Genialis

Normalizing RNA-seq data in Python with RNAnorm

Jure Zmrzlikar, Matjaž Žganec, Luka Ausec, Miha Štajdohar

December 2023

RNA sequencing (RNA-seq) is a workhorse method for quantifying gene expression by measuring the relative abundance of different RNA transcripts in a sample. Normalization is a critical piece of the RNA-seq bioinformatics pathway, used to mitigate bias and make the output more accurate and understandable. Appropriate interpretation of RNA-seq data depends on the type of normalization used—each normalization method has its own strengths and limitations, some being ideal for comparisons of genes within a sample and others being better suited for comparing expression of a gene across different samples. RNAnorm is a Python package and command line interface that flexibly fits into RNA-seq workflows and allows many different normalization approaches with one package. Here, we introduce commonly used RNA-seq normalization methods and demonstrate how to perform normalization using RNAnorm.

Content

Introduction

Have you been frustrated by the available options for normalizing RNA sequencing (RNA-seq) data? Accurate interpretation of the data rests on this critical piece of the bioinformatics pipeline, but maybe the current options for normalization don't fit neatly into your pipeline, or maybe you need a flexible option to normalize data in different ways depending on your biological question. RNAnorm is a Python package used to normalize RNA-seq data that can be used in Python scripts or in the command line to provide a flexible, open-source solution.

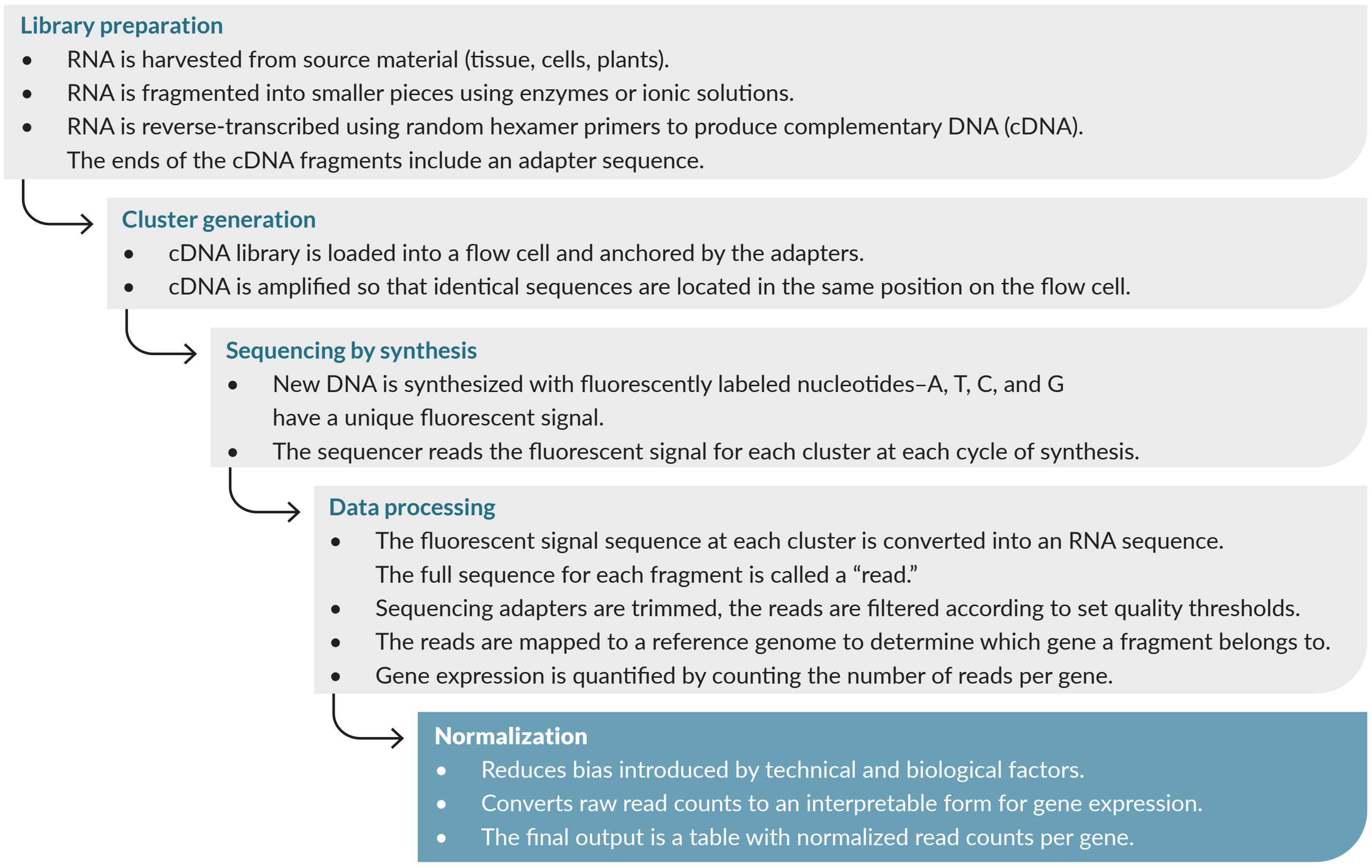
Below is a short tutorial on RNA-seq normalization methods and how to use them with RNAnorm. Check out the RNAnorm Github page and read on!

