Classification of long noncoding RNAs by k-mer content

Jessime M. Kirk [1,2,4,*], Daniel Sprague[1,3,4,*], and J. Mauro Calabrese[1,2,3,4#]

[1] Department of Pharmacology,

[2] Curriculum in Bioinformatics and Computational Biology,

[3] Curriculum in Pharmacology,

[4] Lineberger Comprehensive Cancer Center,

University of North Carolina at Chapel Hill, Chapel Hill, NC, 27599.

[*] Co-first authors

[#] Correspondence, jmcalabr@med.unc.edu

Abstract

K-mer based comparisons have emerged as powerful complements to BLAST-like alignment algorithms, particularly when the sequences being compared lack direct evolutionary relationships. In this chapter, we describe methods to compare k-mer content between groups of long noncoding RNAs (lncRNAs), to identify communities of lncRNAs with related k-mer contents, to identify the enrichment of protein-binding motifs in lncRNAs, and to scan for domains of related k-mer contents in lncRNAs. Our step-by-step instructions are complemented by Python code deposited in Github. Though our chapter focuses on lncRNAs, the methods we describe could be applied to any set of nucleic acid sequences.

1) Introduction

Upwards of 80% of the human genome can be transcribed into RNA. Of the total number of transcribed nucleotides, approximately one half comprise pre-messenger RNAs (pre-mRNAs) that will ultimately become spliced and encode for proteins in the cytoplasm. The other half comprise long noncoding RNAs (lncRNAs), defined as RNA species that are greater than 200 nucleotides in length and have little or no potential to encode for proteins. Compared to transcripts produced from protein-coding genes, lncRNAs are, on average, less conserved, transcribed at lower levels, spliced less efficiently, and more likely to remain in the nucleus [1–6].

Nevertheless, a growing number of lncRNAs have been studied experimentally, and are now known to play important roles in health and development. Some of the most notable of these include the lncRNA *XIST*, which orchestrates transcriptional silencing during X-chromosome Inactivation [7], the lncRNAs *NEAT1* and *MALAT1*, which play roles in nuclear organization and have context-dependent functions in development and in cancer [8–14], and the lncRNA *NORAD*, which helps to maintain genome stability by promoting DNA repair [15, 16]. LncRNAs have also been found to play important roles in developmental transitions [17–23], in the immune system [24–26], in the brain [27–33], and in the heart [34–37]. These identified roles, coupled with the large number of lncRNAs that have yet to be studied experimentally, suggest that lncRNAs with important physiological functions remain to be discovered.

Still, identifying function in lncRNAs remains a major challenge. Many lncRNAs are thought to function as hubs that concentrate proteins, DNA, and possibly other biomolecules in particular regions of the cell, yet the sequence characteristics that give rise to these functions and the mechanisms through which they occur are poorly defined, even for the best studied lncRNAs [38–42]. Moreover, relative to protein-coding genes, lncRNAs are poorly conserved, evolve rapidly, and are prone to changes in gene architecture, limiting the extent to which traditional phylogenetic analyses can be employed to identify the sequence features that are important for specifying their

function [43]. As an example, placental mammals express the *XIST* lncRNA to orchestrate gene silencing during X-Chromosome Inactivation [7], while marsupial mammals independently evolved their own lncRNA to orchestrate X-Chromosome Inactivation, termed *Rsx*. Remarkably, *XIST* and *Rsx* share no significant similarity by standard methods of sequence alignment [44, 45]. Thus, even though *Rsx* and *XIST* presumably function through analogous mechanisms, standard tools of sequence comparison are unable to detect the analogy. This problem extends to all lncRNAs. The sequence patterns that specify recurring functions in lncRNAs are largely unknown and difficult to detect computationally. Thus, to date, lncRNA functions must be determined empirically, on a case-by-case basis.

Recently, we developed a method of sequence comparison based on the notion that different lncRNAs likely encode similar functions through different spatial arrangements of related sequence motifs, and that such similarities might not be detectable by traditional methods of linear sequence alignment [46]. In our method, which we termed SEEKR (sequence evaluation through k-mer representation), the sequences of any number of lncRNAs are evaluated by comparing the standardized abundance of nucleotide substrings termed “k-mers” in each lncRNA, where k specifies the length of the substring being counted, and is typically set to values of k = 4, 5, or 6. SEEKR counts k-mers independent of their position in sequences of interest, much like the “bag of words model” used by many language processing algorithms, in which sentences are classified by word abundance without regards to grammar or syntax [47]. Using SEEKR, we demonstrated that k-mer content correlates with lncRNA subcellular localization, protein-binding, and repressive function, and that evolutionarily unrelated lncRNAs with analogous functions shared significant levels of non-linear sequence similarity even when BLAST-like alignment algorithms could detect none [46].

Below, we walk users through five related applications of SEEKR that we have found to be useful. For each application, we enumerate step-by-step instructions. Where relevant, we include

code to execute specific functions in python. We have deposited standalone python code to run the major applications of SEEKR in Github (https://github.com/CalabreseLab/seekr). For the simplest implementation of SEEKR, we refer users to a web portal (http://seekr.org). K-mer based classification schemes have been used in many biological contexts ([48–56] and others). Therefore, beyond lncRNAs, the methods that we describe should prove useful in the study of other nucleic acid sequences, such as 5' and 3' untranslated regions of mRNAs and DNA regulatory elements.

2) Materials

2.1) Hardware Requirements

Personal computer, preferably with a multi-core processor and at least 8GB of RAM.

2.2) Software Requirements

1. Python >=3.6. The easiest way to get started with Python is by downloading the Anaconda distribution: https://www.anaconda.com/download.

2. The python packages: numpy, pandas, networkx, python-igraph, louvain. All of these can be installed by running `$ pip install [name]`.

3. R, which can be installed from https://www.r-project.org/.

4. The R packages amap and ctc. amap is hosted at https://cran.r-project.org/web/packages/amap/index.html, and ctc at https://bioconductor.org/packages/release/bioc/html/ctc.html. Both can be installed by running: and can be installed by running:

```
source("http://bioconductor.org/biocLite.R")
biocLite("amap")
biocLite("ctc")
```

5. Java 1.8. See this page for help installing java:

https://www.java.com/en/download/help/download_options.xml

6. Java Treeview. http://jtreeview.sourceforge.net/

7. Gephi, which can be installed from https://gephi.org/users/download/.

8. SEEKR (optional). SEEKR is hosted at pypi: https://pypi.org/project/seekr/, and can be installed by running `$ pip install seekr`. SEEKR works on Mac and Linux. As of the time of publication, there is a bug installing several dependencies of SEEKR if using Anaconda Python on MacOS. As a workaround in macOS 10.14.x, run `$ MACOSX_DEPLOYMENT_TARGET=10.14 pip install seekr`. To print the documentation associated with each SEEKR command line tool, simply type the name of the tool in the UNIX terminal (e.g. `$ seekr_download_gencode`).

3) Methods

3.1) Comparing k-mer contents between a group of lncRNAs

3.1.1 *Download lncRNA sequences*

LncRNA sequences can be downloaded from https://www.gencodegenes.org/. For this analysis, we'll use human v22: ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_22/gencode.v22.lncRNA_transcripts.fa.gz and mouse v5: ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_mouse/release_M5/gencode.vM5.lncRNA_transcripts.fa.gz. Unzip these files to produce gencode.v22.lncRNA_transcripts.fa and gencode.vM5.lncRNA_transcripts.fa.gz. The following pipeline will be demonstrated using just the gencode.v22.lncRNA_transcripts.fa file. Mouse, or any other fasta file, can be substituted instead.

Downloading and unzipping can be done manually. Alternatively, if SEEKR is installed locally, you can also download the files from the command line. Use “lncRNA” to specify the biotype of transcripts file, and the “--release" flag to indicate you want a particular version of the fasta file:

```
$ seekr_download_gencode lncRNA -r 22
```

3.1.2 S*elect 01 isoform*

To avoid bias that may be introduced by counting k-mers across multiple isoforms of the same transcript, we typically only select transcripts ending in 01, which in prior versions of GENCODE, represented the canonical isoform of a gene product. Using this filter, each genomic locus is only represented once.

```
fasta_path = 'v22_lncRNA.fa'  (see note 1)
with open(fasta_path) as infasta:
    data = [l.strip() for l in infasta]
    headers = data[::2]
    seqs = data[1::2]


fasta01_path = 'v22-01.fa'
with open(fasta01_path, 'w') as outfasta:
    for header, seq in zip(headers, seqs):
        common_name = header.split('|')[4]
        if common_name.endswith('01'): (see note 2)
            outfasta.write(header+'\n')
            outfasta.write(seq+'\n')
```

To accomplish the same using the command line tool, pass `seekr_canonical_gencode` the name of the GENCODE fasta file and a path to the newly filtered fasta file:

```
$ seekr_canonical_gencode v22_lncRNA.fa v22-01.fa
```

3.1.3 Count k-mers

Next, we define a 2D matrix where each row represents one transcript, each column represents a k-mer, and each element is a normalized and standardized count of how many times a k-mer is found in a transcript. A single row of the matrix, then, defines a “k-mer profile” for a given lncRNA.

```
import pickle
import numpy as np
import pandas as pd

from collections import defaultdict
from itertools import product

# Read fasta file
fasta_path = 'v22-01.fa'
with open(fasta_path) as infasta:
    data = [l.strip() for l in infasta]
    headers = data[::2]
    seqs = data[1::2]

# Initialize data
k=6
kmers = [''.join(i) for i in product('AGTC', repeat=k)]
k_map = dict(zip(kmers, range(4**k)))
counts = np.zeros([len(seqs), 4**k], dtype=np.float32)
```

```
# Do counting
for i, seq in enumerate(seqs):
    row = counts[i]
    count_dict = defaultdict(int) (see note 3)
    length = len(seq)
    increment = 1000/length
    for c in range(length-k+1): (see note 4)
        kmer = seq[c:c+k]
        count_dict[kmer] += increment
    for kmer, n in count_dict.items():
        if kmer in k_map: (see note 5)
            row[k_map[kmer]] = n

# Normalize
counts -= np.mean(counts, axis=0)
counts /= np.std(counts, axis=0)
counts += abs(counts.min()) + 1 (see note 6)
counts = np.log2(counts)

# Save csv file
out_path = 'v22-6mers.csv'
seen = set()(see note 7)
names = []
for h in headers:
    name = h.split('|')[4]
    if name in seen:
```

```
        name += 'B'
    seen.add(name)
    names.append(name)
pickle.dump(names, open('v22_names-B.pkl', 'wb'))
df = pd.DataFrame(counts, names, kmers)
df.to_csv(out_path, float_format='%.4f')
```

Using the command line tool:

```
$ seekr_kmer_counts v22-01.fa -o v22_6mers.csv
```

3.2) Hierarchical clustering of lncRNAs by k-mer content

3.2.1 *Cluster with amap*

The visualization tool Java Treeview allows for interactive exploration of large hierarchical clusters. Treeview parses clusters defined by a set of three plaintext files, which describe the structure of row and column clusters: .gtr, .atr, and .cdt. These files can be conveniently produced by the R packages `amap` and `ctc`, which parse a .csv file such as v22_6mers.csv. The R script `treeview_cluster.r` will create the Treeview files:

```
make_treeview <- function(csv, out_gtr, out_atr, out_cdt){
  library(amap)
  library(ctc)

  kmers <- read.csv(csv, header=TRUE, row.names=1)
  kmers <- round(scale(kmers, scale=FALSE), 6)

  # Generate distance matrix using pearson distance
```

```
  dist_mat <- Dist(kmers, method="correlation",  nbproc=4)
  dist_mat_trans <- Dist(t(kmers), method="correlation",  nbproc=4)

  # Clustering using average agglomeration method
  clust_row <- hclust(dist_mat,method="average")
  clust_col <- hclust(dist_mat_trans,method="average")

  # Exporting the gtr, atr and cdt files
  r2gtr(clust_row,file=out_gtr, distance=clust_row$dist.method,
dec='.', digits=5)
  r2atr(clust_col,file=out_atr, distance=clust_col$dist.method,
dec='.', digits=5)
  r2cdt(clust_row, clust_col, kmers, labels=FALSE, description=FALSE,
file=out_cdt, dec='.')
}

args <- commandArgs(trailingOnly = TRUE)
do.call(make_treeview, as.list(args))
```

While this script is not part of the seekr module, it can be called from the command line using:

```
$ Rscript treeview_cluster.r v22_6mers.csv v22_6mers.gtr v22_6mers.atr
v22_6mers.cdt
```

3.2.2 *Visualize in java treeview*

Launch Treeview, and give Java access to plenty of memory. If insufficient memory is allocated, Treeview will not be able to open the .cdt file. Starting Treeview with 14GB of memory can be done by:

```
$ java -Xmx14000m -jar ~/Downloads/Setups/TreeView-1.1.6r4-
bin/TreeView.jar
```

Substitute the correct path to your TreeView.jar file. Once started, open v22-6mers.cdt via "File" -> "Open". After the file is loaded, change the visualization settings by: "Settings" -> "Pixel Settings...". Find the "Global" section of the pop-up window. Click "Fill" for both "X" and "Y". In the "Contrast" section, set "Value" to 1. In the "Colors" section click "YellowBlue". Close the pop-up window.

The image can be saved by "Export" -> "Save Thumbnail Image" -> "Save".

An ordered list of all transcript names can be exported as well. To do so, you must first select all transcripts. The easiest way to do this is to click on the far left of the dendrogram, so that a portion of the transcripts are highlighted in red. Hold down the up-arrow until all transcripts are highlighted in red. Then click "Export" -> "Save List" -> "Save".

3.3) Identifying communities of lncRNAs with related k-mer contents

It takes several steps to convert k-mer profiles into the form needed for identifying communities. We first build an adjacency matrix, describing all pairwise relationships between all lncRNAs, then use the matrix to build a network of lncRNAs. Finally, we can use a network algorithm to assign a community label to each lncRNA.

3.3.1 *Build adjacency matrix*

First, we need to build an adjacency matrix. This matrix describes how similar each lncRNA is to all other lncRNAs, as measured by their Pearson's r-values.

```
import pandas as pd
```

```
import numpy as np


counts = 'v22_6mers.csv'
counts = pd.read_csv(counts, index_col=0)
adjacency = np.corrcoef(counts.values)
adjacency = pd.DataFrame(adjacency, counts.index, counts.index)
adj_path = 'v22_adj.csv'
adjacency.to_csv(adj_path, float_format='%.4f')
```

To calculate our adjacency matrix, we want to compare our k-mer counts file against itself. The seekr command line tool is capable of comparing two separate counts files, so in this case, we need to pass our counts file twice:

```
$ seekr_pearson v22_6mers.csv v22_6mers.csv -o v22_adj.csv (see note 8)
```

3.3.2 *Sparsify the matrix*

To decrease the runtime of community calculation, we can reduce the number of edges in the network, by sparsifying the adjacency matrix by thresholding below a limit. That is, if the Pearson's r-value between two transcripts is less than the limit, we set that element of the matrix to 0, which removes that edge from our network. However, there is no single best threshold value; it depends heavily on the specific experiment and factors such as the k-mer size used. For example, smaller k-mer sizes will likely need higher thresholds. Therefore, it may be worthwhile to test multiple thresholds. One possible guideline is the mean and standard deviation of the r-values in the adjacency matrix. Two standard deviations above the mean (i.e. the 95th percentile in a normal distribution) is one viable threshold, and easy to compute. In our original publication of SEEKR,

we used 0.13 as a threshold, which is what we will use here, but we will also demonstrate how one would calculate a reasonable threshold *de novo*:

```
import pandas as pd
import numpy as np

adj = 'v22_adj.csv'
adjacency = pd.read_csv(adj, index_col=0)
print(adjacency.values.mean() + 2*adjacency.values.std())
limit = .13 (see note 9)
np.fill_diagonal(adjacency.values, 0) (see note 10)
adjacency[adjacency < limit] = 0
new_adj = 'v22_adj_p13.csv'
adjacency.to_csv(new_adj, float_format='%.4f')
```

In addition to calculating the mean and standard deviation, you can quickly visualize the adjacency matrix from the command line. This will create a pdf file that contains a graph of the distribution of all elements in the adjacency matrix and markings denoting the mean of the distribution as well as one and two standard deviations above the mean. Empirically, we have found that a Pearson's r value of two standard deviations above the mean provides an intuitive threshold that can be used to sparsify any adjacency matrix:

```
$ seekr_visualize_distro v22_adj.csv v22_adj.pdf
```

3.3.3 *Convert adjacency matrix to network and find communities*

Once the sparse adjacency matrix has been made, communities can be called with the Louvain algorithm. To use the Louvain algorithm, the adjacency matrix needs to be converted to a network. In this data structure, each lncRNA is represented as a "node" and each non-zero element of the

adjacency matrix represents an “edge” between two nodes, describing their similarity. The Louvain algorithm attempts to find communities of nodes having significantly more edges between the nodes within a given community than edges connecting nodes between different communities. Finally, we label each node with the name of the transcript and the community it's found in before saving the graph for visualization. In addition to saving the full graph, we will also produce a two-column csv file where the first column is the name of the lncRNA and the second is the community to which the lncRNA belongs.

```
import numpy as np

import networkx

import igraph

import louvain


adj = 'v22_adj_13.csv'

adjacency = pd.read_csv(adj, index_col=0)

graph = networkx. from_pandas_dataframe(adjacency)

adjacency = None (see note 11)


# Save subgraph

subgraphs = list(networkx.connected_component_subgraphs(graph))

graph_sizes = [sub.size() for sub in subgraphs]

main_sub = subgraphs[graph_sizes.index(max(graph_sizes))]

gml_path = 'v22_sub.gml'

networkx.write_gml(main_sub, gml_path) (see note 12)


# Find communities with Louvain

gamma = 1 (see note 13)
```

```
ig_graph = igraph.Graph.Read_GML(gml_path)
partition = louvain.find_partition(
    ig_graph, louvain.RBConfigurationVertexPartition,
    weights='weight', resolution_parameter=gamma)
n_comms = 5 (see note 14)
zipped = zip(main_sub.nodes(), partition.membership)
name2group = {k:v if v <= n_comms-1 else n_comms for k, v in zipped}
networkx.set_node_attributes(
    main_sub, name='Group', values=name2group)
networkx.write_gml(main_sub, gml_path)

# Save lncRNA communities to csv
with open('communities.csv') as out_file:
    for lncRNA in graph.nodes():
        group = name2group.get(lncRNA, n_comms-1)
        out_file.write(f'{lncRNA},{group}\n')
```

Again, the thresholding value is experiment specific. For that reason, it is a required argument for the command line script. In the instance below, we also save the full gml file, with the “-g" flag, and the two-column csv file listing lncRNAs and communities, with the “-c" flag:

```
$ seekr_graph v22_adj.csv 0.13 -g v22_sub.gml -c v22_comms.csv
```

3.3.4 *Visualize in Gephi*

Gephi is open-source software that is useful for visualizing lncRNA community graphs. On launch, Gephi will provide you with a "Welcome" pop-up window. In the "New Project" section, click "Open Graph File". Select `v22_sub.gml`. If loaded correctly, you will receive an "Import report" listing

the number of nodes and edges as well as other graph details. Click "OK". In the center of the main application window, you should see a small black circle. This is the default layout and coloring of the graph. Next, we'll color and properly layout the nodes. In the top left of the window, there will be an "Appearance" section. Click "Nodes" -> "Partition" -> "Choose an attribute" -> "Group" -> "Apply". After a few seconds, the nodes of the graph should be colored by group. On the bottom left, there is a section called "Layout". Click "Choose a layout" -> "Yifan Hu" -> "Run". Running the layout will take time. Progress can be tracked in the bottom right. Once finished, save the image by clicking: "File" -> "Export" -> "SVG/PDF/PNG file" -> "Options". Set "Width" and "Height" to 4096. Click "OK". Name your file and click "Okay" again. Saving the image will also some take time.

3.4) SEEKR Python

The command line tools are a convenient way to use SEEKR. However, to gain additional flexibility and performance, one can also consider using SEEKR as a Python module. The code below demonstrates the same pipeline as above (from downloading a fasta file from GENCODE to producing a csv file of lncRNA communities), but runs >10x faster than the command line tools:

```
import numpy as np
import pandas as pd
from seekr import fasta, kmer_counts, graph


downloader = fasta.Downloader()
downloader.get_gencode(biotype='lncRNA', release='22')
fasta_path = 'v22_lncRNA.fa'
fasta01_path = 'v22-01.fa'
maker = fasta.Maker(fasta_path, fasta01_path)
maker.filter1()
```

```
names = fasta.Maker(fasta01_path).names
counter = kmer_counts.BasicCounter(fasta01_path, log2=False)
counter.get_counts()
adj = pd.DataFrame(np.corrcoef(counter.counts), names, names)
comms_path = 'comms.csv'
gm = graph.Maker(adj, csv_path=comms_path, threshold=0.13,
leiden=False)
gm.make_gml_csv_files()
```

3.5) Scaling k-mer profiles by protein-binding motifs (Positional Weight Matrices)

One of the underlying assumptions of SEEKR is that lncRNAs derive function from the proteins that they bind. Therefore, a logical step is to utilize the k-mer profile of a given sequence to predict proteins that may bind that sequence. To do this, one can scale k-mer profiles by position weight matrix probabilities (PWMs). We outline this methodology below.

Our code is written to input PWMs in the format provided by the CisBP-RNA database [57]. To download, navigate to http://cisbp-rna.ccbr.utoronto.ca/bulk.php. In 'By Species', select Homo_sapiens, then click 'Download Species Archive' and in the new page click 'Download'. However, any PWM can be used if formatted correctly. Individual PWMs must be tab separated and saved in a .txt file. Each PWM must contain a header row with entries [Pos, A,C,G,U]. The 'Pos' column contains integers representing the position within the PWM. Each row must sum to 1, excluding the index column, thereby representing the probability of finding each nucleotide at each position within the motif.

This code iterates through the PWM files in `pwm_directory` and calculates the probability of observing all k-mers within each motif. The probability of observing a k-mer in a motif is calculated as the independent probability of observing each nucleotide of the k-mer at the

corresponding position within the motif. The weight is then the sum of possible frames that a k-mer could occur in, for example a 5-mer could fall in two different frames in a 6bp motif. Prior to running the code below, users need to derive k-mer counts in the lncRNAs of interest, as specified in Section 3.1.

```
import pandas as pd
import numpy as np
from itertools import product
from pathlib import Path

# path to PWMs
pwm_directory = 'cisbp_pwms/pwms_all_motifs/'

pwm_directory = Path(pwm_directory)

# k-mer counts are produced by seekr_kmer_counts (See section 3.1)
counts_path = 'v22_6mers.csv'

k = 5 (see note 15)
kmers = [''.join(p) for p in product('AGTC', repeat=k)] (see note 16)
z_scores = pd.read_csv(counts_path, index_col=0)
score_dict = {}
for pwm_path in pwm_directory.glob('*.txt'):
    try:
        pwm = pd.read_csv(pwm_path, sep='\t')
    except pd.errors.EmptyDataError:
```

```
        print(f'The motif file {pwm_path} is empty. Skipping.')
        continue
    pwm.drop('Pos', axis=1, inplace=True)
    pwm = pwm.rename(columns={'U': 'T'}).to_dict()
    kmer2weight = dict(zip(kmers, np.zeros(4 ** k)))
    motif_len = len(pwm['A']).
    if motif_len < k: (see note 17)
        kmers_within_kmer = [([kmer[i:i+4] for i in range(k-4+1)],
kmer) for kmer in kmers]
        n_kmers = motif_len - 4 + 1
        for sub_kmers, kmer in kmers_within_kmer:
            for sub_kmer in sub_kmers:
                for frame in range(n_kmers):
                    weight = 1
                    for pos, nucleotide in enumerate(sub_kmer):
                        weight *= pwm[nucleotide][pos + frame]
                    kmer2weight[kmer] += weight
    else:
        for kmer in kmers:
            n_kmers = motif_len - k + 1
            for frame in range(n_kmers):
                weight = 1
                for pos, nucleotide in enumerate(kmer):
                    weight *= pwm[nucleotide][pos+frame]
                kmer2weight[kmer] += weight
```

```
    sorted_weights = np.array([kmer2weight[k] for k in
z_scores.columns])
    weighted_z_scores = z_scores.values.copy() * sorted_weights
    scores_sums = weighted_z_scores.sum(axis=1)
    score_dict[pwm_path.name] = scores_sums

#save output
out_df = pd.DataFrame.from_dict(score_dict, orient='index',
columns=z_scores.index)
out_path = 'pwm_weighted_SEEKR.csv'
out_df.to_csv(out_path)
```

3.5.1 *Using the command line tool (specify k-mer length if not using k = 5):*

```
$ seekr_pwm cisbp_pwms/pwms_all_motifs v22_6mers.csv -k 6 -o
pwm_weighted_SEEKR.csv
```

3.6) Scanning lncRNAs for domains of related k-mer contents

This program is designed to scan a set of fasta sequences, or 'targets', for regions of high correlation to a set of sequences that we define as the 'query' sequences. Typical query sequences might represent functional domains in lncRNAs of interest. Targets are broken up into sliding windows with length and slide designated by the user. Correlations from each tile are then compared against a 'reference' set of sequences that are specified by the user.

This program iterates through k-mer counting three times, which we show explicitly below for completeness. The first iteration calculates the k-mer profile of a query sequence, the second iteration calculates the k-mer profiles for each tile in the target sequence, and the final iteration

calculates the k-mer profile for each transcript in the reference set of sequences and correlates them with the query k-mer profiles. This last calculation yields a distribution of Pearon's correlation values from which we can derive the ranks of our targets relative to the queries.

```
import pandas as pd
import numpy as np

from itertools import product
from collections import defaultdict
from scipy.stats import pearsonr
from scipy.stats import percentileofscore

from seekr.kmer_counts import BasicCounter
from seekr.fasta_reader import Reader

# Path to a query of interest (in this example, the sequence of repeat
# B in the lncRNA Xist)
query_path = 'mm10_xist_repeatB.fa'

# This performs standard SEEKR for the query
query = Reader(query_path).get_seqs()[0]
window = 1000 (see note 18)
slide = 100
k = 5
kmers = [''.join(p) for p in product('ATCG', repeat=k)]
k_map = dict(zip(kmers, range(4**k)))
```

```
mean_path, std_path = 'mean.npy', 'std.npy'
mean = np.load(mean_path)
std = np.load(std_path)

query_counter = BasicCounter(k=k, mean=mean, std=std)
query_counter.seqs = [query]
query_counter.get_counts()
query_counts = query_counter.counts

q_vs_t_rvals = []
target_path = 'mm10_kcnq1ot1.fa'
target = Reader(target_path).get_seqs()[0]
tiles = []
for i in range(0, len(target), slide):
    end = i + window
    tiles.append(target[i: end])
tiles[-1] += target[end:]

tile_counter = BasicCounter(k=k, mean=mean, std=std)
tile_counter.seqs = tiles
tile_counter.get_counts()

q_vs_t_rvals = np.array([pearsonr(query_counts[0],
tile_counter.counts[i])[0] for i in range(len(tiles))])
q_vs_ref_rvals = []
```

```
ref_path = 'v22-01.fa'
ref = Reader(ref_path).get_seqs()
ref_counter = BasicCounter(k=k, mean=mean, std=std)
ref_counter.seqs = ref
ref_counter.get_counts()
ref_counts = ref_counter.counts


q_vs_ref_rvals = np.array([pearsonr(query_counts[0], ref_counts[i])[0]
for i in range(len(ref))])
ranks = []
for tile_corr in q_vs_t_rvals:
 ranks.append(percentileofscore(q_vs_ref_rvals, tile_corr,
kind='rank'))


query_target_df = pd.DataFrame(q_vs_t_rvals)
query_target_df_out = 'query_target_pearson.csv'
query_target_df.to_csv(query_target_df_out)
ranks_df = pd.DataFrame(ranks)
ranks_df_path = 'ranks.csv'
ranks_df.to_csv(ranks_df_path)
```

3.6.1 *Command line usage*

This tool requires several pieces of data. 1) A fasta file containing one or more query sequences, 2) A second fasta file containing one or more target sequences which will be tiled into domains, 3) The mean and standard deviation vectors for normalization (e.g. appropriate output from `seekr_norm_vectors`). You can then select the locations for one or both of the possible output

files with the '-r' and the '-p' flags. The '-r' flag prints a matrix of Pearson's r values describing the similarity between each query and each tile in each target, and the '-p' flag prints a corresponding matrix of the percentile rankings of the Pearson's r values relative to a reference set of sequences. If you use the '-p' flag you must also use the '-rp' flag, which specifies the reference set of sequences to be used in percentile calculations; for example, 'v22-01.fa'. Also, ensure that the '-k' flags passed to `seekr_norm_vectors` and `seekr_domain_pearson` are the same:

```
$ seekr_norm_vectors v22-01.fa –k 5
$ seekr_domain_pearson mm10_xist_repeatB.fa mm10_kcnq1ot1.fa mean.npy std.npy -rp v22-01.fa –k 5 –r r_values.csv –p percentiles.csv
```

4) Notes

1. If you manually download this file from GENCODE's website, it will be called "gencode.v22.lncRNA_transcripts.fa".
2. In human, there are a few genomic regions where the canonical isoform is not -001, but -*01 instead, usually -201. It is worth manually examining the GENCODE annotations to ensure that your lncRNA spliceform of interest is included in your analyses.
3. While it is possible to directly increment the numpy array for each k-mer, randomly accessing the array is slow when done billions of times. Instead, a dictionary is used to collect the counts for a single transcript. That way, each element of the array can be accessed only once.
4. Because we want to count overlapping k-mers we cannot use Python's built-in `count` method, and need to manually iterate over the strings ourselves.
5. k-mers that contain non-ACGT nucleotides (eg. ATCGGN) are skipped.
6. In our original publication describing SEEKR, log normalization was not used. In a limited number of tests, we have found that log normalizing k-mer counts prior to performing

Pearson's correlation mildly improves our ability to detect biological meaningful trends. In general, log normalization is an appropriate way to reduce skew in data, and k-mer counts, especially in repetitive regions of RNA, are often skewed. If log2 normalization is not desired, pass the `-nl` flag to `seekr_kmer_counts`.

7. GENCODE transcript names are not necessarily unique. To be able to use names as the index of an R DataFrame, 'B' is appended to transcript names that have already occurred.
8. This array is approximately 250 million elements (16,000 by 16,000 r-values). For a significant efficiency increase (~50x speed, 2x space), consider using the binary flags `--binary_input` and/or `--binary_output` when running `seekr_pearson`. Also note that usage of these flags may require additional adjustment to flags in other stages of the SEEKR pipeline.
9. The 0.13 Pearson value as a threshold was chosen as a balance between computational efficiency and information retention. In GENCODE v22, the Pearson's value of 0.13 is approximately two standard deviations above the mean similarity between all pairwise lncRNA comparisons. Overall, we found little to no difference in community definition, correlation with lncRNA localization, or ability to predict protein-binding patterns over a range of limit values.
10. The diagonal of the matrix contains all '1' values, since the k-mer profile of a transcript versus itself is a perfect correlation. These edges are not useful for defining communities, so we remove them.
11. This is just done to clear some memory.
12. Writing out to disk at this point is simply used as a way to convert between a networkx graph and an igraph graph. The igraph version is needed for running louvain. There are likely better ways of doing this.
13. Gamma is the resolution parameter for the Louvain algorithm, and is used to tune how many communities are found. Gamma must be greater than 0, and the larger the value,

the more communities will be created; consequentially, community sizes are smaller at larger gamma values. We chose to stay with the default resolution parameter, 1, which was supported by CHAMP [58]. CHAMP is an algorithm which can help provide context for which values of gamma might be most appropriate for a given graph.

14. Choosing the number of communities can be difficult. We used an estimate based on the hierarchical heatmap, in combination with the size of the communities. In our original publication of SEEKR, community 6 was significantly smaller than community 5 (relative to the ratio between, community 5 and 4, or 4 and 3, etc.; [46]). n_communities is defined here so we can cap the number of communities found by the Louvain algorithm before adding values to the main subgraph below.
15. k = 5 is a reasonable default because it tends to strike a balance between decreasing sparsity in k-mer profiles while still retaining good discrimination between queries and targets.
16. Nucleotide entries in this list must be in exactly the same order as used in Section 3.1, 'AGTC'.
17. This loop is designed to find all 4-mers within the larger k-mer if the value of k is larger than the length of the motif. For example, the 5-mer ATCGT does not exist within a 4 base pair motif, but two 4-mers within the 5-mer, ATCG and TCGT can fit within a 4 base pair motif. This loop calculates the probabilities of observing the 4-mers separately and then sums the result. No motif in CisBP-RNA database is < 4 base pairs, hence the default of k = 4.
18. The window and slide variables can be set to any positive integer. In our work, we have found that a window approximately the size of the query features, such as the tandem repeat domains of *Xist*, provides good results. In general, increasing the window size smoothens the resulting data whereas decreasing window size gives more detail but increases noise. The slide is best adjusted as a function of the size of your target dataset.

If only a couple sequences are being considered, a slide of 1 may be appropriate, but if the study is over the entire transcriptome or otherwise genome-wide, then larger slides can reduce compute time and storage space exponentially.

References

1. Cabili M, Trapnell C, Goff L, et al (2011) Integrative annotation of human large intergenic noncoding RNAs reveals global properties and specific subclasses. Genes Dev 25:1915–1927. doi:10.1101/gad.17446611
2. Cabili MN, Dunagin MC, McClanahan PD, et al (2015) Localization and abundance analysis of human lncRNAs at single-cell and single-molecule resolution. Genome Biol 16:20. doi:10.1186/s13059-015-0586-4
3. Derrien T, Johnson R, Bussotti G, et al (2012) The GENCODE v7 catalog of human long noncoding RNAs: Analysis of their gene structure, evolution, and expression. Genome Res 22:1775–1789. doi:10.1101/gr.132159.111
4. Melé M, Mattioli K, Mallard W, et al (2017) Chromatin environment, transcriptional regulation, and splicing distinguish lincRNAs and mRNAs. Genome Res 27:27–37. doi:10.1101/gr.214205.116
5. Mukherjee N, Calviello L, Hirsekorn A, et al (2017) Integrative classification of human coding and noncoding genes through RNA metabolism profiles. Nat Struct Mol Biol 24:86–96. doi:10.1038/nsmb.3325
6. Iyer MK, Niknafs YS, Malik R, et al (2015) The landscape of long noncoding RNAs in the human transcriptome. Nat Genet 24:86–96. doi:10.1038/ng.3192
7. Sahakyan A, Yang Y, Plath K (2018) The Role of Xist in X-Chromosome Dosage Compensation. Trends Cell Biol 28:999–1013. doi:10.1016/J.TCB.2018.05.005
8. West JA, Davis CP, Sunwoo H, et al (2014) The Long Noncoding RNAs NEAT1 and MALAT1 Bind Active Chromatin Sites. Mol Cell 55:791–802. doi:10.1016/j.molcel.2014.07.012

9. Arun G, Diermeier S, Akerman M, et al (2016) Differentiation of mammary tumors and reduction in metastasis upon Malat1 lncRNA loss. Genes Dev 30:34–51. doi:10.1101/gad.270959.115

10. Chakravarty D, Sboner A, Nair SS, et al (2014) The oestrogen receptor alpha-regulated lncRNA NEAT1 is a critical modulator of prostate cancer. Nat Commun 20:1844–1849. doi:10.1038/ncomms6383

11. Gutschner T, Hämmerle M, Eißmann M, et al (2013) The noncoding RNA MALAT1 is a critical regulator of the metastasis phenotype of lung cancer cells. Cancer Res 73:1180–1189. doi:10.1158/0008-5472.CAN-12-2850

12. Zhang B, Arun G, Mao YS, et al (2012) The lncRNA malat1 is dispensable for mouse development but its transcription plays a cis-regulatory role in the adult. Cell Rep 2:111–123. doi:10.1016/j.celrep.2012.06.003

13. Nakagawa S, Shimada M, Yanaka K, et al (2014) The lncRNA Neat1 is required for corpus luteum formation and the establishment of pregnancy in a subpopulation of mice. Development 141:4618–4627. doi:10.1242/dev.110544

14. Standaert L, Adriaens C, Radaelli E, et al (2014) The long noncoding RNA Neat1 is required for mammary gland development and lactation. RNA 20:1844–1489. doi:10.1261/rna.047332.114

15. Lee S, Kopp F, Chang TC, et al (2016) Noncoding RNA NORAD Regulates Genomic Stability by Sequestering PUMILIO Proteins. Cell 164:69–80. doi:10.1016/j.cell.2015.12.017

16. Munschauer M, Nguyen CT, Sirokman K, et al (2018) The NORAD lncRNA assembles a topoisomerase complex critical for genome stability. Nature 561:132–6.

doi:10.1038/s41586-018-0453-z

17. Klattenhoff CA, Scheuermann JC, Surface LE, et al (2013) Braveheart, a long noncoding RNA required for cardiovascular lineage commitment. Cell 152:570–583. doi:10.1016/j.cell.2013.01.003

18. Lin N, Chang KY, Li Z, et al (2014) An evolutionarily conserved long noncoding RNA TUNA controls pluripotency and neural lineage commitment. Mol Cell 53:1005–1024. doi:10.1016/j.molcel.2014.01.021

19. Luo S, Lu JY, Liu L, et al (2016) Divergent lncRNAs regulate gene expression and lineage differentiation in pluripotent cells. Cell Stem Cell 18:637–652. doi:10.1016/j.stem.2016.01.024

20. Ng SY, Johnson R, Stanton LW (2012) Human long non-coding RNAs promote pluripotency and neuronal differentiation by association with chromatin modifiers and transcription factors. EMBO J 31:522–533. doi:10.1038/emboj.2011.459

21. Mohamed JS, Gaughwin PM, Lim B, et al (2010) Conserved long noncoding RNAs transcriptionally regulated by Oct4 and Nanog modulate pluripotency in mouse embryonic stem cells. RNA 16:324–337. doi:10.1261/rna.1441510

22. Lai KMV, Gong G, Atanasio A, et al (2015) Diverse phenotypes and specific transcription patterns in twenty mouse lines with ablated lincRNAs. PLoS One 10:e0125522. doi:10.1371/journal.pone.0125522

23. Swiezewski S, Liu F, Magusin A, Dean C (2009) Cold-induced silencing by long antisense transcripts of an Arabidopsis Polycomb target. Nature 462:799–802. doi:10.1038/nature08618

24. Carpenter S, Aiello D, Atianand MK, et al (2013) A long noncoding RNA mediates both activation and repression of immune response genes. Science (80-) 341:789–792. doi:10.1126/science.1240925

25. Elling R, Robinson EK, Shapleigh B, et al (2018) Genetic Models Reveal cis and trans Immune-Regulatory Activities for lincRNA-Cox2. Cell Rep 25:1511–1524. doi:10.1016/j.celrep.2018.10.027

26. Kotzin JJ, Spencer SP, McCright SJ, et al (2016) The long non-coding RNA Morrbid regulates Bim and short-lived myeloid cell lifespan. Nature 537:239–243. doi:10.1038/nature19346

27. Barry G, Briggs JA, Vanichkina DP, et al (2013) The long non-coding RNA Gomafu is acutely regulated in response to neuronal activation and involved in schizophrenia-associated alternative splicing. Mol Psychiatry 19:486

28. Goff LA, Groff AF, Sauvageau M, et al (2015) Spatiotemporal expression and transcriptional perturbations by long noncoding RNAs in the mouse brain. Proc Natl Acad Sci 112:6855–6862. doi:10.1073/pnas.1411263112

29. Mercer TR, Dinger ME, Sunkin SM, et al (2008) Specific expression of long noncoding RNAs in the mouse brain. Proc Natl Acad Sci U S A 105:716–721. doi:10.1073/pnas.0706729105

30. Powell WT, Coulson RL, Crary FK, et al (2013) A Prader-Willi locus lncRNA cloud modulates diurnal genes and energy expenditure. Hum Mol Genet 22:4318–4328. doi:10.1093/hmg/ddt281

31. Raveendra BL, Swarnkar S, Avchalumov Y, et al (2018) Long noncoding RNA GM12371

acts as a transcriptional regulator of synapse function. Proc Natl Acad Sci 115:10197–10205. doi:10.1073/pnas.1722587115

32. Sauvageau M, Goff LA, Lodato S, et al (2013) Multiple knockout mouse models reveal lincRNAs are required for life and brain development. Elife 2013:e01749. doi:10.7554/elife.01749

33. Sone M, Hayashi T, Tarui H, et al (2007) The mRNA-like noncoding RNA Gomafu constitutes a novel nuclear domain in a subset of neurons. J Cell Sci 120:2498–2506. doi:10.1242/jcs.009357

34. Grote P, Wittler L, Hendrix D, et al (2013) The Tissue-Specific lncRNA Fendrr Is an Essential Regulator of Heart and Body Wall Development in the Mouse. Dev Cell 24:206–214. doi:10.1016/j.devcel.2012.12.012

35. Han P, Li W, Lin CH, et al (2014) A long noncoding RNA protects the heart from pathological hypertrophy. Nature 514:102–106. doi:10.1038/nature13596

36. Matkovich SJ, Edwards JR, Grossenheider TC, et al (2014) Epigenetic coordination of embryonic heart transcription by dynamically regulated long noncoding RNAs. Proc Natl Acad Sci 111:12264–12269. doi:10.1073/pnas.1410622111

37. Wang K, Liu CY, Zhou LY, et al (2015) APF lncRNA regulates autophagy and myocardial infarction by targeting miR-188-3p. Nat Commun 2015:6779. doi:10.1038/ncomms7779

38. Kopp F, Mendell JT (2018) Functional Classification and Experimental Dissection of Long Noncoding RNAs. Cell 393–407

39. Geisler S, Coller J (2013) RNA in unexpected places: long non-coding RNA functions in diverse cellular contexts. Nat Rev Mol Cell Biol 14:699–712. doi:10.1038/nrm3679

40. Guttman M, Rinn JL (2012) Modular regulatory principles of large non-coding RNAs. Nature 482:339–346. doi:10.1038/nature10887

41. Rinn JL, Chang HY (2012) Genome regulation by long noncoding RNAs. Annu Rev Biochem 81:145–166. doi:10.1146/annurev-biochem-051410-092902

42. Kornienko AE, Guenzl PM, Barlow DP, Pauler FM (2013) Gene regulation by the act of long non-coding RNA transcription. BMC Biol 11:59. doi:10.1186/1741-7007-11-59

43. Hezroni H, Koppstein D, Schwartz MG, et al (2015) Principles of long noncoding RNA evolution derived from direct comparison of transcriptomes in 17 species. Cell Rep 11:1110–1122. doi:10.1016/j.celrep.2015.04.023

44. Grant J, Mahadevaiah SK, Khil P, et al (2012) Rsx is a metatherian RNA with Xist-like properties in X-chromosome inactivation. Nature 487:254–258. doi:10.1038/nature11171

45. Johnson RN, O'Meally D, Chen Z, et al (2018) Adaptation and conservation insights from the koala genome. Nat Genet 50:1102–1111. doi:10.1038/s41588-018-0153-5

46. Kirk JM, Kim SO, Inoue K, et al (2018) Functional classification of long non-coding RNAs by k-mer content. Nat Genet 50:1474–1482. doi:10.1038/s41588-018-0207-8

47. McTear M, Callejas Z, Griol D (2016) The Conversational Interface

48. Blaisdell BE (1989) Effectiveness of measures requiring and not requiring prior sequence alignment for estimating the dissimilarity of natural sequences. J Mol Evol 29:526–537

49. Burge C, Campbell AM, Karlin S (1992) Over- and under-representation of short oligonucleotides in DNA sequences. Proc Natl Acad Sci U S A 89:1358–1362. doi:10.1073/pnas.89.4.1358

50. Kari L, Hill KA, Sayem AS, et al (2015) Mapping the space of genomic signatures. PLoS One 10:e0119815. doi:10.1371/journal.pone.0119815

51. Lees JA, Vehkala M, Valimaki N, et al (2016) Sequence element enrichment analysis to determine the genetic basis of bacterial phenotypes. Nat Commun 7:12797. doi:10.1038/ncomms12797

52. Pandey P, Bender MA, Johnson R, et al (2018) Squeakr: an exact and approximate k-mer counting system. Bioinformatics 34:568–575. doi:10.1093/bioinformatics/btx636

53. Blanchette M, Tompa M (2002) Discovery of regulatory elements by a computational method for phylogenetic footprinting. Genome Res. 12:739–748

54. Dubinkina VB, Ischenko DS, Ulyantsev VI, et al (2016) Assessment of k-mer spectrum applicability for metagenomic dissimilarity analysis. BMC Bioinformatics 17:38. doi:10.1186/s12859-015-0875-7

55. Friedman RC, Farh KK-H, Burge CB, Bartel DP (2009) Most mammalian mRNAs are conserved targets of microRNAs. Genome Res 19:92–105. doi:10.1101/gr.082701.108

56. Solis-Reyes S, Avino M, Poon A, Kari L (2018) An open-source k-mer based machine learning tool for fast and accurate subtyping of HIV-1 genomes. PLoS One 13:e0206409. doi:10.1371/journal.pone.0206409

57. Ray D, Kazan H, Cook KB, et al (2013) A compendium of RNA-binding motifs for decoding gene regulation. Nature 499:172–177. doi:10.1038/nature12311

58. Weir WH, Emmons S, Gibson R, et al (2017) Post-Processing Partitions to Identify Domains of Modularity Optimization. Algorithms 10:. doi:10.3390/a10030093