```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

1- Explorando dígitos escritos a mano

Para analizar los conceptos vistos anteriormente sobre un problema más interesante, consideremos una parte del problema del reconocimiento óptico de caracteres: la identificación de dígitos escritos a mano. En la práctica, este problema implica ubicar e identificar caracteres en una imagen.

a.- Cargue el dataset de imágenes de dígitos provisto por sklearn y verifique el tipo de dato importado.

```
digits=load_digits()
type(digits)
```

```
sklearn.utils.Bunch
```

b.-Verifique cuales son las claves del objeto Bunch

```
print("Keys of digits: {}".format(digits.keys()))
```

```
Keys of digits: dict_keys(['data', 'target', 'target_names', 'images', 'DESCR'])
```

este conjunto de datos tiene como meta-dato las imágenes en su formato original, que están asociadas a la clave images

c.- Verifique cuantas instancias tiene el dataset y cuantos son los atributos. ¿Nota algo diferente?

```
print(digits.data.shape)
print(digits.DESCR)
```

```
(1797, 64)
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 5620
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

Se puede observar una diferencia en la cantidad de instancias entre la presentadas mediante shape y las que presenta DESCR.

Quizas se deba a un recorte de informacion en el dataset.

d.- Compruebe cual es la información de la primera imagen. ¿Que significan los numeros?

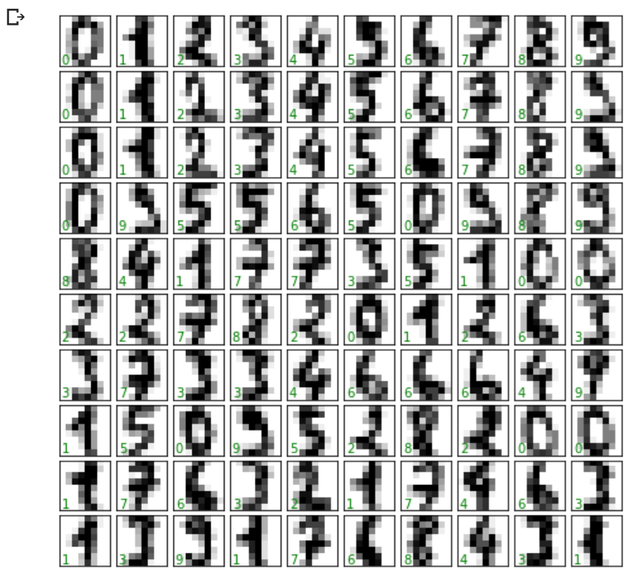
```
digits.images[0]
```

```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

Los números del array representan la intensidad de cada pixel.

e.- El siguiente código muestra de forma más clara la información de las primeras 100 imágenes.

```
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i, ax in enumerate(axes.flat):
  ax.imshow(digits.images[i], cmap='binary', interpolation='nearest')
  ax.text(0.05, 0.05, str(digits.target[i]),
          transform=ax.transAxes, color='green')
```



Para trabajar con estos datos dentro de Scikit-Learn, necesitamos una representación bidimensional, [n_samples, n_features]. Podemos lograr esto tratando cada píxel de la imagen como una característica, es decir, "aplanando" las matrices de píxels de manera tal de tener una matriz de valores de pixels de longitud 64 que representa cada dígito. Además, necesitamos el vector objetivo, que proporciona la etiqueta previamente determinada para cada dígito. Estas dos cantidades, como es usual, ya están directamente disponibles en el conjunto de datos digits bajo las claves data y target, respectivamente.

f.-Aplique el algoritmo de clasificación KNN (K=1) a los digitos. Preeviamente divida el dataset utilizando la función de sklearn para dividir en datos de entrenamiento y test.

g.- Verifique el Accuracy del modelo utilizando.

h.-Aplique el algoritmo de clasificación KNN para K = 3,5,7,9,11. Compare la Accuracy de cada modelo.¿Cuál es el mejor K?

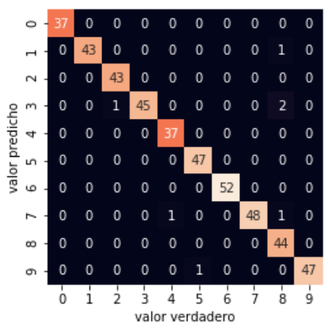
```
xTrain, xTest, yTrain, yTest = train_test_split(digits.data, digits.target,
                                                random_state = 0)
scaler = MinMaxScaler()
xTrain = scaler.fit_transform(xTrain)
xTest = scaler.transform(xTest)

for i in range(1,12,2):
  model = KNeighborsClassifier(i)
  model.fit(xTrain, yTrain)
  pred=model.predict(xTest)
  print('K=',i,' Accuracy:{:.4f}' .format(accuracy_score(yTest,pred)))
```

