Normalisr

Lingfei Wang

Jul 13, 2020

CONTENTS:

1 Normalisr **1**
- 1.1 Installation . . . 1
- 1.2 Usage . . . 1
- 1.3 Documentation . . . 2
- 1.4 Examples and pipelines . . . 2
- 1.5 Contact . . . 2
- 1.6 References . . . 2
- 1.7 FAQ . . . 3

2 User API **5**
- 2.1 Quality control . . . 5
- 2.2 Normalization . . . 5
- 2.3 Differential expression . . . 5
- 2.4 Co-expression . . . 5
- 2.5 API list . . . 6

3 All API **13**

4 Indices and tables **15**

Python Module Index **17**

Index **19**

CHAPTER

ONE

NORMALISR

Normalisr is a parameter-free normalization-association two-step inferential framework for scRNA-seq that solves case-control differential expression, co-expression, and pooled CRISPRi scRNA-seq screen under one umbrella of linear association testing. Normalisr addresses sparsity and technical confounding challenges of scRNA-seq with posterior mRNA abundances, nonlinear cellular summary covariates, and mean and variance normalization. All these enable linear association testing to achieve optimal sensitivity, specificity, and speed in all above scenarios.

Normalisr follows the conventional framework of normalization/imputation of scRNA-seq, and aims to recover the true, biological, but hidden expression levels which any analyses may then operate upon. Then, linear association testing provides a unified inferential framework with numerous advantages: (i) exact P-value estimation without permutation, (ii) native removal of covariates as fixed effects, (iii) non-parametric robustness, (iv) unbeatable time and memory complexities, and (v) extension potentials such as variations in genetic relatedness.

Normalisr is written in Python3 and provides a command-line and a python functional interface. You can read more about Normalisr from our preprint (See *References*).

1.1 Installation

Normalisr is on PyPI and can be installed with pip: `pip install normalisr`. You can also install Normalisr from github: `pip install git+https://github.com/lingfeiwang/normalisr.git`. Make sure you have added Normalisr's install path into PATH environment before using the command-line interface (See *FAQ*). Normalisr's installation should take less than a minute.

There are more advanced installation methods but if you want that, most likely you already know how to do it ;). If not, give me a shout (See *Contact*).

1.2 Usage

Normalisr provides a command-line and a python functional interface below. You can use the examples provided below to guide yourself through Normalisr's use. Sphinx-based documentation is underway.

- **Commmand-line interface** You can run Normalisr by typing `normalisr` on command-line. Normalisr uses submodules for different analysis steps. Type `normalisr` or `normalisr -h` for general help, and for example `normalisr de -h` for help on submodule 'de' of differential expression.

  Normalisr uses tsv (tab separated values) file format for input and output matrices, and text file for row and column names, such as cells and genes, one per line. For initial input, Normalisr also accepts the sparse

mtx format (Cell Ranger output) for raw read count matrix. Gzipped input/output files are automatically recognized if file name suffix '.gz' is present.

- **Python functional interface** Normalisr's python functional interface is more flexible than command-line, but requires knowledge of python programming. Documentation of any function can be obtained with `?` in ipython or jupyter notebook, such as:

```
import normalisr.normalisr as norm
?norm.de
```

The example jupyter notebooks also illustrate the scope of functions Normalisr provides.

1.3 Documentation

Documentations are available as html and pdf.

1.4 Examples and pipelines

You can find several examples in the 'examples' folder, to cover all functions Normalisr currently provides. The example datasets have been scaled down to run on a 16GB-memory personal computer. Although they only serve as demonstrations of work here, the pipelines should be transferable to a full-scale, different dataset. Since Normalisr is non-parametric, the only adjustable parameters are for quality control and final cutoffs of differential or co-expression. You can change down-sampling parameters in the examples to run the full datasets on a larger computer.

You can find more details in the respective examples.

1.5 Contact

We look forward to your feedbacks or questions of any kind.

- Regarding method and manuscript, please reach me by email (Lingfei.Wang.CN@gmail.com).
- Regarding Normalisr package, please raise an issue on github or reach me by email (Lingfei.Wang.github@outlook.com).

1.6 References

Please find the currently available materials below:

- Abstract submitted to ICML 2020 Workshop on Computational Biology

Contact me if you would like to request more details.

