First, load the data,in this notebook we are using the PMI data from the paper:"Environmental predictors impact microbialbased 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
2	1989.56009	3829.070365	412.942989	821.982537	1300.667694	
3	2973.68000	3194.403374	419.280034	848.987596	1688.948376	10
4	4027.69203	3694.823154	475.772750	833.851140	1209.230357	4
...	...	...	...	...	...	
73	4447.32241	2456.386357	702.629879	1096.355563	2612.831828	1
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	2

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 [11]:

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

Out[11]:

```
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 [12]:

```
# 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']
```

OTU

```
In [ ]:
```

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

```
In [ ]:
```

```
In [13]: weights_4taxa = []
```

```
In [14]: 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 [15]: for weight in weights_4taxa:
             print(len(weight))
```

```
7414
177
411
51
```

```
In [ ]:
```

```
In [ ]:
```

```
In [16]: selectedOTU_index_4tax = []
         eps_4tax = []
```

```
In [17]: for weight in weights_4taxa:
             selectedOTU_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 [18]:

```
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
    weights_sig_sorted_4taxa.append(weights_sig_sorted)
    col_names_sig_sorted_4taxa.append(col_names_sig_sorted)
```

In [ ]:

```
```

In [19]:

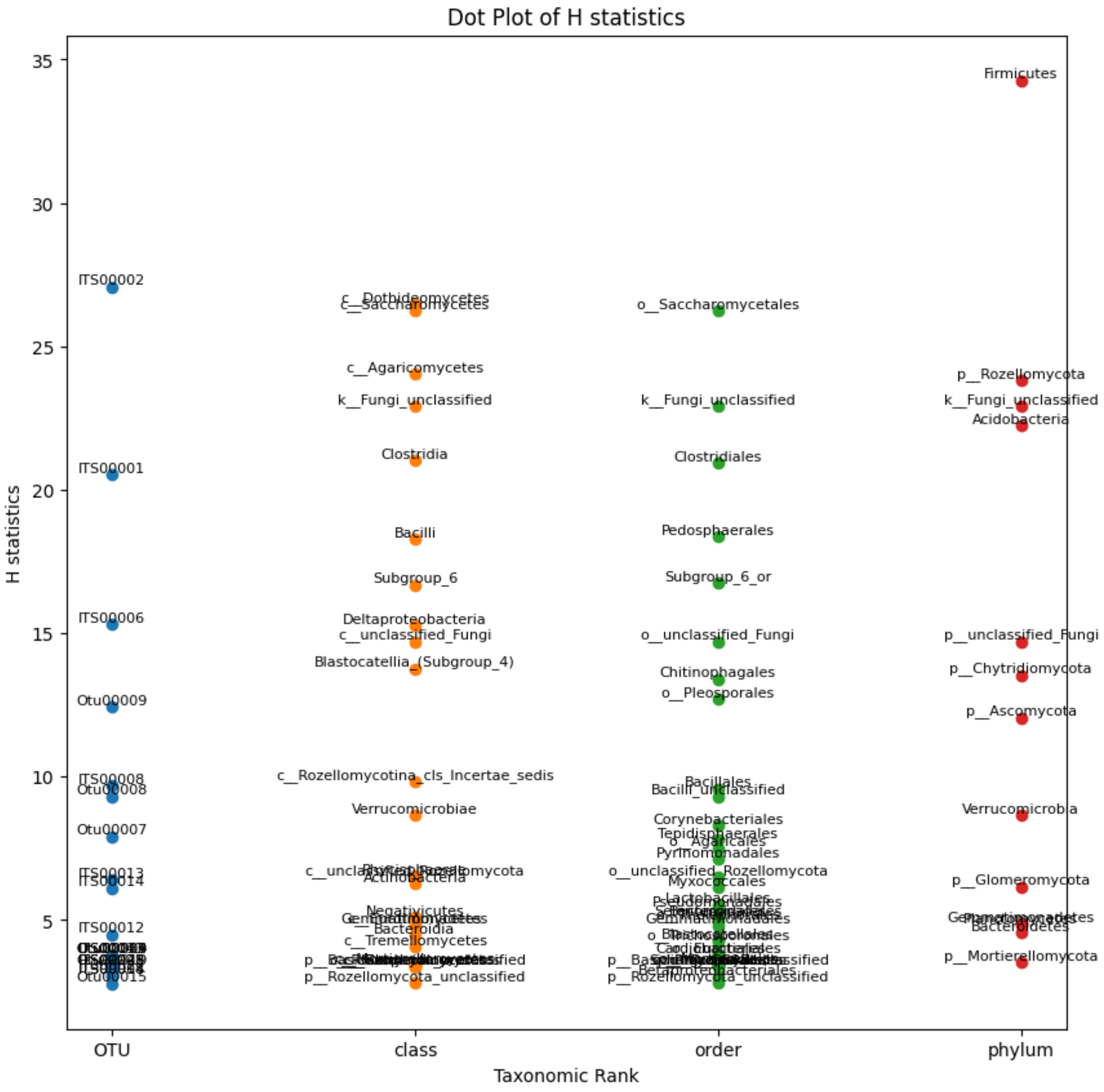
```
taxlabels = ['OTU', 'class', 'order', 'phylum']

# 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 grou
    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 la
        plt.text(taxlabels[i], z, label, ha='center', va='bottom', fontsize=

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



