ICGC Data Parser Documentation

Release 0.2.2

Andres Garcia Garcia

Sep 03, 2018

Contents

CHAPTER 1

What is the ICGC-data-parser?

A library to ease the parsing of data from the International Cancer Genome Consortium data releases, in particular, the simple somatic mutation aggregates.

CHAPTER 2

Tutorial

2.1 Installation

Install via PyPI:

```
$ pip install ICGC_data_parser
```

2.2 Data download

The base data for the scripts is the ICGC's aggregated of the simple somatic mutation data. Which can be downloded using

```
wget https://dcc.icgc.org/api/v1/download?fn=/current/Summary/simple_somatic_mutation.
↪aggregated.vcf.gz
```

To know more about this file, please read About the ICGC's simple somatic mutations file

WARNING: The current release of the data contains a malformed header that causes the library to crash with an `IndexError`:

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~/.local/lib/python3.6/site-packages/vcf/parser.py in _parse_info(self, info_str)
    389                 try:
...
...
...
362     def _parse_info(self, info_str):

ValueError: could not convert string to float: 'PCAWG'
```

This is caused by a bad type specification in the header of the VCF file. To solve it, use the lollowing line after creating the `SSM_Reader` object (asuming the reader is in the `reader` variable)

```
# Fix weird bug due to malformed description headers
reader.infos['studies'] = reader.infos['studies']._replace(type='String')
```

In the future this will be solved in a more elegant way, but for now this is what we've got.

2.3 Usage

The main class in the project is the SSM_Reader. It allows to read easily the ICGC mutations file:

```
>>> from ICGC_data_parser import SSM_Reader

# Reads also compressed files!
>>> reader = SSM_Reader(open('data/simple_somatic_mutations.aggregated.vcf.gz'))

# or...
>>> reader = SSM_Reader(filename='data/simple_somatic_mutations.aggregated.vcf.gz')
#                       ^^^^^^^^
# The filename keyord argument is important, else we get an IndexError
```

The SSM_Reader.parse method allows to iterate through the records of the file and access the parts of the record. You can also specify regular expressions to filter only the lines you want:

```
# Print only the mutations that are in the
# European Union Breast Cancer project (BRCA-EU).

>>> for record in reader.parse(filters=['BRCA-EU']):
...     print(record.ID, record.CHROM, record.POS)

MU66865518 1 100141201
MU65487875 1 100160548
MU66281118 1 100638179
MU66254120 1 101352655
...
```

The INFO field is special in the sense that it contains several subfields, AND those subfields may be list-like entries with more subfields themselves (in particular the CONSEQUENCE and OCCURRENCE subfields):

```
# The subfields of the INFO field:
>>> next(reader).INFO

{'CONSEQUENCE': [
    '||||||intergenic_region||',
    'CD1A|ENSG00000158477|+|CD1A-001|ENST00000289429||upstream_gene_variant||'
    ],
 'OCCURRENCE': [
     'ESAD-UK|1|301|0.00332',
     'EOPC-DE|1|202|0.00495',
     'BRCA-EU|1|569|0.00176'
    ],
 'affected_donors': 3,
 'mutation': 'T>A',
 'project_count': 3,
```

(continues on next page)

(continued from previous page)

```
 'studies': None,
 'tested_donors': 12068}
```

```
# The description of the CONSEQUENCE subfield
>>> print(reader.infos['CONSEQUENCE'].desc)

Mutation consequence predictions annotated by SnpEff
(subfields: gene_symbol|gene_affected|gene_strand|transcript_name|transcript_
↪affected|protein_affected|consequence_type|cds_mutation|aa_mutation)
```

```
# The description of the OCCURRENCE subfield
>>> print(reader.infos['OCCURRENCE'].desc)

Mutation occurrence counts broken down by project
(subfields: project_code|affected_donors|tested_donors|frequency)
```

Sometimes we want to also parse the information in those subfields. For this purpose, the `SSM_Reader.subfield_parser` factory method is useful. This method creates a parser of the specified subfield that allows easy access to the data:

```
# Create the subfield parser for the CONSEQUENCE subfield
>>> consequences = reader.subfield_parser('CONSEQUENCE')


