Ensemble_multiclass_Classes_LLO_2025

March 20, 2025

1 Machine learning :- Stacking ensemble

[3]:
```
%matplotlib widget
%matplotlib inline
%matplotlib notebook
import sklearn
print(sklearn.__version__)
from IPython.display import display
```

```
Warning: Cannot change to a different GUI toolkit: notebook. Using widget
instead.
1.4.2
```

2 Cluster Analysis Training

[5]:
```
%matplotlib widget
%matplotlib inline
%matplotlib notebook
import sklearn
print(sklearn.__version__)
from IPython.display import display
```

```
Warning: Cannot change to a different GUI toolkit: notebook. Using widget
instead.
1.4.2
```

[ ]:
```

```

3 Import Packages

[7]:
```
import pandas as pd
import matplotlib as mpl
from sklearn.datasets import make_blobs
from matplotlib import pyplot
from pandas import DataFrame
from numpy import mean
from numpy import std
```

```python
from sklearn import svm as SVM
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GroupKFold
from sklearn.model_selection import LeaveOneGroupOut
from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier
#from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from matplotlib import pyplot
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
#from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import f1_score
from sklearn.metrics import ConfusionMatrixDisplay
```

4 Read Data

[9]:
```python
tdat = pd.read_csv(r"e:/TomGreenSentinel_python/TrainDataCluster_allVar.csv")
tdat.head() # df.shape (rows and column), .describe() statistics , dtypes =␣
↪dtypes
# tdat = tdat[tdat['Class_name']!=7]
```

[9]:

	Cluster	Class_name	ALEVW	APIXL	BRIHT	GASYM	GCOMT	\
0	4	4	1.736667	86599	3935.433282	0.510045	1.804871	
1	4	4	1.512573	206758	3875.895895	0.436798	1.955397	
2	4	5	2.829637	159362	4233.319050	0.876729	2.436042	
3	4	5	1.238374	3015466	4140.930175	0.139987	2.689293	
4	4	4	3.542128	1981104	4138.262278	0.862609	2.468497	

	GDENS	GRECT	GROND	…	SPC1	SGRN	SPC3	\
0	2.119679	0.844501	1.028468	…	23.108603	13.251096	4.773538	
1	2.030020	0.775687	1.441706	…	17.350734	10.584264	4.448697	
2	1.426260	0.701300	1.618818	…	24.208298	14.596754	3.985665	
3	1.847210	0.552326	1.739497	…	23.058734	13.837575	3.926867	
4	1.296347	0.580688	1.936143	…	19.274063	12.500814	3.974518	

```
        SRED      SSTD      SENT      SASM      SDIS       SPC2       SBLU
0  18.154623  0.433930  0.417258  0.053203  0.446993  12.845778  14.382646
1  13.727230  0.426532  0.528463  0.090212  0.418407  10.830070   7.718029
2  14.644826  0.655310  0.577876  0.084576  0.594870  11.091103   9.204409
3  16.369350  0.583064  0.623994  0.106213  0.536636  10.050137  10.675194
4  12.799524  0.442569  0.546097  0.088167  0.422221  11.631352   6.968034

[5 rows x 67 columns]
```

[10]:

```
tdat['Class_name'].unique()
print(tdat.shape)
```

```
(3469, 67)
```

[11]:

```
grades = [2]
status = [1]

tdat['Class_name'] = tdat['Class_name'].replace(grades, status)

builtrep=  [7]
builts = [6]
# tdat[170:175]

tdat['Class_name'] = tdat['Class_name'].replace(builtrep, builts)

orderlist = [1,4, 5, 6, 8, 9]

#[orderlist.index(a) for a in tdat['Class_name']]

tdat['Class_name'] = tdat['Class_name'].map({1: 1, 4: 2, 5: 3, 6: 4, 8: 5, 9:6})
#tdat['Class_name'] = tdat['Class_name'].map({1: "Cropland", 4: "Grassland", 5:␣
↪"Shrubland", 6: "Built-up1", 7: "Built-up2", 8: "Water", 9: "Shadow"})
#tdat['Class_name'] = tdat['Class_name'].map({1: "Cropland", 2:"Fallowland", 4:␣
↪"Grassland", 5: "Shrubland", 6: "Built-up1", 7: "Built-up2", 8: "Water", 9:␣
↪"Shadow"})
```

[12]:

```
tdat['Class_name'].unique()
tdat['Class_name'] = tdat['Class_name'].astype('category')
```

[13]:

```
# tdat['Class_name'] = tdat['Class_name'].map({4: 2, 5: 3, 6: 4, 8:6, 7:5, 9:7,␣
↪1:1})

# tdat['Class_name'] = tdat['Class_name'].astype('category') # pd.
↪factorize(tdat)[0]
# #tdat['Clust'] = tdat['Class_name'].astype('category')
# tdat.dtypes
```

[14]:

```python
#Subset features
tdat = tdat[["Class_name", "Cluster","ALEVW",
  "APIXL", "BRIHT", "GASYM", "GCOMT", "GDENS", "GRECT", "GROND",
  "GSHAP", "MAXDF", "NMNIR", "NSNIR", "SMNPC", "SMSMI", "SMSMN",
  "SMSMX", "SMSPC1", "SMSPC2", "SMSPC3", "SMSSD", "SSNPC", "SSSMI",
  "SSSMN", "SSSMX", "SSSPC1", "SSSPC3", "SSSPC34", "SSSST", "TMEAN"]]
print(tdat.head)
```

```
<bound method NDFrame.head of      Class_name  Cluster     ALEVW    APIXL
BRIHT     GASYM  \
0             2        4   1.736667    86599  3.935433e+03  0.510045
1             2        4   1.512573   206758  3.875896e+03  0.436798
2             3        4   2.829637   159362  4.233319e+03  0.876729
3             3        4   1.238374  3015466  4.140930e+03  0.139987
4             2        4   3.542128  1981104  4.138262e+03  0.862609
…           …      …        …      …            …        …
3464          5        6   1.071713    47400  4.493687e+03  0.330543
3465          2        6   2.335278    45415  4.033686e+03  0.771519
3466          4        6  11.658339     2848  4.176855e+03  0.994004
3467          2        6   1.742273    64958  3.960804e+03  0.625839
3468          3        6   1.105960   576915 -9.500000e+30  0.372290

         GCOMT     GDENS     GRECT     GROND  …         SMSSD         SSNPC  \
0     1.804871  2.119679  0.844501  1.028468  …  6.000729e-02  1.776927e+02
1     1.955397  2.030020  0.775687  1.441706  …  5.218346e-02  1.767866e+02
2     2.436042  1.426260  0.701300  1.618818  …  4.222881e-02  1.951995e+02
3     2.689293  1.847210  0.552326  1.739497  …  5.943397e-02  3.065921e+02
4     2.468497  1.296347  0.580688  1.936143  …  7.764482e-02  9.676869e+02
…          …        …        …        …   …            …            …
3464  1.424451  2.263752  0.835578  1.067104  …  3.179848e-01  1.754066e+03
3465  2.727268  1.460370  0.601614  1.847027  …  1.253976e-01  1.177911e+03
3466  3.068201  0.631985  0.622096  2.036304  …  1.144908e-01  6.952368e+02
3467  2.866216  1.506282  0.563214  1.919981  …  1.257806e-01  3.645442e+02
3468  1.573563  2.195497  0.811899  1.257753  … -1.430000e+30  4.302515e+08

             SSSMI         SSSMN         SSSMX        SSSPC1        SSSPC3  \
0     2.770221e-02  1.755950e-02  1.758390e-02  5.864197e-02  2.984771e-02
1     1.603211e-02  1.680782e-02  2.442283e-02  5.528118e-02  2.655782e-02
2     3.831834e-02  2.646345e-02  2.224043e-02  8.699163e-02  4.437854e-02
3     6.011213e-02  3.577642e-02  3.403871e-02  1.183755e-01  5.434697e-02
4     3.088346e-02  2.201811e-02  5.572315e-02  7.458532e-02  4.277308e-02
…              …             …             …             …             …
3464  6.728463e-01  4.366240e-01  2.795873e-01  1.470081e+00  4.530421e-01
3465  3.721856e-02  6.251810e-02  8.569962e-02  2.036236e-01  1.066231e-01
3466  6.764927e-02  7.074656e-02  7.625036e-02  2.356814e-01  7.040770e-02
3467  3.803019e-02  2.221112e-02  4.906100e-02  7.342288e-02  9.675785e-02
3468  6.820000e+31  6.820000e+31  6.820000e+31  6.820000e+31  6.820000e+31
```

```
           SSSPC34         SSSST       TMEAN
0     2.085820e-02  8.492637e-03  126.756321
1     1.814192e-02  7.126366e-03  126.637879
2     3.765255e-02  9.751849e-03  127.029304
3     3.726222e-02  1.716263e-02  127.002975
4     5.778858e-02  1.420570e-02  126.781282
…              …             …           …
3464  1.675960e-01  1.568618e-01  127.031458
3465  4.478842e-02  1.825449e-02  129.244753
3466  7.157543e-02  1.765214e-02  127.208682
3467  3.149858e-02  9.726149e-03  126.803067
3468  6.820000e+31  6.820000e+31  130.185809

[3469 rows x 31 columns]>
```

5 Splitting data

5.1 Clustering was done using k-means

For ths sake of simplicity, clustering and its index were extracted from R. This could have been done using python in its entierity However, I was more comfortable in R. Moreover, variogram and other analysis for testing overall training data were accomplished using functions created in R. Which I created in R.

[18]:
```python
## split dataset
traindata = tdat.drop(['Class_name'], axis = 1)
```

[19]:
```python
X = tdat.drop(['Class_name','Cluster'], axis = 1)
Y = tdat['Class_name']

groups = tdat['Cluster']
groups
print(Y)
```

```
0       2
1       2
2       3
3       3
4       2
       ..
3464    5
3465    2
3466    4
3467    2
3468    3
Name: Class_name, Length: 3469, dtype: category
Categories (6, int64): [1, 2, 3, 4, 5, 6]
```

```
[20]: # group_kfold = GroupKFold(n_splits=10)
      # group_kfold
```

```
[21]: # group_kfold.get_n_splits(feat, depv, groups)
```

```
[22]: # gs = group_kfold.split(X, Y, groups)
      # gs
```

```
[23]: # print(gs)
      tdat.head()
```

```
[23]:   Class_name  Cluster     ALEVW    APIXL        BRIHT     GASYM     GCOMT  \
     0           2        4  1.736667    86599  3935.433282  0.510045  1.804871
     1           2        4  1.512573   206758  3875.895895  0.436798  1.955397
     2           3        4  2.829637   159362  4233.319050  0.876729  2.436042
     3           3        4  1.238374  3015466  4140.930175  0.139987  2.689293
     4           2        4  3.542128  1981104  4138.262278  0.862609  2.468497

           GDENS     GRECT     GROND  …     SMSSD       SSNPC     SSSMI  \
     0  2.119679  0.844501  1.028468  …  0.060007  177.692665  0.027702
     1  2.030020  0.775687  1.441706  …  0.052183  176.786602  0.016032
     2  1.426260  0.701300  1.618818  …  0.042229  195.199547  0.038318
     3  1.847210  0.552326  1.739497  …  0.059434  306.592053  0.060112
     4  1.296347  0.580688  1.936143  …  0.077645  967.686945  0.030883

           SSSMN     SSSMX   SSSPC1    SSSPC3   SSSPC34     SSSST       TMEAN
     0  0.017559  0.017584  0.058642  0.029848  0.020858  0.008493  126.756321
     1  0.016808  0.024423  0.055281  0.026558  0.018142  0.007126  126.637879
     2  0.026463  0.022240  0.086992  0.044379  0.037653  0.009752  127.029304
     3  0.035776  0.034039  0.118375  0.054347  0.037262  0.017163  127.002975
     4  0.022018  0.055723  0.074585  0.042773  0.057789  0.014206  126.781282

     [5 rows x 31 columns]
```

Group _k gold could be useful however, leave-group out would be more interesting for train_index, test_index in group_kfold.split(X, Y, groups): print("train:", train_index, "Test:", test_index) X_train, X_test = X.loc[train_index], X.loc[test_index] y_train, y_test = Y.loc[train_index], Y.loc[test_index] print(X_train, X_test, y_train, y_test)

print(X_test); print(X_train)

5.2 Leave One Group out Cross validation

```
[27]: logo = LeaveOneGroupOut()
```

```
[28]: logo.get_n_splits(groups = groups)
```

```
[28]: 10
```

[29]:
```
cv = logo.get_n_splits(groups = groups)
kfold = LeaveOneGroupOut()
```

[30]:
```
rf = RandomForestClassifier(n_jobs=-2, max_depth= 10, n_estimators=1000,␣
 ↪random_state=1234)
```

[31]:
```
scores = cross_val_score(rf, X, Y, scoring='accuracy', cv=cv, n_jobs=-1)
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

```
Accuracy: 0.903 (0.028)
```

[32]:
```
group_kfold = GroupKFold(n_splits=7)
group_kfold.get_n_splits(X, Y, groups)
```

[32]: 7

[33]:
```
print(logo)
print(Y)
print(cv)
```

```
LeaveOneGroupOut()
0       2
1       2
2       3
3       3
4       2
       ..
3464    5
3465    2
3466    4
3467    2
3468    3
Name: Class_name, Length: 3469, dtype: category
Categories (6, int64): [1, 2, 3, 4, 5, 6]
10
```

[34]:
```
# for train_index, test_index in logo.split(X, Y, groups):
#    # print("Train:", train_index, "Test:", test_index)
#     X_train, X_test = X.loc[train_index], X.loc[test_index]
#     y_train, y_test = Y.loc[train_index], Y.loc[test_index]
#    # print(X_train, X_test, y_train, y_test)
```

[35]:
```
import numpy as np
import copy as cp
import matplotlib.pyplot as plt
import seaborn as sns
from typing import Tuple
```

```python
from sklearn.metrics import confusion_matrix
```

[36]:
```python
def cross_val_predict(model, kfold : kfold, groups, X : np.array, y : np.array)␣
↪-> Tuple[np.array, np.array, np.array]:

    model_ = cp.deepcopy(model)

    no_classes = len(np.unique(y))

    actual_classes = np.empty([0], dtype=int)
    predicted_classes = np.empty([0], dtype=int)
    predicted_proba = np.empty([0, no_classes])

    for train_ndx, test_ndx in kfold.split(X, y, groups=groups):

        train_X, train_y, test_X, test_y = X[train_ndx], y[train_ndx],␣
↪X[test_ndx], y[test_ndx]

        actual_classes = np.append(actual_classes, test_y)

        model_.fit(train_X, train_y)
        predicted_classes = np.append(predicted_classes, model_.predict(test_X))

        try:
            predicted_proba = np.append(predicted_proba, model_.
↪predict_proba(test_X), axis=0)
        except:
            predicted_proba = np.append(predicted_proba, np.zeros((len(test_X),␣
↪no_classes), dtype=float), axis=0)

