Introduction to *VariantAnnotation*

Valerie Obenchain, Michael Lawrence

November 16, 2011

1 Introduction

In this vignette we introduce methods in the *VariantAnnotation*package for the annotation and filtering of genetic variants. Variant locatation with respect to gene function can be determined as well as the amino acid coding changes for nonsynonymous variants. *VariantAnnotation* contains functions to access the SIFT and PolyPhen databases packages. The SIFT and PolyPhen methods predict the possible impact of amino acid coding changes on protein function (SIFT) or function and structure (PolyPhen).

To assist with data access and managment the `readVcf` function parses data from Variant Call Format (VCF) files into a *summarizedExperiment* object. Filtering functions are available for subsetting of variants on physical location in the gene or membership in dbSNP.

2 Variant Location

As sample data we use the `GRanges` object, `variants`, that contains both real and faux variants. The ranges in `variants` represent the positions in the reference sequence that are altered in the variation. A single basepair replacement (SNP) has equal start and end positions and a width of length 1. A deletion or substitution will have a width of 1 or greater depending on the number of basepairs being deleted or substituted. Insertions are represented with start = end + 1 which results in a width of length 0.

Reference and variant alleles are represented as `DNAStringSet`s in `refAllele` and `varAllele` columns. An insertion will have a missing value in the `refAllele` column and a deletion will have a missing value in the `varAllele` column.

```
> library(VariantAnnotation)
> data(variants)
> head(variants)
```

```
GRanges with 6 ranges and 3 elementMetadata values:
      seqnames                 ranges strand |      refAllele
         <Rle>              <IRanges>  <Rle> | <DNAStringSet>
  [1]     chr1 [161003087, 161003087]      * |              C
  [2]     chr1 [ 84647761,  84647761]      * |              C
  [3]     chr1 [ 13133880,  13133881]      * |             TC
  [4]     chr1 [ 11326183,  11326182]      * |
  [5]     chr1 [105293754, 105293754]      * |              A
  [6]     chr1 [ 67478546,  67478546]      * |              G
           varAllele                comments
      <DNAStringSet>             <character>
  [1]              T           rs1000050, SNP
  [2]              T            rs6576700 SNP
  [3]                    rs59770105, deletion
```

```
  [4]             AT    rs35561142, insertion
  [5]          ATAAA rs10552169, substitution
  [6]              A          rs11209026, SNP
  ---
  seqlengths:
    chr1 chr13 chr16  chr2
      NA    NA    NA    NA
```

`locateVariants` returns a `DataFrame` with one row for each variant-transcript match. The queryHits column maps back to the variant in the original query. Location categories of 'coding', 'UTR5', 'UTR3', 'intron' and 'intergenic' are assigned based on the criteria in Table 1. Intergenic variants have two gene ID's, representing the genes preceding and following the variant.

```
> library(TxDb.Hsapiens.UCSC.hg18.knownGene)
> txdb <- TxDb.Hsapiens.UCSC.hg18.knownGene
> loc <- locateVariants(variants, txdb)
> head(loc)

DataFrame with 6 rows and 4 columns
  queryHits        txID                    geneID   Location
  <integer> <character> <CompressedCharacterList>   <factor>
1         1        2966                      4921     intron
2         1        2967                      4921     intron
3         2        1660                     58511     intron
4         2        1661                     58511     intron
5         2        1662                     58511     intron
6         3          NA           343068,401940 intergenic
```

Region	Details
coding	variant falls *within* a coding region
UTR5	variant falls *within* a 5' untranslated region
UTR3	variant falls *within* a 3' untranslated region
intron	variant falls *within* a transcript but not *within* a coding region
intergenic	variant does not fall *within* a transcript

Table 1: Variant Location

Multiple alignments for a single variant should be separate rows in the `query` object. Below is an example snp from dbSNP Build 130 which is aligned to different locations for different assemblies. Each alignment is entered as a separate row in the `singleSNP` object.

```
> singleSNP <- GRanges(seqnames="chr16",
+         ranges=IRanges(start=c(23004226, 22316999, 24133766),
+                width=c(1,1,1)),
+         varAllele=DNAStringSet("T"),
+         comments=c("Celera assembly", "HuRef assembly", "reference assembly"))
> singleSNP

GRanges with 3 ranges and 2 elementMetadata values:
      seqnames               ranges strand |      varAllele
         <Rle>            <IRanges>  <Rle> | <DNAStringSet>
```

```
  [1]    chr16 [23004226, 23004226]      * |             T
  [2]    chr16 [22316999, 22316999]      * |             T
  [3]    chr16 [24133766, 24133766]      * |             T
               comments
           <character>
  [1]    Celera assembly
  [2]     HuRef assembly
  [3] reference assembly
  ---
  seqlengths:
   chr16
      NA
```

