Use case: Benchmarking method parameters

Shian Su

2018-12-07

Contents

1 Introduction

This use case shows how to test a range of parameters for a given method. We will use the CelSeq2 mRNA mixture data from 3 lung adenocarcinoma cell-lines and apply knn-smooth with various `k` parameter to see its effects on the output.

2 Setting up benchmark

```r
library(CellBench)
library(SingleCellExperiment)
library(dplyr)
library(purrr)
library(ggplot2)
```

We load in the data and create a list of 1 SingleCellExperiment object.

```r
cellbench_mrna_mix_data <- load_mrna_mix_data()

data_list <- list(
    mrna_mix_celseq = cellbench_mrna_mix_data$mrna_mix_celseq
)
```

We need to write some small wrappers to help run pipelines and make methods uniform in input and output. This is necessary because each step of analysis should take in the same type of data and output the same type of data, however different methods may differ in how they are called, how many steps need to run and what they output. Wrappers help manage this, in this example we want our normalisation step to take in a SingleCellExperiment and output a normalised count matrix. The imputation step should take a count matrix and return an imputed counts matrix.

```r
# take in a SingleCellExperiment and return a scran normalised
# expression matrix
scran_norm_expr <- function(x) {
    stopifnot(is(x, "SingleCellExperiment"))

    x <- scran::computeSumFactors(x)
    x <- scater::normalize(x, return_log = FALSE)

    SingleCellExperiment::normcounts(x)
}

# take in an expression matrix and return the imputed expression matrix
impute_knn_smooth <- function(expr, k) {
    source("https://raw.github.com/yanailab/knn-smoothing/master/knn_smooth.R")
    smoothed_mat <- knn_smoothing(mat = expr, k = k)
    smoothed_mat
}
```

We then create the lists of functions to use with CellBench. We only have one normalisation method, but for imputation we can create a series of partially applied functions with different `k` parameters. Here assuming we have `f(x, y)`, `partial(f, y = 1)` is equivalent to `function(x) f(x, y = 1)`, partial application "fills in" parameter values and returns a function that can be called.

```
norm_method <- list(
    scran = scran_norm_expr
)

# identity simply returns its argument, here it's used to represent
# no imputation
impute_method <- list(
    "none" = identity,
    "impute_knn_smooth(k = 2)" = partial(impute_knn_smooth, k = 2),
    "impute_knn_smooth(k = 4)" = partial(impute_knn_smooth, k = 4),
    "impute_knn_smooth(k = 8)" = partial(impute_knn_smooth, k = 8),
    "impute_knn_smooth(k = 16)" = partial(impute_knn_smooth, k = 16),
    "impute_knn_smooth(k = 32)" = partial(impute_knn_smooth, k = 32)
)
```

First we apply the normalisation method we can store the result of this in `res_norm` and print it.

```
res_norm <- data_list %>%
    apply_methods(norm_method)
## Warning in .get_all_sf_sets(object): spike-in set 'ERCC' should have its own
## size factors

res_norm
## # A tibble: 1 x 3
##   data            norm_method result
##   <fct>           <fct>       <list>
## 1 mrna_mix_celseq scran       <dbl [14,804 x 340]>
```

We see that it's created a tibble where the column name is taken from the name of the lists and values in the columns correspond to names within the lists. The result of the computation performed is stored in the last column. If we had another normalisation method to test then we could go back and add it to the `norm_method` list and run this code again.

We can then apply our next set of methods to the results of the normalisation. This expands our table to more rows, keeping track of the combinations of `data`, `norm_method` and `impute_method`, as well as updating the results with the latest methods applied.

```
res_impute <- res_norm %>%
    apply_methods(impute_method)

res_impute
## # A tibble: 6 x 4
##   data            norm_method impute_method             result
##   <fct>           <fct>       <fct>                     <list>
## 1 mrna_mix_celseq scran       none                      <dbl [14,804 x 340]>
## 2 mrna_mix_celseq scran       impute_knn_smooth(k = 2)  <dbl [14,804 x 340]>
## 3 mrna_mix_celseq scran       impute_knn_smooth(k = 4)  <dbl [14,804 x 340]>
## 4 mrna_mix_celseq scran       impute_knn_smooth(k = 8)  <dbl [14,804 x 340]>
## 5 mrna_mix_celseq scran       impute_knn_smooth(k = 16) <dbl [14,804 x 340]>
## 6 mrna_mix_celseq scran       impute_knn_smooth(k = 32) <dbl [14,804 x 340]>
```

Now we have the computation output of all the pipelines of methods, each producing a different imputed expression in the result column. From here we could calculate some metrics, in this instance we will create principal component plots of the imputed values for a visual assessment.

