Bioinformatics III

Eighth Assignment

Thibault Schowing (2571837)
Wiebke Schmitt (2543675)

June 14, 2018

Exercise 8.1: Data Preprocessing

(a) **Data matrix:** *The supplement contains the data˙matrix.py–file with the outline of a DataMatrix–class in which you should complete.*

Listing 1: Data Matrix class script

```
import pandas as pd
from scipy import stats

class DataMatrix:
    def __init__(self, file_path):
        """
        :param file_path: path to the input matrix file
        """
        self.file_path = file_path
        self.df = None

        # read the matrix in the input file, remove rows with empty values and
            merge duplicate rows
        self.read_data()

    def read_data(self):
        """
        Reads data from a given matrix file, where the first line gives the
            names of the columns and the first column
        gives the names of the rows. Removes rows with empty or non-numerical
            values and merges rows with the same
        name into one.
        """

        # Read the file in a pandas DataFrame
        self.df = pd.read_csv(self.file_path, index_col=False, sep='\t')

        # Drop all NAN before setting the first columns as index, as some
            index label might be NaN/empty
        self.df.dropna(axis=0, how="any", inplace=True)

        # Change the first column's name
        new_columns = self.df.columns.values
        new_columns[0] = "Index"
        self.df.columns = new_columns

        # Sort values for later use (to_tsv)
        self.df = self.df.sort_values('Index')

        # Group by index: remove duplicate rows by meaning the rows values
        # Set 'Index' as index automatically
        self.df = self.df.groupby('Index').mean()
```

```
        # Print to console to have a nice overview
        # print(self.df)


    def get_rows(self):
        """
        :return: dictionary with keys = row names, values = list of row values
        """
        rows = {}

        for index, row in self.df.iterrows():
            rows[index] = list(row)

        return rows


    def get_columns(self):
        """
        :return: dictionary with keys = column names, values = list of column
            values
        """
        cols = {}
        for name, values in self.df.iteritems():
            cols[name] = list(values)

        return cols

    def not_normal_distributed(self, alpha, rows):
        """
        Uses the Shapiro-Wilk test to compute all rows (or columns) that are
            not normally distributed.
        :param alpha: significance threshold
        :param rows: True if the Shapiro-Wilk p-values should be computed for
            the rows, False if for the columns
        :return: dictionary with keys = row/columns names, values = Shapiro-
            Wilk p-value
        """

        ret = {}

        if rows:
            tmp = self.get_rows()
        else:
            tmp = self.get_columns()

        for key, value in tmp.items():
            shapiro = stats.shapiro(tmp[key])
            pvalue = shapiro[1]

            if pvalue < alpha:
                ret[key] = pvalue

        return ret

    def to_tsv(self, file_path):
        """
        Writes the processed matrix into a tab-separated file, with the same
            column order as the input matrix and
        the rows in lexicographical order.
        :param file_path: path to the output file
        """

        self.df.to_csv(file_path, sep='\t')
```

(b) **Process expression and methylation data:** *In the function exercise 1() in main.py, use your DataMatrix–class to read in the expression and methylation tables given in the supple-*

ment and write the processed matrices into files. [1]

Listing 2: Main programm

```
from data_matrix import DataMatrix
from network import CorrelationNetwork
from correlation import CorrelationMatrix
from cluster import CorrelationClustering

def dict_to_file(dict, path):
    """

    :param dict: Dictionnary you want to write to file
    :param path: Path or filename
    :return: nada
    """
    fout = path
    fo = open(fout, "w")

    for k, v in dict.items():
        fo.write(str(k) + ' -> ' + str(v) + '\n')

    fo.close()

def exercise_1():
    # Read data
    data_expression = DataMatrix("./expression.tsv")
    data_methylation = DataMatrix("./methylation.tsv")

    # Uses the Shapiro-Wilk test to test if the data follow a normal
        distribution
    ALPHA = 0.05

    not_normal_expression_genes = data_expression.not_normal_distributed(ALPHA
        , True)
    dict_to_file(not_normal_expression_genes, "./not_normal_expression_genes.
        txt")
    print("Number of genes whose data does not follow a normal distribution (
        EXPRESSION): ", len(not_normal_expression_genes))

    not_normal_expression_sample = data_expression.not_normal_distributed(
        ALPHA, False)
    dict_to_file(not_normal_expression_sample, "./not_normal_expression_sample
        .txt")
    print("Number of sample whose data does not follow a normal distribution (
        EXPRESSION): ", len(not_normal_expression_sample))

    not_normal_methylation_genes = data_methylation.not_normal_distributed(
        ALPHA, True)
    dict_to_file(not_normal_methylation_genes, "./not_normal_methylation_genes
        .txt")
    print("Number of genes whose data does not follow a normal distribution (
        METHYLATION): ", len(not_normal_methylation_genes))

    not_normal_methylation_sample = data_methylation.not_normal_distributed(
        ALPHA, False)
    dict_to_file(not_normal_methylation_sample, "./
        not_normal_methylation_sample.txt")
    print("Number of sample whose data does not follow a normal distribution (
        METHYLATION): ", len(not_normal_methylation_sample))


    # Write processed matrix to file
    data_expression.to_tsv("schmitt_schowing_expression.tsv")
    data_methylation.to_tsv("schmitt_schowing_methylation.tsv")
```

[1] The files are attached with the source files in the email.

```
def exercise_3():
    #

    # Read data
    data_expression = DataMatrix("./expression.tsv")
    data_methylation = DataMatrix("./methylation.tsv")

    NETWORK_THRESHOLD = 0.75

    # Expression
    cm = CorrelationMatrix(data_expression, "Pearson", True)
    cn = CorrelationNetwork(cm,NETWORK_THRESHOLD)
    cn.to_sif("./schmitt_schowing_expression_network_pearson.sif")

    cm = CorrelationMatrix(data_expression, "Spearman", True)
    cn = CorrelationNetwork(cm, NETWORK_THRESHOLD)
    cn.to_sif("./schmitt_schowing_expression_network_spearman.sif")

    cm = CorrelationMatrix(data_expression, "Kendall", True)
    cn = CorrelationNetwork(cm, NETWORK_THRESHOLD)
    cn.to_sif("./schmitt_schowing_expression_network_kendall.sif")

    # Methylation
    cm = CorrelationMatrix(data_methylation, "Pearson", True)
    cn = CorrelationNetwork(cm,NETWORK_THRESHOLD)
    cn.to_sif("./schmitt_schowing_methylation_network_pearson.sif")

    cm = CorrelationMatrix(data_methylation, "Spearman", True)
    cn = CorrelationNetwork(cm, NETWORK_THRESHOLD)
    cn.to_sif("./schmitt_schowing_methylation_network_spearman.sif")

    cm = CorrelationMatrix(data_methylation, "Kendall", True)
    cn = CorrelationNetwork(cm, NETWORK_THRESHOLD)
    cn.to_sif("./schmitt_schowing_methylation_network_kendall.sif")


def exercise_4():
    # TODO
    # correlation matrix -> columns and not rows

    data_expression = DataMatrix("./expression.tsv")
    data_methylation = DataMatrix("./methylation.tsv")

    # With the expression data
    cm = CorrelationMatrix(data_expression, "Kendall", False)
    cc = CorrelationClustering(cm)
    cc.trace_to_tsv("schmitt_schowing_expression_cluster_kendall.tsv")

    cm = CorrelationMatrix(data_expression, "Pearson", False)
    cc = CorrelationClustering(cm)
    cc.trace_to_tsv("schmitt_schowing_expression_cluster_pearson.tsv")

    cm = CorrelationMatrix(data_expression, "Spearman", False)
    cc = CorrelationClustering(cm)
    cc.trace_to_tsv("schmitt_schowing_expression_cluster_spearman.tsv")

    # With the methylation data
    cm = CorrelationMatrix(data_methylation, "Kendall", False)
    cc = CorrelationClustering(cm)
    cc.trace_to_tsv("schmitt_schowing_methylation_cluster_kendall.tsv")

    cm = CorrelationMatrix(data_methylation, "Pearson", False)
    cc = CorrelationClustering(cm)
```

```
115     cc.trace_to_tsv("schmitt_schowing_methylation_cluster_pearson.tsv")

        cm = CorrelationMatrix(data_methylation, "Spearman", False)
        cc = CorrelationClustering(cm)
        cc.trace_to_tsv("schmitt_schowing_methylation_cluster_spearman.tsv")
120

# only execute the following if this module is the entry point of the program,
      not when it is imported into another file
if __name__ == '__main__':
        exercise_1()
125     exercise_3()
        exercise_4()
```

For each input file, report the number of genes and samples whose data does not follow a normal distribution with $\alpha = 0.05$.

Number of genes whose data does not follow a normal distribution (EXPRESSION): 73

Number of sample whose data does not follow a normal distribution (EXPRESSION): 19

Number of genes whose data does not follow a normal distribution (METHYLATION): 66

Number of sample whose data does not follow a normal distribution (METHYLATION): 19

Exercise 8.2: Correlation Measures

Listing 3: Correlation matrix

```
from itertools import combinations
from scipy import stats


def rank(x):
    """
    :param x: a list of values
    :return: ranking of the input list
    """

    return stats.rankdata(x)


def pearson_correlation(x, y):
    """
    :param x: a list of values
    :param y: a list of values
    :return: Pearson correlation coefficient of X and Y
    """

    return stats.pearsonr(x, y)[0]


def spearman_correlation(x, y):
    """
    :param x: a list of values
    :param y: a list of values
    :return: Spearman correlation coefficient of X and Y
    """

    return stats.spearmanr(x, y)[0]


def kendall_correlation(x, y):
    """
    :param x: a list of values
    :param y: a list of values
    :return: Kendall-B correlation coefficient of X and Y
    """

    return stats.kendalltau(x, y)[0]


class CorrelationMatrix(dict):
    """
    This class behaves like a dictionary, where the correlation between two
        elements 1 and 2 is accessible via
    cor_matrix[(element_1, element_2)] or cor_matrix[(element_2, element_1)] since
        the matrix is symmetrical.
    It also stores the row (or column) names of the input DataMatrix.
    """
    def __init__(self, data_matrix, method, rows):
        """
        :param data_matrix: a DataMatrix (see data_matrix.py)
        :param method: string specifying the correlation method, must be 'Pearson',
            'Spearman' or 'Kendall'
        :param rows: True if the correlation matrix should be constructed for the
            rows, False if for the columns
        """
        # initialise the dictionary
        super().__init__(self)

        # if rows = True, then compute the correlation matrix for the row data
```

```
        if rows:
60          data = data_matrix.get_rows()
        # if rows = False, then compute the correlation matrix for the column data
        else:
            data = data_matrix.get_columns()

65      # sorted list of row names (or column names) in the input data matrix
        self.names = list(sorted(data.keys()))

        # compute the correlation between all pairs of rows (or columns)
        for name_1, name_2 in combinations(data.keys(), 2):
70          # use the specified correlation method
            if method == 'Pearson':
                correlation = pearson_correlation(data[name_1], data[name_2])
            elif method == 'Spearman':
                correlation = spearman_correlation(data[name_1], data[name_2])
75          elif method == 'Kendall':
                correlation = kendall_correlation(data[name_1], data[name_2])
            else:
                raise ValueError('The correlation method not supported must be
                    either Pearson, Spearman or Kendall.')

80          # add the correlation symmetrically
            self[(name_1, name_2)] = correlation
            self[(name_2, name_1)] = correlation
```

Exercise 8.3: Gene Co–Expression Networks

(a) Network construction

Listing 4: Correlation network

```
import collections
import pandas as pd
import math

class CorrelationNetwork:
    def __init__(self, correlation_matrix, threshold):
        """
        Constructs a co-expression network from a correlation matrix by adding
             edges between nodes with absolute
        correlation bigger than the given threshold.
        :param correlation_matrix: a CorrelationMatrix (see correlation.py)
        :param threshold: a float between 0 and 1
        """

        interactions = []
        for tup, corr in correlation_matrix.items():

            correlation = str(round(corr, 2))
            node0 = tup[0]
            node1 = tup[1]

            tmp = [node0, node1, correlation]
            tmp.sort(reverse=True)
            interactions.append(tmp)

        # create set of unique node connections (src, dest, corr)
        set_interractions = set(tuple(i) for i in interactions)

        # Sort the set
        set_interractions = sorted(set_interractions)

        # Make a dataframe
        df_interractions = pd.DataFrame.from_records(set_interractions)

        # Set columns names
        df_interractions.columns = ['src', 'dest', 'corr']

        # Creating a dictionary with the structure as below:
        #   dict (src, corr): [dest]
        self.dc_interact = collections.defaultdict(list)

        # Fill the dictionary with unique src - correlation id and a dest.
            list
        for index, row in df_interractions.iterrows():
            # Skip too small correlations (threshold vs absolute value)
            if math.fabs(float(row['corr'])) < threshold:
                continue

            # If the correlation is big enough, add it to the dictionary
            tmp_tuple = (row['src'], row['corr'])
            self.dc_interact[tmp_tuple].append(row['dest'])

    def to_sif(self, file_path):
        """
        Write the network into a simple interaction file (SIF).
        Column 0: label of the source node
        Column 1: interaction type
        Columns 2+: label of target node(s)
        :param file_path: path to the output file
        """
```

```
        f = open(file_path, "w")  # opens file
60
        for key, value in self.dc_interact.items():
            dests = ""
            for dest in self.dc_interact[key]:
                dests += "\t" + dest
65
            f.write(str(key[0]) + "\t" + str(key[1]) + dests + "\n")
        f.close()
```

(b) **Network visualisation**[2]

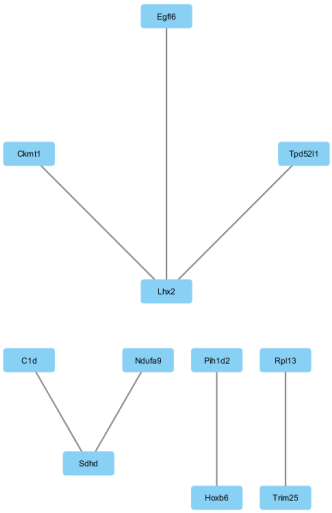

Figure 1: Expression network with Kendall correlation

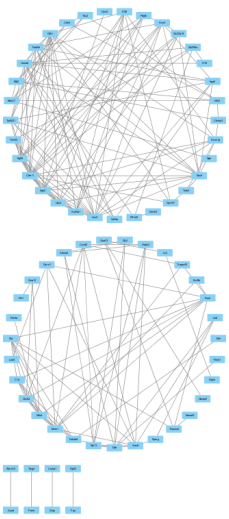

Figure 2: Expression network with Pearson correlation

[2]The files are attached with the source files in the email.

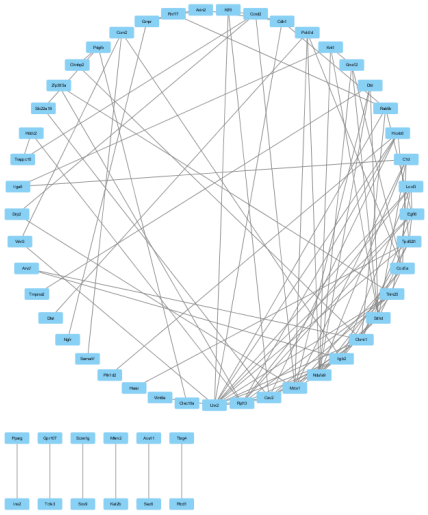

Figure 3: Expression network with Spearman correlation

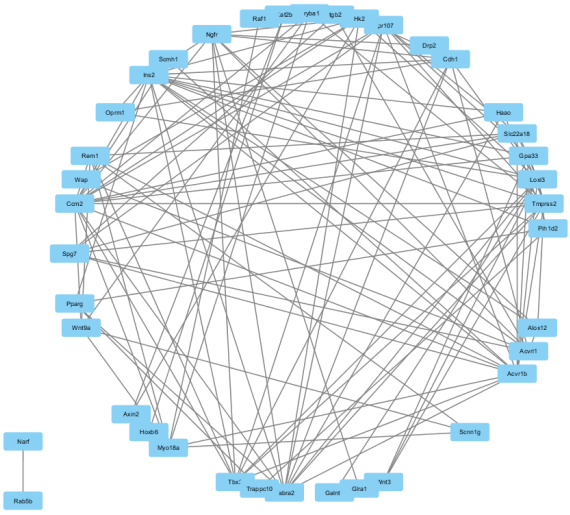

Figure 4: Methylation network with Kendall correlation

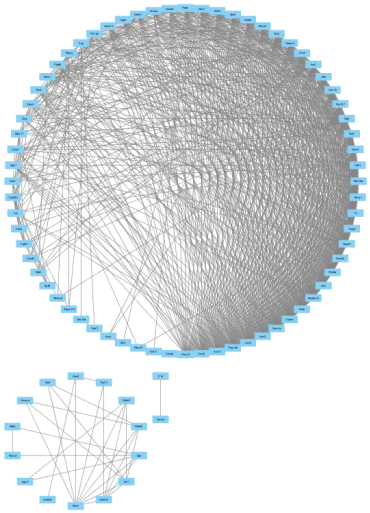

Figure 5: Methylation network with Pearson correlation

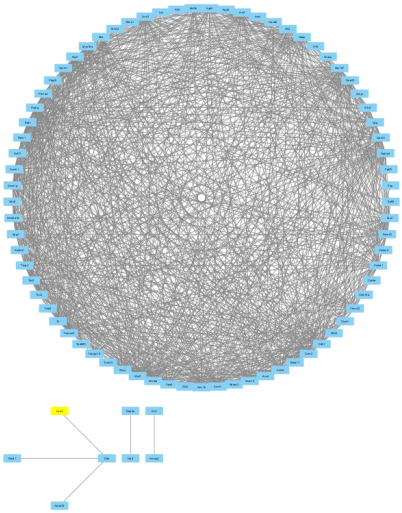

Figure 6: Methylation network with Spearman correlation

(c) **Discussion:** *Briefly comment on the similarities and difference between the networks. Explain and discuss your results.*
We observe that the number of correlated genes expression

Exercise 8.4: Blah

(a) **Blah**