>>> for record in reader.parse():
...     # Which genes are affected?
...     genes_affected = {c.gene_symbol
...                       for c in consequences(record)
...                       if c.gene_affected}
...
...     print(f'Mutation: {record.ID}')
...     print('\t', ", ".join(genes_affected))

Mutation: MU93246178
     TPM3
Mutation: MU66962994
     RP11-350G8.9, SHE
Mutation: MU93246498
     DCST1, ADAM15, RP11-307C12.11
Mutation: MU66377106
     EFNA3, ADAM15, EFNA4
...
```

The library also contains some helper scripts to manipulate VCF files (like the ICGC mutations file):

- `vcf_map_assembly.py`: Creates a new VCF with the positions mapped to another genome assembly. This is useful because currently the positions reported by ICGC are in the human genome assembly GRCh37, while the most recent (and the one the rest of the world uses) is the GRCh38 assembly.
- `vcf_sample.py`: Creates a new VCF with a fraction of the mutations in the original. The mutations are randomly sampled but maintain the order they had in the original file. This is useful when one wants to make small test analysis on the data, but still wants the results to be representative of all the mutations.
- `vcf_split.py`: Splits the input VCF into several (also valid VCFs), this is useful in case one wants to split the analyses into processes that receive one file each.

The specific documentation of the scripts can be obtained by executing:

```
$ python3 <script name>.py --help
```

Also, the library is shipped with some Jupyter Notebooks that elaborate on the examples. Besides, in the notebooks are demonstrated ways to manage common parsing errors that have to do with malformed input files.

2.4 Meta

Author: Ad115 - Github – a.garcia230395@gmail.com

Project pages: Docs - @GitHub - @PyPI

2.5 Contributing

1. Check for open issues or open a fresh issue to start a discussion around a feature idea or a bug.
2. Fork the repository on GitHub to start making your changes to a feature branch, derived from the **master** branch.
3. Write a test which shows that the bug was fixed or that the feature works as expected.
4. Send a pull request and bug the maintainer until it gets merged and published.

2.6 Examples and more

2.6.1 Examples

Now we demonstrate more involved examples of what can be done with the library. Please read the *Tutorial* first to get started.

Finding the distribution of mutation consequences

Whether the mutations affect genes, cause frameshifts, fall in an intronic region or are silent SNPs, we want to know the relative abundance of these consequences:

```
from collections import Counter
from ICGC_data_parser import SSM_Reader


counter = Counter()

# Open the mutations file
mutations = SSM_Reader(filename='data/ssm_sample.vcf')
consequences = mutations.subfield_parser('CONSEQUENCE')

for record in mutations:
    consequence_types = [c.consequence_type for c in consequences(record)]
    counter.update(consequence_types)


total = sum(counter.values())
```

(continues on next page)

(continued from previous page)

```
for consequence_type,n in counter.most_common():
    print(f'{n/total :<10.3%} : {consequence_type}')
```

```
60.787%    : intron_variant
17.344%    : intergenic_region
8.558%     : downstream_gene_variant
8.295%     : upstream_gene_variant
1.657%     : missense_variant
1.327%     : exon_variant
0.746%     : synonymous_variant
0.654%     : 3_prime_UTR_variant
0.167%     : splice_region_variant
0.144%     : 5_prime_UTR_variant
0.103%     : stop_gained
0.099%     : frameshift_variant
...
```

As we can see, rather unexpectedly, an overwhelming amount of mutations fall in intronic regions. This is worth of more investigation.

Finding the distribution of mutation recurrence among patients

The ICGC data allows us to know, for each mutation, how many patients where affected by this mutation (the recurrence of the mutation). Thus, one can solve the question: Do the mutations in cancer patients follow a recurring pattern? In which case, the mutation recurrences must be more or less homogeneously distributed, or: Is it that every patient has their unique set of mutations? In which case, most mutations would appear only once.

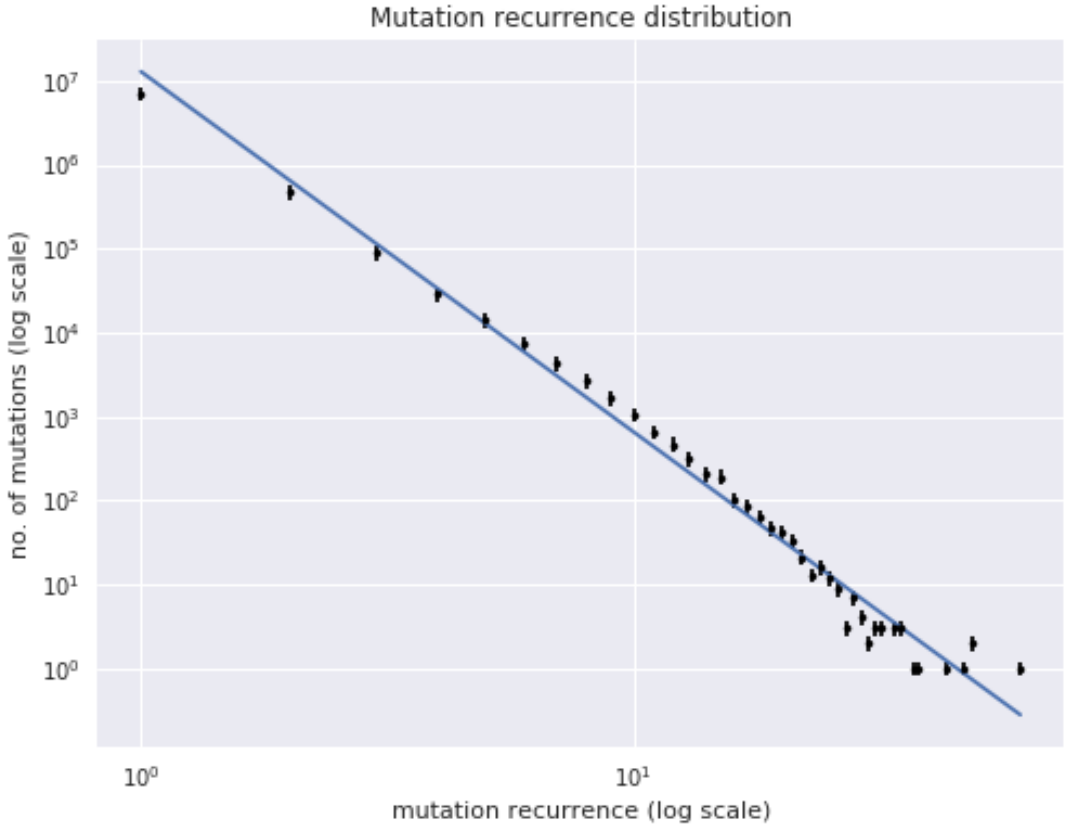
Let's try to solve this:

```
from collections import Counter
from ICGC_data_parser import SSM_Reader

# Open the mutations file
mutations = SSM_Reader(filename='data/ssm_sample.vcf')

# Fetch recurrence data per mutation
recurrence_distribution = Counter(mutation.INFO['affected_donors']
                                  for mutation in mutations)

total = sum(recurrence_distribution.values())
for mut_recurrence,n in recurrence_distribution.most_common():
    print(f'{n/total :<10.3%} : Mutations recurred in {mut_recurrence} patients.')
```

```
92.324%    : Mutations recurred in 1 patients.
6.058%     : Mutations recurred in 2 patients.
0.990%     : Mutations recurred in 3 patients.
0.327%     : Mutations recurred in 4 patients.
0.132%     : Mutations recurred in 5 patients.
0.061%     : Mutations recurred in 6 patients.
0.035%     : Mutations recurred in 7 patients.
0.026%     : Mutations recurred in 8 patients.
0.015%     : Mutations recurred in 9 patients.
0.012%     : Mutations recurred in 10 patients.
0.010%     : Mutations recurred in 11 patients.
...
```

As we can see, most of the mutations are only present in very few patients, and taking into account that the file aggregates more than 10,000 patients' worth of data, this tells us that every patient's mutational footprint is essentially unique.

The Jupyter notebook `recurrence_distribution.ipynb` from the library elaborates on this example and shows how to plot this and fit to a power law. The presence of a power law means that the mutations present themselves as randomly as they can:



Finding the distribution of mutatios among genes

From the above example, we can see that, per nucleotide, the mutations can be considered essentially random. But, we can try to take a more coarse grained approach and quantify the mutations by gene so that it may be that the mutations divide randomly among all genes (in which case we may find almost all genes with the same number of mutations), or we may find that some genes have significantly more mutations than the rest.

This is how we may find out:

```
from collections import Counter
from ICGC_data_parser import SSM_Reader

# -- 1. Get the mutations count per gene

mutations_per_gene = Counter()

mutations = SSM_Reader(filename='data/ssm_sample.vcf')
```

(continues on next page)

(continued from previous page)

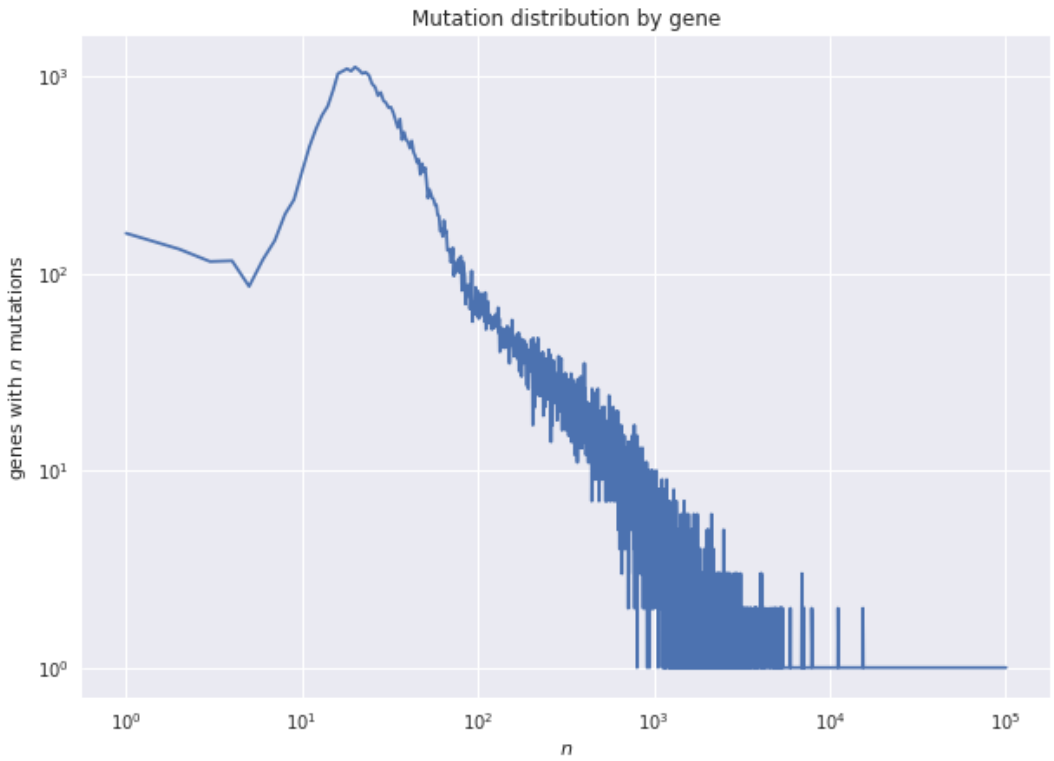
```
consequences = mutations.subfield_parser('CONSEQUENCE')

for record in mutations:
    affected_genes = [c.gene_symbol for c in consequences(record) if c.gene_affected]
    mutations_per_gene.update(affected_genes)


