Programación para *Data Science*

Unidad 7: Análisis de datos en Python

En este Notebook encontraréis un conjunto de ejercicios para practicar. Estos ejercicios no puntúan para la PEC, pero os recomendamos que los intentís resolver como parte del proceso de aprendizaje. Encontraréis ejemplos de posibles soluciones a los ejercicios en el propio notebook, pero es importante que intentéis resolverlos vosotros antes de consultar las soluciones. Las soluciones os permitirán validar vuestras respuestas, así como ver alternativas de resolución de las actividades. También os animamos a preguntar cualquier duda que surja sobre la resolución de las actividades para practicar en el foro del aula.

Preguntas y ejercicios para practicar

Pregunta 1

Investiga y describe el funcionamiento del método de cross-validation *Leave One Out*. Pon un ejemplo de aplicación del algoritmo para resolver un problema de análisis en el ámbito de la salud. Recordad que hay que citar las referencias consultadas para responder la pregunta, y que la respuesta que proporcionéis debe ser original (redactada por vosotros mismos, después de haber leído y entendido las referencias que consideréis oportunas).

Respuesta

Pregunta 2

Debemos evitar evaluar los modelos con los mismos datos que se han utilizado para el aprendizaje. De no hacerlo, podríamos favorecer el problema del **sobre-ajuste**. ¿En qué consiste este problema y qué consecuencias puede tener?

Respuesta

Ejercicio 1

Cargad el conjunto de datos Iris incorporado en la librería `sklearn`. Implementad una función, `describe_iris`, que devuelva un diccionario con la siguiente estructura:

```
{
   "categorias": [],
   "atributos": [],
   "num_muestras": 0
}
```

categorias debe ser un array con el nombre de los **targets** del dataset. *atributos* debe ser un array con el nombre de los **atributos** y finalmente, *num_muestras* debe indicar el número **total de muestras** del dataset.

In [1]:

```
# Respuesta
```

Ejercicio 2

Representad gráficamente en un scatter plot la longitud de los sépalos frente al ancho de los sépalos. Nota: para poder incluir acentos en los textos de las etiquetas o del título del plot, es necesario indicar explícitamente que las cadenas de caracteres son unicode. Podéis hacerlo incluyendo una *u* delante de las comillas que delimitan la cadena de caracteres.

In [2]:

```
# Respuesta
```

Ejercicio 3

Dividir los datos Iris en dos subconjuntos, datos de entrenamiento y test, en una proporción 70% entrenamiento y 30% test.

In [3]:

```
# Respuesta
```

Ejercicio 4

Aplicad el clasificador **KNeighborsClassifier** para predecir el tipo de especie de iris utilizando la longitud y ancho de los pétalos como atributos y utilizando 60% de las muestras de aprendizaje y 40% muestras de test (podéis usar cualquier partición de muestras de aprendizaje y de test).

¿Qué rendimiento se obtiene?

In [4]:

```
#Respuesta
```

Ejercicio 5

Aplicad el algoritmo de *clustering KMeans* tal como hemos visto en el Notebook de teoría, pero esta vez utilizando los siguientes parámetros:

```
Número de clusters: 10
Método de inicialización de los puntos centrales: 'random'
Número de iteraciones para la selección de puntos centrales: 5
Algoritmo: 'elkan'
```

Visualizad gráficamente el resultado. Si los datos fuesen dispersos, ¿qué opción del parámetro *alg* de la función *sklearn.cluster.KMeans* deberíamos utilizar?

In [5]:

```
#Respuesta
```

Ejercicio 6

En los siguientes ejercicios usaremos el conjunto de datos del dataset `breast_cancer` de *sklearn*. Crea una tabla con los estadísticos descriptivos de todas las características de los tumores de la base de datos `breast_cancer` agrupados por tipo (benigno/maligno).

Realizad un test estadístico ttest para comparar el error de la simetría (*symmetry error*) y tersura (*mean smoothness*) de los tumores benignos con las de los malignos. Dado el resultado obtenido, indica qué características podrían ser más relevantes para diagnosticar un tumor maligno.

Nota: Consulta este enlace si quieres ampliar conocimientos sobre el test estadístico ttest. Para realizar el test estadístico automáticamente podéis utilizar el método `ttest_ind` de *Scipy.stats*

In [6]:

```
# Respuesta
```

Respuesta

Ejercicio 7

Representar mediante [boxplots] (https://en.wikipedia.org/wiki/Boxplots) como varían la textura, el area, la concavidad y la simetría de los tumores según el tipo (*maligna* / *benigna*). Recuerda ajustar los parámetros de la visualización para facilitar la lectura e interpretación de la gráfica que generen.

Nota: Crea un dataframe con los datos de `diagnostics.data` i `diagnostics.target` para dibujar los boxplots.

In [7]:

```
# Importamos las librerías

import numpy as np
import pandas as pd
import seaborn as sns  # librería opcional
import matplotlib.pyplot as plt
from sklearn import datasets

# Importamos el dataset
diagnostics = datasets.load_breast_cancer()
```

In [8]:

```
## Respuesta
```

Ejercicio 8

El objetivo de un modelo de **regresión lineal** es encontrar una relación entre una o más características (variables independientes) y una variable objetivo continua (variable dependiente). Cuando sólo usamos una característica predictiva se le llama **Regresión Lineal Univariada** y si hay múltiples predictores se llama **Regresión Lineal Múltiple**.

Cuando la variable dependiente es una variable binaria que contiene datos codificados como 1 (sí, éxito, etc.) o 0 (no, fallo, etc.), como es el caso del dataset `breast_cancer`, tendremos que emplear una regresión logística.

Crea un modelo de regresión logística que estime la probabilidad de que un tumor sea maligno dada su área y su textura. Muestra la matriz de confusión del modelo obtenido.

Nota: Quizás te interesa explorar la función sklearn.metrics.confusion_matrix()

In [9]:

```
# Respuesta
```

Ejercicio 9

El **Multi-layer Perceptron (MLP)** es un algoritmo de aprendizaje supervisado que aprende una función mediante el entrenamiento de un modelo que asocia n dimensiones de entrada (*predictores*) con dimensiones de salida (*targets*). Teniendo en cuenta un conjunto de funciones y un objetivo, puede aprender un aproximador de funciones no lineales por clasificación o regresión. Es diferente de la regresión logística, ya que entre la capa de entrada y la de salida, puede haber una o más capas no lineales, llamadas capas ocultas.

Aplica un clasificador basado en un MLP para predecir los tipos de tumor utilizando el área, la textura, la simetría y la concavidad, como atributos y utilizando el 70% de las muestras de entrenamiento y el 30% de test. Debéis utilizar la función sklearn.neural_network.MLPClassifier.

Utiliza tres niveles (*layers*) con 30 neuronas por nivel. ¿Qué valor de precisión obtenemos en un modelo basado en un MLP?

In [10]:

```
# Respuesta
```

Ejercicio 10

Aplicad un clasificador basado en un árbol de decicisión de un máximo de 3 niveles de profundidad para predecir los tipos de

tumor utilizando el area, la textura y la simetría como atributos y utilizando 60% de las muestras de entrenamiento y el 40% de test. Debéis utilizar la función sklearn.tree.DecisionTreeClassifier.

¿Qué valor de precisión obtenemos en un modelo basado en un árbol de decisión? Representa el árbol de decisión y expórtalo a un archivo PDF. Explora el árbol resultante al PDF y responde a la siguiente pregunta: Dados estos datos, cómo diagnosticarías a un paciente que presenta un tumor de area 400, simetría de 0.18, y un valor de textura de 19 puntos?

Nota: Tal vez la función tree de *sklearn* sea de utilidad.

In [11]:

```
# Respuesta
```

Respuesta

Soluciones ejercicios para practicar

Pregunta 1

Investiga y describe el funcionamiento del método de cross-validation *Leave One Out*. Pon un ejemplo de aplicación del algoritmo para resolver un problema de análisis en el ámbito de la salud. Recordad que hay que citar las referencias consultadas para responder la pregunta, y que la respuesta que proporcionéis debe ser original (redactada por vosotros mismos, después de haber leído y entendido las referencias que consideréis oportunas).

Respuesta

La validación cruzada Leave-one-out es esencialmente una estimación del rendimiento de generalización de un modelo entrenado en 🅰🅱 – 1 muestras de datos, que generalmente es una estimación ligeramente pesimista del rendimiento de un modelo entrenado en 🅰🅱 muestras. Es mejor pensar en la validación cruzada como una forma de estimar el rendimiento de generalización de los modelos generados por un procedimiento particular, en lugar del modelo en sí. Por ejemplo, imaginemos que estamos creando un modelo predictivo del tipo del tipo de melanoma (alto o bajo riesgo) y tenemos una muestra relativamente pequeña (1000 observaciones) que incluyen 3 parámetros descriptivos del melanoma y su nivel de riesgo (alto o bajo). Podemos ajustar el modelo usando 999 de las 1000 observaciones para después evaluar su rendimiento dada la observación que hemos dejado fuera del set de entrenamiento. En este caso, podemos repetir este método hasta 999 veces, para estimar un redimiento medio del modelo.

Pregunta 2

Debemos evitar evaluar los modelos con los mismos datos que se han utilizado para el aprendizaje. De no hacerlo, podríamos favorecer el problema del **sobre-ajuste**. ¿En qué consiste este problema y qué consecuencias puede tener?

Respuesta

El exceso de ajuste de un modelo es un error de modelado que se produce cuando una función se ajusta demasiado a un conjunto limitado de puntos de datos. Por tanto, sólo supone un problema en los modelos de aprendizaje supervisado. Una de las consecuencias del sobre-ajuste es que el modelo toma una forma demasiado compleja para explicar la idiosincrasia a los datos objeto de estudio y dificilmente generaliza a nuevas muestras de datos. La **generalización** es un término utilizado para describir la capacidad del modelo de reaccionar ante los nuevos datos. Es decir, después de ser entrenado en un entrenamiento, un modelo puede digerir datos nuevos y hacer predicciones precisas. La capacidad del modelo para generalizarse es esencial para el éxito de un modelo. Si un modelo se ha formado demasiado bien sobre datos de formación, no podrá generalizarse. Hará predicciones inexactas cuando se den nuevos datos, y hará que el modelo sea inútil aunque sea capaz de hacer predicciones precisas para los datos de formación. De eso se llama sobre-ajuste (*overfitting*).

Uno de los métodos más utilizados para evaluar el problema del sobre-ajuste llama **cross-validation**. Es un modelo de validación de modelos para evaluar cómo se generalizan los resultados del análisis estadístico (modelo) a un conjunto de datos independiente. Se utiliza principalmente en los parámetros donde el objetivo es la predicción y se quiere estimar con exactitud como comportará un modelo predictivo a la práctica. En cuanto hacemos cross-validación, destinaremos una parte de los datos a hacer el entrenamiento del modelo y la otra parte la reservaremos para evaluar el rendimiento del modelo aprendido.

Ejercicio 1

Cargad el conjunto de datos Iris incorporado en la librería `sklearn`. Implementad una función, `describe_iris`, que devuelva un diccionario con la siguiente estructura:

```
{
   "categorias": [],
   "atributos": [],
   "num_muestras": 0
}
```

categorias debe ser un array con el nombre de los **targets** del dataset. *atributos* debe ser un array con el nombre de los **atributos** y finalmente, *num_muestras* debe indicar el número **total de muestras** del dataset.

In [12]:

```
from sklearn import datasets

#Cargamos el dataset de iris

iris = datasets.load_iris()

def describe_iris():
    #Inicializamos el diccionario
    diccionario = {}
    #Asignamos a cada atributo el valor correspondiente en base al dataset Iris
    diccionario["categorias"] = iris.target_names
    diccionario["atributos"] = iris.feature_names
    diccionario["num_muestras"] = len(iris.data)

    return diccionario

print (describe_iris())
```

```
{'categorias': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'), 'atributos': ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)'], 'num_muestras': 150}
```

Ejercicio 2

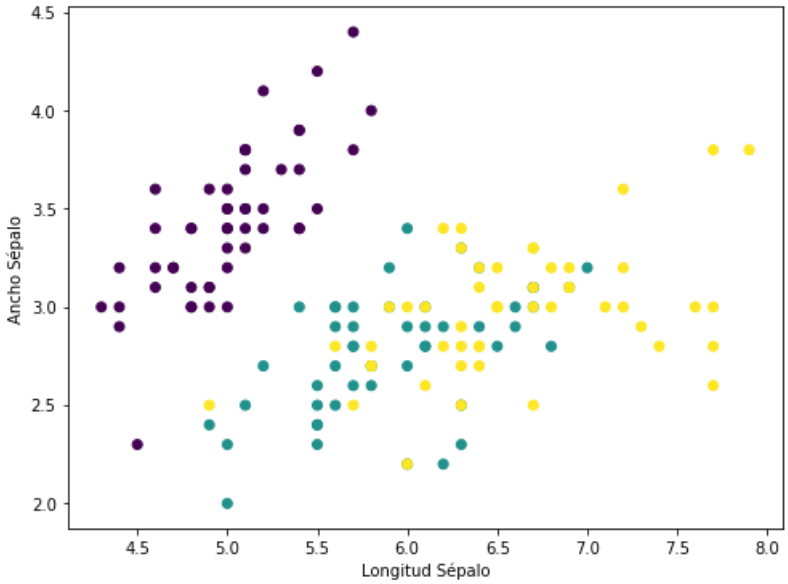
Representad gráficamente en un scatter plot la longitud de los sépalos frente al ancho de los sépalos. Nota: para poder incluir acentos en los textos de las etiquetas o del título del plot, es necesario indicar explícitamente que las cadenas de caracteres son unicode. Podéis hacerlo incluyendo una u delante de las comillas que delimitan la cadena de caracteres.

In [13]:

```
%matplotlib inline

#Importamos las librerías

import matplotlib.pyplot as plt
from sklearn import datasets

# Importamos el dataset

iris=datasets.load_iris()

L_sep = iris.data[:,0]

W_sep= iris.data[:,1]

Y = iris.target

# Creamos la figura

plt.figure(1, figsize=(8,6))

plt.clf()

# Coloreamos utilizando la categoría.

plt.scatter(L_sep,W_sep,c=Y)
plt.xlabel(u'Longitud Sépalo')
plt.ylabel(u'Ancho Sépalo')
```

Out[13]:

```
Text(0, 0.5, 'Ancho Sépalo')
```



Ejercicio 3

Dividir los datos Iris en dos subconjuntos, datos de entrenamiento y test, en una proporción 70% entrenamiento y 30% test.

In [14]:

```
from sklearn.model_selection import train_test_split


iris=datasets.load_iris()

# Dividimos los datos en datos de entrenamiento (70%) y de test (30%) mediante la función
train_test_split

train,test = train_test_split(iris.data, test_size=0.3)

# El atributo test_size=0.3 divide los datos en una proporción de 70% y 30%. entrenamiento=70% y t
est=30%.

print(train.shape)

print(test.shape)

print(test)
```

```
(105, 4)
(45, 4)
[[6.1 2.6 5.6 1.4]
 [6.8 3.  5.5 2.1]
 [5.1 2.5 3.  1.1]
 [6.3 2.5 5.  1.9]
 [4.8 3.1 1.6 0.2]
 [4.4 3.  1.3 0.2]
 [6.4 2.8 5.6 2.1]
 [5.4 3.9 1.3 0.4]
 [6.  2.9 4.5 1.5]
 [5.6 2.8 4.9 2. ]
 [7.4 2.8 6.1 1.9]
 [4.5 2.3 1.3 0.3]
 [5.9 3.2 4.8 1.8]
 [5.4 3.9 1.7 0.4]
 [5.  3.4 1.6 0.4]
 [5.6 2.9 3.6 1.3]
 [5.  3.6 1.4 0.2]
 [7.2 3.2 6.  1.8]
```

```
[5.2 4.1 1.5 0.1]
[6.7 3.1 5.6 2.4]
[6.1 2.8 4.7 1.2]
[4.9 3.1 1.5 0.1]
[5.4 3.7 1.5 0.2]
[6.4 2.7 5.3 1.9]
[5.8 2.6 4.  1.2]
[5.7 3.8 1.7 0.3]
[5.1 3.5 1.4 0.3]
[6.3 3.3 6.  2.5]
[5.  2.  3.5 1. ]
[6.7 2.5 5.8 1.8]
[6.4 3.2 4.5 1.5]
[6.4 3.2 5.3 2.3]
[5.2 3.4 1.4 0.2]
[6.3 2.3 4.4 1.3]
[5.7 2.9 4.2 1.3]
[5.4 3.  4.5 1.5]
[6.5 3.  5.8 2.2]
[4.8 3.  1.4 0.1]
[6.3 2.9 5.6 1.8]
[6.9 3.2 5.7 2.3]
[6.3 2.8 5.1 1.5]
[6.9 3.1 5.4 2.1]
[4.4 3.2 1.3 0.2]
[6.4 3.1 5.5 1.8]
[5.1 3.3 1.7 0.5]]
```

Ejercicio 4

Aplicad el clasificador **KNeighborsClassifier** para predecir el tipo de especie de iris utilizando la longitud y ancho de los pétalos como atributos y utilizando 60% de las muestras de aprendizaje y 40% muestras de test (podéis usar cualquier partición de muestras de aprendizaje y de test).

¿Qué rendimiento se obtiene?

