

Introduction to *customProDB*

Xiaojing Wang, Bing Zhang

August 9, 2018

Contents

1	Introduction	1
2	Preparing annotation files	2
2.1	Refseq annotation from UCSC table browser	3
2.2	ENSEMBL annotation from BIOMART	4
3	Building database from a single sample	4
3.1	Filtering based on transcript expression	4
3.2	Variation annotation	5
3.2.1	SNVs	8
3.2.2	INDELs	10
3.3	Splice junction analysis	10
4	Building database from multiple samples	13
4.1	Filtering based on transcript expression in multiple samples	13
4.2	Variations occurred in multiple samples	13
4.3	Junctions occurred in multiple samples	14
5	Two integrated functions	15
6	FASTA file format	16
6.1	Normal proteins passing the expression cutoff	16
6.2	Variant Proteins induced by SNVs	16
6.3	Aberrant proteins induced by INDELs	16
6.4	Novel junction peptides	17
7	Session Information	17

1 Introduction

Mass spectrometry (MS)-based proteomics technology is widely used in biological researches. For peptide and protein identification, sequence database search is the most popular method. We recently showed that a sample-specific protein database derived from RNA-Seq data could better

approximate the real protein pool and thus improve protein identification. With continuously decreasing cost, more and more groups have started multilayer experiment designs that profile both proteome and transcriptome of the same cohort of samples in order to gain a comprehensive understanding of cellular systems. To facilitate such efforts, we have developed this R package *customProDB*, which is dedicated to the generation of customized databases from RNA-Seq data for proteomics searches.

We designed this package based on a few assumptions (1) undetected or lowly expressed transcripts are less likely to produce detectable proteins, thus excluding them would improve sensitivity and specificity; (2) each sample has a unique set of SNPs, mutations, gene fusions, alternative splicing etc, including them in them in the protein database would allow the identification of sample specific proteins. This is particularly useful in cancer studies, in which tumors typically carry oncogenic genomic alterations.

To filter out undetected or lowly expressed transcripts, the package provides functions to either calculate the RPKM (Reads Per Kilobase per Million mapped reads) values, or accept user-provided measurements from other sources such as the FPKM (Fragments per kilobase of exon per million fragments mapped) from cufflinks. Users may specify a expression threshold, subsequently a FASTA file is generated for proteins that pass the threshold.

customProDB allows users to incorporate variations identified from RNA-seq data into the FASTA database. It annotates all SNVs with their proper locations and functional consequences in transcripts. Non-synonymous coding variations are introduced to protein sequences to create variant protein entries. Aberrant proteins resulted from short INDELs are also predicted and added to the variation database.

One important application of RNA-Seq is to identify previously unannotated structures, such as novel exons, alternative splice variants and gene fusions. The package provides a function to classify splice junctions identified from RNA-Seq data, and then uses three-frame translation to generate peptides that cross the novel junctions. Similarly, fusion genes can also be incorporated into the FASTA database.

This document provides a step by step tutorial of customized database generation.

2 Preparing annotation files

To map RNA-Seq information to the protein level, numerous pieces of genome annotation information are needed, such as genome elements region boundary, protein coding sequence, protein sequence and known SNPs et al. It is possible to manually download these data from different public resources (e.g. NCBI, UCSC and ENSEMBL) and then parse them to an appropriate format. But to make the process more efficient and autonomous, we provide two functions to prepare the gene/transcript annotation files. Users should use the same version of annotations through the entire dataset(s) analysis. All the annotations are saved to a specified directory for latter use.

The dbSNP data is huge and is getting larger and larger. These two functions only download the data in coding region for performance reasons. Use the code below to check the current dbSNP versions for a specified genome provided by the UCSC table browser.

```
> library('rtracklayer')
> session <- browserSession()
> genome(session) <- 'hg19'
```

```
> dbsnps <- trackNames(session)[grep('snp', trackNames(session), fixed=T)]
> dbsnps
```

SNPedia	Common SNPs(150)	Common SNPs(147)	Common SNPs(146)
"snpedia"	"snp150Common"	"snp147Common"	"snp146Common"
Common SNPs(144)	Common SNPs(142)	Common SNPs(141)	All SNPs(150)
"snp144Common"	"snp142Common"	"snp141Common"	"snp150"
All SNPs(147)	All SNPs(146)	All SNPs(144)	All SNPs(142)
"snp147"	"snp146"	"snp144"	"snp142"
All SNPs(141)	Flagged SNPs(150)	Flagged SNPs(147)	Flagged SNPs(146)
"snp141"	"snp150Flagged"	"snp147Flagged"	"snp146Flagged"
Flagged SNPs(144)	Flagged SNPs(142)	Flagged SNPs(141)	Mult. SNPs(150)
"snp144Flagged"	"snp142Flagged"	"snp141Flagged"	"snp150Mult"
Mult. SNPs(147)	Mult. SNPs(146)	Mult. SNPs(144)	Mult. SNPs(142)
"snp147Mult"	"snp146Mult"	"snp144Mult"	"snp142Mult"
Mult. SNPs(138)	All SNPs(138)	Common SNPs(138)	Flagged SNPs(138)
"snp138Mult"	"snp138"	"snp138Common"	"snp138Flagged"

2.1 Refseq annotation from UCSC table browser

The `PrepareAnnotationRefseq` function downloads annotations from the UCSC table browser through `rtracklayer`, extracts and derives the relevant information and then saves them as the required R data structure. However, this function is not totally the automatic, it requires users to download coding sequence and protein sequence FASTA files from UCSC table browser. Since Refseq updates from time to time, we suggest generating the FASTA file the same day as running this function.

The bullet list below summarizes the steps to download coding sequence FASTA files.

- Go to UCSC Table Browser
- Choose genome
- Choose assembly
- Group — Genes and Gene Prediction Tracks
- Track — RefSeq Genes
- Table — refGene
- Region — genome (If you only need some genes, choose paste list or upload list)
- Output format — sequence
- Then choose genomic — CDS exons — one FASTA record per gene
- Press 'get sequence' button

Downloading protein sequence FASTA file is the same as above, just choose 'protein' instead of 'genomic' after clicking the 'get output' button.

```

> library(customProDB)
> transcript_ids <- c("NM_001126112", "NM_033360", "NR_073499", "NM_004448",
+                   "NM_000179", "NR_029605", "NM_004333", "NM_001127511")
> pepfasta <- system.file("extdata", "refseq_pro_seq.fasta",
+                          package="customProDB")
> CDSfasta <- system.file("extdata", "refseq_coding_seq.fasta",
+                          package="customProDB")
> annotation_path <- tempdir()
> PrepareAnnotationRefseq(genome='hg19', CDSfasta, pepfasta, annotation_path,
+                          dbsnp = NULL, transcript_ids=transcript_ids,
+                          splice_matrix=FALSE, ClinVar=FALSE)

```

2.2 ENSEMBL annotation from BIOMART

An alternative resource for annotation is ENSEMBL. The `PrepareAnnotationEnsembl` function downloads the annotation from ENSEMBL through *biomaRt*. This process may take several hours if users choose to download the whole dataset. The ENSEMBL version number can be specified in the `host` in `useMart` function. Go to website <http://useast.ensembl.org/info/website/archives/index.html> to check the currently available archives. It took about 1.5 hour to prepare all annotations for ENSEMBL v82 in our tests.

```

> ensembl <- useMart("ENSEMBL_MART_ENSEMBL", dataset="hsapiens_gene_ensembl",
+                   host="sep2015.archive.ensembl.org", path="/biomart/martservice",
+                   archive=FALSE)
> annotation_path <- tempdir()
> transcript_ids <- c("ENST00000234420", "ENST00000269305", "ENST00000445888",
+                   "ENST00000257430", "ENST00000508376", "ENST00000288602",
+                   "ENST00000269571", "ENST00000256078", "ENST00000384871")
> PrepareAnnotationEnsembl(mart=ensembl, annotation_path=annotation_path,
+                          splice_matrix=FALSE, dbsnp=NULL,
+                          transcript_ids=transcript_ids, COSMIC=FALSE)

```

3 Building database from a single sample

After preparing all the annotation files, there are usually three steps to build a customized database. Users could choose one or multiple steps according to the research interest.

3.1 Filtering based on transcript expression

For a given BAM file, the `calculateRPKM` function computes the RPKM for each transcript based on reads mapped to the exon region. The output is a numeric vector. Users should make sure that the chromosome name in annotation and the BAM file are consistent, otherwise errors will be raised.

After getting RPKMs, users may check the distribution and choose a cutoff to retain relatively highly expressed transcripts that are more likely to produce proteins that are detectable in shotgun proteomics.

```

> load(system.file("extdata/refseq", "exon_anno.RData", package="customProDB"))
> bamFile <- system.file("extdata/bams", "test1_sort.bam", package="customProDB")
> load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
> RPKM <- calculateRPKM(bamFile, exon, proteincodingonly=TRUE, ids)

```

Alternatively, users could input the calculated RPKM/FPKM from other software output rather than to calculate from BAM file, such as the cufflinks output. The cutoff can be defined based on a specific RPKM/FPKM value or a specific percentile. The default cutoff is '30%', which means that only the top 70% transcripts with the largest RPKM values are retained. Then the `Outputproseq` function could output a FASTA format file containing protein sequences with corresponding transcript RPKM/FPKM values above the cutoff.

```

> load(system.file("extdata/refseq", "proseq.RData", package="customProDB"))
> outf1 <- paste(tempdir(), '/test_rpkm.fasta', sep='')
> Outputproseq(RPKM, 1, proteinseq, outf1, ids)

```

3.2 Variation annotation

First, users can input variations from a single VCF file using `InputVcf`. The package generates a list of `GRanges` object as output. It works for VCF file containing either one or multiple samples.

```

> # single sample
> vcffile <- system.file("extdata/vcfs", "test1.vcf", package="customProDB")
> vcf <- InputVcf(vcffile)
> length(vcf)

```

```
[1] 1
```

```
> vcf[[1]][1:3]
```

GRanges object with 3 ranges and 40 metadata columns:

	seqnames	ranges	strand	REF	
	<Rle>	<IRanges>	<Rle>	<character>	
chr1:32386425_T/C	chr1	32386425	*	T	
chr1:32507666_G/T	chr1	32507666	*	G	
chr1:32524459_A/C	chr1	32524459	*	A	
	ALT	QUAL	FILTER	DP	
	<character>	<numeric>	<character>	<integer>	
chr1:32386425_T/C	C	24	.	3	
chr1:32507666_G/T	T	6.2	.	5	
chr1:32524459_A/C	C	3.54	.	5	
	DP4.DP4	DP4.DP4.1	DP4.DP4.2	DP4.DP4.3	MQ
	<integer>	<integer>	<integer>	<integer>	<integer>
chr1:32386425_T/C	0	0	0	3	50
chr1:32507666_G/T	3	0	2	0	50
chr1:32524459_A/C	1	2	0	2	50

	FQ	AF1	AC1	G3.G3	G3.G3.1
	<numeric>	<numeric>	<numeric>	<numeric>	<numeric>
chr1:32386425_T/C	-36	1	2	<NA>	<NA>
chr1:32507666_G/T	8.65	0.4999	1	<NA>	<NA>
chr1:32524459_A/C	5.47	0.4998	1	<NA>	<NA>
	G3.G3.2	HWE	CLR	UGT	
	<numeric>	<numeric>	<integer>	<character>	
chr1:32386425_T/C	<NA>	<NA>	<NA>	<NA>	
chr1:32507666_G/T	<NA>	<NA>	<NA>	<NA>	
chr1:32524459_A/C	<NA>	<NA>	<NA>	<NA>	
	CGT	PV4.PV4	PV4.PV4.1	PV4.PV4.2	
	<character>	<numeric>	<numeric>	<numeric>	
chr1:32386425_T/C	<NA>	<NA>	<NA>	<NA>	
chr1:32507666_G/T	<NA>	1	0.062	1	
chr1:32524459_A/C	<NA>	1	0.0021	1	
	PV4.PV4.3	INDEL	PC2.PC2	PC2.PC2.1	PCHI2
	<numeric>	<logical>	<integer>	<integer>	<numeric>
chr1:32386425_T/C	<NA>	FALSE	<NA>	<NA>	<NA>
chr1:32507666_G/T	0.36	FALSE	<NA>	<NA>	<NA>
chr1:32524459_A/C	1	FALSE	<NA>	<NA>	<NA>
	QCHI2	PR	GT	GQ	
	<integer>	<integer>	<character>	<character>	
chr1:32386425_T/C	<NA>	<NA>	1/1	15	
chr1:32507666_G/T	<NA>	<NA>	0/1	36	
chr1:32524459_A/C	<NA>	<NA>	0/1	30	
	DP.1	SP	PL	PL.1	
	<character>	<character>	<character>	<character>	
chr1:32386425_T/C	<NA>	<NA>	56	9	
chr1:32507666_G/T	<NA>	<NA>	35	0	
chr1:32524459_A/C	<NA>	<NA>	31	0	
	PL.2	PL.3	PL.4	PL.5	
	<character>	<character>	<character>	<character>	
chr1:32386425_T/C	0	56	9	0	
chr1:32507666_G/T	78	35	0	78	
chr1:32524459_A/C	98	31	0	98	

seqinfo: 7 sequences from an unspecified genome; no seqlengths

```
> # multiple samples in one VCF file
> vcffile <- system.file("extdata", "test_mul.vcf", package="customProDB")
> vcfs <- InputVcf(vcffile)
```

After reading the VCF file, users should choose the functions corresponding to different variation types, SNVs or INDELS. Although the package focuses on protein coding transcripts, we intentionally implemented several functions to examine where the SNVs are located, how many of

them are located in the protein coding transcript regions, etc. The `Varlocation` functions classifies variations into eight categories, see Table 1.

Label	Description
Intergenic	Out of transcripts boundary
Intron_nonprocoding	Located in introns of non-coding transcripts
Exon_nonprocoding	Located in exons of non-coding transcripts
Intron	Located in introns of protein coding transcripts
5'UTR	Located in 5utr region of protein coding transcripts
3'UTR	Located in 3utr region of protein coding transcripts
Coding	Located in coding region of protein coding transcripts
Unknown	No annotation for this chromosome

Table 1: Definition of genomic locations of variations

```
> table(values(vcf[[1]])[['INDEL']])
```

```
FALSE TRUE
  54     7
```

```
> index <- which(values(vcf[[1]])[['INDEL']]==TRUE)
> indelvcf <- vcf[[1]][index]
> index <- which(values(vcf[[1]])[['INDEL']]==FALSE)
> SNVvcf <- vcf[[1]][index]
> load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
> txdb <- loadDb(system.file("extdata/refseq", "txdb.sqlite", package="customProDB"))
> SNVloc <- Varlocation(SNVvcf,txdb,ids)
> indelloc <- Varlocation(indelvcf,txdb,ids)
> table(SNVloc[, 'location'])
```

```

      3'UTR          Coding          Intergenic
      11           11           25
Intron Intron_nonprocoding
      5             2

```

For those variations labeled with 'Coding', the `Positionincoding` function computes the position of variation in the coding sequence of each transcript. The dbSNP rsid and COSMIC_id can also be retrieved if they are available.

```
> load(system.file("extdata/refseq", "exon_anno.RData", package="customProDB"))
> load(system.file("extdata/refseq", "dbsnpinCoding.RData", package="customProDB"))
> load(system.file("extdata/refseq", "cosmic.RData", package="customProDB"))
> postable_snv <- Positionincoding(SNVvcf, exon, dbsnpinCoding, COSMIC=cosmic)
> postable_snv
```

	genename	txname	txid	proname	chr	strand	pos
1	KRAS	NM_033360	6	NP_203524	chr12	-	25368462
2	ERBB2	NM_004448	7	NP_004439	chr17	+	37866082
3	MSH6	NM_000179	2	NP_000170	chr2	+	48010558
4	MSH6	NM_000179	2	NP_000170	chr2	+	48018081
5	MSH6	NM_000179	2	NP_000170	chr2	+	48018221
6	MSH6	NM_000179	2	NP_000170	chr2	+	48027990
7	APC	NM_001127511	3	NP_001120983	chr5	+	112162854
8	APC	NM_001127511	3	NP_001120983	chr5	+	112164561
9	APC	NM_001127511	3	NP_001120983	chr5	+	112175639
10	APC	NM_001127511	3	NP_001120983	chr5	+	112176559
11	APC	NM_001127511	3	NP_001120983	chr5	+	112176756

	refbase	varbase	pincoding	rsid	COSMIC_id
1	C	T	483	rs4362222	<NA>
2	G	A	591	<NA>	COSM260714
3	C	A	186	rs1042820	<NA>
4	A	G	276	rs1800932	<NA>
5	C	T	416	<NA>	<NA>
6	G	T	2868	<NA>	COSM172960
7	T	C	1404	rs2229992	<NA>
8	G	A	1581	rs351771	<NA>
9	C	T	4294	rs121913332	COSM19149
10	T	G	5214	rs866006	<NA>
11	T	A	5411	rs459552	<NA>

```
> postable_indel <- Positionincoding(indelvcf, exon)
> postable_indel
```

	genename	txname	txid	proname	chr	strand	pos
1	APC	NM_001127511	3	NP_001120983	chr5	+	112154737
2	APC	NM_001127511	3	NP_001120983	chr5	+	112175897

	refbase	varbase	pincoding
1	CT	C	954
2	GAA	GA	4552

3.2.1 SNVs

Variations can be divided into SNVs and INDELS. There are different consequences for SNVs. By taking outputs of function `Positionincoding`, function `aaVariation` is used to predict the consequences of the SNVs in a protein sequence, i.e. synonymous or non-synonymous.

The non-synonymous variations are labeled as either AposB (A is the reference codon and B is the variation codon, e.g., E13V) or nonsense.

```
> load(system.file("extdata/refseq", "procodingseq.RData", package="customProDB"))
> txlist <- unique(postable_snv[, 'txid'])
> codingseq <- procodingseq[procodingseq[, 'tx_id'] %in% txlist,]
```



```
> mtab <- aaVariation (postable_snv, codingseq)
> mtab
```

	txid	genename	txname	proname	chr	strand	pos
1	2	MSH6	NM_000179	NP_000170	chr2	+	48010558
2	2	MSH6	NM_000179	NP_000170	chr2	+	48018081
3	2	MSH6	NM_000179	NP_000170	chr2	+	48018221
4	2	MSH6	NM_000179	NP_000170	chr2	+	48027990
5	3	APC	NM_001127511	NP_001120983	chr5	+	112162854
6	3	APC	NM_001127511	NP_001120983	chr5	+	112164561
7	3	APC	NM_001127511	NP_001120983	chr5	+	112175639
8	3	APC	NM_001127511	NP_001120983	chr5	+	112176559
9	3	APC	NM_001127511	NP_001120983	chr5	+	112176756
10	6	KRAS	NM_033360	NP_203524	chr12	-	25368462
11	7	ERBB2	NM_004448	NP_004439	chr17	+	37866082

	refbase	varbase	pincooding	rsid	COSMIC_id	refcode	varcode
1	C	A	186	rs1042820	<NA>	CGC	CGA
2	A	G	276	rs1800932	<NA>	CCA	CCG
3	C	T	416	<NA>	<NA>	ACA	ATA
4	G	T	2868	<NA>	COSM172960	GAG	GAT
5	T	C	1404	rs2229992	<NA>	TAT	TAC
6	G	A	1581	rs351771	<NA>	GCG	GCA
7	C	T	4294	rs121913332	COSM19149	CGA	TGA
8	T	G	5214	rs866006	<NA>	TCT	TCG
9	T	A	5411	rs459552	<NA>	GTC	GAC
10	C	T	483	rs4362222	<NA>	AGG	AGA
11	G	A	591	<NA>	COSM260714	CCG	CCA

	vartype	aaref	aapos	aavar
1	synonymous	R	62	R
2	synonymous	P	92	P
3	non-synonymous	T	139	I
4	non-synonymous	E	956	D
5	synonymous	Y	468	Y
6	synonymous	A	527	A
7	non-synonymous	R	1432	*
8	synonymous	S	1738	S
9	non-synonymous	V	1804	D
10	synonymous	R	161	R
11	synonymous	P	197	P

Then `OutputVarproseq` function replace the reference amino acid with the variation, and output a FASTA file containing those variant proteins. There are several options for output, users could choose either put all the SNVs of a protein into the sequence or put one SNVs each time.

```
> outfile <- paste(tempdir(), '/test_snv.fasta', sep='')
> load(system.file("extdata/refseq", "proseq.RData", package="customProDB"))
> OutputVarproseq(mtab, proteinseq, outfile, ids)
```

3.2.2 INDELS

Short insertion/deletion may led to frame shift thus produce aberrant proteins. We provide a function `OutputabrrrentPro` to generate a FASTA file containing such proteins.

```
> txlist_indel <- unique(postable_indel[, 'txid'])
> codingseq_indel <- procodingseq[procodingseq[, 'tx_id'] %in% txlist_indel, ]
> outfile <- paste(tempdir(), '/test_indel.fasta', sep='')
> Outputaberrant(postable_indel, coding=codingseq_indel, proteinseq=proteinseq,
+               outfile=outfile, ids=ids)
```

3.3 Splice junction analysis

One important application of RNA-Seq is the identification of previously unannotated structures, such as novel exons, alternative splicing and gene fusions. `Bed2Range` is used to input a BED file. Based on a BED file that contains splice junctions from RNA-Seq data, the function `JunctionType` classifies all the junctions into six categories, Table 2. The category 'connect two known exon' is further divided into known junction, novel alternative splicing and gene fusion. Users need to set the parameter `splice_matrix` to `TRUE` when preparing the annotation files if planning to do junction analysis in this section.

Label	sub-label
connect two known exon	known junction
connect two known exon	alternative splicing
connect two known exon	gene fusion
connect one known exon and one region overlap with known exon	
connect one known exon and one non-exon region	
connect two regions both overlaped with known exons	
connect one region overlap with known exon and one non-exon region	
connect two non-exon region	

Table 2: Junction Type

A complete BED file is required for this function. The output of function `JunctionType` provides more detailed information of the junction, such as transcript source et al.

```
> bedfile <- system.file("extdata/beds", "junctions1.bed", package="customProDB")
> jun <- Bed2Range(bedfile, skip=1, covfilter=5)
> jun
```

GRanges object with 56 ranges and 8 metadata columns:

	seqnames	ranges	strand	id	cov
	<Rle>	<IRanges>	<Rle>	<character>	<integer>
[1]	chr1	32479978-32495899	+	JUNC00002865	8
[2]	chr1	32496023-32497125	+	JUNC00002866	13
[3]	chr1	32497241-32498789	+	JUNC00002868	20
[4]	chr1	32498935-32502511	+	JUNC00002869	29

```

[5] chr1 32502644-32503436 + | JUNC00002871 48
... ..
[52] chr17 7578554-7579312 - | JUNC00041584 19
[53] chr17 7579590-7579700 - | JUNC00041585 35
[54] chr17 7579721-7579839 - | JUNC00041586 25
[55] chr17 7579940-7590695 - | JUNC00041587 29
[56] chr17 7591879-7591966 + | JUNC00041588 6

```

	part1_len	part2_len	part1_sta	part1_end	part2_sta	part2_end
	<numeric>	<numeric>	<numeric>	<numeric>	<numeric>	<numeric>
[1]	69	44	32479910	32479978	32495899	32495942
[2]	73	72	32495951	32496023	32497125	32497196
[3]	66	66	32497176	32497241	32498789	32498854
[4]	68	74	32498868	32498935	32502511	32502584
[5]	73	72	32502572	32502644	32503436	32503507
...
[52]	74	58	7578481	7578554	7579312	7579369
[53]	75	25	7579516	7579590	7579700	7579724
[54]	22	56	7579700	7579721	7579839	7579894
[55]	66	67	7579875	7579940	7590695	7590761
[56]	54	62	7591826	7591879	7591966	7592027

seqinfo: 6 sequences from an unspecified genome; no seqlengths

```

> load(system.file("extdata/refseq", "spllicemax.RData", package="customProDB"))
> load(system.file("extdata/refseq", "ids.RData", package="customProDB"))
> junction_type <- JunctionType(jun, spllicemax, txdb, ids)
> junction_type[10:19,]

```

	seqnames	start	end	width	strand	id	cov
10	chr2	48032846	48033343	498	+	JUNC00057364	12
11	chr2	48033497	48033591	95	+	JUNC00057365	10
12	chr2	48035386	48035468	83	-	JUNC00057367	9
13	chr5	112200429	112203101	2673	+	JUNC00080007	23
14	chr7	140706335	140710219	3885	-	JUNC00096159	15
15	chr9	86584295	86585077	783	-	JUNC00101237	14
16	chr9	86585246	86585652	407	-	JUNC00101239	171
17	chr9	86585734	86585812	79	-	JUNC00101240	80
18	chr9	86585827	86586188	362	-	JUNC00101241	121
19	chr17	37856564	37863243	6680	+	JUNC00043382	57

	part1_len	part2_len	part1_sta	part1_end	part2_sta	part2_end
10	65	28	48032782	48032846	48033343	48033370
11	43	64	48033455	48033497	48033591	48033654
12	75	53	48035312	48035386	48035468	48035520
13	74	67	112200356	112200429	112203101	112203167
14	53	66	140706283	140706335	140710219	140710284
15	60	72	86584236	86584295	86585077	86585148

16	69	73	86585178	86585246	86585652	86585724
17	68	16	86585667	86585734	86585812	86585827
18	16	75	86585812	86585827	86586188	86586262
19	74	74	37856491	37856564	37863243	37863316
			part1_type		part2_type	part1_exon
10	known	exon (same end)	known	exon (same start)		18
11	known	exon (same end)	known	exon (same start)		19
12	non-exon	region	non-exon	region		NA
13	non-exon	region	non-exon	region		NA
14	non-exon	region	non-exon	region		NA
15	non-exon	region	non-exon	region		NA
16	non-exon	region	non-exon	region		NA
17	non-exon	region	non-exon	region		NA
18	non-exon	region	non-exon	region		NA
19	known	exon (same end)	known	exon (same start)		61
			part2_exon		jun_type	tx_id_part1
10	19		known	junction	2	NM_000179
11	20		known	junction	2	NM_000179
12	NA	connect two	non-exon	region	<NA>	<NA>
13	NA	connect two	non-exon	region	<NA>	<NA>
14	NA	connect two	non-exon	region	<NA>	<NA>
15	NA	connect two	non-exon	region	<NA>	<NA>
16	NA	connect two	non-exon	region	<NA>	<NA>
17	NA	connect two	non-exon	region	<NA>	<NA>
18	NA	connect two	non-exon	region	<NA>	<NA>
19	62		known	junction	7	NM_004448
			ge_name_part1	tx_id_part2	tx_name_part2	ge_name_part2
10	MSH6		2	NM_000179		MSH6
11	MSH6		2	NM_000179		MSH6
12	<NA>		<NA>	<NA>		<NA>
13	<NA>		<NA>	<NA>		<NA>
14	<NA>		<NA>	<NA>		<NA>
15	<NA>		<NA>	<NA>		<NA>
16	<NA>		<NA>	<NA>		<NA>
17	<NA>		<NA>	<NA>		<NA>
18	<NA>		<NA>	<NA>		<NA>
19	ERBB2		7	NM_004448		ERBB2

```
> table(junction_type[, 'jun_type'])
```

```
connect a known exon and a region overlap with known exon      1
connect two non-exon region                                     9
known junction                                                  46
```

Except for 'known junction', all others are treated as putative novel junctions. Then all putative novel junctions are three-frame translated into peptides using the function `OutputNovelJun`. The reference genome sequence is required when using this function.

```
> outf_junc <- paste(tempdir(), '/test_junc.fasta', sep='')
> library('BSgenome.Hsapiens.UCSC.hg19')
> OutputNovelJun <- OutputNovelJun(junction_type, Hsapiens, outf_junc,
+     proteinseq)
```

4 Building database from multiple samples

We provide two functions to help generate a consensus database from multiple samples, especially for a group of similar samples. Even though deep sequencing reveals large scales of heterogeneity, consensus protein database consisting of the commonly expressed proteins and SNVs from a group of samples with similar genetic background will help identify subtype specific proteins.

4.1 Filtering based on transcript expression in multiple samples

The function `OutputsharedPro` outputs proteins with expression level above the cutoff in multiple samples. Unlike `Outputproseq` that uses vector as input, the function `Outputsharedpro` uses expression matrix as input. Users need to specify both the value of sample number and the RPKM cutoff when calling this function. Users could generate RPKM matrix from multiple BAM files as follows, or use RPKM matrix generated by other programs.

```
> path <- system.file("extdata/bams", package="customProDB")
> bamFile<- paste(path, '/', list.files(path, pattern="*bam$"), sep='')
> rpkm <- sapply(bamFile, function(x)
+     calculateRPKM(x, exon, proteincodingonly=TRUE, ids))
> #colnames(rpkm) <- c('1', '2', '3')
> #rpkm
> outfile <- paste(tempdir(), '/test_rpkm_share.fasta', sep='')
> pro <- OutputsharedPro(rpkm, cutoff=1, share_sample=2, proteinseq,
+     outfile, ids)
```

4.2 Variations occurred in multiple samples

The function `Multiple_VCF` outputs variations occurred in more than k samples, with the k specified by a user input parameter. When recurrent variations are identified, the following analysis is the same as shown in the 'Variation annotation' section.

```
> path <- system.file("extdata/vcfs", package="customProDB")
> vcfFiles<- paste(path, '/', list.files(path, pattern="*vcf$"), sep='')
> vcfs <- lapply(vcfFiles, function(x) InputVcf(x))
> shared <- Multiple_VCF(vcfs, share_num=2)
> shared
```

GRanges object with 62 ranges and 3 metadata columns:

seqnames	ranges	strand	REF	ALT	INDEL
<Rle>	<IRanges>	<Rle>	<character>	<character>	<logical>
test.chr1:32386425_T/C	chr1 32386425	*	T	C	FALSE
test.chr1:32507666_G/T	chr1 32507666	*	G	T	FALSE
test.chr1:32524459_A/C	chr1 32524459	*	A	C	FALSE
test.chr1:32622505_G/A	chr1 32622505	*	G	A	FALSE
test.chr12:25357574_CAA/C	chr12 25357574-25357576	*	CAA	C	TRUE
...
test.chr9:86593314_G/C	chr9 86593314	*	G	C	FALSE
test.chr9:86595070_C/T	chr9 86595070	*	C	T	FALSE
test.chr9:86595498_G/A	chr9 86595498	*	G	A	FALSE
test.chr5:112154737_T/A	chr5 112154737	*	T	A	FALSE
test.chr5:112175897_G/T	chr5 112175897	*	G	T	FALSE

 seqinfo: 7 sequences from an unspecified genome; no seqlengths

4.3 Junctions occurred in multiple samples

The function SharedJunc outputs splice junctions occurred in more than k samples, with the k specified by a user input parameter. When recurrent junctions are ready, the following analysis is the same as shown in the 'Splice junction analysis' section.

```
> path <- system.file("extdata/beds", package="customProDB")
> bedFiles<- paste(path, '/', list.files(path, pattern="*bed$"), sep='')
> juncs <- lapply(bedFiles, function(x) Bed2Range(x, skip=1, covfilter=5))
> sharedjun <- SharedJunc(juncs, share_num=2, ext_up=100, ext_down=100)
> sharedjun
```

GRanges object with 55 ranges and 8 metadata columns:

seqnames	ranges	strand	id	cov
<Rle>	<IRanges>	<Rle>	<character>	<numeric>
[1] chr1	32479978-32495899	+	JUNC1	8

```

[2] chr1 32496023-32497125 + | JUNC2 13
[3] chr1 32497241-32498789 + | JUNC3 20
[4] chr1 32498935-32502511 + | JUNC4 29
[5] chr1 32502644-32503436 + | JUNC5 48
... ..
[51] chr17 7578554-7579312 - | JUNC51 19
[52] chr17 7579590-7579700 - | JUNC52 35
[53] chr17 7579721-7579839 - | JUNC53 25
[54] chr17 7579940-7590695 - | JUNC54 29
[55] chr17 7591879-7591966 + | JUNC55 6

```

	part1_len	part2_len	part1_sta	part1_end	part2_sta	part2_end
	<numeric>	<numeric>	<numeric>	<integer>	<integer>	<numeric>
[1]	69	44	32479910	32479978	32495899	32495944
[2]	73	72	32495951	32496023	32497125	32497198
[3]	66	66	32497176	32497241	32498789	32498856
[4]	68	74	32498868	32498935	32502511	32502586
[5]	73	72	32502572	32502644	32503436	32503509
...
[51]	74	58	7578481	7578554	7579312	7579371
[52]	75	25	7579516	7579590	7579700	7579726
[53]	22	56	7579700	7579721	7579839	7579896
[54]	66	67	7579875	7579940	7590695	7590763
[55]	54	62	7591826	7591879	7591966	7592029

seqinfo: 6 sequences from an unspecified genome; no seqlengths

5 Two integrated functions

We provide two integrated functions for the one-step generation of customized databases.

`easyrun` generates a customized database from single sample.

```

> bamFile <- system.file("extdata/bams", "test1_sort.bam",
+                       package="customProDB")
> vcffile <- system.file("extdata/vcfs", "test1.vcf", package="customProDB")
> bedfile <- system.file("extdata", "junctions.bed", package="customProDB")
> annotation_path <- system.file("extdata/refseq", package="customProDB")
> outfile_path <- tempdir()
> outfile_name='test'
> easyRun(bamFile, RPKM=NULL, vcffile, annotation_path, outfile_path,
+        outfile_name, rpkm_cutoff=1, INDEL=TRUE, lablersid=TRUE, COSMIC=TRUE,
+        nov_junction=FALSE)

```

`easyrun_mul` generates a consensus database from multiple samples.

```

> bampath <- system.file("extdata/bams", package="customProDB")
> vcffile_path <- system.file("extdata/vcfs", package="customProDB")

```

```

> annotation_path <- system.file("extdata/refseq", package="customProDB")
> outfile_path <- tempdir()
> outfile_name <- 'mult'
> easyRun_mul(bam_path, RPKM_mtx=NULL, vcfFile_path, annotation_path, rpkm_cutoff=1,
+             share_num=2, var_shar_num=2, outfile_path, outfile_name, INDEL=TRUE,
+             lablersid=TRUE, COSMIC=TRUE, nov_junction=FALSE)

```

6 FASTA file format

The primary outputs of this package are FASTA files. Related information, such as gene symbol, gene description, variation position, change status, and corresponding dbSNP ID (if required and available), are included in the sequence header for interpretation of the search result. There are four types of headers in the FASTA file.