    return actual_classes, predicted_classes, predicted_proba
```

[37]:
```python
def plot_confusion_matrix(actual_classes : np.array, predicted_classes : np.
↪array, sorted_labels : list):

    matrix = confusion_matrix(actual_classes, predicted_classes,␣
↪labels=sorted_labels)

    plt.figure(figsize=(12.8,6))
    sns.heatmap(matrix, annot=True, xticklabels=sorted_labels,␣
↪yticklabels=sorted_labels, cmap="Blues", fmt="g")
    plt.xlabel('Predicted'); plt.ylabel('Actual'); plt.title('Confusion Matrix')

    plt.show()
```

[38]:
```python
rf = RandomForestClassifier(n_jobs=-2, max_depth= 10, n_estimators=1000,␣
↪random_state=1234)
```

[39]:
```
actual_classes, predicted_classes, _ = cross_val_predict(rf, kfold, groups, X.
    ↪to_numpy(), Y.to_numpy())
```

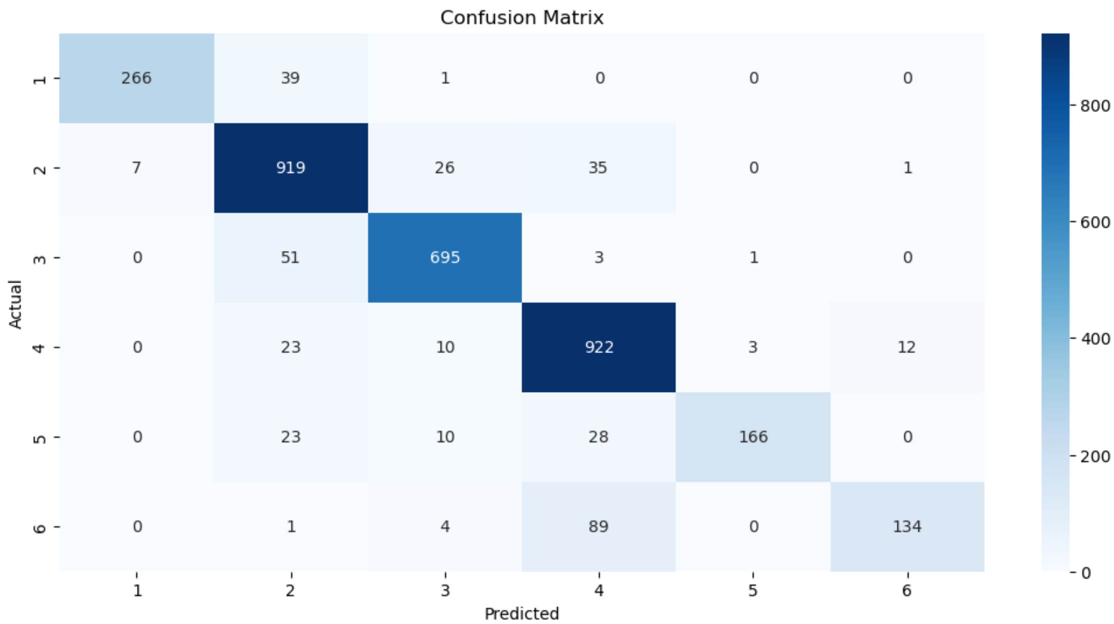
[40]:
```
# Plot confusionmatrix
%matplotlib inline

#target_names = ['Active Crops',  'Grass dominiated', 'Shrub dominated',␣
    ↪'Built-ups',  'Water', 'Shadow']
# conf_mat = confusion_matrix(y_true=y_test,y_pred= rf_fit.
    ↪predict(X_test),labels=target_names)
# fig = plt.figure()
# ax = fig.add_subplot(111)
# cax = ax.matshow(conf_mat)
# ax.set_xticklabels(['']+ target_names)
# ax.set_yticklabels(['']+ target_names)
# plt.xlabel('Predicted Class')
# plt.ylabel('Ground Truth Class')
# plt.show()

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# plot_confusion_matrix
from matplotlib import pyplot as plt
plt.figure(figsize= (15,15), dpi= 1200)
#rf_pred = predicted_classes
#labels =  ['Active Crops', 'Builtup1', 'Builtup2', 'Fallowland', 'Grassland',␣
    ↪'Shadow', 'Shrubland', 'Water']
#labels = ['Cropland',  'Grassland', 'Shrubland', 'Built-up',  'Water',␣
    ↪'Shadow']
labels = [1,2,3,4,5,6]
cm =confusion_matrix(actual_classes, predicted_classes,labels=labels)
#ConfusionMatrixDisplay(cm, display_labels = labels).plot()
plt.figure(figsize=(12.8,6))
sns.heatmap(cm, annot=True, xticklabels=labels, yticklabels=labels,␣
    ↪cmap="Blues", fmt="g")
plt.xlabel('Predicted'); plt.ylabel('Actual'); plt.title('Confusion Matrix')
plt.show()


# plot_confusion_matrix(rf_fit, X_test, y_test,display_labels=labels,cmap = plt.
    ↪cm.BuPu)
# plt.xticks(rotation = 40)
# plt.xlabel('Predicted Class', fontsize = 14,fontweight = "bold", loc =␣
    ↪"center")
# plt.ylabel('True Class', fontsize = 14, fontweight = "bold", loc = "center")
# plt.savefig("test.png", format="png", dpi = 1200, bbox_inches = "tight")
# plt.show()
# #rfconfmat.plot()
```

```
<Figure size 18000x18000 with 0 Axes>
```



[ ]:

[41]:
```python
# transpose confusion matrix reference on columns
np.transpose(cm)
```

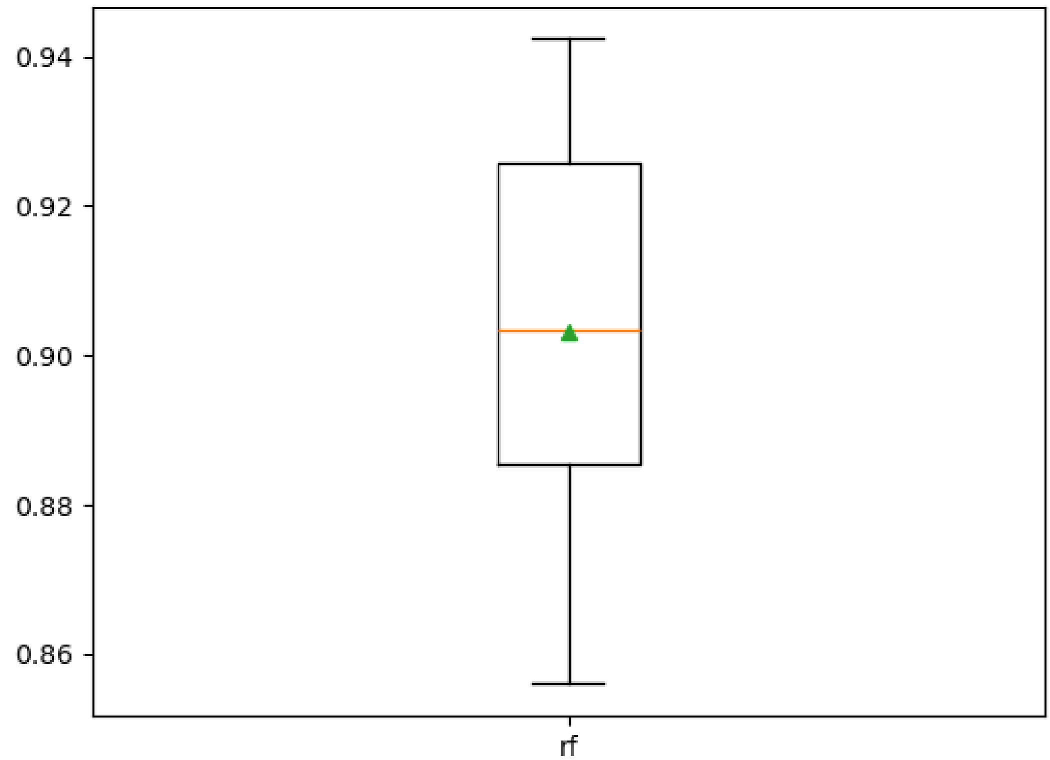
[41]:
```
array([[266,   7,   0,   0,   0,   0],
       [ 39, 919,  51,  23,  23,   1],
       [  1,  26, 695,  10,  10,   4],
       [  0,  35,   3, 922,  28,  89],
       [  0,   0,   1,   3, 166,   0],
       [  0,   1,   0,  12,   0, 134]], dtype=int64)
```

[42]:
```python
def evaluate_model(model, X, y):
    cv = logo.get_n_splits(groups = groups)
    scores = cross_val_score(model, X,y, scoring='accuracy', cv=cv, n_jobs=-1,␣
 ↪error_score='raise')
    return scores
```

[43]:
```python
results, names = list(), list()
scores = evaluate_model(rf, X, Y)
results.append(scores)
names.append('rf')
print('>%s %.3f (%.3f)' % ('rf', mean(scores), std(scores)))
#plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
```