```
K= 1  Accuracy:0.9911
K= 3  Accuracy:0.9889
K= 5  Accuracy:0.9844
K= 7  Accuracy:0.9800
K= 9  Accuracy:0.9778
K= 11  Accuracy:0.9733
```

i.- ¿Como saber donde se equivoco el modelo? Utilice la librería seaborn para imprimir la matriz de confusión.

```
model=KNeighborsClassifier(5)
model.fit(xTrain,yTrain)
pred=model.predict(xTest)
mat = confusion_matrix(yTest, pred)
sns.heatmap(mat.T, square=True, annot=True, cbar=False)
plt.xlabel('valor verdadero')
plt.ylabel('valor predicho')
xTest.shape
```

```
(450, 64)
```

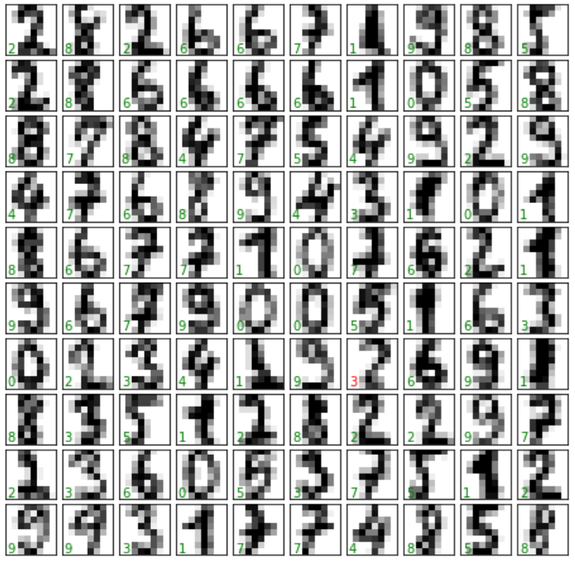


j.- Viendo la matriz de confusión. Que puede decir acerca de las predicciones? Cual es el número/s que tiende a ser mal etiquetado?

En general el modelo produce buenos aciertos. Solo tiende a fallar con numeros de caracteristicas similares. El mayor numero de errores se da al confundir el numero 3 con el numero 8.

k.- El siguiente código muestra las imágenes y las etiquetas predecidas por el clasificador.

```
fig, axes = plt.subplots(10, 10, figsize=(8, 8),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.1, wspace=0.1))
test_images = xTest.reshape(-1, 8, 8)
for i, ax in enumerate(axes.flat):
  ax.imshow(test_images[i], cmap='binary', interpolation='nearest')
  ax.text(0.05, 0.05, str(pred[i]), transform=ax.transAxes,
          color='green' if (yTest[i] == pred[i]) else 'red')
```



```
import numpy as np
import sklearn
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
import graphviz
```

2) Trabajando con el dataset de diagnóstico de Cancer de Mama del Hospital de Winsconsin

a.- Cargue el dataset de load_breast_cancer provisto por sklearn y verifique el tipo de dato importado.

```
cancer = load_breast_cancer()
type(cancer)
```

```
sklearn.utils.Bunch
```

b.- Verifique cuales son las claves del objeto Bunch.

```
print("Keys of cancer: {}".format(cancer.keys()))
```

```
Keys of cancer: dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

c.- Verifique la forma del dataset. Cantidad de instancias y features.

```
print(cancer.data.shape)
print(cancer.DESCR)
```

```
(569, 30)
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
        - concave points (number of concave portions of the contour)
        - symmetry
        - fractal dimension ("coastline approximation" - 1)

        The mean, standard error, and "worst" or largest (mean of the three
        largest values) of these features were computed for each image,
        resulting in 30 features.  For instance, field 3 is Mean Radius, field
        13 is Radius SE, field 23 is Worst Radius.

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ====== ======
                                           Min    Max
    ===================================== ====== ======
    radius (mean):                        6.981  28.11
    texture (mean):                       9.71   39.28
    perimeter (mean):                     43.79  188.5
    area (mean):                          143.5  2501.0
    smoothness (mean):                    0.053  0.163
    compactness (mean):                   0.019  0.345
    concavity (mean):                     0.0    0.427
    concave points (mean):                0.0    0.201
    symmetry (mean):                      0.106  0.304
    fractal dimension (mean):             0.05   0.097
    radius (standard error):              0.112  2.873
    texture (standard error):             0.36   4.885
    perimeter (standard error):           0.757  21.98
    area (standard error):                6.802  542.2
    smoothness (standard error):          0.002  0.031
    compactness (standard error):         0.002  0.135
    concavity (standard error):           0.0    0.396
    concave points (standard error):      0.0    0.053
    symmetry (standard error):            0.008  0.079
    fractal dimension (standard error):   0.001  0.03
    radius (worst):                       7.93   36.04
    texture (worst):                      12.02  49.54
    perimeter (worst):                    50.41  251.2
    area (worst):                         185.2  4254.0
    smoothness (worst):                   0.071  0.223
    compactness (worst):                  0.027  1.058
    concavity (worst):                    0.0    1.252
    concave points (worst):               0.0    0.291
    symmetry (worst):                     0.156  0.664
    fractal dimension (worst):            0.055  0.208
    ===================================== ====== ======

    :Missing Attribute Values: None
```

```
        :Class Distribution: 212 - Malignant, 357 - Benign

        :Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

        :Donor: Nick Street

        :Date: November, 1995

    This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
    https://goo.gl/U2Uwz2

    Features are computed from a digitized image of a fine needle
    aspirate (FNA) of a breast mass.  They describe
    characteristics of the cell nuclei present in the image.

    Separating plane described above was obtained using
    Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
    Construction Via Linear Programming." Proceedings of the 4th
    Midwest Artificial Intelligence and Cognitive Science Society,
    pp. 97-101, 1992], a classification method which uses linear
    programming to construct a decision tree.  Relevant features
    were selected using an exhaustive search in the space of 1-4
    features and 1-3 separating planes.

    The actual linear program used to obtain the separating plane
    in the 3-dimensional space is that described in:
    [K. P. Bennett and O. L. Mangasarian: "Robust Linear
    Programming Discrimination of Two Linearly Inseparable Sets",
    Optimization Methods and Software 1, 1992, 23-34].

    This database is also available through the UW CS ftp server:

    ftp ftp.cs.wisc.edu
    cd math-prog/cpo-dataset/machine-learn/WDBC/

    .. topic:: References

      - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
        for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
        Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
        San Jose, CA, 1993.
      - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
        prognosis via linear programming. Operations Research, 43(4), pages 570-577,
        July-August 1995.
      - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
        to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
        163-171.
```

d.- Genere un modelo de árbol de desición usando el clasificador DecisionTreeClassifier

e.- Separe el dataset en 63% para train set y 37% para test set. Use la función train_test_split

f.- Usando el método score() del modelo entrenado verifique la Accuracy para el conjunto de entrenamiento y para el conjunto de pruebas.

```
xTrain, xTest, yTrain, yTest = train_test_split(cancer.data,cancer.target,stratify=cancer.target,test_size=0.37)
def build_tree(profundidad=None):
    tree = DecisionTreeClassifier(max_depth=profundidad,random_state=0)
    tree.fit(xTrain, yTrain)
    print("Profundidad: {0}".format(profundidad))
    print("Training Accuracy: {0:.4f}".format(tree.score(xTrain, yTrain)))
    print("Testing Accuracy: {0:.4f}".format(tree.score(xTest, yTest)))
```

```
build_tree()
```

```
Profundidad: None
Training Accuracy: 1.0000
Testing Accuracy: 0.9242
```

¿Por qué el conjunto de entrenamiento tiene una accuracy más alto? ¿Qué significa que tenga una precisión de 1.00 sobre el conjunto de entrenamiento? ¿Es posible que exista un sobreajuste? ¿De qué forma podemos aliviar este problema?