1.7 FAQ

- **What does Normalisr stand for?** **N**ormalisr **O**ffers **R**obust **M**odelling of **A**ssociations **L**inearly **I**n **S**ingle-cell **R**NA-seq. Yes, it's a recursive acronym. See GNU and pip.
- **I installed Normalisr but typing `normalisr` says 'command not found'.**
- **How do I use a specific python version for Normalisr's command-line interface?** You can always use the python command to run Normalisr, such as `python3 -m normalisr` to replace command `normalisr`. You can also use a specific path or version for python, such as `python3.7 -m normalisr` or `/usr/bin/python3.7 -m normalisr`. Make sure you have installed Normalisr for this python version.
- **Why don't the examples work?** Please make sure you followed every step in the README.md of the respective example folder with Internet connection, and then submit an issue report detailing at which executed line the error occurred with input and output.
- **Does Normalisr run on Windows?** I have not tested Normalisr on Windows. However, it is purely in python and should be able to function properly.

CHAPTER

TWO

USER API

2.1 Quality control

qc_reads	Quality control by bounding UMI read counts.
qc_outlier	Quality control by removing cell outliers by variance.

2.2 Normalization

lcpm	Computes Bayesian log CPM from raw UMI read counts.
normcov	Normalizes each continuous covariate to 0 mean and unit variance.
scaling_factor	Computes scaling factor of variance normalization for every gene.
compute_var	Computes variance normalization scale for each cell.
normvar	Performs mean and variance normalizations.

2.3 Differential expression

de	Performs differential expression analyses for all genes against all groupings.

2.4 Co-expression

coex	Performs co-expression analyses for all gene pairs.
binnet	Binarizes P-value co-expresion network to thresholded Q-value network.
gotop	Finds the top variable GO enrichment of top principal genes in the binary co-expression network.
pccovt	Introduces an extra covariate from the top principal component of given genes.

2.5 API list

normalisr.normalisr.**binnet**(*net*, *qcut*)

Binarizes P-value co-expresion network to thresholded Q-value network.

Q-values are computed separately per row to account for differences in the number of genes co-expressed, especially by master regulators, using Benjamini–Hochberg procedure. Co-expression Q-value matrix is thresholded for return.

Parameters

- **net** (*numpy.ndarray(shape=(n_gene,n_gene),dtype=float)*) – Symmetric co-expression P-value matrix.
- **qcut** (*float*) – Cutoff for Q-value network.

Returns Binarized, assymmetric co-expression matrix.

Return type numpy.ndarray(shape=(n_gene,n_gene),dtype=bool)

normalisr.normalisr.**coex**(*dt*, *dc*, *bs=0*, *nth=1*, ***ka*)

Performs co-expression analyses for all gene pairs.

Performs parallel computation with multiple processes on the same machine.

Model for co-expression between genes i & j: X_i=gamma*X_j+alpha*C+epsilon,

epsilon~i.i.d. N(0,sigma**2).

Test statistic: conditional R**2 (or proportion of variance explained) between X_i and X_j.

Null hypothesis: gamma=0.

Parameters

- **dt** (*numpy.ndarray(shape=(n_gene,n_cell),dtype=float)*) – Normalized expression matrix X.
- **dc** (*numpy.ndarray(shape=(n_cov,n_cell),dtype=float)*) – Normalized covariate matrix C.
- **bs** (*int*) – Batch size, i.e. number of genes in each computing batch. Use 0 for default: Data transfer limited to 1GB, capped at bs=500.
- **nth** (*int*) – Number of parallel processes. Set to 0 for using automatically detected CPU counts.
- **ka** (*dict*) – Keyword arguments passed to normalisr.association.association_test_1. See below.

Returns

- **P-values** (*numpy.ndarray(shape=(n_gene,n_gene))*) – Co-expression P-value matrix.
- **dot** (*numpy.ndarray(shape=(n_gene,n_gene))*) – Inner product of expression between gene pairs, after removing covariates.
- **var** (*numpy.ndarray(shape=(n_gene))*) – Variance of gene expression after removing covariates. Pearson R=(((dot/numpy.sqrt(var)).T)/numpy.sqrt(var)).T.

