Squirls Documentation

Release v2.0.0

Daniel Danis, Peter N Robinson

Mar 30, 2022

CONTENTS:

1 Set up Squirls **3**
- 1.1 Squirls downloadable resources . . . 3
- 1.2 Build Squirls from source . . . 4

2 Run Squirls **5**
- 2.1 `annotate-pos` - Annotate variant positions . . . 6
- 2.2 `annotate-csv` - Annotate variant positions stored in a CSV file . . . 6
- 2.3 `annotate-vcf` - Annotate variants in a VCF file . . . 7
- 2.4 `precalculate` - Precalculate SQUIRLS scores . . . 8

3 Output formats **11**
- 3.1 HTML output format . . . 11
- 3.2 VCF output format . . . 11
- 3.3 CSV/TSV output format . . . 12

4 Result interpretation **13**
- 4.1 Variant categories . . . 14
- 4.2 Figure types . . . 16

5 Squirls anatomy **19**
- 5.1 Splice features . . . 19
- 5.2 Random forest estimators . . . 20
- 5.3 Logistic regression . . . 20
- 5.4 Glossary . . . 21

6 Use Squirls as a library **23**
- 6.1 Install Squirls modules into your local Maven repository . . . 23
- 6.2 Bootstrap Squirls . . . 23
- 6.3 Spring Boot application . . . 24

Index **27**

Super-quick Information Content and Random Forest Learning for Splice Variants.

This application performs prediction of deleteriousness of genomic variants with respect to mRNA splicing.

CHAPTER

ONE

SET UP SQUIRLS

Squirls is a desktop Java application that requires several external files to run. This document explains how to download these files and prepare to run Squirls.

Note: Squirls is written with Java version 11 and will run and compile under Java 11+.

1.1 Squirls downloadable resources

There are several external files that must be downloaded prior running Squirls.

1.1.1 Prebuilt Squirls executable

To download the prebuilt Squirls JAR file, go to the Releases section on the Squirls GitHub page and download the latest precompiled version of Squirls.

1.1.2 Squirls database files

Squirls database files are available for download from the following locations:

Version	Genome build	URL	Size
2103	hg19/GRCh37	https://storage.googleapis.com/squirls/2103_hg19.zip	~10.5 GB for download, ~15 GB unpacked
2103	hg38/GRCh38	https://storage.googleapis.com/squirls/2103_hg38.zip	~11.1 GB for download, ~16.5 GB unpacked
2203	hg19/GRCh37	https://storage.googleapis.com/squirls/2203_hg19.zip	~9.5 GB for download, ~11.9 GB unpacked
2203	hg38/GRCh38	https://storage.googleapis.com/squirls/2203_hg38.zip	~9.9 GB for download, ~12.2 GB unpacked

Note: The `2103` works with Squirls `v1.0.0`, the `2203` works with Squirls `v2.0.0`.

Use `curl` or `wget` utilities to download the files from command line:

```
$ wget https://storage.googleapis.com/squirls/2203_hg38.zip
or
$ curl --output 2203_hg38.zip https://storage.googleapis.com/squirls/2203_hg38.zip
```

Alternatively, use a GUI FTP client such as FileZilla.

After the download, unzip the archive(s) content into a folder and note the folder path.

1.2 Build Squirls from source

As an alternative to using prebuilt Squirls JAR file, the Squirls JAR file can also be built from Java sources.

Run the following commands to download Squirls source code from GitHub repository and to build Squirls JAR file:

```
$ git clone https://github.com/TheJacksonLaboratory/Squirls
$ cd Squirls
$ ./mvnw package
```

Note: To build Squirls from sources, JDK 11 or better must be available in the environment

After the successful build, the JAR file is located at `squirls-cli/target/squirls-cli-2.0.0.jar`.

To verify that the building process went well, run:

```
$ java -jar squirls-cli/target/squirls-cli-2.0.0.jar --help
```

CHAPTER

TWO

RUN SQUIRLS

Squirls is a command-line Java tool that runs with Java version 11 or higher.

Before using Squirls, you must setup Squirls as describe in the *Set up Squirls* section.

Squirls provides four commands to annotate variants in different input formats:

- `annotate-pos` - quickly annotate a couple of variants, e.g. `chr9:136224694A>T`
- `annotate-csv` - annotate variants stored in a CSV file
- `annotate-vcf` - annotate variants in VCF file
- `precalculate` - precalculate SQUIRLS scores for provided regions and store the results in a compressed VCF file

In the examples below, we assume that `$SQUIRLS_DATA` points to Squirls resource directory obtained by unzipping the archive, as described in the *Set up Squirls* section:

```
# e.g.
SQUIRLS_DATA=path/to/squirls/data
```

where:

```
path/to/squirls/data
  ├── assembly_report.txt
  ├── genome.fa
  ├── genome.fa.dict
  ├── genome.fa.fai
  ├── phylop.bw
  ├── squirls.mv.db
  ├── tx.ensembl.ser
  ├── tx.refseq.ser
  └── tx.ucsc.ser
```

2.1 `annotate-pos` - Annotate variant positions

The easiest way to quickly calculate Squirls scores for a couple of variants is to use the `annotate-pos` command:

```
java -jar squirls-cli.jar annotate-pos -d $SQUIRLS_DATA "chr9:136224694A>T"
↪"chr3:52676065CA>C"
```

Note: Do not forget to surround the variants with double quotes (`"chr9:136224694A>T"` and *not* `chr9:136224694A>T`) to prevent interpretation of the > as a shell operator.

