gepyto Documentation

Release 0.10.1

Marc-André Legault, Louis-Philippe Lemieux Perreault

October 26, 2015

Contents

Contents:

CHAPTER 1

Python Objects (`structures`)

1.1 Genes

class `gepyto.structures.genes.`**`Gene`**(***kwargs*)

Python object representing a gene.

Store the following information:

Required

- build: The genome build.
- chrom: The chromosome.
- start and end: The genomic positions for the gene.
- strand: The strand (either 1 or -1).
- xrefs: A dict of id mappings to multiple databases.
- transcripts: A list of Transcript objects.

Optional

- symbol: An HGNC symbol.
- desc: A short description.
- exons: A list of pairs of positions for exons.

You can only pass kwargs to build the genes. This makes for more eloquent code and avoids mistakes.

classmethod **`factory_ensembl_id`**(*ensembl_id*, *xrefs=None*, *build='GRCh37'*)

Builds a gene object from it's Ensembl ID.

Parameters **`ensembl_id`** (*str*) – The Ensembl ID.

Returns The Gene object.

Return type *Gene*

classmethod **`factory_symbol`**(*symbol*, *build='GRCh37'*)

Builds a gene object from it's HGNC symbol.

Parameters **`symbol`** (*str*) – The HGNC symbol.

Returns The Gene object.

Return type *Gene*

get_ortholog_sequences()
Queries Ensembl to get Sequence objects representing orthologs.

Returns A list of *gepyto.structures.sequences.Sequence*

Return type list

get_paralog_sequences()
Queries Ensembl to get Sequence objects representing paralogs.

Returns A list of *gepyto.structures.sequences.Sequence*

Return type list

classmethod get_xrefs(*field*, *query*, *build='GRCh37'*)

Fetches the HGNC (HUGO Gene Nomenclature Commitee) service to get a gene ID for other databases.

Parameters

- **field** (*str*) – A searchable fields.
- **query** (*str*) – The query.

Returns A dict representing information on the gene.

Return type dict

If no gene with this symbol can be found, *None* is returned.

classmethod get_xrefs_from_ensembl_id(*ensembl_id*, *build='GRCh37'*)

Fetches the HGNC (HUGO Gene Nomenclature Commitee) service to get a gene ID for other databases.

Parameters **ensembl_id** (*str*) – The gene Ensembl ID to query.

Returns A dict representing information on the gene.

Return type dict

If no gene with this Ensembl ID can be found, *None* is returned.

classmethod get_xrefs_from_symbol(*symbol*, *build='GRCh37'*)

Fetches the HGNC (HUGO Gene Nomenclature Commitee) service to get a gene ID for other databases.

Parameters **symbol** (*str*) – The gene symbol to query.

Returns A dict representing information on the gene.

Return type dict

If no gene with this symbol can be found, *None* is returned.

region
Lazily loads the Region object for this Gene.

class gepyto.structures.genes.**Transcript**(***kwargs*)
Python object representing a transcript.

Store the following information:

Required

•build: The genome build.

•chrom: The chromosome.

•start and end: The genomic positions for the gene.

•enst: The corresponding Ensembl transcript id.

Optional

•appris_cat: The APPRIS category.

•parent: The corresponding Gene object.

•biotype: The biotype as given by Ensembl.

classmethod **factory_position**(*region*, *build='GRCh37'*)
Gets a list of transcripts overlapping with the given position.

Parameters **pos** (*str*) – A genomic position of the form *chr2:12345-12347*.

Returns A list of *Transcript*

Return type list

This method uses the Ensembl API.

get_sequence(*seq_type='genomic'*)
Build a Sequence object representing the transcript.

Parameters **seq_type** (*str*) – This can be either genomic, cds, cdna or protein.

Returns A Sequence object representing the feature.

Return type *gepyto.structures.sequences.Sequence*

region
Lazily loads the Region object for this Transcript.

1.2 Variants

class gepyto.structures.variants.**SNP**(**args*, ***kwargs*)
Class representing a Single Nucleotide Polymorphism (SNP).

Instances can be created in two ways: either by providing ordered fields: `chrom, pos, rs, ref, alt` or by using named parameters.

classmethod **from_ensembl_api**(*rs*, *build='GRCh37'*)
Gets the information from the Ensembl REST API.

