```
                                                    y_data,
                                                    test_size=0.20,
                                                    shuffle=True)

# Train model
#clf.fit(X_train, y_train)

# Predict the test data
#y_pred = clf.predict(X_test)
```

4. Build your classifier

Now everything (almost) ready to build your classifier.

Below code is an example for creating an Random Forest classifier, training , and calculating its accuracy

Entrée [ ]:

Entrée [167]:
```
# import PCA from sklearn
from sklearn.decomposition import PCA

pca = PCA(n_components=40)
pca.fit(x_data)
X_r = pca.transform(x_data)
print(f"data.shape after PCA : {X_r.shape}")
X_train, X_test, y_train, y_test = train_test_split(X_r,
                                                    y_data,
                                                    test_size=0.20,
                                                    shuffle=True)


#RANDOM FOREST CLASSIFIER
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

clf = RandomForestClassifier(max_depth=70)
#en mettant max_depth a 9 on obtient 90%
#clf.fit(x_data, y_data)
# Train model
clf.fit(X_train, y_train)
# Predict the test data
y_pred = clf.predict(X_test)
# the resulted accuracy is on a small set which is same for train and test
#print("Accuracy",clf.score(x_data, y_data))
print("Accuracy : ",accuracy_score(y_test,y_pred))
```

```
data.shape after PCA : (589, 40)
Accuracy :  0.6949152542372882
```

Entrée [200]:
```
# import PCA from sklearn
from sklearn.decomposition import PCA

pca = PCA(n_components=40)
pca.fit(x_data)
X_r =pca.transform(x_data)
print(f"data.shape after PCA :{X_r.shape}")
X_train, X_test, y_train, y_test = train_test_split(X_r,
                                                    y_data,
```

```
                                                    test_size=0.20,
                                                    shuffle=True)


#GAUSSIAN NAIVES BAYES CLASSIFIER
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
#clf.fit(x_data, y_data)
# Train model
clf.fit(X_train, y_train)
# Predict the test data
y_pred = clf.predict(X_test)
# the resulted accuracy is on a small set which is same for train and test
#print("Accuracy with all data : ",clf.score(x_data, y_data))
print("Accuracy_1 : ",accuracy_score(y_test,y_pred))
```

```
data.shape after PCA :(589, 40)
Accuracy_1 :  0.5423728813559322
```

Entrée [192]:

```
# import PCA from sklearn
from sklearn.decomposition import PCA

pca = PCA(n_components=40)
pca.fit(x_data)
X_r =pca.transform(x_data)
print(f"data.shape after PCA :{X_r.shape}")
X_train, X_test, y_train, y_test = train_test_split(X_r,
                                                    y_data,
                                                    test_size=0.20,
                                                    shuffle=True)



#C-SUPPORT VECTOR CLASSIFIER
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
#clf = make_pipeline(StandardScaler(), SVC(gamma='auto'))
clf = make_pipeline(StandardScaler(), SVC(gamma='scale'))
#clf.fit(x_data, y_data)
# Train model
clf.fit(X_train, y_train)
# Predict the test data
y_pred = clf.predict(X_test)
print("Accuracy : ",accuracy_score(y_test,y_pred))

#print("Accuracy with all data : ",clf.score(x_data, y_data))
```

```
data.shape after PCA :(589, 40)
Accuracy :  0.7203389830508474
```

Entrée [199]:

```
# import PCA from sklearn
from sklearn.decomposition import PCA

pca = PCA(n_components=40)
pca.fit(x_data)
X_r =pca.transform(x_data)
print(f"data.shape after PCA :{X_r.shape}")
X_train, X_test, y_train, y_test = train_test_split(X_r,
```

```
                                                    y_data,
                                                    test_size=0.20,
                                                    shuffle=True)



#MULTILAYER PERCEPTION CLASSIFIER
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
#clf = MLPClassifier(random_state=1, max_iter=300).fit(x_data, y_data)
clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)

#clf = MLPClassifier(solver='sgd', activation='tanh',max_iter=2000,
#                        hidden_layer_sizes=(5000,5000,))
#clf = GaussianNB()
#clf.fit(x_data, y_data)
# Train model
clf.fit(X_train, y_train)
# Predict the test data
y_pred = clf.predict(X_test)

print("Accuracy : ",accuracy_score(y_test,y_pred))

#print("score" clf.score(X_train,y_train))
# the resulted accuracy is on a small set which is same for train and test
#il faut diviser la base de donnée et utiliser 80% des data_test pour test
#utiliser y_pred = clf.predict(X_test) pour prédire l'accuracy au lieu de
#useAccuracy(y_test,y_pred)
#print("Accuracy",clf.score(x_data, y_data))
```

```
data.shape after PCA :(589, 40)
Accuracy :  0.5084745762711864
```

5. Have you used different data for train and test?

Entrée [ ]:
```
#Yes i have used shuffle to separe my data base for test and train data
```

6. Find a model with the best accuracy

In order to find the model with highest accuracy the performance of below combiniations should be tested.

1. Compare two feature extractors
2. Find the best hyperparameter for models : for example you can google "sklearn RandomForestClassifier" and go to this link (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) to find the RandomForestClassifier hyperparameteres (some of RandomForestClassifier's hyperparametere : n_estimators , criterion , max_depth)
3. Compare different classification algorithems

Below you can find a lits of algorithem with hyperparameters that can be tested:

Gaussian Naive Bayes (https://scikit-learn.org/stable/modules/generated