El modelo esta sobreajustado, el mismo no es capaz de generalizar lo suficiente porque le permitimos crecer hasta que cada hoja estuviera pura. Debido a esto cada instancia de los datos de entrenamiento eventualmente va a llegar a la hoja que lo contiene. Una solucion seria lograr la generalización del arbol mediante la poda.

g.- Configure el hiperparámetro max_depth=4. y realice un nuevo entrenamiento. ¿Que logra este hiperparámetro? Verifique los nuevos valores de Accuracy para el conjunto de entrenamiento y el de pruebas.

```
build_tree(4)
```

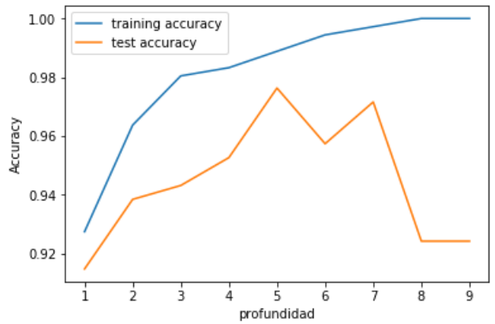
```
Profundidad: 4
Training Accuracy: 0.9832
Testing Accuracy: 0.9526
```

Este hiper parametro logra limitar la profundidad del arbol, logrando asi quitar complejidad al modelo y ganar generalizacion para la clasificacion.

h.- Pruebe el siguiente código y analice la gráfica resultante. ¿Que puede concluir sobre esta?

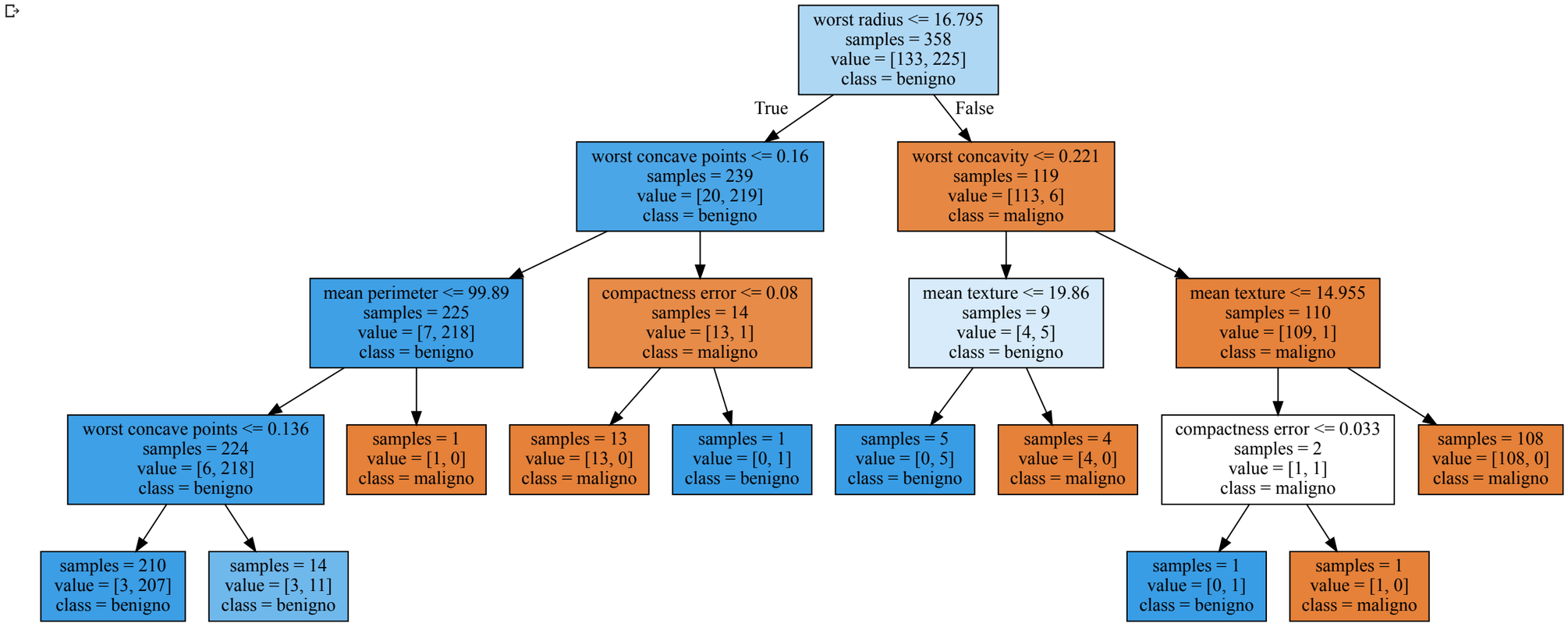
```
training_accuracy = []
test_accuracy = []
# probar profundidades de 1 a 10
profundidades_settings = range(1, 10)
for n_prof in profundidades_settings:
  # build the model
  clf = DecisionTreeClassifier(max_depth=n_prof, random_state=0)
  clf.fit(xTrain, yTrain)
  # record training set accuracy
  training_accuracy.append(clf.score(xTrain, yTrain))
  # record generalization accuracy
  test_accuracy.append(clf.score(xTest, yTest))
plt.plot(profundidades_settings, training_accuracy, label="training accuracy")
plt.plot(profundidades_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("profundidad")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f88b3ed5cc0>
```



