRLLRRQEKTDTTNGD...NIRYASMLQDFLNTYHRRGV ALI\_PLTG\_72  
## [8] 186 QCSDVHVVFARGSGEAAGLGI...SDGSVTTAAQKAAALVKANA ALI\_PLTG\_5

getOutMatfasta(tm)

## A AAStringSet instance of length 8  
## width seq names   
## [1] 632 AECTVDELTEISTIYSEAMTD...SEDPVYPLVKEYSDVVSKHP ALI\_PLTG\_23  
## [2] 131 ITTKLKQFNGTGFPAWGVQDK...SMVEHIKTTKRVIDEVQDHV ALI\_PLTG\_83  
## [3] 101 YGPPADVRPMVVRMKSDYRSY...MGGLMPNLHVDLENGIGNYP ALI\_PLTG\_81  
## [4] 476 KATRVRKSWTAYTSDEKEIYL...FAIVTIGASDGRVRYMNPPS ALI\_PLTG\_38  
## [5] 133 NVIDLPMPLTLESPNFVESAA...YAKLLLAYLNDFKRTSGLRS ALI\_PLTG\_73  
## [6] 109 SLKPTVGKAITVGGNHQIPTK...DQGAHVATSGTSNNVLLWHK ALI\_PLTG\_59  
## [7] 97 DDNSNKRLLRRQEKTDTTNGD...NIRYASMLQDFLNTYHRRGV ALI\_PLTG\_72  
## [8] 186 QCSDVHVVFARGSGEAAGLGI...SDGSVTTAAQKAAALVKANA ALI\_PLTG\_5

### 3.3 Targetp

**Targetp** predicts subsellular localization of secreted eukaryotic proteins based on the presence of signal peptide (SP), chloroplast transit peptide (cTP) or mitochondrial targeting peptide (mTP) in the N-terminus (Olof Emanuelsson and Heijne 2008). Including **targetp** in the pipeline can provide additional evidence that a protein with a predicted signal peptide is not targeted to plastids or mitochondria and is indeed extracellular. targetp function accepts input objects belonging to the CBSResult class.

targetp requires specifying network\_type to run correctly. Possible values include:

* *'P'* - for plants;
* *'N'* - for non\_plants.

It is possible to run targetp in both *piper* and *starter* modes. Imagine, we want to start our pipeline by running targetp using already created inp object as an input:

tp <- targetp(inp,  
 network = 'N', # for non-plant networks  
 run\_mode = 'starter')  
## running targetp locally...  
## Ok for single processing  
## Number of submitted sequences... 100  
## Number of candidate secreted sequences 10

Alternatively, we can run targetp on the output of other functions, for example signalp in the *piper* mode:

tp\_pipe <- targetp(step1\_sp2,  
 network = 'N',  
 run\_mode = 'piper'  
 )  
## running targetp locally...  
## Ok for single processing  
## Number of submitted sequences... 8  
## Number of candidate secreted sequences 6

In both cases targetp will output an object of TargetpResult class with the following slots:

class(tp)

## [1] "TargetpResult"  
## attr(,"package")  
## [1] "SecretSanta"

slotNames(tp)

## [1] "tp\_tibble" "in\_fasta" "out\_fasta"

* in\_fasta and out\_fasta are common attributes of all classes inheriting from CBSResult class;
* tp\_tibble - parsed tragetp tabular output for candidates not targeted to mitochondria or plastids, i.e most likely to be secreted.

To access tp\_tibble slot use getTPtibble method:

getTPtibble(tp)

## # A tibble: 10 x 7  
## gene\_id length mTP sp other TP\_localization RC  
## <fctr> <int> <dbl> <dbl> <dbl> <fctr> <int>  
## 1 ALI\_PLTG\_5 210 0.027 0.944 0.096 S 1  
## 2 ALI\_PLTG\_19 957 0.176 0.359 0.306 S 5  
## 3 ALI\_PLTG\_23 653 0.025 0.953 0.072 S 1  
## 4 ALI\_PLTG\_38 495 0.123 0.951 0.012 S 1  
## 5 ALI\_PLTG\_69 164 0.138 0.527 0.267 S 4  
## 6 ALI\_PLTG\_72 118 0.086 0.877 0.054 S 2  
## 7 ALI\_PLTG\_73 154 0.035 0.938 0.108 S 1  
## 8 ALI\_PLTG\_74 419 0.024 0.922 0.257 S 2  
## 9 ALI\_PLTG\_77 347 0.090 0.475 0.365 S 5  
## 10 ALI\_PLTG\_81 133 0.134 0.581 0.133 S 3

targetp will automatically switch to a parallel mode if number of input sequences is grater than 1000.