```
In [ ]:
```

```
In [20]: #plot the h statistics and cutoff descendingly
         #for i in range(len(weights_4taxa)):
             #FS.plotWeightedIndex(weights_4taxa[i],threshold=eps_4tax[i])
```

```
In [21]: data_4taxa[0]
```

Out[21]:

```
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 numer 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 [22]:

```
iter =30
cls = ["RF","SVM"]
```

In [23]:

```
targetLabel=y
```

In [24]:

```
data_subset_4taxa = []
X_lasso_4taxa = []
xlabel_lasso_4taxa = []
for i, data  in enumerate(data_4taxa):
    X_lasso,xlabel_lasso = RunML_continue.LassoFeatureSelection(data,targetL
    X_lasso_4taxa.append(X_lasso_4taxa)
    xlabel_lasso_4taxa.append(xlabel_lasso)
    data_subset = {"AllFeatures":data,
                 "SelectMicro": data[:,selectedOTU_index_4tax[i]],
               "Lasso":X_lasso,
               "Random":data
              }
    data_subset_4taxa.append(data_subset)
```

In [25]:

```
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 [65]:

```
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 [27]:

```
#dict_cm = RunML_continue.runClassifier_FScompare(data_subsets= data_subset,
```

In [ ]:

In [ ]:

In [ ]:

compare the first 15 index by their present ratio

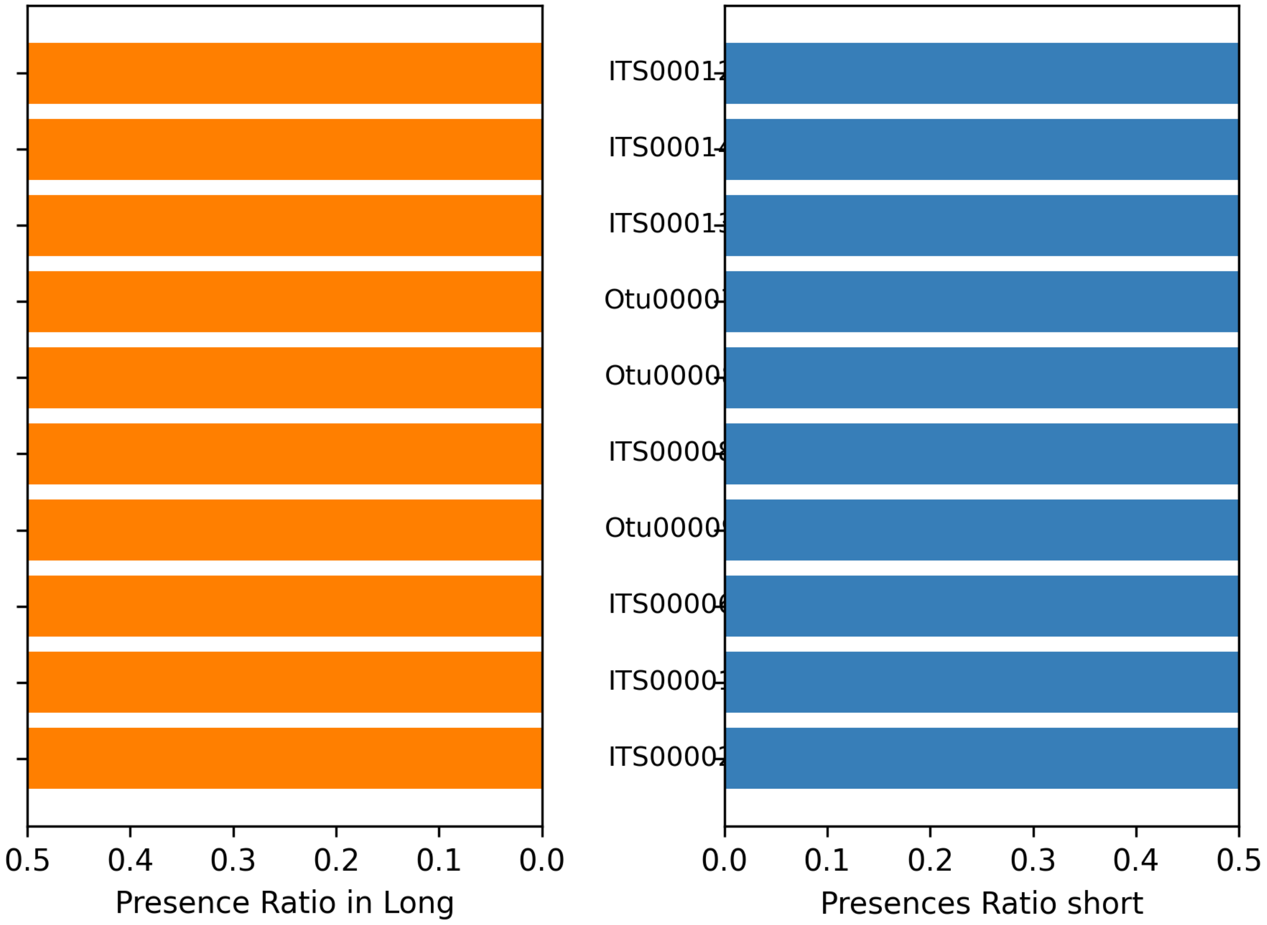
In [28]:

```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
```

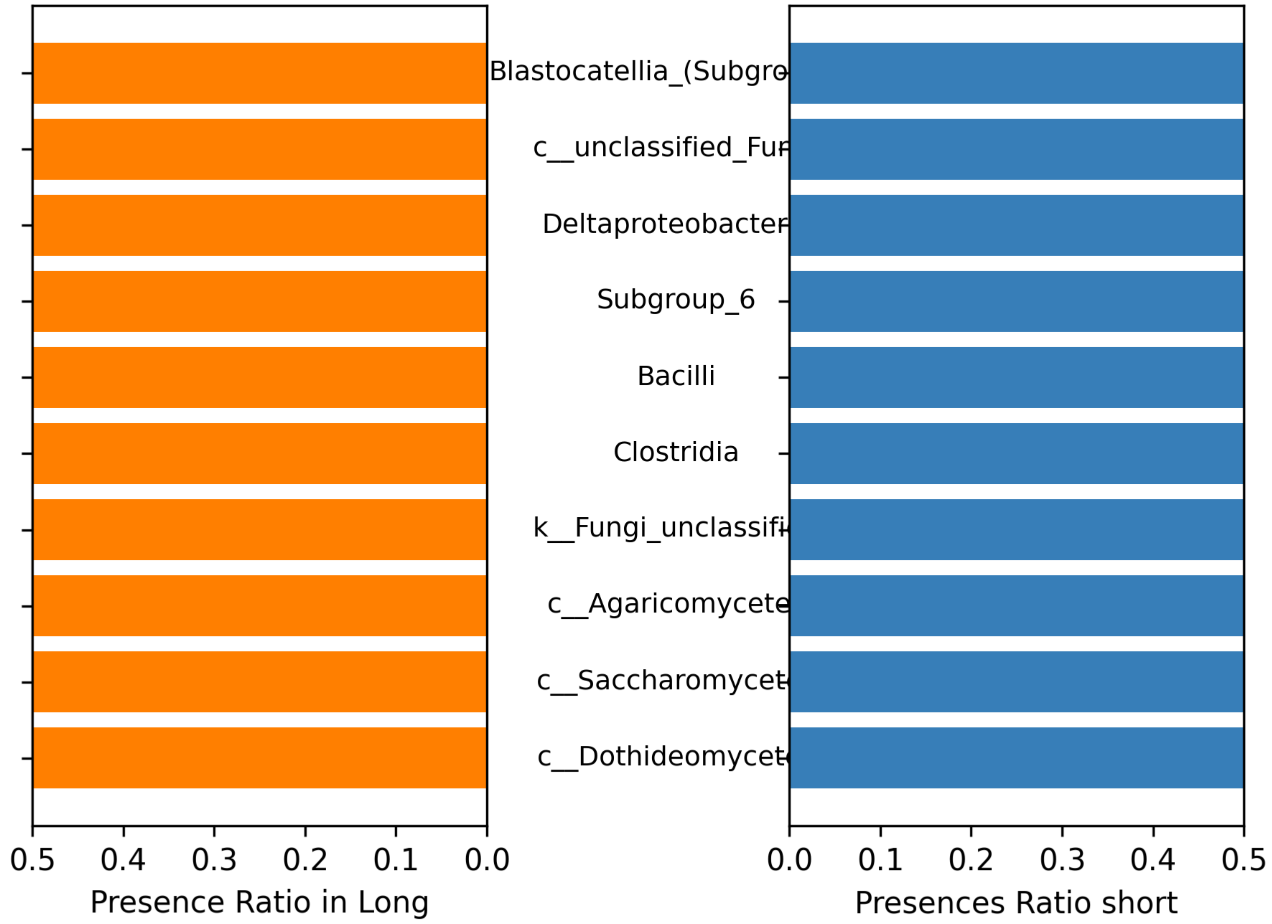
```
entries=10

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)
    RunML.plotPresenseRatio(X_FS_15,targetLabel,selectedASVs_15,posLabel="L
```

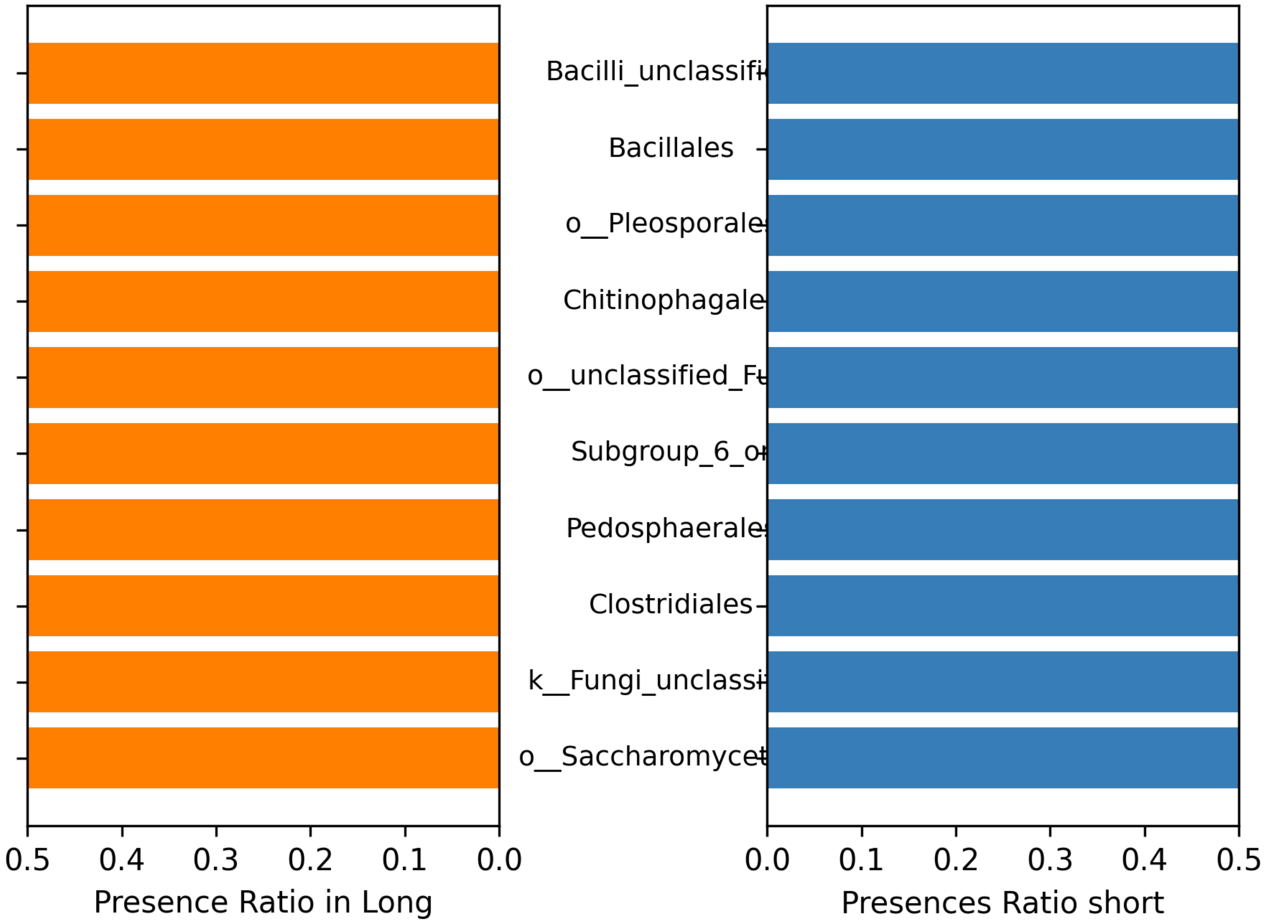
```
['ITS00002', 'ITS00001', 'ITS00006', 'Otu00009', 'ITS00008', 'Otu00008', 'Ot
u00007', 'ITS00013', 'ITS00014', 'ITS00012']
35 43
```



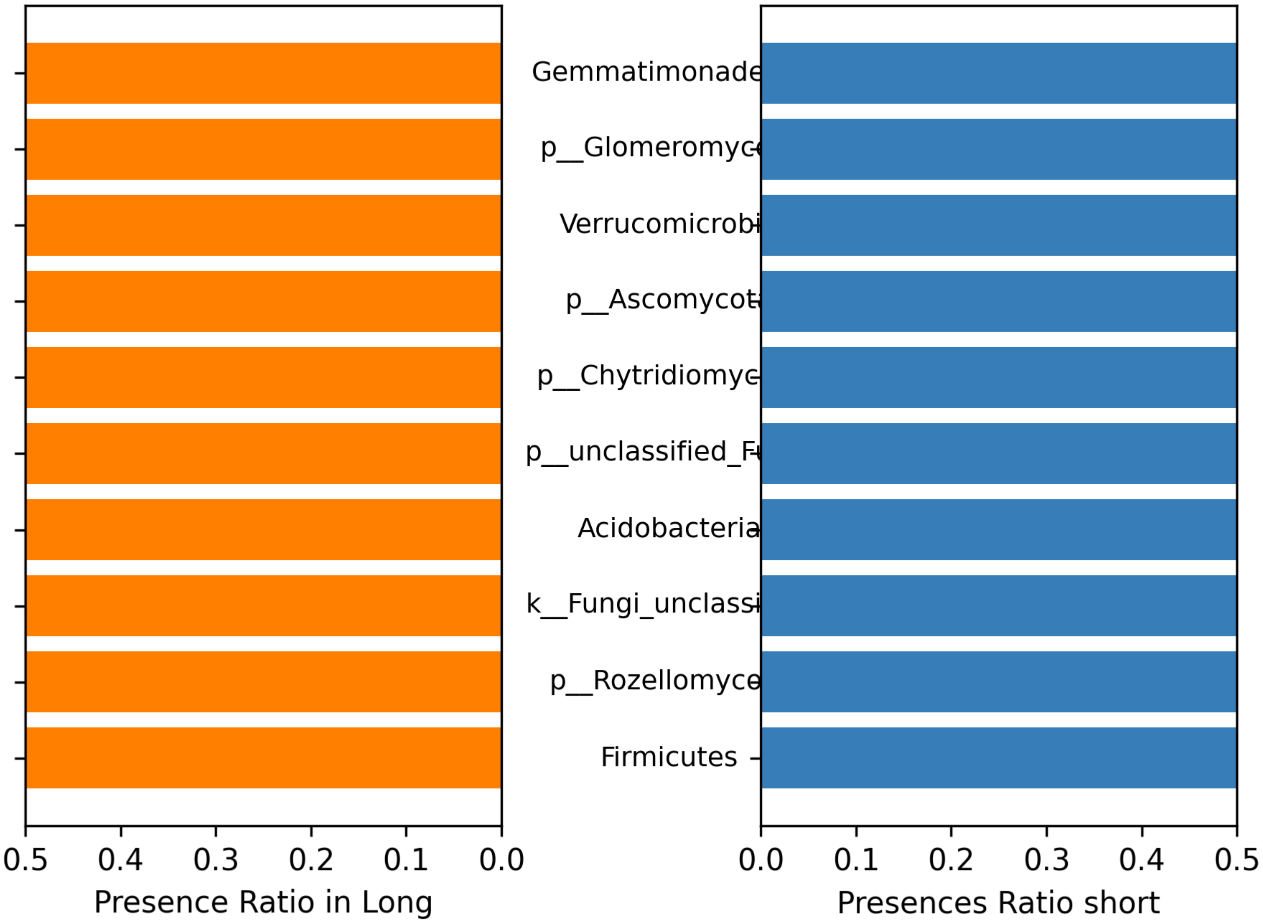
```
['c__Dothideomycetes', 'c__Saccharomycetes', 'c__Agaricomycetes', 'k__Fungi_
unclassified', 'Clostridia', 'Bacilli', 'Subgroup_6', 'Deltaproteobacteria',
'c__unclassified_Fungi', 'Blastocatellia_(Subgroup_4)']
35 43
```



```
['o__Saccharomycetales', 'k__Fungi_unclassified', 'Clostridiales', 'Pedospha
erales', 'Subgroup_6_or', 'o__unclassified_Fungi', 'Chitinophagales', 'o__Pl
eosporales', 'Bacillales', 'Bacilli_unclassified']
35 43
```



```
['Firmicutes', 'p__Rozellomycota', 'k__Fungi_unclassified', 'Acidobacteria',
'p__unclassified_Fungi', 'p__Chytridiomycota', 'p__Ascomycota', 'Verrucomicr
obia', 'p__Glomeromycota', 'Gemmatimonadetes']
35 43
```



In [ ]:

In [ ]:

In [ ]:

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 [29]:

```
len(data_4taxa)
```

Out[29]:

```
4
```

In [30]:

```
np.unique(y, return_counts=True)
```

Out[30]:

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

In [31]:

```
# NG for selected OTU

NG_4tax = []
for i, data  in enumerate(data_4taxa):
    X_FS = data[:,selectedOTU_index_4tax[i]]
    X_lasso = data[:,xlabel_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,)
```

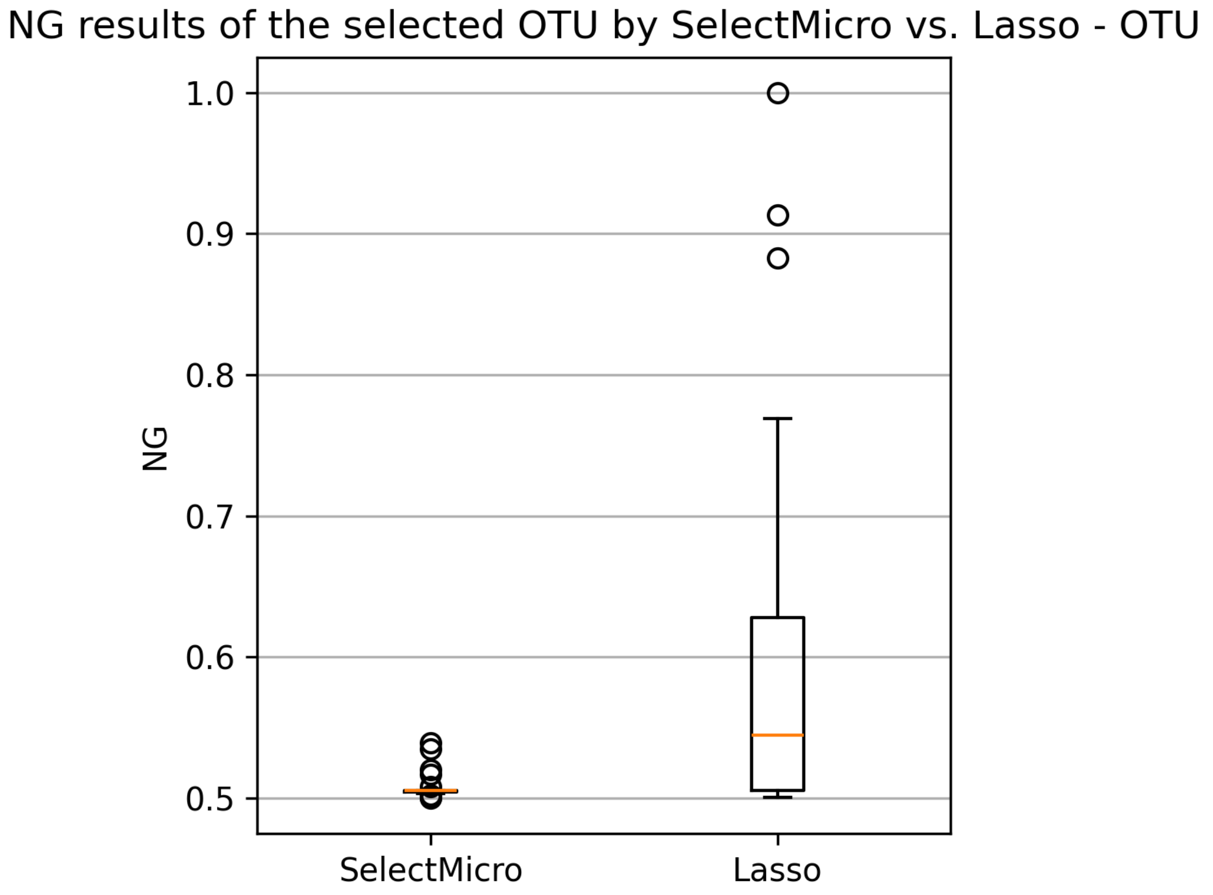
In [ ]:

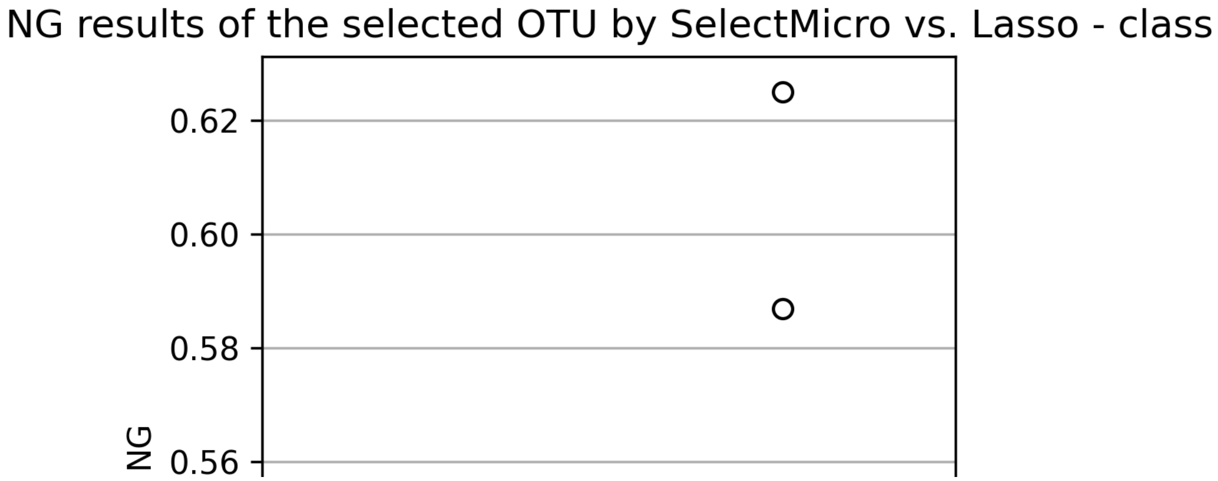
In [32]:

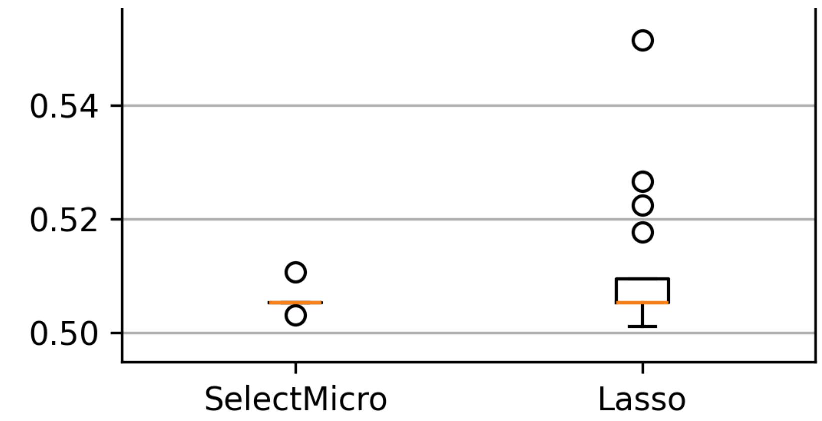
```
# compare the selected and non select by lasso
# Number of subplots
num_plots = len(data_4taxa)

# Create a figure with a grid of subplots
plt.figure(figsize=(4, 4 * num_plots))
```

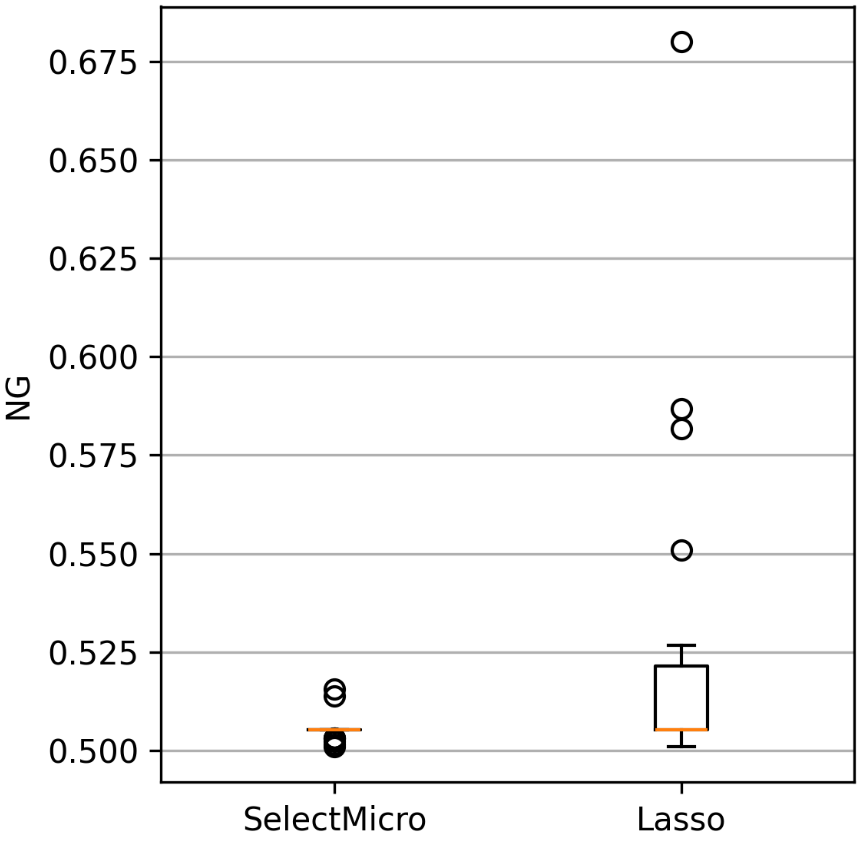
```
# Loop through each index and create a subplot
for i in range(num_plots):
    plt.subplot(num_plots, 1, i + 1)  # (nrows, ncols, index)
    plt.boxplot([NG_4tax[i][0], NG_4tax[i][1]], tick_labels=['SelectMicro',
    plt.title(f'NG results of the selected OTU by SelectMicro vs. Lasso - {t
    plt.ylabel('NG')
    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

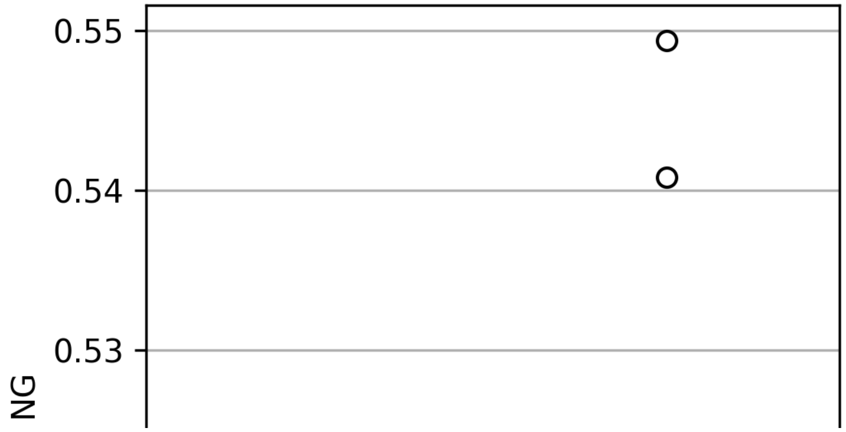

NG results of the selected OTU by SelectMicro vs. Lasso - class

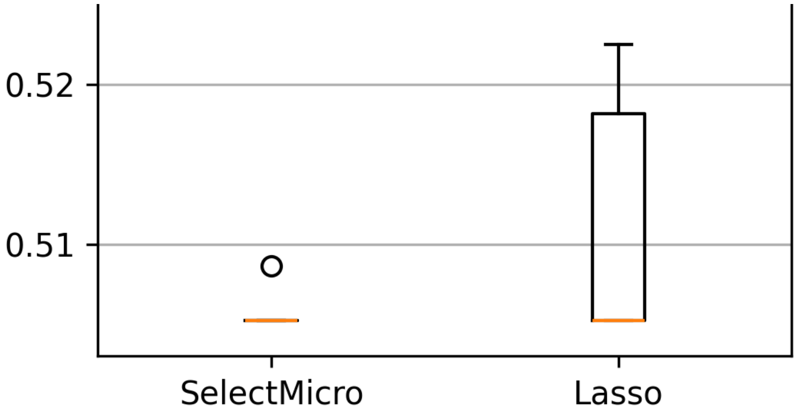


NG results of the selected OTU by SelectMicro vs. Lasso - order



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





```
In [37]: ###  calculate the outlier label
         NG_4tax_lasso =  [x[1] for x in NG_4tax]
```

```
In [34]: NG_4tax_lasso
```

```
Out[34]: [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 [ ]:
```

```
In [61]: NG_outlier_list = []
         NG_outlier_label_list = []
         for i, ng_lasso in enumerate(NG_4tax_lasso):
             NG_outlier, outlier_indice = outlier_array(ng_lasso,xlabel_lasso_4taxa[i
             NG_outlier_list.append(NG_outlier)
             outlier_label = [col_names_4taxa[i][j] for j in outlier_indice]
             NG_outlier_label_list.append(outlier_label)
```

```
In [64]: 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 [63]: print(NG_outlier_label_list)
```

```
[['Otu01137', 'Otu03746', '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']]
```