```
pyplot.show()
print(results)
```

```
>rf 0.903 (0.028)
```




```
[array([0.88760807, 0.85590778, 0.91354467, 0.94236311, 0.88472622,
       0.93659942, 0.91930836, 0.870317  , 0.89337176, 0.92774566])]
```

[44]:
```
# MLP
```

[45]:
```
# from sklearn.preprocessing import StandardScaler
# from sklearn.neural_network import MLPClassifier

# sc = StandardScaler()
# #scaler = sc.fit(X_train)
# trainX_scaled = sc.fit_transform(X)
# #testX_scaled  = sc.fit_transform(X_test)

# mlp = MLPClassifier(hidden_layer_sizes=(150,100,50),
#                          max_iter = 1000,activation = 'relu',
#                          solver = 'adam')
# mlpa = MLPClassifier(solver='lbfgs', alpha=1e-5,
#               hidden_layer_sizes=(150, 2), random_state=1)
```

```
# #mlp.fit(trainX_scaled, y_train)

# # Make predictions
# y_train_pred = mlp.predict(trainX_scaled)
# y_test_pred = mlp.predict(testX_scaled)

# # Training set performance
# mlp_train_accuracy = accuracy_score(y_train, y_train_pred) # Calculate␣
↪Accuracy
# mlp_train_mcc = matthews_corrcoef(y_train, y_train_pred) # Calculate MCC
# mlp_train_f1 = f1_score(y_train, y_train_pred, average='weighted') #␣
↪Calculate F1-score

# # Test set performance
# mlp_test_accuracy = accuracy_score(y_test, y_test_pred) # Calculate Accuracy
# mlp_test_mcc = matthews_corrcoef(y_test, y_test_pred) # Calculate MCC
# mlp_test_f1 = f1_score(y_test, y_test_pred, average='weighted') # Calculate␣
↪F1-score

# print('Model performance for Training set')
# print('- Accuracy: %s' % mlp_train_accuracy)
# print('- MCC: %s' % mlp_train_mcc)
# print('- F1 score: %s' % mlp_train_f1)
# print('----------------------------------')
# print('Model performance for Test set')
# print('- Accuracy: %s' % mlp_test_accuracy)
# print('- MCC: %s' % mlp_test_mcc)
# print('- F1 score: %s' % mlp_test_f1)
```

[46]:
```
# res_mlp, names = list(), list()
# scores = evaluate_model(mlpa, trainX_scaled, Y)
# res_mlp.append(scores)
# names.append('MLP')
# print('>%s %.3f (%.3f)' % ('rf', mean(scores), std(scores)))
# #plot model performance for comparison
# pyplot.boxplot(res_mlp, labels=names, showmeans=True)
# pyplot.show()
# print(res_mlp)
```

[ ]:
```

```

[47]:
```
# from sklearn.tree import DecisionTreeClassifier

# sc = StandardScaler()
# #scaler = sc.fit(X_train)
# trainX_scaled = sc.fit_transform(X)
```

```
# dt = DecisionTreeClassifier(max_depth = 20,criterion='gini', random_state=␣
↪1234) # Define classifier
```

[ ]:

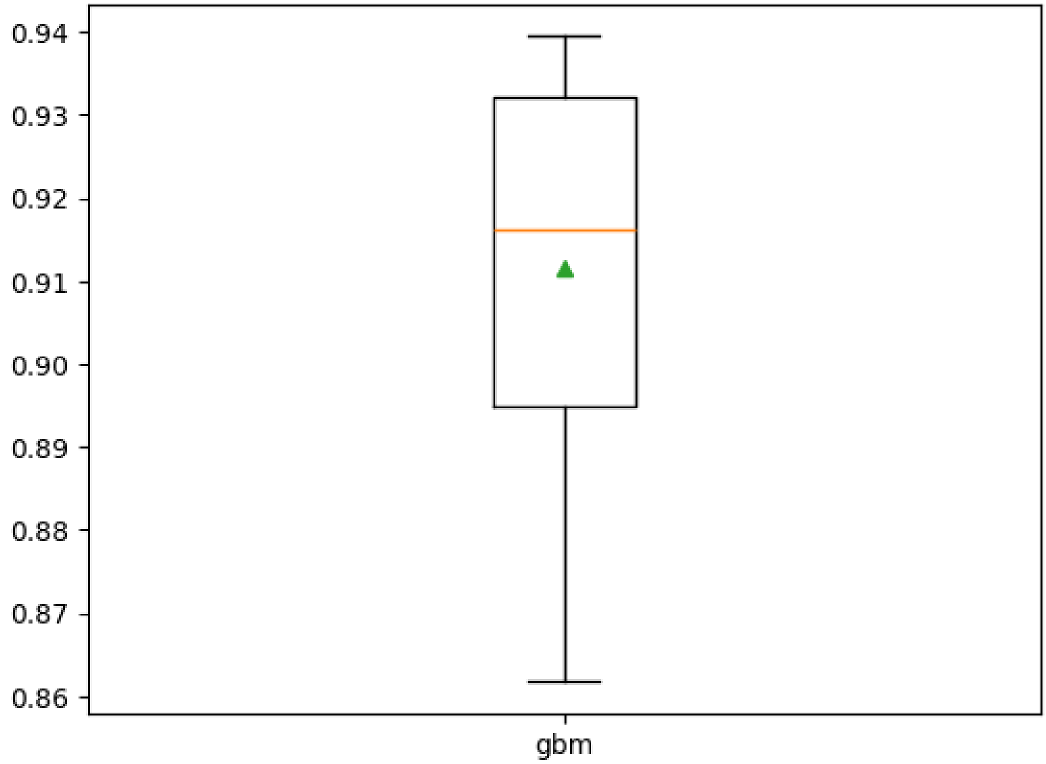
[48]:
```
# decision, names = list(), list()
# scores = evaluate_model(dt, X, Y)
# decision.append(scores)
# names.append('decision')
# print('>%s %.3f (%.3f)' % ('dt', mean(scores), std(scores)))
# #plot model performance for comparison
# pyplot.boxplot(decision, labels=names, showmeans=True)
# pyplot.show()
# print(decision)
```

[49]:
```
# Gradient Boosting Machine

gbm = GradientBoostingClassifier(learning_rate=0.1,
                                 n_estimators=100,
                                 max_depth = 10,
                                 subsample=0.80,
                                 random_state=1234,
                                )
```

[50]:
```
gbma, names = list(), list()
scores = evaluate_model(gbm, X, Y)
gbma.append(scores)
names.append('gbm')
print('>%s %.3f (%.3f)' % ('gbm', mean(scores), std(scores)))
#plot model performance for comparison
pyplot.boxplot(gbma, labels=names, showmeans=True)
pyplot.show()
print(gbma)
```

```
>gbm 0.912 (0.024)
```




```
[array([0.89048991, 0.86167147, 0.93371758, 0.93948127, 0.88472622,
       0.93659942, 0.92795389, 0.90778098, 0.91642651, 0.91618497])]
```

6 xgB

[52]:
```python
print(gbma)
```

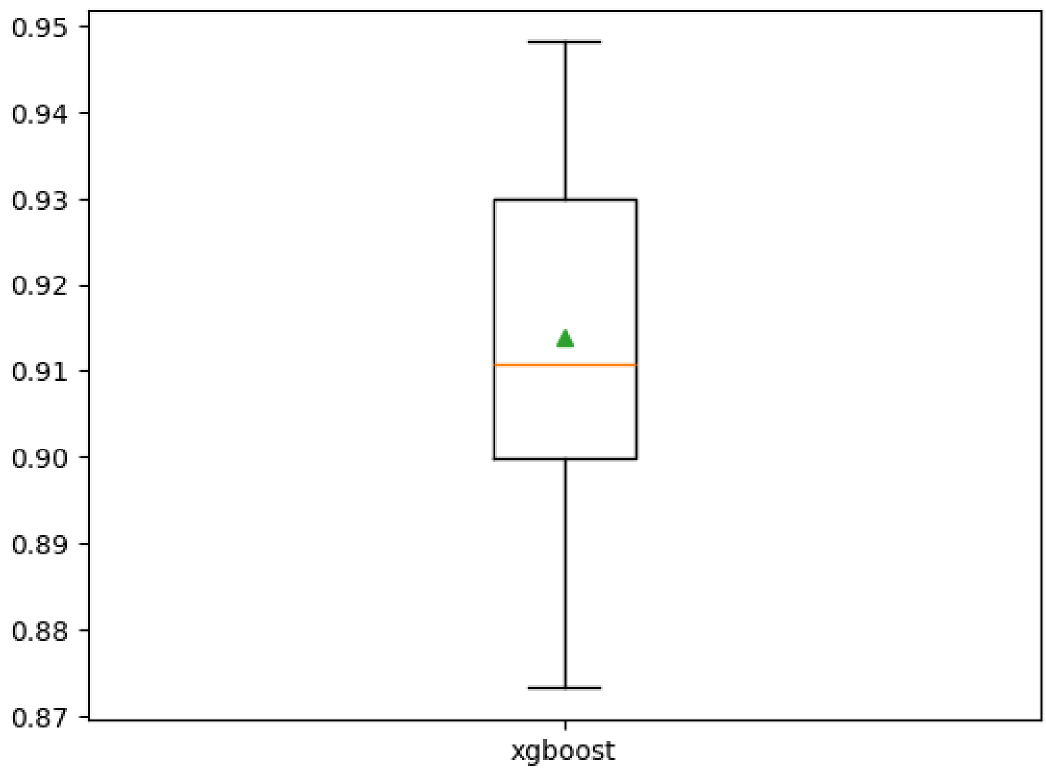
```
[array([0.89048991, 0.86167147, 0.93371758, 0.93948127, 0.88472622,
       0.93659942, 0.92795389, 0.90778098, 0.91642651, 0.91618497])]
```

[53]:
```python
# Extreme Gradient Boosting Machine
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
Y = le.fit_transform(Y)
from xgboost import XGBClassifier
xgb =  XGBClassifier(#num_class=7,
    n_estimators = 1500,
    learning_rate=0.010,
    num_iterations=1000,
   max_depth=50,
    #feature_fraction=0.80,
    #scale_pos_weight=1.5,
```

```
    booster='gbtree',
    #metric='multiclass',
    #val_metric='mlogloss',
    use_label_encoder=True,
    random_state = 1234)
```

[54]:
```
xgboost, names = list(), list()
scores = evaluate_model(xgb, X, Y)
xgboost.append(scores)
names.append('xgboost')
print('>%s %.3f (%.3f)' % ('xgboost', mean(scores), std(scores)))
#plot model performance for comparison
pyplot.boxplot(xgboost, labels=names, showmeans=True)
pyplot.show()
print(xgboost)
```

```
>xgboost 0.914 (0.022)
```




```
[array([0.90201729, 0.87319885, 0.92795389, 0.94236311, 0.89337176,
       0.9481268 , 0.91642651, 0.90489914, 0.89913545, 0.93063584])]
```

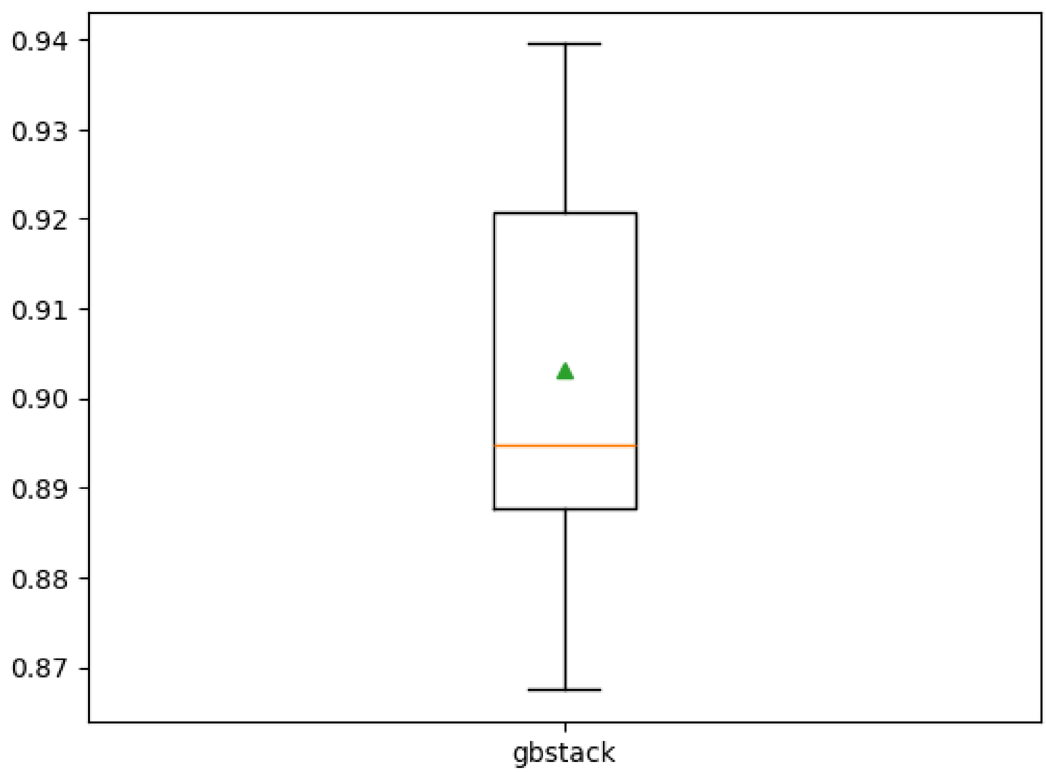
[55]:
```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
```

[56]:
```
estimator_list = [
    ('rf', rf),
    ('xgb', xgb),
    ('gbm', gbm)
]
```

[57]:
```
# Build Stack Model
stack_model = StackingClassifier(
    estimators = estimator_list, final_estimator =␣
 ↪GradientBoostingClassifier(),n_jobs = -1, cv = logo.get_n_splits(groups =␣
 ↪groups))
```

[58]:
```
gbstack, names = list(), list()
scores = evaluate_model(stack_model, X, Y)
gbstack.append(scores)
names.append('gbstack')
print('>%s %.3f (%.3f)' % ('gbstack', mean(scores), std(scores)))
#plot model performance for comparison
pyplot.boxplot(gbstack, labels=names, showmeans=True)
pyplot.show()
print(gbstack)
```

```
>gbstack 0.903 (0.022)
```

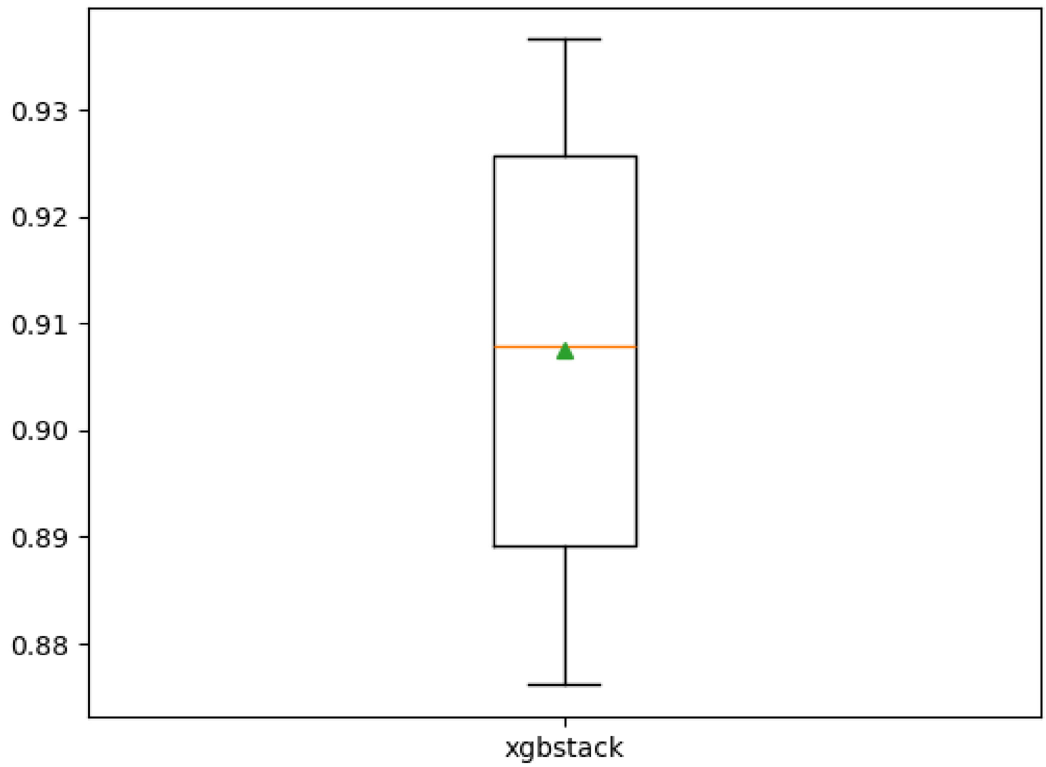


```
[array([0.88760807, 0.86743516, 0.9221902 , 0.93371758, 0.88760807,
       0.93948127, 0.89913545, 0.89048991, 0.88760807, 0.91618497])]
```

[59]:
```python
#- put it in the pending
stack_xgb = StackingClassifier(
estimators = estimator_list,
    final_estimator = XGBClassifier(#num_class=7,
    n_estimators = 1000,
    learning_rate = 0.010,
    num_iterations = 1000,
    max_depth=10,
    #feature_fraction=0.80,
    #scale_pos_weight=1.5,
    booster='gbtree',
    #metric='multiclass',
    #val_metric='mlogloss',
    use_label_encoder=True,
    random_state = 1234),cv = logo.get_n_splits(groups = groups),n_jobs=-1)
```

[60]:
```python
xgbstack, names = list(), list()
scores = evaluate_model(stack_xgb, X, Y)
xgbstack.append(scores)
names.append('xgbstack')
print('>%s %.3f (%.3f)' % ('xgbstack', mean(scores), std(scores)))
#plot model performance for comparison
pyplot.boxplot(xgbstack, labels=names, showmeans=True)
pyplot.show()
print(xgbstack)
```

```
>xgbstack 0.907 (0.021)
```




```
[array([0.88472622, 0.87608069, 0.92795389, 0.93659942, 0.89913545,
        0.93371758, 0.91642651, 0.88760807, 0.89337176, 0.91907514])]
```

[61]:
```
xgb_st = pd.DataFrame(np.transpose(xgbstack), columns = ['xbgStack'])
xgb_md = pd.DataFrame(np.transpose(xgboost), columns = ['xgbModel'])
gbm_md = pd.DataFrame(np.transpose(gbma),  columns = ['gbmModel'])
rf_md  = pd.DataFrame(np.transpose(results), columns = ['RandForModel'])

df = pd.concat([xgb_st, xgb_md,gbm_md, rf_md], axis = 1)
```

[64]:
```
print(df)
df.to_csv(r"e:/TomGreenSentinel_python/Variance_LL01.csv")
```

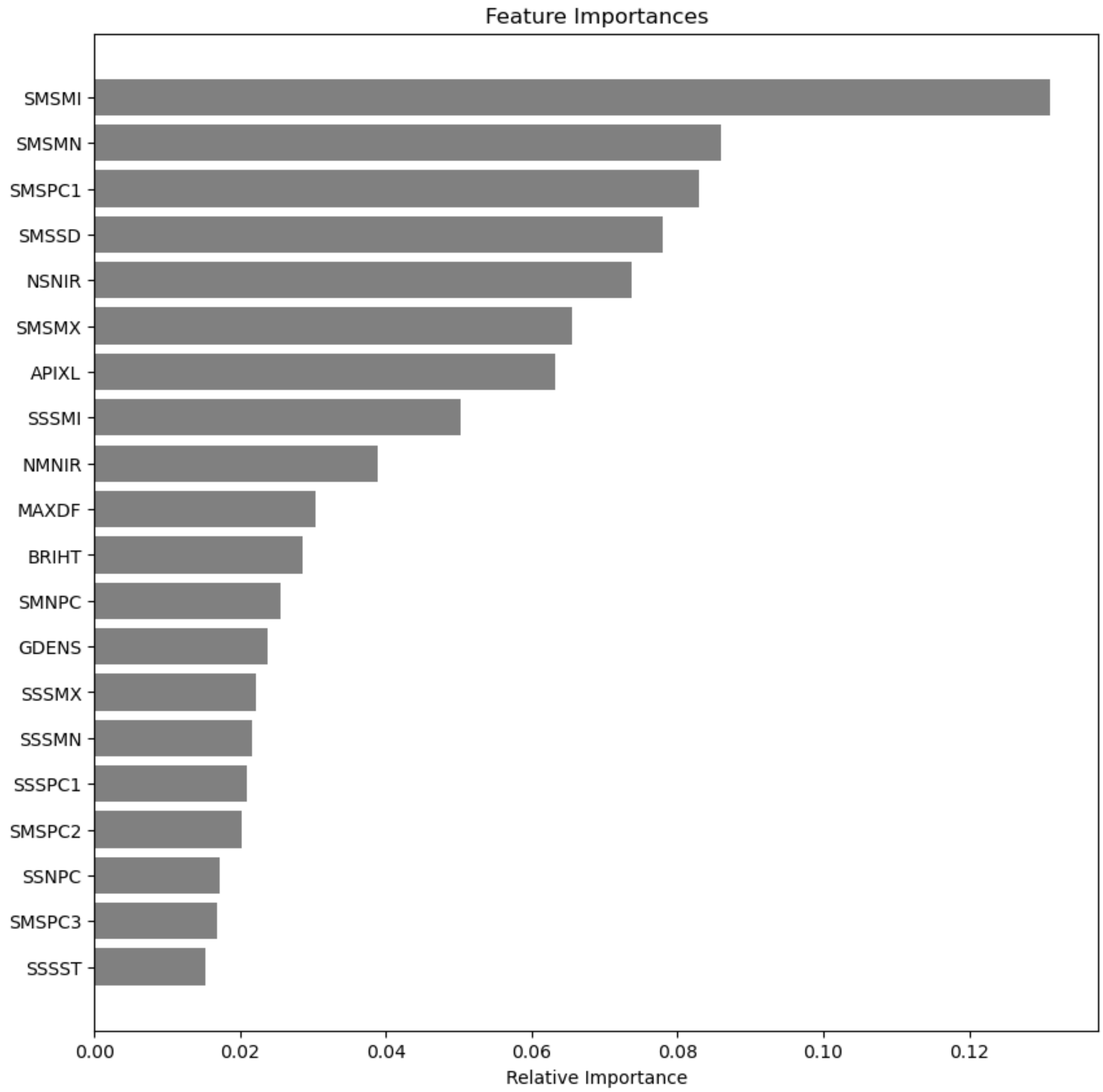
```
   xbgStack  xgbModel  gbmModel  RandForModel
0  0.884726  0.902017  0.890490      0.887608
1  0.876081  0.873199  0.861671      0.855908
2  0.927954  0.927954  0.933718      0.913545
3  0.936599  0.942363  0.939481      0.942363
4  0.899135  0.893372  0.884726      0.884726
5  0.933718  0.948127  0.936599      0.936599
6  0.916427  0.916427  0.927954      0.919308
7  0.887608  0.904899  0.907781      0.870317
8  0.893372  0.899135  0.916427      0.893372
```

```
9  0.919075  0.930636  0.916185      0.927746
```

[66]:
```
#  Importances
```

[68]:
```
rf_fit = rf.fit(X,Y)
features = X.columns
importances = rf_fit.feature_importances_
# summarize feature importance
num_features = 20
indices = np.argsort(importances)
plt.figure(figsize=(10,10))
plt.title('Feature Importances')

# only plot the customized number of features
plt.barh(range(num_features), importances[indices[-num_features:]],␣
 ↪color='grey', align='center')
plt.yticks(range(num_features), [features[i] for i in indices[-num_features:]])
plt.xlabel('Relative Importance')
plt.show()
```



[70]:
```
#--------- random forest
rf_results = rf.fit(X,Y)
# get importance
rf_importances = rf_fit.feature_importances_
rf_indices = np.argsort(rf_importances)
# summarize feature importance
num_features = 20

#------- gbm
gbm_results = gbm.fit(X,Y)
# get importance
gbm_importances = gbm_results.feature_importances_
gbm_indices = np.argsort(gbm_importances)
```

```
#----------- xgb
xgb_results = xgb.fit(X,Y) #scoring='accuracy'
# get importance
xgb_importances = xgb_results.feature_importances_
xgb_indices = np.argsort(xgb_importances)
```

```
C:\Users\suved\anaconda3\Lib\site-packages\xgboost\core.py:160: UserWarning:
[23:37:52] WARNING: C:\b\abs_0fh_d4x2ng\croot\xgboost-
split_1713973188995\work\cpp_src\src\learner.cc:742:
Parameters: { "num_iterations" } are not used.

  warnings.warn(smsg, UserWarning)
```

[71]:
```
#-------- stack
from sklearn.inspection import permutation_importance
xgbpred = stack_xgb.fit(X,Y)
stack_results = permutation_importance(xgbpred, X, Y) #scoring='accuracy'
# get importance
stack_importances = stack_results.importances_mean
stack_indices = np.argsort(stack_importances)
```

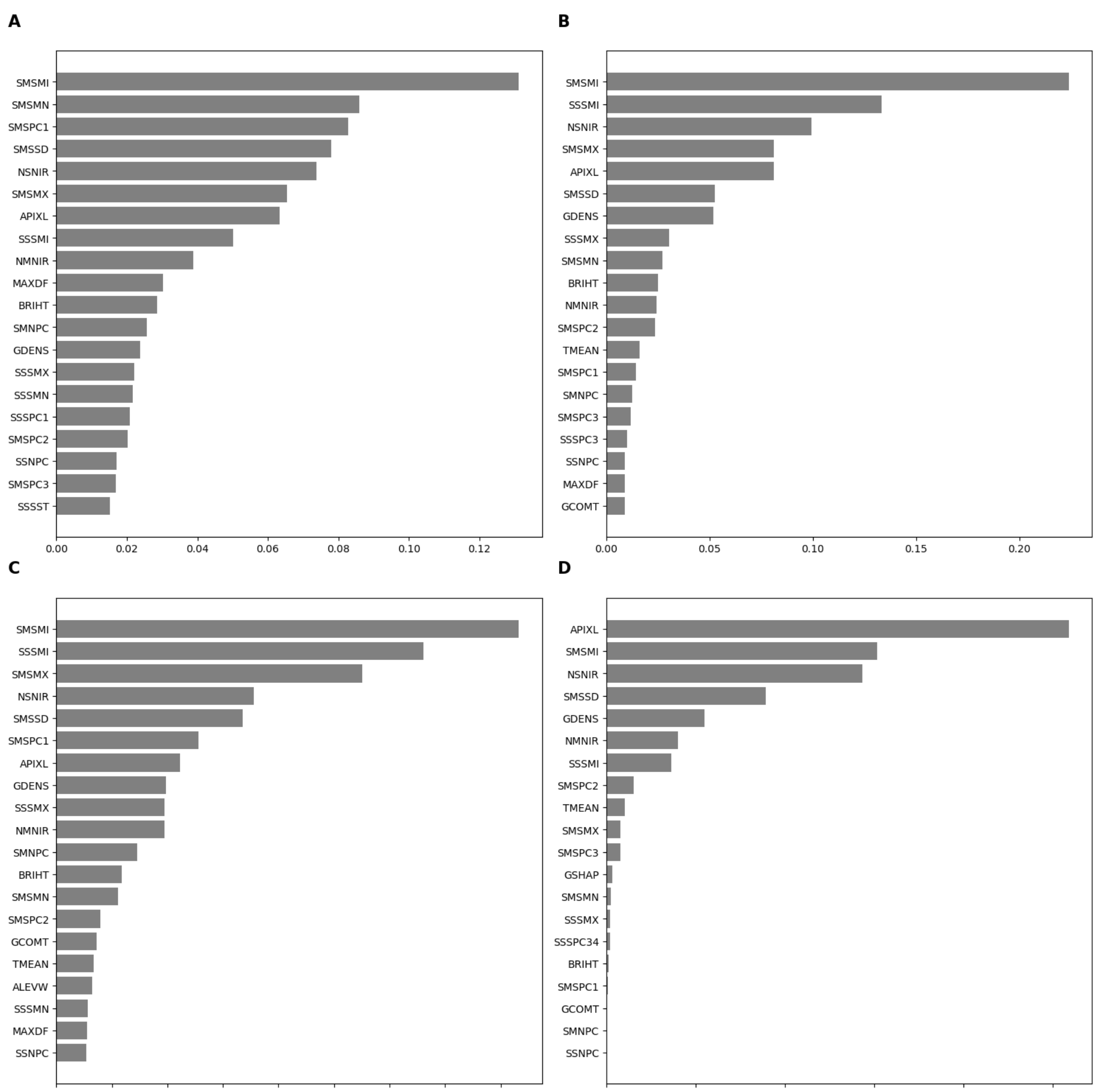
```
C:\Users\suved\anaconda3\Lib\site-packages\xgboost\core.py:160: UserWarning:
[23:43:27] WARNING: C:\b\abs_0fh_d4x2ng\croot\xgboost-
split_1713973188995\work\cpp_src\src\learner.cc:742:
Parameters: { "num_iterations" } are not used.

  warnings.warn(smsg, UserWarning)
```

[72]:
```
import string
fig, axs = plt.subplots(2,2,figsize=(15,15))
plt.sca(axs[0, 0])
plt.yticks(range(num_features), [features[i] for i in rf_indices[-num_features:
 ↪]])
axs[0,0].barh(range(num_features), rf_importances[rf_indices[-num_features:]],␣
 ↪color='grey', align='center')
plt.sca(axs[0,1])
plt.yticks(range(num_features), [features[i] for i in gbm_indices[-num_features:
 ↪]])
axs[0,1].barh(range(num_features), gbm_importances[gbm_indices[-num_features:
 ↪]], color='grey', align='center')
plt.sca(axs[1,0])
plt.yticks(range(num_features), [features[i] for i in xgb_indices[-num_features:
 ↪]])
axs[1,0].barh(range(num_features), xgb_importances[xgb_indices[-num_features:
 ↪]], color='grey', align='center')
plt.sca(axs[1,1])
```

```
plt.yticks(range(num_features), [features[i] for i in stack_indices[-num_features:]])
axs[1,1].barh(range(num_features), stack_importances[stack_indices[-num_features:]], color='grey', align='center')
axs = axs.flat
for n, ax in enumerate(axs):
    ax.text(-0.1, 1.05, string.ascii_uppercase[n], transform=ax.transAxes,
            size=16, weight='bold')
fig.tight_layout()

plt.savefig("FeatureImportance_LL01.png", format="png", dpi = 1200, bbox_inches = "tight")
plt.show()
```



[73]:
```
from dask import dataframe as dd

allData = pd.read_csv(r"e:/TomGreenSentinel_python/AllTilesClipSpatial.csv")
```

[74]:
```
allDatac = allData[[
 'ALEVW',
 'APIXL',
 'BRIHT',
 'GASYM',
 'GCOMT',
 'GDENS',
 'GRECT',
 'GROND',
 'GSHAP',
 'MAXDF',
 'NMNIR',
 'NSNIR',
 'SMNPC',
 'SMSMI',
 'SMSMN',
 'SMSMX',
 'SMSPC1',
 'SMSPC2',
 'SMSPC3',
 'SMSSD',
 'SSNPC',
 'SSSMI',
 'SSSMN',
 'SSSMX',
 'SSSPC1',
 'SSSPC3',
 'SSSPC34',
 'SSSST',
 'TMEAN']]
```

[75]:
```
#allDatac = allData.dropna()
#allDatac.info()

rfpred = rf_results.predict(allDatac)
```

[76]:
```
gbmpred = gbm_results.predict(allDatac)
```

[77]:
```
names = xgb_results.get_booster().feature_names
print(names)
```

```
['ALEVW', 'APIXL', 'BRIHT', 'GASYM', 'GCOMT', 'GDENS', 'GRECT', 'GROND',
'GSHAP', 'MAXDF', 'NMNIR', 'NSNIR', 'SMNPC', 'SMSMI', 'SMSMN', 'SMSMX',
'SMSPC1', 'SMSPC2', 'SMSPC3', 'SMSSD', 'SSNPC', 'SSSMI', 'SSSMN', 'SSSMX',
'SSSPC1', 'SSSPC3', 'SSSPC34', 'SSSST', 'TMEAN']
```

[78]:
```
allDatac = allDatac[names]
```

[79]:
```
xgbpred = rf_results.predict(allDatac)
```

[80]:
```
stack_res = stack_xgb.fit(X,Y) #scoring='accuracy'

stacPred = stack_res.predict(allDatac)
```

```
C:\Users\suved\anaconda3\Lib\site-packages\xgboost\core.py:160: UserWarning:
[23:54:34] WARNING: C:\b\abs_0fh_d4x2ng\croot\xgboost-
split_1713973188995\work\cpp_src\src\learner.cc:742:
Parameters: { "num_iterations" } are not used.

  warnings.warn(smsg, UserWarning)
```

[81]:
```
allData.head()
```

[81]:
```
   OBJECTID  Join_Count  TARGET_FID     ALEVW    APIXL         BRIHT  \
0         1           5           1  1.096132   4322.0  4.188364e+03
1         2           3           2  1.258065   2648.0  4.211632e+03
2         3           3           3  1.052632    261.0  4.148224e+03
3         4           3           4  3.932133  16451.0 -9.859658e+30
4         5           2           5  1.811964    366.0  4.085605e+03

      GASYM     GCOMT     GDENS     GRECT  …   SNDVI_1       SNIR  \
0  0.261222  1.826121  2.140321  0.817912  …  0.068026   8.651500
1  0.331462  1.826284  2.265649  0.841116  …  0.092530   9.304941
2  0.145700  1.455939  2.132749  0.885941  …  0.070055   6.106334
3  0.933833  1.720202  1.357548  0.786174  …  0.068026   8.651500
4  0.716758  2.150927  1.640183  0.723847  …  0.101478  13.593320

        SPC1       SPC2      SPC3       SRED      SSTD     SNDSI  Shape_Leng  \
0  17.107325   8.962758  3.958753  12.872898  0.883999  0.045227       320.4
1  23.005904  11.856336  5.046224  18.020789  1.024682  0.061560       196.8
2  23.876816  11.792100  4.297005  18.550592  0.864930  0.056497        48.0
3  17.107325   8.962758  3.958753  12.872898  0.883999  0.045227       414.4
4  33.045175  11.731357  4.628132  22.630674  1.492548  0.062753        76.8

   Shape_Area
0     1555.92
1      953.28
2       93.96
3     2534.88
```

```
4      131.76

[5 rows x 76 columns]
```

[82]:
```
import pandas as pd
Allpd = pd.DataFrame()
Allpd['oldOid'] = allData['OBJECTID']
```

[83]:
```
allData['rf_pred5llo']    = rfpred
allData['gbm_pred5llo']    = gbmpred
allData['xgb_pred5llo']    = xgbpred
allData['stack_pred5llo'] = stacPred
```

[84]:
```
allData.to_csv('PredictedData5_110_1.csv')
allData.size
```

[84]:
```
212006640
```

[85]:
```
# save the model to disk
import pickle
rfmod = 'rf_model.sav'
pickle.dump(rf_results, open(rfmod, 'wb'))
```

[86]:
```
gbmmod = 'gbm_model.sav'
pickle.dump(gbm_results, open(gbmmod, 'wb'))
```

[87]:
```
xgbmod = 'xgb_model.sav'
pickle.dump(xgb_results, open(xgbmod, 'wb'))
#------------ stack
stackmod = 'stack_model.sav'
pickle.dump(xgbpred, open(stackmod, 'wb'))
```

[88]:
```
# load the model from disk
loadRF = pickle.load(open('rf_model.sav', 'rb'))
result = loadRF.score(X, Y)
print(result)
```

```
0.9769385990198904
```