Bagging-final-model

December 15, 2020

[1]:
```
import sys
sys.path.append("../")
import pandas as pd
from ortho_lib3 import *
import os
import matplotlib.pyplot as plt
import numpy as np
import math
```

[2]:
```
exercises = Exercises.load("..//Pickle/
↪sliced_transformed_exercises_9_12_all_categories.pickle")
#exercises = exercises.drop_category(1)
exercises.df
```

[2]:

	angle_left_shoulder_xz_max_AF	angle_left_shoulder_xz_max_RF	\
0	2.705811	1.200405	
1	2.757520	1.381570	
2	2.691818	1.372206	
3	2.524043	0.687781	
4	2.640875	1.189572	
..	…	…	
115	1.201943	0.428302	
116	2.186121	0.580735	
117	1.882206	0.808175	
118	1.237203	0.822418	
119	2.602110	1.114584	

	angle_right_shoulder_xz_max_AF	angle_right_shoulder_xz_max_RF	\
0	2.880950	1.022868	
1	2.693907	1.300333	
2	2.597640	1.352112	
3	2.581135	0.931032	
4	2.606757	1.312657	
..	…	…	
115	2.347702	0.429440	
116	2.082403	0.712122	
117	2.064096	0.987518	

```
118                        2.171692                         0.780059
119                        2.614965                         1.054871

     angle_left_shoulder_yz_max_AB  angle_right_shoulder_yz_max_AB  \
0                         2.536825                         2.785053
1                         2.729818                         2.894459
2                         2.312296                         2.432454
3                         2.606495                         2.639872
4                         2.666926                         2.706126
..                             …                                …
115                       0.897281                         2.104353
116                       2.649783                         2.220724
117                       2.459425                         2.348511
118                       1.519678                         2.371818
119                       2.574107                         2.385383

     diff_x_wrist_std_EL  diff_x_wrist_std_AF  diff_x_wrist_std_RF  \
0               0.051140             0.045686             0.047964
1               0.046985             0.032754             0.022710
2               0.018752             0.043444             0.059608
3               0.045662             0.032816             0.036892
4               0.042752             0.048538             0.042671
..                   …                    …                    …
115             0.131204             0.461456             0.041716
116             0.022393             0.089273             0.157484
117             0.055195             0.085915             0.043581
118             0.113686             0.182995             0.066054
119             0.037779             0.128872             0.047050

     diff_x_elbow_std_EL  …  angular_acc_xz_elbow_r_mean_AF  \
0               0.066650  …                        0.013454
1               0.035020  …                        0.009875
2               0.035862  …                        0.012321
3               0.030882  …                        0.008630
4               0.015893  …                        0.007462
..                   …  …                             …
115             0.038488  …                        0.030021
116             0.014435  …                        0.036877
117             0.025565  …                        0.026531
118             0.041324  …                        0.055863
119             0.029085  …                        0.032568

     angular_acc_xz_elbow_r_std_AF  angular_acc_xz_elbow_r_mean_RF  \
0                         0.009997                         0.009142
1                         0.007717                         0.009636
2                         0.009420                         0.010026
3                         0.007913                         0.007300
```

```
4                          0.006183                          0.007197
..                              …                                 …
115                        0.039972                          0.009998
116                        0.036002                          0.044232
117                        0.030848                          0.030147
118                        0.051294                          0.031281
119                        0.019488                          0.023850

     angular_acc_xz_elbow_r_std_RF  angular_vel_yz_elbow_l_std_AB  \
0                          0.008593                          0.035317
1                          0.008744                          0.017855
2                          0.008610                          0.028611
3                          0.007061                          0.021368
4                          0.006843                          0.026717
..                              …                                 …
115                        0.007187                          0.013514
116                        0.048294                          0.066856
117                        0.026745                          0.042285
118                        0.030357                          0.047952
119                        0.021694                          0.057128

     angular_vel_yz_elbow_r_std_AB  angular_acc_yz_elbow_l_mean_AB  \
0                          0.034019                          0.010569
1                          0.019760                          0.009327
2                          0.027344                          0.009416
3                          0.021825                          0.006564
4                          0.023068                          0.007782
..                              …                                 …
115                        0.040844                          0.009843
116                        0.062815                          0.036638
117                        0.048514                          0.019478
118                        0.058858                          0.028998
119                        0.066880                          0.029559

     angular_acc_yz_elbow_l_std_AB  angular_acc_yz_elbow_r_mean_AB  \
0                          0.008540                          0.009179
1                          0.008234                          0.008376
2                          0.007159                          0.008765
3                          0.007230                          0.007175
4                          0.011197                          0.007356
..                              …                                 …
115                        0.008446                          0.017797
116                        0.035517                          0.042551
117                        0.018303                          0.019738
118                        0.029493                          0.032410
119                        0.022851                          0.030719
```

```
     angular_acc_yz_elbow_r_std_AB
0                         0.008611
1                         0.006633
2                         0.007664
3                         0.007337
4                         0.005814
..                             …
115                       0.019753
116                       0.033384
117                       0.018876
118                       0.037222
119                       0.026552

[120 rows x 78 columns]
```