# Show partial results
for gene,mutations in mutations_per_gene.most_common():
    print(f'{gene:<10}: {mutations}')
```

```
PCDH15    : 1651
RBFOX1    : 1041
CSMD1     : 979
DLG2      : 941
SPOCK3    : 929
DPP10     : 649
CTNND2    : 632
...
```

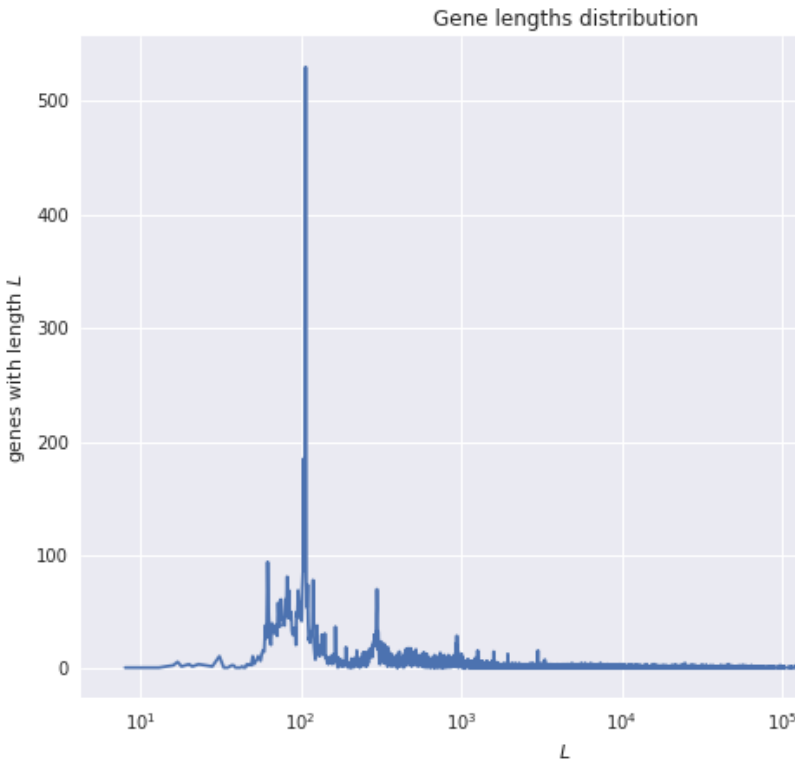
```
# -- 2. Now group by number of mutations

distribution = Counter(mutations_per_gene.values())
```

```
X    | NO. OF GENES WITH X MUTATIONS
----------------------------------------
1    | 8600
2    | 3624
3    | 1638
4    | 1403
6    | 1001
5    | 877
8    | 712
...
```

In the script `mutations_distribution_genes.ipynb` we can see how we plot this data. For now, the result-



ing figure is the following:



But remember that genes have wildly varying lengths:

So, the distribution of mutations may be convoluted by the distribution of gene lengths. In order to smooth out this effect, we want to plot not the total number of mutations per gene, but the mutation density (the number of mutations normalized by the gene length).

To do this we need to check gene lengths, and the easiest way to do this is via the Ensembl REST API, which we may use with the module `ensembl_rest`. The following shows how to do this:

```
# In order to find out the length of the
# genes, we will use the Ensembl REST API.
import ensembl_rest
from itertools import islice

def chunks_of(iterable, size=10):
    """A generator that yields chunks of fixed size from the iterable."""
    iterator = iter(iterable)
    while True:
        next_ = list(islice(iterator, size))
        if next_:
            yield next_
        else:
            break
# ---


# -- 3. Normalize mutation counts by gene length
```

(continues on next page)

(continued from previous page)

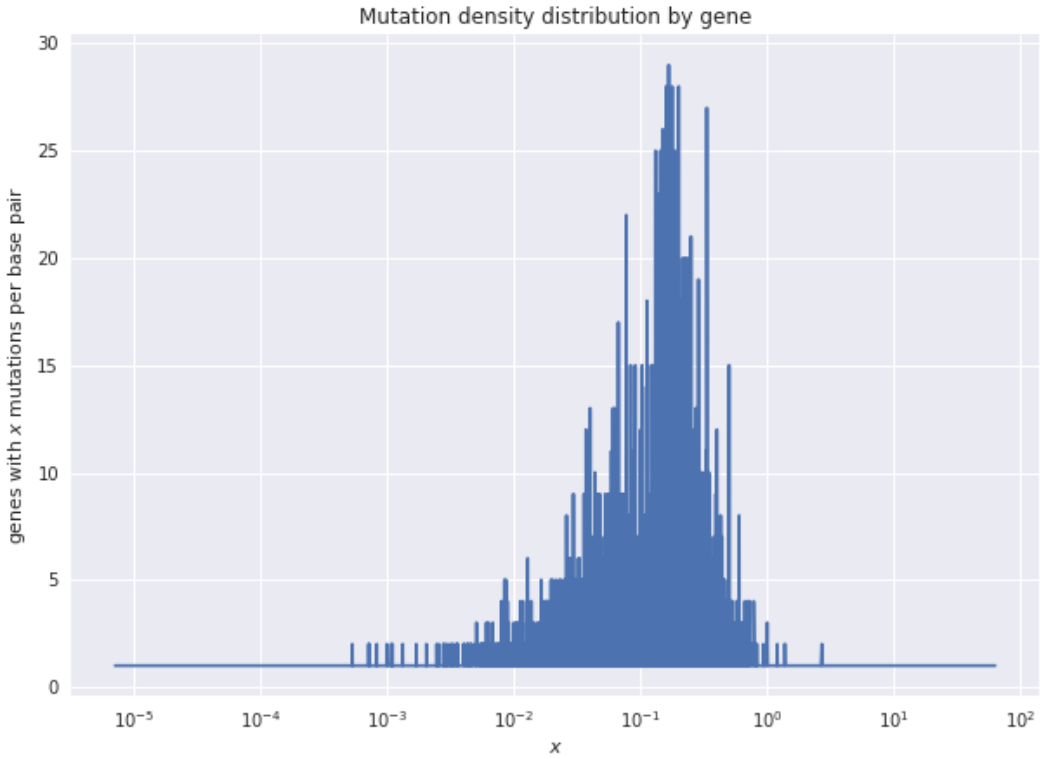
```
# Instantiate a client for communication with
# the Ensembl REST API.
client = ensembl_rest.EnsemblClient()