Parameters

- **rs** – The rs number for the variant of interest.
- **build** – The genome build (e.g. GRCh37 or GRCh38).

classmethod **from_str**(*s*, *rs=None*)
Parses a variant object from a str of the form chrXX:YYYY_R/A.

Parameters

- **s** – The string to parse the SNP from (Format: chrXX:YYY_R/A).
- **rs** – An optional parameter specifying the rs number.

Returns A list of SNP objects.

If it is a multi-allelic loci, the list will contain one SNP per alternative allele.

If it is not, this will be a list of length 1...

class gepyto.structures.variants.**Indel**(**args*, ***kwargs*)

Class representing short insertions/deletions (Indels).

Either initialize with the parameters corresponding to: `chrom, pos, rs, ref, alt` or by using the corresponding named parameters.

The notation we are using is consistent with the VCF format, but it is different from some API (e.g. Ensembl). We will try to standardize everything to comply with the VCF format.

The latter represents deletions by using the preceding nucleotide for the reference.

e.g. If the genomic sequence is AAGAA -> AAAA (deletion of the G) the indel will be represented as follows:

•Start: 2

•Ref: *AG*

•Alt: *A*

•length: 1 (difference between allele lengths)

e.g. If the genomic sequence is AAGAA -> AAGCAA (insertion of the C) the indel will be represented as follows:

•Start: 3

•Ref: *G*

•Alt: *GC*

•length: 1

classmethod from_ensembl_api(*rs*, *build='GRCh37'*)

Gets the information from the Ensembl REST API.

Parameters

- **rs** – The rs number for the variant of interest.
- **build** – The genome build (e.g. GRCh37 or GRCh38).

length

Computes the size of the indel.

This is the difference between the length of the alleles.

gepyto.structures.variants.**variant_list_to_dataframe**(*variants*)

Converts a regular Python list of Variant objects to a Pandas dataframe.

Parameters **variants** (*list*) – A list of `Variant` objects.

Returns A Pandas dataframe with genomic information as well as extra fields defined in the *_info* parameter.

Return type DataFrame

This function will add annotations from the *_info* dict if such a parameter is set for the considered variants. This dict has to be comparable between all elements of the list or an Exception will be raised.

This functionality can be very useful to flexibly add annotations to variants and to write them to a CSV file.

1.3 Sequences

class gepyto.structures.sequences.**Sequence**(*uid*, *s*, *seq_type*, *info=None*)

Object to represent biological sequences.

Parameters

- **uid** (*str*) – The identifier for this sequence.
- **s** (*str*) – The actual sequence.
- **seq_type** (*str*) – The sequence type (DNA, RNA or AA).
- **info** (*dict*) – A python dict of extra parameters (optional).

Common examples for the info attributes:

- species: Homo sapiens
- species_ncbi_tax_id: 9606
- description: dystroglycan 1
- db_name: RefSeq
- db_acc: NM_004393

base_base_correlation(*k=10*, *alphabet=None*)

Compute the base base correlation (BBC) for the sequence.

Parameters

- **k** (*int*) – k is a parameter of the BBC. Intuitively, it represents the maximum distance to observe correlation between bases.
- **alphabet** (*iterable*) – List of possible characters. This can be used to avoid autodetection of the alphabet in the case where sequences with missing letters are to be compared.

Returns A 16 dimensional vector representing the BBC.

Return type np.ndarray

A description of the method can be found here: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4272582

Liu, Zhi-Hua, et al. "Base-Base Correlation a Novel Sequence Feature and its Applications." Bioinformatics and Biomedical Engineering, 2007. ICBBE 2007. The 1st International Conference on. IEEE, 2007.

This implementation is generalized for any sequence type.

bbc(*k=10*, *alphabet=None*)

Shortcut to base_base_correlation.

find_coding_sequences(*cpu=6*)

Tries all the ORFs and translates every possible protein.

Returns A tuple containing the information of the coding sequence. (ORF, start, end, sequence)

Return type tuple

> **Warning:** This is currently **untested**.

find_translations(*cpu=6*)

Finds and translates peptides from any ORF in the sequence.

classmethod **from_reference**(*chrom*, *start*, *end=None*, *length=None*)
Create a Sequence object from a given locus.

gc_content()
Computes the GC content for the sequence.

get_annotations()
Return a list of bound SequenceAnnotation objects.

Returns A list of annotations for the sequence representing different kind of information about sub-sequences like protein domains.

Return type list

reverse_complement()
Reverse complement the sequence (compatible with IUPAC codes).

to_fasta(*line_len=80*, *full_header=False*)
Converts the sequence to a valid fasta string.

Parameters

- **line_len** (*int*) – The maximum line length for the sequence.
- **full_header** (*bool*) – Add the contents of the info field to the header. (default: False).

Returns A fasta string.

Return type str

translate(*no_check=False*)
Use the genetic code to translate a DNA or RNA sequence into an amino acid sequence.

gepyto.structures.sequences.**maketrans**()
Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

gepyto.structures.sequences.**smith_waterman**(*seq1*, *seq2*, *penalties=None*, *output='sequences'*)
Compute a pairwise local sequence alignment using the Smith Waterman algorithm.

The output parameter determines how results will be represented:

If "sequences" is chosen, the two aligned sequences will be returned with gaps represented by dashes ("-").

If "alignment" is chosen, a single encoded string will be reterned where "M" represents matches, "I" represents insertions, "D" represents deletions and "X" represents mismatches. This is done with respect to the first sequence.

In any mode, the first returned element is always the raw similarity score. Note that because this is local alignment, gaps at both ends won't be penalized. Also, only one of the potentially many best alignments will be output.

Warning: This implementation is not very optimized. It is not written in a low level language. It can be used for small sequences or for low number of comparisons, but should not be used in large scale products.

Note: Some functionality like affine gap penalties, or substitution matrices are not implemented.

The default penalty scheme is the following:

```
{
    "match": 2,
    "mismatch": -1,
    "gap": -1
}
```

You can follow this pattern to set your own penalty scores.

1.4 Region

class gepyto.structures.region.**Region**(*chrom*, *start*, *end*)
Region object to represent a part of the genome.

Parameters

- **chrom** (*str*) – The chromosome.
- **start** (*int*) – The start of the region.
- **end** (*int*) – The end of the region.

This can either represent a contiguous region or a fragmented region with multiple non-overlaping segments. This object can easily be converted to a Sequence using the *Region.sequence()* property. It is also easy to test overlap with another region or to test if an object is contained within this region using the *in* operator.

as_range(*iterator=False*)
Get a range corresponding to all the nucleotide positions.

distance_to(*region*)
Computes the distance to the given Region.

static from_str(*s*)
Parses the region object (contiguous only) from a string.

The expected format is *chrX:START-END*.

overlaps_with(*region*)
Tests overlap with another region.

sequence
Builds a Sequence object representing the region.

If the region is non contiguous, a tuple of sequences is returned.

union(*region*)
Primary method to create non contiguous regions.

This will create a region represented by the union of the current Region and the provided Region. This means that overlapping segments will be merged to avoid redundancy.

Interface to the Human Genome Reference (`reference`)

exception `gepyto.reference.`**`InvalidMapping`**(*value*)
Exception representing an invalid mapping that we can't fix automatically.

This can happen if the provided allele is incorrect for non-SNP variants. In this case we don't know if the locus is bad or if the sequence is bad. Since this is ambiguous, we raise this exception for the user to fix.

class `gepyto.reference.`**`Reference`**(*remote=False*)
Interface to the human genome reference file.

This class uses `pyfaidx` to parse the genome reference file referenced by `settings.REFERENCE_PATH`.

This can only be a single plain fasta file.

Also note that if the path is not in the `~/.gtconfig/gtrc.ini` file, gepyto will look for an environment variable named `REFERENCE_PATH`.

If the genome file can't be found, this class fallbacks to the Ensembl remote API to get the sequences.

This behaviour can also be forced by using the `remote=True` argument.

`check_variant_reference`(*variant*, *flip=False*)
Given a variant, makes sure that the 'ref' allele is consistent with the human genome reference.

Parameters

- **`variant`** (`gepyto.structures.variants.Variant` subclass) – The variant to verify.
- **`flip`** (*bool*) – If `True` incorrect `(ref, alt)` pairs will be flipped (Default: False).

Returns If flip is True, it returns the correct variant or raises a `ValueError` in case it is not salvageable. If flip is False, a bool is simply returned.

`get_nucleotide`(*chrom*, *pos*)
Get the nucleotide at the given genomic position.

`get_sequence`(*chrom*, *start*, *end=None*, *length=None*)
Get the nucleotide sequence at the given genomic locus.

Parameters

- **`chrom`** (*str*) – The chromosome.
- **`start`** (*int*) – The start position of the locus.
- **`end`** (*int*) – The end position.
- **`length`** (*int*) – The length of the sequence to fetch.

Either an `end` or a `length` parameter has to be provided.

The ranges are incluse, this means that (start, end) positions will both be included in the sequence.

`gepyto.reference.`**`check_indel_reference`**(*indel*, *ref*, *fix*)

Check and/or fix alleles for Indels.

Parameters **`ref`** (*Reference*) – A reference object.

In fix mode, this function will try to standardise the alleles for the given indel. This means that the VCF format will be enforced. No "-" alleles will be authorized.

_e.g._ ref: 'TC', alt: '-' will become ref: 'CTC', alt: 'C' given that the previous nucleotide in the reference is a 'C'. The position will be adjusted accordingly.

In the regular mode, the only test will be that the *ref* allele is consistent with the reference. That is, the sequence given as the *ref* allele equals the one on the same length starting at *pos* in the genome.

`gepyto.reference.`**`check_snp_reference`**(*snp*, *ref*, *flip*)

Utility function to check if a snp has the correct reference allele.

Parameters

- **`snp`** (*gepyto.structures.variants.SNP*) – The *gepyto.structures.variants.SNP* object.
- **`ref`** (*Reference*) – The *Reference* reference object.
- **`flip`** (*bool*) – A flag. If True, the return value is a variant with alleles flipped if necessary. If False, a bool is returned: True if the alleles are correct.

Returns Either a `gepyto.structures.variant.SNP` with flipped alleles or a bool.

This is used internally by *Reference*, but it is also available to users, but you need to provide a pyfaidx Fasta object.

CHAPTER 3

Database module (db)

3.1 Ensembl

Ensembl provides a very useful REST API. We added an interface that does some throttling and manages the JSON response from the API.

gepyto.db.ensembl.**query_ensembl**(*url*)
Query the given (Ensembl rest api) url and get a json reponse.

Parameters **url** (*str*) – The API url to query.

Returns A python object loaded from the JSON response from the server.

gepyto.db.ensembl.**get_url_prefix**(*build*)
Generate a Ensembl REST API URL prefix for the given build.

3.2 Appris

The Appris database aims to annotate alternative splice isoforms. We provide an interface to this database that can annotate *gepyto.structures.genes.Transcript* objects.

gepyto.db.appris.**get_category_for_transcript**(*enst*)
Fetches the annotation for a transcript (ENST).

Parameters **enst** (*str*) – The Ensembl transcript id.

Returns The APPRIS category for this transcript.

Return type str

gepyto.db.appris.**get_main_transcripts**(*ensg*)

Gets the main Ensembl transcript id for the provided gene based on the APPRIS annotation.

Parameters

- **ensg** – The Ensembl gene number (ENSG000000).
- **ensg** – str

Returns The "main" transcrit (ENST). If there is an *appris_principal* annotation, this will be returned. If it is not the case, the order of priority is the following: *appris_candidate_longest_ccds*, *appris_candidate_ccds*, *appris_candidate_longest_seq*, *appris_candidate*.

Return type str

gepyto.db.appris.**get_transcripts_for_gene**(*ensg*)
Fetches the transcripts and their annotation for a given gene (ENSG).

Parameters **ensg** (*str*) – The Ensembl gene id.

Returns A list of transcript IDs and their categories (tuples).

Return type tuple

gepyto.db.appris.**init_db**()
This is an initialization method for the database.

We use this to load the database only if a function is called.

3.3 Index

Index is a lower level implementation of an indexing data structure. It was designed to be able to handle any text delimited file with a chromosome column and a position column.

Indexing is fast because not all the file is read: jumps of a fixed size are used. The jump size is estimated by the `index_rate` parameter which is an estimated coverage of the indexed file. Lower rates will take less time to index and create smaller index files, but will result in slower lookups.

The structure of the index is a pickled python dictionary using chromosomes as keys and lists of `(position, file seek)` as values.

gepyto.db.index.**build_index**(*fn*, *chrom_col*, *pos_col*, *delimiter='\t'*, *skip_lines=0*, *index_rate=0.2*, *ignore_startswith=None*)
Build a index for the given file.

