BitPhylogeny: A probabilistic framework for reconstructing intra-tumor phylogenies

Contents

1 Installation

The `BitPhylogeny` packages depends on several `python` and `R` packages. The first step is to make sure the following the packages are installed. * `python`: numpy, scipy, scikit-learn: http://scikit-learn.org/stable/, rpy2: http://rpy.sourceforge.net/. * `R`: rPython, mcclust, e1071, igraph, gplots, riverplot.

Secondly, clone the BitPhylogeny repository

```
git clone git@bitbucket.org:ke_yuan/bitphylogeny.git
cd bitphylogeny
```

The third step is install the BitPhylogeny python package. To do this, navigate into the python directory and run the following

```
cd python
sudo python setup.py install
```

Finally, install the R package

```
cd ../R
R CMD INSTALL bitphylogenyR_0.1.tar.gz
```

2 An example

2.1 BitPhylogeny

We use an example dataset

```
library('bitphylogenyR')
```

```
## Loading required package: rPython
## Loading required package: RJSONIO
## Loading required package: igraph
```

```
example_file <- system.file('sample_data.csv', package='bitphylogenyR')
tmp <- read.csv( example_file )
head(tmp)
```

```
##   V1 V2 V3 V4 V5 V6 V7 V8 V9
## 1  0  0  0  0  0  0  0  0  1
## 2  0  0  0  0  0  0  0  0  1
## 3  0  0  0  0  0  0  0  0  1
## 4  0  0  0  0  0  0  0  0  1
## 5  0  0  0  0  0  0  0  0  1
## 6  0  0  0  0  0  0  0  0  1
```

Note that the last column is set to be the true cluster label of each data point. We separate the data and its label.

```
x <- tmp[,-dim(tmp)[2]]
true_label <- tmp[,dim(tmp)[2]]
```

Run the BitPhylogeny analysis as the following

```
bitphyloR(example_file, './output', T, 200, 50, 5)
```

```
## NULL
```

By default, `bitphyloR` runs with methylation model setting. To analyse mutation data, one can use the `mode` parameter

```
bitphyloR(example_file, './output', T, 200, 50, 5, mode = "mutation")
```

The program saves the results in the directory 'output'.

```
dir('./output', recursive=T)
```

```
##  [1] "sample_data.csv/mcmc-traces/branch_traces.csv"
##  [2] "sample_data.csv/mcmc-traces/label_traces.csv"
##  [3] "sample_data.csv/mcmc-traces/node_depth_traces.csv"
##  [4] "sample_data.csv/mcmc-traces/other_traces.csv"
##  [5] "sample_data.csv/mcmc-traces/params_traces/array_0.npz"
##  [6] "sample_data.csv/mcmc-traces/params_traces/array_1.npz"
##  [7] "sample_data.csv/mcmc-traces/params_traces/array_2.npz"
##  [8] "sample_data.csv/mcmc-traces/params_traces/array_3.npz"
##  [9] "sample_data.csv/mcmc-traces/params_traces/array_4.npz"
## [10] "sample_data.csv/mcmc-traces/params_traces/array_5.npz"
## [11] "sample_data.csv/mcmc-traces/params_traces/array_6.npz"
## [12] "sample_data.csv/mcmc-traces/params_traces/array_7.npz"
## [13] "sample_data.csv/mcmc-traces/params_traces/array_8.npz"
## [14] "sample_data.csv/mcmc-traces/params_traces/array_9.npz"
## [15] "sample_data.csv/mcmc-traces/params_traces.h5"
## [16] "sample_data.csv/mcmc-traces/root_param_traces.csv"
## [17] "sample_data.csv/treescripts/nodes-3.gdl"
## [18] "sample_data.csv/treescripts/nodes-4.gdl"
## [19] "sample_data.csv/treescripts/nodes-5.gdl"
## [20] "sample_data.csv/treescripts/nodes-6.gdl"
## [21] "sample_data.csv/treescripts/tree-freq.csv"
```

