Supervised learning

The goal of this project is to predict the results of a set of PyRat game. The characteristics of the PyRat games are the following:

- layout: 5*5
- number of cheese: 10
- no mud
- no wall
- synchronous

Choosing a classifier

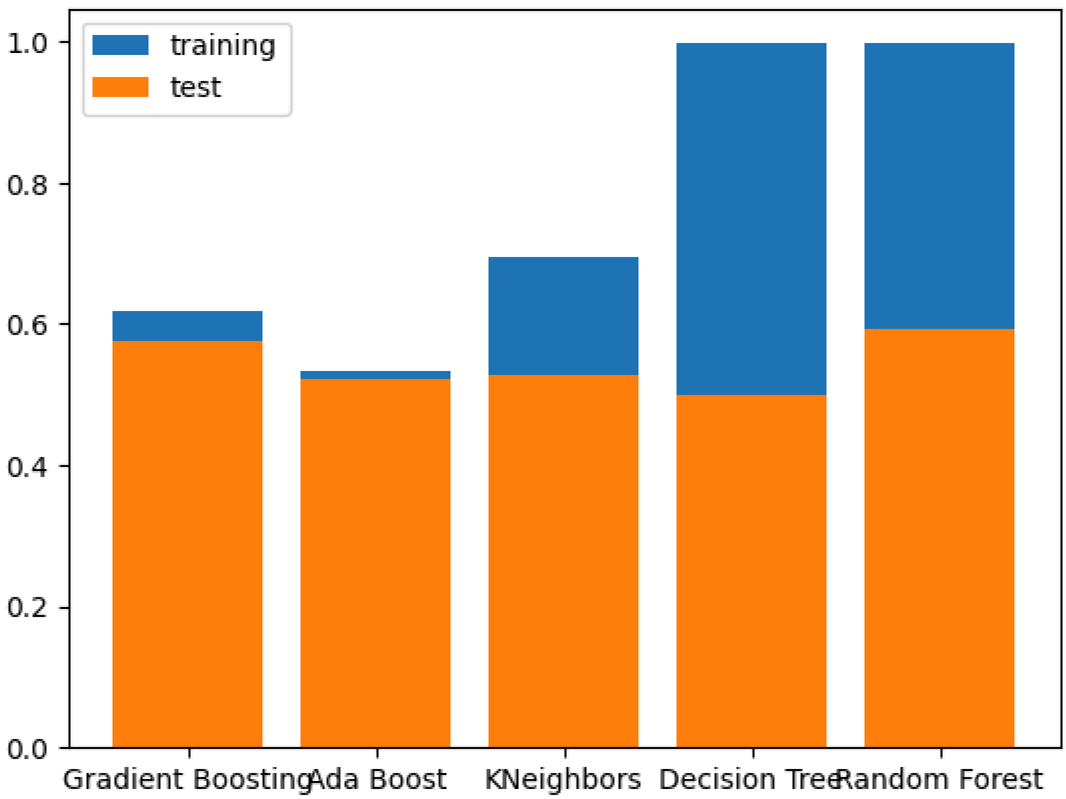
To choose a classifier, we decided to run some of them without tuning, that is without modifying the parameters of the classifier.

We first need to load the date to train and test ours classifiers:

In [34]:
```
import os
import tqdm
import ast
import matplotlib.pyplot as plt
import numpy as np

### LOADING THE DATA
filename = "./game_data.npz" # change it to point to your data set
loaded_npz = np.load(filename)
X = loaded_npz["x"]
y = loaded_npz["y"]
```

In [35]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.8, ra

classifiers = []
classifiers_score = []

classifiers.append({"name": "Gradient Boosting", "classifier": GradientBoost
classifiers.append({"name": "Ada Boost", "classifier": AdaBoostClassifier})
classifiers.append({"name": "KNeighbors", "classifier": KNeighborsClassifier
classifiers.append({"name": "Decision Tree", "classifier": DecisionTreeClass
classifiers.append({"name": "Random Forest", "classifier": RandomForestClass
```

In [36]:
```
for classifier in classifiers:
  cl_name = classifier["name"]
  cl = classifier['classifier']()
  cl.fit(x_train, y_train)

  cl_training_score = cl.score(x_train, y_train)
  cl_test_score = cl.score(x_test, y_test)

  classifiers_score.append({"name": cl_name, "training_score": cl_training_s


plt.bar([cl["name"] for cl in classifiers_score], [cl["training_score"] for
plt.bar([cl["name"] for cl in classifiers_score], [cl["test_score"] for cl i
plt.legend(["training", "test"])
plt.show()
```



We see that Gradient Boosting obtains better results overall on the test data set. We also notice over-fitting issue with the Decision Tree and Random Forest classifiers.

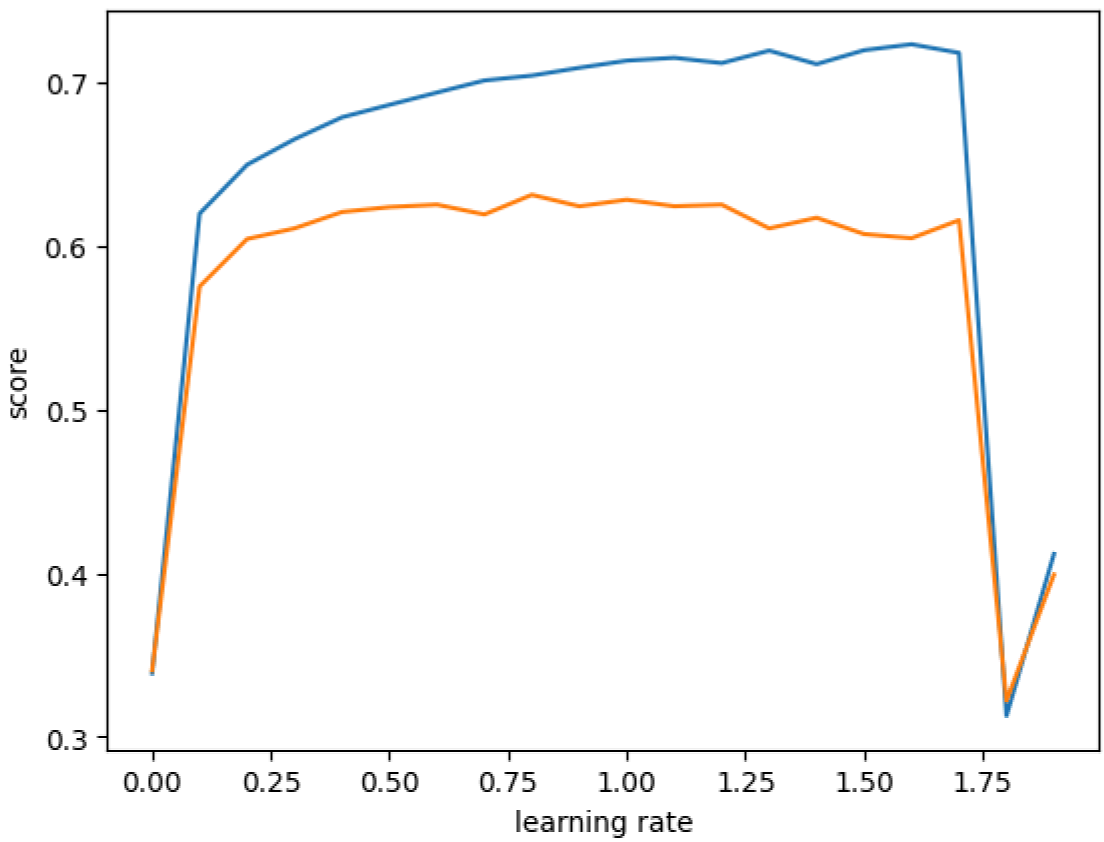
Training the Gradient Boosting

Gradient Boosting is what we call a boosting technique. It means that we'll not have one but multiple predictors, in our case, we will construct multiple successive decision trees. The first tree is our base model. And the following trees will try to predict the residuals of the previous ones.

Gradient Boosting are very fast at building trees, to slow them down and obtain a better result, we can adjust the `learning rate` parameter. To choose the appropriate value, we compute the precision obtained with different value of the learning rate.

In [37]:
```
learning_rate = np.arange(0, 2, 0.1)
training_score = []
test_score = []

for rate in learning_rate:
  gb_classifier = GradientBoostingClassifier(learning_rate=rate)
  gb_classifier.fit(x_train, y_train)
  training_score.append(gb_classifier.score(x_train, y_train))
  test_score.append(gb_classifier.score(x_test, y_test))

plt.plot(learning_rate, training_score, label="Training")
plt.plot(learning_rate, test_score, label="Test")
plt.xlabel("learning rate")
plt.ylabel("score")
plt.show()
```

```
Accuracy on the training data set: 0.719375
Accuracy on the test data set: 0.607
```



We notice that the score on the test data set is the highest when the learning rate is approximatively 0.8. We then train our classifier on a set of 18 234 games.

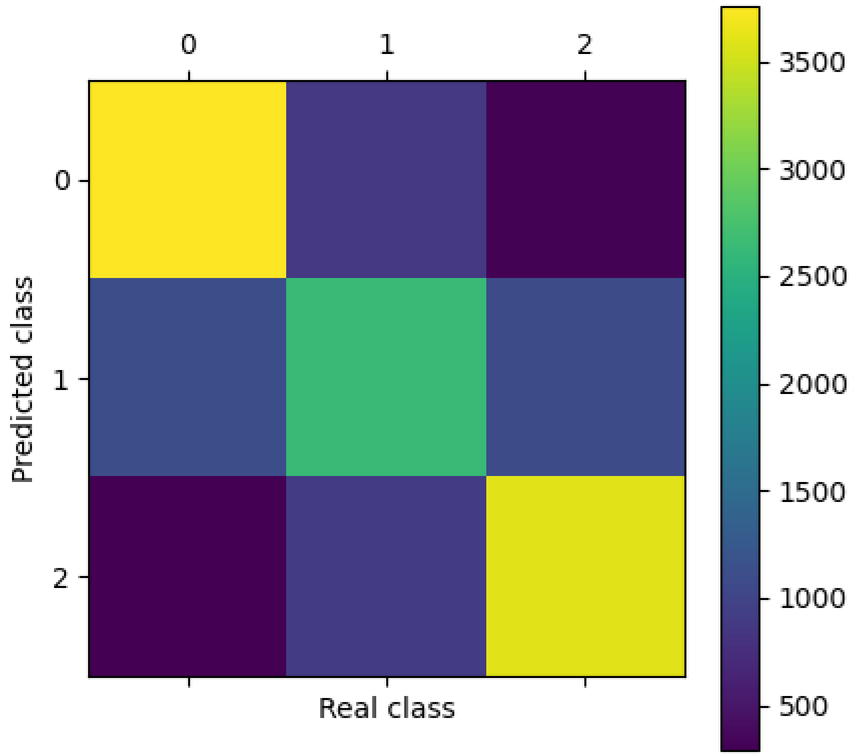
In [39]:
```python
filename = "./training_set.npz" # change it to point to your data set
loaded_npz = np.load(filename)
X = loaded_npz["x"]
y = loaded_npz["y"]

x_train, x_test, y_train, y_test = train_test_split(X, y, train_size=0.8, ra

gb_classifier = GradientBoostingClassifier(learning_rate=0.8)
gb_classifier.fit(x_train, y_train)

gb_classifier_training_score = gb_classifier.score(x_train, y_train)
gb_classifier_test_score = gb_classifier.score(x_test, y_test)

print(f"Accuracy on the training data set: {gb_classifier_training_score}")
print(f"Accuracy on the test data set: {gb_classifier_test_score}")
```

```
Accuracy on the training data set: 0.6851305957359293
Accuracy on the test data set: 0.6287359473539896
```

In [40]:
```python
from sklearn.metrics import classification_report,confusion_matrix

y_pred_train = gb_classifier.predict(x_train)
report = classification_report(y_true=y_train,y_pred=y_pred_train)
matrix = confusion_matrix(y_true=y_train,y_pred=y_pred_train)
print("Training Set:")
print(report)
print(matrix)
plt.matshow(matrix)
plt.colorbar()
plt.xlabel("Real class")
plt.ylabel("Predicted class")
```

```
Training Set:
              precision    recall  f1-score   support

        -1.0       0.73      0.76      0.74      4935
         0.0       0.60      0.54      0.57      4852
         1.0       0.72      0.75      0.74      4800

    accuracy                           0.69     14587
   macro avg       0.68      0.68      0.68     14587
weighted avg       0.68      0.69      0.68     14587

[[3758  872  305]
 [1123 2629 1100]
 [ 291  902 3607]]
```

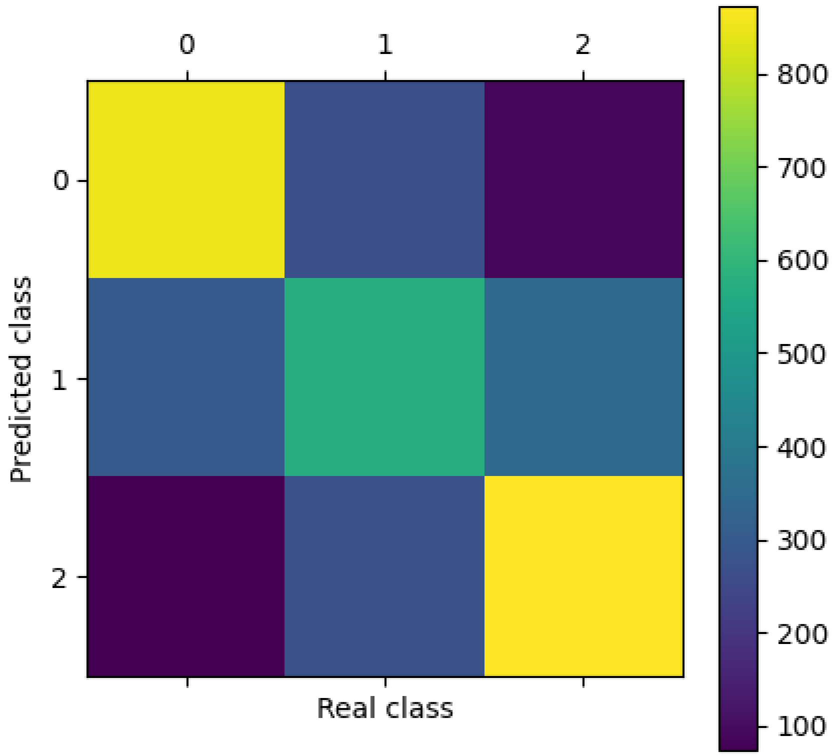
Out[40]:
```
Text(0, 0.5, 'Predicted class')
```



In [41]:
```
y_pred_test = gb_classifier.predict(x_test)
report = classification_report(y_true=y_test,y_pred=y_pred_test)
matrix = confusion_matrix(y_true=y_test,y_pred=y_pred_test)
print("Test Set:")
print(report)
print(matrix)
plt.matshow(matrix)
plt.colorbar()
plt.xlabel("Real class")
plt.ylabel("Predicted class")
```

```
Test Set:
              precision    recall  f1-score   support

        -1.0       0.69      0.71      0.70      1207
         0.0       0.51      0.47      0.49      1222
         1.0       0.67      0.72      0.69      1218

    accuracy                           0.63      3647
   macro avg       0.62      0.63      0.63      3647
weighted avg       0.62      0.63      0.63      3647

[[851 268  88]
 [302 570 350]
 [ 74 272 872]]
```

Out[41]:
```
Text(0, 0.5, 'Predicted class')
```


0
1
2
0
1
2
Predicted class
Real class
800
700
600
500
400
300
200
100