### 3.4 WoLFPSORT

**WoLFPSORT** - predicts protein subcellular localization based on PSORT principle. It converts amino acid sequences into numerical localization features based on sorting signals, amino acid composition and functional motifs. The converted data is then classified based on k-nearest neighbor algorithm (Paul Horton and Nakai 2006).

wolfpsort requires organism parameter, it's possible values include:

* *'plant'*;
* *'animal'*;
* *'fungi'*.

wolfpsort function accepts objects of CBSResult class with non-empty out\_fasta slot, so it is recommended to run this prediction on later stages of the pipeline.

wlf <- wolfpsort(step1\_sp2,  
 organism = 'fungi')  
## running WoLF PSORT locally...  
## Number of submitted sequences... 8  
## Candidate sequences with extracellular localisation... 5

wolfpsort returns an object with 3 slots:

class(wlf)

## [1] "WolfResult"  
## attr(,"package")  
## [1] "SecretSanta"

slotNames(wlf)

## [1] "wolf\_tibble" "in\_fasta" "out\_fasta"

* in\_fasta and out\_fasta are common attributes of all classes inheriting from CBSResult class;
* wolf\_tibble - parsed WoLFPsort tabular output for sequences with 'extracellular' predicted to be the most probable subcellular localization.

To access contents of wolf\_tibble slot use getWOlFtibble method:

getWOLFtibble(wlf)

## # A tibble: 5 x 2  
## gene\_id localization  
## <fctr> <fctr>  
## 1 ALI\_PLTG\_23 extr  
## 2 ALI\_PLTG\_81 extr  
## 3 ALI\_PLTG\_38 extr  
## 4 ALI\_PLTG\_73 extr  
## 5 ALI\_PLTG\_5 extr

Since wolfpsort has the same purpose as targetp it might be useful to run these functions side by side and compare the obtained results (see [targetp results](#Targetp)).

### 3.5 Check the presence of terminal ER-retention motifs

In addition to having signal peptides some proteins might have ER-retention signal in the C-terminal domain, preventing protein from being secreted outside the cell. There are at least 2 known ER-retention motifs (*KDEL* and *HDEL*) (Munro S 1987).

The check\_khdel function uses simple pattern matching to scan amino acid sequences and remove those with ER retention signal in the C-terminus. To run it, simply pass a CBSResult object as an input. You can run check\_khdel in either *piper* or *starter* mode. The function does not rely on external dependencies, so providing path container is not required.

er\_result <- check\_khdel(step1\_sp2,  
 run\_mode = 'piper')  
## checking for terminal ER retention signals...  
## Submitted sequences... 8  
## Sequences with terminal ER retention signals detected... 0  
## Candidate without terminal ER retention signals detected... 8

This will return and object with 3 slots:

slotNames(er\_result)

## [1] "retained" "in\_fasta" "out\_fasta"

* in\_fasta and out\_fasta are common attributes of all classes inheriting from CBSResult class;
* retained - all the sequences with C-terminal '(K/H)DEL' motifs, in case you would like to have a look at them.