Parameters

- **fn** (*str*) – The filename
- **chrom_col** (*int*) – The column representing the chromosome (0 based).
- **pos_col** (*int*) – The column for the position on the chromosome (0 based).
- **delimiter** (*str*) – The delimiter for the columns (default tab).
- **skip_lines** (*int*) – Number of header lines to skip.
- **index_rate** (*float*) – The approximate rate of line indexing. As an example, a file with 1000 lines and the default index_rate of 0.2 will have an index with ~200 entries.
- **ignore_startswith** (*str*) – Ignore lines that start with a given string. This can be used to skip headers, but will not be used to parse the rest of the file.

Returns The index filename.

Return type str

gepyto.db.index.**get_index**(*fn*)

Restores the index for a given file or builds it if the index was not previously created.

Parameters **fn** (*str*) – The filname of the file to index.

Returns The numpy array representing the actual index.

Return type `numpy.ndarray`

gepyto.db.index.**goto**(*f*, *index*, *chrom*, *pos*)
Given a file, a locus and the index, go to the genomic coordinates.

Parameters

- **f** (*file*) – An open file.
- **index** – This is actually a tuple. The first element is an information dict containing the delimiter, chromosome column and position column. The second element is a numpy matrix containing the encoded loci and the "tell" positions.
- **index** – tuple
- **chrom** – The queried chromosome.
- **pos** – The queried position on the chromosome.

Returns True if the position was found and the cursor moved, False if the queried chromosome, position wasn't found.

Return type bool

3.4 UCSC

class gepyto.db.ucsc.**UCSC**(*db=None*)
Provides raw access to the UCSC MySQL database.

The database will be set to the *db* parameter or to the *BUILD* as defined in *gepyto*'s settings.

Later versions could implement features like throttling, but for now this is a very simple interface.

query_gap_table(*chromosome*, *ucsc_type*)
Get either the "telomere" or "centromere" of a given chromosome.

raw_sql(*sql*, *params*)
Execute a raw SQL query.

gepyto.db.ucsc.**get_centromere**(*chromosome*)
Returns a contiguous region representing the centromere of a chromosome.

Parameters **chromosome** (*str*) – The chromosome, _e.g._ "3"

Returns A region corresponding to the centromere.

Return type Region

This is done by connecting to the UCSC MySQL server.

gepyto.db.ucsc.**get_phylop_100_way**(*region*)
Get a vector of phyloP conservation scores for a given region.

Scores represent the -log(p-value) under a H0 of neutral evolution. Positive values represent conservation and negative values represent fast-evolving bases.

The UCSC MySQL database only contains aggregate scores for chunks of 1024 bases. We return the results for the subset of the required region that is fully contained in the UCSC bins.

Because UCSC uses 0-based indexing, we adjust the gepyto region before querying. This means that the user should use 1-based indexing, as usual when creating the Region object.

Warning: This function has a fairly low resolution. You should download the raw data (e.g. from goldenpath) if you need scores for each base. Also note that gepyto can't parse bigWig, but it can parse Wig files.

Warning: This function is **untested**.

gepyto.db.ucsc.**get_telomere**(*chromosome*)

Returns a Noncontiguous region representing the telomeres of a chromosome.

Parameters **chromosome** (*str*) – The chromosome, _e.g._ "3"

Returns A region corresponding to the telomeres.

Return type Region

This is done by connecting to the UCSC MySQL server.

CHAPTER 4

Bioinformatics common file formats parsing (`formats`)

4.1 Impute2

class gepyto.formats.impute2.**Impute2File**(*fn*, *mode='line'*, ***kwargs*)

Class representing an Impute2File.

This is used to either generate a dosage matrix where columns represent variants and rows represent samples or to read the file line by line using the generator syntax.

This also implements the context manager interface.

Usage:

```
# Read as probabilities (Line tuples).
with open(Impute2File(fn)) as f:
    for line in f:
        # line has name, chrom, pos, a1, a2, probabilities
        print(line)

# Read as dosage.
with open(Impute2File(fn), "dosage") as f:
    for dosage_vector, info in f:
        pass

# Read as a matrix.
with open(Impute2File(fn)) as f:
    # 1 row per sample and 1 column per variant. Values between 0 and 2
    m = f.as_matrix()
```

If you use the `dosage` mode, you can also add additional arguments:

- prob_threshold: Genotype probability cutoff for no call values (NaN).
- **is_chr23: Not implemented yet, but dosage is computed differently** for sexual chromosomes for men (hemizygote).
- **sex_vector: Not implemented yet, but this is a vector representing** the gender of every sample (for dosage computation on sexual chromosomes).

