* **RandomForestClassifier para clasificación, discreto**
* **RandomForestRegression para output continuo**

from sklearn.ensemble import RandomForestClassifier

import numpy as np

from sklearn.datasets import load\_digits

digits = load\_digits()

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(digits.data,digits.target,test\_size=0.3)

from sklearn.model\_selection import cross\_val\_score

grupos = 10

def evaluar(arboles):

    scores1 = cross\_val\_score(RandomForestClassifier(n\_estimators=arboles),digits.data,

                                                      digits.target, cv=grupos)

    print(np.average(scores1))

evaluar(5)

evaluar(20)

evaluar(30)

evaluar(40)

evaluar(5) 0.8796

evaluar(20) 0.935

evaluar(30) 0.9494

evaluar(40) 0.9482

El mejor score esta entre crear 30 – 40 arboles

Hacemos modelo Random Forest con 30

rf = RandomForestClassifier(n\_estimators=30)

rf.fit(X,y)

Predecimos

valor = X\_test[1]

rf.predict([valor]) **#4**

Utilizamos RandomRegression

from sklearn.model\_selection import RandomizedSearchCV

from sklearn.ensemble import RandomForestRegressor

modelo = RandomForestRegressor()

parametros = {

   'n\_estimators': range(4,15),  #busque entre 4 y 15 arboles

   'criterion': ['mse','mae'],

   'max\_depth': range(2,11)  #que tan profundo el arbol

}

X = X\_train

y = y\_train

rand\_est = RandomizedSearchCV(modelo,parametros,n\_iter=10, cv=3, scoring='neg\_mean\_absolute\_error').fit(X,y)

print("")

print("Mejores criterios a usar son ",rand\_est.best\_estimator\_)

print("Mejores parametros a usar son ", rand\_est.best\_params\_)

Output

Mejores criterios a usar son RandomForestRegressor(criterion='mae', max\_depth=9, n\_estimators=10)

Mejores parametros a usar son {'n\_estimators': 10, 'max\_depth': 9, 'criterion': 'mae'}

rfr= RandomForestRegressor(bootstrap=True, ccp\_alpha=0.0, criterion='mae',

                      max\_depth=9, max\_features='auto', max\_leaf\_nodes=None,

                      max\_samples=None, min\_impurity\_decrease=0.0,

                      min\_samples\_leaf=1,

                      min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,

                      n\_estimators=10, n\_jobs=None, oob\_score=False,

                      random\_state=None, verbose=0, warm\_start=False)

score = rfr.fit(X, y)

ypred = rfr.predict(X\_test)

from sklearn.metrics import mean\_squared\_error

mse = mean\_squared\_error(y\_test, ypred)

print("MSE: ", mse)

print("RMSE: ", mse\*(1/2.0))

MSE: 1.4308888

RMSE: 0.7154

rfr.score(X\_test,y\_test)

Output

0.82089

Tiene menos exactitud que RandomForestClassification que logra 94% cuando hace 30 arboles

rfr.predict([valor])

Output: 4 igual que en RandomForest / RandomRegression

Whether you use a classifier or a regressor **only** depends on the kind of problem you are solving.

You have a binary **classification** problem, so use the classifier.

I could run randomforestregressor first and get back a set of estimated probabilities.

You don't get probabilities from regression. It just tries to "extrapolate" the values you give (in this case only 0 and 1). This means values above 1 or below 0 are perfectly valid as a regression output as it does not expect only two discrete values as output (that's called classification!) but continuous values.

If you want to have the "probabilities" (be aware that these don't have to be well calibrated probabilities) for a certain point to belong to a certain class, train a classifier (so it learns to classify the data) and then use .predict\_proba(), which then predicts the probability.

Just to mention it here: .predict vs .predict\_proba (for a classifier!)  
.predict just takes the .predict\_proba output and changes everything to 0 below a certain threshold (usually 0.5) respectively to 1 above that threshold.

Remark: sure, internally, they are the very same except from the "last layer" etc.! Still, see them (or better the problem they are solving) as completely different!