### 3.6 M-slicer

This is an experimental option. m\_slicer function takes the input amino acid sequences and generates all possible subsequences starting with methionine based on the assumption that translation start sites might be mis-predicted in the original set of proteins resulting in signal peptides also being mis-predicted. Output of this step can be used as an input for secretome prediction pipeline to rescue secreted proteins with mis-predicted start sites. Sequence ids for the newly generated slices are build by concatenating original sequence id, 'slice' string and position of the methionine sliced from. For example: ALI\_PLTG\_3\_slice\_M86 means that ALI\_PLTG\_3 sequence was sliced starting from the methionine in the position 86.

m\_slicer has 2 running modes:

* slice - to simply slice input fasta regardless of it's origin;

slice <- m\_slicer(aa, # a set of amino acid seqeunces  
 run\_mode = 'slice',  
 min\_len = 100 # minimal length of the outputed slices  
)

* rescue - having output from any other up-stream function it extracts proteins not predicted to be secreted on the initial run and generate slices for them.

rescued <- m\_slicer(step1\_sp2, # signalp2 output  
 run\_mode = 'rescue',  
 min\_len = 100  
)

Results of both run modes could be used as an input for other SecretSanta predictors. Say, we want to run signalp on the rescued proteins. Note, that m\_slicer outputs AAStringSet objects, so in order to pass it to signalp we first need to create CBSResult object with in\_fasta = rescued.

r\_inp <- SignalpResult(in\_fasta = rescued)  
sp\_rescue <- signalp(r\_inp,  
 version = 2,  
 organism = 'euk',  
 run\_mode = 'starter',  
 truncate = TRUE)  
## Version used... signalp2  
## running signalp locally...  
## Ok for single processing  
## fasta size exceedes maximal total residue limit, sequences longer 416 will be truncated  
## 206 sequences to be truncated  
## Submitted sequences... 546  
## signalp < 4, calling parser for the output...  
## Candidate sequences with signal peptides... 15

The slicing procedure returned 15 additional potentially secreted candidates.

## 4. Pipers and/or starters

Here is a short summary of individual methods available via the **SecretSanta** package:

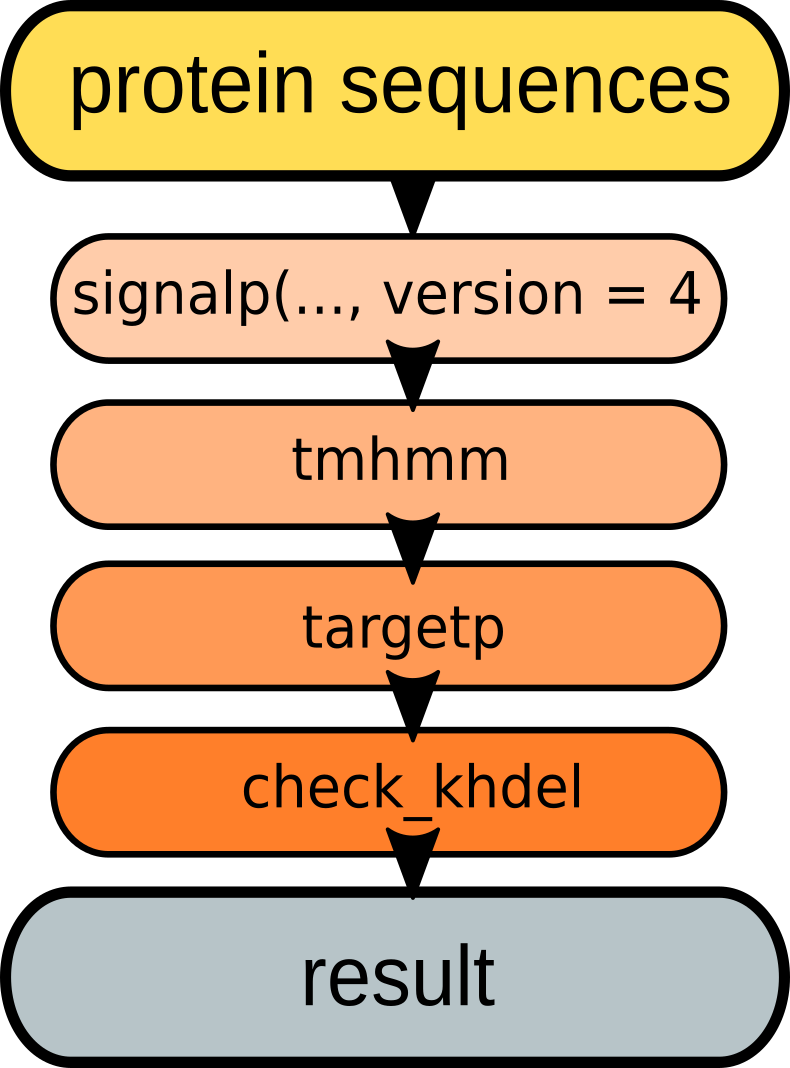
**Table2**: Individual methods summary.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| tool | function | purpouse | organisms | starter | piper | parallelised |
| Signalp-2.0 | signalp | predict signal peptides | eukaryotes, gram+, gram- | Yes | Yes | Yes |
| Signalp-3.0 | signalp | predict signal peptides | eukaryotes, gram+, gram- | Yes | Yes | Yes |
| Signalp-4.1 | signalp | predict signal peptides | eukaryotes, gram+, gram- | Yes | Yes | Yes |
| Targetp-1.1 | targetp | predict subcelluar localisation | plants, not plants | Yes | Yes | Yes |
| wolfpsort | wolfpsort | predict subcelluar localisation | plant, animal, fungi | No | Yes | No |
| Tmhmm-2.0 | tmhmm | predict transmembrane domains | all taxons | No | Yes | Yes |
| NA | check\_khdel | scan for ER-retention motifs | all taxons | Yes | Yes | No |
| NA | m\_slicer | generate sequences with alternative translation start sites | all taxons | Yes | Yes | No |

## 5. Build pipelines

Now, we are ready to build a pipeline using all our available methods. Say, we would like to have a relatively short pipeline (**Figure 2**) with the following analytic steps:

* use **signalp4** to predict signal peptides and cleavage sites;
* run **tmhmm** on the output, to ensure that predicted proteins with signal peptides do not contain TM domains, i.e won't be stuck in the membrane;
* run **targetp** on the output, to make sure that a set of selected proteins is not targeted to plastids or mitochondria;
* collect the targtep output and scan for ER-retention signals - to ensure that proteins won't be stuck in ER.



**Figure 3.** Minimal pipeline for secretome prediction

Now we will run this pipeline using SecretSanta functions, starting with aa - an AAStringSet object containing 100 amino acid sequences.

#### Step 1: predict signal peptides

input <- CBSResult(in\_fasta = aa)  
  
input  
## An object of class "CBSResult"  
## Slot "in\_fasta":  
## A AAStringSet instance of length 100  
## width seq names   
## [1] 94 MVEPSSPECVVQIIKDEKLK...LLVCGFIYFAVLQSWLLES ALI\_PLTG\_1 Hypoth...  
## [2] 240 MTRQGYLIVHDKRSRPTVRY...VTPHYVPTHFGRGNNQEAH ALI\_PLTG\_2 Unknow...  
## [3] 533 MQPTSSSAPQSDVVVADPPS...QQQQEQQANSTALKNERSI ALI\_PLTG\_3 Hypoth...  
## [4] 456 QFLSELRKVSVSTKLDGVQR...QSQQNSRPREGCCSSCIIF ALI\_PLTG\_4 Hypoth...  
## [5] 210 MKLSSVCTIIFGLVFIDFNN...DGSVTTAAQKAAALVKANA ALI\_PLTG\_5 Cutin ...  
## ... ... ...  
## [96] 650 MQRLVQRDKDAPVVREGDDD...DRLSLNALLEAFPPMNSRL ALI\_PLTG\_96 Prote...  
## [97] 236 MSTKRSRSAPILRVKKLTPE...DDDNSAKVLEWINAFIESN ALI\_PLTG\_97 Deoxy...  
## [98] 239 MSLNYLAHTNMRRARENFLK...KTYIEYEAVENNTLNKTEG ALI\_PLTG\_98 Hypot...  
## [99] 566 MCTSFSDQEDKKLVVLATEY...PRLDRKFFGLQVAPPPTSQ ALI\_PLTG\_99 Histo...  
## [100] 180 MDNYYTSVQLLLDLQLKGLC...DPHRTFLTDLVGVELQVMG ALI\_PLTG\_100 Tran...  
##   
## Slot "out\_fasta":  
## A AAStringSet instance of length 0  
step1\_sp4 <- signalp(input,  
 version = 4,  
 organism = 'euk',  
 run\_mode = 'starter',  
 truncate = TRUE  
 )  
## Version used... signalp4  
## running signalp locally...  
## 2 sequences to be truncated  
## Ok for single processing  
## Submitted sequences... 100  
## Candidate sequences with signal peptides... 5  
  
getSPtibble(step1\_sp4)  
## # A tibble: 5 x 9  
## gene\_id Cmax Cpos Ymax Ypos Smax Spos Smean Prediction  
## <fctr> <dbl> <int> <dbl> <int> <dbl> <int> <dbl> <chr>  
## 1 ALI\_PLTG\_5 0.712 25 0.678 25 0.846 10 0.652 Signal peptide  
## 2 ALI\_PLTG\_38 0.603 22 0.746 20 0.982 10 0.933 Signal peptide  
## 3 ALI\_PLTG\_23 0.362 22 0.533 22 0.910 12 0.783 Signal peptide  
## 4 ALI\_PLTG\_72 0.739 22 0.794 22 0.948 3 0.851 Signal peptide  
## 5 ALI\_PLTG\_83 0.179 21 0.338 21 0.843 17 0.640 Signal peptide  
  
getOutfasta(step1\_sp4)  
## A AAStringSet instance of length 5  
## width seq names   
## [1] 210 MKLSSVCTIIFGLVFIDFNNV...SDGSVTTAAQKAAALVKANA ALI\_PLTG\_5  
## [2] 495 MVVRVLLVVLALLAVGVQSKA...FAIVTIGASDGRVRYMNPPS ALI\_PLTG\_38  
## [3] 653 MKIVALVTFCIATLDSSIVFA...SEDPVYPLVKEYSDVVSKHP ALI\_PLTG\_23  
## [4] 118 MRLKFSILIFGAVLLATTTNA...NIRYASMLQDFLNTYHRRGV ALI\_PLTG\_72  
## [5] 164 MVSTSRVFALLLLPSPSARIF...SMVEHIKTTKRVIDEVQDHV ALI\_PLTG\_83  
  
getMatfasta(step1\_sp4)  
## A AAStringSet instance of length 5  
## width seq names   
## [1] 186 QCSDVHVVFARGSGEAAGLGI...SDGSVTTAAQKAAALVKANA ALI\_PLTG\_5  
## [2] 474 TRVRKSWTAYTSDEKEIYLSA...FAIVTIGASDGRVRYMNPPS ALI\_PLTG\_38  
## [3] 632 AECTVDELTEISTIYSEAMTD...SEDPVYPLVKEYSDVVSKHP ALI\_PLTG\_23  
## [4] 97 DDNSNKRLLRRQEKTDTTNGD...NIRYASMLQDFLNTYHRRGV ALI\_PLTG\_72  
## [5] 144 FQQVTMSATTATAITTKLKQF...SMVEHIKTTKRVIDEVQDHV ALI\_PLTG\_83

**Result**: 5 candidate proteins with signal peptides.

#### Step 2: check for presence of TM domains in mature peptides:

step2\_tm <- tmhmm(step1\_sp4,  
 TM = 0) # we allow 0 TM domains in mature peptides  
## running TMHMM locally...  
## Submitted sequences... 5  
## Candidates with signal peptides and 0 TM domains in mature seq 5  
  
getTMtibble(step2\_tm)  
## # A tibble: 5 x 6  
## gene\_id length ExpAA First60 PredHel Topology  
## <chr> <dbl> <dbl> <dbl> <dbl> <chr>  
## 1 ALI\_PLTG\_5 186 1.29 1.16 0 o  
## 2 ALI\_PLTG\_38 474 0.09 0.06 0 o  
## 3 ALI\_PLTG\_23 632 0.00 0.00 0 o  
## 4 ALI\_PLTG\_72 97 0.00 0.00 0 i  
## 5 ALI\_PLTG\_83 144 1.47 0.00 0 o  
  