The result from `locateVariants` shows the Celera assembly location matching two transcripts in the same gene, both intron regions. The HuRef assembly falls in an intergenic region. The reference assembly hits two different transcripts in the same gene, both in the 3' UTR region.

```
> locateVariants(singleSNP, txdb)
DataFrame with 5 rows and 4 columns
  queryHits        txID                     geneID    Location
  <integer> <character> <CompressedCharacterList> <character>
1         1       52091                     57478      intron
2         1       52092                     57478      intron
3         2          NA                 1039,57478  intergenic
4         3       50256                      5579       3'UTR
5         3       50259                      5579       3'UTR
```

3 Amino Acid Coding

Amino acid coding changes can be determined for non-synonymous variants with `predictCoding`. This function identifies overlaps between the variants in `query` and the coding regions of a `TranscriptDb` object. Sequences for the reference codons are retrieved from either a `BSgenome` or fasta file. Codon sequences for the variants are constructed by substituting, inserting or deleting the `varAllele` values into the reference sequence. Amino acid codes are computed for the resulting variant codon sequence when the length is a multiple of 3. Examples of various coding situations are shown in Table 2. Positions of substitution, insertion or deletion in Table 2 are purely random and were chosen for example purposes only.

Type	refAllele	varAllele	refSeq	varSeq	AA coding of varSeq
substitution	G	T	aag	aaT	yes
substitution	G	TG	tga	tTGa	no
substitution	G	TGCG	gtc	TGCGtc	yes
insertion	''	G	cgg	Gcgg	no
insertion	''	TTG	gaa	gaTTGa	yes
deletion	A	''	atc	tc	no
deletion	GGCCTA	''	acggcctaa	aca	yes

Table 2: Amino Acid Coding

Variants that fall within coding regions are in chromosomes 1, 2 and 16. We subset the `GRanges` of coding regions by transcripts to retain only the chromosomes of interest. Alternatively one could use the full `TranscriptDb`.

```
> library(BSgenome.Hsapiens.UCSC.hg18)
> cdsByTx <- cdsBy(txdb)
> grl <- keepSeqlevels(cdsByTx, c("chr1", "chr2", "chr16"))
> aaCoding <- predictCoding(variants, grl, seqSource=Hsapiens,
+     varAllele="varAllele")
> head(aaCoding)

DataFrame with 6 rows and 10 columns
  queryHits        txID         refSeq          varSeq         refAA
  <integer> <character> <DNAStringSet> <DNAStringSet> <AAStringSet>
1         6        2520            CGA             CAA             R
2         6        2521            CGA             CAA             R
3         6        2522            CGA             CAA             R
4         6        2524            CGA             CAA             R
5         7       10566            ACT             GCT             T
6         7       10567            ACT             GCT             T
          varAA   Consequence    cds_id    cds_name exon_rank
  <AAStringSet>      <factor> <integer> <character> <integer>
1             Q nonsynonymous      8820          NA         5
2             Q nonsynonymous      8821          NA         6
3             Q nonsynonymous      8822          NA         7
4             Q nonsynonymous      8823          NA         8
5             A nonsynonymous      8824          NA         9
6             A nonsynonymous      8825          NA        10
```

The result is a `DataFrame` containing only the variants in coding regions. Columns include queryHits, txID, refSeq, varSeq, refAA, varAA, Consequence and any metadata columns that were present in the `subject`.

The result has one row for each transcript matched by a variant. The queryHits column is the map back to the variants in the original `query`. Multiple rows are returned for variants that match multiple transcripts. For example, the variant in the sixth row of `variants` fell within a coding region and matched four transcripts. The first four rows of the aaCoding result are the data for this variant.