i.- Es interesante visualizar el árbol de decisión generado, exportándolo como un archivo ".dot" (formato de archivo de textos para almacenar grafos) y visualizandolo con graphviz. Pruebe el siguiente código y analice su salida:

```
tree = DecisionTreeClassifier(max_depth=4)
tree.fit(xTrain, yTrain)
export_graphviz(tree, out_file="tree.dot", class_names=["maligno", "benigno"],feature_names=cancer.feature_names, impurity=False, filled=True)
with open("tree.dot") as f:
  dot_graph = f.read()
  display(graphviz.Source(dot_graph))
```



j.- Otra herramienta que nos puede ayudar en el análisis es determinar la importancia de los atributos en la construcción del árbol. Es un número entre 0 y 1 para cada feature donde 0 significa "no es considerado en absoluto" y 1 significa "perfectamente predice la clase". Las importancias de todas las características siempre deben sumar 1: Podemos obtener esa información por medio de la propiedad feature_importances_ del modelo.

Por ej. print("Feature importances:\n{}".format(tree.feature_importances_))

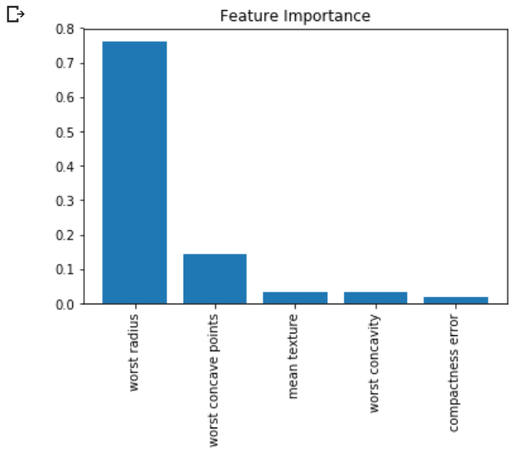
Verifique y nombre cuales són los 5 atributos más importantes considerados por el árbol de construcción en el problema de Cancer de Mama.

```
importances=tree.feature_importances_
print("Feature importances:\n{}".format(importances))
```

```
Feature importances:
[0.         0.03466149 0.01204644 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.01825065 0.         0.
 0.         0.         0.76097695 0.         0.         0.
 0.         0.         0.03173867 0.1423258  0.         0.        ]
```

```
indices = np.argsort(importances)[::-1]
indices=indices[:5]
names = [cancer.feature_names[i] for i in indices]
plt.figure()
```

```
    plt.title("Feature Importance")
    plt.bar(range(5), importances[indices])
    plt.xticks(range(5), names, rotation=90)
    plt.show()
```



```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

3) Trabajando con las 2 media lunas y el dataset de Cancer de Mama

a.- sklearn tiene disponible una función para generar datos aleatorios en forma de medialunas.

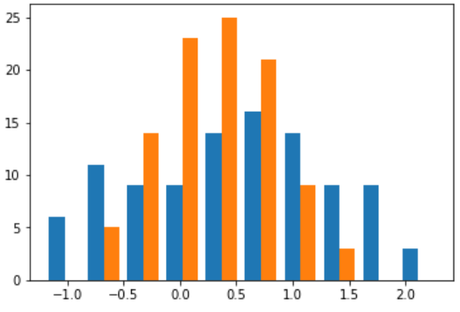
```
X, y = make_moons(n_samples=20, noise=0.25, random_state=3)
xTrain, xTest, yTrain, yTest = train_test_split(X, y, stratify=Y, random_state=42)
```

b.- Construya un dataset de medialunas de 100 muestras. Grafique el dataset y compruebe la distribución del dataset.

```
X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
```

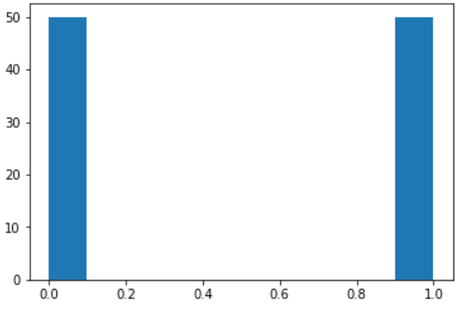
```
plt.hist(X)
```

```
([array([ 6., 11.,  9.,  9., 14., 16., 14.,  9.,  9.,  3.]),
  array([ 0.,  5., 14., 23., 25., 21.,  9.,  3.,  0.,  0.])],
 array([-1.20034998, -0.85163336, -0.50291674, -0.15420012,  0.19451649,
         0.54323311,  0.89194973,  1.24066635,  1.58938296,  1.93809958,
         2.2868162 ]),
 <a list of 2 Lists of Patches objects>)
```

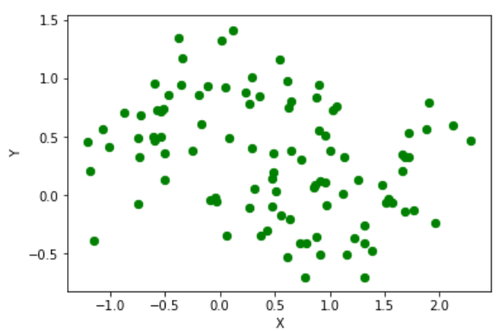


```
plt.hist(y)
```

```
(array([50.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 50.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```



```
plt.scatter(X[:, 0], X[:, 1], s = 40, color ='g')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
plt.clf()
```




```
<Figure size 432x288 with 0 Axes>
```

c.- Utilice el clasificador RandomForest usando: from sklearn.ensemble import RandomForestClassifier

Construya un modelo del dataset de medialunas usando los hiperparámetros (n_estimators=5, random_state=2).