getOutfasta(step2\_tm)  
## A AAStringSet instance of length 5  
## width seq names   
## [1] 210 MKLSSVCTIIFGLVFIDFNNV...SDGSVTTAAQKAAALVKANA ALI\_PLTG\_5  
## [2] 495 MVVRVLLVVLALLAVGVQSKA...FAIVTIGASDGRVRYMNPPS ALI\_PLTG\_38  
## [3] 653 MKIVALVTFCIATLDSSIVFA...SEDPVYPLVKEYSDVVSKHP ALI\_PLTG\_23  
## [4] 118 MRLKFSILIFGAVLLATTTNA...NIRYASMLQDFLNTYHRRGV ALI\_PLTG\_72  
## [5] 164 MVSTSRVFALLLLPSPSARIF...SMVEHIKTTKRVIDEVQDHV ALI\_PLTG\_83  
  
getOutMatfasta(step2\_tm)  
## A AAStringSet instance of length 5  
## width seq names   
## [1] 186 QCSDVHVVFARGSGEAAGLGI...SDGSVTTAAQKAAALVKANA ALI\_PLTG\_5  
## [2] 474 TRVRKSWTAYTSDEKEIYLSA...FAIVTIGASDGRVRYMNPPS ALI\_PLTG\_38  
## [3] 632 AECTVDELTEISTIYSEAMTD...SEDPVYPLVKEYSDVVSKHP ALI\_PLTG\_23  
## [4] 97 DDNSNKRLLRRQEKTDTTNGD...NIRYASMLQDFLNTYHRRGV ALI\_PLTG\_72  
## [5] 144 FQQVTMSATTATAITTKLKQF...SMVEHIKTTKRVIDEVQDHV ALI\_PLTG\_83

**Result**: 5 candidate proteins with signal peptides; 0 TM domains in mature sequences.

#### Step 3: predict sub-cellular localization

Here we are using targetp, could have also used wolfpsort.

step3\_tp <- targetp(step2\_tm,  
 network = 'N',  
 run\_mode = 'piper')  
## running targetp locally...  
## Ok for single processing  
## Number of submitted sequences... 5  
## Number of candidate secreted sequences 4  
  
getTPtibble(step3\_tp)  
## # A tibble: 4 x 7  
## gene\_id length mTP sp other TP\_localization RC  
## <fctr> <int> <dbl> <dbl> <dbl> <fctr> <int>  
## 1 ALI\_PLTG\_5 210 0.027 0.944 0.096 S 1  
## 2 ALI\_PLTG\_38 495 0.123 0.951 0.012 S 1  
## 3 ALI\_PLTG\_23 653 0.025 0.953 0.072 S 1  
## 4 ALI\_PLTG\_72 118 0.086 0.877 0.054 S 2  
  
getInfasta(step3\_tp)  
## A AAStringSet instance of length 5  
## width seq names   
## [1] 210 MKLSSVCTIIFGLVFIDFNNV...SDGSVTTAAQKAAALVKANA ALI\_PLTG\_5  
## [2] 495 MVVRVLLVVLALLAVGVQSKA...FAIVTIGASDGRVRYMNPPS ALI\_PLTG\_38  
## [3] 653 MKIVALVTFCIATLDSSIVFA...SEDPVYPLVKEYSDVVSKHP ALI\_PLTG\_23  
## [4] 118 MRLKFSILIFGAVLLATTTNA...NIRYASMLQDFLNTYHRRGV ALI\_PLTG\_72  
## [5] 164 MVSTSRVFALLLLPSPSARIF...SMVEHIKTTKRVIDEVQDHV ALI\_PLTG\_83  
  
getOutfasta(step3\_tp)  
## A AAStringSet instance of length 4  
## width seq names   
## [1] 210 MKLSSVCTIIFGLVFIDFNNV...SDGSVTTAAQKAAALVKANA ALI\_PLTG\_5  
## [2] 495 MVVRVLLVVLALLAVGVQSKA...FAIVTIGASDGRVRYMNPPS ALI\_PLTG\_38  
## [3] 653 MKIVALVTFCIATLDSSIVFA...SEDPVYPLVKEYSDVVSKHP ALI\_PLTG\_23  
## [4] 118 MRLKFSILIFGAVLLATTTNA...NIRYASMLQDFLNTYHRRGV ALI\_PLTG\_72

This step allowed us to filter 1 sequence potentially targeted to mitochondria.

