

# 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 Data-Matrix-class in which you should complete.*

Listing 1: Data Matrix class script

```
0 import pandas as pd
  from scipy import stats

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

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

      def read_data(self):
          """
          15      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.
                  """

          20      # 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
          25      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"
          30      self.df.columns = new_columns

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

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

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

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

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

        return rows

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

        return cols

65    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
70        :return: dictionary with keys = row/columns names, values = Shapiro-
            Wilk p-value
        """

        ret = {}

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

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

            if pvalue < alpha:
85                ret[key] = pvalue

        return ret

    def to_tsv(self, file_path):
90        """
        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
        """
95        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. <sup>1</sup>

Listing 2: Main programm

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

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

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

    fo.close()

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

25    # 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")
30    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))

35    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))

40    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))

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

---

<sup>1</sup>The files are attached with the source files in the email.

```
50 def exercise_3():
    #

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

    NETWORK_THRESHOLD = 0.75

    # Expression
60    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)
65    cn = CorrelationNetwork(cm, NETWORK_THRESHOLD)
    cn.to_sif("./schmitt_schowing_expression_network_spearman.sif")

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

    # Methylation
    cm = CorrelationMatrix(data_methylation, "Pearson", True)
75    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")
80

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

85

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

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

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

100    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)
105    cc = CorrelationClustering(cm)
    cc.trace_to_tsv("schmitt_schowing_expression_cluster_spearman.tsv")

    # With the methylation data
    cm = CorrelationMatrix(data_methylation, "Kendall", False)
110    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

```
0 from itertools import combinations
  from scipy import stats

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

    10     return stats.rankdata(x)

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

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

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

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

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

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

    class CorrelationMatrix(dict):
      """
    45      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.
      """

    50     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
      """

    55     # 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

```
0 import collections
  import pandas as pd
  import math

  class CorrelationNetwork:
5      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)
10         :param threshold: a float between 0 and 1
          """

          interactions = []
          for tup, corr in correlation_matrix.items():
15              correlation = str(round(corr, 2))
              node0 = tup[0]
              node1 = tup[1]

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

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

          # Sort the set
          set_interactions = sorted(set_interactions)

30          # Make a dataframe
          df_interactions = pd.DataFrame.from_records(set_interactions)

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

40          # Fill the dictionary with unique src - correlation id and a dest.
              list
          for index, row in df_interactions.iterrows():
              # Skip too small correlations (threshold vs absolute value)
              if math.fabs(float(row['corr'])) < threshold:
                  continue
45              # 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'])

50      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<sup>2</sup>

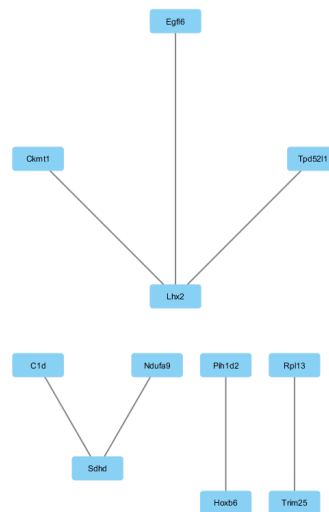


Figure 1: Expression network with Kendall correlation

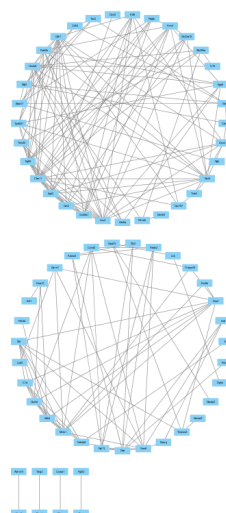


Figure 2: Expression network with Pearson correlation

<sup>2</sup>The files are attached with the source files in the email.

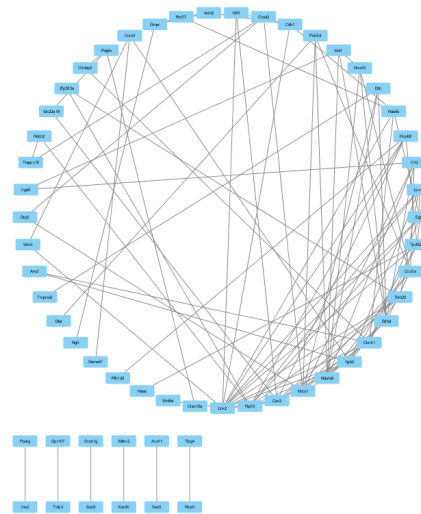


Figure 3: Expression network with Spearman correlation

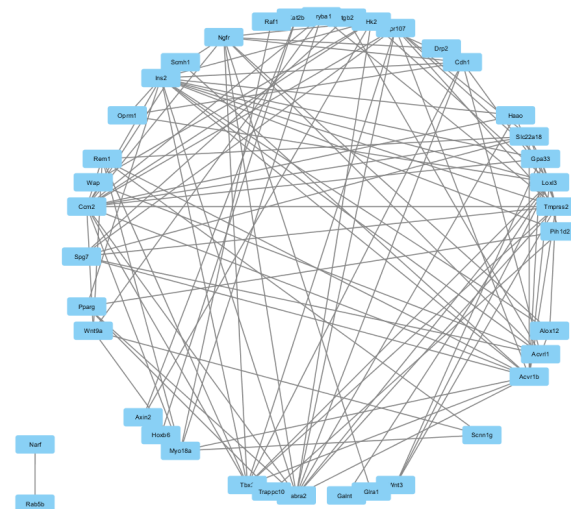


Figure 4: Methylation network with Kendall correlation

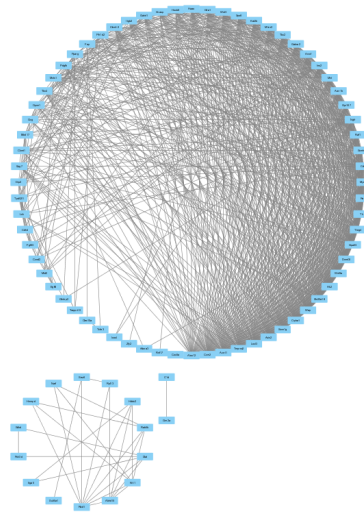


Figure 5: Methylation network with Pearson correlation

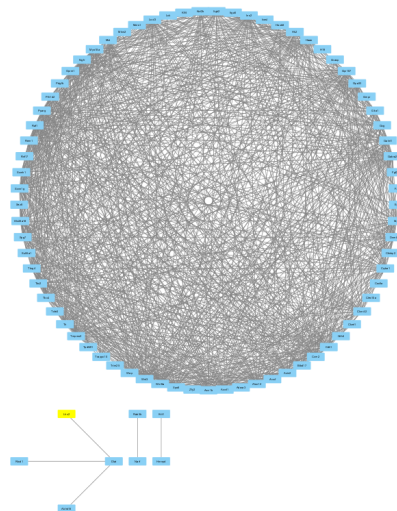


Figure 6: Methylation network with Spearman correlation

- (c) **Discussion:** *Briefly comment on the similarities and difference between the networks. Explain and discuss your results.*

We observe that the number of highly correlated gene is much higher when we look at the methylation in opposition as when we look at the expression.

## Exercise 8.4: Hierarchical Clustering

### (a) Implementation:

Listing 5: Hierarchical clustering

```
0 import itertools

class Cluster(frozenset):
    """
    This class behaves like a frozenset, meaning it only contains unique items
    like a set but you cannot add or remove
5 items, which makes it hashable and thus suitable for dictionary keys or as
    elements of a normal set.
    You can use the modified union method to merge two clusters as follows:
    merged_cluster = cluster_1.union(cluster_2)
    The to_string method was modified as well to help with the trace_to_tsv()
    method in exercise 8.4.
    """
10 def __str__(self):
    """
    :return: string with the sorted elements of the current cluster:
        element_1, element_2, ...
    """
    return ', '.join(sorted(self))

15 def union(self, iterable):
    """
    :param iterable: a Cluster, list, set, iterator, ...
    :return: a new Cluster containing all elements in the current cluster
        and the iterable
    """
20 return Cluster(list(self) + list(iterable))

class CorrelationClustering:
25 def __init__(self, correlation_matrix):
    """
    Initialises and executes hierarchical clustering based on a
        correlation matrix.
    :param correlation_matrix: a CorrelationMatrix (see correlation.py)

30 Structure of correlation matrix:
    {('Abhd15', 'Acvr1b'): -0.20814079896736404, ('Acvr1b', 'Abhd15'):
        -0.20814079896736404, ('Abhd15', 'Acvrl1'): -0.13245323570650439,
    """
    # distance metric
    self.d = correlation_matrix
35 # list of tuples: [(cluster 1 to merge, cluster 2 to merge, linkage
        value between the two clusters), ...]
    self.trace = []
    # cluster the elements in the correlation matrix
    self.cluster()

40 def cluster(self):
    """
    Hierarchically clusters the elements in the input correlation matrix
        and stores each step in the trace.
    """
    # Create a set of unique correlation (a, b, corr) but not (b, a, corr)
45 interactions = []
    for tup, corr in self.d.items():
        correlation = str(round(corr, 2))
        node0 = tup[0]
50 node1 = tup[1]

        tmp = [node0, node1, correlation]
```

```
        tmp.sort(reverse=True)
        interactions.append(tmp)
55
        # create set of unique node connections (corr, src, dest)
        set_interactions = set(tuple(i) for i in interactions)

        # set of nodes (experiments)
60
        set_experiment = set(i[0] for i in interactions)

        all_individual_clusters = []
        for element in set_experiment:
            tmp_cluster = Cluster([element])
65
            all_individual_clusters.append(tmp_cluster)

        # while we have more than one cluster

        while len(all_individual_clusters) > 1:
70
            # Compute linkage for all pair
            all_pairs = list(itertools.combinations(all_individual_clusters,
2))

            index_max_linkage = 0
            max_linkage = 0
75
            for i in range(len(all_pairs)):
                tmp_linkage = self.average_linkage(all_pairs[i][0], all_pairs[
                    i][1])
                if tmp_linkage > max_linkage:
                    max_linkage = tmp_linkage
80
                    index_max_linkage = i

            # Now we have the two clusters to merge: merge them and remove
            # them from the list
            # First add them to the trace
            #self.trace.append(str(all_pairs[index_max_linkage][0]) + " - " +
            #str(all_pairs[index_max_linkage][1]) + " - " + str(max_linkage)
            #)
85
            self.trace.append([all_pairs[index_max_linkage][0], all_pairs[
                index_max_linkage][1], max_linkage])
            new_cluster = all_pairs[index_max_linkage][0].union(all_pairs[
                index_max_linkage][1])

            # Remove the two clusters that are gonna be merged from the list
            all_individual_clusters.remove(all_pairs[index_max_linkage][0])
90
            all_individual_clusters.remove(all_pairs[index_max_linkage][1])

            # Append the new cluster resulting from the merging of the two old
            # ones blah
            all_individual_clusters.append(new_cluster)

95

def average_linkage(self, cluster_1, cluster_2):
    """
    :return: average linkage between cluster 1 and cluster 2
    """
100
    sum_ = 0
    for key1 in cluster_1:
        for key2 in cluster_2:
105
            sum_ += abs(self.d[(key1, key2)])

    return 1/(len(cluster_1) * len(cluster_2)) * sum_

def trace_to_tsv(self, file_path):
110
    """
```

```

    Writes the clustering trace into a tab-separated file. Each line
    represents a step in the clustering, in which
    two clusters are merged.
    Column 0: comma-separated names in cluster 1
    Column 1: comma-separated names in cluster 2
    Column 2: linkage value
115 :param file_path: path to the output file
    """

    f = open(file_path, "w") # opens file
120

    # At each step we have the two merged cluster and their linkage value
    # The linkage value is rounded to 4 digits to make it nice
    for step in self.trace:
        f.write(str(step[0]) + "\t" + str(step[1]) + "\t" + str(round(step
125 [2], 4)) + "\n")

    f.close()

```

- (b) **Application:** In the function `exercise 4()` in `main.py` (listing 2), use your implementation to hierarchically cluster the expression and methylation data tables with the Pearson, Spearman and Kendall correlation coefficient. This should give you a total of 6 TSV files
- (c) **Discussion:** Can hierarchical clustering be used to differentiate between blood cells and skin tissues? Are there differences between the correlation coefficients or data type? Why?

Let's first recall the two different types of cells. In this experiment we have the samples HSC, MPP1, MPP2, CLP, CMP, GMP, MEP, CD4, CD8, B cell, Eryth, Granu and Mono are from blood cells, whereas the samples TBSC, ABSC, MTAC, CLDC, EPro and EDif are from skin tissues. In the listing 6 at line 15, one of the last merge, we can see that the two merged cluster are from samples from different cells (blood or skin) meaning that the clustering is working well. We can assume here that the genes in a skin cell or in a blood cell are not expressed and/or methylated in the same way as the function of the cells are not the same.

Listing 6: Hierarchical clustering example

```

0 Granu    GMP      0.9988
  MPP2     CMP      0.9985
  CD4      CD8      0.9981
  CLP      CMP, MPP2 0.9972
  GMP, Granu    CLP, CMP, MPP2 0.9965
5 Mono     CLP, CMP, GMP, Granu, MPP2 0.9961
  MPP1     HSC      0.9956
  CLP, CMP, GMP, Granu, MPP2, Mono    HSC, MPP1    0.9919
  EPro     EDif     0.9918
  CD4, CD8    CLP, CMP, GMP, Granu, HSC, MPP1, MPP2, Mono 0.9901
10 MEP     CD4, CD8, CLP, CMP, GMP, Granu, HSC, MPP1, MPP2, Mono 0.9889
  MTAC     TBSC     0.9888
  B_cell   CD4, CD8, CLP, CMP, GMP, Granu, HSC, MEP, MPP1, MPP2, Mono 0.9871
  EDif, EPro    MTAC, TBSC    0.9787
  CLDC     EDif, EPro, MTAC, TBSC 0.9764
15 B_cell , CD4, CD8, CLP, CMP, GMP, Granu, HSC, MEP, MPP1, MPP2, Mono    CLDC,
  EDif, EPro, MTAC, TBSC    0.9164
  Eryth    B_cell , CD4, CD8, CLDC, CLP, CMP, EDif, EPro, GMP, Granu, HSC, MEP,
  MPP1, MPP2, MTAC, Mono, TBSC    0.8998

```