[3]:
```
exp = Experiment(exercises, y_condition= lambda y: np.all([y != 'Category_1'],␣
 ↪axis=0))
print(exp.df.shape)
columns = exp.df.columns.to_numpy()
```

```
(120, 78)
```

[4]:
```
# check scikit-learn version
import sklearn
print(sklearn.__version__)
```

```
0.23.2
```

[5]:
```
from sklearn.datasets import make_regression
from sklearn.model_selection import cross_val_score, KFold, StratifiedKFold
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import classification_report
# define dataset
X = exp.df.values
y = exp.y
print(X)
```

```
[[2.70581125 1.2004053  2.88095043 … 0.00854023 0.0091792  0.00861107]
 [2.75751986 1.38156992 2.69390747 … 0.00823437 0.00837636 0.00663325]
 [2.69181792 1.37220563 2.59764029 … 0.00715904 0.00876495 0.00766446]
 …
 [1.88220638 0.80817506 2.06409626 … 0.01830264 0.01973769 0.01887624]
 [1.23720302 0.82241828 2.17169177 … 0.02949289 0.03240998 0.03722172]
 [2.60211036 1.11458353 2.61496538 … 0.02285141 0.03071891 0.0265518 ]]
```

[6]:
```
skf = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)

df_scores = pd.DataFrame()

precision_list = []
recall_list = []
f1_score_list = []
accuracy_list=[]
i=1
dtr = DecisionTreeRegressor()

for train_index, test_index in skf.split(X, y):
#     print("TRAIN:", train_index, "TEST:", test_index)
    Xtrain, Xtest = X[train_index], X[test_index]
    ytrain, ytest = y[train_index], y[test_index]
    regr = BaggingRegressor(dtr, n_estimators=100, max_samples=0.5)
    regr.fit(Xtrain, ytrain)
    ypred = regr.predict(Xtest)
    report = classification_report(ytest, ypred.round(), output_dict=True)
    recall = (report.get('weighted avg').get('recall'))
    precision = (report.get('weighted avg').get('precision'))
    f1_score = (report.get('weighted avg').get('f1-score'))
    accuracy = (report.get('accuracy'))

    precision_list.append(precision)
    recall_list.append(recall)
    f1_score_list.append(f1_score)
    accuracy_list.append(accuracy)

    feature_importances = np.mean([tree.feature_importances_ for tree in regr.
 ↪estimators_], axis=0)
    #feature_scores = pd.Series(regr.feature_importances_, index=exp.df.
 ↪columns).sort_values(ascending=False)
    #feature_scores = pd.Series(dtr.feature_importances_, index=exp.df.columns)
    feature_scores = pd.Series(feature_importances, index=exp.df.columns)
    df_scores['column' + str(i)] = feature_scores.values
    i=i+1


df_scores['average'] = df_scores.mean(axis=1)
df_scores['feature'] = exp.df.columns
print(f'precision: {np.mean(precision_list)} \nrecall: {np.mean(recall_list)}␣
 ↪\nf1_score: {np.mean(f1_score_list)} \naccuracy: {np.mean(accuracy_list)}')
```

```
precision: 0.9036934824434824
recall: 0.8916666666666666
f1_score: 0.8852599559916634
```

```
accuracy: 0.8916666666666666
```

[7]:
```
df_scores[['feature', 'average']].sort_values(by='average', ascending=False).
↪head(10).reset_index(drop=True)
```

[7]:

	feature	average
0	z_elbow_max_AF	0.139699
1	z_wrist_max_AB	0.102989
2	z_wrist_max_AF	0.093109
3	z_elbow_max_AB	0.068611
4	diff_y_wrist_std_AB	0.050212
5	x_wrist_max_EL	0.049581
6	diff_z_wrist_std_AB	0.029628
7	diff_x_shoulder_std_EL	0.027127
8	angle_left_shoulder_yz_max_AB	0.022151
9	acc_wrists_x_r_std_EL	0.018211