normalized_counts = Counter()
for gene_batch in chunks_of(mutations_per_gene, size=1000):
    # Get information of the genes
    gene_data = client.symbol_post('human',
                                   params={'symbols': gene_batch})

    gene_lengths = {gene: data['end'] - data['start'] + 1
                        for gene, data in gene_data.items()}

    # Get the normalization
    normalized_counts.update({
        gene: mutations_per_gene[gene] / gene_lengths[gene]
            for gene in gene_data
    })



# Show partial results
for gene,mutations in normalized_counts.most_common():
     print(f'{gene:<10}: {mutations}')
```

```
IGHD7-27  : 0.5454545454545454
IGKJ1     : 0.2894736842105263
IGKJ3     : 0.2894736842105263
IGKJ2     : 0.28205128205128205
SNORD112  : 0.18181818181818182
IGHJ3P    : 0.18
IGHJ5     : 0.16326530612244897
...
```

```
# -- 4. Aggregate by mutation density

normalized_distribution = Counter(normalized_counts.values())
```

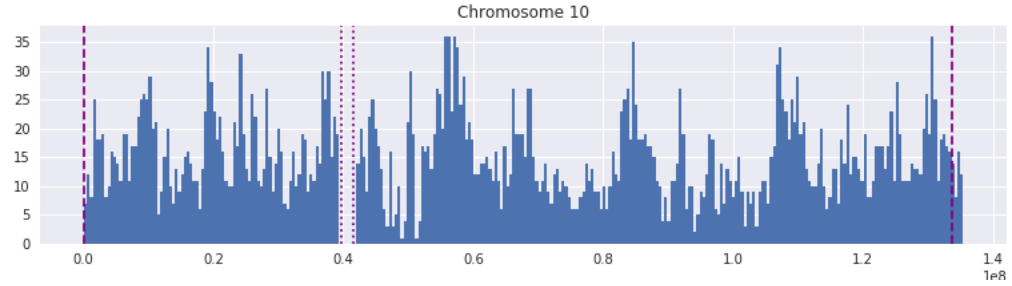
```
X         | NO. OF GENES WITH X MUTATION DENSITY
----------------------------------------
0.9346%   | 112
0.9615%   | 33
0.9524%   | 26
0.9434%   | 23
1.6129%   | 20
1.8692%   | 20
0.9804%   | 19
1.2195%   | 19
...
```

Now we can plot this data. The code to do so is in the notebook `mutations_distribution_genes.ipynb`. For now, the figure that results is the following:



Plotting the mutation density in the chromosomes

The example notebook `mutation_distribution_chroms.ipynb` shows how to plot the mutations distribution in the chromosomes. This is useful when one wants to compare the variations among different projects. The resulting figures are as the following one (with the chromosome and centromere boundaries shown):



2.6.2 The ICGC's mutations file

This is about the infamous `simple_somatic_mutations.aggregated.vcf` file presented in each ICGC Data Release which contains an aggregated of the information of all simple somatic mutations found accross all patients in all cancer projects is found.

Download

This file can be downloaded from the ICGC site data releases site or using:

```
$ wget https://dcc.icgc.org/api/v1/download?fn=/current/Summary/simple_somatic_
↪mutation.aggregated.vcf.gz
```

To resume an interrupted download use the `-c` switch on the previous command.

Then, the file can be extracted with the `gunzip` command (or not, the *ICGC_data_parser.SSM_Reader* can read compressed files too ;D).

Structure

The `simple_somatic_mutations.aggregated.vcf` file, from now on referred as the SSM file, is a VCF file as specified in HTS format specifications, and in particular, the SSM files are created using the SnpEff annotation tool.

In general, the format is similar to a TSV file in which the comments are marked with `##` and the headers line with `#` and there is one line per simple-somatic-mutation found.

Fields and Header lines

Next are the 13 heading lines from a SSM file (data release 22):

```
##fileformat=VCFv4.1
##INFO=<ID=CONSEQUENCE,Number=.,Type=String,Description="Mutation consequence␣
↪predictions annotated by SnpEff (subfields: gene_symbol|gene_affected|gene_
↪strand|transcript_name|transcript_affected|protein_affected|consequence_type|cds_
↪mutation|aa_mutation)">
##INFO=<ID=OCCURRENCE,Number=.,Type=String,Description="Mutation occurrence counts␣
↪broken down by project (subfields: project_code|affected_donors|tested_
↪donors|frequency)">
##INFO=<ID=affected_donors,Number=1,Type=Integer,Description="Number of donors with␣
↪the current mutation">
##INFO=<ID=mutation,Number=1,Type=String,Description="Somatic mutation definition">
##INFO=<ID=project_count,Number=1,Type=Integer,Description="Number of projects with␣
↪the current mutation">
##INFO=<ID=tested_donors,Number=1,Type=Integer,Description="Total number of donors␣
↪with SSM data available">
##comment=ICGC open access Simple Somatic Mutations (SSM) data dump in VCF format
##fileDate=2016-08-16T16:32:17.882-04:00
##geneModel=ENSEMBL75
##reference=GRCh37
##source=ICGC22-12
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO
```

This is what we can see in those lines:

- *fileformat*: A line specifying the VCF version (4.1).
- *INFO*: Six lines breaking down each part of the INFO field.
- *comment*:A general description of the file (*ICGC open access Simple Somatic Mutations (SSM) data dump in VCF format*).
- *fileDate*: The creation date of the file (August 2016).

- *geneModel*: A specification of the gene model (ENSEMBL75). This is the nameset used for the annotations in the INFO field. In particular the identifiers and names in the CONSEQUENCE subfield. **WATCH OUT!** Some genes, transcripts and identifiers annotated may have changed for the current release and so may not be found in a direct query. At the moment of writing, the most recent build is Ensembl 87.
- *reference*: The genome assembly version used for the positions in the reference genome (GRCh37). **WATCH OUT THIS!** The positions may have dramatic changes from one assembly to another. At the moment, the most recent version of the human genome reference is the GRCh38 assembly.
- *source*: The data source (ICGC Data release 22)
- **The column headers**: See section *The column headers*.

The column headers

The data is split in these fields:

- **CHROM**: The chromosome the mutation is in.
- **POS**: The position in the chromosome of the start of the mutation. This is in the reference assembly specified in the initial comments.
- **ID**: The current mutation's ICGC identifier.
- **REF**: The sequence found in the reference.
- **ALT**: The sequence found in the mutated sample, so that the mutation definition is REF>ALT.
- **QUAL**: The quality of the read. As a general rule, a quality <10 is unreliable.
- **FILTER**.
- **INFO**: Additional annotation on the mutation consequences, and occurrence along patients and projects. It is further commented on *The INFO Field*

The INFO field

This field annotates predicted consequences, and seen occurrences of the current mutation. The consequences are as seen by the SnpEff package.

There may be multiple consecuences and occurrences of the same mutation, and those need to be clearly specified. Thus the complex form of this field.

In the file, *the parts are separated with a semicolon (`;`), and each part may have itself subfields, which are separated with pipes (`|`). Alternative parts* (e.g. different consequences for the mutation or occurrences in different cancer projects) *are separated by a comma (`,`).*

- **CONSEQUENCE:** Mutation consequence predictions annotated by SnpEff. Which has itself the next subfields:

1. *gene_symbol,*
2. *gene_affected,*
3. *gene_strand,*
4. *transcript_name,*
5. *transcript_affected,*
6. *protein_affected,*

7. *consequence_type,*
8. *cds_mutation,*
9. *aa_mutation*

- **OCCURRENCE**: Mutation occurrence counts broken down by project. Which has itself the next subfields:
  1. *project_code,*
  2. *affected_donors,*
  3. *tested_donors,*
  4. *frequency*
- **affected_donors**: Total number of donors with the current mutation.
- **mutation**: Somatic mutation definition, in the form BEFORE>AFTER.
- **project_count**: Number of projects with the current mutation.
- **tested_donors**: Total number of donors with SSM data available.

Interpreting a sample mutation

Now we are ready to interpret an example mutation line from the data.

The mutation

```
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO
1       100000022       MU39532371      C       T       .       .       ␣
↪CONSEQUENCE=||||||intergenic_region||,RP11-413P11.1|ENSG00000224445|1|RP11-413P11.1-
↪001|ENST00000438829||upstream_gene_variant||;OCCURRENCE=SKCA-BR|1|70|0.01429;
↪affected_donors=1;mutation=C>T;project_count=1;tested_donors=10638
```

The interpretation

We can see the data for the mutation **MU39532371**, which is in the chromosome number *1*, at the position *100000022*, and is defined as *C>T*, with no quality or filtering information available. We can also see in the INFO that this mutation has two consequences: one as a mutation ocurring in an intergenic region, and one as a mutation that affects the *ENSG00000224445 (RP11-413P11.1)* gene and it's *ENST00000438829 (RP11-413P11.1-001)* transcript provoking an *upstream_gene_variant*. Besides, it was found in a sample from the Great Britain's skin cancer ICGC project (*SKCA-BR*) with *1* patient affected out of the *70* in the project and of the *10638* accross all projects.

2.6.3 API Documentation

class ICGC_data_parser.**SSM_Reader**(**args*, ***kwargs*)

Reader class for the International Cancer Genome Consortium aggregate file of simple somatic mutations from the Data Releases.

Example:

```
>>> reader = SSM_Reader(filename='data/ssm_sample.vcf')

>>> for record in reader.parse(filters=['BRCA-EU']):
...     print(record.ID, record.CHROM, record.POS)
MU66865518 1 100141201
MU65487875 1 100160548
MU66281118 1 100638179
MU66254120 1 101352655
    ...
```

iter_lines (*filters=None*)
Iterate through the file's raw lines, filtering out the ones not matching the regular expressions given.

next_array (*strict_whitespace=False*)
Fetch the next line splitted into fields.

If `strict_whitespace` is True, then split on tabs rather than whitespace. This allows for fields with spaces in them.

next_line ()
Fetch the next raw line from the file.

parse (*filters=None*)
Iterate through the records of the file, filtering out the lines that do not match the regular expressions given.

Example:

```
>>> reader = SSM_Reader(filename='data/ssm_sample.vcf')

>>> for record in reader.parse(filters=['BRCA-EU']):
...     print(record.ID)
MU66865518
MU65487875
MU66281118
MU66254120
    ...
```

push_line (*line*)
Rebuffers line so that it is parsed next.

subfield_parser (*sf_name*, *sep='|'*)
Get a parser for the items of the subfield.

Useful to parse the CONSEQUENCE and OCCURRENCE subfields of the INFO field.

Example:

```
>>> reader = SSM_Reader(filename='data/ssm_sample.vcf')

>>> CONSEQUENCE = reader.subfield_parser('CONSEQUENCE')

>>> for record in reader.parse(filters=['BRCA-EU']):
...     # Which genes are affected?
...     print(CONSEQUENCE(record)[0].gene_symbol)
SLC27A3
GATAD2B
TPM3
SHE
ADAM15
  ...
```

Index