About RNA-seq

The central dogma of biology is that DNA is transcribed into messenger RNA (mRNA), and mRNA is translated into proteins. Expression of a protein-coding gene is controlled primarily by transcription, where the abundance of a particular mRNA is increased or decreased. RNA-seq is a method to measure the relative abundance of mRNA transcripts (Figure 1).

▼ Figure 1:

RNA-Seq: from raw material to quantified and normalized gene expression data.



Library preparation

- RNA is harvested from source material (tissue, cells, plants).
- RNA is fragmented into smaller pieces using enzymes or ionic solutions.
- RNA is reverse-transcribed using random hexamer primers to produce complementary DNA (cDNA). The ends of the cDNA fragments include an adapter sequence.

Cluster generation

- cDNA library is loaded into a flow cell and anchored by the adapters.
- cDNA is amplified so that identical sequences are located in the same position on the flow cell.

Sequencing by synthesis

- New DNA is synthesized with fluorescently labeled nucleotides–A, T, C, and G have a unique fluorescent signal.
- The sequencer reads the fluorescent signal for each cluster at each cycle of synthesis.

Data processing

- The fluorescent signal sequence at each cluster is converted into an RNA sequence. The full sequence for each fragment is called a "read."
- Sequencing adapters are trimmed, the reads are filtered according to set quality thresholds.
- The reads are mapped to a reference genome to determine which gene a fragment belongs to.
- Gene expression is quantified by counting the number of reads per gene.

Normalization

- Reduces bias introduced by technical and biological factors.
- Converts raw read counts to an interpretable form for gene expression.
- The final output is a table with normalized read counts per gene.

Why normalize gene expression data?

Everything is relative: RNA-seq does not measure the total RNA in a sample but provides the relative proportion of reads per gene within a library (total number of reads per sample). The usefulness of this raw data is limited by sources of bias. Normalization compensates for biological and technical sources of bias.

Bias source	Reason	Mitigation
Library size (total number of reads per sample)	RNA-seq can only quantify proportion of RNA molecules in comparison to the whole RNA output	Divide counts by library size
Gene length	Longer genes are cut into more fragments and therefore appear more expressed than shorter genes	Divide counts by gene length
RNA composition	A set of "outlier" genes that are differentially expressed can distort the library size	Find a more robust measure of RNA output than library size - scaling factors

▲ **Table 1:**

Common sources of bias and how normalization can compensate for it.

Because there is no normalization method that can compensate for everything, the selection of a normalization method will depend on the study question. Of the techniques available, RNA-seq normalization data can be grouped into "within-sample" or "between-sample" methods. Within-sample methods are used when you want to compare the expression of different genes in the same sample; between-sample methods are best for comparing the expression of a gene across different samples, such as comparing diseased vs healthy samples, screening in drug discovery, or time series methods.

Counts per million (CPM), fragments per kilobase million (FPKM), and transcripts per million (TPM) normalize to library size and are well-established methods for within-sample normalization. Upper quartile (UQ) and trimmed mean of M-values (TMM) are the basis for many between-sample methods, using RNA composition as a normalization factor to reduce the effect of outlier genes on the library.