Keyword Arguments **dimreduce** (*numpy.ndarray(shape=(n_gene,),dtype=int) or int*) – If dt doesn't have full rank, such as due to prior covariate removal (although the recommended method is to leave covariates in dc), this parameter allows to specify the loss of ranks/degrees of freedom to allow for accurate P-value computation. Default is 0, indicating no rank loss.

normalisr.normalisr.**compute_var**(*dt*, *dc*, *stepmax=1*, *eps=1e-06*)

Computes variance normalization scale for each cell.

Performs a log-linear fit of the variance of each cell with covariates. Optionally use EM-like method to iteratively fit mean and variance. For EM-like method, early-stopping is suggested because of overfitting issues.

Parameters

- **dt** (*numpy.ndarray(shape=(n_gene,n_cell))*) – Bayesian LogCPM expression level matrix.
- **dc** (*numpy.ndarray(shape=(n_cov,n_cell))*) – Covariate matrix.
- **stepmax** (*int*) – Maximum number of EM-like iterations of mean and variance normalization. Stop of iteration is also possible when relative accuracy target is reached. Defaults to 1, indicating no iterative normalization.
- **eps** (*float*) – Relative accuracy target for early stopping. Constrains the maximum relative difference of fitted variance across cells compared to the last step. Defaults to 1E-6.

Returns Inverse sqrt of fitted variance for each cell, i.e. the multiplier for variance normalization. For iterative normalization, the optimal step will be returned, defined as having the minimal max relative change across cells.

Return type numpy.ndarray(shape=(n_cell,))

normalisr.normalisr.**de**(*dg*, *dt*, *dc*, *bs=0*, *nth=1*, *single=0*, ***ka*)

Performs differential expression analyses for all genes against all groupings.

Allows multiple options to treat other groupings when testing on one grouping.

Performs parallel computation with multiple processes on the same machine.

Model for differential expression between gene Y and grouping X: Y=gamma*X+alpha*C+epsilon,

epsilon~i.i.d. N(0,sigma**2).

Test statistic: conditional R**2 (or proportion of variance explained) between Y and X.

Null hypothesis: gamma=0.

Parameters

- **dg** (*numpy.ndarray(shape=(n_group,n_cell))*) – Grouping matrix for a list of X to be tested, e.g. grouping by gene knock-out.
- **dt** (*numpy.ndarray(shape=(n_gene,n_cell),dtype=float)*) – Normalized expression matrix Y.
- **dc** (*numpy.ndarray(shape=(n_cov,n_cell),dtype=float)*) – Normalized covariate matrix C.
- **bs** (*int*) – Batch size, i.e. number of groupings and genes in each computing batch. For single=0,1, splits groupings & genes. Defaults to 500. For single=4, only allows bs=0 (default automatic setting).
- **nth** (*int*) – Number of parallel processes. Set to 0 for using automatically detected CPU counts.
- **single** (*int*) – Option to deal with other groupings when testing one groupings v.s. gene expression.
  - 0: Ignores other groupings.
  - 1: Excludes all cells belonging to any other grouping (value==1), assuming dg=0,1 only.

  - 4: Treats other groupings as covariates for mean expression.

- **ka** (*dict*) – Keyword arguments passed to normalisr.association.association_test_*. See below.

Returns

- **P-values** (*numpy.ndarray(shape=(n_group,n_gene))*) – Differential expression P-value matrix.
- **gamma** (*numpy.ndarray(shape=(n_group,n_gene))*) – Differential expression log fold change matrix.
- **alpha** (*numpy.ndarray(shape=(n_group,n_gene,n_cov))*) – Maximum likelihood estimators of alpha, separatly tested for each grouping.
- **varg** (*numpy.ndarray(shape=(n_group))*) – Variance of grouping after removing covariates.
- **vart** (*numpy.ndarray(shape=(n_group,n_gene))*) – Variance of gene expression after removing covariates. It can depend on the grouping being tested depending on parameter single.

Keyword Arguments

- **dimreduce** (*numpy.ndarray(shape=(n_gene,),dtype=int) or int*) – If dt doesn't have full rank, such as due to prior covariate removal (although the recommended method is to leave covariates in dc), this parameter allows to specify the loss of ranks/degrees of freedom to allow for accurate P-value computation. Default is 0, indicating no rank loss.
- **method** (*str, only for single=4*) – Method to compute eigenvalues in SVD-based matrix inverse (for removal of covariates):
  - auto: Uses scipy for n_matrix<mpc or mpc==0 and sklearn otherwise. Default.
  - scipy: Uses scipy.linalg.svd.
  - scipys: NOT IMPLEMENTED.Uses scipy.sparse.linalg.svds.
  - sklearn: Uses sklearn.decomposition.TruncatedSVD.
- **mpc** (*int, only for single=4*) – Uses only the top mpc singular values as non-zero in SVD-based matrix inverse. Here effectively reduces covariates to their top principal components. This reduction is performed after including other groupings as additional covariates. Defaults to 0 to disable dimension reduction. For very large grouping matrix, use a small value (e.g. 100) to save time at the cost of accuracy.
- **qr** (*int, only for single=4*) – Whether to use QR decomposition method for SVD in matrix inverse. Only effective when method=sklearn, or =auto and defaults to sklearn. Takes the following values:
  - 0: No (default).
  - 1: Yes with default settings.
  - 2+: Yes with n_iter=qr for sklearn.utils.extmath.randomized_svd.
- **tol** (*float, only for single=4*) – Eigenvalues < tol*(maximum eigenvalue) are treated as zero in SVD-based matrix inverse. Default is 1E-8.

normalisr.normalisr.**gotop**(*net*, *namet*, *go_file*, *goa_file*, *n=100*, ***ka*)

Finds the top variable GO enrichment of top principal genes in the binary co-expression network.

Principal genes are those with most co-expressed genes. They reflect the most variable pathways in the dataset. When the variable pathways are housekeeping related, they may conceal cell-type-specific co-expression pat-

terns from being observed and understood. This function identifies the most variable pathway with gene ontology enrichment study of the top principal genes. Background genes are all genes provided.

Parameters

- **net** (*numpy.ndarray(shape=(n_gene,n_gene),dtype=bool)*) – Binary co-expression network matrix.
- **namet** (*list of str*) – Gene names matching the rows and columns of net.
- **go_file** (*str*) – Path of GO DAG file
- **goa_file** (*str*) – Path of GO annotation file
- **n** (*int*) – Number of top principal genes to include for GO enrichment. Default is 100, giving good performance in general.
- **ka** (*dict*) – **IMPORTANT**: Keyword arguments passed to normalisr.gocovt.goe to determine how to perform GO enrichment study. If you see no gene mapped, check your gene name conversion rule in conversion parameter of goe. GO annotation have a specific gene ID system.

Returns

- **principals** (*list of str*) – List of principal genes.
- **goe** (*pandas.DataFrame*) – GO enrichment results.
- **gotop** (*str*) – Top enriched GO ID.
- **genes** (*list of str*) – List of genes in the gotop GO ID.

normalisr.normalisr.**lcpm**(*reads*, *seed=None*, *nth=0*, *ntot=None*, *varscale=0*, *normalize=True*)

Computes Bayesian log CPM from raw UMI read counts.

The technical sampling process is modelled as a Binomial distribution. The logCPM given UMI read counts is a Bayesian inference problem and follows (shifted) Beta distribution. We use the expectation of posterior logCPM as the estimated expression levels. Resampling function is also provided to account for variances in the posterior distribution.

Parameters

- **reads** (*numpy.ndarray(shape=(n_gene,n_cell),dtype='uint')*) – UMI read count matrix.
- **seed** (*int*) – Initial random seed if set.
- **ntot** (*int*) – Manually sets value of total number of UMIs in binomial distribution. Since the posterior distribution stablizes quickly as ntot increases, a large number, e.g. 1E9 is good for general use. Defaults to None to disable manual value.
- **varscale** (*float*) – Resamples estimated expression using the posterior Beta distribution. varscale sets the scale of variance than its actual value from the posterior distribution. Defaults to 0, to compute expectation with no variance.
- **normalize** (*bool*) – Whether to normalize output to logCPM per cell. Default: True.
- **nth** (*int*) – Number of threads to use. Defaults to 0 to use all cores automatically detected.

Returns

- **lcpm** (*numpy.ndarray(shape=(n_gene,n_cell))*) – Estimated expression as logCPM from UMI read counts.
- **mean** (*numpy.ndarray(shape=(n_gene,n_cell))*) – Mean/Expectation of lcpm's every entry's posterior distribution.

- **var** (*numpy.ndarray(shape=(n_gene,n_cell))*) – Variance of lcpm's every entry's posterior distribution.
- **cov** (*numpy.ndarray(shape=(3,n_cell))*) – Cellular summary covariates computed from UMI read count matrix that may confound lcpm. Contains:
  - cov[0]: Log total read count per cell
  - cov[1]: Number of 0-read genes per cell
  - cov[2]: cov[0]**2

normalisr.normalisr.**normcov**(*dc*, *c=True*)

Normalizes each continuous covariate to 0 mean and unit variance.

Optionally introduces constant 1 covariate as intercept. Categorical covariates should be in binary/one-hot form, and will be left unchanged.

Parameters

- **dc** (*numpy.ndarray(shape=(n_cov,n_cell))*) – Current covariate matrix. Use empty matrix with n_cov=0 if no covariate.
- **c** (*bool*) – Whether to introduce a constant 1 covariate.

Returns Processed covariate matrix.

Return type numpy.ndarray(shape=(n_cov+1 if c else n_cov,n_cell))

normalisr.normalisr.**normvar**(*dt*, *dc*, *w*, *wt*, *dextra=None*, *cat=1*, *keepvar=True*)

Performs mean and variance normalizations.

Expression levels are normalized at mean and then at variance levels. Effectively each gene x is multiplied by w**wt[x] before removing covariates as dc*(w**wt[x]). Continuous covariates are normalized at variance levels. Effectively covariates are transformed to dc*w. Therefore, variance normalization for expression are scaled differently for each gene.

Parameters

- **dt** (*numpy.ndarray(shape=(n_gene,n_cell))*) – Bayesian logCPM matrix.
- **dc** (*numpy.ndarray(shape=(n_cov,n_cell))*) – Covariate matrix.
- **w** (*numpy.ndarray(shape=(n_cell,))*) – Computed variance normalization multiplier.
- **wt** (*numpy.ndarray(shape=(n_gene,))*) – Computed scaling factor for each gene.
- **dextra** (*numpy.ndarray(shape=(n_extra,n_cell))*) – Extra data matrix also to be normalized like continuous covariates.
- **cat** (*int*) – Whether to normalize categorical/binary covariates (those with only 0 or 1s). Defaults to 1.
  - 0: No
  - 1: No except constant-1 covariate (intercept)
  - 2: Yes
- **keepvar** (*bool*) – Whether to maintain the variance of each gene invariant in mean normalization step. If so, expression variances are scaled back to original after mean normalization and before variance normalization. This function only affects overall variance level and its downstreams (e.g. differential expression log fold change). This function would not affect P-value computation. Default: True.

Returns

- *(dtn,dcn) or (dtn,dcn,dextran) if dextra is not None*
- **dtn** (*numpy.ndarray(shape=(n_gene,n_cell))*) – Normalized gene expression matrix.
- **dcn** (*numpy.ndarray(shape=(n_cov,n_cell))*) – Normalized covariate matrix.
- **dextran** (*numpy.ndarray(shape=(n_extra,n_cell))*) – Normalized extra data matrix.

normalisr.normalisr.**pccovt**(*dt*, *dc*, *namet*, *genes*, *condcov=True*)

Introduces an extra covariate from the top principal component of given genes.

The extra covariate is the top principal component of normalized expressions of the selected genes. Adding a covariate from housekeeping pathway can reveal cell-type-specific activities in co-expression networks.

Parameters

- **dt** (*numpy.ndarray(shape=(n_gene,n_cell))*) – Normalized expression matrix.
- **dc** (*numpy.ndarray(shape=(n_cov,n_cell))*) – Existing normalized covariate matrix.
- **namet** (*list of str*) – List of gene names for rows in dt.
- **genes** (*list of str*) – List of gene names to include in finding top PC of their expression as an extra covariate.
- **condcov** (*bool*) – Whether to condition on existing covariates before computing top PC. Default: True.

Returns New normalized covariate matrix.

Return type numpy.ndarray(shape=(n_cov+1,n_cell))

normalisr.normalisr.**qc_outlier**(*dw*, *pcut=1e-10*, *outrate=0.02*)

Quality control by removing cell outliers by variance.

Fit normal distribution on the inverse sqrt variance to detect outliers. This is performed by iterative estimation of normal distribution with non-outliers and then determination of outliers with the normal distribution.