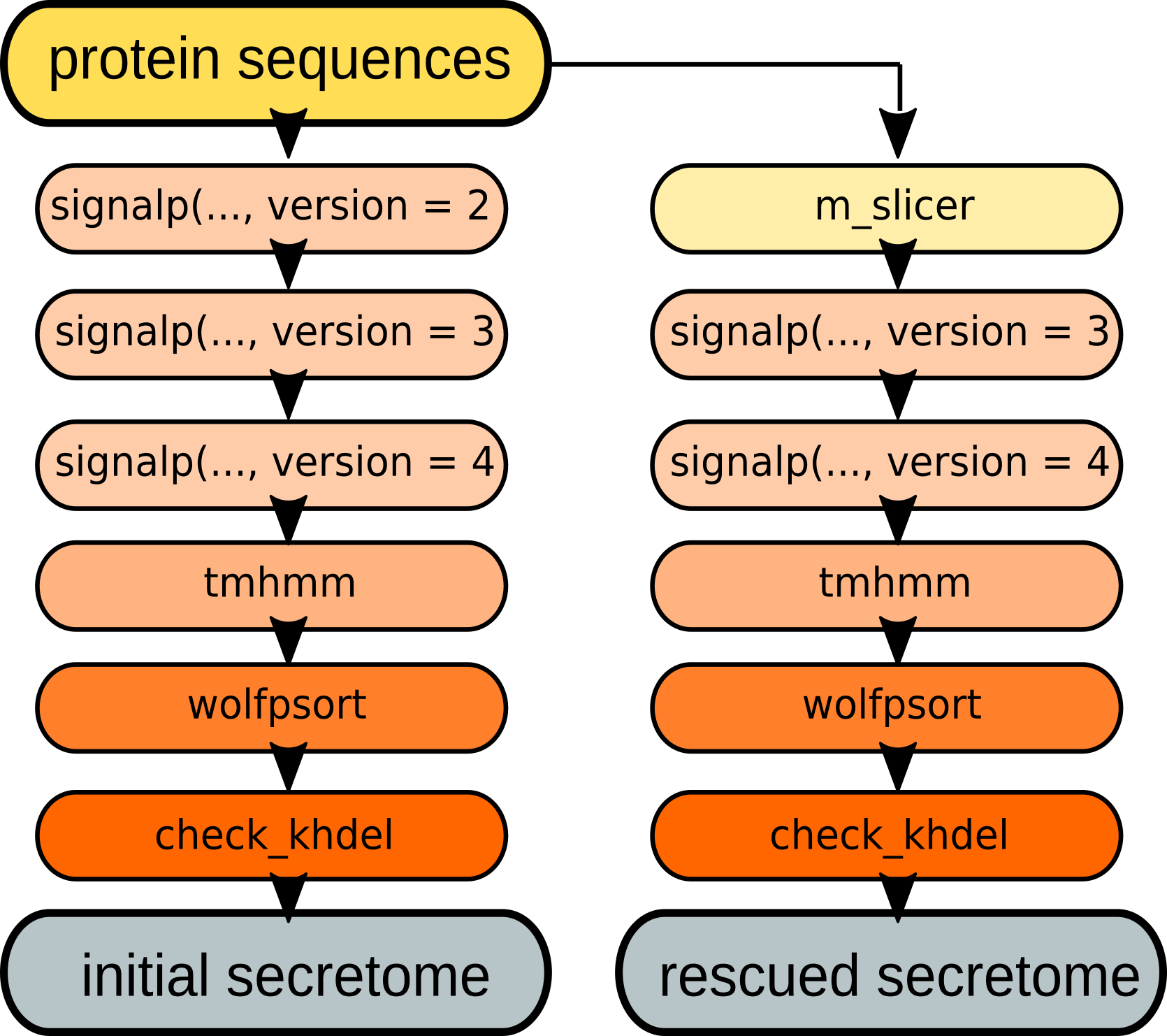
**Result**: 4 candidate proteins with signal peptides; 0 TM domains in mature sequences; not targeted to plastids or mitochondria.

