import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import GradientBoostingClassifier

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

from sklearn.metrics import accuracy\_score

#Eliminar warnings

import warnings

warnings.simplefilter("ignore")

dt\_heart = pd.read\_csv('heart.csv')

y= dt\_heart['target']

x= dt\_heart.drop(['target'], axis=1)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.35)

estimators = range(10, 200, 10)

total\_accuracy = []

for i in estimators:

    boost = GradientBoostingClassifier(n\_estimators=i).fit(x\_train, y\_train)

    boost\_pred = boost.predict(x\_test)

    total\_accuracy.append(accuracy\_score(y\_test, boost\_pred))

import matplotlib.pyplot as plt

plt.plot(estimators, total\_accuracy)

plt.xlabel('Estimators')

plt.ylabel('Accuracy')

plt.show()

#plt.savefig('Boost.png')

import numpy as np

print(np.array(total\_accuracy).max())

|  |  |
| --- | --- |
|  | IMPLICA QUE CUANDO LLEGA A 150 ESTIMACIONES YA NO AUMENTA MAS LA EXACTITUD  Si da  estimators = range(150, 400, 150)  Da una recta ??? |

boosting = GradientBoostingClassifier(loss='exponential',learning\_rate=0.15, n\_estimators=100, max\_depth=5).fit(x\_train, y\_train)

estimators = range(150, 400, 150)

total\_accuracy = []

for i in estimators:

    boost = GradientBoostingClassifier(loss='exponential',n\_estimators=i).fit(x\_train, y\_train)

    boost\_pred = boost.predict(x\_test)

    total\_accuracy.append(accuracy\_score(y\_test, boost\_pred))

import matplotlib.pyplot as plt

plt.plot(estimators, total\_accuracy)

plt.xlabel('Estimators')

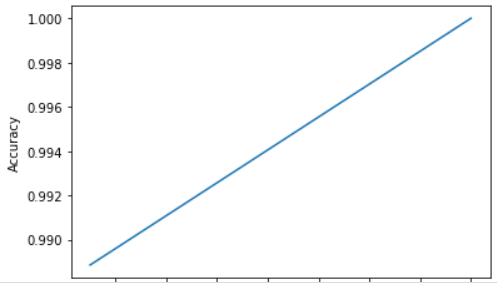
plt.ylabel('Accuracy')

plt.show()

plt.savefig('Boost.png')

import numpy as np

print(np.array(total\_accuracy).max())



**loss*{‘deviance’, ‘exponential’}, default=’deviance’***

The loss function to be optimized. ‘deviance’ refers to deviance (= logistic regression) for classification with probabilistic outputs. For loss ‘exponential’ gradient boosting recovers the AdaBoost algorithm.

**learning\_rate*float, default=0.1***

Learning rate shrinks the contribution of each tree by learning\_rate. There is a trade-off between learning\_rate and n\_estimators.

**n\_estimators*int, default=100***

The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance.

**subsample*float, default=1.0***

The fraction of samples to be used for fitting the individual base learners. If smaller than 1.0 this results in Stochastic Gradient Boosting. subsample interacts with the parameter n\_estimators. Choosing subsample < 1.0 leads to a reduction of variance and an increase in bias.

**criterion*{‘friedman\_mse’, ‘squared\_error’, ‘mse’, ‘mae’}, default=’friedman\_mse’***

The function to measure the quality of a split. Supported criteria are ‘friedman\_mse’ for the mean squared error with improvement score by Friedman, ‘squared\_error’ for mean squared error, and ‘mae’ for the mean absolute error. The default value of ‘friedman\_mse’ is generally the best as it can provide a better approximation in some cases.