In [15]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets

#Cargamos los datos

iris = datasets.load_iris()

iris_df = pd.DataFrame(iris.data)

# Asignamos categoría a cada una de les especies
iris_df['Species'] = iris.target
iris_df.loc[(iris_df.Species == 0),"Species"] = iris.target_names[0]
iris_df.loc[(iris_df.Species == 1),"Species"] = iris.target_names[1]
iris_df.loc[(iris_df.Species == 2),"Species"] = iris.target_names[2]

iris_df.columns = ['SepalLength', 'SepalWitdh', 'PetalLength','PetalWitdh','Species']
```

In [16]:

```
# Respuesta

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings('ignore')

# Dividimos los datos en datos de entrenamiento (60%) y de test (40%) mediante la función
train_test_split

train,test = train_test_split(iris_df, test_size=0.4)

# Creamos el clasificador

knn = KNeighborsClassifier()
```

```
knn = KNeighborsClassifier()

#Entrenamos el clasificador
knn.fit( train[['PetalLength','PetalWitdh']],train[['Species']])

#Probamos el clasificador
iris_predicted=knn.predict(test[['PetalLength','PetalWitdh']])

#Mostramos los resultados de la predicción sobre el conjunto de test

print ("Accuracy: \t\t" + str(knn.score(test[['PetalLength','PetalWitdh']],test[['Species']])))
```

```
Accuracy:   0.95
```

Ejercicio 5

Aplicad el algoritmo de *clustering KMeans* tal como hemos visto en el Notebook de explicación, pero esta vez utilizando los siguientes parámetros:

```
Número de clusters: 10
Método de inicialización de los puntos centrales: 'random'
Número de iteraciones para la selección de puntos centrales: 5
Algoritmo: 'elkan'
```

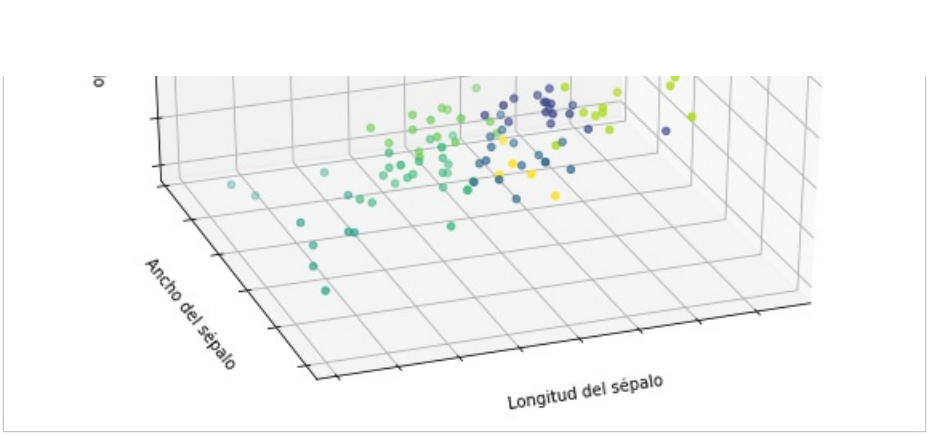
Visualizad gráficamente el resultado. Si los datos fuesem dispersos , ¿qué opción del parámetro *alg* de la función *sklearn.cluster.KMeans* deberíamos utilizar?

In [17]:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import cluster, datasets

iris = datasets.load_iris()
X_iris = iris.data

#Consultamos la ayuda de K-means para ver como se definen los parámetros
?cluster.KMeans

# Cargamos el algoritmo K-means y ajustamos los paràmetros según el enunciado
k_means = cluster.KMeans(n_clusters=10,init='random',n_init=5, algorithm='elkan')
k_means.fit(X_iris)

#Definimos los parámetros de la figura
fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
ax.scatter(X_iris[:,[0]], X_iris[:,[1]], X_iris[:,[2]], c=k_means.labels_)
ax.set_title(u"Taxonomía de las 150 muestras utilizando K-means")
ax.set_xlabel(u"Longitud del sépalo")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel(u"Ancho del sépalo")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel(u"Longitud del pétalo")
ax.w_zaxis.set_ticklabels([])
```

Out[17]:

```
[]
```



Respuesta

Si los datos fuesen dispersos, el parámetro *algorithm* debería ser 'full'.

Ejercicio 6

En los siguientes ejercicios usaremos el conjunto de datos del dataset `breast_cancer` de *sklearn*. Crea una tabla con los estadísticos descriptivos de todas las características de los tumores de la base de datos `breast_cancer` agrupados por tipo (benigno/maligno).

Realizad un test estadístico ttest para comparar el error de la simetría (*symmetry error*) y tersura (*mean smoothness*) de los tumores benignos con las de los malignos. Dado el resultado obtenido, indica qué características podrían ser más relevantes para diagnosticar un tumor maligno.

Nota: Consulta este enlace si quieres ampliar conocimientos sobre el test estadístico ttest. Para realizar el test estadístico automáticamente podéis utilizar el método `ttest_ind` de *Scipy.stats*

In [18]:

```
# Respuesta

# Importamos las librerías
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
import scipy.stats as stats

# Importamos el dataset
diagnostics = datasets.load_breast_cancer()

# Los transformamos en un dataframe
diagnostics_df = pd.DataFrame(diagnostics.data)

# Nombramos las columnas de las variables
diagnostics_df.columns = diagnostics['feature_names']

# Añadimos la columna Type con el resultado del diagnóstico (benigno/maligno)
diagnostics_df['Type'] = diagnostics.target

# Mostramos pos pantalla los parámetros descriptivos de la muestra
diagnostics_df.groupby('Type').describe()
```

Out[18]:

	mean radius								mean texture		...	worst symmetry		worst fractal dimension		
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std
Type																
0	212.0	17.462830	3.203971	10.950	15.075	17.325	19.59	28.11	212.0	21.604906	...	0.359225	0.6638	212.0	0.091530	0.02
1	357.0	12.146524	1.780512	6.981	11.080	12.200	13.37	17.85	357.0	17.914762	...	0.298300	0.4228	357.0	0.079442	0.013

2 rows × 240 columns

In [19]:

```
# Realizamos un test estadístico para cada variable comparando los tumores benignos (tipo 0) y mal
ignos (tipo 1).
# Ejecutamos un prueba t de student para muestras independientes ya que cada registro corresponde
con un paciente diferente.
t1, p1 = stats.ttest_ind(diagnostics_df.query("Type==0")['mean
smoothness'],diagnostics_df.query("Type==1")['mean smoothness'])
print("Análisis de la tersura. Estadístico t= " + str(t1) + ", p-val = " + str(p1) )
t2, p2 = stats.ttest_ind(diagnostics_df.query("Type==0")['symmetry
error'],diagnostics_df.query("Type==1")['symmetry error'])
print("Análisis del error de la simetría. Estadístico t= " + str(t2) + ", p-val = " + str(p2) )
```

```
Análisis de la tersura. Estadístico t= 9.146098808149038, p-val = 1.0518503592032013e-18
Análisis del error de la simetría. Estadístico t= -0.1552978000059369, p-val = 0.8766418183858812
```

Respuesta

Preferiremos la variable tersura como diferenciadora de los tumores benignos y malignos dado que el test sugiere que ambas muestras de lisura comparadas provienen de poblaciones diferentes.

Ejercicio 7

Representar mediante [boxplots] (https://en.wikipedia.org/wiki/Boxplots) como varían la textura, el area, la concavidad y la simetría de los tumores según el tipo (*maligna* / *benigna*). Recuerda ajustar los parámetros de la visualización para facilitar la lectura e interpretación de la gráfica que generen.

Nota: Crea un dataframe con los datos de `diagnostics.data` i `diagnostics.target` para dibujar los boxplots.

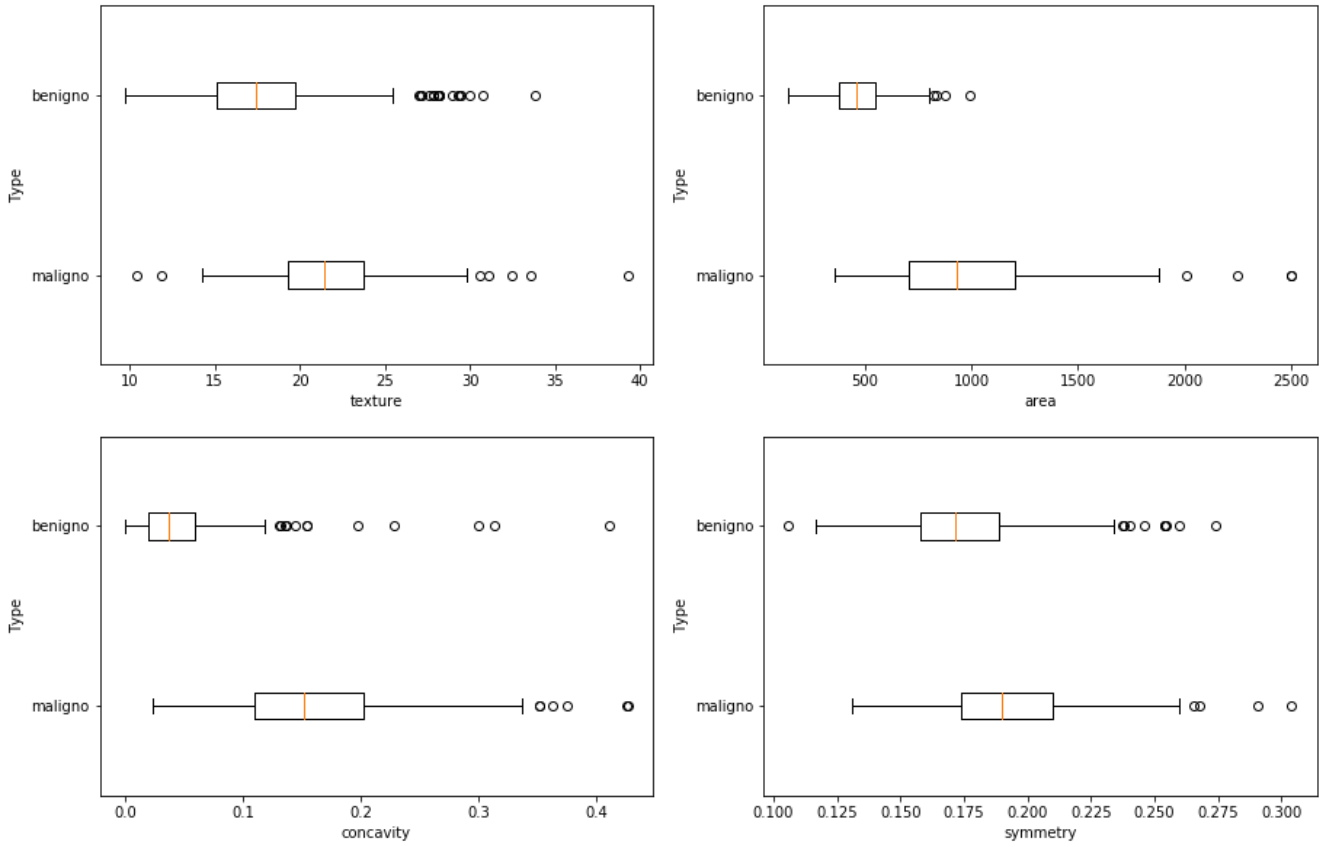
In [20]:

```
# Importamos las librerías

import numpy as np
import pandas as pd
import seaborn as sns  # librería opcional
import matplotlib.pyplot as plt
from sklearn import datasets

# Importamos el dataset
diagnostics = datasets.load_breast_cancer()
```

In [21]:

```
## Respuesta
diagnostics_df = pd.DataFrame(diagnostics.data)

# Asignamos una categoría a cada tumor
diagnostics_df.columns = diagnostics['feature_names']
diagnostics_df['Type'] = diagnostics.target
diagnostics_df.loc[(diagnostics_df.Type == 0),"Type"] = diagnostics.target_names[0]
diagnostics_df.loc[(diagnostics_df.Type == 1),"Type"] = diagnostics.target_names[1]

# Definimos parámetros de la figura
label=['maligno','benigno']
pos=[1,2]
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 10))

# Dibujamos cada uno de los gráficos
vplot1=axes[0,0].boxplot([np.array(diagnostics_df['mean texture'][diagnostics_df.Type == 'malignant
']),
                          np.array(diagnostics_df['mean texture'][diagnostics_df.Type == 'benign']
                                   )],vert=False)
axes[0,0].set_xlabel('texture')
axes[0,0].set_ylabel('Type')
axes[0,0].set_yticks(pos)
axes[0,0].set_yticklabels(label)

vplot2=axes[0,1].boxplot([np.array(diagnostics_df['mean area'][diagnostics_df.Type == 'malignant'])
```

```
,
                            np.array(diagnostics_df['mean area'][diagnostics_df.Type == 'benign'],
                                    )],vert=False)
axes[0,1].set_xlabel('area')
axes[0,1].set_ylabel('Type')
axes[0,1].set_yticks(pos)
axes[0,1].set_yticklabels(label)

vplot3=axes[1,0].boxplot([np.array(diagnostics_df['mean concavity'][diagnostics_df.Type == 'maligna
nt']),
                            np.array(diagnostics_df['mean concavity'][diagnostics_df.Type == 'benign
],
                                    )],vert=False)
axes[1,0].set_xlabel('concavity')
axes[1,0].set_ylabel('Type')
axes[1,0].set_yticks(pos)
axes[1,0].set_yticklabels(label)

vplot4=axes[1,1].boxplot([np.array(diagnostics_df['mean symmetry'][diagnostics_df.Type == 'malignan
t']),
                            np.array(diagnostics_df['mean symmetry'][diagnostics_df.Type == 'benign'
,
                                    )],vert=False)
axes[1,1].set_xlabel('symmetry')
axes[1,1].set_ylabel('Type')
axes[1,1].set_yticks(pos)
axes[1,1].set_yticklabels(label)
colors = ['forestgreen', 'darkorange']


# Assignem color a cada un dels cossos del boxplot
plt.show()
```



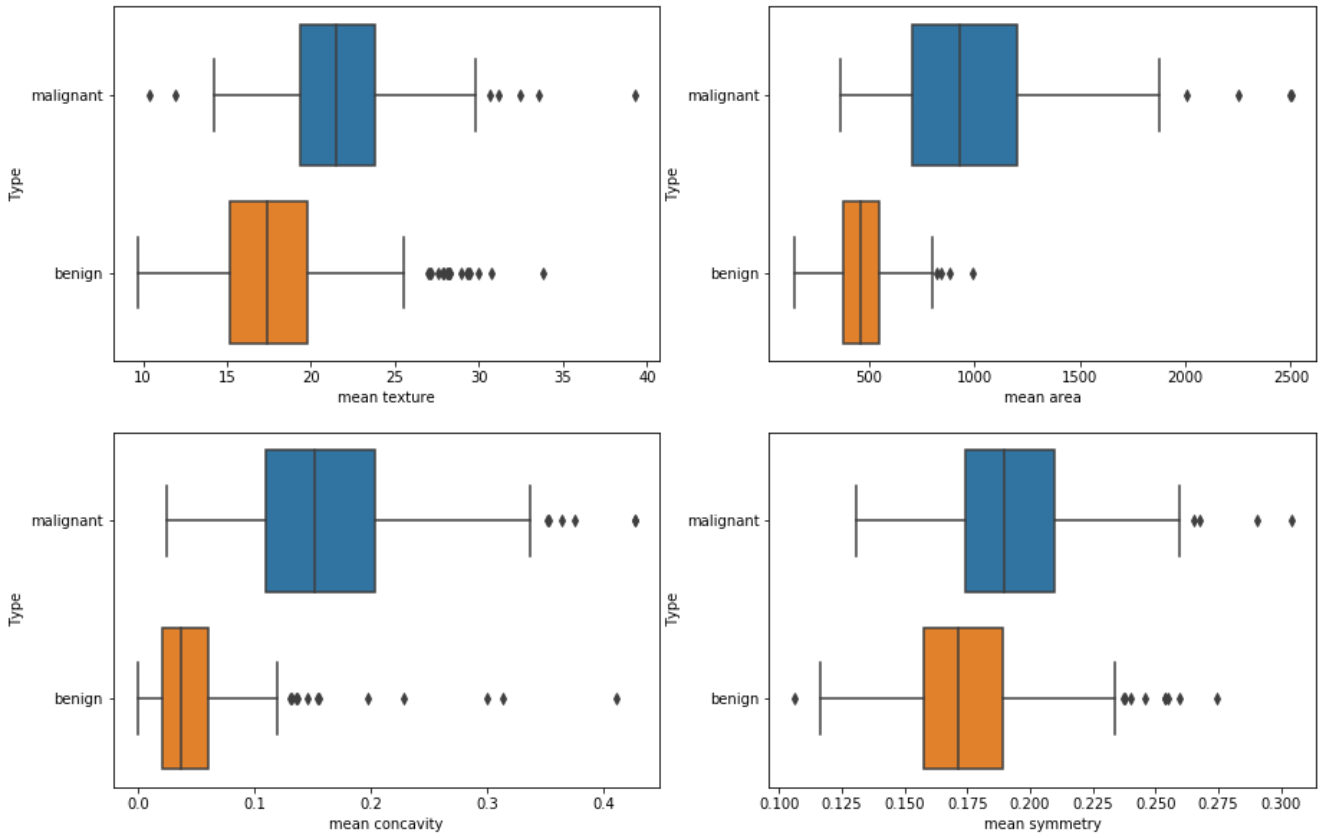
In [22]:

```
#Opción 2: Otra forma de realizar el ejercicio es mediante la librería
# Seaborn, que veremos en la próxima unidad

# Importamos las librerías
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from sklearn import datasets

# Importamos el dataset
diagnostics = datasets.load_breast_cancer()
diagnostics_df = pd.DataFrame(diagnostics.data)

# Asignamos una categoría a cada tumor
diagnostics_df.columns = diagnostics['feature_names']
diagnostics_df['Type'] = diagnostics.target
diagnostics_df.loc[(diagnostics_df.Type == 0),"Type"] = diagnostics.target_names[0]
diagnostics_df.loc[(diagnostics_df.Type == 1),"Type"] = diagnostics.target_names[1]

# Dibujamos cada una de las gráficas
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.boxplot(x=diagnostics_df['mean texture'],y=diagnostics_df.Type)
plt.subplot(2,2,2)
sns.boxplot(x=diagnostics_df['mean area'],y=diagnostics_df.Type)
plt.subplot(2,2,3)
sns.boxplot(x=diagnostics_df['mean concavity'],y=diagnostics_df.Type)
plt.subplot(2,2,4)
sns.boxplot(x=diagnostics_df['mean symmetry'],y=diagnostics_df.Type)
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1b5dde50>
```



Ejercicio 8

El objetivo de un modelo de **regresión lineal** es encontrar una relación entre una o más características (variables independientes) y una variable objetivo continua (variable dependiente). Cuando sólo usamos una característica predictiva se le llama **Regresión Lineal Univariada** y si hay múltiples predictores se llama **Regresión Lineal Múltiple**.

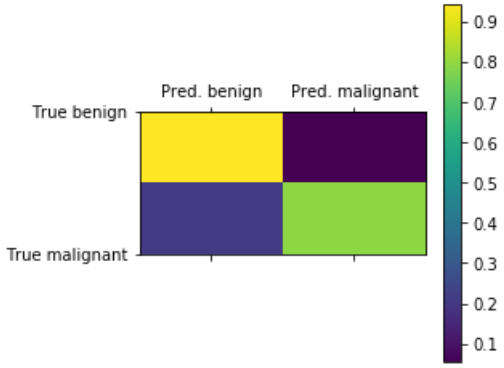
Cuando la variable dependiente es una variable binaria que contiene datos codificados como 1 (sí, éxito, etc.) o 0 (no, fallo, etc.), como es el caso del dataset `breast_cancer`, tendremos que emplear una regresión logística.

Crea un modelo de regresión logística que estime la probabilidad de que un tumor sea maligno dada su área y su textura. Muestra la matriz de confusión del modelo obtenido.

Nota: Quizás te interesa explorar la función sklearn.metrics.confusion_matrix()

In [23]:

```
# Respuesta
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X = diagnostics_df[['mean texture','mean area']]
y = diagnostics_df['Type']

# Dividimos los datos en datos de entrenamiento (60%) y de test (40%) mediante la función
train_test_split
train,test = train_test_split(diagnostics_df, test_size=0.4)
train_X = train[['mean texture','mean area']]
train_y = train['Type']
test_X = test[['mean texture','mean area']]
test_y = test['Type']

# Aplicamos la regresión
clf = LogisticRegression(random_state=0, solver='lbfgs').fit(train_X, train_y)
clf.score(test_X,test_y)

# Predecimos el tipo de cáncer dados los predictores Textura y Área
predicted = clf.predict( diagnostics_df[['mean texture','mean area']].values )
real      = diagnostics_df['Type'].values

# Representamos la matriz de confusión
import matplotlib.pyplot as plt
result_cofusion_mat = []
for real_c in ['benign','malignant']:
    tmp = []
    for pred_c in ['benign','malignant']:
        # Dividim per el nombre total de casos per calcular al proporcio de:
        # vertaders positius, vertaders negatius, falsos positius, i falsos negatius
        tmp.append( np.where(np.logical_and(predicted==pred_c, real==real_c))[0].size /np.where(rea
l==real_c)[0].size)
    result_cofusion_mat.append(tmp)

plt.matshow(result_cofusion_mat)
plt.xticks( [0,1], ['Pred. benign','Pred. malignant'] )
plt.yticks( [0,1], ['True benign','True malignant'] )
plt.colorbar()
plt.show()
```



Ejercicio 9

El **Multi-layer Perceptron (MLP)** es un algoritmo de aprendizaje supervisado que aprende una función mediante el entrenamiento de un modelo que asocia n dimensiones de entrada (*predictores*) con dimensiones de salida (*targets*). Teniendo en cuenta un conjunto de funciones y un objetivo, puede aprender un aproximador de funciones no lineales por clasificación o regresión. Es diferente de la regresión logística, ya que entre la capa de entrada y la de salida, puede haber una o más capas no lineales, llamadas capas ocultas.

Aplica un clasificador basado en un MLP para predecir los tipos de tumor utilizando el área, la textura, la simetría y la concavidad, como atributos y utilizando el 70% de las muestras de entrenamiento y el 30% de test. Debéis utilizar la función sklearn.neural_network.MLPClassifier.

Utiliza tres niveles (*layers*) con 30 neuronas por nivel. ¿Qué valor de precisión obtenemos en un modelo basado en un MLP?

In [24]:

```
# Respuesta
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

# Dividimos los datos en datos de entrenamiento (70%) y de test (30%) mediante la función
train_test_split
train,test = train_test_split(diagnostics_df, test_size=0.3)
train_X = train[['mean texture','mean area','mean symmetry','mean concavity']]
train_y = train['Type']
test_X = test[['mean texture','mean area','mean symmetry','mean concavity']]
test_y = test['Type']

# Aplicamos el MLP
clf = MLPClassifier(hidden_layer_sizes=(3, 30), max_iter=1000)
clf.fit(train_X, train_y)

# Mostramos su precisión
score = clf.score(test_X, test_y)
print("El modelo tiene un " + str(np.around(score,2)*100) + '% de precisión en predecir el tipo de
tumor.' )
```

```
El modelo tiene un 80.0% de precisión en predecir el tipo de tumor.
```

Ejercicio 10

Aplicad un clasificador basado en un árbol de decicisión de un máximo de 3 niveles de profundidad para predecir los tipos de tumor utilizando el area, la textura y la simetría como atributos y utilizando 60% de las muestras de entrenamiento y el 40% de test. Debéis utilizar la función sklearn.tree.DecisionTreeClassifier.

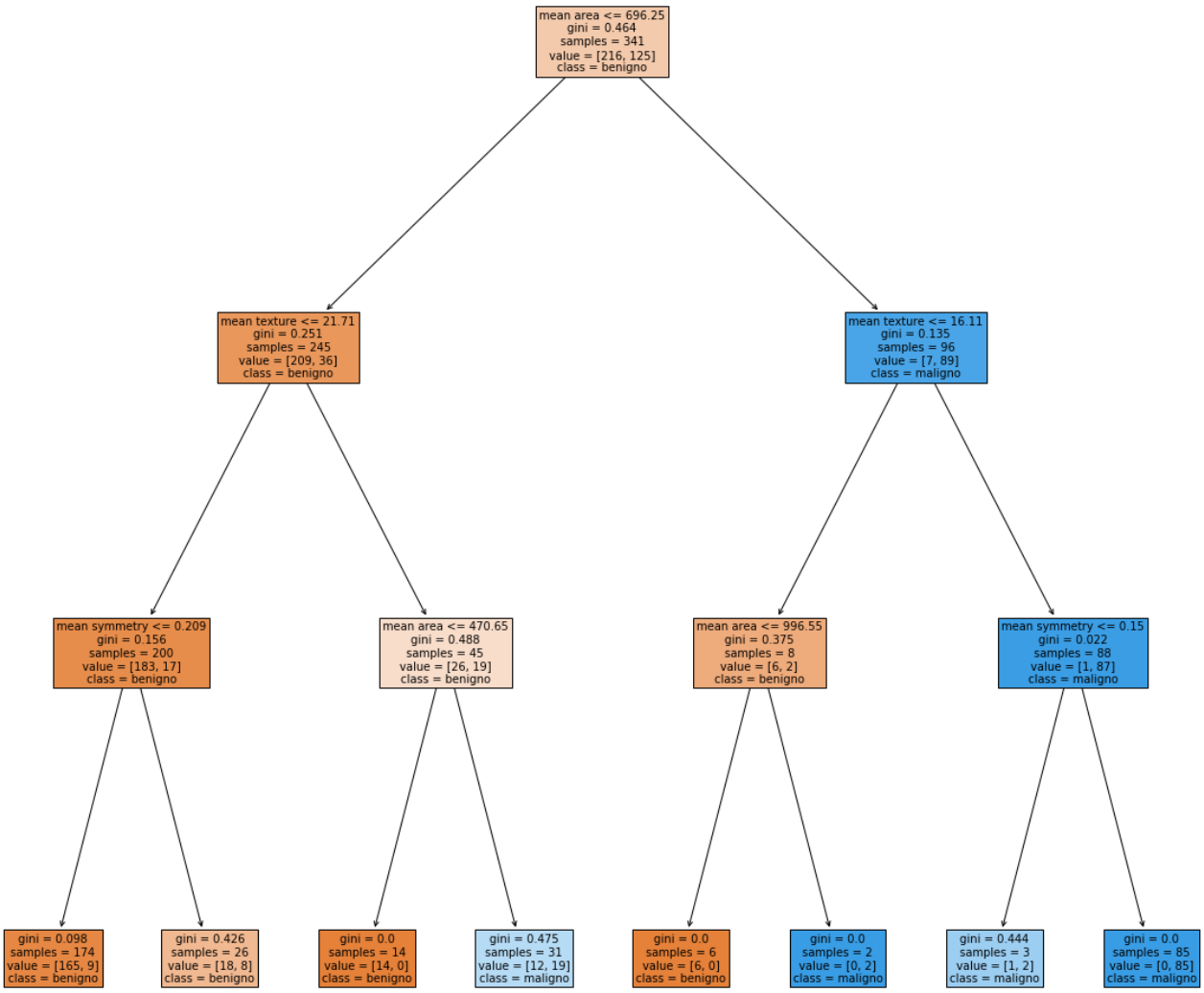
¿Qué valor de precisión obtenemos en un modelo basado en un árbol de decisión? Representa el árbol de decisión y expórtalo a un archivo PDF. Explora el árbol resultante al PDF y responde a la siguiente pregunta: Dados estos datos, cómo diagnosticarías a un paciente que presenta un tumor de area 400, simetría de 0.18, y un valor de textura de 19 puntos?

Nota: Tal vez la función tree de *sklearn* sea de utilidad.

In [25]:

```
# Respuesta
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import tree

# Dividimos los datos en datos de entrenamiento (60%) i de test (40%) mediante la función
train_test_split
train,test = train_test_split(diagnostics_df, test_size=0.40)

# Creamos el clasificador
dtree = DecisionTreeClassifier(max_depth=3)

# Entrenamos el clasificador
fit = dtree.fit(train[['mean area','mean symmetry', 'mean texture']],train[['Type']])

# Evaluamos la precisión del clasificador
score = dtree.score(test[['mean area','mean symmetry', 'mean texture']],test[['Type']])
print("El modelo tiene un " + str(np.around(score,2)*100) + '% de precisión en predecir el tipo de
tumor.' )

# Mostramos el árbol de decisión y lo exportamos a un archivo PDF
fig = plt.figure(figsize=(20,20), facecolor='w')
tree.plot_tree(fit, filled=True, feature_names=['mean area','mean symmetry', 'mean texture'],
class_names=['benigno','maligno']);
plt.savefig('tree.pdf')
```

```
El modelo tiene un 89.0% de precisión en predecir el tipo de tumor.
```



Respuesta

Clasificaremos el tumor como beningno (correspondiente con la primera hoja del árbol) con una probabilidad de error (impureza de Gini) bastante baja.