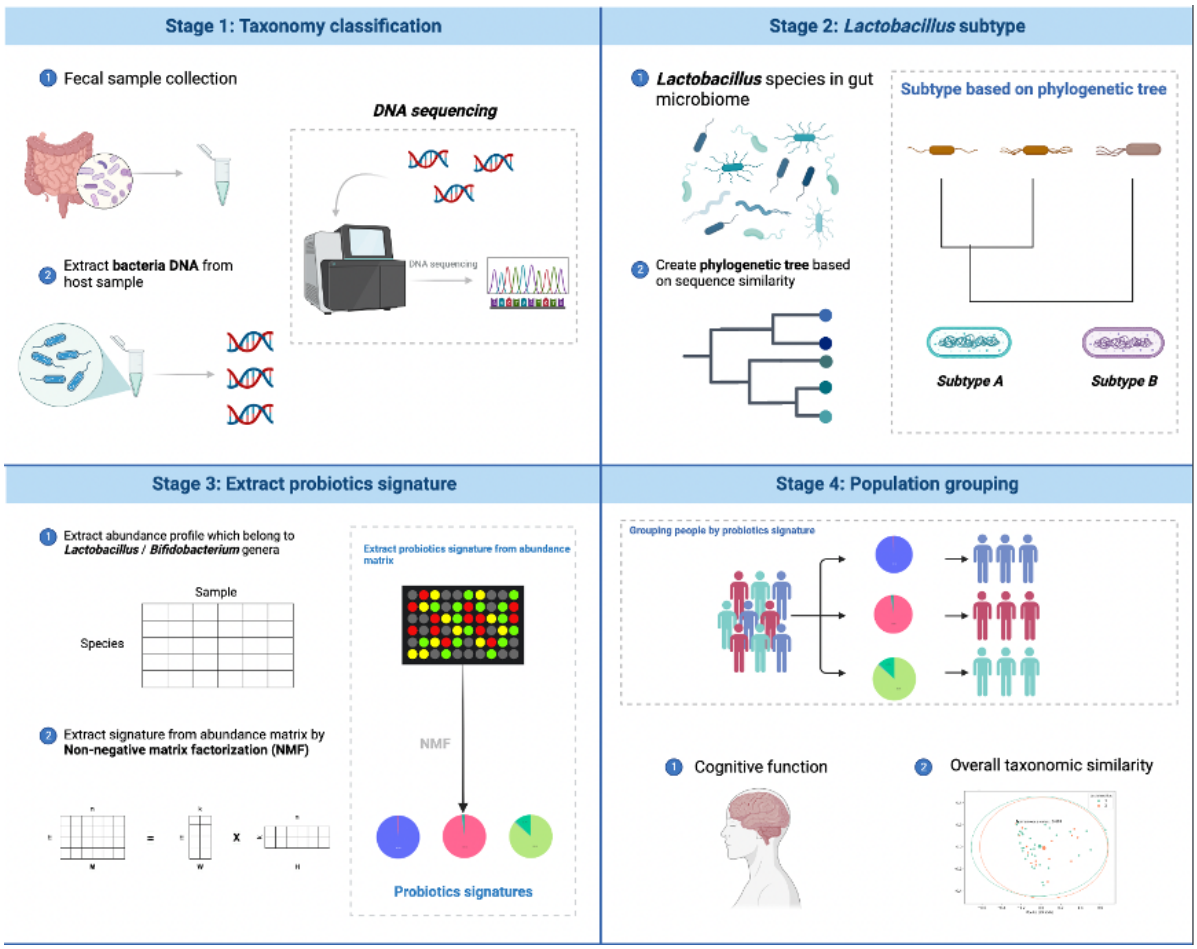
Probiotics signature decomposition tutorial

- Overview :



- Requirement :
  1. **NMF** : https://renozao.github.io/NMF/master/PAGE-INSTALLATION.html
  2. **ConsensusClusteringPlus** : https://bioconductor.org/packages/release/bioc/html/ConsensusClusterPlus.html
  3. **curatedMetagenomicData (optional):** https://bioconductor.org/packages/release/data/experiment/html/curatedMetagenomicData.html
  4. **Sklearn** : https://scikit-learn.org/stable/install.html
  5. **Nimfa** : https://github.com/mims-harvard/nimfa
  6. **Skbio** : https://scikit.bio/
  7. **Matplotlib, Seaborn, Plotly**

- Section :
  1. Create Weighted / unweighted Unifrac distance.
  2. Lactobacillus subtype.
  3. Evaluate the optimal k for NMF.
  4. Decompose the probiotic signature.
  5. Consensus clustering.
  6. Visualization.

---

1. Create weighted Unifrac distance :

- Requirement :
  1. `calculate_unifrac.R` : https://github.com/biobakery/MetaPhlAn/blob/4beta/metaphlan/utils/calculate_unifrac.R
  2. `The phylogenetic tree file (.nwk)` : https://github.com/biobakery/MetaPhlAn/blob/4beta/metaphlan/utils/mpa_v30_CHOCOPhlAn_201901_species_tree.nwk
  3. `metaphlan output file` : The merge abundance from metaphlan.
- Usage :

```
Rscript calculate_unifrac.R metaphlan_output.txt mpa_v30_CHOCOPhlAn_201901_species_tree.nw
```

---

2. Lactobacillus subtype :

- Objective : Concatenate the low prevalence Lactobacillus abundance into subtype to increase the abundance & prevalence of Lactobacillus.
- Reference : A taxonomic note on the genus Lactobacillus: Description of 23 novel genera, emended description of the genus Lactobacillus Beijerinck 1901, and union of Lactobacillaceae and Leuconostocaceae
- Requirement :
  1. Subtype reference table.
  2. Lactobacillus abundance matrix.

▼ Function :

```
def concat_species_into_subtype(self,input_matrix,reference_df,genus='Lactobacillus',speci
        """
         Merge abundance from rate species into subtype. (ex : Lactobacillus genus)

        Args:
            input_matrix (pd.DataFrame): Target abundance matrix. row is species, column i
            reference_df (pd.DataFrame): Table with subtype information. Please make sure
            species_colname (str, optional): Colname of species in reference_df. Defaults
            subtype_colname (str, optional): Colname of subtype in reference_df. Defaults

        Returns:
            subtype_matrix (pd.DataFrame): Abundance matrix of each subtype.
            subtype_dict (dict) : Dict of subtype and its components. Key is subtype, valu
        """
        speceis2subtype = dict(zip([x.replace(' ','_') for x in reference_df['species'].va
        subtype_dict = defaultdict(list)
```

```
        # subtype_dict format like : {'Levilactobacillus' : ['Lactobacillus_acetotolerans'
        for species in input_matrix.index :
            if species in speceis2subtype :
                subtype = speceis2subtype[species]
            else :
                subtype = 'no phylogroup'

            if subtype == 'no phylogroup' :
                subtype_dict[genus + '_others'].append(species)
            else :
                subtype_dict[subtype+'_subtype'].append(species)

        subtype_matrix = pd.DataFrame(np.zeros([len(subtype_dict),input_matrix.shape[1]]),
        index = list(subtype_dict.keys()),columns=input_matrix.columns)
        for subtype in subtype_dict :
            target_species = list(set(subtype_dict[subtype]).intersection(input_matrix.ind
            subtype_matrix.loc[subtype,:] = input_matrix.loc[target_species,:].sum()

        return subtype_matrix,subtype_dict
```

- Usage :

```
ps = probiotic_signature(metadata,abundance_matrix,distance_matrix)
subtype_matrix,subtype_dict = ps.concat_species_into_subtype(input_matrix,reference_df)
```

---

3. Evaluate the optimal k for NMF :

- Objective : Select optimal number of component for NMF decomposition.
- Requirement :
  1. Abundance matrix.
  2. `nimfa` & `sklearn` package

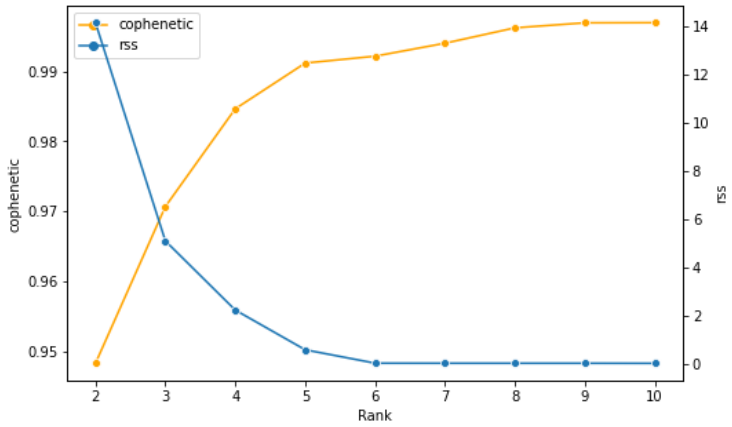
▼ Function :

```
def evaluate_nmf_component(self,input_matrix,min_k=2,max_k=10) :
        """
        Evaluate optimal k component for NMF decomposition processing.

        Args:
            input_matrix (numpy.ndarray) : The original matrix for NNF (V).
            k_min (int, optional): The minimum component number. Defaults to 2.
            k_max (int, optional): The maximum component number. Defaults to 10.
        """
        nmf = nimfa.Nmf(input_matrix,rank=max_k, max_iter=200)
        nmf_fit = nmf()
        #evaluation
        rank_list = list(range(min_k,max_k+1))
        evalation = nmf.estimate_rank(rank_range=[x for x in rank_list],n_run=100)
        #output the estimation result
        measurements = ['rss','evar','dispersion','cophenetic','kl']
        measurement_table = pd.DataFrame({'Rank' : rank_list})
```

```
        for m in measurements :
            measurement_table[m] = [evalation[x][m] for x in rank_list]
        return measurement_table

def plot_nmf_rank(self,measurement_table,
                  measurement_1 = 'cophenetic',
                  measurement_2 = 'rss',
                  fig_output_path='nmf_evaluation.pdf',
                  fig_format='pdf') :
        """
        Visualation of NMF rank evaluation.

        Args:
            measurement_table (pd.DataFrame): The measurement result, including Rank, rss,
            measurement_1 (str, optional): The first measurement of NMF rank. Defaults to
            measurement_2 (str, optional): The second measurement of NMF rank. Defaults to
            fig_output_path (str, optional): The figure output path. Defaults to 'nmf_eval
            fig_format (str, optional): The figure output format. Defaults to 'pdf'.
        """
        if measurement_1 not in measurement_table.columns :
            print("Please confirm %s in measurement table !" % measurement_1)
            return
        if measurement_2 not in measurement_table.columns :
            print("Please confirm %s in measurement table !" % measurement_2)
            return

        plt.figure(figsize=(8,5))
        g = sns.lineplot(data=measurement_table,x ='Rank',y=measurement_1, color="orange",
        sns.lineplot(data=measurement_table,x = 'Rank',y=measurement_2, ax=g.axes.twinx(),
        g.legend(handles=[Line2D([], [], marker='o', color="orange", label=measurement_1),
                          Line2D([], [], marker='o', label=measurement_2)])
        plt.savefig(fig_output_path,dpi=300,format=fig_format)
```

- Usage :

```
measure_df = ps.evaluate_nmf_component(subtype_matrix.to_numpy(),k_min=2,k_max=10)
ps.plot_nmf_rank(measure_df)
```



- The measure_df including following measurements for different rank / k :
    1. `rss` :
    2. `evar` :
    3. `dispersion` :
    4. `cophenetic` :
    5. `kl` :

3.1 Evaluate the optimal k for NMF (R version):

- Requirement :
    1. Abundance matrix.
    2. `NMF` package.
- Usage :

```
library(NMF)
library(ggplot2)

lacto_df = read.table('/home/bruce1996/repo/Microbiome_health_indicator/tutorial/data/lact
bifido_df = read.table('/home/bruce1996/repo/Microbiome_health_indicator/tutorial/data/bif

ranks <- 2:7
bifido_mat <- as.matrix(bifido_df)
i0 <- which(colSums(bifido_mat) == 0)
i_na <- which(colSums(is.na(bifido_mat)) > 0)
nmf_input = bifido_mat[, -c(col_0, col_na)] + 10 ** -8
bifido_estim.coad <- nmf(nmf_input,ranks,nrun = 5,.opt='v')
bifido_p = plot(bifido_estim.coad) + ggtitle('Clustering evaluation of Bifidobacterium')
ggsave('/home/bruce1996/repo/Microbiome_health_indicator/tutorial/nmf_evaluation/bifido_nm

lacto_mat <- as.matrix(lacto_df)
col_0 <- which(colSums(lacto_mat) == 0)
col_na <- which(colSums(is.na(lacto_mat)) > 0)
nmf_input = lacto_mat[, -c(col_0, col_na)] + 10 ** -8
lacto_estim.coad <- nmf(nmf_input,ranks,'lee',nrun = 5,.opt='v')
lacto_p = plot(lacto_estim.coad) + ggtitle('Clustering evaluation of Lactobacillus subtype
ggsave('/home/bruce1996/repo/Microbiome_health_indicator/tutorial/nmf_evaluation/lacto_nmf
```

---

4. Decompose the probiotic signature

- Objective : Decompose the matrix to k components.
- Requirement :
    1. Abundance matrix.
    2. `nimfa` & `sklearn` package
    3. Optimal k number.

▼ Function :

```
def finger_print_proportion(self,x,w,h):
        """
        Calculate the contribution of each NMF decompose component.
        Args:
            x (np.array): The original matrix.
            w (np.array): The weight matrix of NMF decomposition.
            h (np.array): The coefficient matrix of NMF decomposition

        Returns:
            proportion_matrix (np.array): The contribution of each component.
        """
        n_finger_print = w.shape[1]
        n_sample = w.shape[0]
        proportion_matrix = np.zeros([n_sample,n_finger_print])

        for i in range(n_sample) :
            total = sum(x[i,:])
            if total == 0 :
                continue
            else :
                for j in range(n_finger_print) :
                    ab = sum(np.dot(w[i,j],h[j,:]))# type: ignore
                    proportion_matrix[i,j] = ab / total
        return proportion_matrix

def sklearn_decompose_probiotics_signature(self,input_matrix,k,prefix) :
        """
        Decompose k signatures from input abundance matrix.

        Args:
            input_matrix (pd.DataFrame) : The origin matrix (n_species * n_sample) to be d
            k (int): Number of component expected to be decomposed.
        """
        X = input_matrix.T.to_numpy() # n_sample * n_species
        #sklearn.decomposition version
        nmf_model = NMF(n_components=k, init='random', random_state=0,max_iter=1000)
        nmf_model.fit(X)
        W = nmf_model.transform(X) # n_sample * n_component
        H = nmf_model.components_ # n_component * n_species

        finger_print_matrix = self.finger_print_proportion(X,W,H) # n_sample * n_component
        index = [prefix + ' signature' + str(x) for x in range(1,k+1)]
        # format signature coefficient matrix (n_component * n_species)
        sig_coefficient = pd.DataFrame(H.T,index=input_matrix.index,columns=index)
        # format signature weight matrix (n_component * n_sample)
        finger_print_df = pd.DataFrame(finger_print_matrix.T,index=index,columns=input_mat
        finger_print_df[finger_print_df > 1] = 1
        finger_print_df[finger_print_df < 0] = 0

        return finger_print_df,sig_coefficient
```

```
def nimfa_decompose_probiotics_signature(self,input_matrix,k,prefix) :
        """
        Decompose k signatures from input abundance matrix.

        Args:
            input_matrix (pd.DataFrame) : The origin matrix (n_species * n_sample) to be d
            k (int): Number of component expected to be decomposed.
        """
        X = input_matrix.T.to_numpy() # n_sample * n_species
        #Nimfa version
        nmf_model = nimfa.Nmf(X,rank=k, max_iter=1000)
        nmf_fit = nmf_model()
        W = np.array(nmf_fit.basis()) # n_sample * n_component
        H = np.array(nmf_fit.coef()) # n_component * n_species

        finger_print_matrix = self.finger_print_proportion(X,W,H) # n_sample * n_component
        index = [prefix + ' signature' + str(x) for x in range(1,k+1)]
        # format signature coefficient matrix (n_component * n_species)
        sig_coefficient = pd.DataFrame(H.T,index=input_matrix.index,columns=index)
        # format signature weight matrix (n_component * n_sample)
        finger_print_df = pd.DataFrame(finger_print_matrix.T,index=index,columns=input_mat
        finger_print_df[finger_print_df > 1] = 1
        finger_print_df[finger_print_df < 0] = 0

        return finger_print_df,sig_coefficient
```

- Usage :

```
lacto_proportion_matrix, lacto_coef_matrix = ps.nimfa_decompose_probiotics_signature(subty
bifido_proportion_matrix, bifido_coef_matrix = ps.nimfa_decompose_probiotics_signature(bif
```

---

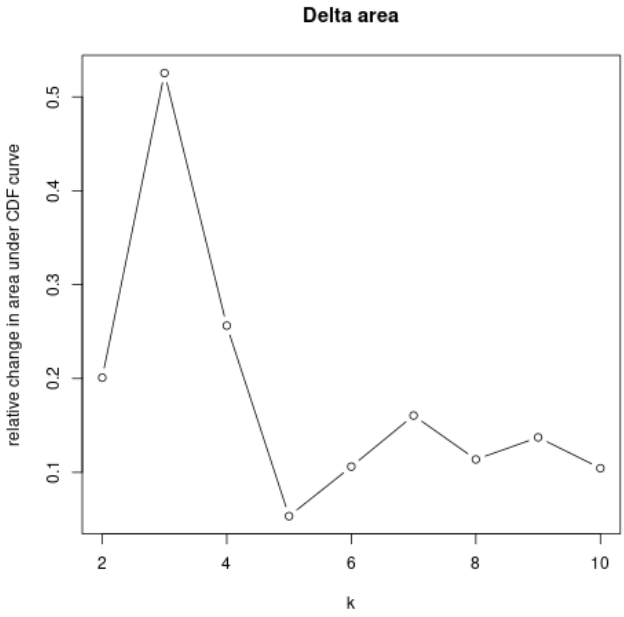
5. Consensus clustering :

- Objective : Separate the sample / patient into different cluster.
- Requirement :
  1. Signature proportion matrix.
  2. `ConsensusClusterPlus` package.
- Usage :

```
consensus_p = "/home/bruce1996/repo/Microbiome_health_indicator/tutorial/consensus_evaluat
output_p = "/home/bruce1996/repo/Microbiome_health_indicator/tutorial/consensus_clustering

exp_m = read.table("/home/bruce1996/repo/Microbiome_health_indicator/tutorial/sig_matrix/s
                   header = T,row.names = 1,sep = '\t',encoding = "UTF-8")
exp_m = as.matrix(exp_m)
res = ConsensusClusterPlus(exp_m,maxK=10,reps=50,pItem=0.8,pFeature=1,
                           clusterAlg="hc",title=consensus_p,
                           distance = 'euclidean',
                           seed=1262118388.71279,
                           plot="png")
```

```
#output the result of consensus clustering
n_cluster = 4
df = as.data.frame(res[[n_cluster]]$consensusClass)
colnames(df) = c('cluster')
cm = consensus_matrix(res,n_cluster)
write.table(cm,file =output_p,sep = '\t',quote = F)
```

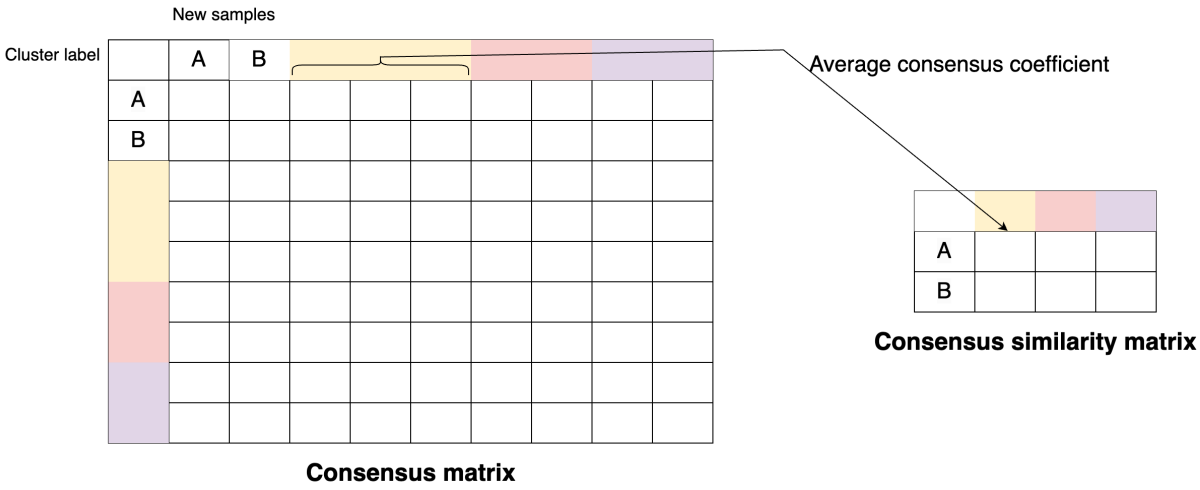
- Evaluation :



---

6. Determine new sample cluster based on previous clustering result

- Objective : Determine the proper cluster for new participant / samples which not involve in previous consensus clustering.
- Requirement :
  1. The matrix for consensus clustering. (`signature proportion matrix`)
  2. Previous clustering result. (New sample is assigned to `None`)
  3. `ConsensusClusterPlus` package.
- illustration :



▼ Function :

```
get_consensus_matrix <- function(consensus_clustering_result,consensus_number){
  # Extract the consensus matrix from ConsensusClusteringPlus output.
  #
  # Args:
  #   consensus_clustering_result (list): The ConsensusClusteringPlus output.
  # consensus_number (int): The number of cluster.
  m = consensus_clustering_result[[consensus_number]]$consensusMatrix
  samples = names(consensus_clustering_result[[consensus_number]]$consensusClass)
  df = as.matrix(m)
  rownames(df) = samples
  colnames(df) = samples
  return(df)
}

determine_new_sample_cluster <- function(exp_m, metadata, cluster_colnames='cluster', new_
  # Evaluation new sample cluster based on previous clustering result.
  #
  # Args:
  #   exp_m (matrix): The origin matrix for ConsensusClusteringPlus.
  # metadata (data.frame): The data.frame including the patient metadata.
  # cluster_colnames (str, optional): The colname of clustering result. Defaults to 'clust
  # new_sample_label (str, optional): The element in clustering result column for new / no

  cluster_label <- unique(metadata[[cluster_colnames]])  # Unique cluster labels are clust
  cluster_label <- setdiff(cluster_label, new_sample_label)
  consensus_res = ConsensusClusterPlus(exp_m,maxK=10,reps=50,pItem=0.8,pFeature=1,
                             clusterAlg="hc",
                             distance = 'euclidean',
                             seed=1262118388.71279,
                             plot="png")
  consensus_matrix <- get_consensus_matrix(consensus_res,length(cluster_label))
  #subset the sample list
  new_samples = rownames(metadata)[metadata$cluster == new_sample_label]
  #Create a blank consensus similarity matrix
  cluster_consensus_sim <- matrix(0, nrow = length(new_samples), ncol = length(cluster_lab
```

```
  rownames(cluster_consensus_sim) <- new_samples
  colnames(cluster_consensus_sim) <- cluster_label

  for (sample in new_samples) {
    for (cluster in cluster_label) {
      cluster_sample_list <- rownames(metadata)[metadata[[cluster_colnames]] == cluster]
      # Calculate the average consensus for new sample to cluster_sample_list (samples in
      cluster_consensus_sim[sample, cluster] <- mean(consensus_matrix[sample, cluster_samp
    }
  }
  return(cluster_consensus_sim)
}
```

- Usage :

```
#new
library(readxl)
library(ConsensusClusterPlus)

#example
repo_dir = "/home/bruce1996/repo/Microbiome_health_indicator/"
exp_m = read.table(paste0(repo_dir,"tutorial/sig_matrix/sig_proprotion_matrix.txt"),
                   header = T,row.names = 1,sep = '\t',encoding = "UTF-8")
tmp = read_excel(paste0(repo_dir,"tutorial/data/TPMIC_Diagnosis_297_KCF_0704.xlsx"))
metadata = as.data.frame(tmp)
rownames(metadata) = metadata$ID
metadata = metadata[colnames(exp_m),]
exp_m = as.matrix(exp_m)
##########
metadata$cluster = sample(c('a','b','c','d'),dim(metadata)[1],replace=TRUE)
cluster_colnames = 'cluster'
new_sample_label = 'd'
res = determine_new_sample_cluster(exp_m = exp_m ,metadata = metadata,new_sample_label = '
```