Parameters

- **dw** (*numpy.ndarray(shape=(n_cell,))*) – Fitted inverse sqrt cell variance.
- **pcut** (*float*) – Bonferroni P-value cutoff for asserting outliers in a normal distribution of fitted cell variance. Default: 1E-10.
- **outrate** (*float*) – Upper bound of proportion of outliers on either side of variance distribution. Used for initial outlier assignment and final validity check. Default: 0.02.

Returns Whether each cell passed QC

Return type numpy.ndarray(shape=(n_cell,),dtype=bool)

normalisr.normalisr.**qc_reads**(*reads*, *n_gene*, *nc_gene*, *ncp_gene*, *n_cell*, *nt_cell*, *ntp_cell*)

Quality control by bounding UMI read counts.

Quality control is perform separately on genes based on their cell statisics and on cells based on their gene statistics, iteratively until dataset remains unchanged. A gene or cell is removed if any of the QC criteria is violated at any time in the iteration. All QC parameters can be set to 0 to disable QC filtering for that criterion.

Parameters

- **reads** (*numpy.ndarray((n_gene,n_cell),dtype='uint')*) – UMI read count matrix.

- **n_gene** (*int*) – Lower bound on total read counts for gene QC.
- **nc_gene** (*int*) – Lower bound on number of expressed cells for gene QC.
- **ncp_gene** (*float*) – Lower bound on proportion of expressed cells for gene QC.
- **n_cell** (*int*) – Lower bound on total read counts for cell QC.
- **nt_cell** (*int*) – Lower bound on number of expressed genes for cell QC.
- **ntp_cell** (*float*) – Lower bound on proportion of expressed genes for cell QC.

Returns

- **genes_select** (*numpy.ndarray(dtype='uint')*) – Array of indices of genes passed QC.
- **cells_select** (*numpy.ndarray(dtype='uint')*) – Array of indices of cells passed QC.

normalisr.normalisr.**scaling_factor**(*dt*, *varname='nt0mean'*, *v0=0*, *v1='max'*)

Computes scaling factor of variance normalization for every gene.

Lowly expressed genes need full variance normalization because of technical confounding from sequencing depth. Highly expressed genes do not need variance normalization because they are already accurately measured. The scaling factor operates as a exponential factor on the variance normalization scale for each gene. It should be maximum/minimum for genes with lowest/highest expression.

Parameters

- **dt** (*numpy.ndarray(shape=(n_gene,n_cell),dtype='uint')*) – UMI read count matrix.
- **varname** (*str*) – Variable used to compute scaling factor for each gene. Can be:
  - logtpropmean: log(dt.mean(axis=1)/dt.mean(axis=1).sum())
  - logtmeanprop: log((dt/dt.sum(axis=0)).mean(axis=1))
  - nt0mean: (dt==0).mean(axis=1)
  - lognt0mean: log((dt==0).mean(axis=1))
  - log1-nt0mean: log(1-(dt==0).mean(axis=1))

  Defaults to nt0mean.
- **v0,v1** (*float*) – Variable values to set scaling factor to 0 (for v0) and 1 (for v1). Linear assignment is applied for values inbetween. Can be:
  - max: max
  - min: min
  - any float: that float

Returns Scaling factor of variance normalization for each gene

Return type numpy.ndarray(shape=(n_gene,))

CHAPTER

THREE

ALL API

TBA

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

n

normalisr.normalisr, 6

INDEX

B

binnet() (*in module normalisr.normalisr*), 6

C

coex() (*in module normalisr.normalisr*), 6
compute_var() (*in module normalisr.normalisr*), 6

D

de() (*in module normalisr.normalisr*), 7

G

gotop() (*in module normalisr.normalisr*), 8

L

lcpm() (*in module normalisr.normalisr*), 9

M

module
    normalisr.normalisr, 6

N

normalisr.normalisr
    module, 6
normcov() (*in module normalisr.normalisr*), 10
normvar() (*in module normalisr.normalisr*), 10

P

pccovt() (*in module normalisr.normalisr*), 11

Q

qc_outlier() (*in module normalisr.normalisr*), 11
qc_reads() (*in module normalisr.normalisr*), 11

S

scaling_factor() (*in module normalisr.normalisr*), 12