uORFseqr

v1.0 release date: 08.21.18

Overview

uORF-seqr is a novel machine learning approach to uORF identification that uses ribosome profiling and RNA-seq data to quantify expression features of known uORFs across three replicates. These weighted features are then used to predict novel uORFs that have similar expression profiles. These are compared to features generated by several null models that randomly permute the data. High confidence predictions are output as bed files.

Spealman, P. and Naik, A. et al. Conserved non-AUG uORFs revealed by a novel regression analysis of ribosome profiling data. Genome Research 28, 214–222 (2018). doi: 10.1101/gr.221507.117

Data requirements:

uORFseqr requires triplicate ribosome profiling and RNA-seq data that has previously been aligned using STAR aligner (Alex Dobin, et. al. Bioinformatics, 2013).

Installation requirements:

Install python (2.7.x)

Install R (3.2.3+)

Pip install list:

```
pip install rpy2==2.8.6

pip install pysam

pip install -U statsmodels

pip install -U scikit-learn

pip install gffutils

pip install matplotlib

pip install scipy
```

Begin using virtualenv

If you have virtualenv installed it can be used to load a virtual environment with the python packages preinstalled. Note: virtualenv portability has known problems and has produced issues in some server builds.

From the uorfseqr directory:

```
source uorf_env/bin/activate
```

Test installation using uorfseqr –test command

You can verify the installation of uorfseqr using the following command from the uORF folder:

```
python uorfseqr.py -test
```

Demonstration of basic uorfseqr commands uorfseqr –demo command

A simple case study of usage, format, and examples is available using the demo command:

```
python uorfseqr.py -demo
```

Step by step uORF-seqr analysis.

Task 1: Set up a new species

Note: this needs to be done only once per species

```
#Usage:
python uorfseqr.py -preprocess -genome_name <str> -fa <path to
reference fasta file>


#Example: python uorfseqr.py -pre -genome_name s_cerervisiae -fa
data/reference_genomes/Scer_SacCer3.fa
```

This command will generate a new directory as defined in 'genome_name', as well as the creation of a command file 'preprocess.cmd'. This will also generate a cache of the per-chromosome sequence in a binary format.

Task 2: Quantify features for a new experiment

```
#Usage:
python uorfseqr.py -quantify -genome_name <str> -samples <sample
_name> <path to sample RPF.bam> <path to sample mRNA.mRNA> -o
<output_dir>

#Example:
python uorfseqr.py -quantify -genome_name s_cerervisiae -samples
Scer_A ../data/bam/Scer_A_RPF.bam ../data/bam/Scer_A_mRNA.bam Scer_B
../data/bam/Scer_B_RPF.bam ../data/bam/Scer_B_mRNA.bam Scer_C
../data/bam/Scer_C_RPF.bam ../data/bam/Scer_C_mRNA.bam -o scer.demo
```

The –quantify command scores the features of known uORFs across three replicates as well as null models. It relies on the cached processed genome from the –preprocess task. It will create the output directory, several subdirectories, and a command file called 'analysis.json'.

Numerous optional commands are available for the end user to alter the performance of the analysis.

GFF files

Users can set their own gff file to define 5'UTRs (aka. transcript leaders), 3'UTRs, Transcription start sites, poly-a sites, and main orf coordinates. Otherwise the standard gff for S.cerevisiae (from Spealman and Naik, Genome Research, 2017) is loaded. (located in 'data/reference_genomes/saccharomyces_cerevisiae.gff'). **Note:** The current gff file format contains region and field names that are not present in the NCBI and ENSMBL gff files for *S. cerevisiae*.

```
–gff_file data/reference_genomes/saccharomyces_cerevisiae.gff
```

Filter uORFs

Users can set their own genes or regions to be filtered by loading a bed file # otherwise the scer_baduorf file can be loaded. (this is located 'data/labelled_uorfs/scer_baduorf.bed'). **Note** if it is desired that no genes or regions be filtered a blank file should be loaded instead.

```
– filter_uorfs data/labelled_uorfs/scer_baduorf.bed'
```

Known uORFs

Users list of labelled positive uORFs by loading a bed file containing the uORF from start codon to stop codon and strand. Otherwise the STANDARD-golden bed file is loaded. This file contains 17 molecularly validated uORFs identified in *S. cerevisiae* before 2017. (this is located at 'data/labelled_uorfs/STANDARD-golden.bed') **Note:** A longer list of containing the 432 predicted S.cerevisae uORFs from Spealman and Naik (Genome Research, 2017) is also available 'data/labelled_uorfs/Spealman_Naik_2017.bed'

```
-known data/labelled_uorfs/STANDARD-golden.bed
```

Ribosome footprint sizes

Users can set minimum and maximum sizes. By default these are 27 and 33nt in length.

```
-max_rpf 33
-min_rpf 22
```

Non-start near-cognate codons (NCCs)

Near-cognate codons can act as translation initiation sites. NCCs differ from the canonical start codon 'AUG' by one nucleotide. All NCCs have been observed functioning as start codons *in vivo* and *vitro* experiments, with the exception of 'AGG' and 'AAG'. By default all NCCs are considered as potential start codons except 'AGG' and 'AAG'.

```
-non_start AGG AAG
```

Kozak consensus sequence

The sequence around potential start codons are scored based on similarity to the provided Kozak consensus sequence. By default this is "AAAAAAATGT" where the underlined sequence is the start codon.

```
-kozak_seq "AAAAAAATGT"
```

Main ORF mask

The start codons of some main ORFs exhibit ribosome pileups as ribosomes 'queue' in preparation of initiation at the main ORF start codon. As these queues can contain many more ribosomes than are present within the rest of the transcript leader they can present problems for scoring and identification. The main ORF mask filters a given number of nucleotides upstream of the main ORF start codon. By default this is set to 15 nt.

```
-morf_mask 15
```

Minimum 5'UTR length

This allows for the filtering of any gene with an annotated 5'UTR shorter than the given number of nucleotides. By default this is set to 15nt.

```
-min_utr 15
```

Minimum supporting RPF

This defines the minimum number of RPFs that a potential uORF requires in order to be considered as valid. By default this is set to 3. **Note:** This does not mean that the potential uORF is a predicted candidate uORF. Potential uORFs must still outperform the expectation at random of features observed in the random permutation null models.

```
-min_rpf_ct 3
```

Task 3: Predict candidate uORFs based on calculated features

```
#Usage:
python uorfseqr.py -predict -i <previous output_dir>
```

```
#Example:
python uorfseqr.py -predict -i scer.demo
```

With the –predict command we load the calculated features from the –quantify command and begin the construction of the regression and scoring of the potential uORFs. Those potential uORFs, as well as their respective qval scores are reported in bed file format in the 'results' directory. **Note: While each file contains information, the *-candidate_uORF.bed file is the strictest and highest confidence of the predictions, this is copied into the –input_directory for your convenience.** This file is a copy of the *-significant-resolved-qvals.bed file in the results directory.

Understanding the predicted candidate uORF ouput.

The ***-candidate_uORF.bed** file generated by uorfseqr is a standard six-column bed format that contains lines such as:

```
chrV    140112  140124  YEL009C.140124  0       -
```

By tab delimited column these are:

Chromosome

Feature start nucleotide

Feature stop nucleotide

uORF_id

Note: uORF id is the name of the gene that the uORF is associated with (ie. in the transcript leader of) and the nucleotide of the uORF start codon.

Qval score

Note: With Qval score the lower the better. Actual numerical value is (Qval score *0.001).

Feature strand sign