

First, load the data, in this notebook we are using the PMI data from the paper: "Environmental predictors impact microbial-based postmortem interval (PMI) estimation models within human decomposition soils". The preprocessed data includes OTU/phylum/class/order abundance matrices (includes or not include environmental factors)

```
In [1]: import sys
sys.path.append('../Code')
import loadData
import RunML
import RunML_continue
import FS
import metric

import pandas as pd
import numpy as np
import random
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
import pickle
import matplotlib.pyplot as plt
```

```
In [2]: import glob
import os
```

```
In [3]: PMIdata_path = '../Data/PMI/'
```

```
In [ ]:
```

## No env model

16s (OTU/phylum/class/order) - no env

ITS (OTU/phylum/class/order) - no env

16s+ITS (OTU/phylum/class/order) - no env (only use this data when calculating H)  
bact.ITS.n.class.env

## Data preprocess

```
In [4]: bact_ITS_noenv_files = glob.glob(PMIdata_path + 'bact.ITS.n.*.noenv.csv')
```

```
In [5]: bact_ITS_noenv_files
```

```
Out[5]: ['../Data/PMI/bact.ITS.n.otu.noenv.csv',
        '../Data/PMI/bact.ITS.n.class.noenv.csv',
        '../Data/PMI/bact.ITS.n.order.noenv.csv',
        '../Data/PMI/bact.ITS.n.phylum.noenv.csv']
```

```
In [6]: # Read each CSV file into a list of dataframes
        bact_ITS_noenv_df_list = [pd.read_csv(file) for file in bact_ITS_noenv_files]
```

```
In [ ]:
```

```
In [7]: for df in bact_ITS_noenv_df_list:
        print(df.shape)
```

```
(78, 7415)
```

```
(78, 178)
```

```
(78, 412)
```

```
(78, 52)
```

```
In [8]: bact_ITS_noenv_df_list[3]
```

```
Out[8]:
```

	ADH_10_actual	Proteobacteria	Verrucomicrobia	Acidobacteria	Actinobacteria
0	0.00000	2405.899397	957.553453	1988.986693	2524.409168
1	1484.14700	3209.276660	757.215776	1727.647573	1840.003679
2	1989.56009	3829.070365	412.942989	821.982537	1300.667694
3	2973.68000	3194.403374	419.280034	848.987596	1688.948376
4	4027.69203	3694.823154	475.772750	833.851140	1209.230357
...	...	...	...	...	...
73	4447.32241	2456.386357	702.629879	1096.355563	2612.831828
74	0.00000	2919.908680	1004.519405	1906.070913	954.200252
75	1477.74800	4411.744255	735.145847	1127.142509	1262.559538
76	2554.43400	2821.384741	962.492330	1377.965842	1773.368787
77	4536.41300	2756.755257	598.263330	755.701049	2031.293347

78 rows × 52 columns

```
In [9]: data_4taxa = []
        col_names_4taxa = []
        for df in bact_ITS_noenv_df_list:
```

```
data = df.drop(df.columns[0], axis=1)
cols_name = data.columns.tolist()
data = data.values
data_4taxa.append(data)
col_names_4taxa.append(cols_name)
```

In [ ]:

```
In [10]: # target variable
y = bact_ITS_noenv_df_list[3].iloc[:, 0].values
y
```

```
Out[10]: array([[ 0.      , 1484.147 , 1989.56009, 2973.68   , 4027.69203,
  0.      , 1026.394 , 1423.473 , 214.938   , 493.696   ,
  0.      , 1543.201 , 2970.8    , 3542.968 , 0.      ,
1445.51502, 2962.59226, 3504.55394, 4583.2796 , 0.      ,
2639.949 , 3573.444 , 0.      , 1507.78   , 3103.803 ,
4567.716 , 0.      , 1510.142 , 2847.791 , 4578.995 ,
  0.      , 1589.377 , 3031.373 , 4512.836 , 0.      ,
1502.226 , 3019.923 , 4574.528 , 0.      , 1364.604 ,
3009.054 , 3407.734 , 0.      , 1530.639 , 2949.689 ,
4391.578 , 0.      , 1588.226 , 3011.635 , 3976.49   ,
  0.      , 1445.393 , 2999.732 , 4527.275 , 0.      ,
1532.17172, 3552.96112, 4474.30808, 0.      , 1329.627 ,
2854.888 , 4326.392 , 0.      , 1600.15   , 3024.861 ,
3985.61   , 0.      , 1087.43   , 1533.552 , 510.076   ,
  0.      , 1500.02566, 2988.04838, 4447.32241, 0.      ,
1477.748 , 2554.434 , 4536.413  ])
```

```
In [11]: # Define the threshold
y_threshold = 2500

# Categorize the series based on the threshold
y = np.where(y > y_threshold, 'LONG', 'SHORT')

print(y)
```

```
['SHORT' 'SHORT' 'SHORT' 'LONG' 'LONG' 'SHORT' 'SHORT' 'SHORT' 'SHORT'
'SHORT' 'SHORT' 'SHORT' 'LONG' 'LONG' 'SHORT' 'SHORT' 'LONG' 'LONG'
'LONG' 'SHORT' 'LONG' 'LONG' 'SHORT' 'SHORT' 'LONG' 'LONG' 'SHORT'
'SHORT' 'LONG' 'LONG' 'SHORT' 'SHORT' 'LONG' 'LONG' 'SHORT' 'SHORT'
'LONG' 'LONG' 'SHORT' 'SHORT' 'LONG' 'LONG' 'SHORT' 'SHORT' 'LONG' 'LONG'
'SHORT' 'SHORT' 'LONG' 'LONG' 'SHORT' 'SHORT' 'LONG' 'LONG' 'SHORT'
'SHORT' 'LONG' 'LONG' 'SHORT' 'SHORT' 'LONG' 'LONG' 'SHORT' 'SHORT'
'LONG' 'LONG' 'SHORT' 'SHORT' 'SHORT' 'SHORT' 'SHORT' 'SHORT' 'LONG'
'LONG' 'SHORT' 'SHORT' 'LONG' 'LONG']
```

```
In [12]: list(y).count('LONG')
```

```
Out[12]: 35
```

```
In [13]: list(y).count('SHORT')
```

```
Out[13]: 43
```

## OTU

```
In [ ]:
```

### 1. calculate H statistics for OTU/phylum/class/order (both 16s and ITS)

```
In [ ]:
```

```
In [14]: weights_4taxa = []
```

```
In [15]: for df in data_4taxa:
          data=FS.relative_abundance(df)
          print(np.shape(data))
          weights=FS.OTU_H_Score_fun(data,y)
          weights_4taxa.append(weights)
```

```
(78, 7414)
```

```
(78, 177)
```

```
(78, 411)
```

```
(78, 51)
```

```
In [16]: for weight in weights_4taxa:
          print(len(weight))
```

```
7414
```

```
177
```

```
411
```

```
51
```

```
In [17]: max(weights_4taxa[3])
```

```
Out[17]: 34.26918116083644
```

```
In [ ]:
```

```
In [18]: selected0TU_index_4tax = []
          eps_4tax = []
```

```
In [19]: for weight in weights_4taxa:
          selected0TU_index, eps=FS.indice_H_unisig(weight,y)
```

```
print(eps)
selectedOTU_index_4tax.append(selectedOTU_index)
eps_4tax.append(eps)
```

23  
26  
34  
13

2. Select indices of the features based on H statistics and form the subset based on the selected features.

The default p value of the function is 10%, the resulted index is ranked by its H statistics descendingly.

Use "indice\_H\_unisig" if there is only one response, use "indice\_H\_multisig" for multiple responses.

weights\_4taxa,selectedOTU\_index\_4tax,col\_names\_4taxa,eps\_4tax

```
In [20]: weights_sig_sorted_4taxa = []
col_names_sig_sorted_4taxa = []
for i in range(len(weights_4taxa)):
    weights_sig_sorted = weights_4taxa[i][selectedOTU_index_4tax[i]]
    col_names_sig_sorted = [col_names_4taxa[i][j] for j in selectedOTU_index_4tax[i]]
    weights_sig_sorted_4taxa.append(weights_sig_sorted)
    col_names_sig_sorted_4taxa.append(col_names_sig_sorted)
```

In [ ]:

```
In [21]: taxlabels = ['OTU', 'class', 'order', 'phylum']

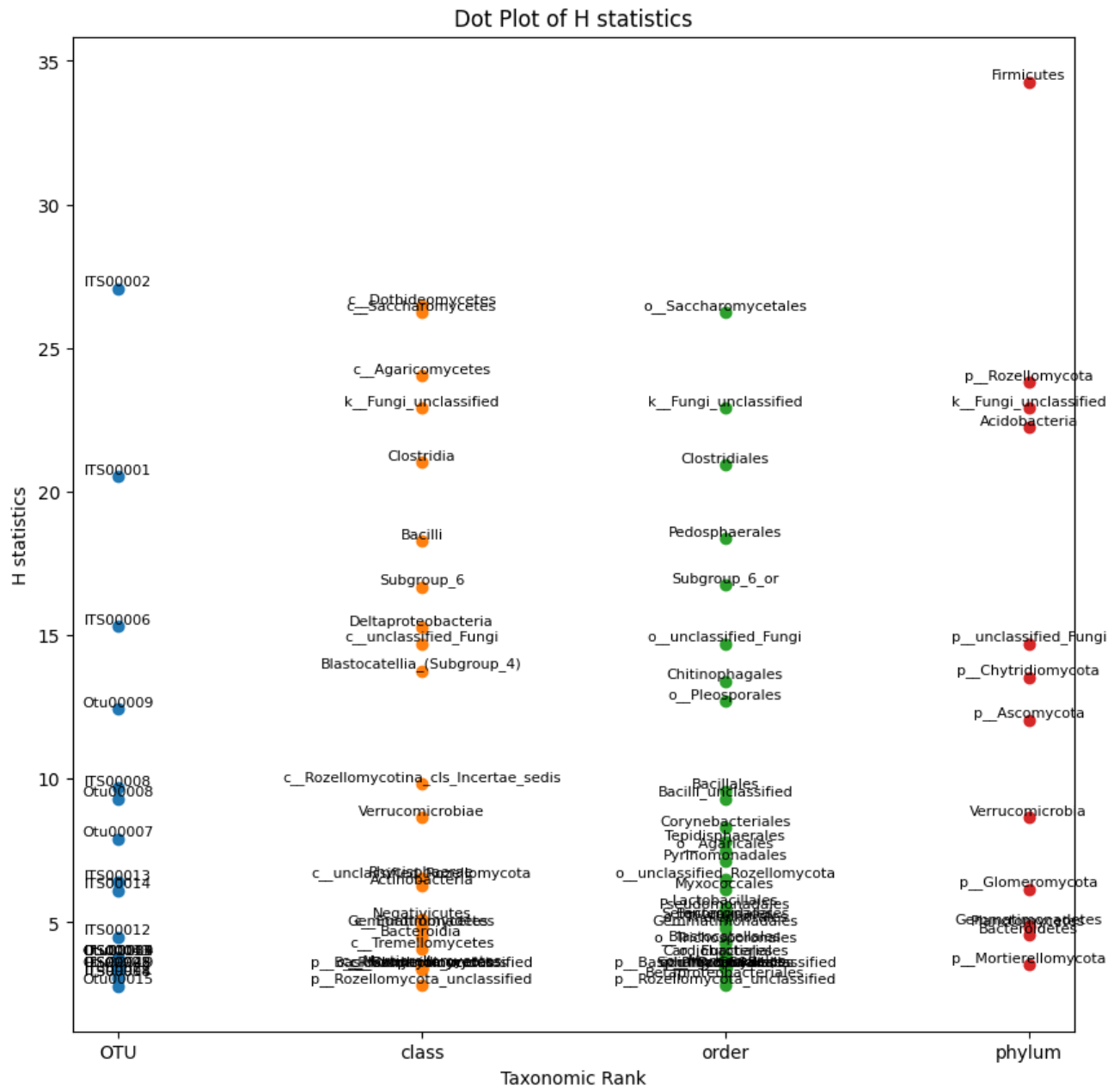
# Assuming weights_sig_sorted_4taxa contains numeric arrays
# Ensure col_names_sig_sorted_4taxa contains the corresponding string labels

plt.figure(figsize=(10, 10))
for i, array in enumerate(weights_sig_sorted_4taxa):
    x_values = [taxlabels[i]] * len(array) # Label each point with its group
    plt.scatter(x_values, array, label=f'{taxlabels[i]}')

    # Annotate each point with its name from col_names_sig_sorted_4taxa[i][j]
    for j, z in enumerate(array):
        label = col_names_sig_sorted_4taxa[i][j] # Get the corresponding label
        plt.text(taxlabels[i], z, label, ha='center', va='bottom', fontsize=10)

plt.title('Dot Plot of H statistics')
plt.xlabel('Taxonomic Rank')
plt.ylabel('H statistics')
```

```
plt.show()
```



```
In [ ]:
```

```
In [22]: #plot the h statistics and cutoff descendingly
#for i in range(len(weights_4taxa)):
#FS.plotWeightedIndex(weights_4taxa[i],threshold=eps_4tax[i])
```

```
In [23]: data_4taxa[0]
```

```
Out[23]: array([[0.00000000e+00, 0.00000000e+00, 0.00000000e+00, ...,
                0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
               [2.49850550e+02, 1.24158862e+01, 1.53282546e-01, ...,
                0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
               [2.28043143e+01, 1.23266564e+00, 0.00000000e+00, ...,
                0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
               ...,
               [2.93780204e+01, 1.49358551e+03, 1.56450996e+00, ...,
                0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
               [1.76288607e+02, 1.57368583e+02, 1.15054203e+00, ...,
                0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
               [6.92171115e+01, 3.91444266e+02, 5.54846585e-01, ...,
                0.00000000e+00, 0.00000000e+00, 0.00000000e+00]])
```

## 4. Model

Prepare 4 datasets: full dataset, our selected dataset, Lasso selected dataset(based on the target variable), randomly selected data (selected the same number of variables as in our method)

Use random forest and SVM as classifier, and will build both models for each response variable.

For Lasso, the dataset will be determined by the response variable, so the lasso subset is different for the models for different response variables.

For random selection, the process will repeat iter=30 times to find the mean accuracy and AUC

SMOTE is used (the data is not balanced, as we can see the performance is really bad especially for SVM model when not using SMOTE)

```
In [24]: iter =30
        cls = ["RF", "SVM"]
```

```
In [25]: targetLabel=y
```

```
In [26]: data_subset_4taxa = []
        X_lasso_4taxa = []
        xind_lasso_4taxa = []
        for i, data in enumerate(data_4taxa):
            X_lasso, xind_lasso = RunML_continue.LassoFeatureSelection(data, targetLabel)
            X_lasso_4taxa.append(X_lasso)
            xind_lasso_4taxa.append(xind_lasso)
            data_subset = {"AllFeatures": data,
                          "SelectMicro": data[:, selectedOTU_index_4tax[i]],
```

```

        "Lasso":X_lasso,
        "Random":data
    }
    data_subset_4taxa.append(data_subset)

```

```

In [27]: for dataset in data_subset_4taxa:
        data_subset = dataset
        for datatype, subset in data_subset.items():
            print(np.shape(subset))

```

```

(78, 7414)
(78, 23)
(78, 40)
(78, 7414)
(78, 177)
(78, 26)
(78, 24)
(78, 177)
(78, 411)
(78, 34)
(78, 22)
(78, 411)
(78, 51)
(78, 13)
(78, 10)
(78, 51)

```

```

In [28]: with open('../Data/PMI/data_subset_4taxa_noenv_label.pkl', 'wb') as file:
        pickle.dump(data_subset_4taxa, file)

```

The function will print out the accuracy and AUC for each dataset using each classifier, and also will return the y\_actual, y\_predict, y\_predprob for future use.

```

In [29]: #dict_cm = RunML_continue.runClassifier_FScompare(data_subsets= data_subset,

```

```

In [30]: xind_lasso_4taxa

```

```

Out[30]: [array([ 8, 35, 153, 703, 719, 941, 1040, 1134, 1237, 1242, 1469,
                1523, 1901, 2271, 2476, 3116, 3297, 3394, 3670, 3714, 4067, 4195,
                4240, 4288, 4541, 4830, 4846, 4905, 5024, 5034, 5049, 5196, 5200,
                5374, 5738, 6004, 6207, 7041, 7108, 7211]),
          array([ 1, 7, 12, 22, 35, 46, 48, 58, 97, 111, 116, 123, 125,
                126, 129, 132, 136, 137, 139, 140, 144, 146, 163, 164]),
          array([ 2, 6, 7, 19, 81, 111, 224, 259, 264, 270, 290, 293, 294,
                300, 312, 313, 332, 335, 337, 348, 355, 378]),
          array([ 2, 4, 9, 10, 15, 37, 39, 40, 44, 49])]

```

```

In [ ]:

```



```

In [31]: def plotPresenseRatio(X, label, featurenames, posLabel, posText="", negText="", th
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 300

presenceCntPos = []
presenceCntNeg = []

X_relative = FS.relative_abundance(X)

X_relative = X_relative.T
if abundanceCutoff==0:
    flatten_list = list(chain.from_iterable(X_relative))
    flatten_list_sorted=sorted(flatten_list)
    abundanceCutoff=flatten_list[int(len(flatten_list_sorted)*float(thre

if posText=="" or negText=="":
    posText=posLabel
    negText="Not "+posLabel

for k in range(len(X_relative)):# for each OTU
    OTUs = X_relative[k]## the samples for this OTU
    pos = 0
    neg = 0
    for i in range(len(OTUs)):
        if label[i] == posLabel:
            if OTUs[i] > abundanceCutoff:# if the value of OTU exceed th
                pos += 1
        else:
            if OTUs[i] > abundanceCutoff:
                neg += 1
    presenceCntPos.append(pos)# len= # of samples; each value is the num
    presenceCntNeg.append(neg)

all_pos_label_cnt=list(label).count(posLabel)
all_neg_label_cnt=len(label)-all_pos_label_cnt
print(all_pos_label_cnt,all_neg_label_cnt)# these 3 lines can use valu

presenceRatioPos=[float(x)/all_pos_label_cnt for x in presenceCntPos]# e
presenceRatioNeg=[float(x)/all_neg_label_cnt for x in presenceCntNeg]

import matplotlib.pyplot as plt
y = range(entries)
fig, axes = plt.subplots(ncols=2, sharey=True)
axes[0].barh(y, presenceRatioPos, align='center', color='#ff7f00')
axes[1].barh(y, presenceRatioNeg, align='center', color='#377eb8')
axes[0].set_xlabel("Presence Ratio in "+posText)
axes[1].set_xlabel("Presences Ratio "+negText)

axes[0].set_xlim(0,1.2)
axes[1].set_xlim(0,1.2)

```

```

axes[0].invert_xaxis()# Invert the x-axis of the first subplot

axes[0].set(yticks=y, yticklabels=[])
for yloc, selectedASVs in zip(y, featurenames):
    axes[0].annotate(selectedASVs, (0.5, yloc), xycoords=('figure fraction', 'figure fraction'),
                      ha='center', va='center', fontsize=9)
fig.tight_layout(pad=2.0)
plt.show()