Method and reference	Library size	Gene length	RNA composition	Between / within sample	Reference
CPM - Counts per million	✓			Within	/
FPKM - Fragments per kilobase million	✓	✓		Within	Mortazavi et al., 2008
TPM - Transcripts per million	✓	✓		Within	Wagner et al., 2012
UQ - Upper quartile	✓		✓	Between	Bullard et al., 2010
CUF - Count adjusted with UQ factors			✓	Between	Johnson and Krishnan, 2022
TMM - Trimmed mean of M-values	✓		✓	Between	Robinson and Oschlak, 2010
CTF - Count adjusted with TMM factors			✓	Between	Johnson and Krishnan, 2022
RLE - Relative log expression*	✓		✓	Between	Love et al., 2014
GeTMM - Gene length corrected*	✓	✓	✓	Between and within	Smid et al., 2018
MRN - Median Ratio Normalization*	✓		✓	Between	Maza et al., 2013

▲ Table 2:

Commonly used RNA-seq normalization methods, factors used for normalization, their within/between sample classification, and supporting references for further reading.

*These methods are being implemented into the RNAnorm package at the time of writing of this document (October 2023).

RNAnorm: a Python toolkit for RNA-seq normalization

The RNA-seq normalization methods presented above are implemented in various applications, generally in the R programming language, including EdgeR and DESeq2. We developed RNAnorm as a one-stop shop for RNA normalization that neatly fits into existing bioinformatics workflows, with features such as

- Native Python implementation,
- Unified API interface, compatible with scikit-learn,
- Verification with the original implementation,
- Command-line interface,
- Maintained and well-documented code,
- Open-source code with permissive Apache 2.0 license.

Sample dataset

You can install RNAnorm from PyPI to run the following examples on your own. In every example below, we will use the method load_toy_data() to load our sample dataset. We constructed this dataset to make it easy to understand how different methods transform the raw counts. The dataset has 4 samples, each with 5 genes (lengths 200, 300, 500, 1000, and 1000 bases). Sample 1 is a reference. Sample 2 is simply 2x all the counts of S1 - after library size normalization, S1 and S2 should be equal. Samples 3 and 4 are the same as S1 except for Gene #5, which is changed so that the library size is 0.5x or 2x of Sample 1, respectively. This produces an interpretable range of scaling factors for TMM/UQ.

CPM

Counts per million (CPM) is the simplest normalization method; it removes library size bias by dividing read counts by the library size expressed in millions. For read count X_g of gene g:

$$\begin{aligned} \mathrm{CPM}(X_g) &= X_g \cdot \frac{10^6}{N} \\ N &= \sum_g X_g \end{aligned}$$

The following example shows CPM in RNAnorm:

```
>>> from rnanorm.datasets import load_toy_data
>>> from rnanorm import CPM
>>> from sklearn import set_config
>>> set_config(transform_output="pandas")
>>> dataset = load_toy_data()
>>> dataset.exp
          Gene_1  Gene_2  Gene_3  Gene_4  Gene_5
Sample_1     200     300     500    2000    7000
Sample_2     400     600    1000    4000   14000
Sample_3     200     300     500    2000   17000
Sample_4     200     300     500    2000    2000

>>> CPM().fit_transform(dataset.exp) # perform CPM normalization
           Gene_1   Gene_2    Gene_3    Gene_4    Gene_5
Sample_1  20000.0  30000.0   50000.0  200000.0  700000.0
Sample_2  20000.0  30000.0   50000.0  200000.0  700000.0
Sample_3  10000.0  15000.0   25000.0  100000.0  850000.0
Sample_4  40000.0  60000.0  100000.0  400000.0  400000.0
```

We can see how CPM works in this 5-gene library across 4 samples. Sample 2 has double the raw counts as sample 1, but each gene is in the same proportion. Normalized to total library size, CPM renders their normalized counts the same.

Samples 3 and 4 are identical except for Gene_5, which is an outlier in sample 3. As a result, CPM makes the normalized counts look substantially different across all genes due to Sample 3's larger library size. This demonstrates why CPM can perform poorly on real world data that contains outlier genes.

RPKM, FPKM, and TPM

Reads per kilobase million (RPKM) and fragments per kilobase million (FPKM) normalize for library size and gene length. RPKM is designed for single-end reads, while FPKM is a variation of RPKM that is generalized for paired-end reads, where sequencing runs in both directions and the two reverse-complementary sequences correspond to one fragment.

Transcripts per million (TPM) normalizes for library size and gene length. This is the number of transcripts per million transcripts, giving a simple expression ratio within a sample.

FPKM and TPM are defined as:

$$\mathrm{FPKM}(X_g) = \mathrm{CPM}(X_g) \cdot \frac{10^3}{L_g} = \frac{X_g}{NL_g} \cdot 10^9$$

$$\mathrm{TPM}(X_g) = \mathrm{CPM}\left(X_g \cdot \frac{10^3}{L_g}\right) = \frac{X_g}{L_g} \cdot \left(\sum_j \frac{X_j}{L_j}\right)^{-1} \cdot 10^6$$

These three methods all normalize for library size and gene length, but do so in a different order. The example below shows the different outputs from FPKM and TPM:

```
>>> from rnanorm import FPKM, TPM
>>> dataset.exp
          Gene_1  Gene_2  Gene_3  Gene_4  Gene_5
Sample_1     200     300     500    2000    7000
Sample_2     400     600    1000    4000   14000
Sample_3     200     300     500    2000   17000
Sample_4     200     300     500    2000    2000
>>> FPKM(gtf=dataset.gtf_path).fit_transform(dataset.exp) # perform FPKM normalization
            Gene_1    Gene_2    Gene_3    Gene_4    Gene_5
Sample_1  100000.0  100000.0  100000.0  200000.0  700000.0
Sample_2  100000.0  100000.0  100000.0  200000.0  700000.0
Sample_3   50000.0   50000.0   50000.0  100000.0  850000.0
Sample_4  200000.0  200000.0  200000.0  400000.0  400000.0
>>> TPM(gtf=dataset.gtf_path).fit_transform(dataset.exp)# perform FPKM normalization
             Gene_1     Gene_2     Gene_3     Gene_4     Gene_5
Sample_1   83333.33   83333.33   83333.33  166666.66  583333.33
Sample_2   83333.33   83333.33   83333.33  166666.66  583333.33
Sample_3   45454.54   45454.54   45454.54   90909.09  772727.27
Sample_4  142857.14  142857.14  142857.14  285714.28  285714.28
```

In the raw data, we can see once again that the genes in samples 1 and 2 have the same ratios, but different read counts. Unlike our CPM example, where the read count was normalized only to library size, FPKM and TPM consider the gene length, so the ratios between genes after normalization are different from the raw counts.

UQ and CUF

The upper quartile (UQ) and Count adjusted with UQ factors (CUF) methods are less sensitive to the effects of outlier genes than the methods described previously.

UQ uses the expression value of the gene representing the 3rd quartile (upper quartile or 75th percentile) as the basis for scaling. It assumes that the gene in this position is unlikely to be highly variable, and can use the gene in that position to normalize a sample. UQ calculation has 3 steps:

1. determine the scaling factor by dividing the gene's 3rd quartile read count by library size,
2. rescale all factors to a geometric mean of 1,
3. multiply the scaling factor with the library size to determine the effective library size for use in a "CPM" calculation.

UQ and CUF are defined as:

$$
\begin{aligned}
\mathrm{UQ}(X_{sg}) &= X_{sg} \cdot \frac{10^6}{N_s \cdot F_s} \\
\mathrm{CUF}(X_{sg}) &= X_{sg} \cdot \frac{1}{F_s} \\
F_s &= \frac{f_s}{\left(\prod_i^n f_i\right)^{\frac{1}{n}}} \quad \text{for} \quad f_s \neq 0 \\
f_s &= \frac{\mathrm{Q}_{3s}}{N_s}
\end{aligned}
$$

CUF uses the same principles as UQ, but does not normalize for library size. CUF omits the 3rd step and normalizes counts by the scaling factors. See the next example for the normalization of our sample data with UQ and CUF:

```
>>> from rnanorm import UQ, CUF
>>> dataset.exp
          Gene_1  Gene_2  Gene_3  Gene_4  Gene_5
Sample_1     200     300     500    2000    7000
Sample_2     400     600    1000    4000   14000
Sample_3     200     300     500    2000   17000
Sample_4     200     300     500    2000    2000
>>> uq = UQ().fit(dataset.exp)
>>> uq.get_norm_factors(dataset.exp) # get UQ scaling factors
Sample_1  1.0
Sample_2  1.0
Sample_3  0.5
Sample_4  2.0
>>> uq.transform(dataset.exp) # perform UQ normalization
           Gene_1   Gene_2   Gene_3    Gene_4     Gene_5
Sample_1  20000.0  30000.0  50000.0  200000.0   700000.0
Sample_2  20000.0  30000.0  50000.0  200000.0   700000.0
Sample_3  20000.0  30000.0  50000.0  200000.0  1700000.0
Sample_4  20000.0  30000.0  50000.0  200000.0   200000.0
>>> CUF().fit_transform(dataset.exp) # perform CUF normalization
           Gene_1  Gene_2  Gene_3  Gene_4   Gene_5
Sample_1   200.0   300.0   500.0  2000.0   7000.0
Sample_2   400.0   600.0  1000.0  4000.0  14000.0
Sample_3   400.0   600.0  1000.0  4000.0  34000.0
Sample_4   100.0   150.0   250.0  1000.0   1000.0
```

As we can see in this example, UQ and CUF normalizations are robust against the influence of outlier genes, resulting in effective library sizes that preserve and reveal the similarities between all samples. This is more obvious with UQ, where each sample is normalized to library size, appearing more intuitive when comparing between samples. CUF is more suitable when the goal is to correct the RNA composition without adjusting for library size and can be thought of as simply scaling the raw read counts.

TMM and CTF

Trimmed mean of M-values (TMM) and counts adjusted with TMM factors (CTF) use a similar strategy to UQ and CUF, but instead of using the upper quartile, TMM uses the mean value of a trimmed set of values. TMM was first used in EdgeR, a popular R package for differential gene expression analysis, and assumes that most genes are not differentially expressed between conditions. This uses the mean value of a population of stable genes to normalize a sample.

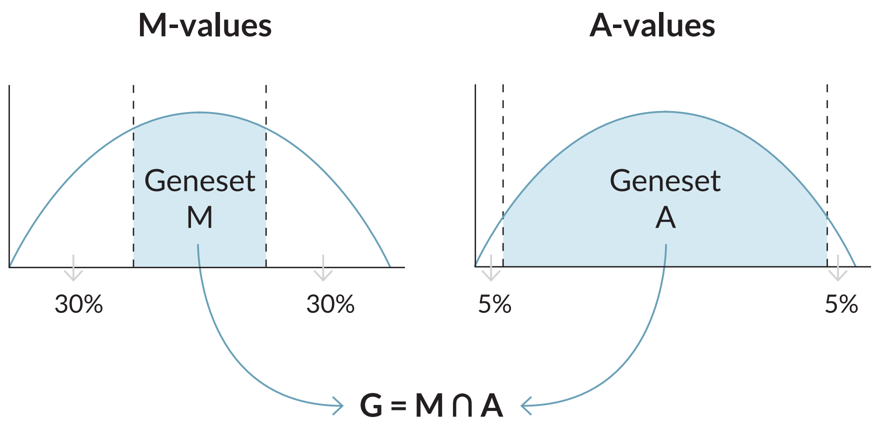
TMM and CTF normalization algorithms are mathematically expressed as follows:

$$
\begin{aligned}
\mathrm{TMM}(X_{sg}) &= X_{sg} \cdot \frac{10^6}{N_s \cdot F_s} \\
\mathrm{CTF}(X_{sg}) &= X_{sg} \cdot \frac{1}{F_s} \\
F_s &= \frac{f_s}{(\prod_i^n f_i)^{\frac{1}{n}}} \\
log_2(f_s^r) &= \frac{\sum_{g \in G} M_{sg}^r w_{sg}^r}{\sum_{g \in G} w_{sg}^r} \\
w_{sg}^r &= \frac{N_s - X_{sg}}{N_s X_{sg}} + \frac{N_r - X_{rg}}{N_k X_{rg}} \\
M_{sg}^r &= \log_2 \frac{X_{sg}/N_s}{X_{rg}/N_r} \\
A_{sg}^r &= \frac{1}{2} \log_2 \left(X_{sg}/N_s \cdot X_{rg}/N_r \right) \\
X_{sg}, X_{rg} \neq 0
\end{aligned}
$$

TMM can be summarized with the following steps:

1. Select a reference sample r
2. For each remaining sample,
   a. Compute M values and A values
   b. Determine “stable” gene set G by double-trimming M-values and A-values
   c. Compute factors as the weighted mean of M-values using only the gene set G
3. Rescale all factors to make the geometric mean equal to 1
4. Calculate effective library sizes and use them in a CPM calculation

Simply put, the scaling factor is determined by gene set G as the intersection of the M-trimmed mean (M) values and the absolute expression levels (A-values):



TMM and CTF are more complex models that normalize for RNA composition and can account for library size (TMM) or omit library size normalization (CTF).

Below is an example of TMM and CTF normalization in RNAnorm:

```
>>> from rnanorm import TMM, CTF
>>> dataset.exp
          Gene_1  Gene_2  Gene_3  Gene_4  Gene_5
Sample_1     200     300     500    2000    7000
Sample_2     400     600    1000    4000   14000
Sample_3     200     300     500    2000   17000
Sample_4     200     300     500    2000    2000
>>> tmm = TMM().fit(dataset.exp)
>>> tmm.get_norm_factors(dataset.exp) # calculate TMM normalization factors
Sample_1  1.0
Sample_2  1.0
Sample_3  0.5
Sample_4  2.0
>>> tmm.transform(dataset.exp) # perform TMM normalization
           Gene_1   Gene_2   Gene_3    Gene_4     Gene_5
Sample_1  20000.0  30000.0  50000.0  200000.0   700000.0
Sample_2  20000.0  30000.0  50000.0  200000.0   700000.0
Sample_3  20000.0  30000.0  50000.0  200000.0  1700000.0
Sample_4  20000.0  30000.0  50000.0  200000.0   200000.0
>>> CTF().fit_transform(dataset.exp) # perform CTF normalization
          Gene_1  Gene_2  Gene_3  Gene_4   Gene_5
Sample_1   200.0   300.0   500.0  2000.0   7000.0
Sample_2   400.0   600.0  1000.0  4000.0  14000.0
Sample_3   400.0   600.0  1000.0  4000.0  34000.0
Sample_4   100.0   150.0   250.0  1000.0   1000.0
```

Using RNAnorm in the command line

RNA-seq normalization is only one step in a bioinformatics pipeline. Instead of having the user create boilerplate scripts in Python, we implemented a command line interface of RNAnorm for easier integration in some workflows. In the example below, we show some options and arguments to use RNAnorm in the command line and demonstrate the single line command to perform a CPM normalization on a dataset ("exp.csv"):

```
$ rnanorm --help
Usage: rnanorm [OPTIONS] COMMAND [ARGS]...

  Common RNA-seq normalization methods.

Options:
  --help  Show this message and exit.

Commands:
  cpm   Counts per million
  ctf   Counts adjusted with TMM factors
  cuf   Counts adjusted with UQ factors
  fpkm  Fragments per kilo-base million
  tmm   Trimmed mean of M-values
  tpm   Transcripts per million
  uq    Upper quartile

$ rnanorm cpm --help
Usage: rnanorm cpm [OPTIONS] [EXP]

  Compute CPM.

Options:
  --out FILENAME  Output results in this file instead of stdout
  --force         Overwrite already existing output file
  --help          Show this message and exit.

$ cat exp.csv
,Gene_1,Gene2
Sample_1,500000,1500000
$ rnanorm cpm exp.csv
,Gene_1,Gene2
Sample_1,250000,750000
```

Scikit-learn compatibility

Scikit-learn is one of the most popular machine learning Python packages. RNAnorm's fit-transform design makes it compatible with scikit-learn, allowing users to build machine learning code on top of RNAnorm's basic functionality.