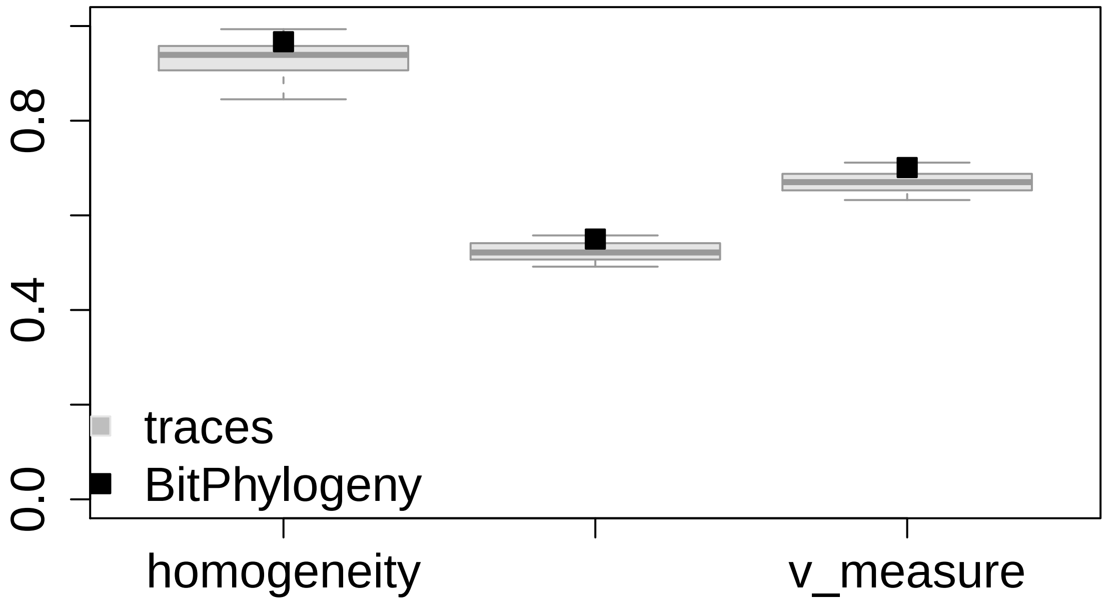
The clustering performance is assessed by the V-measure. In addition, the label trace is summarised by the maximum posterior expected adjusted Rand method.

```r
compute_vmeasures('./output/sample_data.csv',
                  system.file('sample_data.csv', package='bitphylogenyR'))
```

```
## NULL
```

We can visualise the results in the following way

```r
fp <- get_path('./output/sample_data.csv', 'mcmc-traces')
vmeasure_traces <- as.matrix(load_vmeasures(fp, 'vmeasure_traces.csv'))
mpear_vmeasure <- as.matrix(load_vmeasures(fp, 'mpear_label_vmeasure.csv'))
class(vmeasure_traces) <- 'numeric'
class(mpear_vmeasure) <- 'numeric'
par(cex.lab=1.5, cex.axis=1.5)
boxplot(vmeasure_traces, outline=F, cex.main=1.3, ylim=c(0,1),
        border=c('gray60'), col='gray90')
points(c(1,2,3),mpear_vmeasure, pch=22,cex = 1.5, bg= 'black')
colors1 <- c("gray90",'black')
colors2 <- c("gray",'black')
add_legend("bottomleft", legend=c("traces", 'BitPhylogeny'),
           pch=c(22,22), inset = c(0.1,0.20), col=colors1,
           pt.bg=colors2,
           horiz=F, bty='n', cex=1.5)
```

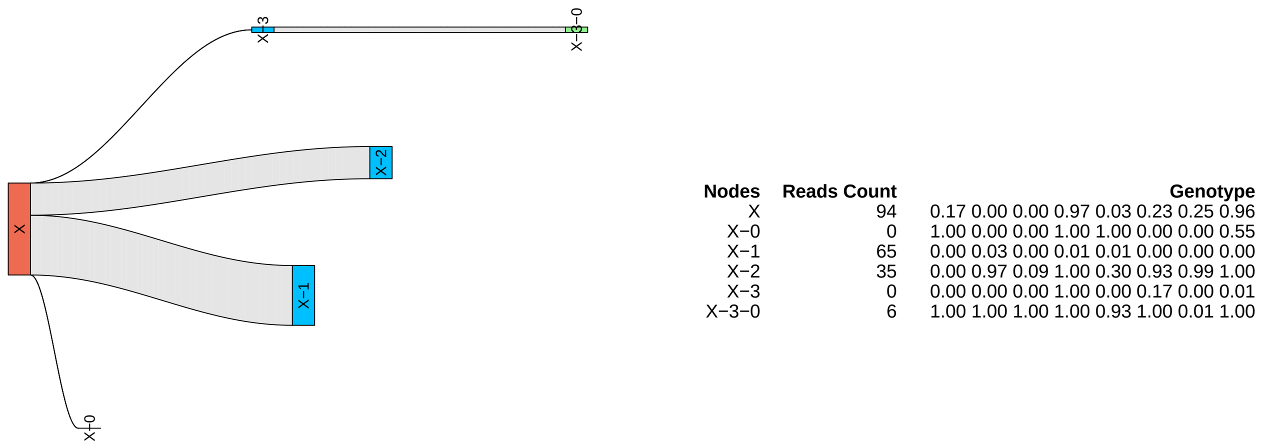


The resulting trees are stored in the `treescripts` directory. The file `tree-freq` contains the appearance frequency of each tree in the folder.

```
treefreq <- read.csv('./output/sample_data.csv/treescripts//tree-freq.csv')
treefreq
```

```
##   unique_node_num  freq
## 1               3 0.325
## 2               4 0.550
## 3               5 0.100
## 4               6 0.025
```

```
plot_sankey_mft('./output/sample_data.csv/treescripts//tree-freq.csv')
```



Nodes	Reads Count	Genotype
X	94	0.17 0.00 0.00 0.97 0.03 0.23 0.25 0.96
X−0	0	1.00 0.00 0.00 1.00 1.00 0.00 0.00 0.55
X−1	65	0.00 0.03 0.00 0.01 0.01 0.00 0.00 0.00
X−2	35	0.00 0.97 0.09 1.00 0.30 0.93 0.99 1.00
X−3	0	0.00 0.00 0.00 1.00 0.00 0.17 0.00 0.01
X−3−0	6	1.00 1.00 1.00 1.00 0.93 1.00 0.01 1.00

Alternatively, the .gdl files can be visualised with the aisee3 software.

2.2 Baseline methods

The baseline methods are consisted of clustering followed by minimum spanning tree construction.

2.2.1 Hierarchical clustering

We first compute the Jacard distance matrix for the sequences.

```
dis <- dist(x, 'binary')
```

Then, we use the R function `hclust` to perform hierarchical clustering based on the previously computed distance matrix.

```
hc <- hclust(dis)
hc
```

```
##
## Call:
## hclust(d = dis)
##
## Cluster method   : complete
## Distance         : binary
## Number of objects: 200
```

The resulting labels of each data point is obtained with the function `cutree`. The function takes the output of `hclust` and the number of desired clusters `k`. In this case, we set `k` to be 7.

```
label <- cutree(hc, k=7)
label
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1
##  [36] 1 1 1 3 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 4 4 4 4
##  [71] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 4 4 4 4 4 4 4 4 4 6 4 4 4 4 4
## [106] 6 4 4 4 4 4 4 4 6 4 4 4 4 4 5 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [141] 4 4 4 4 4 4 4 4 4 4 4 4 7 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 7 7 7
## [176] 7 7 7 6 6 6 6 6 5 5 5 5 5 7 7 7 7 7 7 4 4 7 7 7 7
```

When there is a range of cluster number hypothesis, we compute a list of possible labels.

```
K <- seq(2,14,1)
hc_cand <- lapply(K, function(ii) cutree(hc, ii) )
```

Each of these hypothesis is evaluated by the Silhouette score. The one with the highest score is chosen as the clustering result.

```
library(cluster)
hc_silhouette_res <- sapply(1:length(K),
                              function(ii)
                                summary( silhouette(hc_cand[[ii]] ,dis) )$avg.width )
idx <- which.max( hc_silhouette_res )
hc_label <- hc_cand[[idx]]
hc_label
```

```
##   [1]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##  [24]  1  1  1  1  2  1  1  1  1  1  1  1  1  1  1  3  1  1  1  1  1  1  1
##  [47]  1  3  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  4  4  4  5
##  [70]  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  6  6  6  6  6  7  6  6
##  [93]  6  6  6  6  6  6  6  8  6  6  6  6  6  8  6  6  6  6  6  6  6  9  6
## [116]  6  6  6  6 10  6  6  6  6  6  6  6  4  4  4  4  4  4  4  4  5  5  5
## [139]  5  5  5  5  5  5  5  5  5  5  5  5  5  5 11  5  5  5  5  5  7  7  7
## [162]  7  7  7  7 12  7  7  7  7  7  7 11 11 11 11 11 11  9  9  9  9  9 13
## [185] 13 13 13 13 14 14 14 14 14 14  4  5 11 11 11 14
```

Once the label is computed, we compute the genotype of each cluster as the following

```
clone <- sapply(unique(hc_label), function(i) which(hc_label==i) )
n <- length(clone)
hc_genotype <- matrix(0, n, dim(x)[2])
for (i in 1:n){
    idx <- clone[[i]]
    if ( length(idx)==1 ){
      hc_genotype[i,] <- as.matrix(x[idx,])
    }else{
      hc_genotype[i,] <- as.numeric( colMeans(as.matrix(x[idx,])) > 0.5 )
    }
}
hc_genotype
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
##  [1,]    0    0    0    0    0    0    0    0
##  [2,]    0    0    0    0    1    0    0    0
##  [3,]    0    1    0    0    0    0    0    0
##  [4,]    0    0    0    1    0    0    1    1
##  [5,]    0    1    0    1    0    1    1    1
##  [6,]    0    0    0    1    0    0    0    1
##  [7,]    0    0    0    1    0    1    0    1
##  [8,]    0    0    0    1    0    0    0    0
##  [9,]    1    0    0    1    0    0    0    1
## [10,]    0    0    0    0    0    0    0    1
## [11,]    0    1    0    1    1    1    1    1
## [12,]    0    0    0    0    0    1    0    1
## [13,]    0    0    0    1    1    1    0    1
## [14,]    1    1    1    1    1    1    0    1
```

Finally, we put the above steps into a function which gives the label and genotype estimates.

```
get_label_hc
```

```
## function (x, K)
## {
##     dis <- dist(x, "binary")
##     hc_cand <- lapply(K, function(ii) cutree(hclust(dis), ii))
##     hc_silhouette_res <- sapply(1:length(K), function(ii) summary(silhouette(hc_cand[[ii]],
##         dis))$avg.width)
##     idx <- which.max(hc_silhouette_res)
##     hc_label <- hc_cand[[idx]]
##     clone <- sapply(unique(hc_label), function(i) which(hc_label ==
##         i))
##     n <- length(clone)
##     genotype <- matrix(0, n, dim(x)[2])
##     for (i in 1:n) {
##         idx <- clone[[i]]
##         if (length(idx) == 1) {
##             genotype[i, ] = as.matrix(x[idx, ])
##         }
##         else {
##             genotype[i, ] = as.numeric(colMeans(as.matrix(x[idx,
##                 ])) > 0.5)
##         }
##     }
##     return(list(hc_label = hc_label, hc_genotype = genotype))
## }
## <environment: namespace:bitphylogenyR>
```

2.2.2 K-centroids clustering

The k-centroids methods uses the same distance matrix compute above.

```
kc = pam(dis, 7)
kc
```

```
## Medoids:
##       ID
## [1,]  65  65
## [2,] 194 194
## [3,] 158 158
## [4,] 195 195
## [5,] 127 127
## [6,] 172 172
## [7,] 135 135
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1
##  [36] 1 1 1 3 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 4 4 3 4
##  [71] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 6 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [106] 5 5 5 5 5 5 5 5 7 5 5 5 5 5 5 5 5 5 5 5 5 5 7 7 7 7 7 7 7 7 3 3 3 3 3
## [141] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 6 6 6 6 6 6 6 6 6 6 6 6 3 3 3
## [176] 3 3 3 7 7 7 7 7 6 6 6 6 6 2 2 2 2 2 2 4 3 2 2 2 2
## Objective function:
##   build    swap
## 0.05225 0.05225
##
## Available components:
## [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"
## [6] "clusinfo"   "silinfo"    "diss"       "call"
```

In this case, the genotypes can be obtained as the metroids. The Silhouette score is used to choose the number of clusters.

```
kc_cand <- lapply(K, function(ii) pam( dis, ii) )
kc_silhouette_res <- sapply(1:length(K), function(ii)
                            summary( silhouette(kc_cand[[ii]]$clustering,dis) )$avg.width )
idx <- which.max( kc_silhouette_res )

kc_label <- kc_cand[[idx]]$clustering
kc_label
```

```
##   [1]  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
##  [24]  1  1  1  1  2  1  1  1  1  1  1  1  1  1  1  3  1  1  1  1  1  1  1
##  [47]  1  3  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  4  4  4  5
##  [70]  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  6  6  6  6  6  7  6  6
##  [93]  6  6  6  6  6  6  6  8  6  6  6  6  6  8  6  6  6  6  6  6  6  9  6
## [116]  6  6  6  6 10  6  6  6  6  6  6  6 11 11 11 11 11 11 11 11  5  5  5
## [139]  5  5  5  5  5  5  5  5  5  5  5  5  5  5 12  5  5  5  5  5  5  7  7
## [162]  7  7  7  7  7  7  7  7  7  7  7 12 12 12 12 12 12  9  9  9  9  9 13
## [185] 13 13 13 13 14 14 14 14 14 14  4  5 12 12 12 14
```

```
kc_genotype <- x[kc_cand[[idx]]$medoids,]
kc_genotype
```

```
##     V1 V2 V3 V4 V5 V6 V7 V8
## 65   0  0  0  0  0  0  0  0
## 28   0  0  0  0  1  0  0  0
## 48   0  1  0  0  0  0  0  0
## 195  0  0  0  1  0  0  1  1
## 158  0  1  0  1  0  1  1  1
## 127  0  0  0  1  0  0  0  1
## 172  0  0  0  1  0  1  0  1
## 106  0  0  0  1  0  0  0  0
## 183  1  0  0  1  0  0  0  1
## 120  0  0  0  0  0  0  0  1
## 135  1  0  0  1  0  0  1  1
## 178  0  1  0  1  1  1  1  1
## 188  0  0  0  1  1  1  0  1
## 194  1  1  1  1  1  1  0  1
```

We also wrapped up a function for k-centroids clustering.

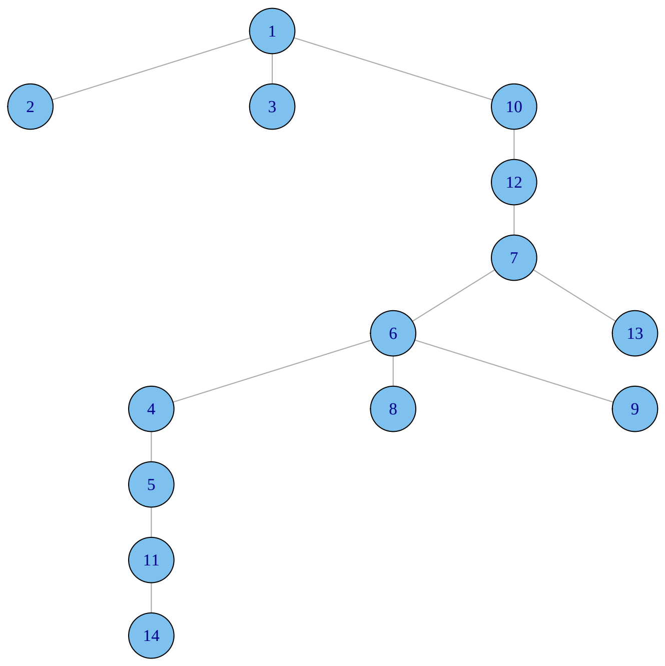
```
get_label_kc
```

```
## function (x, K)
## {
##     dis <- dist(x, "binary")
##     kc_cand <- lapply(K, function(ii) pam(dis, ii))
##     kc_silhouette_res <- sapply(1:length(K), function(ii) summary(silhouette(kc_cand[[ii]]$clusterin
##         dis))$avg.width)
##     idx <- which.max(kc_silhouette_res)
##     kc_label <- kc_cand[[idx]]$clustering
##     kc_genotype <- x[kc_cand[[idx]]$medoids, ]
##     return(list(kc_label = kc_label, kc_genotype = kc_genotype))
## }
## <environment: namespace:bitphylogenyR>
```

2.2.3 Tree building

We construct the minimum spanning tree based the clustering results from the previous stage.

```
mst <- get_mst(hc_genotype)
plot_mst(hc_genotype, hc_label, flag=F)
```



Nodes	Read Counts	Genotype
1	62	0 0 0 0 0 0 0 0
2	1	0 0 0 0 1 0 0 0
3	2	0 1 0 0 0 0 0 0
4	27	0 0 0 1 0 0 1 1
5	24	0 1 0 1 0 1 1 1
6	38	0 0 0 1 0 0 0 1
7	14	0 0 0 1 0 1 0 1
8	2	0 0 0 1 0 0 0 0
9	6	1 0 0 1 0 0 0 1
10	1	0 0 0 0 0 0 0 1
11	10	0 1 0 1 1 1 1 1
12	1	0 0 0 0 0 1 0 1
13	5	0 0 0 1 1 1 0 1
14	7	1 1 1 1 1 1 0 1

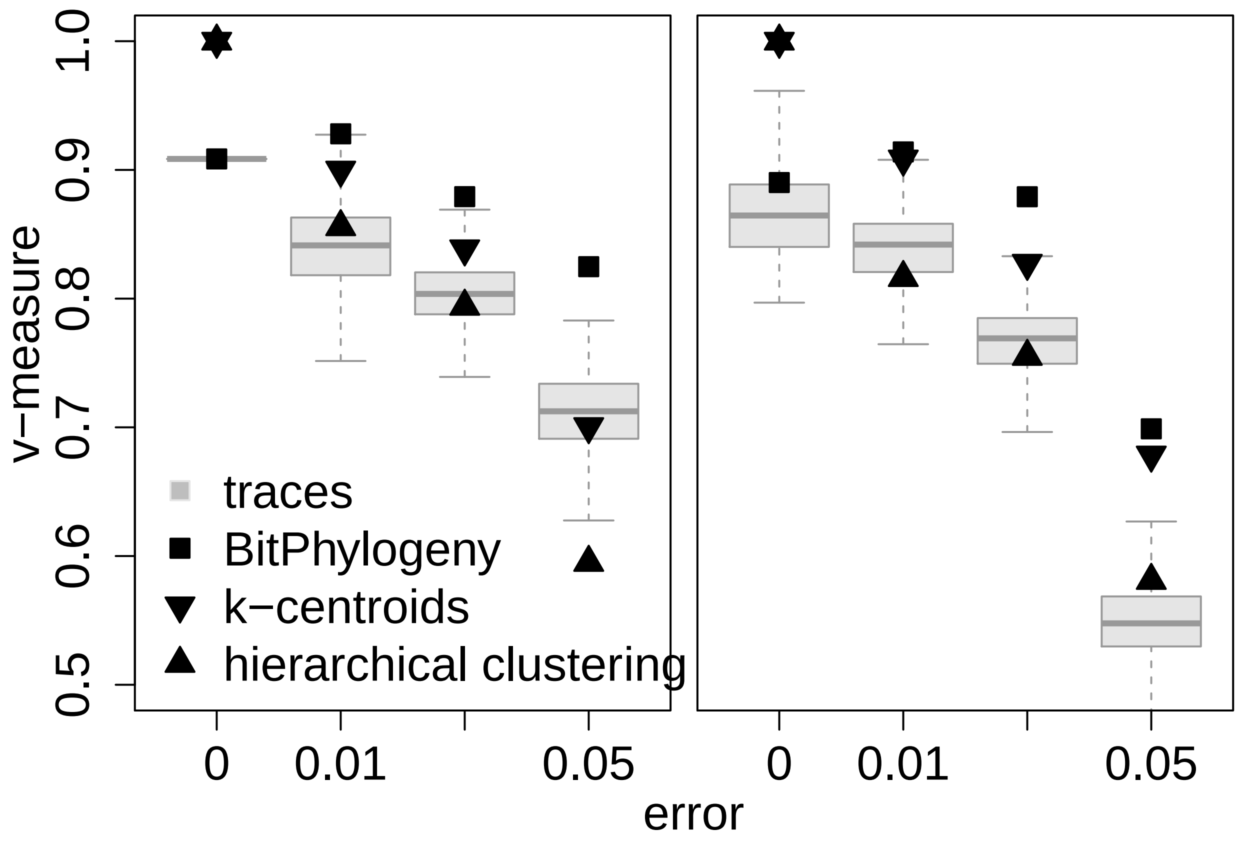
3 Reproduce figure 3B

```r
data(saved_vmeasures)

mcmc_vmeasures <- saved_vmeasures$mcmc_vmeasures
hc_vmeasures <- saved_vmeasures$hc_vmeasures
kc_vmeasures <- saved_vmeasures$kc_vmeasures
mpear_vmeasures <- saved_vmeasures$mpear_vmeasures

par(mfrow=c(1,2), oma = c(3,3,0,0) + 0.1,
    mar = c(0,0,1,0.5) + 0.1, cex.lab=1.5, cex.axis=1.5)
boxplot(mcmc_vmeasures$big_clone, outline=F,
        ylim=c(0.5,1) ,
        cex.main=1.3,
        border=c('gray60'), col='gray90')
points( mpear_vmeasures$big_clone, pch=22,cex = 1.5, bg= 'black')
points( hc_vmeasures$big_clone, pch=24,cex = 1.5, bg= 'black')
points( kc_vmeasures$big_clone, pch=25,cex = 1.5, bg= 'black')

boxplot(mcmc_vmeasures$small_clone, outline=F,
        ylim=c(0.5,1) ,
        yaxt='n',cex.main=1.3,
        border=c('gray60'), col='gray90')
points( mpear_vmeasures$small_clone, pch=22, cex = 1.5, bg= 'black')
points( hc_vmeasures$small_clone, pch=24, cex=1.5, bg= 'black')
points( kc_vmeasures$small_clone, pch=25, cex = 1.5, bg= 'black')
colors1 <- c("gray90",'black', 'black', "black")
colors2 <- c("gray",'black', 'black', "black")
add_legend("bottomleft", legend=c("traces", 'BitPhylogeny',
                                  'k-centroids',
                                   'hierarchical clustering'),
           pch=c(22,22,25,24), inset = c(0.1,0.13), col=colors1,
           pt.bg=colors2,
           horiz=F, bty='n', cex=1.5)
title(xlab = "error",
      ylab = "v-measure",
      outer = TRUE, line = 2.2)
```



4 Reproduce figure 3C

```
big_clone_t <- c(3, 7)
big_clone_bit <- c(2, 5)
big_clone_hc <- c(4, 7)
big_clone_kc <- c(4, 7)
big_clone_bit <- rbind(big_clone_bit,
                      c(3, 7), c(2,9 ), c(2,7) )
big_clone_hc <- rbind(big_clone_hc,
                     c(5, 19), c(5,20), c(6,20))
big_clone_kc <- rbind(big_clone_kc,
                     c(5, 20), c(5, 20), c(5,20))

small_clone_t <- c(4, 12)
small_clone_bit <- c(2, 8)
small_clone_hc <- c(5, 12)
```

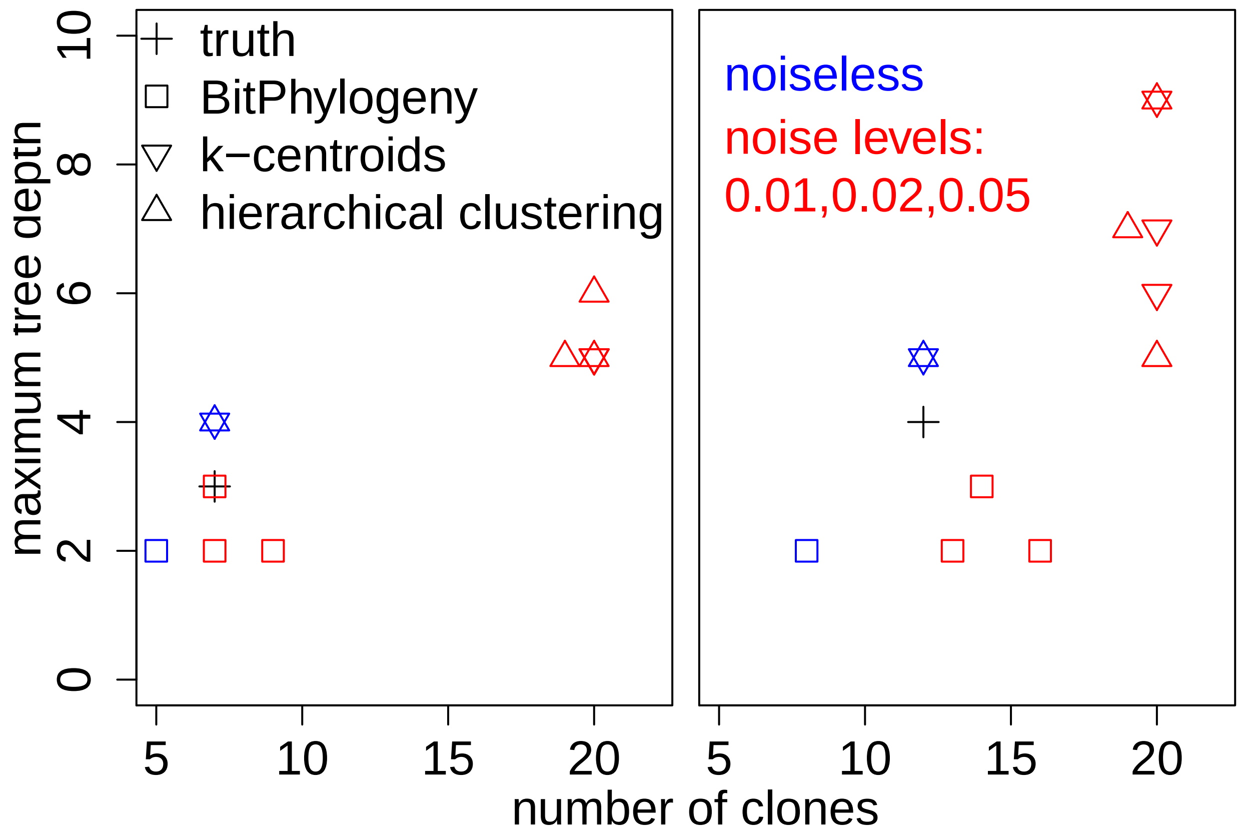
```r
small_clone_kc <- c(5, 12)

small_clone_bit <- rbind(small_clone_bit,
                         c(3, 14),c(2,16 ), c(2,13) )
small_clone_hc <- rbind(small_clone_hc,
                        c(7, 19),c(5,20), c(9,20))
small_clone_kc <- rbind(small_clone_kc,
                        c(9, 20),c(7, 20), c(6,20))

par(mfrow=c(1,2), oma = c(3,3,0,0) + 0.1,
    mar = c(0,0,1,0.5) + 0.1, cex.lab=1.5, cex.axis=1.5)
color <- c('blue', 'red', 'red', 'red')
plot(big_clone_t[2], big_clone_t[1],pch=3, ylim=c(0,10) ,xlim= c(5,22)
     ,cex=1.5)
points(big_clone_bit[,2], big_clone_bit[,1],pch=0,cex=1.5, col=color)
points(big_clone_hc[,2], big_clone_hc[,1],pch=2,cex=1.5, col=color)
points(big_clone_kc[,2], big_clone_kc[,1],pch=6,cex=1.5, col=color)

plot(small_clone_t[2], small_clone_t[1],pch=3, ylim=c(0,10) ,xlim= c(5,22),cex=1.5,yaxt='n')
points(small_clone_bit[,2], small_clone_bit[,1],pch=0,cex=1.5, col=color)
points(small_clone_hc[,2], small_clone_hc[,1],pch=2,cex=1.5, col=color)
points(small_clone_kc[,2], small_clone_kc[,1],pch=6,cex=1.5, col=color)

colors1 <- c("black",'black', 'black', "black")
colors2 <- c("black",'black', 'black', "black")
add_legend("topleft", legend=c("truth", 'BitPhylogeny',
                               'k-centroids',
                               'hierarchical clustering'),
           pch=c(3,0,6,2), inset = c(0.08,0.02), col=colors1,
           pt.bg=colors2,
           horiz=F, bty='n', cex=1.5)
add_legend("topleft", legend=c("noiseless", 'noise levels: \n0.01,0.02,0.05'),
           inset = c(0.50,0.02), text.col=c('blue','red'),
           horiz=F, bty='n', cex=1.5)
title(xlab = "number of clones",
      ylab = "maximum tree depth",
      outer = TRUE, line = 2.2)
```


truth
BitPhylogeny
k−centroids
hierarchical clustering
noiseless
noise levels:
0.01,0.02,0.05
maximum tree depth
number of clones
0
2
4
6
8
10
5
10
15
20