Note: Both `"chr1:12345A>T"` and `"chr1:g.12345A>T"` notations are supported.

The command above generates the following terminal output:

```
...
2000-01-01 12:34:56.309  INFO 12345 --- [           main] o.m.s.c.c.a.AnnotatePosCommand␣
↪          : Analyzing 2 change(s): `chr9:136224694A>T, chr3:52676065CA>C`

chr9:136224694A>T    pathogenic    0.970    NM_001278928.1=0.970115;NM_017503.4=0.970115
chr3:52676065CA>C    neutral       0.007    NM_018313.4=0.006350;XM_005265275.1=0.006350;
↪XM_005265276.1=0.006350;XM_005265277.1=0.006350;XM_005265278.1=0.006350;XM_005265279.
↪1=0.006350;XM_005265280.1=0.006350;XM_005265281.1=0.006350;XM_005265282.1=0.006350;XM_
↪005265283.1=0.006350;XM_005265284.1=0.006350;XM_005265285.1=0.006350;XM_005265286.1=0.
↪006350;XM_005265287.1=0.006350;XM_005265288.1=0.006350;XM_005265289.1=0.006350;XM_
↪005265290.1=0.006350;XM_005265291.1=0.006350;XM_005265292.1=0.006350
```

Squirls reports scores in four columns:

- variant position
- variant interpretation, either *pathogenic* or *neutral*
- maximum Squirls pathogenicity prediction rounded up to 3 significant digits
- Squirls pathogenicity predictions calculated for each transcript the variant overlaps with

2.2 `annotate-csv` - Annotate variant positions stored in a CSV file

To annotate more than just a few variant positions, it may be more convenient to use the `annotate-csv` command.

Let's run the `annotate-csv` command to annotate four variants stored in the example.csv file (an example CSV file with 4 variants stored in Squirls repository):

```
java -jar squirls-cli.jar annotate-csv -d $SQUIRLS_DATA example.csv path/to/output/file
```

Squirls reads the variants and stores the scores into `path/to/output/file.html` file. The *HTML* is the default output format, see *Output formats* section for more details.

2.2.1 Mandatory arguments

The `annotate-csv` command requires three mandatory *arguments*:

- `-d | --data-directory` - path to Squirls data directory
- path to CSV file with variants
- output prefix for the generated files

2.2.2 Optional arguments

In addition to the mandatory arguments, Squirls allows to fine tune the annotation using optional arguments:

- `--all-transcripts` - report Squirls scores for all overlapping transcripts. Default: `false`
- `--compress` - compress the output files using `gzip` (*tabular*) or `bgzip` (*VCF*). The option has no effect on *HTML* output format. Default: `false`
- `-f | --output-format` - comma separated list of *Output formats*. Use `html,vcf,csv,tsv` to store results in all output formats. Default: `html`
- `-n, --n-variants-to-report` - number of most pathogenic variants to include in *HTML* report. The option has no effect on *VCF* output format. Default: `100`
- `--report-features` - include Squirls features into the output. Default: `false`
- `-t | --transcript-source` - transcript source to use. Choose one of `{REFSEQ, ENSEMBL, UCSC}`. Default: `REFSEQ`
- `--threads` - process variants on *n* threads. Default: `2`

2.3 `annotate-vcf` - Annotate variants in a VCF file

The aim of this command is to annotate variants in a VCF file and to store the results in one or more *Output formats*.

To annotate variants in the example.vcf file (an example VCF file with 6 variants stored in Squirls repository), run:

```
$ java -jar squirls-cli.jar annotate-vcf -d $SQUIRLS_DATA example.vcf path/to/output/file
```

After the annotation, the results are stored at `path/to/output/file.html`.

2.3.1 Mandatory arguments

The `annotate-vcf` command requires three mandatory arguments:

- `-d | --data-directory` - path to Squirls data directory
- path to the VCF file with variants
- output prefix for the generated files

2.3.2 Optional arguments

In addition to the mandatory arguments, Squirls allows to fine tune the annotation using optional arguments:

- `--all-transcripts` - report Squirls scores for all overlapping transcripts. Default: `false`
- `--compress` - compress the output files using `gzip` (*tabular*) or `bgzip` (*VCF*). The option has no effect on *HTML* output format. Default: `false`
- `-f | --output-format` - comma separated list of *Output formats*. Use `html,vcf,csv,tsv` to store results in all output formats. Default: `html`
- `-n, --n-variants-to-report` - number of most pathogenic variants to include in *HTML* report. The option has no effect on *VCF* output format. Default: `100`
- `--report-features` - include Squirls features into the output. Default: `false`
- `-t | --transcript-source` - transcript source to use. Choose one of `{REFSEQ, ENSEMBL, UCSC}`. Default: `REFSEQ`
- `--threads` - process variants on *n* threads. Default: `2`

2.4 `precalculate` - Precalculate SQUIRLS scores

We do not provide a tabular file with precalculated scores for all possible genomic variants. Instead, we provide a command for precalculating the scores for your genomic regions of interest. This command precalculates Squirls scores for all possible variants (including INDELs up to specified length) and stores the scores in a compressed VCF file.

Example:

```
$ java -jar squirls-cli.jar precalculate -d $SQUIRLS_DATA CM000669.1:44187000-44187600␣
↪CM000669.1:44186000-44186500
```

The command computes scores for two regions, each region encompassing an exons of the *GCK* gene plus some neighboring intronic sequence. `SQUIRLS` recognizes *GenBank*, *RefSeq*, *UCSC*, and *simple* (`1`, `2`, ..., `X`, `Y`, `MT`) contigs accessions.

The region coordinates must be provided using *zero-based* coordinates where the start position is *not* part of the region.

By default, SQUIRLS generates all possible SNVs for the bases of the region, including deletion of the base. For example, a region r spanning `ctg1:3-5` of a 10bp-long reference contig `ctg1`:

```
>ctg1
ACGTACGTAC
```

yields the variants:

chrom	pos	SNVs	DELs	INSs
ctg1	4	T>A, T>C, T>G	T>	N/A
ctg1	5	A>C, A>G, A>T	A>	N/A

the annotated variants are stored in a compressed VCF file named `squirls-scores.vcf.gz` that is by default stored in the current working directory.

Please note that the VCF file *not* sorted. Please sort and index the VCF file yourself, e.g. by running:

```
bcftools sort squirls-scores.vcf.gz | bgzip -c > squirls-scores.sorted.vcf.gz
tabix squirls-scores.sorted.vcf.gz
```

2.4.1 Mandatory arguments

The only mandatory argument for `precalculate` is `-d` to provide path to Squirls data directory. Following that, `0..n` region definitions, e.g. `CM000669.1:44187000-44187600`, `CM000669.1:44186000-44186500` can be provided.

2.4.2 Optional arguments

There are several options to adjust:

- `-i | --input` - path to a BED file with the target regions. Lines starting with `#` are ignored. See example regions.bed
- `--individual` - if the flag is present, predictions with respect to all overlapping transcripts will be stored within the *INFO* field.
- `-l | --max-length` - maximum length of the generated variants on the reference genome, see *Variant generation* below (Default: `1`)
- `-o | --output` - path to VCF file where to write the results. The VCF output is compressed, so we recommend to use `*.vcf.gz` suffix. (Default: `squirls.scores.vcf.gz`)
- `-t | --transcript-source` - transcript source to use. Choose one of `{REFSEQ, ENSEMBL, UCSC}`. Default: `REFSEQ`
- `--threads` - number of threads to use for calculating the scores. (Default: `2`)

2.4.3 Parallel processing

When predicting the scores, each region is handled by a single thread, while at most `--threads` threads being used for prediction at the same time. Therefore, to fully leverage the parallelism offered by modern multi-core CPUs, we recommend to split large regions into several smaller ones.

2.4.4 Variant generation

The default value of the `-l, --max-length` parameter is set to `1`. As explained above, the parameter controls the length of the generated variants. However, length can be set to any positive integer, leading to calculation of scores for variants of different lengths.

Using the region r and the contig `ctg1` defined above, setting `-l` to `2` will calculate scores for variants:

Table 1: The variant generation pattern

chrom	pos	SNVs	DELs	INSs
ctg1	4	T>A, T>C, T>G	T>, TA>T	T>TA, T>TC, T>TG, T>TT
ctg1	5	A>C, A>G, A>T	A>	A>AA, A>AC, A>AG, A>AT

Note: The number of possible variants grows exponentially with increasing of the `--length` value. This can lead to substantial run times and to extending your computational budget. Use at your own risk ;)

CHAPTER

THREE

OUTPUT FORMATS

Squirls supports storing results in 4 output formats: *HTML*, *VCF CSV*, and *TSV*. Use the `-f | --output-format` option to select one or more of the desired output formats (e.g. `-f html,vcf`). *HTML* report is generated by default.

