The entire source code for this can be found on https://github.com/SpacemanSpiff7/ESKeDiT

Here, I will go over the spaghetti code you will find inside the source files.

eskedit_main.py has the argument parsing for the CLI if you wish to use it. You may find it easier to adapt the existing code to your purposes as there are a few unrefined options.

To calculate frequency, we need to count kmers in the reference genome and compare that to the observed mutability we see in gnomAD.

In [1]:

```
def ktrain_region_driver(bedregions: iter, vcfpath: str, fastapath: str, kmer_size: int, methylation_vcf_path: str):
    # may want to add this as a keyword argument later
    AC_cutoff = 1
    base_methylation_probability_cache = base_methylation_probability
    is_cpg_ct_ga = is_cpg_ct
    fasta = Fasta(fastapath, sequence_always_upper=True)
    gnomadVCF = VCF(vcfpath)
    reference_kmer_counts = Counter()
    low_meth_kmer_counts = Counter()
    mid_meth_kmer_counts = Counter()
    hi_meth_kmer_counts = Counter()

    rare_transitions_count = defaultdict(lambda: array.array('L', [0, 0, 0, 0]))
    # rare_transitions_ac = defaultdict(lambda: array.array('L', [0, 0, 0, 0]))
    # common_transitions_ac = defaultdict(lambda: array.array('L', [0, 0, 0, 0]))
    # common_transitions_count = defaultdict(lambda: array.array('L', [0, 0, 0, 0]))
    num_singletons, num_all_variants = 0, 0
    meth_vcf = VCF(methylation_vcf_path)
    lo_count, mid_count, hi_count = 0, 0, 0
    no_count = 0
    low_meth = defaultdict(lambda: array.array('L', [0, 0, 0, 0]))  # < 0.2
    mid_meth = defaultdict(lambda: array.array('L', [0, 0, 0, 0]))  # 0.2-0.6
    hi_meth = defaultdict(lambda: array.array('L', [0, 0, 0, 0]))  # # > 0.6

    nuc_idx = {'A': 0, 'C': 1, 'G': 2, 'T': 3}
    complement = {'A': 'T', 'C': 'G', 'G': 'C', 'T': 'A'}
    for region in bedregions:
        # bed indices will be same as start/stop stored by GRegion
        # Since gnomad variants are aligned to forawrd str and, do not consider strandedness
        try:
            seq = region.get_seq_from_fasta(fasta, kmer_size=kmer_size)
        except KeyError:
            print('Fasta record {} : {} - {} not found'.format(region.chrom, region.start, region.stop),
                  file=sys.stderr, flush=True)
            continue
        if seq is None or 'N' in seq:
            continue
        kmer_results = kmer_search(seq, kmer_size)
        # Add count of kmers to master dictionary
        reference_kmer_counts += kmer_results['count_kmers']

        low_meth_kmer_counts_local = Counter()
        mid_meth_kmer_counts_local = Counter()
        hi_meth_kmer_counts_local = Counter()

        # count kmer contexts for methylation probabilities
        for variant in meth_vcf(region.gnomad_rep()):
            seq_idx = variant.POS - region.start + kmer_size // 2 - 1
            forward_seq_context = seq[seq_idx - kmer_size // 2: seq_idx + kmer_size // 2 + 1]

            if not is_cpg_ct_ga(forward_seq_context, variant.ALT[0]):
                continue
            if variant.REF != forward_seq_context[len(forward_seq_context) // 2]:
                print(
                    f'ERROR: Fasta REF {forward_seq_context[len(forward_seq_context) // 2]} and VCF REF {variant.REF}
don\'t match at position {variant.POS} on {variant.CHROM}',
                    flush=True, file=sys.stderr)

            reverse_seq_context = ''.join([complement.get(base) for base in forward_seq_context[::-1]])
            for seq_context in [forward_seq_context, reverse_seq_context]:
                if 'N' in seq_context or len(seq_context) == 0:
                    continue

                if len(seq_context) != kmer_size:
                    raise IndexError(f'ERROR: kmer_size: {kmer_size} recovered seq len: {len(seq_context)}')
                # check index

                # check for reverse complement
                # if variant.REF == 'G':
                #     seq_context = ''.join([complement.get(base, 'N') for base in list(seq_context[::-1])])

                methylation = float(variant.format('methylation')[0]) / 100
                if methylation is None:
                    methylation = -1.0

                if methylation < 0:
                    print(
                        f'No methylation data for {variant.CHROM}:{variant.POS}-{variant.POS} with context {seq_contex
t}',

                        file=sys.stderr)
                elif methylation < 0.2:  # low/none
                    low_meth_kmer_counts_local[seq_context] += 1
                elif 0.2 <= methylation <= 0.6:
                    mid_meth_kmer_counts_local[seq_context] += 1
                elif 0.6 < methylation <= 1:
                    hi_meth_kmer_counts_local[seq_context] += 1
                else:
                    print('ERROR: Irregular meth prob', file=sys.stderr)

        low_meth_kmer_counts += low_meth_kmer_counts_local
        mid_meth_kmer_counts += mid_meth_kmer_counts_local
        hi_meth_kmer_counts += hi_meth_kmer_counts_local

        for variant in gnomadVCF(region.gnomad_rep()):
            varAC = variant.INFO.get('AC')
            if is_quality_snv(variant):
                num_all_variants += 1
                if varAC <= AC_cutoff:
                    num_singletons += 1
                    # welcome to indexing hell
                    # VCF is 1-based[,], BED is 0-based [,)
                    # zero based idx = (VCFidx - 1) - (BED start
                    seq_idx = variant.POS - region.start + kmer_size // 2 - 1
                    forward_seq_context = seq[seq_idx - kmer_size // 2: seq_idx + kmer_size // 2 + 1]
                    reverse_seq_context = ''.join([complement.get(base) for base in forward_seq_context[::-1]])
                    sequences = [(forward_seq_context, variant.ALT[0]),
                                 (reverse_seq_context, complement.get(variant.ALT[0]))]
                    for seq_context, seq_variant in sequences:
                        if 'N' in seq_context or len(seq_context) == 0:
                            continue
                        # check index
                        # if variant.REF != seq_context[len(seq_context) // 2]:
                        #     print(
                        #         f'ERROR: Fasta REF {seq_context[len(seq_context) // 2]} and VCF REF {variant.REF} do
n\'t match at position {variant.POS} on {variant.CHROM}',
                        #         flush=True, file=sys.stderr)

                        # This is for counting allele information
                        # check methylation if C followed by G or

                        if methylation_vcf_path is not None and is_cpg_ct_ga(seq_context, variant.ALT[0]):
                            methylation = base_methylation_probability_cache(
                                meth_vcf(f'{variant.CHROM}:{variant.POS}-{variant.POS}'))
                            if methylation < 0:
                                print(
                                    f'No methylation data for {variant.CHROM}:{variant.POS}-{variant.POS} with context
{seq_context}',
                                    file=sys.stderr)
                                no_count += 1
                            elif methylation < 0.2:  # low/none
                                lo_count += 1
                                low_meth[seq_context][nuc_idx[variant.ALT[0]]] += 1
                            elif 0.2 <= methylation <= 0.6:
                                mid_count += 1
                                mid_meth[seq_context][nuc_idx[variant.ALT[0]]] += 1
                            elif 0.6 < methylation <= 1:
                                hi_count += 1
                                hi_meth[seq_context][nuc_idx[variant.ALT[0]]] += 1
                            else:
                                print('Irregular meth prob')

                        rare_transitions_count[seq_context][nuc_idx[variant.ALT[0]]] += 1

        # print summary of kmer search on GRegion fields to stdout
        region.add_field('num_variants', f'AC_lte_{AC_cutoff}={num_singletons},all_snvs={num_all_variants}')
        if methylation_vcf_path is not None:
            region.add_field('methylation',
                             f'low={lo_count},intermediate={mid_count},high={hi_count},no_data={no_count}')
        with print_lock:
            print(str(region), flush=True)

    # Package return values

    def freeze_transitions(t_dict):
        newdict = defaultdict(tuple)
        for key, value in t_dict.items():
            newdict[key] = tuple(value)
        return frozenset(newdict.items())

    if methylation_vcf_path is not None:
        return frozenset(reference_kmer_counts.items()), \
               frozenset(low_meth_kmer_counts.items()), \
               frozenset(mid_meth_kmer_counts.items()), \
               frozenset(hi_meth_kmer_counts.items()), \
               freeze_transitions(rare_transitions_count), \
               freeze_transitions(low_meth), \
               freeze_transitions(mid_meth), \
               freeze_transitions(hi_meth)
    else:
        return frozenset(reference_kmer_counts.items()), freeze_transitions(rare_transitions_count)
```

Above is all of the main method which I will break down below:

First we iterate through every region in the bed file where we quickly count each kmers using the `kmer_search` function

In [2]:

```
# here's what that function looks like (not included above)
def kmer_search(sequence: str, kmer_length: int, additional_functions: iter = None) -> dict:
    """
    Driver for get_kmer_count
    :param additional_functions: an iterable consisting of functions that accept sequence and kmer_length (in that ord
er) as positional arguments
    :param sequence:
    :param kmer_length:
    :return: a dictionary containing the results of the analysis (will always count_kmers by default). Maps function n
ame to result
    """
    if additional_functions is None:
        additional_functions = []
    if count_kmers not in additional_functions:
        additional_functions.append(count_kmers)
    results = {}

    if additional_functions is not None:
        for custom_function in additional_functions:
            results.update({custom_function.__name__: custom_function(sequence, kmer_length)})

    return results
```

In [4]:

```
# this is a driver method for including any further analysis you want to perform on the sequence,
# default behavior only includes counting the kmers, function shown  below
from collections import Counter
def count_kmers(ref_seq: str, kmer_length: int) -> Counter:
    counts = Counter()
    complement = {'A': 'T', 'C': 'G', 'G': 'C', 'T': 'A'}
    reverse = ''.join([complement.get(base) for base in ref_seq[::-1]])
    for sequence in [ref_seq, reverse]:
        for i in range(len(sequence) - (kmer_length - 1)):
            next_seq = sequence[i:(i + kmer_length)]
            if not ('N' in next_seq):
                counts[next_seq] += 1
    return counts
```

As you can see, the code above requires the sequence to already be in a string which will improve performance. For additional performance improvemnets you can try encoding the nucleotides as ints and use a numpy array, rather than process characters in a string.

In [5]:

```
# an important consideration is whether to change the case of the sequence.
# I always made the sequence uppercase (there's an option for that in pyfaidx)
# there's also a lot of info in not changing case at all... something you'll definitely want to consider

# this is how I get the sequence as a string instead of a Fasta region
try:
    seq = region.get_seq_from_fasta(fasta, kmer_size=kmer_size)
except KeyError:
    print('Fasta record {} : {} - {} not found'.format(region.chrom, region.start, region.stop),
          file=sys.stderr, flush=True)
    continue
if seq is None or 'N' in seq:
    continue

# ignore the error, this is just a code snippet
#              |
#              V
```

```
  File "<ipython-input-5-1dc24de2754e>", line 10
SyntaxError: 'continue' not properly in loop
```

It's also worth mentioning that I have an explicit class for regions that makes the code more readable.

In [6]:

```
class GRegion:

    def __init__(self, *args, **kwargs):
        self.default_names = ['chrom', 'start', 'stop', 'name', 'score', 'strand']
        self.fields = {'chrom': None, 'start': None, 'stop': None, 'name': None, 'score': None, 'strand': None}
        # self.fields = 6 * [None]
        for idx, key in enumerate(self.fields.keys()):
            if idx < len(args):
                self.fields[key] = str(args[idx]).strip()
        for key, value in kwargs.items():
            self.fields.update({key: value.strip()})

        try:
            self.fields['start'] = int(float(self.fields['start']))
            self.fields['stop'] = int(float(self.fields['stop']))
        except ValueError:
            raise ValueError('GRegion start and stop positions must be integers')

        self.chrom = self.fields['chrom']
        self.start = self.fields['start']
        self.stop = self.fields['stop']

        if self.fields['strand'] is not None:
            self.strand = self.fields['strand']
        else:
            self.strand = None

    def gnomad_rep(self):
        return '{}:{}-{}'.format(self.chrom, self.start, self.stop)

    def strand(self):
        try:
            return self.fields['strand']
        except KeyError:
            return 'none'

    def num_fields(self):
        num_fields = 0
        for v in self.fields.values():
            if v is not None:
                num_fields += 1
        return num_fields

    def add_field(self, key_string: str, value):
        # if not isinstance(key_string, str):
        #     raise KeyError('GRegion_object.add_field(key, value) must have a type \'str\' key')
        try:
            tval = self.fields.get(key_string)
            self.fields.update({key_string: value})
            return tval
        except KeyError:
            self.fields.update({key_string: value})
            return None

    def get_seq_from_fasta(self, fasta: Fasta, kmer_size=1):
        # shift start left by half ksize to capture nucleotide level mutability (default is no shift)
        fa_idx_start = max(self.start - kmer_size // 2 + 1, 0)
        fa_idx_stop = self.stop + kmer_size // 2
        try:
            return fasta.get_seq(self.chrom, fa_idx_start, fa_idx_stop).seq
        except (KeyError, FetchError):
            raise KeyError('Fasta record {} : {} - {} not found'.format(self.chrom, self.start, self.stop))

    def __str__(self):
        outstring = []
        headers = []
        for key, value in self.fields.items():
            if value is not None:
                if key in self.default_names:
                    outstring.append(str(value))
                else:
                    outstring.append(f'{value}')
        return '\t'.join(outstring)

    def __repr__(self):
        return str(self)

    def __hash__(self):
        return hash(str(self))

    def __eq__(self, other):
        return str(other) == str(self)

    def __lt__(self, other):
        if self.fields['chrom'] == other.fields['chrom']:
            return self.fields['start'] < other.fields['start']
        else:
            # Sort numeric chromosomes numerically and non-numeric ones lexicographically
            try:
                return int(''.join(filter(str.isdigit, self.fields['chrom']))) < int(
                    ''.join(filter(str.isdigit, other.fields['chrom'])))
            except ValueError:
                return self.fields['chrom'] < other.fields['chrom']
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-94830854d886> in <module>
----> 1 class GRegion:
      2 
      3     def __init__(self, *args, **kwargs):
      4         self.default_names = ['chrom', 'start', 'stop', 'name', 'score', 'strand']
      5         self.fields = {'chrom': None, 'start': None, 'stop': None, 'name': None, 'score': None, 'strand': Non
e}

<ipython-input-6-94830854d886> in GRegion()
     53             return None
     54 
---> 55     def get_seq_from_fasta(self, fasta: Fasta, kmer_size=1):
     56         # shift start left by half ksize to capture nucleotide level mutability (default is no shift)
     57         fa_idx_start = max(self.start - kmer_size // 2 + 1, 0)

NameError: name 'Fasta' is not defined
```

Again, please ignore the error. I didn't end up using a lot of these methods because, in some cases, it's faster to implement the method in a different way and others I anticipated using but never found a use for.

Please contact me with any questions if this code doesn't make any sense.

Here is where we start counting gnomAD stuff. You'll see that between this code and the previous code, there is some extra stuff. That's all for counting methylation if that's something you're interested in doing.

In [7]:

```
for variant in gnomadVCF(region.gnomad_rep()):
    varAC = variant.INFO.get('AC')
    if is_quality_snv(variant):
        num_all_variants += 1
        if varAC <= AC_cutoff:
            # here a "singleton" refers to any variant below the cutoff AC, sorry for the  bad variable name
            num_singletons += 1
            # welcome to indexing hell
            # VCF is 1-based[,], BED is 0-based [,)
            # zero based idx = (VCFidx - 1) - (BED start)
            seq_idx = variant.POS - region.start + kmer_size // 2 - 1
            forward_seq_context = seq[seq_idx - kmer_size // 2: seq_idx + kmer_size // 2 + 1]
            reverse_seq_context = ''.join([complement.get(base) for base in forward_seq_context[::-1]])
            sequences = [(forward_seq_context, variant.ALT[0]),
                         (reverse_seq_context, complement.get(variant.ALT[0]))]

            # more methylation stuff
            # this is for binning similarly methylated sites
            for seq_context, seq_variant in sequences:
                if 'N' in seq_context or len(seq_context) == 0:
                    continue

                if methylation_vcf_path is not None and is_cpg_ct_ga(seq_context, variant.ALT[0]):
                    methylation = base_methylation_probability_cache(
                        meth_vcf(f'{variant.CHROM}:{variant.POS}-{variant.POS}'))
                    if methylation < 0:
                        print(
                            f'No methylation data for {variant.CHROM}:{variant.POS}-{variant.POS} with context {seq_co
ntext}',
                            file=sys.stderr)
                        no_count += 1
                    elif methylation < 0.2:  # low/none
                        lo_count += 1
                        low_meth[seq_context][nuc_idx[variant.ALT[0]]] += 1
                    elif 0.2 <= methylation <= 0.6:
                        mid_count += 1
                        mid_meth[seq_context][nuc_idx[variant.ALT[0]]] += 1
                    elif 0.6 < methylation <= 1:
                        hi_count += 1
                        hi_meth[seq_context][nuc_idx[variant.ALT[0]]] += 1
                    else:
                        print('Irregular meth prob')

                rare_transitions_count[seq_context][nuc_idx[variant.ALT[0]]] += 1

    # GRegion class allows for as many custom fields as you want

# print summary of kmer search on GRegion fields to stdout
region.add_field('num_variants', f'AC_lte_{AC_cutoff}={num_singletons},all_snvs={num_all_variants}')
if methylation_vcf_path is not None:
    region.add_field('methylation',
                     f'low={lo_count},intermediate={mid_count},high={hi_count},no_data={no_count}')

# here we need to use a thread lock so that the terminal output doesn't write over itself.
# the lock is defined earlier so you won't see it in this block
with print_lock:
    print(str(region), flush=True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-dcb0ef0772b7> in <module>
----> 1 for variant in gnomadVCF(region.gnomad_rep()):
      2     varAC = variant.INFO.get('AC')
      3     if is_quality_snv(variant):
      4         num_all_variants += 1
      5         if varAC <= AC_cutoff:

NameError: name 'gnomadVCF' is not defined
```

In [9]:

```
# all this stuff at the end is for thread safe return values. they are basically glorified dictionaries.
def freeze_transitions(t_dict):
    newdict = defaultdict(tuple)
    for key, value in t_dict.items():
        newdict[key] = tuple(value)
    return frozenset(newdict.items())

if methylation_vcf_path is not None:
    return frozenset(reference_kmer_counts.items()), \
           frozenset(low_meth_kmer_counts.items()), \
           frozenset(mid_meth_kmer_counts.items()), \
           frozenset(hi_meth_kmer_counts.items()), \
           freeze_transitions(rare_transitions_count), \
           freeze_transitions(low_meth), \
           freeze_transitions(mid_meth), \
           freeze_transitions(hi_meth)
else:
    return frozenset(reference_kmer_counts.items()), freeze_transitions(rare_transitions_count)
```

```
  File "<ipython-input-9-7224fa1ed55d>", line 12
    return frozenset(reference_kmer_counts.items()), \
    ^
SyntaxError: 'return' outside function
```

Generating frequencies

In [10]:

```
def generate_frequencies(transitions_path, counts_path):
    ts = pd.read_csv(transitions_path, index_col=0).sort_index()
    cts = pd.read_csv(counts_path, index_col=0).sort_index()
    cts.columns = ['counts']
    # merge to align indices
    ts = ts.merge(cts, left_on=None, right_on=None, left_index=True, right_index=True)

    # commented out becuase doesn't consider all kmers that mutate
    # ts = ts.iloc[:, :4].div(ts.iloc[:, 4], axis=0)
    def row_freq_calc(row: pd.Series):
        row = row.astype(np.float128)
        for i, v in row.iteritems():
            row[i] = v / max((row[-1] - v), 0.1)
        return row

    ts = ts.apply(row_freq_calc, axis=1)
    # ts = ts.iloc[:, :4].div((ts.iloc[:, 4] - ts.iloc[:, :4].apply(sum, axis=1)), axis=0)
    return ts.iloc[:, :4]
```

This part you may want to play around with. You'll see there's some funky arithmetic going on.

Please get in touch with me if you have any questions about this code or any thing else in that gargantuan source file. I did try to separate concerns between the python files but there's definitely some overlap.

Cheers,

Simone

In [ ]: