Reimplementing `OutSingle` into an R Package

Group 7
Kevin Jin
Yuchen Lei
Lingxuan Zhu

Abstract

The detection of RNA-Seq outliers can provide much information in the identification of rare genetic diseases. However, using common methods like calculating z-scores fails to detect all outliers due to the existence of confounders. However, there are not many methods specifically designed for this problem. In a very recently published study, Salkovic et al proposed a method named OutSingle to detect the outliers in the RNA-Seq data. In this method, SVD decomposition and optimal hard threshold were applied to gene-specific z-scores based on the log-transformation of the gene count data. The denoised z-score matrix will then be used to calculate the outlier score. In our project, we followed their algorithm and wrote the OutSingle method into an R package, which currently only has the Python script. We also tested the calculation speed with the original method developed in Python and compared it to another existing method for outlier detection. We find that this algorithm has a good performance in outlier detection in our dataset. Furthermore, it also works very efficiently. We also discussed some limitations of this algorithm based on our implementation. The R package has been uploaded https://github.com/LingxuanZ/OutSingleR.

Introduction

For most patients suffering from Mendelian disorders, no clear pathogenic variant can be determined after whole-genome sequencing. One possible reason is that individuals have a large number of rare non-coding variants, and the interpretation for these regions on the genome remains challenging [1]. Therefore, RNA-Seq can be used as an alternative method to identify pathogenic regulatory defects. RNA-Seq can help to reveal splicing defects, the mono-allelic expression of heterozygous loss-of-function variants, as well as expression outliers (i.e., genes aberrantly expressed outside their physiological range) [2, 3]. Before the development of any algorithms specifically used for outlier detection from RNA-Seq data, some studies have tackled this problem by using the calculation of z-scores of log-transformed gene-length-normalized read counts by subtracting the mean count and dividing by the standard deviation [2]. And by integrating DESeq2, a statistical test originally designed for differential expression analysis [4], researchers started being able to detect some outliers [3]. But methods specifically for outlier detection are still needed.

In 2018, OUTRIDER was developed as an algorithm to provide a statistical test for outlier detection in RNA-seq samples while controlling for covariates among the gene read counts, which takes advantage of an autoencoder to control for the common covariation patterns among genes [1]. More recently, OutPyR [5] and its improved version OutPyRX [6] were published, which estimate the parameters of the Bayesian model by using Markov chain Monte Carlo method. Even though these methods have relatively good results, the complicated computational steps they incorporated take a lot of time to run. In addition, OutPyR and OutPyRX lack cofounder control, and therefore, it is not good for the datasets with outliers that are masked by confounding effects [5, 6].

Recently, Salkovic et al proposed a method named OutSingle to detect outliers in RNA-Seq data [7]. In this method, SVD decomposition and optimal hard thresholding (OHT) were applied onto gene-specific z-scores which were derived from the log-transformation of the gene count data. This will result in a denoised z-score matrix and the outlier scores can be further calculated. Because these tasks are not as computationally challenging as the tasks done in previous methods, OutSingle should be able to not only run faster, but it will also have the capability to remove environmental noise from the dataset. Additionally, using SVD decomposition means that we can also reverse this process, allowing for the injection of outliers in datasets to create benchmarks for evaluating methods. Unfortunately, OutSingle can only be run through a Python script, so it is limited in use as some researchers prefer to use R language for analyzing the RNA-seq data.

Purpose

Given the wide practice of using the R programming in analyzing biological datasets, we would like to reimplement OutSingle into an R package. This package will become a useful tool to detect outliers from RNA-Seq data. In addition, when looking at the original Python implementation from the paper, we noticed that there was redundant/unnecessary code that also led to longer computational time. Therefore, during this reimplementation into an R package, we also optimized it. In particular, we optimized the SVD decomposition, which greatly increased the speed and efficiency of the package. We also compared this method to OUTRIDER [1], further showing that this algorithm can do a better job. Furthermore, while testing the package, we also noticed some limitations of this algorithm and the dataset we used for testing. We discuss these limitations, which should be revisited in further studies.

Description of OutSingle Algorithm (adapted from the original OutSingle paper, Salkovic et al, 2023 [7])

<u>Log-normal z-scores</u>

The input should be a $J \times N$ matrix where k_{ji} represents the gene counts from an RNA-Seq dataset. Note the rows represent genes $j \in 1, 2, \dots, J$ and the columns represent samples $i \in 1, 2, \dots, N$, where $J \gg N$.

Gene count k_{ji} is considered to follow a log normal distribution. As such, gene-specific z-scores for every count in the matrix were calculated. Sequencing depth of each sample was controlled using sample-specific DESeq size factor s_i. Then, the controlled counts c_{ji} was calculated and log transformed.

$$s_i = median \frac{k_{ji}}{(\prod_{t=1}^{N} k_{jt})^{\frac{1}{N}}}$$

$$c_{ji} = k_{ji}/s_i$$

$$l_{ji} = \log_2(\frac{c_{ji}+1}{\bar{c_j}+1}), \text{ where } \overline{c_j} \text{ are gene-specific mean values of } c_j$$

The l_{ji} values tend to be normally distributed with gene specific means $\overline{l_j}$ and standard deviation λ_j. Additionally, every matrix l_j is a vector holding a normal distribution $l_j \sim N(\overline{l_j}, \lambda_j)$. From this, the z-score for each gene were calculated to get a $J \times N$ z-score matrix $\tilde{Z}$ with elements $\tilde{z}_{ji} = \frac{l_{ji}-\mu_j}{\tau_j}$, where μ_j and τ_j are gene-specific means and standard deviations of l_{ji} values.

This z-score matrix $\tilde{Z}$ can be separated into two parts, $\tilde{Z} = Z + E$, where E is a Gaussian noise matrix. Z is a low-rank matrix, which is also called the "signal" and can "capture" the confounding effects. $\tilde{Z}$ is decomposed using SVD: $\tilde{Z} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$, where $\tilde{U}$ and $\tilde{V}$ are orthogonal matrices with $J \times J$ and $N \times N$ dimensions, and $\tilde{\Sigma}$=$diag_{J\times N}(\tilde{\sigma}_1^2, \dots, \tilde{\sigma}_N^2)$ is a rectangular diagonal $J \times N$ matrix. Its diagonal values are non-negative, called singular values, in descending order from the top left value to the bottom right value on the diagonal.

If the rank r of the low-rank signal matrix Z was known, then $\tilde{\Sigma}$ could be split into two matrices corresponding to Z and E. The rank of Σ would then be r, while the rank of Σ_E would $N - r$.

$$\tilde{\Sigma} = \Sigma + \Sigma_E ,$$

where

$$\Sigma = \begin{pmatrix} \Sigma_{[r,r]}^{r} & 0_{[r,N-r]} \\ 0_{[J-r,r]} & 0_{[J-r,N-r]} \end{pmatrix} \qquad \Sigma_E = \begin{pmatrix} 0_{[r,r]} & 0_{[r,N-r]} \\ 0_{[J-r,r]} & \Sigma_{[J-r,N-r]}^{n} \end{pmatrix}$$

$$\Sigma_{[r,r]}^{r} = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r^2 \end{pmatrix} \qquad \Sigma_{[J-r,N-r]}^{n} = \begin{pmatrix} \epsilon_1^2 & 0 & \cdots & 0 \\ 0 & \epsilon_2^2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \epsilon_{N-r}^2 \\ 0 & 0 & & 0 \end{pmatrix}$$

Then

$$\tilde{Z} = \tilde{U}(\Sigma + \Sigma_E)\tilde{V}^T = \tilde{U}\Sigma\tilde{V}^T + \tilde{U}\Sigma_E\tilde{V}^T$$
$$Z \approx \tilde{U}\Sigma\tilde{V}^T \qquad E \approx \tilde{U}\Sigma_E\tilde{V}^T$$

Optimal hard threshold [8]

An optimal method to eliminate the noise E from the matrix $\tilde{Z}$ is to find the rank r (i.e. singular value cutoff called OHT) of the low-rank matrix Σ. The assumption of E is as follows:

$$E = \gamma E_{SND} ,$$

where γ represents the scalar value corresponding to the magnitude of the noise and E_{SND} is a matrix whose values follow a standard normal distribution. In the case that γ is not known, an exact solution to find the rank r is as follows: $r = \omega(\beta)\sigma_{median}$, where σ_{median} is the median σ value of diagonal values of $\tilde{\Sigma}$ and $\beta = \frac{N}{J}$.

$$\omega(\beta) = \frac{\lambda(\beta)}{\mu_\beta},$$

where:

$$\lambda(\beta) = \left(2(\beta + 1) + \frac{8\beta}{(\beta + 1) + (\beta^2 + 14\beta + 1)^{\frac{1}{2}}} \right),$$

And μ_β is the solution to:

$$\int_{(1-\beta)^2}^{\mu_\beta} \frac{(((1+\sqrt{\beta})^2 - t)(t - (1-\sqrt{\beta})^2))^{1/2}}{2\pi t} dt = 1/2$$

This procedure will improve the computational speed compared with finding the latent dimension of an autophagy encoder because it avoids the iterated calculations for all potential r values.

Final outlier score calculated by OutSingle

Similar to the derivation of the z-scores, the e_{ji} elements of E follow a gene-specific normal distribution. From these distributions, the z-scores $\tilde{z}_{ji}$ were calculated. The rationale for implementing a similar derivation is because both masked and unmasked outliers (from the confounding effect), if present, will be included in the E matrix, considering that they are not part of the signal matrix Z. The true outliers will still be outliers corresponding to a single gene since the noise levels in E may vary across different genes. Then for each gene count, a p-value score $\hat{p}_{ji}$ can be calculated as $\hat{p}_{ji} = 2min\{\Phi(\hat{z}_{ji}), 1 - \Phi(\hat{z}_{ji})\}$, where Φ is the cumulative distribution function of standard normal distribution. Given that biological confounding effects cannot be completely removed, the expression levels of multiple genes within the same sample are not independent. The Benjamini–Yekutieli false discovery rate (FDR) approach [9], which is applicable with positive dependency, was applied for multiple tests correction. The outlier can be then determined by the p-values, i.e. the ones with small p-values.

Injecting artificial outliers by "inverting" the outlier detection procedure

Using OutSingle, it is possible to obtain matrix Z that contains confounding effects, and we can artificially generate the noise matrix similar to the original E matrix and then adding outliers with different magnitude. New $\tilde{Z}$ can be then calculated based on $\tilde{Z} = Z + E$ and it will be used to inversely calculate the gene counts. In this way, we can get a dataset with artificial injected outliers. Even though the outliers may be different from what happens in the real-world data, the injection method provides us with a way to create a dataset in which we know where the outliers are, and this dataset can be used for testing and comparing the performance of different algorithms.

Reimplementation

We first began by reimplementing the Outsingle Python package into a R package. The first thing we had to reimplement was the standardization for each gene and z-score calculation. We then performed SVD on the input data. During this step, we changed from the original methodology in order to be more efficient. One step we changed was that during the SVD step, we used the skinny version of the SVD, which makes it more efficient because we are not dealing with and storing a large matrix (as the number of genes is much greater than the number of patients). Furthermore, we also got rid of the step in the original Python implementation that performed SVD twice: once on the data and again on the transposed data. Instead, from our skinny SVD calculation, we are able to get the most optimal results more effectively.

After this SVD step, we then used the Optimal Hard Threshold (OHT) on our resulting diagonal matrix from SVD to control for the confounders. To do this, we reimplemented the OPTHT python package in R, and we used `apply` functions instead of the multiple iterative process they were originally doing. Lastly, we then implemented the p-value score calculation and readjustment by using the BY method according to the FDR.

Because the SVD is the most time-consuming step in the algorithm, we also tested the speed of SVD using the different BLAS and LAPACK libraries, svd under R's default BLAS, fast.svd under library corpcor, and svd using OPENBLAS.

Datasets

Two original datasets were used to test our package as well as to compare the performance of our R package with other existing approaches.

- **Kremer-119**: a dataset of transcriptome gene expression counts from skin fibroblast tissues of 119 patients with rare diseases published by Kremer et al (2017) [3].
- **GTEX-249**: RNA counts from skin were chosen. Data was Preprocessed by removing samples with low RNA integrity number (<5.7).

For both datasets, we kept the genes that have more than 5% of the samples which had FPKM (fragments per kilobase per millions of reads) greater than 1 and discarded genes that have 0 counts in more than 75% of the samples as described previously [1]. After these preprocessing, we get Kremer-119 dataset with 11462 genes and 119 samples and GTEX–249 datasets with 16986 genes and 249 samples. We include both datasets to show the performance of the algorithm for both small and relatively big datasets with more samples.

To create the data with a ground truth to test the performance of the method, we injected the outliers to one random gene per sample with different magnitude (z equal to 6,7 and 8) to generate benchmark datasets using the outlier injection function of the package. For each of the two original dataset and for each z values, we injected overexpressed outliers ("o"), whose values were greater than the mean of a particular gene, underexpressed outliers (“u”), whose values were lower than the mean of a particular gene and both outliers (“b”) with a ratio of 50%/50% of overexpressed and underexpressed outliers.

After injection, we got in total 18 datasets with ground truth outliers to test our package. We also run our package and OUTRIDER on the cleaned Kremer-119 without any artificial injection to test the performance with the usage of these methods in the real-world data.

Results

Comparison with original Python script

We optimized the SVD decomposition by using skinny SVD decomposition, instead of the full decomposition which is huge and computationally slow when performing the decomposition and construction of the matrix after removing the noise. In addition, we also removed some of the unnecessary parts from the original python code to make the method more efficient. Afterwards, we tested the speed of our R package relative to the original python script. We used one of the injected datasets with both overexpressed and underexpressed outliers (using a z score of 6) to test the speed of the two packages. The code for testing the time used by R and Python methods was documented in the Supplementary file.

It is of note that two functions are needed to run to get the final p value matrix, and we included the run time of each function in the supplementary. The time reported in Table 1 is the total time, i.e. the addition of time used by two functions. In addition, the time reported in the Supplementary is shorter than what has been included in the presentation and the first draft of the final report. This is possibly due to other concurrent programs running in the background of the local machine when running the two tests. We kept the original running times in the report but in both tests (report and supplementary) and both datasets, the running time using the R package is faster than the original Python code (Table 1 and Supplementary). This shows that our R implementation is more optimized than the python package. In addition, the final z-score results from running the two packages were the same, which suggests that the algorithm was implemented correctly; our modification for efficiency does not introduce errors.

Table 1. Running time of the two datasets tested by one of the injected datasets. The programs were running under the same conditions.

	OutSingle-R	OutSingle-Python
Kremer-119	14.912s	32.079s
GTEX-249	47.061s	96.419s

Comparison with OUTRIDER

More importantly, since this method was published very recently, it has not been used and evaluated a lot. As such, we would like to see how well this algorithm works in solving actual biological problems and compare it with the existing OUTRIDER algorithm (the details of running the OUTRIDER method were included in the Supplementary). We ran the 18 datasets using both methods and evaluated their performance by calculating the area under the precision-recall curve (AUC). Precision is defined as the ratio of real reported outliers and reported outliers. Recall is defined as the ratio of real reported outliers and the ground truth outlier numbers. The results are shown in Table 2 and 3 and Figure 1 and 2. The code for plotting and AUC calculation is documented in Supplementary with one example from each original dataset.

Compared with OUTRIDER, we found that OutSingle had better performance in all test datasets. Both algorithms worked better for the GTEX-249 datasets when compared to the Kremer datasets, which makes sense since the RNA-seq data in GTEX-249 were from normal tissues and should contain little to no outliers, but the Kremer data are from the patients with mitochondrial diseases, which contain outliers even before injection.

In addition, the running time of OutSingle is much lower than the running time of OUTRIDER. For the Kermer-119 dataset, OUTRIDER took around 135s while OutSingle only took 15s. For the larger dataset GTEX-249, OUTRIDER took about 10 min, but OutSingle only needed around 47s. (Again, this is the running time when we performed the test before the presentation, which is slower than the times shown in the supplementary materials. Importantly, we note that in both tests, the OutSingle method is much faster than the OUTRIDER. In addition, the AUC of the OUTRIDER may be different between runs possibly due to the usage of machine learning algorithm, but they are very close.)

Table 2. AUC of Kremer-119 datasets using OUTRIDER and OutSingle algorithms.

	Over- and Underexpressed	Overexpressed	Underexpressed
	z score = 6		
OUTRIDER	0.345208935454786	0.50733838061869	0.20314709878101
OutSingle	0.535617723401392	0.668214400800057	0.540694540169703
	z score = 7		
OUTRIDER	0.640920799834898	0.714548780362145	0.373295814464002
OutSingle	0.829807225910289	0.83514579066672	0.755145731994382
	z score = 8		
OUTRIDER	0.782070785714602	0.808300961767782	0.747433757533575
OutSingle	0.8618289679677	0.910969355337727	0.838554266944378

Table 3. AUC of GTEX-249 datasets using OUTRIDER and OutSingle algorithms.

	Over- and Underexpressed	Overexpressed	Underexpressed
	z score = 6		
OUTRIDER	0.404919866572425	0.614808498916274	0.222883032760593
OutSingle	0.720562381978434	0.723538189588019	0.694738394955205
	z score = 7		
OUTRIDER	0.644966479126506	0.846965090036513	0.591251375345833
OutSingle	0.867906241343171	0.896646320574535	0.83964706213674
	z score = 8		
OUTRIDER	0.840050051884489	0.898475267104337	0.732940287610872
OutSingle	0.908468437892513	0.939028868796413	0.878588738097098

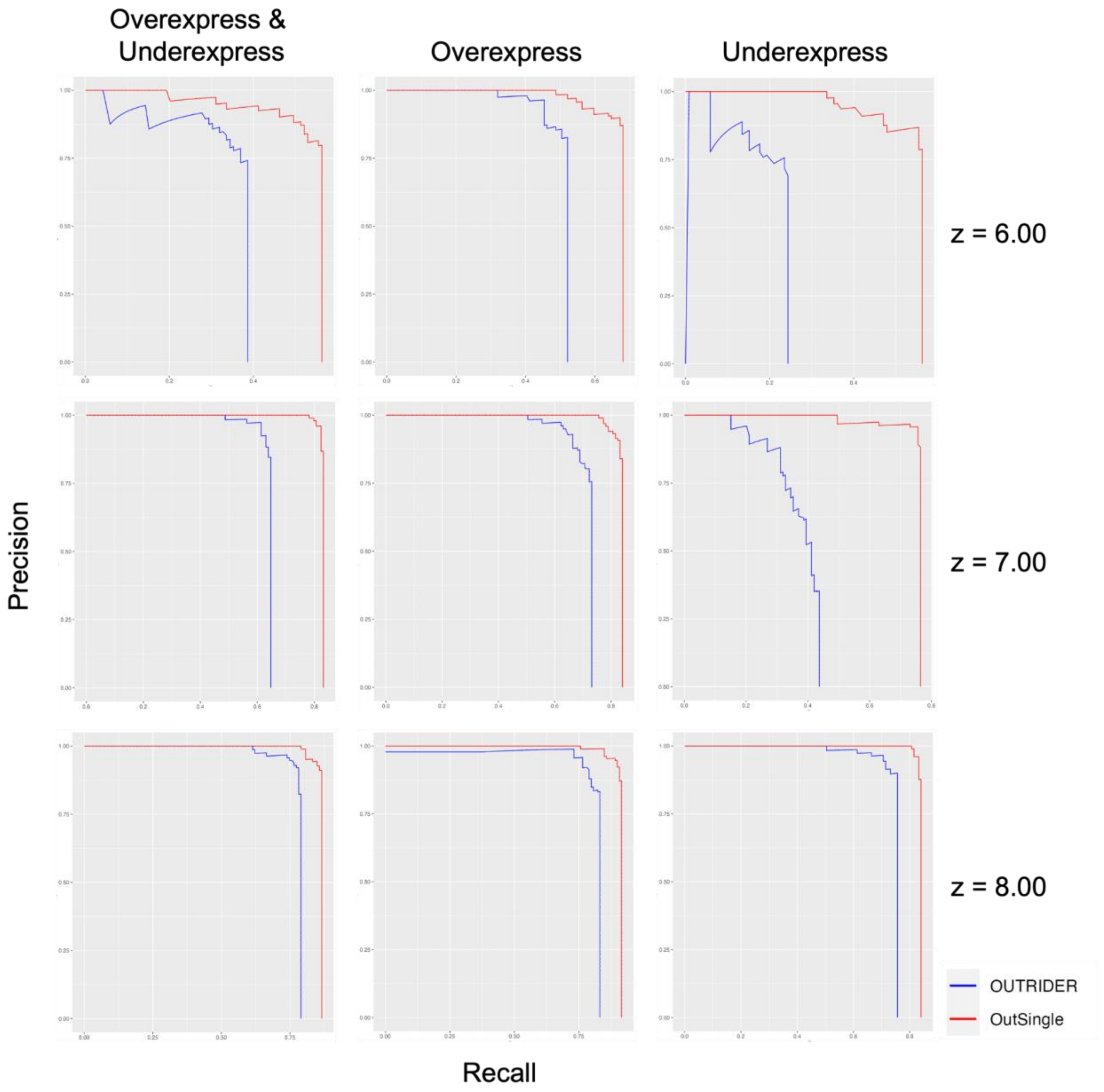

Figure 1. Precision recall curve for the Kremer-119 datasets. Rows are representing z-scores injection, from top to bottom as 6, 7 and 8. Each column represents the type of the injection, from left to right as both, overexpressed and underexpressed. AUC value see Table 2.

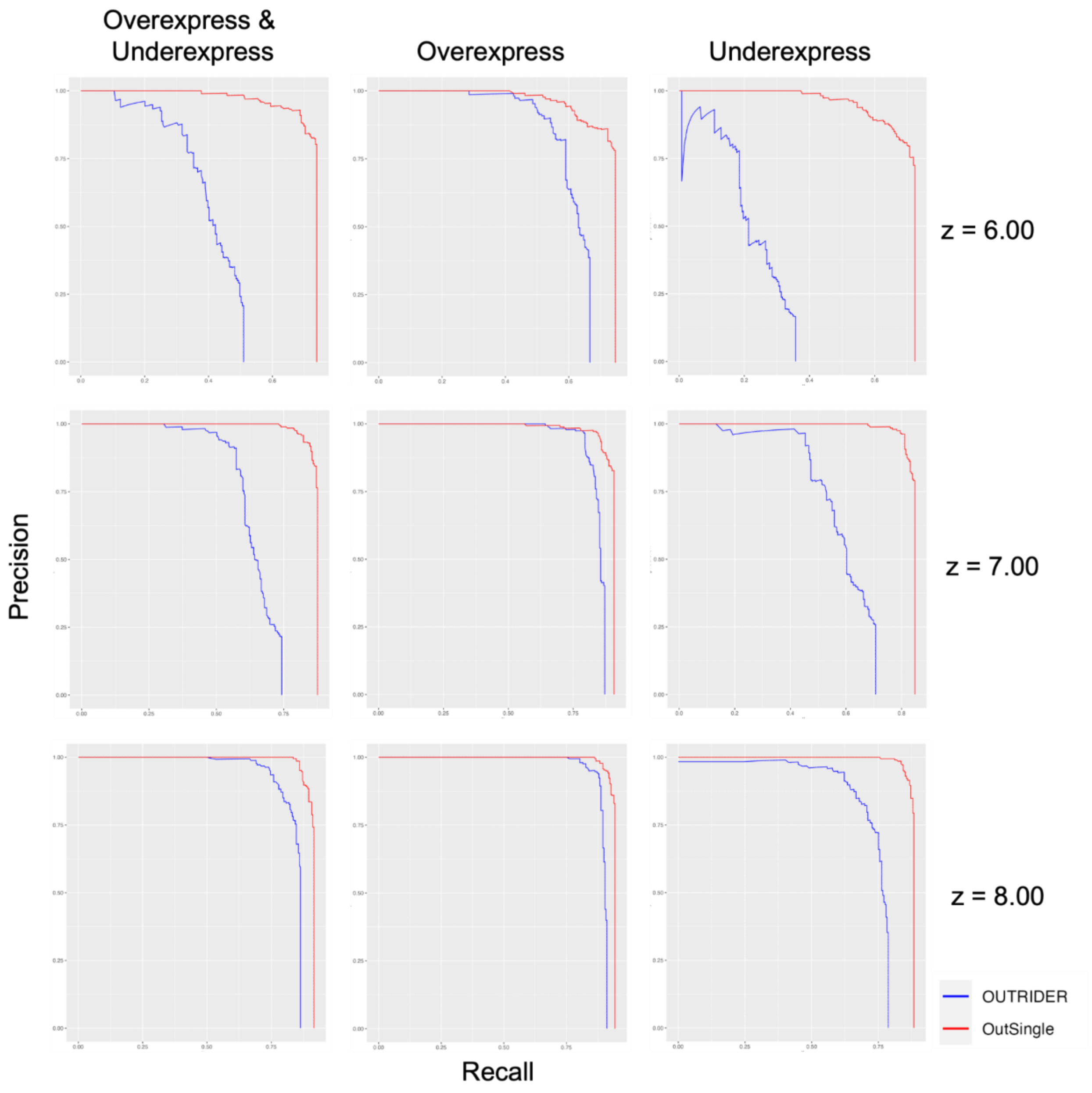

Figure 2. Precision recall curve for the GTEX-249 datasets. Rows are representing z-scores injection, from top to bottom as 6, 7 and 8. Each column represents the type of the injection, from left to right as both, overexpressed and underexpressed. AUC value see Table 3.

Evaluation using real-world data

Besides the artificially created datasets with injected outliers, we also evaluate the performance of this method directly with the real-world data. From Kremer et al [3], 6 outliers were reported and validated. We wanted to see whether our method can catch all these validated outliers. The p values were small for all 6 validated outliers, and they were reported with a common choice of p value threshold. When we set the p threshold as 0.05, 305 outliers will be reported; setting p threshold as 0.01, 199 outliers were reported. Whether these reports are the real outliers or reported as false positives needs further validation.

Comparison of SVD from different libraries

Lastly, we also evaluated using the OpenBLAS library to optimize the SVD in R. Through our testing, we noticed that the SVD step, despite being more optimized, was still taking a significant amount of time. We wanted to see how much more optimized it could be if we used the OpenBLAS library and compared it to the fast.svd function for corpcor. Using the exact same dataset from all methods, we evaluated how long it took for SVD in the table below.

Table 4. AUC of Kremer-119 datasets using OUTRIDER and OutSingle algorithms.

Method	Default BLAS \`svd\`	corpcor \`fast.svd\`	OpenBLAS \`svd\`
Time (s)	0.285	0.176	0.133

Looking at Table 4, we can see that the default \`svd\` method using the default BLAS library was the slowest as the corpcor \`fast.svd\` method was around 30% faster. However, using the OpenBLAS library and running the base \`svd\` took less than half the time of the default BLAS \`svd\`. As such, this shows that using the OpenBLAS library will optimize our OutSingle code even further than we have optimized it already, which will provide even more of a better performance compared to the original python implementation. We also decided to optimize reading the file into our code to provide additional efficiency. We used the \`fread\` function instead of the base R \`read.csv\` function which took around one third of the base time (0.016 seconds compared to 0.048 seconds).

Discussion

By reimplementing the OutSingle algorithm into an R package with additional optimizations, we achieved a computational efficient method to detect outliers from the RNA sequencing data, which could be applied in the diagnosis of Mendelian disorders and in the study of its mechanisms. However, during our reimplementation of OutSingle, we achieved a deeper understanding of the algorithm. From this newfound understanding, we found some limitations to OutSingle that we discuss below.

Firstly, this method is not good for the detection of outliers that have a relatively small z score. We tested injecting outliers with a magnitude of 3, 4 and 5 and OutSingle had a poor performance, with an AUC of around 0.2 even for the GTEX datasets, which are supposed to have a cleaner background. However, these small magnitude outliers could have biological interpretations, but people will not be aware of them due to the absence of accurate outlier reports. Additionally, when we evaluated OutSingle, we noticed these methods work better for detecting overexpressed outliers than underexpressed outliers. Since the method converts the gene count into normal distribution, this could potentially indicate this assumption is not valid to make, as this result suggests that the data is not symmetric so we cannot standardize assuming a normal distribution. Finally, when evaluating the importance of the algorithm, we were using the dataset with outliers injected by this method. It may not be very objective when evaluating the performance, but at least it is a way to get a dataset with ground truth. Datasets with outliers injected following other algorithms can be used for a further evaluation of the robustness of the OutSingle algorithm.

Contribution

KJ, YL and LZ worked together to write the code for the OutSingle algorithm, prepare the slides for presentation and write the report.

References

1. Brechtmann, F. *et al.* OUTRIDER: A Statistical Method for Detecting Aberrantly Expressed Genes in RNA Sequencing Data. *Am J Hum Genet* **103**, 907-917 (2018).
2. Cummings, B.B. *et al.* Improving genetic diagnosis in Mendelian disease with transcriptome sequencing. *Sci Transl Med* **9** (2017).
3. Kremer, L.S. *et al.* Genetic diagnosis of Mendelian disorders via RNA sequencing. *Nat Commun* **8**, 15824 (2017).
4. Love, M.I., Huber, W. & Anders, S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol* **15**, 550 (2014).
5. Salkovic, E., Abbas, M.M., Belhaouari, S.B., Errafii, K. & Bensmail, H. OutPyR: Bayesian inference for RNA-Seq outlier detection. *Journal of Computational Science* **47** (2020).
6. Salkovic, E. & Bensmail, H. A Novel Bayesian Outlier Score Based on the Negative Binomial Distribution for Detecting Aberrantly Expressed Genes in RNA-Seq Gene Expression Count Data. *IEEE Access* **9**, 75789-75800 (2021).
7. Salkovic, E., Sadeghi, M.A., Baggag, A., Salem, A.G.R. & Bensmail, H. OutSingle: a novel method of detecting and injecting outliers in RNA-Seq count data using the optimal hard threshold for singular values. *Bioinformatics* **39** (2023).
8. Gavish, M. & Donoho, D.L. The Optimal Hard Threshold for Singular Values is <inline-formula> <tex-math notation="TeX">\(4/\sqrt {3}\) </tex-math></inline-formula>. *IEEE Transactions on Information Theory* **60**, 5040-5053 (2014).
9. Benjamini, Y. & Yekutieli, D. The control of the false discovery rate in multiple testing under dependency. *The Annals of Statistics* **29** (2001).