3.1 HTML output format

Squirls creates an *HTML* file with the analysis summary and with variants sorted by Squirls score in descending order. The number of the reported variants is adjusted by the `-n | --n-variants-to-report` option. See the *Result interpretation* section to learn more about the content of the *HTML* report.

3.2 VCF output format

When including `vcf` into the `-f` option, a VCF file with all input variants is created. The annotation process adds a novel *FILTER* and *INFO* field(s) to each variant that overlaps with one or more transcript region:

- `SQUIRLS` - a *FILTER* flag indicating that the variant is considered to have a deleterious effect on >=1 overlapping transcript
- `SQUIRLS_SCORE` - an *INFO* string containing maximum SQUIRLS scores for the overlapping transcripts.
- `SQUIRLS_TXS` - an *INFO* string containing SQUIRLS scores for each variant-transcript combination. The field is present when `--all-transcripts` option is specified.

For the example variant `chr1:1234C>A,G`, the *INFO* field will contain the following:

```
SQUIRLS_SCORE=0.988654;SQUIRLS_TXS=A|NM_123456.1=0.988654|ENST00000987654.1=0.988654
SQUIRLS_SCORE=0.330112;SQUIRLS_TXS=G|NM_12356.1=0.330112|ENST00000987654.1=0.330112
```

Multi-allelic variants are broken down into separate records and processed individually. Predictions with respect to the overlapping transcripts are separated by a pipe (`|`) symbol.

Note: The `-n` option has no effect for the *VCF* output format.

Note: The `vcfgz` output format has been deprecated. Use `--compress` and `-f vcf` options to get results as a compressed VCF file.

3.3 CSV/TSV output format

To write *n* most deleterious variants into a *CSV* (or *TSV*) file, use `csv` (`tsv`) in the `-f` option.

The results are written into a tabular file with the following columns:

Table 1: Tabular output

chrom	pos	ref	alt	gene_symbol	tx_accession	interpretation	squirls_score
chr3	165504107	A	C	*BCHE*	NM_000055.2	pathogenic	0.99997203304
…	…	…	…	…	…	…	…

Note: Use `--report-features` and/or `--all-transcripts` options to add columns with Squirls features and/or Squirls scores for all overlapping transcripts.

CHAPTER

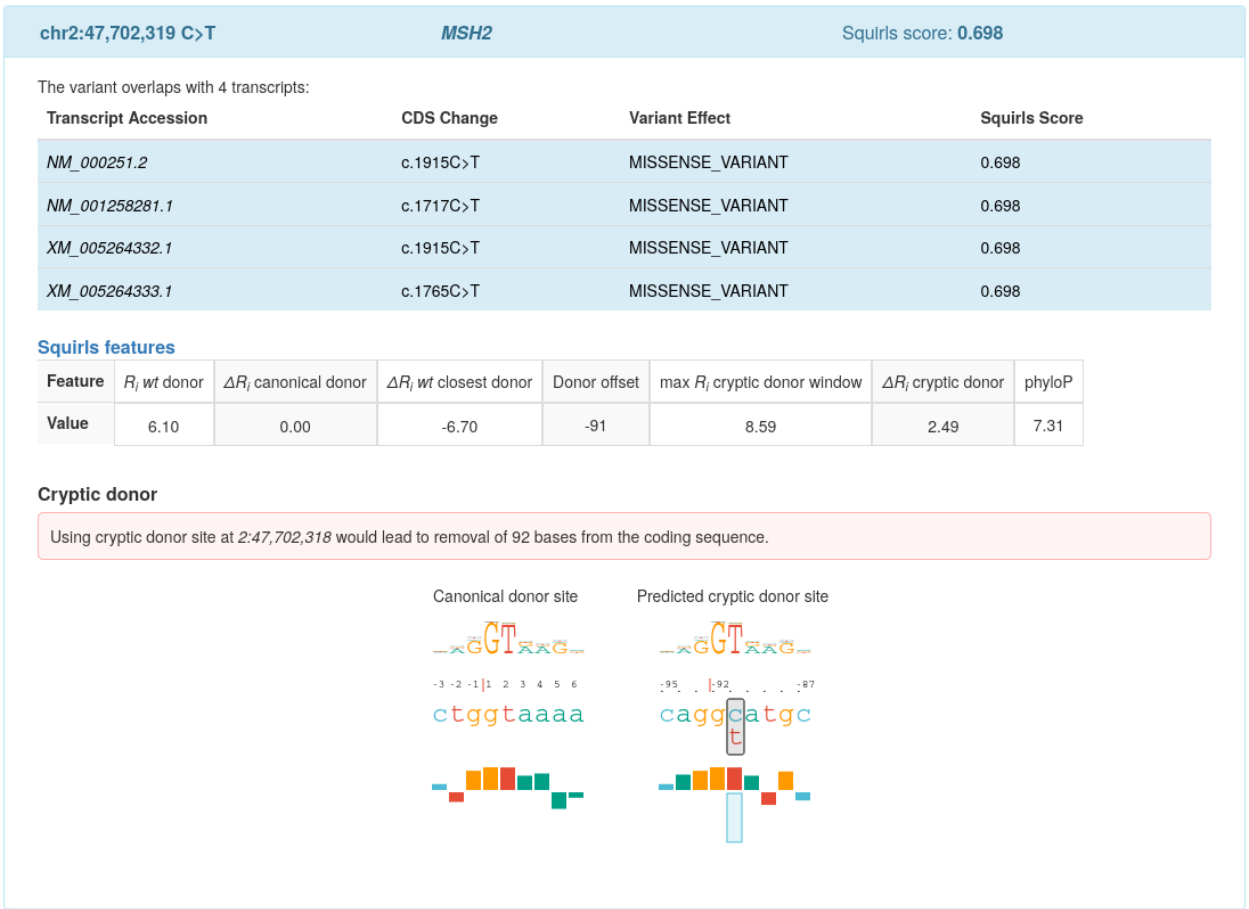
FOUR

RESULT INTERPRETATION

When designing Squirls, our motivation was to create an *interpretable* algorithm for identification of splice deleterious variants. We addressed this goal by limiting ourselves to use a small set of biologically interpretable attributes for learning how to separate splice deleterious variants from neutral polymorphisms.

To help with interpretation of a variants that has been marked as splicing deleterious, we developed HTML result format that presents all available information in a visually attractive way. When reporting variants, we sort the variants by Squirls score in descending order - the most deleterious variants are placed on the top of the list.

The following picture shows an example output for variant *NM_000251.2:c.1915C>T* (*chr2:47702319C>T*), predicted to create a novel cryptic donor site (click for a full size image):



Squirls summarizes the information available for the variant in a box. The header of the box contains three fields:

- **Variant coordinates:** Summary of variant's location on used genomic assembly, e.g. *chr2:47,702,319 C>T*
- **HGVS gene symbol:** e.g. *MSH2*
- **Squirls score:** maximum predicted splicing pathogenicity score, e.g. *0.698*

The box content consists of three sections:

- **Variant effects on overlapping transcripts:** Squirls uses Jannovar to predict effect of the variant on the transcript, and represents the effect using HGVS Sequence Variant Nomenclature. Then, the effects and Squirls scores calculated for the overlapping transcripts are listed in a table. The transcript accessions corresponding to the maximum Squirls score are emphasized by blue color (all 4 rows in the above example).
- **Splicing features:** Squirls shows splicing feature values in a table, see *Splice features* section for explanations.
- **Figures:** Squirls presents SVG graphics that show the most important predicted effects.

4.1 Variant categories

When generating the HTML report, Squirls makes a decision about which set of figures to make for a given variant. Based on the most likely splice altering-pathomechanism, the variant is assigned into one of four categories that dictate which figures will be generated:

4.1.1 Canonical donor

Squirls creates a *Sequence trekker* for variants that are likely to disrupt a canonical donor site and to lead to either exon skipping or to utilization of a weaker cryptic site located nearby. In addition, Squirls plots the position of ΔR_i canonical donor in the distribution of random changes to sequences of the same length (see *\Delta R_i score distribution*).

Sequence trekker summarizes the sequence context and the impact of the variant on the binding site. Let's consider a `A>T` variant located at position 4 with respect to exon/intron border. The *ref* allele represented by normal `a` character is substituted by `t`. The change location is highlighted by a black box. The unfavourable contact between spliceosome and the *alt* allele is represented by drawing the red bar corresponding to `t` upside down, and by drawing box for the `a` of the ref allele upwards:

The *distribution of random changes* shows position of ΔR_i canonical donor of the particular variant in the *distribution of random changes* to sequence of the same length. When ΔR_i score is positive and close to the distribution edge, then the variant reduces the sequence information and the resulting allele is less likely to be recognized as a donor site.

4.1.2 Cryptic donor

For a variant predicted to create a cryptic donor site, we generate *Sequence trekker*s to compare the candidate site to the closest canonical donor site.

Let's consider the case of a missense variant `chr2:47,702,319C>T` (*NM_000251.2: c.1915C>T*) reported by Liu et al., 1994 (Table 2, Kindred JV). The variant is located 91 bases upstream of the canonical donor site and introduces a cryptic donor site into coding sequence of the *MSH2* gene.

Squirls generates a Sequence trekker for the *Canonical donor site*, the site features $R_i = 6.10$ bits:

The candidate cryptic donor site consists of the following alleles, where (again) the change `C>T` is located 91 bases upstream of the canonical donor site:

- `CAGGCATGC` (*ref*)
- `CAGGTATGC` (*alt*)

Squirls represents alleles of the *Predicted cryptic donor site* by the following sequence trekker:

The `T` base introduced by the variant increases R_i of the site by `2.49` bits to $R_{i\ alt} = 8.59$ bits. The increase is graphically represented by drawing an upside-down blue box for `c` (an unfavorable contact), and upwards pointing box for `t` to represent a favourable interaction between the *alt* allele and the spliceosome.

4.1.3 Canonical acceptor

For a variant that is likely to disrupt a canonical acceptor site, we create *Sequence trekker*, and we plot position of ΔR_i canonical acceptor in the distribution of random changes to sequence of the same length (see *\Delta R_i score distribution*).

Sequence trekker shows relative importance of the individual positions of the acceptor site and the impact of the variant on the site.

We also show position of ΔR_i canonical acceptor in the *distribution of random changes to sequence of the same length*. Here, the ΔR_i score will be positive if the variant reduces the sequence information and if the variant is likely to reduce recognition of the acceptor site.

Additionally, the variants that introduce `(Y)AG` sequence into the *AG-exclusion zone* might lead to exon skipping or to cryptic splicing (see Wimmer et al., 2020). The info regarding violation of the *AG-exclusion zone* is located in the splice features table.

4.1.4 Cryptic acceptor

For the variant that leads to creation of a cryptic acceptor site, Squirls generates the same graphics as for the cryptic donor sites - two *Sequence trekker*s to compare the candidate cryptic acceptor site to the closest canonical acceptor site.

Let's consider the case of the variant `chr1:16,451,824C>T` (*NM_004431.3: c.2826-9G>A*) located 9 bases upstream of the canonical acceptor site that introduces a cryptic acceptor site into the *EPHA2* gene (Zhang et al., 2009).

The first sequence walker represents the *Canonical acceptor site*, located 9 bp downstream of the variant site:

The *alt* allele of the canonical site has $R_i = 7.26$ bits.

Then, the *Predicted cryptic acceptor site* consists of these alleles:

- `ctaactctccctctctccctccc`**`g`**`gCC` (*ref*)
- `ctaactctccctctctccctccc`**`a`**`gCC` (*alt*)

The corresponding sequence trekker is:

The cryptic acceptor site features $R_i = 11.98$ bits. Sequence trekker depicts the change by drawing the orange box for `g` upside down (an unfavorable contact), and by drawing the green box for `a` upwards (a favourable interaction). The changed position is emphasized by a black box on the sequence ruler.

Note: Please note that Squirls uses the *alt* allele to generate sequences necessary to draw sequence trekkers for *both* canonical site and cryptic site. This is because we are interested in comparing the sites and not the individual alleles.

4.2 Figure types

This section provides detailed explanations of the figures we generate for the variants, as described in the previous section. We consider these figures to be the most helpful for clinical interpretation of the splice variants.

4.2.1 Sequence ruler

Sequence rulers are SVG graphics that show the sequence of the donor or acceptor site, mark the intron-exon boundary (red vertical bar), and show the position of any alternate bases that diverge from the reference sequence (black rectangle).

Note: We intentionally omit the position *zero* in sequence rulers, to make the result interpretation easier for biologists, who are more comfortable with numbering of intronic/exonic bases that starts at *one*.

However, please note that the correct numbering scheme starts at *zero*. Please visit website of professor Tom Schneider, where among Pitfalls in Information Theory he also explains the correct numbering scheme for sequences.

4.2.2 Sequence logo

In 1990, Tom Schneider introduced Sequence logos as a way of graphically displaying consensus sequences. The characters representing the sequence are stacked on top of each other for each position in the aligned sequences. The height of each letter is made proportional to its frequency, and the letters are sorted so the most common one is on top. The height of the entire stack is then adjusted to signify the information content of the sequences at that position. From these *sequence logos*, one can determine not only the consensus sequence but also the relative frequency of bases and the information content (measured in bits) at every position in a site or sequence. The logo displays both significant residues and subtle sequence patterns (Nucleic Acids Res 1990;18:6097-100).

4.2.3 Sequence walker

Tom Schneider introduced *Sequence walkers* in 1995 as a way of graphically displaying how binding proteins and other macromolecules interact with individual bases of nucleotide sequences. Characters representing the sequence are either oriented normally and placed above a line indicating favorable contact, or upside-down and placed below the line indicating unfavorable contact. The positive or negative height of each letter shows the contribution of that base to the average sequence conservation of the binding site, as represented by a sequence logo (Nucleic Acids Res 1997;25:4408-15).

In 1998, Peter Rogan introduced the application of individual information content and *Sequence walkers* to splicing variants (Hum Mutat 1998;12:153-71).

Note: Squirls does *not* generate *Sequence walker* graphics for sequences. Instead, Squirls uses *Sequence trekker*, a graphics based on *Sequence walker* that is explained in the next section.

4.2.4 Sequence trekker

Squirls combines the sequence ruler, sequence logo, and sequence walker into a new figure that we call *Sequence trekker* (because a trek goes further than a walk).

On top of that, sequence trekker integrates the information regarding the reference and the alternate alleles into a single graphics.

Sequence trekker replaces the letters used in Sequence walker by bars. The bars are colored using the standard *"Sanger"* color conventions. Similarly to Sequence walker, the bar orientation indicates favorable (up) or unfavorable (down) contacts. The bar height shows the contribution of that base to the average sequence contribution of the binding site. To present data for reference and alternate alleles in the same time, the bar corresponding to the reference allele at the variant position is drawn with a semi-transparent fill.

In many disease-associated variants, the bar corresponding to the reference base will be positioned upright and the alternate base will be facing down.

4.2.5 ΔR_i score distribution

The individual sequence information of a sequence $R_{i\ ref}$ and an alternate sequence $R_{i\ alt}$ are presented using the *Sequence trekker*. This graphic shows the value of the difference between the reference sequence and an alternate sequence as well as the distribution of random changes to sequences of the same length. A variant that reduces the sequence information is associated with a positive ΔR_i score ($\Delta R_i = 8.96$ bits in this case).

CHAPTER

FIVE

SQUIRLS ANATOMY

This document outlines the anatomy of the Squirls model, specifically, how a Squirls score is calculated for a variant.

As outlined in the Squirls manuscript, Squirls consists of two *random forest estimators* (one for the donor and the other for the acceptor site) followed by a *logistic regression*. Both random forests calculate predictions for a single variant, the predictions are subsequently transformed by the logistic regression into the final Squirls score. For a single variant, Squirls calculates scores for all overlapping transcripts.

5.1 Splice features

The first step of the prediction process is the calculation of a small set of interpretable numeric features for machine learning. The features are then passed to random forest estimators. The random forests use different feature subsets to perform the prediction.

5.1.1 Donor site-specific estimator

This section lists the features used by the donor random forest estimator:

R_i ***wt* donor** *Information content* (R_i) of the closest canonical donor site.

ΔR_i **canonical donor** Difference between R_i of *ref* and *alt* alleles of the closest donor site (0 bits if the variant does not affect the site).

ΔR_i ***wt* closest donor** Difference between R_i of the closest donor and the downstream (3') donor site (0 bits if this is the donor site of the last intron).

Donor offset Number of 1 bp-long steps required to pass through the exon/intron border of the closest *donor* site. The number is negative if the variant is located upstream from the border.

max R_i **cryptic donor window** Maximum R_i of sliding window of all 9 bp sequences that contain the *alt* allele.

ΔR_i **cryptic donor** Difference between max R_i of sliding window of all 9 bp sequences that contain the *alt* allele and R_i of *alt* allele of the closest donor site.

phyloP Mean phyloP score of the *ref* allele region.

5.1.2 Acceptor site-specific estimator

These are the features used by the acceptor random forest estimator:

ΔR_i **canonical acceptor** Difference between *information content* (R_i) of *ref* and *alt* alleles of the closest acceptor site (`0` if the variant does not affect the acceptor site).

ΔR_i **cryptic acceptor** Difference between max R_i of sliding window applied to *alt* allele neighboring sequence and R_i of *alt* allele of the closest acceptor site.

Creates `AG` in *AGEZ* 1 if the variant creates a novel `AG` di-nucleotide in *AGEZ*, `0` otherwise.

Creates `YAG` in *AGEZ* 1 if the variant creates a novel `YAG` tri-nucleotide in *AGEZ* where `Y` stands for a pyrimidine derivative (cytosine or thymine), `0` otherwise (see Wimmer et al., 2020).

Acceptor offset Number of 1 bp-long steps required to pass through the exon/intron border of the closest *acceptor* site. The number is negative if the variant is located upstream from the border.

Exon length Number of nucleotides spanned by the exon where the variant is located in (`-1` for non-coding variants that do not affect the canonical donor/acceptor regions).

ESRSeq Estimate of impact of random hexamer sequences on splicing efficiency when inserted into five distinct positions of two different minigene exons obtained by in vitro screening (Ke et al., 2011).

SMS Estimated splicing efficiency for 7-mer sequences obtained by saturating a model exon with single and double base substitutions (saturation mutagenesis derived splicing score, Ke et al., 2018).

phyloP Mean phyloP score of the *ref* allele region.

Note: The values of all features based on information theory are in *bits* of information

---

5.2 Random forest estimators

Squirls algorithm consists of two *random forest estimators* trained to recognize variants that change splicing of a donor or acceptor site. Given a set of splice features, the estimator calculates deleteriousness for the corresponding variant.

If a feature cannot be calculated for a variant, the missing feature value is imputed by a median feature value that was observed during training of the model.

The random forest consists of n decision trees that use the splice features to make a decision regarding deleteriousness of the variant in question.

5.3 Logistic regression

Squirls uses logistic regression as the final step to integrate outputs of the donor and acceptor random forests into the final Squirls score.