#### Step 4: check for ER-retention signals

step4\_er <- check\_khdel(step3\_tp,  
 run\_mode = 'piper')  
## checking for terminal ER retention signals...  
## Submitted sequences... 4  
## Sequences with terminal ER retention signals detected... 0  
## Candidate without terminal ER retention signals detected... 4  
  
getOutfasta(step4\_er)  
## A AAStringSet instance of length 4  
## width seq names   
## [1] 210 MKLSSVCTIIFGLVFIDFNNV...SDGSVTTAAQKAAALVKANA ALI\_PLTG\_5  
## [2] 495 MVVRVLLVVLALLAVGVQSKA...FAIVTIGASDGRVRYMNPPS ALI\_PLTG\_38  
## [3] 653 MKIVALVTFCIATLDSSIVFA...SEDPVYPLVKEYSDVVSKHP ALI\_PLTG\_23  
## [4] 118 MRLKFSILIFGAVLLATTTNA...NIRYASMLQDFLNTYHRRGV ALI\_PLTG\_72

**Final Result**: 4 candidate proteins with signal peptides; 0 TM domains in mature sequences; not targeted to plastids or mitochondria; not retained in ER.

This is an example of a fairly simple pipeline, but you can create more complex ones. The stringent pipeline might involve multi-step filtration on signal peptide prediction as well as additional predictions for 'rescued' (**Figure 4**) sequences.



**Figure 3.** Example of a stringent pipeline.

### References

Dyrl�v Bendtsen, Gunnar von Heijne annick, Henrik Nielsen, and S�ren Brunak. 2004. “Improved Prediction of Signal Peptides: SignalP 3.0.” *J. Mol. Biol.*, no. 340: 783–95.

Henrik Nielsen, S�ren Brunak, Jacob Engelbrecht, and Gunnar von Heijne. 1997. “Identification of Prokaryotic and Eukaryotic Signal Peptides and Prediction of Their Cleavage Sites.” *Protein Engineering* 10: 1–6.

Krogh, Anders, Bjorn Larsson Gunnar von Heijne, and Erik L.L. Sonnhammer. 2001. “Predicting Transmembrane Protein Topology with a Hidden Markov Model: Application to Complete Genomes.” *J. Mol. Biol.*, no. 305: 567–80.

Munro S, Pelham HBR. 1987. “A c-Terminal Signal Prevents Secretion of Luminal Er Proteins.” *Cell*, no. 48: 899–907.

Nielsen, Henrik, and Anders Krogh. 1998. “Prediction of Signal Peptides and Signal Anchors by a Hidden Markov Model.” *Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology (ISMB 6)*, 122–30.

Olof Emanuelsson, Soren Brunak, Henrik Nielsen, and Gunnar von Heijne. 2008. “Predicting Subcellular Localization of Proteins Based on Their N-Terminal Amino Acid Sequence.” *J. Mol. Biol.*, no. 300: 1005–16.

Paul Horton, Takeshi Obayashi, Keun-Joon Park, and Kenta Nakai. 2006. “Protein Subcellular Localization Prediction with Wolf Psort.” *Asian Pacific Bioinformatics Conference, APCB2006*.

Sperschneider J, Hane JK, Williams AH. 2015. “Evaluation of Secretion Prediction Highlights Differing Approaches Needed for Oomycete and Fungal Effectors.” *Front Plant Sci*, no. 6: 1168.

Thomas Nordahl Petersen, Gunnar von Heijne, Soren Brunak, and Henrik Nielsen. 2011. “SignalP 4.0: Discriminating Signal Peptides from Transmembrane Regions.” *Nature Methods* 8 (10): 785–86.