In [1]:
```
import sys
sys.path.append("../")
import pandas as pd
from ortho_lib3_Copy2 import *
import os
import matplotlib.pyplot as plt
import numpy as np
import math
```

Loading data using pickle

In [2]:
```
exercises = Exercises.load('..//Pickle/sliced_transformed_exercises_9_12_all_categories.pickle')
#exercises = exercises.drop_category(1)
```

In [3]:
```
#create experiment
exp = Experiment(exercises, y_condition= lambda y: np.all([y != 'Category_1'], axis=0))
exp.df
columns = exp.df.columns.to_numpy()

exp
```

Out[3]:
```
<ortho_lib3_Copy2.Experiment at 0x7fc4c1a98780>
```

In [4]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import LeaveOneOut
from pprint import pprint
```

Research hyperparameters

In [5]:
```
#Make X and y

X = exp.df.values
y = exp.y
X
```

Out[5]:
```
array([[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 ]])
```

In [7]:
```
from sklearn.model_selection import RandomizedSearchCV

rfc2 = RandomForestClassifier()

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 2000, num = 100)]

# Number of features to consider at every split
max_features = ['auto', 'sqrt', 'log2']

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 10)]
max_depth.append(None)

# Minimum number of samples required to split a node
min_samples_split = [int(x) for x in np.linspace(start = 1, stop = 41, num =20)]

# Minimum number of samples required at each leaf node
min_samples_leaf = [int(x) for x in np.linspace(start = 1, stop = 11, num = 10)]

# Method of selecting samples for training each tree
bootstrap = [True, False]

# Criterion "gini", "entropy"
criterion = ['gini', 'entropy']

#min_impurity_decrease
min_impurity_decrease = [int(x) for x in np.linspace(start = 0.0, stop = 5, num = 10)]

#oob_score. Using out of bag samples
oob_score = [False, True]

#Set verbose. Controls verbosity when fitting and predicting
verbose = [0, 1, 2, 3]

#Warm_start: Reuse solution of previous call
warm_start = [False, True]

#ccp_alpha complexity parameter
ccp_alpha = [int(x) for x in np.linspace(start = 0.0, stop = 5, num = 10)]

n_jobs = [1, -1]
n_jobs.append(None)


# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap,
               'criterion': criterion,
               'min_impurity_decrease' : min_impurity_decrease,
               'n_jobs' : n_jobs,
               'verbose' : verbose,
               'warm_start' : warm_start,
               'ccp_alpha' : ccp_alpha,
               'oob_score' : oob_score}
```

```
rf_random = RandomizedSearchCV(estimator = rfc2, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, n_jobs = -1)

#Fit the random search model
rf_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 48 concurrent workers.
[Parallel(n_jobs=-1)]: Done  66 tasks      | elapsed:    4.5s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:   16.0s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s

building tree 1 of 1115
building tree 2 of 1115
building tree 3 of 1115
building tree 4 of 1115
building tree 5 of 1115
building tree 6 of 1115
building tree 7 of 1115
building tree 8 of 1115
building tree 9 of 1115
building tree 10 of 1115
building tree 11 of 1115
```

```
In [9]: rf_random.best_params_

Out[9]: {'warm_start': True,
         'verbose': 3,
         'oob_score': False,
         'n_jobs': 1,
         'n_estimators': 1115,
         'min_samples_split': 9,
         'min_samples_leaf': 1,
         'min_impurity_decrease': 0,
         'max_features': 'sqrt',
         'max_depth': 98,
         'criterion': 'gini',
         'ccp_alpha': 0,
         'bootstrap': False}
```

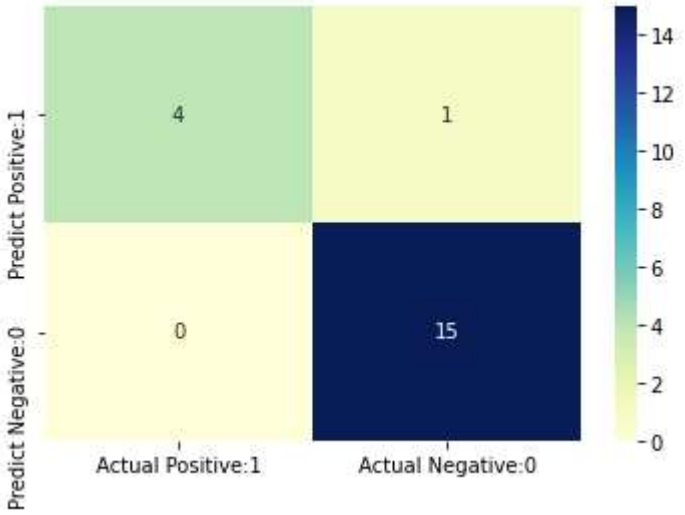
Applying best hyperparamers

In [10]:
```
rfc = RandomForestClassifier(n_estimators = 190,
 min_samples_split = 11,
 min_samples_leaf=4,
 max_features= 'sqrt',
 max_depth= None,
 bootstrap= False,
 min_impurity_decrease = 0,
 criterion= 'entropy',
 ccp_alpha = 0,
 n_jobs = -1,
 verbose = 1,
 oob_score = False)
```

In [11]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report




df_scores = pd.DataFrame()
i=1

featuresdf = pd.DataFrame()

for train_index, test_index in skf.split(X, y):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    rfc.fit(X_train, y_train)
    y_pred = rfc.predict(X_test)
    print('Model accuracy score with 190 decision-trees : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
    print(classification_report(y_test, y_pred))

    plt.subplots()
    #implement confusion matrix.
    cm = confusion_matrix(y_test, y_pred)
    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='YlGnBu')

    feature_scores = pd.Series(rfc.feature_importances_, index=exp.df.columns).sort_values(ascending=False)

    feature_scores = pd.Series(rfc.feature_importances_, index=exp.df.columns)
    df_scores['column' + str(i)] = feature_scores.values
    i=i+1

df_scores['feature'] = exp.df.columns
df_scores['gemiddeld'] = df_scores.mean(axis=1)
```

Output bovenstaande code:

```
Model accuracy score with 1115 decision-trees : 0.9500
              precision    recall  f1-score   support

         0.0       1.00      0.80      0.89         5
         1.0       0.94      1.00      0.97        15

    accuracy                           0.95        20
   macro avg       0.97      0.90      0.93        20
weighted avg       0.95      0.95      0.95        20
```



Feature importance

In [13]:
```
pd.set_option('display.max_rows', None)

df_scores.sort_values(axis=0, by='gemiddeld', ascending=False)
```

Out[13]:

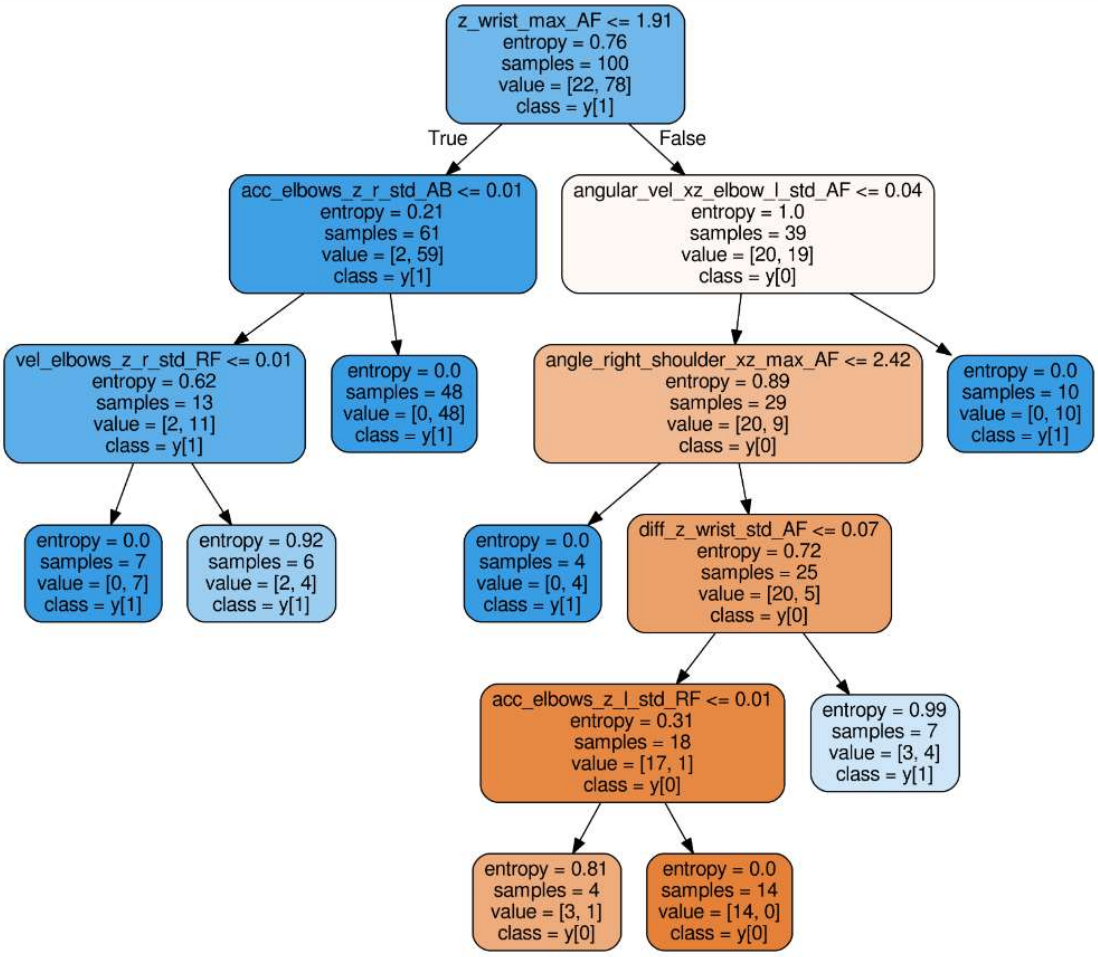
	column1	column2	column3	column4	column5	column6	feature	gemiddeld
32	0.125721	0.090908	0.079888	0.079448	0.081933	0.083592	z_wrist_max_AB	0.090248
30	0.062616	0.086533	0.103528	0.069340	0.062542	0.075371	z_elbow_max_AF	0.076655
33	0.097880	0.067243	0.050207	0.039065	0.063323	0.055184	z_wrist_max_AF	0.062150
18	0.039626	0.065121	0.041691	0.048663	0.071551	0.053793	diff_y_wrist_std_AB	0.053408
60	0.030193	0.036288	0.033717	0.022520	0.050677	0.066681	angular_vel_xz_elbow_l_std_AF	0.040013
29	0.037917	0.037900	0.032101	0.046724	0.043017	0.042086	z_elbow_max_AB	0.039957
62	0.021271	0.047258	0.043012	0.032180	0.056581	0.032171	angular_vel_xz_elbow_r_std_AF	0.038745
0	0.025437	0.028207	0.029949	0.067079	0.012691	0.017577	angle_left_shoulder_xz_max_AF	0.030157
63	0.010066	0.011678	0.059559	0.016530	0.015324	0.028505	angular_vel_xz_elbow_r_std_RF	0.023610
34	0.024794	0.021708	0.031797	0.015775	0.019939	0.020963	z_wrist_max_RF	0.022496
73	0.014648	0.016122	0.025415	0.036711	0.016991	0.016455	angular_vel_yz_elbow_r_std_AB	0.021057

Boom printen:

```
# Extract single tree
estimator = rfc.estimators_[100]

from sklearn.tree import export_graphviz
# Export as dot file
export_graphviz(estimator, out_file='treeRF.dot',
                feature_names = exercises.df.columns,
                class_names = True,
                rounded = True, proportion = False,
                precision = 2, filled = True)

# Convert to png using system command (requires Graphviz)
from subprocess import call
call(['dot', '-Tpng', 'treeRF.dot', '-o', 'treeRF.png', '-Gdpi=600'])

# Display in jupyter notebook
from IPython.display import Image
Image(filename = 'treeRF.png')
```

Output:

Out[14]:



Testdata inladen:

```
dffs_test_1_234 = create_dfframes(['Category_1', 'Category_2', 'Category_3', 'Category_4'],
                          extype = ['AB', 'AF', 'RF', 'EL'],
                          data_dir = '../sliced_transformed_testdata',
                          print_errors=True)
exercises_test_1_234 = dffs_to_exercises(dffs_test_1_234)

exercises_test_2_34 = exercises_test_1_234.drop_category(1)
```

Files: 198

Model toepassen op testdata:

```
exp_test_1_234 = Experiment(exercises_test_1_234, y_condition= lambda y: y != 'Category_1')
columns = exp_test_1_234.df.columns.to_numpy()

exp_test_2_34 = Experiment(exercises_test_1_234, y_condition= lambda y: y != 'Category_2')
columns = exp_test_2_34.df.columns.to_numpy()

rfc3 = RandomForestClassifier(n_estimators = 190,
 min_samples_split = 11,
 min_samples_leaf=4,
 max_features= 'sqrt',
 max_depth= None,
 bootstrap= False,
 min_impurity_decrease = 0,
 criterion= 'entropy',
 ccp_alpha = 0,
 n_jobs = -1,
 verbose = 1,
 oob_score = False)


rfc3.fit(X,y)

# initialise the X test set and y test set

X_test_1_234 = exp_test_1_234.df.values
y_test_1_234 = exp_test_1_234.y

X_test_2_34 = exp_test_2_34.df.values
y_test_2_34 = exp_test_2_34.y
```

```
ypred_1_234 = rfc3.predict(X_test_1_234)
```

```
print(classification_report(y_test_1_234, ypred_1_234))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.67      0.80         6
         1.0       0.90      1.00      0.95        19

    accuracy                           0.92        25
   macro avg       0.95      0.83      0.88        25
weighted avg       0.93      0.92      0.91        25
```

```
ypred_2_34 = rfc3.predict(X_test_2_34)
```

In [22]:
```
print(classification_report(y_test_2_34, ypred_2_34))
```

```
                precision    recall  f1-score   support

           0.0       0.00      0.00      0.00         7
           1.0       0.67      0.78      0.72        18

      accuracy                           0.56        25
     macro avg       0.33      0.39      0.36        25
  weighted avg       0.48      0.56      0.52        25
```