The Consequence column indicates 'synonymous' or 'nonsynonymous' for variants for which the codons could be translated. If translation was not possible the Consequence was considered a 'frameshift'. Variants for which no amino acid codes were computed have a missing in value in the varAA column.

```
> aaCoding[width(aaCoding$varAA) == 0,]

DataFrame with 8 rows and 10 columns
  queryHits        txID         refSeq          varSeq         refAA
  <integer> <character> <DNAStringSet> <DNAStringSet> <AAStringSet>
1        10       51251            CAG             CACG            Q
2        11       51251            CAG          CACTGGG            Q
3        13       51251            GCC               C            A
4        15       51249            CGG             TTGG            R
5        15       51250            CGG             TTGG            R
6        15       51251            CGG             TTGG            R
7        15       51252            CGG             TTGG            R
8        15       51253            CGG             TTGG            R
          varAA Consequence    cds_id    cds_name exon_rank
  <AAStringSet>    <factor> <integer> <character> <integer>
1                frameshift      8822          NA         7
```

```
2               frameshift      8822          NA         7
3               frameshift      8822          NA         7
4               frameshift      8830          NA         5
5               frameshift      8821          NA         6
6               frameshift      8822          NA         7
7               frameshift      8823          NA         8
8               frameshift      8824          NA         9
```

4 Filtering

Identifying subsets of variants that meet a specific criteria can be accomplished using the `dbSNPfilter` or `regionFilter` functions. Filters are created with the appropriate reference object such as a `SNPlocs` or `TranscriptDb` object.

```
> library("SNPlocs.Hsapiens.dbSNP.20090506")
> ## create filter to identify variants present in dbSNP
> snpFilt <- dbSNPFilter("SNPlocs.Hsapiens.dbSNP.20090506")
> ## create filter to identify variants present in introns
> regionFilt <- regionFilter(txdb, region="intron")
> regionFilt
```

```
function (x, subset = TRUE)
{
    res <- function (x)
    {
        loc <- locateVariants(x, txdb)
        res <- logical(length(x))
        res[unique(loc$queryHits[loc$Location %in% region])] <- TRUE
        res
    }(x)
    VAFilterResult(x, res, "regionFilter", subset)
}
<environment: 0x0d3ff8ac>
attr(,"name")
"regionFilter"
attr(,"class")
[1] "VAFilter"
attr(,"class")attr(,"package")
[1] "VariantAnnotation"
```

When filtering data with `dbSNPFilter` or `regionsFilt` two options are available for the return object. If the argument `subset=TRUE` (default) a `GRanges` is returned that contains only the records that passed the filter. If `subset=FALSE`, the result is a `VAFilterResult` object. This object contains a logical vector indicating which records passed the filter and some stats on the number of records removed and what filter was applied.

```
> ## return the subset of records that passed the filter
> snpResult <- snpFilt(variants)
> snpResult
```

```
GRanges with 6 ranges and 3 elementMetadata values:
      seqnames                 ranges strand |      refAllele
```

```
        <Rle>              <IRanges>  <Rle> | <DNAStringSet>
  [1]     chr1 [161003087, 161003087]      * |              C
  [2]     chr1 [ 84647761,  84647761]      * |              C
  [3]     chr1 [ 67478546,  67478546]      * |              G
  [4]     chr2 [233848107, 233848107]      * |              A
  [5]    chr16 [ 49303427,  49303427]      * |              C
  [6]    chr16 [ 49314041,  49314041]      * |              G
           varAllele        comments
      <DNAStringSet>     <character>
  [1]              T  rs1000050, SNP
  [2]              T   rs6576700 SNP
  [3]              A rs11209026, SNP
  [4]              G  rs2241880, SNP
  [5]              T  rs2066844, SNP
  [6]              C  rs2066845, SNP
  ---
  seqlengths:
    chr1 chr13 chr16  chr2
      NA    NA    NA    NA

> metadata(snpResult)

$name
"dbSNPFilter"

$stats
         Name Input Passing   Op
1 dbSNPFilter    17       6 <NA>

> ## return a VAFilterResult object
> snpFilt(variants, subset=FALSE)

class: VAFilterResult
name: dbSNPFilter
output: TRUE TRUE ... FALSE FALSE
stats:
         Name Input Passing   Op
1 dbSNPFilter    17       6 <NA>
```

Filters can be combined to act in unison,

```
# return a subset of data passing both filters
> combofilt <- compose(snpFilt, regionFilt)
> combofilt(variants)
region= intron
GRanges with 2 ranges and 3 elementMetadata values
    seqnames               ranges strand |      refAllele      varAllele
       <Rle>            <IRanges>  <Rle> | <DNAStringSet> <DNAStringSet>
[1]     chr1 [161003087, 161003087]      * |              C              T
[2]     chr1 [ 84647761,  84647761]      * |              C              T
         comments
       <character>
[1] rs1000050, SNP
```

```
[2]  rs6576700 SNP

seqlengths
  chr1 chr13 chr16  chr2
    NA    NA    NA    NA
```

5 Variant Call Format (VCF) files

The `readVcf` function reads VCF files and parses them into a `SummarizedExperiment` object. When the compressed VCF file has a tabix file index a `GRanges` can be used to specify a subset of ranges. The sample VCF in `VariantAnnotation` is not compressed so in this example we compress on the fly, create a tabix index and retrieve a subset of ranges.

```
> ## compress vcf file , create index
> vcfFile <- system.file("extdata", "ex1.vcf", package="VariantAnnotation")
> from <- vcfFile
> to <- tempfile()
> compressVcf <- bgzip(from, to)
> idx <- indexTabix(compressVcf, "vcf")
> tab <- TabixFile(compressVcf, idx)
> head <- headerTabix(tab)
> ## select ranges with param
> gr <- GRanges(seqnames="1",
+               ranges=IRanges(start=0, end=100000))
> vcf <- readVcf(tab, param=gr)
```

All data are parsed according to the header information in the VCF file. The header is stored as a `SimpleList` in the `exptData` slot and can be viewed by querying the elements of the list.

```
> vcf

class: SummarizedExperiment
dim: 7 2
exptData(1): HEADER
assays(3): GT GQ DP
rownames(7): . . ... . .
rowData values names(5): REF ALT QUAL FILTER DP
colnames(2): A B
colData names(1): Samples

> exptData(vcf)[["HEADER"]]

SimpleDataFrameList of length 4
names(4): META FILTER FORMAT INFO

> exptData(vcf)[["HEADER"]][["META"]]

DataFrame with 1 row and 1 column
                  Value
            <character>
fileformat     VCFv4.0
```

The eight required fields of the VCF file are parsed into a `GRanges` object in the `rowData` slot. The `ALT` alleles are provided as a `DataFrameList` to accomodate for multiple values per variant. All fields in the `INFO` column are parsed into individual elementMetadata columns.

```
> exptData(vcf)[["HEADER"]][["FILTER"]]

DataFrame with 1 row and 1 column
         Description
         <character>
q10 Quality below 10

> exptData(vcf)[["HEADER"]][["INFO"]]

DataFrame with 1 row and 3 columns
        Number        Type Description
   <character> <character> <character>
DP           1     Integer Total Depth

> rowData(vcf)

GRanges with 7 ranges and 5 elementMetadata values:
    seqnames     ranges strand |            REF
       <Rle>  <IRanges>  <Rle> | <DNAStringSet>
  .        1 [100, 100]      * |              G
  .        1 [110, 110]      * |              C
  .        1 [120, 120]      * |              T
  .        1 [130, 132]      * |            GAA
  .        1 [140, 141]      * |             GT
  .        1 [150, 154]      * |          TAAAA
  .        1 [160, 161]      * |             TA
                               ALT      QUAL      FILTER        DP
    <CompressedSplitDataFrameList> <numeric> <character> <integer>
  .                        ########      1806         q10        35
  .                        ########      1792        PASS        32
  .                        ########       628         q10        21
  .                        ########      1016        PASS        22
  .                        ########       727        PASS        30
  .                        ########       246        PASS        10
  .                        ########       246        PASS        10
  ---
  seqlengths:
    1
   NA

> ## ALT as a DNAStringSetList
> head(as.list(values(rowData(vcf))[["ALT"]]), 4)

$`1`
DataFrame with 1 row and 1 column
             ALT
  <DNAStringSet>
1              C
```

```
$`2`
DataFrame with 0 rows and 1 column

$`3`
DataFrame with 1 row and 1 column
             ALT
  <DNAStringSet>
1              G

$`4`
DataFrame with 2 rows and 1 column
             ALT
  <DNAStringSet>
1              G
2            GTA
```

The genotype data described in the `FORMAT` fields are parsed into matrices in the `assays` slot.

```
> exptData(vcf)[["HEADER"]][["FORMAT"]]

DataFrame with 3 rows and 3 columns
        Number        Type      Description
   <character> <character>      <character>
GT            1      String         Genotype
GQ            1     Integer Genotype Quality
DP            1     Integer       Read Depth

> assays(vcf)

SimpleList of length 3
names(3): GT GQ DP

> head(assays(vcf)$GT)

  A       B
. "1/1"   "1/1"
. "0/0"   "0/0"
. "0|1|1" "0|1|1"
. "1/2"   "1/2"
. "0/1"   "0/1"
. "1/2"   "1/2"
```

6 SIFT and PolyPhen Databases

SIFT (Sorting Intolerant From Tolerant) and PolyPhen (Polymorphism Phenotyping) are methods for predicting 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 predicitons about the structure and function of the protein. Both methods offer interactive tools and documentation on their web sites,
`http://sift.bii.a-star.edu.sg/` `http://genetics.bwh.harvard.edu/pph2/`

Collated predictions for specific dbSNP builds are available as downloads from the SIFT and PolyPhen web sites. These results have been stored as sqlite databases and made available through the `Bioconductor`

packages *SIFT.Hsapiens.dbSNP132.db* and *PolyPhen.Hapiens.dbSNP131.db*. The packages are designed to be serached by rsid.

```
> library(SIFT.Hsapiens.dbSNP132)
> library(PolyPhen.Hsapiens.dbSNP131)
> ## rsids in the package
> head(keys(SIFT.Hsapiens.dbSNP132))

[1] "rs10000692" "rs10001580" "rs10002700" "rs10003238" "rs10003369"
[6] "rs10004"

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

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

> rsids <- c("rs2142947", "rs8692231", "rs3026284")
> subst <- c("RSID", "PREDICTION", "SCORE", "AACHANGE", "PROTEINID")
> select(SIFT.Hsapiens.dbSNP132, keys=rsids, cols=subst)

        RSID    PROTEINID AACHANGE  PREDICTION SCORE
1  rs2142947 NP_001019832    F430L   TOLERATED  1.00
2  rs2142947 NP_001019832    F430L   TOLERATED  0.74
3  rs2142947 NP_001019832    F430L   TOLERATED  0.72
4  rs2142947 NP_001019832    F430L   TOLERATED     1
5  rs8692231    XP_537288     P37R  NOT SCORED  <NA>
6  rs8692231    XP_537288     P37R  NOT SCORED  <NA>
7  rs8692231    XP_537288     P37R  NOT SCORED  <NA>
8  rs8692231    XP_537288     P37R  NOT SCORED  <NA>
9  rs3026284    NP_004920    G202D DELETERIOUS  0.03
10 rs3026284    NP_004920    G202D   TOLERATED  1.00
11 rs3026284    NP_004920    G202D DELETERIOUS  0.00
12 rs3026284    NP_004920    G202D   TOLERATED     1

> select(PolyPhen.Hsapiens.dbSNP131, keys="rs3026284",
+     cols=c("TRAININGSET", "PREDICTION", "PPH2PROB", "PPH2FPR", "PPH2FDR"))

       RSID TRAININGSET        PREDICTION PPH2PROB PPH2FPR PPH2FDR
1 rs3026284      humdiv probably damaging    0.936  0.0746   0.168
2 rs3026284      humvar probably damaging    0.919  0.1330      NA
```

Detailed column descriptions can be found at ?SIFTDbColumns and ?PolyPhenDbColumns.

7 References

Wang K, Li M, Hakonarson H, (2010), ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. Nucleic Acids Research, Vol 38, No. 16, e164.

McLaren W, Pritchard B, RiosD, et. al., (2010), Deriving the consequences of genomic variants with the Ensembl API and SNP Effect Predictor. Bioinformatics, Vol. 26, No. 16, 2069-2070.

8 Session Information

```
R version 2.14.0 (2011-10-31)
Platform: i386-pc-mingw32/i386 (32-bit)

locale:
[1] LC_COLLATE=C
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods
[7] base

other attached packages:
 [1] PolyPhen.Hsapiens.dbSNP131_1.0.0
 [2] SIFT.Hsapiens.dbSNP132_1.0.0
 [3] RSQLite_0.10.0
 [4] DBI_0.2-5
 [5] SNPlocs.Hsapiens.dbSNP.20090506_0.99.6
 [6] BSgenome.Hsapiens.UCSC.hg18_1.3.17
 [7] BSgenome_1.22.0
 [8] TxDb.Hsapiens.UCSC.hg18.knownGene_2.6.2
 [9] GenomicFeatures_1.6.2
[10] VariantAnnotation_1.0.5
[11] Rsamtools_1.6.1
[12] AnnotationDbi_1.16.2
[13] Biobase_2.14.0
[14] Biostrings_2.22.0
[15] GenomicRanges_1.6.3
[16] IRanges_1.12.1

loaded via a namespace (and not attached):
 [1] Matrix_1.0-1       RCurl_1.7-0.1      XML_3.4-2.2
 [4] biomaRt_2.10.0     bitops_1.0-4.1     grid_2.14.0
 [7] lattice_0.20-0     rtracklayer_1.14.0 snpStats_1.4.0
[10] splines_2.14.0     survival_2.36-10   tools_2.14.0
[13] zlibbioc_1.0.0
```