The following example shows an experiment to evaluate normalization methods for tissue classification:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
X, y = load_data()
X  # Matrix with read counts. Samples in rows, genes in columns.
y  # Target column (e.g. tissue type). Samples in rows.
pipeline = Pipeline(
    steps=[
        ('normalization', UQ()),
        ('scaler', StandardScaler()),
        ('classifier', LogisticRegression()),
    ]
)
params_grid = {
    'normalization': [UQ(), CUF(), TMM(), CTF()],
    'classifier__C': np.logspace(-5, 5, 11),
}
search = GridSearchCV(pipeline, params_grid, refit=False)
search.fit(X, y)
```

Best practices in RNA normalization

There is no single best RNA-seq normalization method, and it can be challenging to know which is best for your application. Here are a few best practices to help guide your decision:

Analysis goal		Recommendations
Compare expression values of genes within a sample	✓	TPM
Analyze differential expression of a gene between samples	✓	Follow guidelines of your differential expression tool (eg: EdgeR, DESeq2)
Use RNA-seq data in machine learning	✓	Use UQ, TMM, CUF, CTF
	✗	Avoid TPM and FPKM
	✓	For ML applications, scaling for gene length features is best done separately. We recommend Z-score standardization for scaling.
In all experiments	✓	Evaluate multiple methods when possible

For further reading, consult the references below.

▲ Table 3:

Recommendations for RNA normalization depending on your experiment.

About Genialis

Genialis, the RNA biomarker company, is creating a world where healthcare delivers the best possible outcomes for patients, their families and communities. ResponderID™, Genialis' machine-learning-driven disease modeling platform, delivers actionable biomarkers and optimally positions novel drugs to accelerate translational research, streamline drug development and ensure the best possible clinical care. Genialis is trusted by pharma and diagnostics partners, and together, we are transforming medicine through data. For more information, please visit www.genialis.com.

References

- Bullard, J. H., Purdom, E., Hansen, K. D., & Dudoit, S. (2010). Evaluation of statistical methods for normalization and differential expression in mRNA-Seq experiments. BMC Bioinformatics, 11(1), 94. https://doi.org/10.1186/1471-2105-11-94
- Johnson, K. A., & Krishnan, A. (2022). Robust normalization and transformation techniques for constructing gene coexpression networks from RNA-seq data. Genome Biology, 23(1), 1. https://doi.org/10.1186/s13059-021-02568-9
- Love, M. I., Huber, W., & Anders, S. (2014). Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology, 15(12), 550. https://doi.org/10.1186/s13059-014-0550-8
- Maza, E., Frasse, P., Senin, P., Bouzayen, M., & Zouine, M. (2013). Comparison of normalization methods for differential gene expression analysis in RNA-Seq experiments: A matter of relative size of studied transcriptomes. Communicative & Integrative Biology, 6(6), e25849. https://doi.org/10.4161/cib.25849
- Mortazavi, A., Williams, B. A., McCue, K., Schaeffer, L., & Wold, B. (2008). Mapping and quantifying mammalian transcriptomes by RNA-Seq. Nature Methods, 5(7), Article 7. https://doi.org/10.1038/nmeth.1226
- Robinson, M. D., & Oshlack, A. (2010). A scaling normalization method for differential expression analysis of RNA-seq data. Genome Biology, 11(3), R25. https://doi.org/10.1186/gb-2010-11-3-r25
- Smid, M., Coebergh van den Braak, R. R. J., van de Werken, H. J. G., van Riet, J., van Galen, A., de Weerd, V., van der Vlugt-Daane, M., Bril, S. I., Lalmahomed, Z. S., Kloosterman, W. P., Wilting, S. M., Foekens, J. A., IJzermans, J. N. M., Coene, P. P. L. O., Dekker, J. W. T., Zimmerman, D. D. E., Tetteroo, G. W. M., Vles, W. J., Vrijland, W. W., … on behalf of the MATCH study group. (2018). Gene length corrected trimmed mean of M-values (GeTMM) processing of RNA-seq data performs similarly in intersample analyses while improving intrasample comparisons. BMC Bioinformatics, 19(1), 236. https://doi.org/10.1186/s12859-018-2246-7
- Wagner, G. P., Kin, K., & Lynch, V. J. (2012). Measurement of mRNA abundance using RNA-seq data: RPKM measure is inconsistent among samples. Theory in Biosciences = Theorie in Den Biowissenschaften, 131(4), 281–285. https://doi.org/10.1007/s12064-012-0162-3