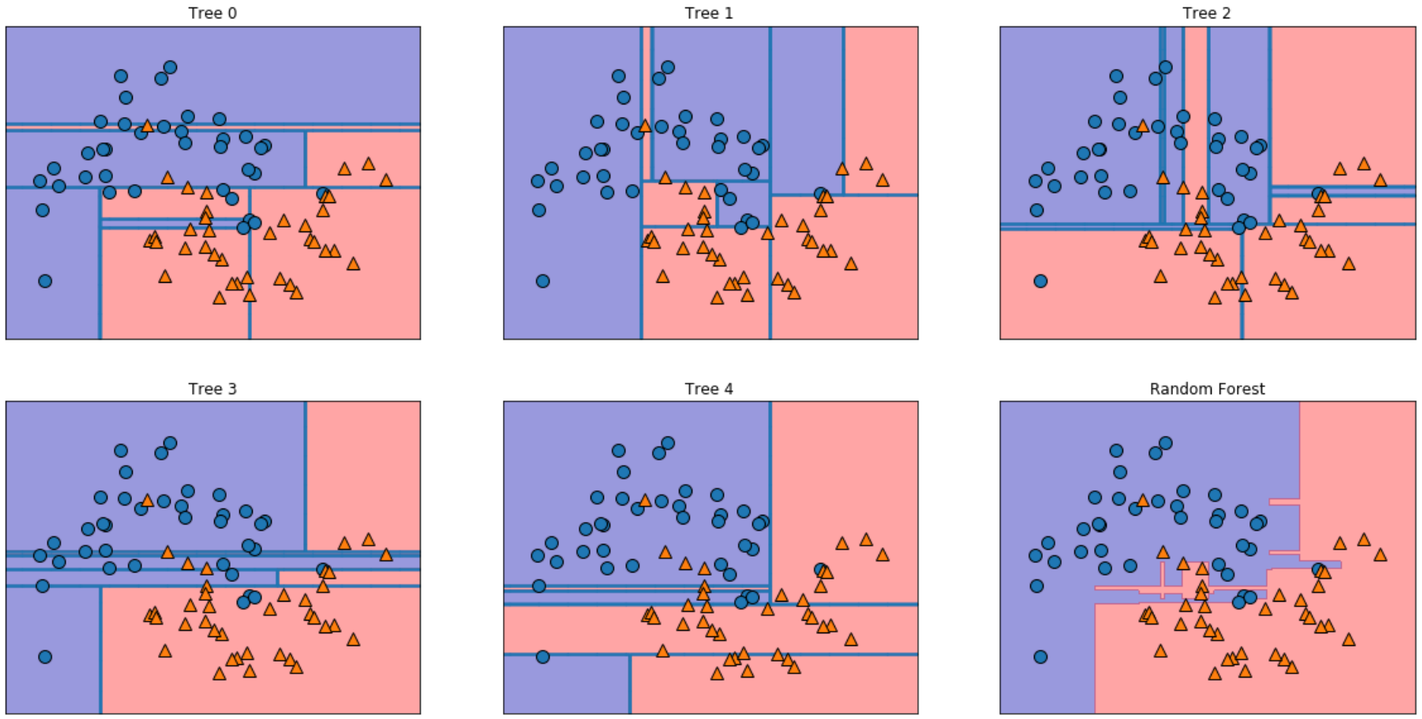
```
xTrain, xTest, yTrain, yTest = train_test_split(X, y, stratify=y,random_state=42)
model = RandomForestClassifier(n_estimators=5, random_state=2)
model.fit(xTrain, yTrain)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=5,
n_jobs=None, oob_score=False, random_state=2, verbose=0,
warm_start=False)
```

d.- Use el siguiente código para graficar cada uno de los árboles de decisión construidos por Random Forest y el modelo propio de Random Forest.

```
fig, axes = plt.subplots(2, 3, figsize=(20, 10))
for i, (ax, tree) in enumerate(zip(axes.ravel(), model.estimators_)):
  ax.set_title("Tree {}".format(i))
  mglearn.plots.plot_tree_partition(xTrain, yTrain, tree, ax=ax)
mglearn.plots.plot_2d_separator(model, xTrain, fill=True, ax=axes[-1, -1],alpha=.4)
axes[-1, -1].set_title("Random Forest")
mglearn.discrete_scatter(xTrain[:, 0], xTrain[:, 1], yTrain)
```

```
[<matplotlib.lines.Line2D at 0x7fd8461fe6d8>,
 <matplotlib.lines.Line2D at 0x7fd845da3128>]
```



Compruebe que los árboles construidos són diferentes entre ellos. ¿Cuál es el mejor modelo construido? ¿Por qué piensa que es mejor?

Cada uno de los arboles difieren en su toma de decisiones a la hora de clasificar. El mejor modelo construido es Ramdom Fosrest porque se basa en un conjunto de votaciones respecto a las decisiones que tomo cada uno de los demas arboles. Se puede ver como el area de decision se encuentra menos superpuesta con las otras clases.

e.- Construya un modelo Random Forest para el dataset de cáncer con 100 árboles. Verifique la Accuracy. Compare los valores con los generados en el Ejercicio 2. ¿Es mejor o no?

```
cancer = load_breast_cancer()
xTrain, xTest, yTrain, yTest = train_test_split(cancer.data, cancer.target,stratify=cancer.target, test_size=0.37)
model = RandomForestClassifier(n_estimators=100, random_state=0)
model.fit(xTrain, yTrain)
print("Training Accuracy: {:.4f}".format(model.score(xTrain, yTrain)))
print("Test Accuracy: {:.4f}".format(model.score(xTest, yTest)))
```

```
Training Accuracy: 1.0000
Test Accuracy: 0.9431
```

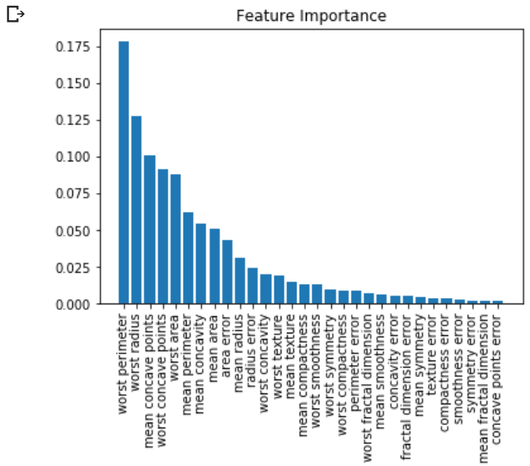
En la mayoria de las ejecuciones la accuracy mejora muy por encima del ejercicio 2, el cual esta realizado con un solo arbol de decisión.

f.- Verifique las importancias de los features en este nuevo modelo. Que diferencias existen con respecto a las del ejercicio 2.

```
importances=model.feature_importances_
print("Feature importances:\n{}".format(importances))
```

```
Feature importances:
[0.03096086 0.01459344 0.06217162 0.05081382 0.00623242 0.01290853
 0.0547763  0.10042495 0.00476858 0.00206989 0.02393947 0.00401752
 0.0085198  0.04328744 0.00235121 0.00350441 0.00505977 0.00165601
 0.00208477 0.00488668 0.12765268 0.01883927 0.1781557  0.08765129
 0.01282542 0.00897626 0.01968829 0.09110287 0.00941935 0.00666139]
```

```
indices = np.argsort(importances)[::-1]
names = [cancer.feature_names[i] for i in indices]
plt.figure()
plt.title("Feature Importance")
plt.bar(range(cancer.data.shape[1]), importances[indices])
plt.xticks(range(cancer.data.shape[1]), names, rotation=90)
plt.show()
```



Cambio con respecto al ejercicio 2, se consideraron todos los features debido a que cada arbol de decision generado por random forest trabajo aleatoriamente con cada uno de ellos. En cuanto a la importancia de los principales features con respecto al ejecicio 2 tambien cambio, ahora se consideran de mas peso otros features.

```
from sklearn.datasets import load_digits
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import classification_report
```

Ejercicio 4

a.-Aplique Random Forest sobre el dataset de reconocimiento de dígitos. ¿Mejora el modelo respecto a KNN?

```
digits=load_digits()
xTrain, xTest, yTrain, yTest = train_test_split(digits.data, digits.target,stratify=digits.target)
model = RandomForestClassifier(n_estimators=100, random_state=0)
model.fit(xTrain, yTrain)
print("Training Accuracy: {:.4f}".format(model.score(xTrain, yTrain)))
print("Test Accuracy: {:.4f}".format(model.score(xTest, yTest)))
```

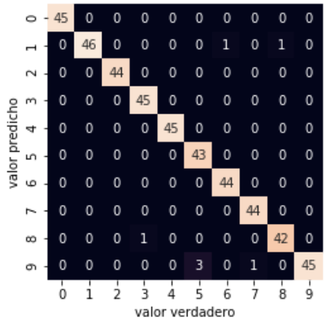
```
Training Accuracy: 1.0000
Test Accuracy: 0.9844
```

El Accuracy es similar al del modelo entrenado con KNN del ejercicio 1.

b.- Construya la matriz de confusión. Analice el Recall. ¿Qué puede decir acerca del análisis realizado?

```
pred=model.predict(xTest)
mat = confusion_matrix(yTest, pred)
sns.heatmap(mat.T, square=True, annot=True, cbar=False)
plt.xlabel('valor verdadero')
plt.ylabel('valor predicho');

print(classification_report(yTest, pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	45
1	0.96	1.00	0.98	46
2	1.00	1.00	1.00	44
3	1.00	0.98	0.99	46
4	1.00	1.00	1.00	45
5	1.00	0.93	0.97	46
6	1.00	0.98	0.99	45
7	1.00	0.98	0.99	45
8	0.98	0.98	0.98	43
9	0.92	1.00	0.96	45
accuracy			0.98	450
macro avg	0.99	0.98	0.98	450
weighted avg	0.99	0.98	0.98	450



Teniendo en cuenta el recall y la precision del informe anterior se puede deducir que la mayoria de los errores de las predicciones se deben a falsos positivos.