We create a new method list containing a PCA transformation that returns a data.frame of two columns containing the principal component coordinates. We could add more dimensionality reduction methods into our list later on to look at it in different ways. We also transform our counts into log-scale before we perform PCA for more stable scaling.

```
dim_red <- list(
    pca = compute_pca
)

# log-transform the counts
res_impute$result <- lapply(res_impute$result, function(x) log2(x + 1))

res <- res_impute %>%
    apply_methods(dim_red)

res
## # A tibble: 6 x 5
##   data           norm_method impute_method          dim_red result
##   <fct>          <fct>       <fct>                  <fct>   <list>
## 1 mrna_mix_cel~ scran       none                   pca     <data.frame [340~
## 2 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
## 3 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
## 4 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
## 5 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
## 6 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
```

We now have a benchmark_tbl where the results column contains two PCA coordinates. If we plot the PCA values in the results column as is, we will have a bunch of points on a plot with no additional context. We actually have annotation for these cells in terms of mixture proportions so we can append it to our results and create a more informative visualisation. This could be done using a for-loop where you iterate through the results and reassign it with the annotated version, but I will introduce a functional programming approach to this task.

We define a function that takes in a data_key, the name of the dataset and the result data.frame. This function will add a new column to the result data.frame called "truth" which is the concatenated proportion values, so each combination of mixutre proportions is a unique group. Because we need information from both data and result columns, we use `map2` from `purrr` which allows use to map functions with two arguments to two columns.

```
# function to add annotation to results table
# we could have used a single variable because we're only looking at one dataset
# but this gives us more flexibility if we want to add more datasets to our list
```

```
append_anno <- function(data_key, result) {
    # extract the desired SingleCellExperiment from our data list
    data <- data_list[[data_key]]

    # take the sample annotations from colData
    sample_anno <- colData(data)

    # create the truth values by concatenating cell line proportions
    truth <- paste(
        sample_anno$H2228_prop,
        sample_anno$H1975_prop,
        sample_anno$HCC827_prop
    )

    # add truth values as a column to result and return the result
    result$truth <- truth
    result
}

# replace the result column with annotated results
annotated_res_list <- map2(res$data, res$result, append_anno)

annotated_res <- res %>%
    dplyr::mutate(result = annotated_res_list)

annotated_res
## # A tibble: 6 x 5
##   data          norm_method impute_method          dim_red result
##   <fct>         <fct>       <fct>                  <fct>   <list>
## 1 mrna_mix_cel~ scran       none                   pca     <data.frame [340~
## 2 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
## 3 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
## 4 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
## 5 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
## 6 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
```

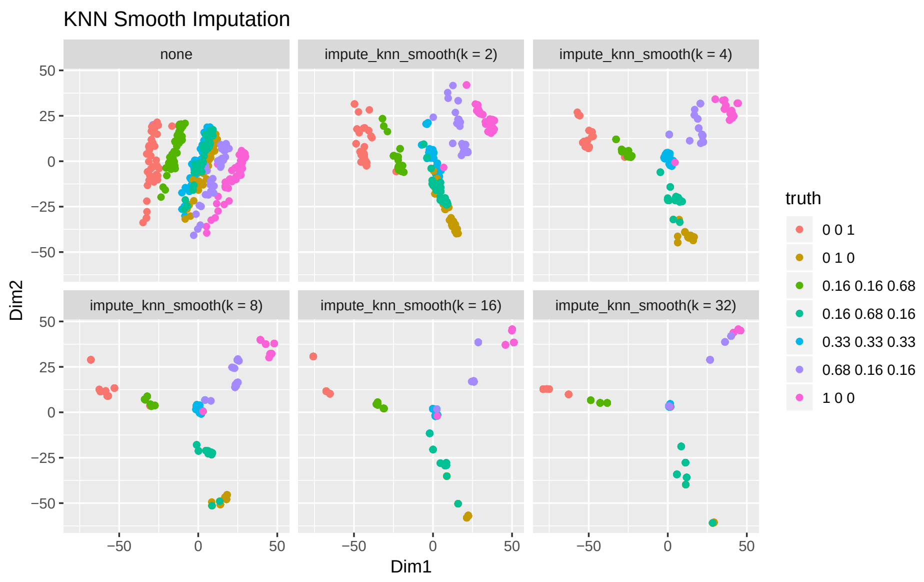
There's now a new column in the results data.frames containing the "truth" mixture groups. We want to plot all of these PCA values, it's possible to use a for-loop to iterate through the columns and call many plots, but here we can use `unnest` from `tidy` to get a nice tabular format that allows us to use ggplots. The data.frames in the result column are unnested in that they are combined row-wise, and the remaining columns are duplicated to match the new structure. For this to work the data.frames in the result column must have the same columns.

```
plot_df <- tidyr::unnest(annotated_res)

plot_df
## # A tibble: 2,040 x 7
##    data          norm_method impute_method dim_red   Dim1   Dim2 truth
##    <fct>         <fct>       <fct>         <fct>    <dbl>  <dbl> <chr>
##  1 mrna_mix_cel~ scran       none          pca     -28.8   20.0  0.68 0.16 ~
##  2 mrna_mix_cel~ scran       none          pca      -1.27  -4.79 0.33 0.33 ~
##  3 mrna_mix_cel~ scran       none          pca        4.46  10.5  0.33 0.33 ~
##  4 mrna_mix_cel~ scran       none          pca        3.72  11.4  0.33 0.33 ~
##  5 mrna_mix_cel~ scran       none          pca        3.64   8.51 0.33 0.33 ~
##  6 mrna_mix_cel~ scran       none          pca        4.19  10.1  0.33 0.33 ~
##  7 mrna_mix_cel~ scran       none          pca        3.20   6.92 0.33 0.33 ~
##  8 mrna_mix_cel~ scran       none          pca        1.40   6.78 0.33 0.33 ~
##  9 mrna_mix_cel~ scran       none          pca        5.95  15.5  0.33 0.33 ~
## 10 mrna_mix_cel~ scran       none          pca        5.49  15.8  0.33 0.33 ~
## # ... with 2,030 more rows
```

With this tidy data.frame it is easy to use ggplots to visualise the PCA, colouring the points by the truth group. We use `facet_wrap` to create a facetted plot so that each plot has a common scale, the impute_method is used as the facetting variable so each plot shows a different imputation.

```
plot_df %>%
    ggplot(aes(x = Dim1, y = Dim2, col = truth)) +
    geom_point() +
    facet_wrap(~impute_method, nrow = 2) +
    ggtitle("KNN Smooth Imputation")
```



We see that performing PCA on the unimputed data already separates the groups out, but not quite in the structure we expected and with quite a bit of spread within the groups. It appears that increasing the k parameter in the knn-smooth algorithm will pull the points toward the designed structure reflecting the mrna proportions. Though it appears if k is set too high then points will be pulled into almost singular points and potentially lose much of their true variability.

3 Summary

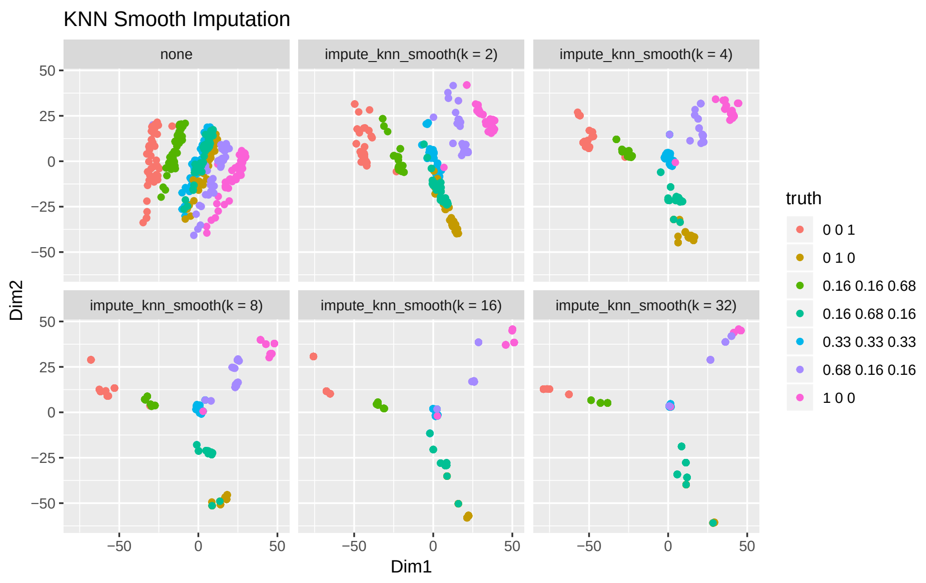
We have seen an application of CellBench to test various parameters values in the knn-smooth algorithm. Multiple steps of methods were applied to the original data and at each step it's possible to store the result and print it in an easy to interpret form. The main complexity is in writing the wrappers, once wrappers for methods are written they are simply placed into lists and applied to data or upstream benchmark_tbls. With the wrappers and annotation function written, we could have compressed this entire example into the following code.

```
res <- data_list %>%
    apply_methods(norm_method) %>%
    apply_methods(impute_method) %>%
    mutate(result = lapply(result, function(x) log2(x + 1))) %>%
    apply_methods(dim_red) %>%
    mutate(result = map2(data, result, append_anno))
## Warning in .get_all_sf_sets(object): spike-in set 'ERCC' should have its own
## size factors

res
## # A tibble: 6 x 5
##   data          norm_method impute_method          dim_red result
##   <fct>         <fct>       <fct>                  <fct>   <list>
## 1 mrna_mix_cel~ scran       none                   pca     <data.frame [340~
## 2 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
## 3 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
## 4 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
## 5 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~
## 6 mrna_mix_cel~ scran       impute_knn_smooth(k =~ pca     <data.frame [340~

plot_df <- tidyr::unnest(res)
plot_df %>%
    ggplot(aes(x = Dim1, y = Dim2, col = truth)) +
    geom_point() +
    facet_wrap(~impute_method, nrow = 2) +
    ggtitle("KNN Smooth Imputation")
```

Use case: Benchmarking method parameters



Then we could easily add new data or methods to the pipeline by modifying our methods lists.