Assessing Tumor Microsatellite Instability from Tumor Exome Somatic Mutations—The MSIseq Package

Mini Huang

June 11, 2015

The MSIseq package provides a mechanism for detecting microsatellite instability (MSI) in somatic mutation data from whole exome sequencing. The package provides both a classifier (detector), as well as a means to train new classifiers, which may be necessary depending on changes in variant-calling algorithms.

This package contains two main functions. The function `MSIseq.train()` generates a detector for MSI status from training data that consists of somatic mutation information and MSI status. The function `MSIseq.classify()` uses the generated detector to classify the MSI status of new tumors. The package also provides a helper function, `Compute.input.variables()`, to generate the input needed by these two functions.

1 Input data

As input, the MSIseq package requires somatic mutation information (i.e. from The Cancer Genome Atlas (TCGA) website, `https://tcga-data.nci.nih.gov/tcga/`) in the format of a "mutation annotation file" (`https://wiki.nci.nih.gov/display/tcga/File+Format+Specifications`). For example, `NGStraindata` and `NGStestdata` are in this format.

```
> library('MSIseq')
> data(NGStraindata)
> data(NGStestdata)
> head(NGStraindata)
```

```
  Chrom Start_Position End_Position Variant_Type Tumor_Sample_Barcode
1 chr19       58862932     58862932          SNP         TCGA-D1-A15Z
3 chr10       52575855     52575856          INS         TCGA-BG-A0M0
4 chr10       52575855     52575856          INS         TCGA-BG-A0M3
5 chr10       52575855     52575856          INS         TCGA-BG-A0M9
6 chr12        9229467      9229467          SNP         TCGA-D1-A16B
7 chr12        9229527      9229527          SNP         TCGA-BG-A0M4
```

Usually the TCGA mutation annotation file contains 37 columns. The `NGStraindata` only contains the 5 columns that are necessary for this package and the `Compute.input.variables()` function. Any other columns are ignored. The names of the 5 columns must be exactly as shown.

In the 5 columns, Chrom indicates the chromosome identifier. Start_Position and End_Position are the start and end positions of the mutation in the chromosome. Variant_Type indicates the type of variant, for which the legal values are "SNP", "INS" and "DEL". Other values will cause an error. Tumor_Sample_Barcode is the sample ID.

To obtain such a somatic mutation information table for your own data, you will need to create it from your sequence alignments and suitable annotation.

Another information that MSIseq package needs is the sequence length, which is the total length of the genomic regions from a DNA sample captured by sequencing techniques. For example, `NGStrainseqLen` and `NGStestseqLen` contain the information.

```
> data(NGStrainseqLen)
> data(NGStestseqLen)
> head(NGStrainseqLen)
```

```
  Tumor_Sample_Barcode Sequence_Length
1         TCGA-CM-6677              44
2         TCGA-CA-6717              44
3         TCGA-AZ-4315              44
4         TCGA-D5-6531              44
5         TCGA-CM-6162              44
6         TCGA-AA-3663              44
```

The sequence length table contains 2 columns, Tumor_Sample_Barcode and Sequence_Length, which indicate the sample IDs and their corresponding sequence lengths in megabases.

The genomic coordinates of simple sequence repeats in the reference genome are also required by MSIseq. The sequence repeats table contains 3 columns. Chrom indicates the chromosome identifier. Start_Position and End_Position are the start and end positions.

For example, we can get the simple sequence repeats in human genome (version Hg19) from the following link:

```
> url <-
+ "http://steverozen.net/data/Hg19repeats.rda"
> file <- basename(url)
> download.file(url, file)
> load("Hg19repeats.rda")
> head(Hg19repeats)
```

```
  Chrom Start_Position End_Position
1 chr10          60213        60217
```

```
2 chr10          60287         60291
3 chr10          60379         60383
4 chr10          60518         60523
5 chr10          60741         60745
6 chr10          60898         60902
```

Hg19repeats is a static dataset which can be used for sequencing data generated with the same reference genome. You can also get it from `http://hgdownload.cse.ucsc.edu/goldenpath/hg19/database/` for other reference genome.

Hg19repeats is a bed format data containing all the simple repeat regions (mono-, di-, tri-, tetra-nucleotide repeats) in the human genome version hg19. In this data, di-nucleotide, tri-nucleotide and tetra-nucleotide repeats are from the table in UCSC Genome Bioinformatics Site: `ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/simpleRepeat.txt.gz`. Mono-nucleotide repeats with a length ≥ 5 are generated with the following two functions, `find.mono.repeats` and `find.mono.repeats` against the human hg19 genome.

Here is an example of how one can generate the mono-nucleotide repeats in one chromosome:

```
> ## download the chromosome 20 sequence from UCSC
> url2 <-
+ "ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/chr20.fa.gz"
> file <- basename(url2)
> download.file(url2, file)
> gunzip(file)
> file <- 'chr20.fa'
> ## generate mono-nucleotide repeats regions from chromosome 20
> data.chr20 = read.fasta(file)
> mono.repeats.chr20 = find.mono.repeats(data.chr20)
> names(mono.repeats.chr20)<-c('Chrom', 'Start_Position', 'End_Position')
```

Users can easily apply these functions to their own fastq file and generate their own repeats file.

The MSI status information is required by `MSIseq.train()` function specifically. If your want to train a classifier with your own data, you need to have a classification table showing the clinical test result of MSI status of your samples.

```
> data(NGStrainclass)
> head(NGStrainclass)

   Tumor_Sample_Barcode MSI_status
1          TCGA-A6-5661      MSI-H
2          TCGA-A6-5665      MSI-H
3          TCGA-A6-6653      MSI-H
4          TCGA-A6-6781      MSI-H
5          TCGA-AA-3492      MSI-H
10         TCGA-AA-3663      MSI-H
```

In the classification table, Tumor_Sample_Barcode represents the sample ID. MSI_status is a factor with two levels, "MSI-H" and "Non-MSI-H". Other values will cause an error.

The cancer type information is optional. If you would like to use cancer type as a candidate input for your classifier, you need to have a cancer type table.

```
> data(NGStraintype)
> data(NGStesttype)
> head(NGStraintype)

  Tumor_Sample_Barcode cancer_type
1         TCGA-D1-A15Z endometrial
2         TCGA-D1-A17D endometrial
3         TCGA-BG-A0MQ endometrial
4         TCGA-BS-A0U9 endometrial
5         TCGA-D1-A16D endometrial
6         TCGA-BG-A0M7 endometrial
```

In the cancer type table, Tumor_Sample_Barcode represents the sample ID. cancer_type is a factor which gives the corresponding cancer types.

2 Functions

First, the helper function `Compute.input.variables()` is used to generate the input needed by the two main functions.

```
> train.mutationNum<-Compute.input.variables(NGStraindata,
+           repeats = Hg19repeats, seq.len = NGStrainseqLen)
```

This function takes four arguments: Compute.input.variables(data, repeats, uniform.seq.len = 38, seq.len = NULL). The formats for data, repeats, and seq.len are explained in the Input data section. And the default seq.len argument is 38. This argument is used when sequences for all samples have the same length.

This function computes and extracts mutation count information from the argument data. The variable sequence length is used as a denominator to generate mutation count per megabase. The mutation can be either a single nucleotide substitution (SNS) or a short insertion/deletion (indel).

The returned value is a data frame containing the following 9 variables:

- T.sns: total count of SNSs/Mb
- S.sns: count of SNSs in simple sequence repeats/Mb
- T.ind: total count of indels/Mb
- S.ind: count of indels in simple sequence repeats/Mb

- T: total mutation count/Mb
- S: mutation count in simple sequence repeats/Mb
- Ratio.sns: S.sns/T.sns
- Ratio.ind: S.ind/T.ind
- Ratio: S/T

Now let's look at the two main functions.

```
> sampleclassifier<-MSIseq.train(mutationNum = train.mutationNum,
+     classification=NGStrainclass, cancerType = NGStraintype)
```

```
5 fold cross validation result: 98.61496
```

The function `MSIseq.train()` takes three arguments: MSIseq.train(mutationNum, classification, cancerType = NULL). The format of mutationNum should be the same as the returned value of the helper function `Compute.input.variables()`. The format for classification and cancerType are explained in the Input data section. Again, the cancerType argument is optional. It depends on whether you want to train your classifier with cancer type information.

This function uses the 'RWeka' package to build and evaluate a J48 decision tree with the 9 variables (or 10 variables including 'cancer type'). The function will also give a five-fold cross validation result for the classification accuracy of the model.

The return value for `MSIseq.train()` is a Weka_classifier object, a J48 decision tree classifier.

```
> sampleclassifier
```

```
J48 pruned tree
------------------

S.ind <= 0.394737: Non-MSI-H (295.0/3.0)
S.ind > 0.394737: MSI-H (66.0)

Number of Leaves  :         2

Size of the tree :         3
```

The two output classses of the decision tree classifier are MSI-H and Non-MSI-H. 3 variables (S.ind, T.sns, S) are chosen to build the decision tree. When training with other data, you will get a different decision tree.

In this sample classifier, the desicion tree is based on a single variable, S.ind. If S.ind > 0.395, the tumor is classified as MSI-H. Otherwise, the classification is non-MSI-H. This classifier is also provided by MSIseq named as `NGSclassifier`. And it is offered as the default classifier for the second function, `MSIseq.classify()`.

The function `MSIseq.classify()` classifies tumors with unknown MSI status.

```
> test.mutationNum<-Compute.input.variables(NGStestdata,
+                 repeats = Hg19repeats, seq.len = NGStestseqLen)
> result <- MSIseq.classify(mutationNum = test.mutationNum,
+                 cancerType = NGStesttype)
```

This function takes three arguments: MSIseq.classify(mutationNum, classifier, cancerType = NULL). The format of mutationNum should be the same as the returned value of the helper function `Compute.input.variables()`. The default classifier is a built-in classifier, NGSclassifier. You can also use your own classifier, which should be a returned value from the function `MSIseq.train()`. Remember if the input classifier is trained with cancerType argument, you should also give cancer type information in this function. And the format of cancerType should be the same as mentioned before.

```
> head(result)

  Tumor_Sample_Barcode MSI_status Likely_POLE_deficiency
1         TCGA-A5-A0GE  Non-MSI-H                     No
2         TCGA-D1-A176      MSI-H                     No
3         TCGA-BG-A0W1  Non-MSI-H                     No
4         TCGA-E0-A1Y5  Non-MSI-H                     No
5         TCGA-D1-A17U      MSI-H                     No
6         TCGA-AX-A064      MSI-H                     No
```

The return value for `MSIseq.classify()` is a data frame with three columns. The first two columns are Tumor_Sample_Barcode and the corresponding classified MSI_status. The third column indicates whether the sample is likely POLE deficient based on the criteria of T.sns > 60/Mb and S.ind < 0.18/Mb.