1 Reports

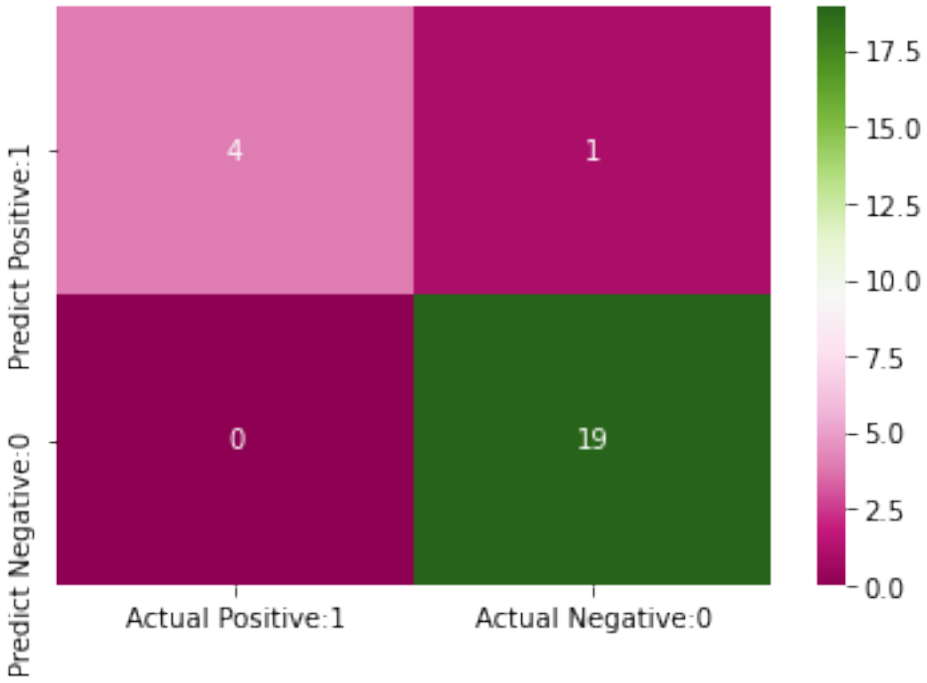
[8]:
```
report = classification_report(ytest, ypred.round())
print(report)
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.80      0.89         5
         1.0       0.95      1.00      0.97        19

    accuracy                           0.96        24
   macro avg       0.97      0.90      0.93        24
weighted avg       0.96      0.96      0.96        24
```

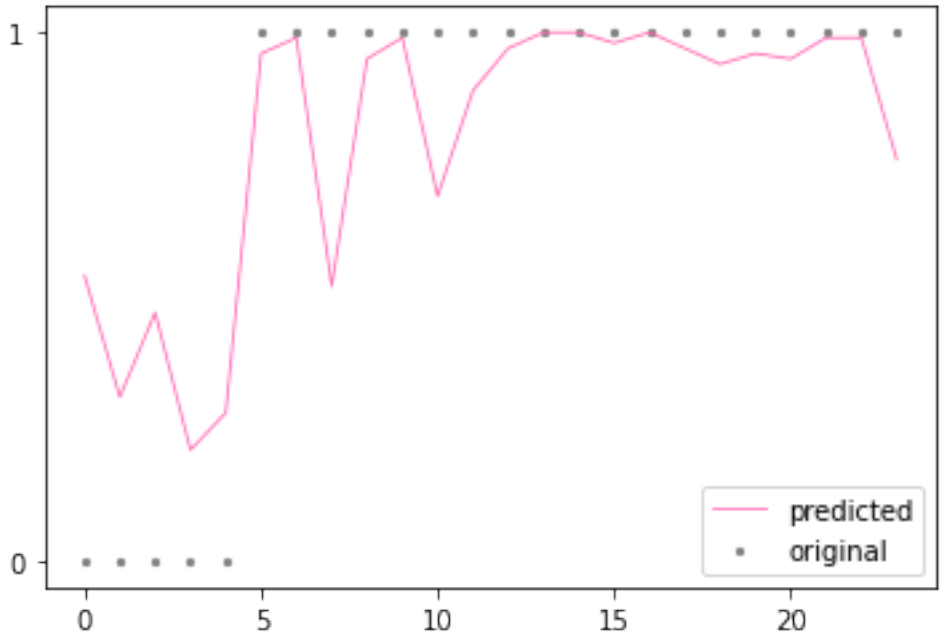
2 Confusion Matrix

[9]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ytest, ypred.round())
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual␣
↪Negative:0'],
                                  index=['Predict Positive:1', 'Predict Negative:
↪0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='PiYG')
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f04d2a90e80>
```



[10]:
```
x_ax = range(len(ytest))
plt.scatter(x_ax, ytest, s=5, color="grey", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="hotpink", label="predicted")
plt.yticks([0,1])
plt.legend()
plt.show()
```



3 Bomen printen

[11]:
```
estimators = regr.estimators_
len(estimators)
```

[11]: 100

[12]:
```
#for bomen in estimators:
 #   print (bomen)
```

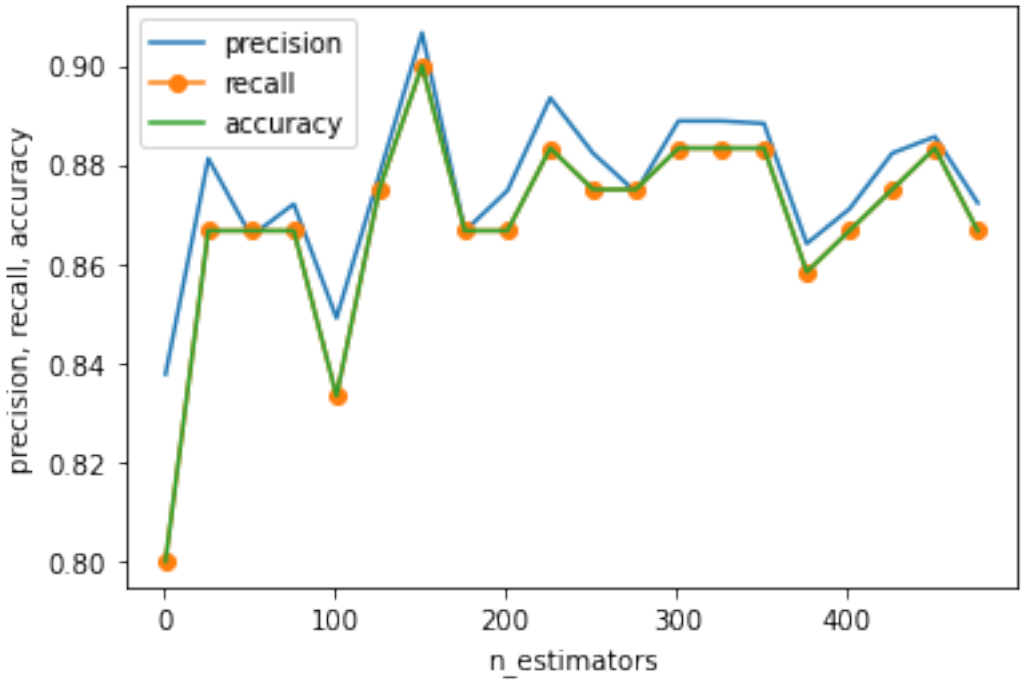
[13]:
```
#from sklearn import tree
#from matplotlib import pyplot as plt
#for bomen in estimators[5:]:
    #boom = bomen
    #boom_score = boom.score(X,y)
    #fig, axes = plt.subplots(nrows =1,ncols = 1,figsize = (5,5), dpi=300)
    #tree.plot_tree(boom, filled=True, feature_names= exercises.df.columns)
```

4 Hyper parameter tuning

5 n_estimators

[14]:
```
skf = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)

mean_precision_list = []
mean_recall_list = []
mean_accuracy_list=[]

n_estimators_list = [k for k in range(1,500,25)]

for n_estimators in n_estimators_list:
    precision_list = []
    recall_list = []
    accuracy_list=[]
    for train_index, test_index in skf.split(X, y):
        Xtrain, Xtest = X[train_index], X[test_index]
        ytrain, ytest = y[train_index], y[test_index]
        regr = BaggingRegressor(n_estimators=n_estimators, max_samples= 0.3)
        regr.fit(Xtrain, ytrain)
        ypred = regr.predict(Xtest)
        report = classification_report(ytest, ypred.round(), output_dict=True)
        recall = (report.get('weighted avg').get('recall'))
        precision = (report.get('weighted avg').get('precision'))
        accuracy = (report.get('accuracy'))

        precision_list.append(precision)
        recall_list.append(recall)
        accuracy_list.append(accuracy)

    mean_precision_list.append(np.mean(precision_list))
    mean_recall_list.append(np.mean(recall_list))
    mean_accuracy_list.append(np.mean(accuracy_list))
```

[15]:
```
plt.plot(n_estimators_list, mean_precision_list, label='precision')
plt.plot(n_estimators_list, mean_recall_list, label='recall', marker = 'o')
plt.plot(n_estimators_list, mean_accuracy_list, label='accuracy')
plt.xlabel('n_estimators')
plt.ylabel('precision, recall, accuracy')
plt.legend()
plt.show()
```



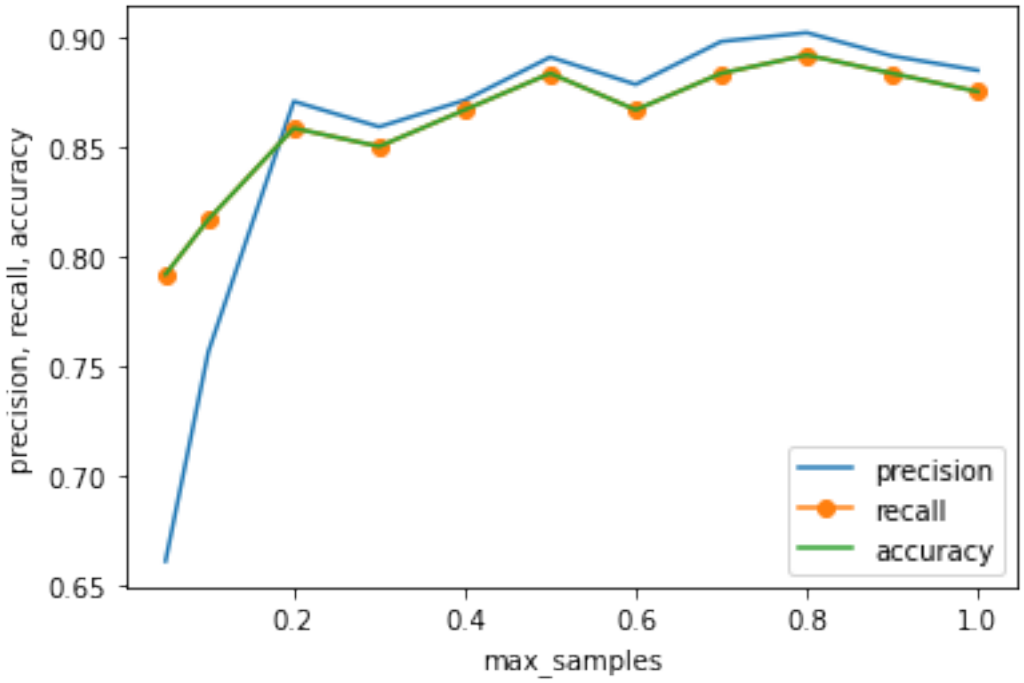
6 max_samples

[16]:
```
skf = StratifiedKFold(n_splits=5, random_state=None, shuffle=False)

mean_precision_list = []
mean_recall_list = []
mean_accuracy_list=[]

max_samples_list = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]

for max_samples in max_samples_list:
    precision_list = []
    recall_list = []
    accuracy_list=[]
    for train_index, test_index in skf.split(X, y):
        Xtrain, Xtest = X[train_index], X[test_index]
        ytrain, ytest = y[train_index], y[test_index]
        regr = BaggingRegressor(n_estimators=175, max_samples = max_samples)
        regr.fit(Xtrain, ytrain)
        ypred = regr.predict(Xtest)
        report = classification_report(ytest, ypred.round(), output_dict=True)
        recall = (report.get('weighted avg').get('recall'))
        precision = (report.get('weighted avg').get('precision'))
```

```
        accuracy = (report.get('accuracy'))

        precision_list.append(precision)
        recall_list.append(recall)
        accuracy_list.append(accuracy)

    mean_precision_list.append(np.mean(precision_list))
    mean_recall_list.append(np.mean(recall_list))
    mean_accuracy_list.append(np.mean(accuracy_list))
```

```
/opt/jupyterhub/anaconda/lib/python3.6/site-
packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

/opt/jupyterhub/anaconda/lib/python3.6/site-
packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

/opt/jupyterhub/anaconda/lib/python3.6/site-
packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

/opt/jupyterhub/anaconda/lib/python3.6/site-
packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

/opt/jupyterhub/anaconda/lib/python3.6/site-
packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

/opt/jupyterhub/anaconda/lib/python3.6/site-
packages/sklearn/metrics/_classification.py:1221: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

[17]:
```
plt.plot(max_samples_list, mean_precision_list, label='precision')
plt.plot(max_samples_list, mean_recall_list, label='recall', marker= 'o')
plt.plot(max_samples_list, mean_accuracy_list, label='accuracy')
plt.xlabel('max_samples')
plt.ylabel('precision, recall, accuracy')
plt.legend()
plt.show()
```



[ ]: