VariantAnnotation

Valerie Obenchain [*]

Fred Hutchinson Cancer Research Center
Seattle, WA

24 July 2012

Abstract

This lab provides an overview of the *VariantAnnotation* package. Variants from a VCF file are read into `R` and stored in `VCF`-class object. We explore the class and see how the data are organized and accessed. Variants are then classified by region such as 'intronic', 'coding', '3UTR' etc. Coding variants are further analyzed for amino acid coding changes. Predictions as to how damaging these coding changes may be are determined by querying the SIFT database package.

1 Annotation of Variants

A major product of DNASeq experiments are catalogs of called variants (e.g., SNPs, indels). We will use the *VariantAnnotation* package to explore this type of data. Sample data included in the package are a subset of chromosome 22 from the 1000 Genomes project. Variant Call Format (VCF; full description) text files contain meta-information lines, a header line with column names, data lines with information about a position in the genome, and optional genotype information on samples for each position.

1.1 Variant call format (VCF) files

Data are read from a VCF file and variants identified according to region such as `coding`, `intron`, `intergenic`, `spliceSite` etc. Amino acid coding changes are computed for the non-synonymous variants. SIFT and PolyPhen databases provide predictions of how severely the coding changes affect protein function.

Data exploration

Exercise 1
The objective of this exercise is to compare the quality of called SNPs that are located in dbSNP, versus those that are novel.

[*] vobencha@fhcrc.org

Locate the sample data in the file system. Explore the metadata (information about the content of the file) using `scanVcfHeader`. Discover the 'info' fields `VT` (variant type), and `RSQ` (genotype imputation quality).

Input sample data in using `readVcf`. You'll need to specify the genome build (`genome="hg19"`) on which the variants are annotated. Take a peak at the `rowData` to see the genomic locations of each variant.

dbSNP uses abbreviations such as `ch22` to represent chromosome 22, whereas the VCF file uses 22. Use `rowData` and `renameSeqlevels` to extract the row data of the variants, and rename the chromosomes.

The SNPlocs.Hsapiens.dbSNP.20101109 contains information about SNPs in a particular build of dbSNP. Load the package, use the `dbSNPFilter` function to create a filter, and query the row data of the VCF file for membership.

Create a data frame containing the dbSNP membership status and imputation quality of each SNP. Create a density plot to illustrate the results.

Solution: Explore the header:

```
> library(VariantAnnotation)
> fl <- system.file("extdata", "chr22.vcf.gz",
+                   package="VariantAnnotation")
> (hdr <- scanVcfHeader(fl))

class: VCFHeader
samples(5): HG00096 HG00097 HG00099 HG00100 HG00101
meta(1): fileformat
fixed(1): ALT
info(22): LDAF AVGPOST ... VT SNPSOURCE
geno(3): GT DS GL

> info(hdr)[c("VT", "RSQ"),]

DataFrame with 2 rows and 3 columns
         Number        Type                                       Description
    <character> <character>                                       <character>
VT            1      String indicates what type of variant the line represents
RSQ           1       Float      Genotype imputation quality from MaCH/Thunder
```

Input the data and peak at their locations:

```
> vcf <- readVcf(fl, "hg19")
> head(rowData(vcf), 3)

GRanges with 3 ranges and 1 elementMetadata col:
              seqnames               ranges strand | paramRangeID
                 <Rle>            <IRanges>  <Rle> |     <factor>
    rs7410291       22 [50300078, 50300078]      * |         <NA>
  rs147922003       22 [50300086, 50300086]      * |         <NA>
  rs114143073       22 [50300101, 50300101]      * |         <NA>
  ---
  seqlengths:
   22
   NA
```

Rename chromosome levels:

```
> rowData(vcf) <- renameSeqlevels(rowData(vcf), c("22"="ch22"))
```

Discover whether SNPs are located in dbSNP:

```
> library(SNPlocs.Hsapiens.dbSNP.20101109)
> snpFilt <- dbSNPFilter("SNPlocs.Hsapiens.dbSNP.20101109")
> inDbSNP <- snpFilt(rowData(vcf), subset=FALSE)
> table(inDbSNP)
```

```
inDbSNP
FALSE  TRUE
 6126  4250
```

Create a data frame summarizing SNP quality and dbSNP membership:

```
> metrics <-
+     data.frame(inDbSNP=inDbSNP, RSQ=values(info(vcf))$RSQ)
```

Finally, visualize the data, e.g., using *ggplot2* (Figure 1).

```
> library(ggplot2)
> ggplot(metrics, aes(RSQ, fill=inDbSNP)) +
+     geom_density(alpha=0.5) +
+     scale_x_continuous(name="MaCH / Thunder Imputation Quality") +
+     scale_y_continuous(name="Density") +
+     opts(legend.position="top")
```

1.2 Coding consequences

Locating variants in and around genes Variant location with respect to genes can be identified with the `locateVariants` function. Regions are specified in the `region` argument and can be one of the following constructors: `CodingVariants()`, `IntronVariants()`, `FiveUTRVariants()`, `ThreeUTRVariants()`, `IntergenicVariants()`, `SpliceSiteVariants()`, or `AllVariants()`. Location definitions are shown in Table 1.

Exercise 2
Load the TxDb.Hsapiens.UCSC.hg19.ensGene annotation with the `loadDb` *function from GenomicFeatures. The annotation file is located at "/home/valerie/VariantAnnotation/". Read in the* `chr22.vcf.gz` *example file from the VariantAnnotation package.*

Remembering to re-name sequence levels, use the `locateVariants` *function to identify coding variants.*

Summarize aspects of your data, e.g., did any coding variants match more than one gene? How many coding variants are there per gene ID?

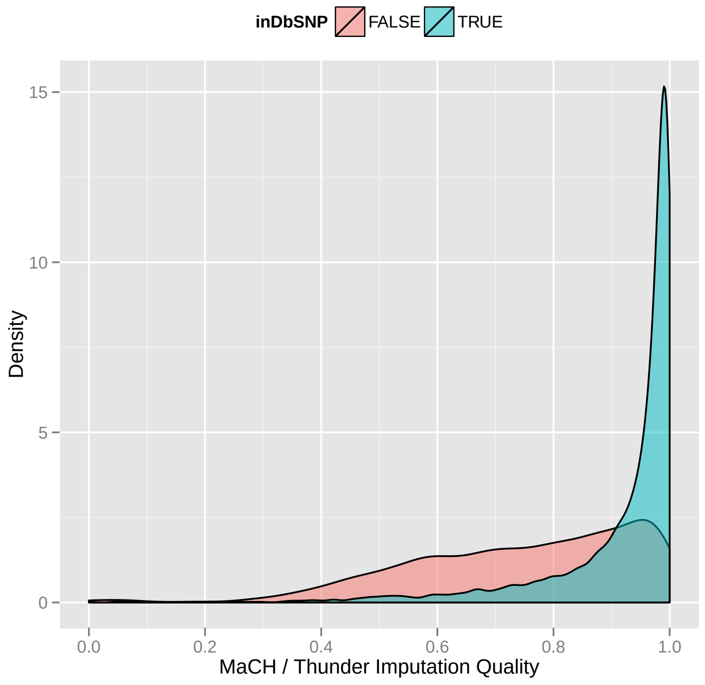

Figure 1: Quality scores of variants in dbSNP, compared to those not in dbSNP.

Table 1: Variant locations

Location	Details
`coding`	Within a coding region
`fiveUTR`	Within a 5' untranslated region
`threeUTR`	Within a 3' untranslated region
`intron`	Within an intron region
`intergenic`	Not within a transcript associated with a gene
`spliceSite`	Overlaps any of the first or last 2 nucleotides of an intron

Solution: The `TxDb.Hsapiens.UCSC.hg19.ensGene.sqlite` file contains annotations for the Ensembl gene model. Load the TranscripDb,

```
> library(GenomicFeatures) # for loadDb
> txdb <-
+   loadDb("/home/valerie/VariantAnnotation/TxDb.Hsapiens.UCSC.hg19.ensGene.sqlite")
```

and read in the VCF file.

```
> fl <- system.file("extdata", "chr22.vcf.gz",
+                   package="VariantAnnotation")
> vcf <- readVcf(fl, "hg19")
> vcf <- renameSeqlevels(vcf, c("22"="chr22"))
```

The next lines locate coding variants.

```
> rd <- rowData(vcf)
> loc <- locateVariants(rd, txdb, CodingVariants())
> head(loc, 3)
```

```
GRanges with 3 ranges and 5 elementMetadata cols:
      seqnames               ranges strand | LOCATION   QUERYID      TXID
         <Rle>            <IRanges>  <Rle> | <factor> <integer> <integer>
  [1]    chr22 [50301422, 50301422]      * |   coding        24    165767
  [2]    chr22 [50301422, 50301422]      * |   coding        24    165768
  [3]    chr22 [50301422, 50301422]      * |   coding        24    165769
          CDSID          GENEID
      <integer>     <character>
  [1]    256696 ENSG00000182858
  [2]    256695 ENSG00000182858
  [3]    256696 ENSG00000182858
  ---
  seqlengths:
   chr22
      NA
```

To answer gene-centric questions data can be summarized by gene regardless of transcript.

```
> ## Did any coding variants match more than one gene?
> splt <- split(values(loc)$GENEID, values(loc)$QUERYID)
> table(sapply(splt, function(x) length(unique(x)) > 1))
```

```
FALSE
 1026
```

```
> ## Summarize the number of coding variants by gene ID
> splt <- split(values(loc)$QUERYID, values(loc)$GENEID)
> head(sapply(splt, function(x) length(unique(x))), 3)
```

```
ENSG00000025708 ENSG00000025770 ENSG00000073146
             13              48              82
```

Amino acid coding changes `predictCoding` computes amino acid coding changes for non-synonymous variants. Only ranges in `query` that overlap with a coding region in `subject` are considered. Reference sequences are retrieved from either a `BSgenome` or fasta file specified in `seqSource`. Variant sequences are constructed by substituting, inserting or deleting values in the `varAllele` column into the reference sequence. Amino acid codes are computed for the variant codon sequence when the length is a multiple of 3.

The `query` argument to `predictCoding` can be a `GRanges` or `VCF`. When a `GRanges` is supplied the `varAllele` argument must be specified. In the case of a `VCF`, the alternate alleles are taken from `values(alt(<VCF>))$ALT` and the `varAllele` argument is not specified.

The result is a modified `query` containing only variants that fall within coding regions. Each row represents a variant-transcript match so more than one row per original variant is possible.

```
> library(BSgenome.Hsapiens.UCSC.hg19)
> coding <- predictCoding(vcf, txdb, seqSource=Hsapiens)
> coding[5:9]
```

```
GRanges with 5 ranges and 13 elementMetadata cols:
                seqnames               ranges strand | paramRangeID
                   <Rle>            <IRanges>  <Rle> |     <factor>
      rs8135963    chr22 [50301476, 50301476]      - |         <NA>
    rs8135963.1    chr22 [50301476, 50301476]      - |         <NA>
     22:50301488   chr22 [50301488, 50301488]      - |         <NA>
   22:50301488.1   chr22 [50301488, 50301488]      - |         <NA>
   22:50301488.2   chr22 [50301488, 50301488]      - |         <NA>
                   varAllele     CDSLOC             PROTEINLOC   QUERYID
                <DNAStringSet>  <IRanges> <CompressedIntegerList> <integer>
      rs8135963              G [ 91,  91]                      31        25
    rs8135963.1              G [416, 416]                     139        25
     22:50301488             A [873, 873]                     291        26
   22:50301488.1             A [ 79,  79]                      27        26
   22:50301488.2             A [404, 404]                     135        26
                       TXID     CDSID          GENEID   CONSEQUENCE
                <character> <integer>     <character>      <factor>
      rs8135963      165768    256695 ENSG00000182858 nonsynonymous
    rs8135963.1      165769    256696 ENSG00000182858 nonsynonymous
     22:50301488     165767    256696 ENSG00000182858    synonymous
   22:50301488.1     165768    256695 ENSG00000182858 nonsynonymous
   22:50301488.2     165769    256696 ENSG00000182858 nonsynonymous
```

```
                     REFCODON        VARCODON          REFAA          VARAA
               <DNAStringSet> <DNAStringSet> <AAStringSet> <AAStringSet>
    rs8135963             ACT             GCT              T              A
  rs8135963.1             CAC             CGC              H              R
   22:50301488             CCG             CCA              P              P
 22:50301488.1             GAC             AAC              D              N
 22:50301488.2             CGA             CAA              R              Q
 ---
 seqlengths:
  chr22
     NA
```

Using variant rs114264124 as an example, we see `varAllele A` has been substituted into the `refCodon CGG` to produce `varCodon CAG`. The `refCodon` is the sequence of codons necessary to make the variant allele substitution and therefore often includes more nucleotides than indicated in the range (i.e. the range is 50302962, 50302962, width of 1). Notice it is the second position in the `refCodon` that has been substituted. This position in the codon, the position of substitution, corresponds to genomic position 50302962. This genomic position maps to position 698 in coding region-based coordinates and to triplet 233 in the protein. This is a non-synonymous coding variant where the amino acid has changed from `R` (Arg) to `Q` (Gln).

When the resulting `varCodon` is not a multiple of 3 it cannot be translated. The consequence is considered a `frameshift` and `varAA` will be missing. There are no frameshifts in this file but we do have some variants classified as `nonsense` which indicates premature stop codons.

```
> table(values(coding)$CONSEQUENCE)

     nonsense nonsynonymous    synonymous
           22          1884          1449
```

SIFT and PolyPhen databases From `predictCoding` we identified the amino acid coding changes for the non-synonymous variants. For this subset we can retrieve predictions of how damaging these coding changes may be. SIFT (Sorting Intolerant From Tolerant) and PolyPhen (Polymorphism Phenotyping) are methods that predict the impact of amino acid substitution on a human protein. The SIFT method uses sequence homology and the physical properties of amino acids to make predictions about protein function. PolyPhen uses sequence-based features and structural information characterizing the substitution to make predictions about the structure and function of the protein.

Collated predictions for specific dbSNP builds are available as downloads from the SIFT and PolyPhen web sites. These results have been packaged into *SIFT.Hsapiens.dbSNP132.db* and *PolyPhen.Hapiens.dbSNP131.db* and are designed to be searched by rsid. Variants that are in dbSNP can be searched with these database packages. When working with novel variants, SIFT and PolyPhen must be called directly. See references for home pages.

The pre-calculated predictions from SIFT and PolyPhen are based on particular gene models. SIFT uses Ensembl and PolyPhen uses the UCSC Known Genes track. It is important that the annotation file used to identify coding / non-coding variants is based on the same gene model as these predictions. We will be using SIFT and the TranscriptDb we used had Ensembl gene ids.

Identify the non-synonymous variants and obtain the rsids.

```
> nms <- names(coding)
> idx <- values(coding)$CONSEQUENCE == "nonsynonymous"
> nonsyn <- coding[idx]
> names(nonsyn) <- nms[idx]
> rsids <- unique(names(nonsyn)[grep("rs", names(nonsyn), fixed=TRUE)])
```

Detailed descriptions of the database columns can be found with `?SIFTDbColumns` and `?PolyPhenDbColumns`. Variants in these databases often contain more than one row per variant. The variant may have been reported by multiple sources and therefore the source will differ as well as some of the other variables.

```
> library(SIFT.Hsapiens.dbSNP132)
> ## rsids in the package
> head(keys(SIFT.Hsapiens.dbSNP132), 3)

[1] "rs10000692" "rs10001580" "rs10002700"

> ## list available columns
> cols(SIFT.Hsapiens.dbSNP132)

 [1] "RSID"        "PROTEINID"   "AACHANGE"    "METHOD"      "AA"
 [6] "PREDICTION"  "SCORE"       "MEDIAN"      "POSTIONSEQS" "TOTALSEQS"

> ## select a subset of columns
> ## a warning is thrown when a key is not found in the database
> subst <- c("RSID", "PREDICTION", "SCORE", "AACHANGE", "PROTEINID")
> sift <- select(SIFT.Hsapiens.dbSNP132, keys=rsids, cols=subst)
> head(sift, 3)

         RSID PROTEINID AACHANGE PREDICTION SCORE
1 rs114335781      <NA>     <NA>       <NA>  <NA>
2   rs8135963      <NA>     <NA>       <NA>  <NA>
3 rs114264124 NP_077010    R233Q  TOLERATED  0.59
```

PolyPhen provides predictions using two different training datasets and has considerable information about 3D protein structure. See `?PolyPhenDbColumns` or the PolyPhen web site listed in the references for more details.