

VIPER

Mengjie Chen, Xiang Zhou

May 18, 2018

VIPER version: 0.1.1

Please cite:

Mengjie Chen, Xiang Zhou.

Variability-Preserving Imputation for Accurate Gene Expression Recovery in
Single Cell RNA Sequencing Studies (VIPER)

Contents

- 1 Introduction of *VIPER* 3
 - 1.1 Installation 3
 - 1.2 Quick Start 4
- 2 Method. 4
 - 2.1 Preliminary analysis: checking cell-to-cell predictive power . . 5
 - 2.2 Running the imputation method 6
 - 2.3 Visualization 7
- 3 Session Info 9

1 Introduction of VIPER

VIPER is a R/C++ package that performs imputation for single cell RNA sequencing data.

While scRNAseq holds great promise in studies with complex cellular compositions, it also suffers from several important technical disadvantages that limit its use in many settings. These disadvantages include low transcript capture efficiency, low sequencing depth per cell and wide-spread dropout events. As a consequence, the gene expression measurements obtained in scRNAseq often contain a large amount of zero values, many of which are due to dropout events. Excess of zero values hinders the application of scRNAseq in accurate quantitative analysis. In addition, standard analytic methods developed under bulk RNAseq settings do not account for the excess of zero values observed in scRNAseq data, thus direct application of these methods to scRNAseq often results in sub-optimal performance.

To mitigate the influence of excessive zero values, we develop VIPER, a straightforward, accurate, free-of-tuning, and relatively computationally efficient scRNAseq imputation method, which we refer as Variability-Preserving Imputation for Accurate Gene Expression Recovery in Single Cell RNA Sequencing Studies (VIPER). VIPER borrows information across cells of similar expression pattern to impute the expression measurements in the cell of interest. However, unlike some of the previous cell-based methods, VIPER does not perform cell clustering before imputation nor uses only cells that belong to the same cell subpopulation for imputation. Instead, VIPER applies a weighted penalized regression model to actively select a sparse set of local neighborhood cells that are most predictive of the cell of interest. The selection of this sparse set of cells is done in a progressive manner and their associated imputation weights are estimated in the final inference step to ensure both robustness and computational scalability. In addition, VIPER explicitly accounts for expression measurement uncertainty of the zero values in scRNAseq by modeling the dropout probability in a cell-specific and gene-specific fashion. VIPER uses an efficient quadratic programming algorithm that infers all modeling parameters from the data at hand while keeping computational cost in check. We have shown the effectiveness of our approach in our manuscript. This method is implemented in function `VIPER()`.

1.1 Installation

VIPER relies on the following R packages: `Rcpp`, `RcppArmadillo`, `glmnet` and `quadprog`. It can be installed directly from github.

```
install.packages("devtools")
library(devtools)
install_github("ChenMengjie/VIPER")
```

Package `easyGgplot2` will be used in some graphic functions. But it is not necessary for imputation.

```
library(devtools)
install_github("kassambara/easyGgplot2")
```

1.2 Quick Start

`VIPER()` is the major function that implements the imputation methods. Details about the steps can be found in later sections. To perform imputation, simply run:

```
library(VIPER)
data(grun)
res <- VIPER(gene.expression, num = 5000, percentage.cutoff = 0.1,
             minbool = FALSE, alpha = 0.5, report = TRUE, outdir = "/dir/to/output")
```

where `gene.expression` is a $p \times n$ matrix of gene expression levels for p genes from n samples. When `report = TRUE`, the results are written to csv files under the specified directory or the current working directory when not specified. `VIPER()` returns the following list of values:

- `imputed log`: A $p \times n$ matrix of log transformed gene expression levels after imputation.
- `imputed`: A $p \times n$ matrix of gene expression counts converted from imputed log transformed values.
- `sample weights`: A $n \times n$ matrix of estimated imputation weights. Each row represents a cell.
- `outliers`: The indexes of cells that have no selected candidate neighbors according to the penalized regression model. The zero values in these cells are not imputed.

2 Method

VIPER uses a sparse nonnegative regression model to model the gene expression levels of the cell of interest as a weighted summation of a sparse set of its neighborhood cells. These neighborhood cells and their imputation weights are inferred through a computationally efficient two-step procedure that include a pre-selection step and an estimation step. In the pre-selection step, VIPER identifies a moderate-sized set of candidate cells that are likely predictive of the expression levels of the cell of interest and that will serve as a candidate pool for the final selection of neighborhood cells. The pre-selection step is done efficiently using a standard penalized regression

method based on either lasso or elastic net, and is designed to mitigate the computation burden of the later estimation step. In the estimation step, with the selected candidate cells, VIPER fits a sparse nonnegative regression model using a quadratic programming algorithm to further identify a final set of neighborhood cells and estimate their weights for imputation. The technical details of VIPER can be found in our manuscript.

2.1 Preliminary analysis: checking cell-to-cell predictive power

VIPER primarily borrows information across neighborhood cells to perform imputation. So it is important to make sure that the expression levels of a given cell can be predicted by expression levels of other cells in the data. To check on this, we can apply a standard penalized linear regression model to predict the expression of a given cell by regressing on the expression of all other cells. The results on prediction performance can be visualized by a density plot of in-sample R^2 . Similarly we can predict the expression of a given gene by regressing on the expression of all other genes.

```
library(VIPER)
data(grun)
testcell <- PredictCell(gene.expression, GeneNum = 1000,
  ZeroRate = 0.1)
testGene <- PredictGene(gene.expression, GeneNum = 500,
  ZeroRate = 0.1)
PlotR2(testcell, testGene)
```

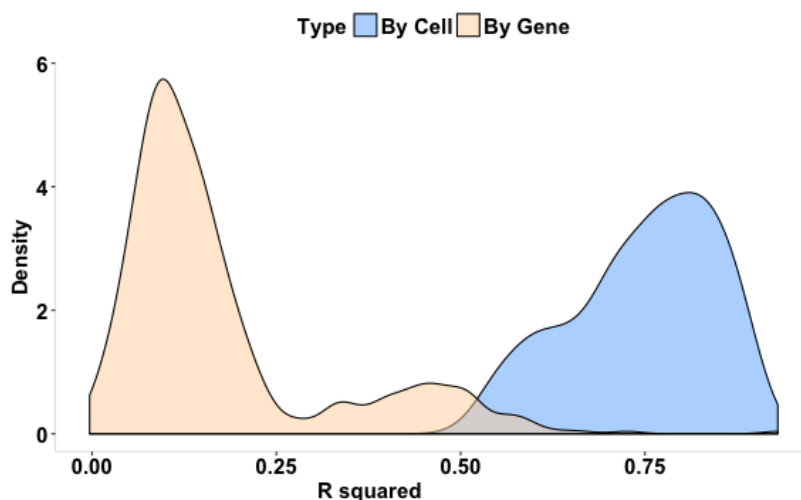


Figure 1: R square of cell-to-cell prediction and gene-to-gene prediction

Using Grun et al dataset as an example, we can see that prediction of the expression of a given cell based on other cells reaches a mean of 0.75 across all cells, which is much higher than that from genes (a mean of 0.16). It suggests neighborhood

cell based imputation strategy is suitable for this dataset. In our limited experiments (less than 10 datasets), we find the above conclusion holds for all. However, we acknowledge that all examined datasets contain a smaller number of cells than the number of genes. It is likely that gene-to-gene prediction will have high power when the number of cells exceeds the number of genes. Please let us know if you encounter a such example. This step may take hours for large datasets.

2.2 Running the imputation method

```
system.time(res <- VIPER(gene.expression, num = 5000, percentage.cutoff = 0.1,
  minbool = FALSE, alpha = 0.5, report = FALSE))

##      user  system elapsed
##      787      23      810
```

Our method is not restricted to the units of measurement and is applicable to all normalized measurements such as RPM (reads per million reads), TPM (transcripts per kilobase per millions reads) or RPKM (reads per kilobase per millions reads).

- `num` is the number of random sampled genes used to fit the penalized regression model to identify the set of candidate cells. The default value is 5000.
- To reduce the influence of missing values in the weight estimation, the nonnegative regression model is fitted using genes with a zero rate less than a certain threshold. The threshold is set using `percentage.cutoff`.
- VIPER calls `cv.glmnet()` in `glmnet` to perform fitting cross validation. Two penalty levels are available for selection of penalty level λ : `lambda.min`, the value of λ that gives minimum mean cross-validated error, and `lambda.1se`, the value of λ that gives the most regularized model such that error is within one standard error of the minimum. The default is `lambda.1se`, i.e., `minbool = FALSE`. Parameter `alpha` sets the elastic net mixing parameter. The default value is 0.5. Setting `alpha = 1` is equivalent to a lasso model.

`VIPER()` returns the following list of values:

- `imputed log`: A $p \times n$ matrix of log transformed gene expression levels after imputation.
- `imputed`: A $p \times n$ matrix of gene expression counts converted from imputed log transformed values.
- `sample weights`: A $n \times n$ matrix of estimated imputation weights. Each row represents a cell.
- `outliers`: The indexes of cells that have no selected candidate neighbors according to the penalized regression model. The zero values in these cells are not imputed.

2.3 Visualization

```
celltype <- gsub("_[0-9][0-9]$", "", colnames(gene.expression))
celltype <- gsub("_[0-9]$", "", celltype)
PlotWeights(res, KnownLabel = celltype)
```

Function `PlotWeights()` can be used to visualize the local neighbors selected for each cell. Each row represents a cell. All the cells with an estimated weight greater than 0.01 are colored in red in the heatmap (Figure 2).

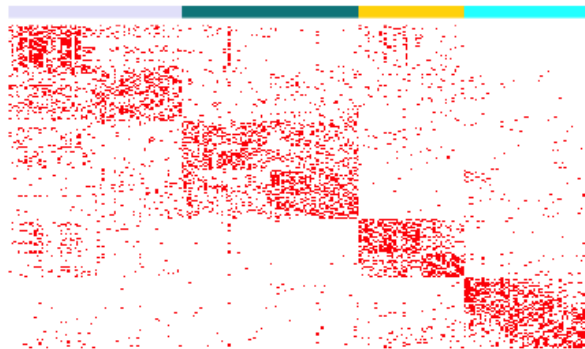


Figure 2: Visualization of the local neighbors selected for each cell

```
PlotCV(res, gene.expression, KnownLabel = celltype, GeneNum = 2000,
       selection = "Zero")
```

Finally, we can check how the variabilities cross cells change after the imputation (Figure 3). To do so, for each gene in turn, we computed the coefficient of variation (CV) across cells after imputation and compared it to the CV of the non-zero values before imputation. We contrasted these two CV values and stratified the contrast by showing different zero proportions with colors in gradient. Intuitively, for a given gene, if the zero values across cells are all due to dropout events, then we would expect the CV after imputation to be similar to the CV before imputation - because the imputed data would follow the same distribution as the non-zero values before imputation. In contrast, if the zero values are all due to low gene expression levels, then we would expect the CV after imputation to be higher than the CV before imputation - because the imputed data would generally have lower values than the non-zero values before imputation.

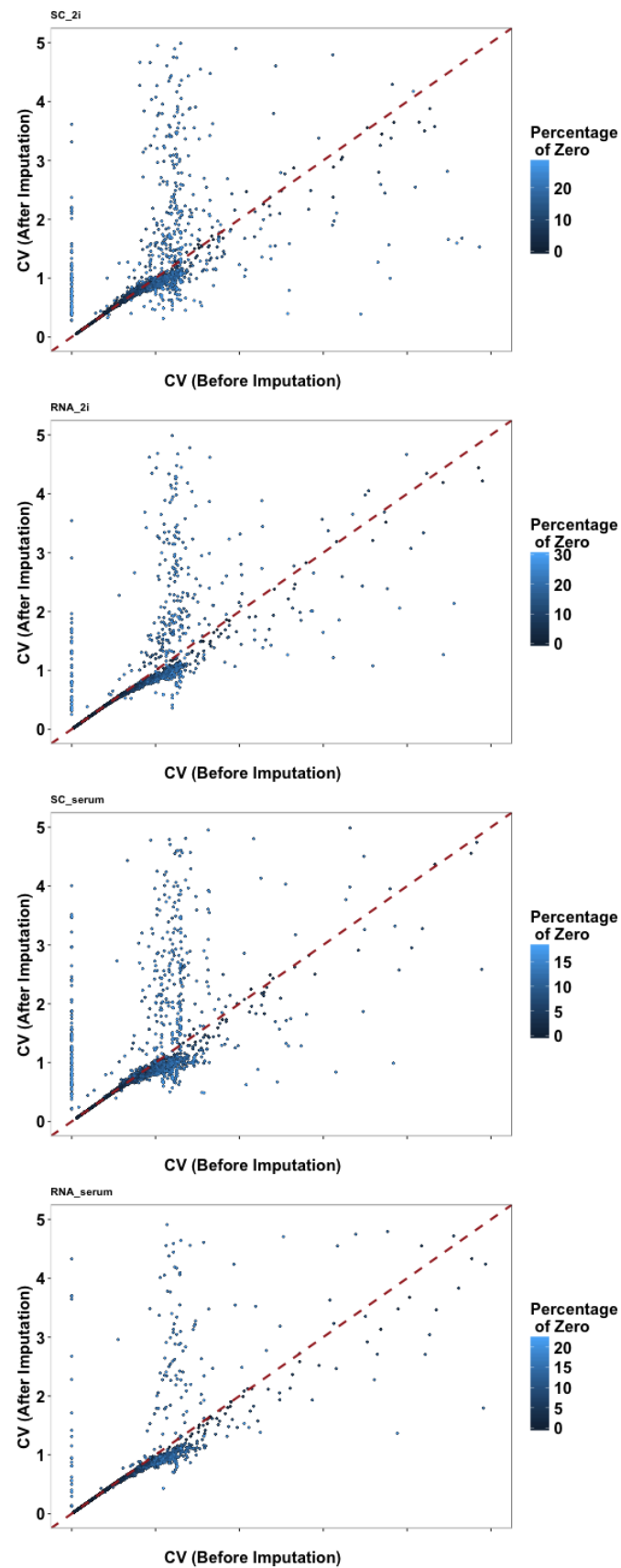


Figure 3: Visualization of CV before and after imputation

3 Session Info

```

sessionInfo()

## R version 3.5.0 (2018-04-23)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.4
##
## Matrix products: default
## BLAS: /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framework/Ver
## LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets
## [6] methods    base
##
## other attached packages:
## [1] gplots_3.0.1          quadprog_1.5-5
## [3] easyGgplot2_1.0.0.9000 ggplot2_2.2.1
## [5] glmnet_2.0-16         foreach_1.4.4
## [7] Matrix_1.2-14         knitr_1.20
## [9] VIPER_0.1.1
##
## loaded via a namespace (and not attached):
## [1] Rcpp_0.12.16          compiler_3.5.0
## [3] pillar_1.2.2          formatR_1.5
## [5] highr_0.6             plyr_1.8.4
## [7] bitops_1.0-6          iterators_1.0.9
## [9] tools_3.5.0           digest_0.6.15
## [11] evaluate_0.10.1       tibble_1.4.2
## [13] gtable_0.2.0          lattice_0.20-35
## [15] rlang_0.2.0           yaml_2.1.19
## [17] stringr_1.3.0         gtools_3.5.0
## [19] caTools_1.17.1        rprojroot_1.3-2
## [21] grid_3.5.0            rmarkdown_1.9
## [23] gdata_2.18.0          magrittr_1.5
## [25] backports_1.1.2       scales_0.5.0
## [27] codetools_0.2-15      htmltools_0.3.6
## [29] BiocStyle_2.8.0       colorspace_1.3-2
## [31] labeling_0.3          KernSmooth_2.23-15
## [33] stringi_1.1.7         lazyeval_0.2.1
## [35] munsell_0.4.3

```