5.4 Glossary

AGEZ AG-exclusion zone, the sequence between the branch point and the proper 3'ss `AG` that is devoid of `AG`s, as defined by Gooding et al., 2006

Information content Individual information content of a nucleotide sequence $R_i(j)$ that is related to thermodynamic entropy and the free energy of binding. R_i can also be used to compare sites with one another.

CHAPTER

SIX

USE SQUIRLS AS A LIBRARY

Squirls is implemented as a modular Java application to allow to be used both as a standalone application and as a library. This document explains how to use Squirls as a component/library, to predict deleteriousness of variants on splicing within a larger application for analysis of genome variants.

The following sections describe how to use Squirls as a module in other Java tool.

6.1 Install Squirls modules into your local Maven repository

As the first step, Squirls needs to be installed into the local Maven repository. The installation requires JDK 11 or better to be present in the environment. Squirls uses the amazing Maven Wrapper to build the project:

```
git clone https://github.com/TheJacksonLaboratory/Squirls
cd Squirls
./mvnw install
```

After the successful build, Squirls artifacts are installed in your local Maven repository and, therefore, available for using as dependencies of other projects.

6.2 Bootstrap Squirls

Squirls can be easily used as a module within a larger Java application. To add Squirls into your codebase, first include the `squirls-bootstrap` as a dependency, e.g. by adding the following into the `pom.xml` of your Maven project:

```
<dependency>
  <groupId>org.monarchinitiative.squirls</groupId>
  <artifactId>squirls-bootstrap</artifactId>
  <version>${project.version}</version>
</dependency>
```

Note: Replace `${project.version}` placeholder with an actual Squirls release, i.e. `2.0.0`.

The programmatic initialization of Squirls is very straightforward:

```
Path dataDirectory = ... ; // path to Squirls data directory
SquirlsProperties squirlsProperties = SimpleSquirlsProperties.builder().build();
```

(continues on next page)

(continued from previous page)

```
SquirlsOptions squirlsOptions = SquirlsOptions.of(FeatureSource.REFSEQ); // ENSEMBL and␣
↪UCSC are available too
SquirlsConfigurationFactory squirlsFactory = SquirlsConfigurationFactory.
↪of(dataDirectory, squirlsProperties, squirlsOptions);
```

`Squirls` provides high-level API for access to the reference genome, to calculate the splice features, and the Squirls score for given variant.

This is a minimal example for annotating the variant *NM_000251.2:c.1915C>T* (*chr2:47702319C>T*), predicted to create a novel cryptic donor site in *MSH2*:

```
VariantSplicingEvaluator variantEvaluator = squirls.variantSplicingEvaluator();
GenomicAssembly assembly = squirls.squirlsDataService().genomicAssembly();
VcfConverter vcfConverter = new VcfConverter(assembly, VariantTrimmer.
↪rightShiftingTrimmer(VariantTrimmer.retainingCommonBase()));

// chr2       47702319        MSH2_cryptic_donor      C       T       1000    .       ␣
↪AC=2;AF=1      GT      1/1
Variant variant = vcfConverter.convert(assembly.contigByName("chr2"), "MSH2_cryptic_donor
↪", 47_702_319, "C", "T");
SquirlsResult squirlsResult = variantEvaluator.evaluate(variant);

assertThat(squirlsResult.isPathogenic(), is(true));
assertThat(squirlsResult.maxPathogenicity(), is(closeTo(0.698, 1E-5)));
```

Use `SquirlsResult` for the downstream variant analysis.

6.3 Spring Boot application

Squirls includes a `squirls-spring-boot-starter` module for including Squirls into an application that uses Spring boot framework. Using the starter requires even less lines of code than using `squirls-bootstrap`.

> **Warning:** The Spring Boot module is deprecated and will be removed in `v3.0.0`. Use the `squirls-bootstrap` module instead.

To use Squirls in a Spring boot app, add the following dependency into your `pom.xml`:

```
<dependency>
  <groupId>org.monarchinitiative.squirls</groupId>
  <artifactId>squirls-spring-boot-starter</artifactId>
  <version>${project.version}</version>
</dependency>
```

After adding the dependency, Spring configures Squirls beans, as long as you define the following `\@Bean` in your `@Configuration` class:

```
@Bean
public Path squirlsDataDirectory() {
  return Paths.get("path/to/squirls/data");
}
```

`squirls-spring-boot-starter` provides several high-level `@Beans`:

- `SquirlsDataService` to get transcripts that overlap with given coordinates, to fetch reference genome sequence, etc.
- `SplicingAnnotator` to calculate splice features for a variant
- `SquirlsClassifier` to calculate Squirls score given splice features, and
- `VariantSplicingEvaluator` to perform everything described above within a single method call.

`squirls-cli` shows an example how to use `squirls-spring-boot-starter`.

INDEX

A

AGEZ, 21

I

Information content, 21