> **Warning:** Be careful with the *`Impute2File.as_matrix()`* function as it will try to load the WHOLE Impute2 file in memory.

as_matrix()

Creates a numpy dosage matrix from this file.

Returns A numpy matrix where columns represent variant dosage between 0 and 2 and a dataframe describing the variants (major, minor, maf).

Type tuple

> **Warning:** This will attempt to load the whole file in memory.

readline()
Read a single line from the Impute2File.

This will return either a `Line` including the genotype probabilities or a dosage vector. This depends on the *mode* (the second argument given to the file when it was opened).

Available modes are `dosage` and `line`.

4.2 SeqXML

class gepyto.formats.seqxml.**SeqXML**(*fn*)
Parses the SeqXML format representing sequence data.

Parameters **fn** (*str*) – The filename of the SeqXML file. The format description is available at orthoxml.org (visited Nov. 2014).

The returned object will have a list of entries which are `Sequence` objects.

get_seq(*uid*)
Get a sequence from it's unique identifier.

Parameters **uid** (*str*) – The sequence id.

4.3 GTF/GFF

gepyto.formats.gtf.**GFFFile**
alias of *GTFFile*

class gepyto.formats.gtf.**GTFFile**(*fn*)
Parser for GTF files.

This implementation was based on the format specification as described here: http://www.sanger.ac.uk/resources/software/gff/spec.html.

You can use this parser on both local files (compressed using gzip, or not) and on remote files (on a HTTP server).

For every line, this class will return a named tuple with the following fields:

- seqname
- source
- features
- start
- end
- score
- strand

- frame
- attributes

Example usage:

```
>>> import gepyto.formats.gtf
>>> url = "http://www.uniprot.org/uniprot/O60503.gff"
>>> gtf = gepyto.formats.gtf.GTFFile(url)
>>> gtf
<gepyto.formats.gtf.GTFFile object at 0x1006dd590>
>>> gtf.readline()
_Line(seqname=u'O60503', source=u'UniProtKB', features=u'Chain', start=1,end=1353, score=None, s
```

Line
alias of `_Line`

4.4 Wiggle (fixedStep)

Parser for Wiggle Track Format files.

class `gepyto.formats.wig.`**`WiggleFile`**(*stream*)
Parser for WIG files.

This returns a pandas dataframe with all the necessary information. In the process, all the inherent compactness of the Wiggle format is lost in exchange for an easier to manage representation. This means that more efficient parsers should be used for large chunks of data.

This implementation is based on the specification from: http://genome.ucsc.edu/goldenpath/help/wiggle.html

> **Warning:** `fixedStep` is the only implemented mode for now. Future releases might improve this parser to be more flexible.

To access the parsed information, use the `WiggleFile.as_dataframe()` function.

Usage (given a file on disk):

```
>>> import gepyto.formats.wig
>>> with gepyto.formats.wig.WiggleFile("my_file.wig") as f:
...     df = f.as_dataframe()
...
>>> df
  chrom     pos  value
0  chr3  400601     11
1  chr3  400701     22
2  chr3  400801     33
```

CHAPTER 5

Utilities (`utils`)

5.1 Genes

gepyto.utils.genes.**ensembl_genes_in_region**(*region*, *bare=False*, *build='GRCh37'*)

Queries a genome region of the form chr3:123-456 for genes using Ensembl API.

Parameters

- **region** (*str*) – The region to query.
- **bare** (*boolean*) – If *True*, no information about transcript will be fetched
- **build** (*str*) – The genome build to use (GRCh37 or GRCh38).

Returns A list of *gepyto.structures.genes.Gene*.

Return type list

5.2 Variants

gepyto.utils.variants.**ensembl_variants_in_region**(*region*, *build='GRCh37'*)

Queries a genome region of the form chr3:123-456 for variants using Ensembl API.

Parameters

- **region** (*str or gepyto Region*) – The region to query.
- **build** (*str*) – The genome build to use (GRCh37 or GRCh38).

Returns A list of `gepyto.structures.variants.Variant` or a subclass.

Return type list

> **Warning:** If the Region is not contiguous, this will also return all the variants that are in the "gaps".

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

g

I

L

M

O

Q

R

S

T

U

V

W