6.1 Normal proteins passing the expression cutoff

The header starts with RefSeq protein id, followed by RPKM/FPKM value in each sample (separated by ';') and the average RPKM/FPKM, RefSeq transcript id, gene symbol and description.

```

> outfile_path <- system.file("extdata/tmp", package="customProDB")
> readLines(file(paste(outfile_path, '/test_rpkm.fasta', sep=''), 'rt'), 1)

```

```
[1] ">NP_004439 |148172.2567|NM_004448|ERBB2|receptor tyrosine-protein kinase erbB-2 isoform a p
```

6.2 Variant Proteins induced by SNVs

The variation information, including variation position, amino acid change status and corresponding dbSNP ID (if available), is added to the RefSeq protein id followed by '_'. Different variations are separated by ','.

```

> readLines(file(paste(outfile_path, '/test_snv.fasta', sep=''), 'rt'), 1)

```

```
[1] ">NP_000170_T139I,E956D |15810.2686|NM_000179|MSH6|DNA mismatch repair protein Msh6"
```

6.3 Aberrant proteins induced by INDELS

The INDEL information is added to protein id followed by '_'. Here the INDELS position represents the position where this INDELS occurs in a coding sequence, not the position in protein sequence, which is different from proteins with SNVs.

```

> readLines(file(paste(outfile_path, '/test_indel.fasta', sep=''), 'rt'), 1)

```

```
[1] ">NP_004439_3508:CCC>C |148172.2567|NM_004448|ERBB2|receptor tyrosine-protein kinase erbB-2
```


6.4 Novel junction peptides

The junction id, genomic position, coverage (For single sample, it's the reads coverage. For multiple samples, it's the sample coverage), ORF, the source of left/right part and the junction type are added to the ID line of the FASTA file.

```
> readLines(file(paste(outfile_path, '/test_junc.fasta', sep=''), 'rt'), 1)
[1] ">JUNC00041588|6|ORF1|Junpos:18-19|+|NA|NA|connect two non-exon region"
```

7 Session Information

```
R version 3.5.1 Patched (2018-07-12 r74967)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.5 LTS
```

```
Matrix products: default
BLAS: /home/biocbuild/bbs-3.7-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.7-bioc/R/lib/libRlapack.so
```

locale:

```
[1] LC_CTYPE=en_US.UTF-8      LC_NUMERIC=C
[3] LC_TIME=en_US.UTF-8      LC_COLLATE=C
[5] LC_MONETARY=en_US.UTF-8  LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=en_US.UTF-8     LC_NAME=C
[9] LC_ADDRESS=C             LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
```

attached base packages:

```
[1] parallel stats4 stats graphics grDevices utils
[7] datasets methods base
```

other attached packages:

```
[1] BSgenome.Hsapiens.UCSC.hg19_1.4.0
[2] BSgenome_1.48.0
[3] Biostrings_2.48.0
[4] XVector_0.20.0
[5] GenomicFeatures_1.32.0
[6] customProDB_1.20.2
[7] biomaRt_2.36.1
[8] AnnotationDbi_1.42.1
[9] Biobase_2.40.0
[10] rtracklayer_1.40.3
[11] GenomicRanges_1.32.6
[12] GenomeInfoDb_1.16.0
[13] IRanges_2.14.10
```

[14] S4Vectors_0.18.3
[15] BiocGenerics_0.26.0

loaded via a namespace (and not attached):

[1] Rcpp_0.12.18	plyr_1.8.4
[3] compiler_3.5.1	prettyunits_1.0.2
[5] bitops_1.0-6	tools_3.5.1
[7] zlibbioc_1.26.0	progress_1.2.0
[9] digest_0.6.15	bit_1.1-14
[11] AhoCorasickTrie_0.1.0	RSQLite_2.1.1
[13] memoise_1.1.0	lattice_0.20-35
[15] pkgconfig_2.0.1	rlang_0.2.1
[17] Matrix_1.2-14	DelayedArray_0.6.4
[19] DBI_1.0.0	GenomeInfoDbData_1.1.0
[21] RMariaDB_1.0.6	httr_1.3.1
[23] stringr_1.3.1	hms_0.4.2
[25] bit64_0.9-7	grid_3.5.1
[27] R6_2.2.2	XML_3.98-1.14
[29] BiocParallel_1.14.2	magrittr_1.5
[31] blob_1.1.1	Rsamtools_1.32.2
[33] matrixStats_0.54.0	GenomicAlignments_1.16.0
[35] assertthat_0.2.0	SummarizedExperiment_1.10.1
[37] stringi_1.2.4	RCurl_1.95-4.11
[39] VariantAnnotation_1.26.1	crayon_1.3.4