```

## compare the first 15 index by their present ratio

```

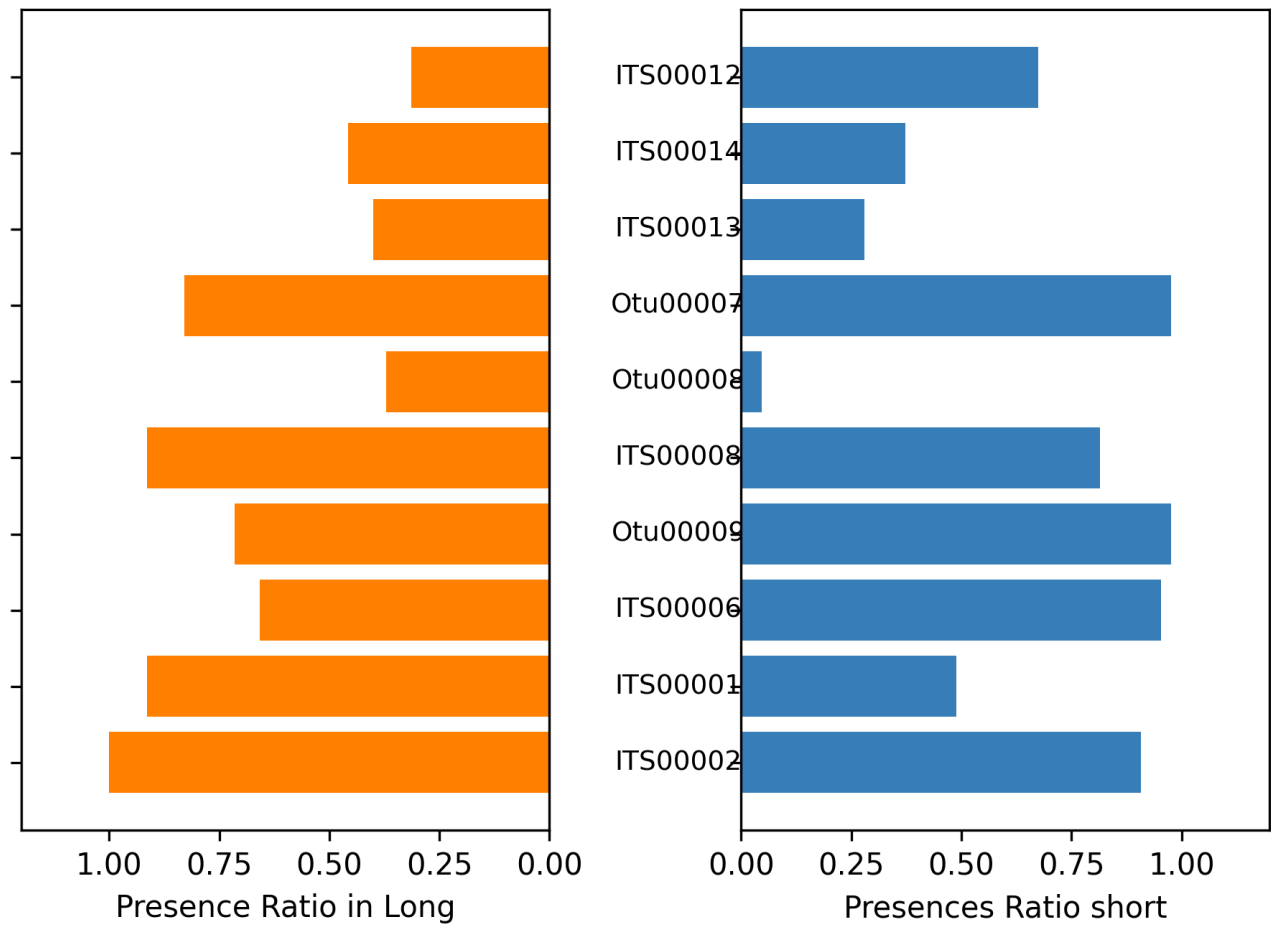
In [32]: import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

entries=10

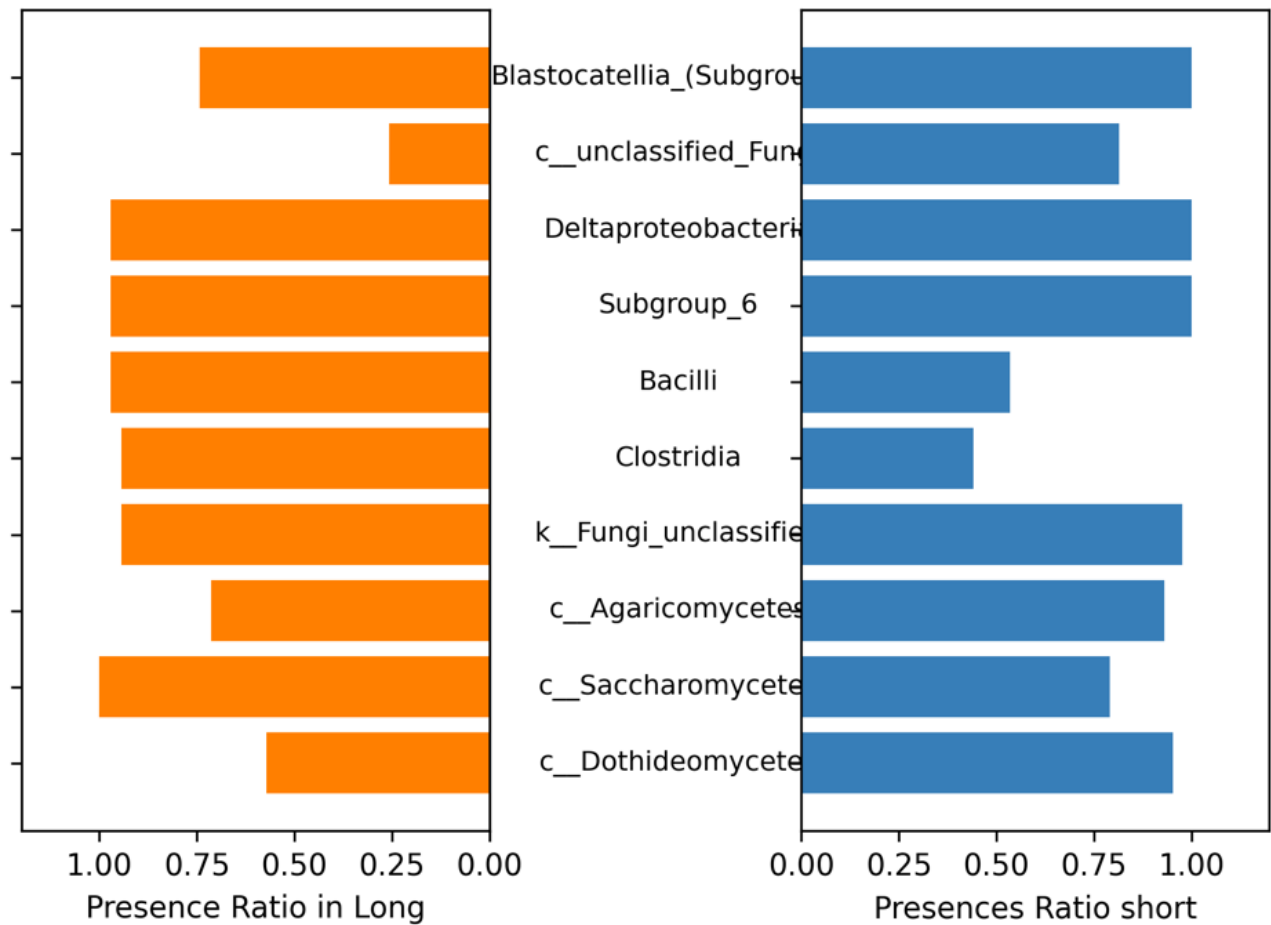
for i, index in enumerate(selectedOTU_index_4tax):
    selectedOTU_index_15=index[:entries]
    #print(selectedOTU_index_15)
    selectedASVs_15=col_names_sig_sorted_4taxa[i][:entries]
    print(selectedASVs_15)
    X_FS_15=data_4taxa[i][:,selectedOTU_index_15]
    #df=pd.DataFrame(data=X_FS_15)
    FS.plotPresenseRatio(X_FS_15,targetLabel,selectedASVs_15,posLabel="LONG")

['ITS00002', 'ITS00001', 'ITS00006', 'Otu00009', 'ITS00008', 'Otu00008', 'Otu00007', 'ITS00013', 'ITS00014', 'ITS00012']
35 43

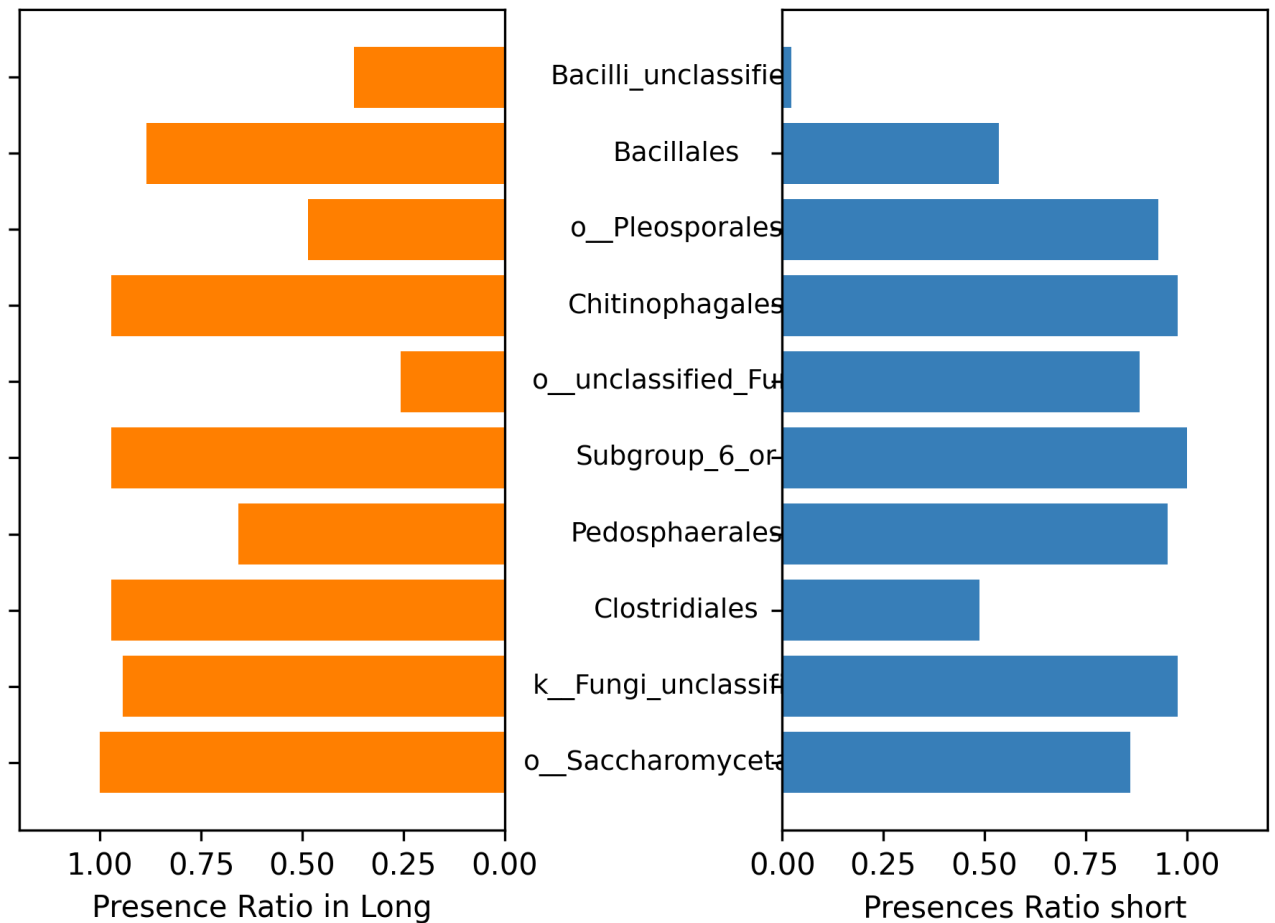
```



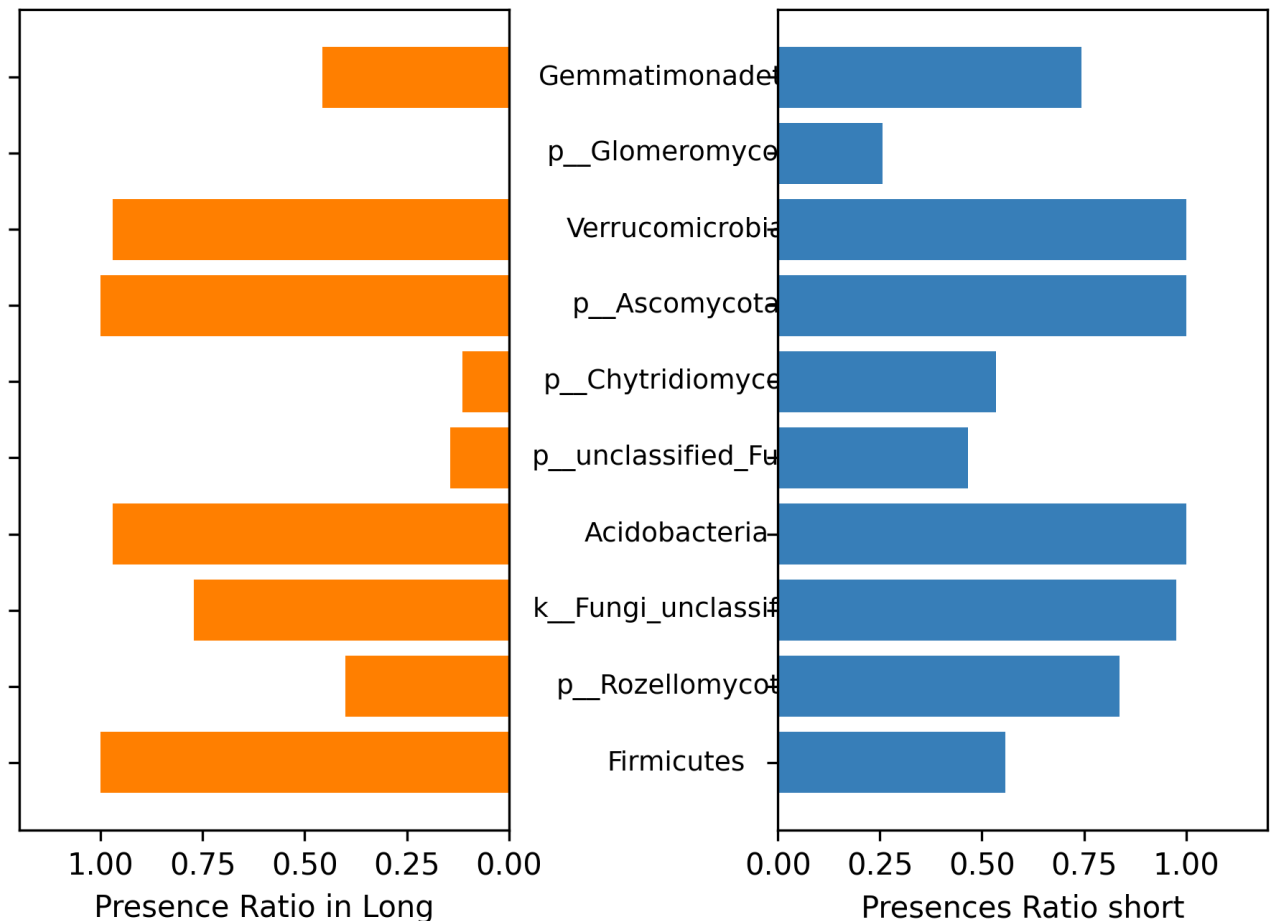
['c\_\_Dothideomycetes', 'c\_\_Saccharomycetes', 'c\_\_Agaricomycetes', 'k\_\_Fungi\_unclassified', 'Clostridia', 'Bacilli', 'Subgroup\_6', 'Deltaproteobacteria', 'c\_\_unclassified\_Fungi', 'Blastocatellia\_(Subgroup\_4)']  
35 43



['o\_\_Saccharomycetales', 'k\_\_Fungi\_unclassified', 'Clostridiales', 'Pedosphaerales', 'Subgroup\_6\_or', 'o\_\_unclassified\_Fungi', 'Chitinophagales', 'o\_\_Plasmodiales', 'Bacillales', 'Bacilli\_unclassified']  
35 43



['Firmicutes', 'p\_Rozellomycota', 'k\_Fungi\_unclassified', 'Acidobacteria',  
'p\_unclassified\_Fungi', 'p\_Chytridiomycota', 'p\_Ascomycota', 'Verrucomicrobia', 'p\_Glomeromycota', 'Gemmatimonadetes']  
35 43



```
In [33]: ##### check the plot results (use phylum as example)
Phy_select_index = selected0TU_index_4taxa[3]
Phy_select_index_5 = Phy_select_index[0:5]
Phy_select_index_5
```

```
Out[33]: [4, 39, 38, 2, 44]
```

```
In [34]: Phy_select_label_5 = [col_names_4taxa[3][i] for i in Phy_select_index_5]
print(Phy_select_label_5)
print(col_names_sig_sorted_4taxa[3][0:5])
print(weights_sig_sorted_4taxa[3][0:5])

['Firmicutes', 'p_Rozellomycota', 'k_Fungi_unclassified', 'Acidobacteria',
'p_unclassified_Fungi']
['Firmicutes', 'p_Rozellomycota', 'k_Fungi_unclassified', 'Acidobacteria',
'p_unclassified_Fungi']
[34.26918116 23.82576096 22.9256874 22.247891 14.69913199]
```

```
In [35]: X_pyhlum = data_4taxa[3][:, Phy_select_index_5]
#X_pyhlum = np.where(X_pyhlum > 0.01, 1, 0)
```

```
In [36]: # test
```

```
data_phy_test=FS.relative_abundance(data_4taxa[3])
FS.OTU_H_Score_arr(data_phy_test[:,selected0TU_index_4tax[3][0:5]],targetLab
```

```
Out[36]: array([34.26918116, 23.82576096, 22.9256874 , 22.247891 , 14.69913199])
```

```
In [37]: print
         for i in Phy_select_index_5:
             print(
                 FS.OTU_H_Score(data_4taxa[3][:,i],targetLabel,cutOff=0.01)
             )
```

```
35.19309474746626
19.406930484881627
22.6285966609193
22.247890996257183
25.48530215736571
```

```
In [38]: print(weights_sig_sorted_4taxa[3])
         print(weights_4taxa[3][selected0TU_index_4tax[3]])
```

```
[34.26918116 23.82576096 22.9256874 22.247891 14.69913199 13.53380762
 12.04796669 8.66578317 6.16136704 4.87578408 4.84128675 4.55765047
 3.53086702]
[34.26918116 23.82576096 22.9256874 22.247891 14.69913199 13.53380762
 12.04796669 8.66578317 6.16136704 4.87578408 4.84128675 4.55765047
 3.53086702]
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

## Negative Gini Impurity

Gini Impurity is the probability of incorrectly classifying a randomly chosen element in the dataset if it were randomly labeled according to the class distribution in the dataset. It's calculated as:

$$G = 1 - \sum_{i=1}^C p_i^2$$

where C is the number of classes. (which means it can be used to measure for multiple level classification)

Here I will use the negative Gini Impurity to measure each OTU, if NG is large (1) which

means the OTU only exist in one class, if NG value is small( $\$1/c\$$ ) which means the OTU is evenly distributed among the classes.

$$NG = \sum_{i=1}^C p_i^2$$

```
In [39]: np.unique(y, return_counts=True)
```

```
Out[39]: (array(['LONG', 'SHORT'], dtype='<U5'), array([35, 43]))
```

```
In [40]: # NG for selected OTU
```

```
NG_4tax = []
for i, data in enumerate(data_4taxa):
    X_FS = data[:,selectedOTU_index_4tax[i]]
    X_lasso = data[:,xind_lasso_4taxa[i]]
    NG_selected = metric.Neg_GINI(X_FS,y,cutOff=0.01)
    NG_Lasso = metric.Neg_GINI(X_lasso,y,cutOff=0.01)
    print(NG_selected.shape)
    print(NG_Lasso.shape)
    NG_4tax.append([NG_selected,NG_Lasso])
```

```
(23,)
(40,)
(26,)
(24,)
(34,)
(22,)
(13,)
(10,)
```

```
In [ ]:
```

```
In [41]: # compare the selected and non select by lasso
```

```
# Number of subplots
```

```
num_plots = len(data_4taxa)
```

```
# Create a figure with a grid of subplots
```

```
plt.figure(figsize=(4, 4 * num_plots))
```

```
# Loop through each index and create a subplot
```

```
for i in range(num_plots):
```

```
    plt.subplot(num_plots, 1, i + 1) # (nrows, ncols, index)
```

```
    plt.boxplot([NG_4tax[i][0], NG_4tax[i][1]], tick_labels=['SelectMicro',
```

```
    plt.title(f'NG results of the selected OTU by SelectMicro vs. Lasso - {t
```

```
    plt.ylabel('NG')
```

```
    plt.grid(axis='y')
```

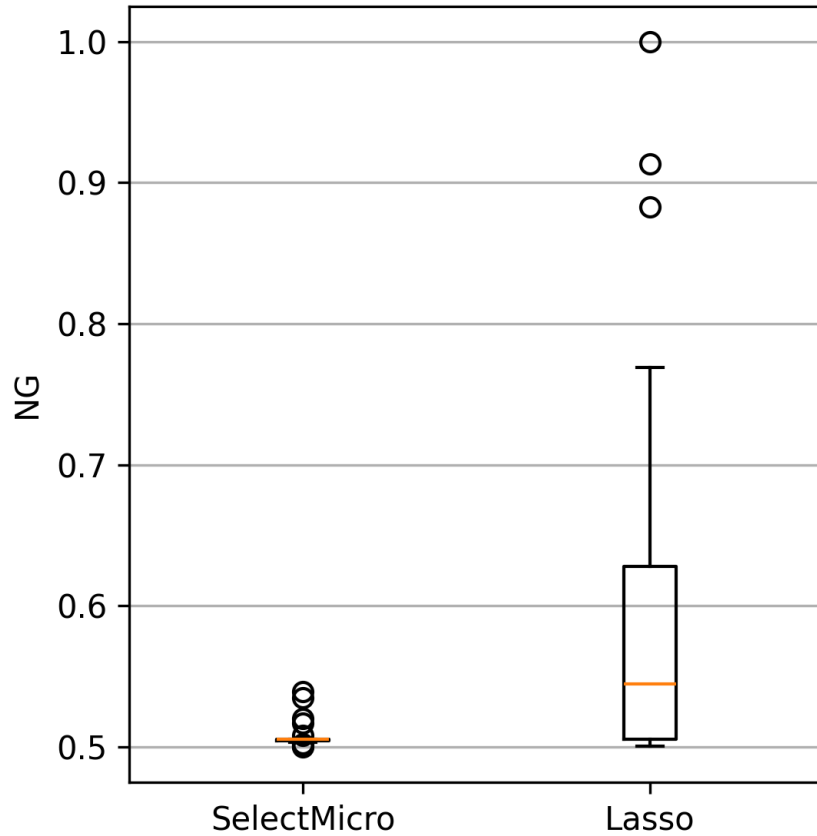
```
# Adjust layout
```

```
plt.tight_layout() # Adjusts the subplots to fit into the figure area.
```

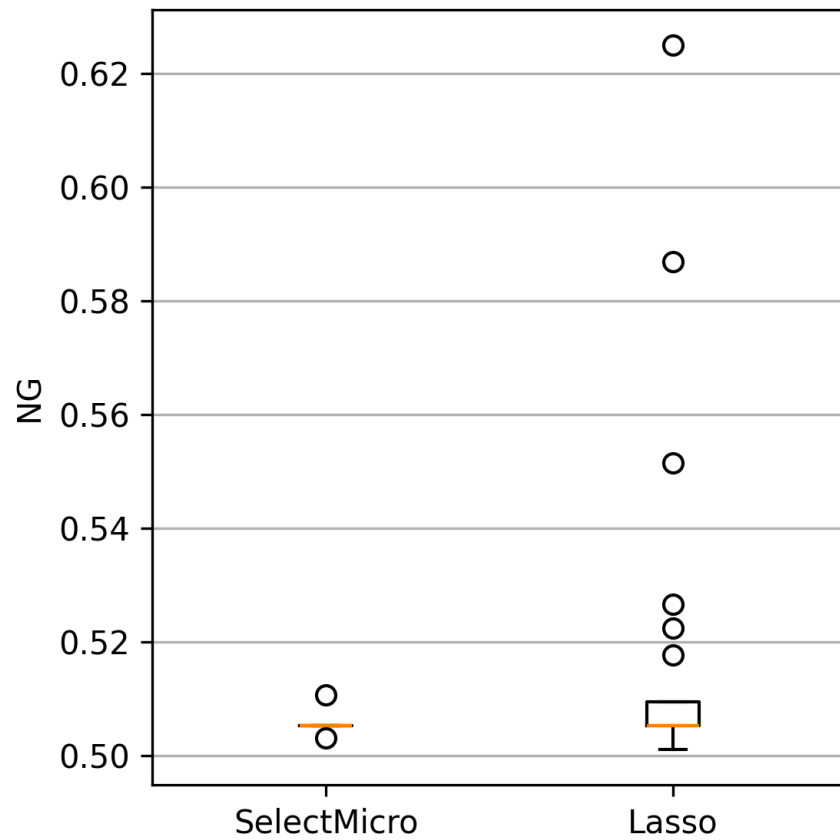
```
plt.show() # Show all plots at once
```



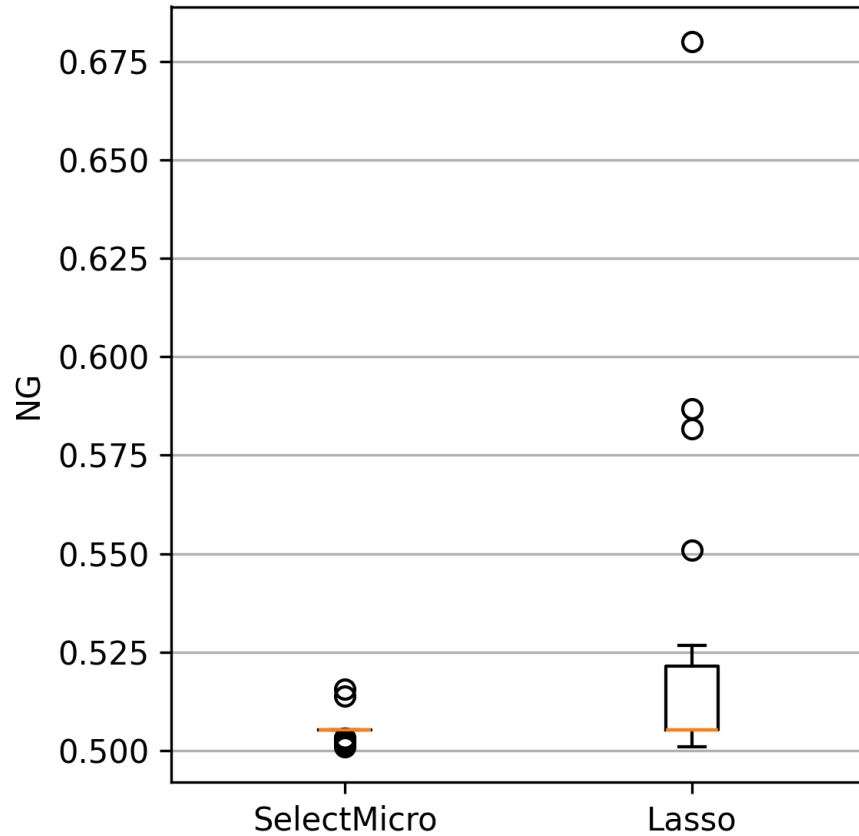
## NG results of the selected OTU by SelectMicro vs. Lasso - OTU



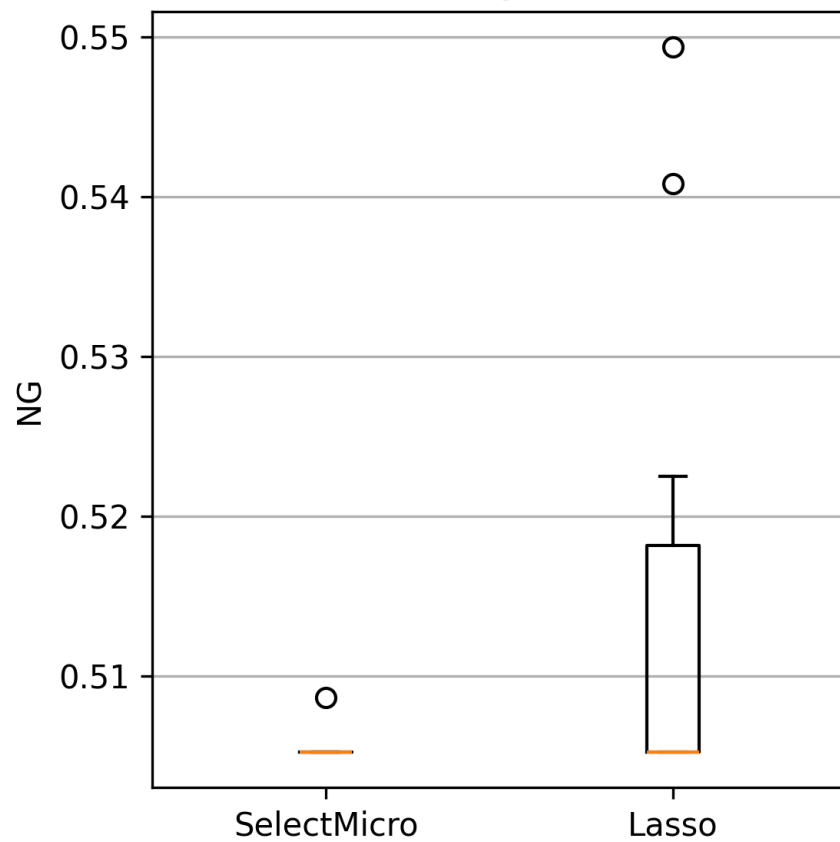
## NG results of the selected OTU by SelectMicro vs. Lasso - class



## NG results of the selected OTU by SelectMicro vs. Lasso - order



## NG results of the selected OTU by SelectMicro vs. Lasso - phylum



```
In [42]: ### calculate the outlier label
NG_4tax_lasso = [x[1] for x in NG_4tax]
```

```
In [43]: NG_4tax_lasso
```

```
Out[43]: [array([0.5052597 , 0.5052597 , 0.545      , 0.50146092, 0.58477509,
                0.50413223, 0.51209877, 0.91322314, 0.50803412, 0.76888889,
                0.72222222, 0.51833409, 0.50106509, 0.5463138 , 0.53125    ,
                0.71125    , 0.50378072, 0.54081633, 0.54419284, 1.          ,
                0.63718821, 0.59981892, 0.5498615 , 0.75510204, 0.69160998,
                0.50468262, 0.61795918, 0.5739645 , 0.625      , 0.76888889,
                0.51020408, 0.5052597 , 0.5052597 , 0.50013437, 0.55555556,
                0.50195312, 0.625      , 0.52      , 0.8828125 , 0.53125    ]),
          array([0.5052597 , 0.5052597 , 0.50311634, 0.5224977 , 0.50295858,
                0.51774837, 0.5052597 , 0.5052597 , 0.58680556, 0.5052597 ,
                0.5052597 , 0.5052597 , 0.5052597 , 0.5052597 , 0.50106509,
                0.50347222, 0.5052597 , 0.5066568 , 0.5052597 , 0.5052597 ,
                0.52662722, 0.55144179, 0.50138504, 0.625      ]),
          array([0.5052597 , 0.50089268, 0.5052597 , 0.50311634, 0.50295858,
                0.5052597 , 0.58680556, 0.68      , 0.5052597 , 0.5052597 ,
                0.50106509, 0.5052597 , 0.51557093, 0.50683083, 0.5052597 ,
                0.5066568 , 0.52662722, 0.51300728, 0.58171118, 0.55092802,
                0.52326028, 0.50138504]),
          array([0.5052597 , 0.5052597 , 0.5052597 , 0.5224977 , 0.54938776,
                0.5052597 , 0.5052597 , 0.5052597 , 0.5052597 , 0.54081633]))]
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [44]: def Outlier_array(arr_input):
# Calculate Q1, Q3, and IQR
    Q1 = np.percentile(arr_input, 25)
    Q3 = np.percentile(arr_input, 75)
    IQR = Q3 - Q1

    # Define outlier bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
```

```
# Get the indices of outliers
outlier_indices = np.where((arr_input < lower_bound) | (arr_input > upper_bound))

# Print the result
return arr_input[outlier_indices],outlier_indices
```

```
In [45]: NG_outlier_list = []
NG_outlier_label_list = []
for i, ng_lasso in enumerate(NG_4tax_lasso):
    NG_outlier, outlier_indice = Outlier_array(ng_lasso)
    NG_outlier_list.append(NG_outlier)
    outlier_ind_origin = xind_lasso_4taxa[i][outlier_indice]
    ## the outlier indice is the indice in the lasso data not in the origin
    outlier_label = [col_names_4taxa[i][j] for j in outlier_ind_origin]
    NG_outlier_label_list.append(outlier_label)
```

```
In [ ]:
```

```
In [46]: print(NG_outlier_list)

[array([0.91322314, 1.          , 0.8828125 ]), array([0.5224977 , 0.51774837,
0.58680556, 0.52662722, 0.55144179,
          0.625          ]), array([0.58680556, 0.68          , 0.58171118, 0.5509280
2]), array([0.54938776, 0.54081633])]
```

```
In [47]: print(NG_outlier_label_list)

[['0tu01137', '0tu03746', 'ITS02105'], ['Fusobacteriia', 'Latescibacteria',
'BRH-c20a', 'c__Malasseziomycetes', 'c__Exobasidiomycetes', 'c__Polychytriom
ycetes'], ['BRH-c20a_or', 'Mollicutes_RF39', 'o__Microstromatales', 'o__Chyt
ridiales'], ['Epsilonbacteraeota', 'p__Blastocladiomycota']]
```

```
In [48]: # check the phylum outlier in lasso
outlier_ind_origin
```

```
Out[48]: array([15, 49])
```

```
In [49]: FS.OTU_H_Score_arr(FS.relative_abundance(data_4taxa[3]),targetLabel)[outlier
```

```
Out[49]: array([0., 0.])
```

```
In [50]: lasso_phylum_outlier_arr = FS.relative_abundance(data_4taxa[3])[:,outlier_in
```

```
In [51]: # Check if any element is larger than the threshold
result = (lasso_phylum_outlier_arr > 0.01).any()

print("Is there any value larger than the threshold?", result)
```

Is there any value larger than the threshold? False

In [ ]:

In [ ]:

In [ ]:

In [ ]:

## Analysis of the top features

```
In [54]: for i, label in enumerate(col_names_sig_sorted_4taxa):
          print(taxlabels[i])
          print(label)
```

OTU

```
['ITS00002', 'ITS00001', 'ITS00006', 'Otu00009', 'ITS00008', 'Otu00008', 'Otu00007', 'ITS00013', 'ITS00014', 'ITS00012', 'Otu00003', 'Otu00014', 'Otu00039', 'ITS00011', 'ITS00043', 'Otu00019', 'ITS00022', 'ITS00028', 'ITS00045', 'ITS00027', 'ITS00004', 'ITS00018', 'Otu00015']
```

class

```
['c__Dothideomycetes', 'c__Saccharomycetes', 'c__Agaricomycetes', 'k__Fungi_unclassified', 'Clostridia', 'Bacilli', 'Subgroup_6', 'Deltaproteobacteria', 'c__unclassified_Fungi', 'Blastocatellia(Subgroup_4)', 'c__Rozellomycotina_cls_Incertae_sedis', 'Verrucomicrobiae', 'Phycisphaerae', 'c__unclassified_Rozellomycota', 'Actinobacteria', 'Negativicutes', 'c__Eurotiomycetes', 'Gemmatimonadetes', 'Bacteroidia', 'c__Tremellomycetes', 'c__Mortierellomycetes', 'c__Rhizophydiomycetes', 'p__Basidiomycota_unclassified', 'c__Glomeromycetes', 'c__Sordariomycetes', 'p__Rozellomycota_unclassified']
```

order

```
['o__Saccharomycetales', 'k__Fungi_unclassified', 'Clostridiales', 'Pedosphaerales', 'Subgroup_6_or', 'o__unclassified_Fungi', 'Chitinophagales', 'o__Plaesporales', 'Bacillales', 'Bacilli_unclassified', 'Corynebacteriales', 'Tepididysphaerales', 'o__Agaricales', 'Pyrinomonadales', 'o__unclassified_Rozellomycota', 'Myxococcales', 'Lactobacillales', 'Pseudomonadales', 'Selenomonadales', 'Bacteroidales', 'o__Thelephorales', 'Gemmatimonadales', 'Blastocatellales', 'o__Trichosporonales', 'Cardiobacteriales', 'o__Eurotiales', 'o__Mortierellales', 'Sphingomonadales', 'o__Rhizophydiales', 'p__Basidiomycota_unclassified', 'o__GS08', 'o__Russulales', 'Betaproteobacteriales', 'p__Rozellomycota_unclassified']
```

phylum

```
['Firmicutes', 'p__Rozellomycota', 'k__Fungi_unclassified', 'Acidobacteria', 'p__unclassified_Fungi', 'p__Chytridiomycota', 'p__Ascomycota', 'Verrucomicrobia', 'p__Glomeromycota', 'Gemmatimonadetes', 'Planctomycetes', 'Bacteroidetes', 'p__Mortierellomycota']
```