codej1.pdf

postdata9

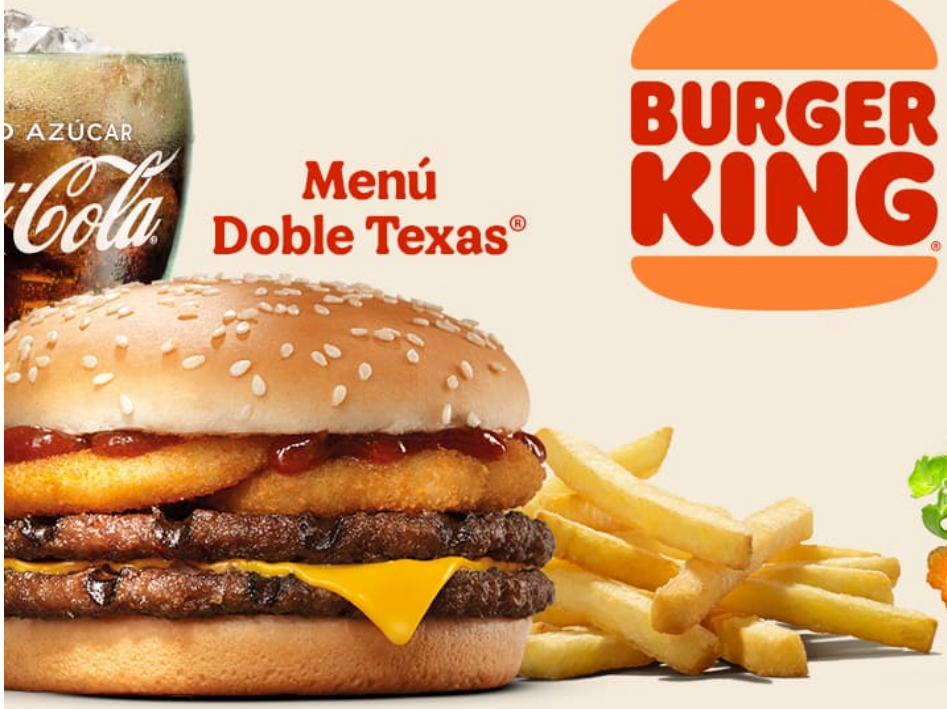
Aprendizaje Automatico

3º Grado en Ingeniería Informática

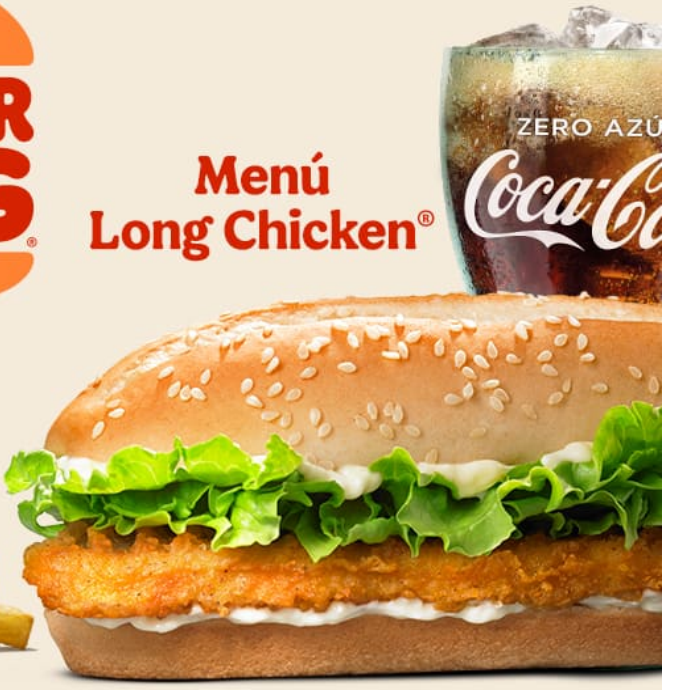
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

BURGER KING

Menú Doble Texas®

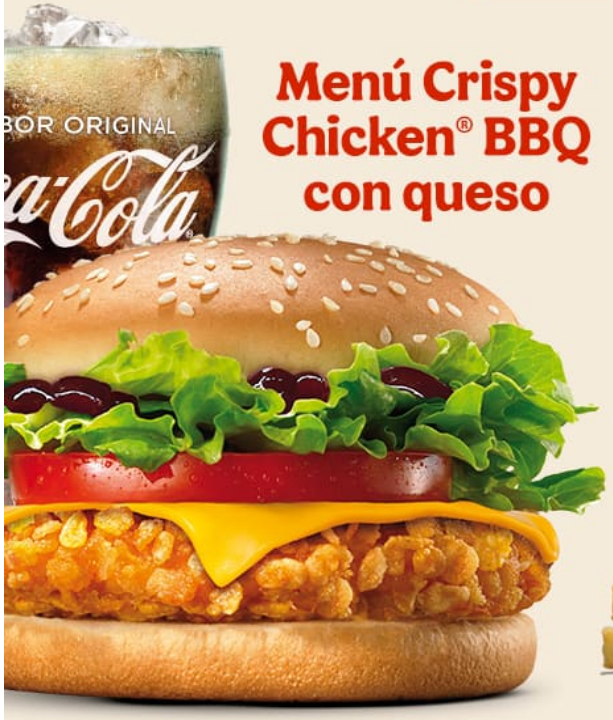


Menú Long Chicken®

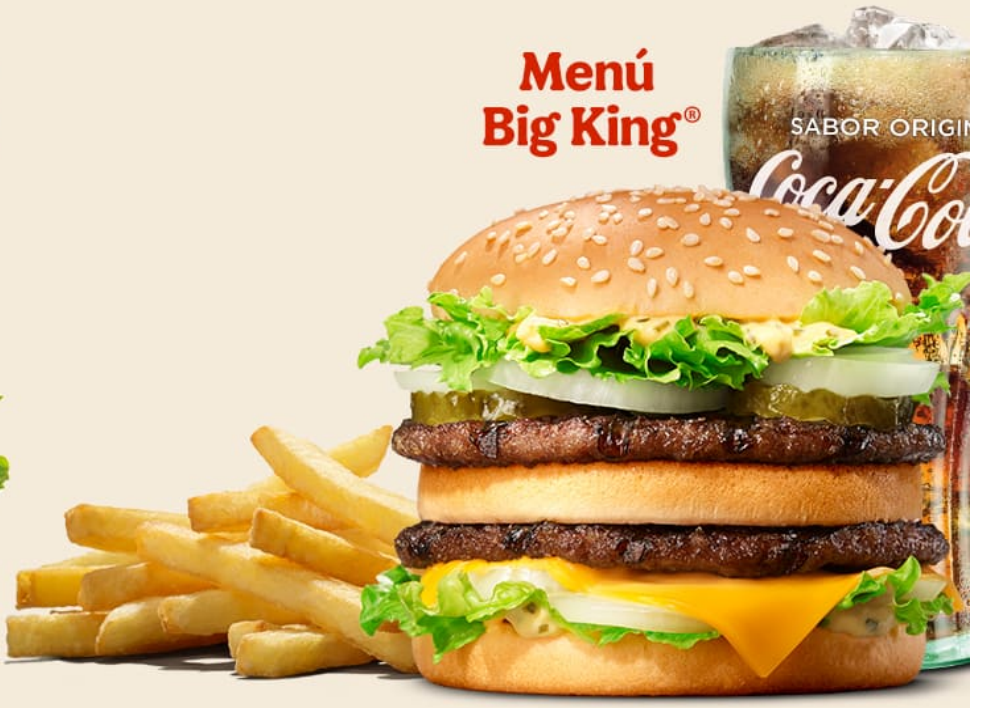


2x7€

Menú Crispy Chicken® BBQ con queso



Menú Big King®



AUTO KING

PARA LLEVAR

RESTAURANTE

Válido hasta 10/05/21. Elección de 2 menús pequeños entre menú Crispy Chicken® BBQ con queso, Big King®, Doble Texas o Long Chicken®. Por +0,50€/menú mediano, por +1€/menú grande. Patatas Supreme excepto para menús pequeños. Agua de 0,33cl en menú pequeño y 0,5cl en el resto de menús. Excluidos refrescos embotellados. Cerveza no disponible en menús pequeños. Restaurantes no adheridos en www.burgerking.es. COCA-COLA® y COCA-COLA ZERO® son marcas registradas de THE COCA-COLA COMPANY. TM Burger King Corporation. © 2021 Burger King Europe GmbH. BURGER KING® se reserva el derecho a ampliar el periodo promocional. Todos los derechos reservados.

PRÁCTICA 2:

Ejercicio 1

```python
import numpy as np
import matplotlib.pyplot as plt

print("\n")
print("############################################################\n")
print('--- EJERCICIO SOBRE LA COMPLEJIDAD DE H Y EL RUIDO ---')
input("############################################################\n")

# Fijamos la semilla
np.random.seed(1)

##################################
#                                #
#      FUNCIONES AUXILIARES      #
#                                #
##################################

#simula_unif
# --- Parámetros:
#       - N: número de vetores / tamaño de la lista de retorno
#       - dim: tamaño de cada vector
#       - rango: intervalo de los valores de los números de cada vector
#
# --- Retorno:
#       - Una lista de N vectores con tamaño dim cada vector, cuyos números
#       están en el intervalo rango
def simula_unif(N, dim, rango):
      return np.random.uniform(rango[0],rango[1],(N,dim))

#simula_gaus
# --- Parámetros:
#       - N: número de vectores / tamaño de la lista de retorno
#       - dim: tamaño de cada vector
#       - sigma: intervalo que determinará la función gaussiana, son dos valores
#            el primero es la media y el segundo la desviación típica
#
# --- Retorno:
#       Una lista de N vectores de tamaño dim con valores que estarán dentro de la
#       función gaussiana
def simula_gaus(N, dim, sigma):
    media = 0
    out = np.zeros((N,dim),np.float64)
    for i in range(N):
        # Para cada columna dim se emplea un sigma determinado. Es decir, para
        # la primera columna se usará una N(0,sqrt(5)) y para la segunda N(0,sqrt(7))
        out[i,:] = np.random.normal(loc=media, scale=np.sqrt(sigma), size=dim)

    return out

#simula_recta
# --- Parámetros:
#       - intervalo: rango del cual se extraerán los valores aleatoriamente los
#           puntos (x1,y1) y (x2,y2) que definen una recta
#
# --- Retorno:
#       Los valores de a y b, donde a es la pendiente de la recta, y
#       b, el término independiente
def simula_recta(intervalo):
    points = np.random.uniform(intervalo[0], intervalo[1], size=(2, 2))
    x1 = points[0,0]
    x2 = points[1,0]
    y1 = points[0,1]
    y2 = points[1,1]
    # y = a*x + b
    a = (y2-y1)/(x2-x1) # Calculo de la pendiente.
    b = y1 - a*x1       # Calculo del termino independiente

    return a, b
```

```
#fun_etq
#usamos la función f(x,y) = y - ax - b
# --- Parámetros:
#       - x: coordenada x del punto
#       - y: coordenada y del punto
#       - pend: constante a, pendiente de la recta
#       - term: constante b, término independiente
#
# --- Retorno:
#       Devuelve el signo de la función en el punto (x,y), con las
#       constantes a y b que determinará la etiqueta del punto (x,y)
def fun_etq(x, y, pend, term):
        return np.sign(y - pend*x - term)


#plot_datos_cuad
# --- Parámetros:
#       - X: nube de puntos, que es un vector con dos columnas (x,y)
#       - y: lista con las etiquetas con ruido
#       - fz: función a usar
#
# --- Retorno:
#       Dibuja una gráfica con los datos X cuyas etiquetas y tienen ruido
#       según la función fz
def plot_datos_cuad(X, y, fz, title='Point cloud plot', xaxis='x axis', yaxis='y
axis'):
    #Preparar datos
    min_xy = X.min(axis=0)
    max_xy = X.max(axis=0)
    border_xy = (max_xy-min_xy)*0.01

    #Generar grid de predicciones
    xx, yy = np.mgrid[min_xy[0]-border_xy[0]:max_xy[0]+border_xy[0]+0.001:border_xy[0],
                      min_xy[1]-border_xy[1]:max_xy[1]+border_xy[1]+0.001:border_xy[1]]
    grid = np.c_[xx.ravel(), yy.ravel(), np.ones_like(xx).ravel()]

    pred_y = fz(grid)
    # pred_y[(pred_y>-1) & (pred_y<1)]
    pred_y = np.clip(pred_y, -1, 1).reshape(xx.shape)

    #Plot
    f, ax = plt.subplots(figsize=(8, 6))
    contour = ax.contourf(xx, yy, pred_y, 50, cmap='PuOr',vmin=-1, vmax=1)
    ax_c = f.colorbar(contour)
    ax_c.set_label('$f(x, y)$')
    ax_c.set_ticks([-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1])
    ax.scatter(X[:, 0], X[:, 1], c=y, s=50, linewidth=2,
                cmap='Wistia')

    XX, YY = np.meshgrid(np.linspace(round(min(min_xy)),
round(max(max_xy)),X.shape[0]),np.linspace(round(min(min_xy)),
round(max(max_xy)),X.shape[0]))
    positions = np.vstack([XX.ravel(), YY.ravel()])
    #ax.contour(XX,YY,fz(positions.T).reshape(X.shape[0],X.shape[0]),[0],
colors='green', linewidth='5')

    ax.set(
       xlim=(min_xy[0]-border_xy[0], max_xy[0]+border_xy[0]),
       ylim=(min_xy[1]-border_xy[1], max_xy[1]+border_xy[1]),
       xlabel=xaxis, ylabel=yaxis)
    plt.title(title)
    plt.show()
```

```python
#funcion_a
# --- Parámetros:
#       - X: nube de puntos, que es un vector con dos columnas (x,y)
#
# --- Retorno:
#       Una lista con todos los valores que toma la función a en cada punto de X.
#       a == 0 -> f(x, y) = (x − 10)² + (y − 20)² − 400
#       a == 1 -> f(x, y) = 0,5(x + 10)² + (y − 20)² − 400
#       a == 2 -> f(x, y) = 0,5(x − 10)² − (y + 20)² − 400
#       a == 3 -> f(x, y) = y − 20x² − 5x + 3
def funcion_0(X):
    return (X[:,0] -10)**2 + (X[:,1]-20)**2 - 400

def funcion_1(X):
    return 0.5*(X[:,0]+10)**2 + (X[:,1]-20)**2 - 400

def funcion_2(X):
    return 0.5*(X[:,0]-10)**2 - (X[:,1]+20)**2 - 400

def funcion_3(X):
    return X[:,1] - 20*X[:,0]**2 - 5*X[:,0] + 3


############################
#                          #
#       Ejercicio 1        #
#                          #
############################

print("********************\n")
print('*** EJERCICIO 1 ***')
input('********************\n')

---------------------------------
|                               |
|          Apartado a)          |
|                               |
---------------------------------

input(" -- apartado a)")
print("————————————————")
print("Considerando una lista de 50 vectores, cada vector de tamaño 2, cuyos valores
estarán entre [-50,50].")

N = 50
dim = 2
intervalo = [-50,50]
int_gaus = [5,7]

#lista de vectores con números aleatorios dentro del intervalo [-50,50]
unif = simula_unif(N, dim, intervalo)

#lista de vectores con números aleatorios dentro del intervalo de gauss
gauss = simula_gaus(N, dim, int_gaus)

#Pintamos la gráfica
plt.title("1.a Gráfica de nube de puntos:")

plt.scatter(unif[:,0], unif[:,1], c='purple', label='Nube de puntos uniforme')
plt.scatter(gauss[:,0], gauss[:,1], c='orange', label='Nube de puntos de Gauss')

plt.xlabel("Intervalo del eje x [-50, 50]")
plt.ylabel("Intervalo del eje y [-50, 50]")

plt.legend()
plt.show()
```

```
############################
#                          #
#       Ejercicio 2        #
#                          #
############################

print("\n\n")
print("********************\n")
print('*** EJERCICIO 2 ***')
input('********************')

---------------------------------
|                               |
|          Apartado a)          |
|                               |
---------------------------------

input(" -- apartado a)")
print("_________________")

#obtenemos las constantes de la recta en el intervalo [-50, 50]
a, b = simula_recta(intervalo)


# ---- PRIMERA FORMA DE HACER EL APARTADO A ----

# - Para pintar la gráfica,realicé un for que recorriera la lista unif y fuera pintando
#   los puntos, pero me encontré con el problema que la leyenda se componía de 50
#   elementos, me dibujaba una en cada iteración.

# - Para solucionar el problema,realicé un for que contabilizara el número de elementos
#   que tenían etiqueta 1 y -1 para poder dibujar la etiqueta cuando fuera a pintar el
#   último punto de la etiqueta.


#para poder dibujar la leyenda en la gráfica, tal y como lo tengo implementado
#al poner la leyenda sale tantas veces como puntos dibujo, por ello, voy a contabilizar
#el número de elementos que hay con etiquetas -1 y 1
#cuando vaya a dibujar con scatter, cuando sea el último elemento de dicha etiqueta
#será cuando dibuje la leyenda
#cont_menos = 0
#cont_uno = 0
#for j in etiq:
#    if(j == -1):
#        cont_menos = cont_menos + 1
#
#    elif(j == 1):
#        cont_uno = cont_uno + 1

#cont_1 = 0  #contador de uno
#cont__1 = 0 #contador de menos uno

#for i in range(0, len(etiq)):

#    #si la etiqueta es positiva (1.0), pintamos el punto morado
#    if(etiq[i] == 1.0):
#
#        #si vamos a pintar el último punto de la etiqueta, lo pintamos con la etiqueta
#        if(cont_1 == cont_1 - 1):

#            plt.scatter(unif[i,0], unif[i,1], c='purple', label='Etiqueta 1.0')
#            cont_1 = cont_1 + 1
#
#        #si no es el último punto, pintamos el punto e incrementamos el contador
#        else:
#            plt.scatter(unif[i,0], unif[i,1], c='purple')
```

```
#           cont_1 = cont_1 + 1
#
#    #si la etiqueta es positiva (-1.0), pintamos el punto naranja
#    elif(etiq[i] == -1.0):
#
#        #si vamos a pintar el último punto de la etiqueta, pintamos el punto con la
etiqueta
##        if(cont__1 == cont_menos - 1):

#            plt.scatter(unif[i,0], unif[i,1], c='orange', label='Etiqueta -1.0')
#            cont__1 = cont__1 + 1
#
#        #si no es el último punto, pintamos el punto e incrementamos el contador
#        else:
#            plt.scatter(unif[i,0], unif[i,1], c='orange')
#            cont__1 = cont__1 + 1



# - Después de realizar el ejercicio de esta manera, caí en que podría hacer lo mismo
que
#   antes, pero en vez de pintar la leyenda cuando fuera a dibujar el último elemento,
#   pintar la leyenda cuando fuera el primer elemento de la etiqueta, de esta forma
#   no haría falta contabilizar el número de elementos de cada clase.


#cont_1 = 0  #contador de uno
#cont__1 = 0 #contador de menos uno
#for i in range(0, len(etiq)):

#    #si la etiqueta es positiva (1.0), pintamos el punto morado
#    if(etiq[i] == 1.0):
#
#        #si vamos a pintar el primer punto de la etiqueta, lo pintamos con la etiqueta
#        if(cont_1 == 0):

#            plt.scatter(unif[i,0], unif[i,1], c='purple', label='Etiqueta 1.0')
#            cont_1 = cont_1 + 1
#
#        #si no es el primer punto, lo pintamos
#        else:
#            plt.scatter(unif[i,0], unif[i,1], c='purple')
#
#    #si la etiqueta es positiva (-1.0), pintamos el punto naranja
#    elif(etiq[i] == -1.0):
#
#        #si vamos a pintar el primer punto de la etiqueta, lo pintamos con la etiqueta
#        if(cont__1 == 0):
#            plt.scatter(unif[i,0], unif[i,1], c='orange', label='Etiqueta -1.0')
#            cont__1 = cont__1 + 1
#
#        #si no es el primer punto, lo pintamos
#        else:
#            plt.scatter(unif[i,0], unif[i,1], c='orange')
```

```
# ---- SEGUNDA FORMA DE HACER EL APARTADO A ----

# - En un array voy a insertar el signo de cada punto que hay en unif,
#   es decir, es un array de signos que contiene la etiqueta (1, -1)

#creamos un array en el que almacenamos el signo de la función en cada índice
#que depende de las constantes y del punto en unif
etiq = np.zeros(len(unif), np.float64)
cs = 0  #contador para el índice

for c in unif:

    #vamos añadiendo en etiq el signo de la función f(x,y) = y - ax - b
    #la x es la columna 0 de unif y la y la columna 1
    etiq[cs]= (fun_etq(c[0], c[1], a, b))
    cs = cs + 1


# - Para dibujar la gráfica de forma indexada, me creo 2 listas. En una
#   almaceno todos los puntos cuya etiqueta sea -1 y en la otra, los puntos
#   cuya etiqueta sea +1
et_neg = unif[etiq == -1.0]
et_pos = unif[etiq == 1.0]

#vamos a dibujar en una gráfica los datos según su etiqueta
plt.title("2.a Gráfica de puntos según la etiqueta")


# - Dibujo la gráfica con la forma indexada. Aquellos puntos con etiqueta negativa
#   los dibujo morado, aquellos con etiqueta positiva de color naranja
plt.scatter(et_neg[:, 0], et_neg[:, 1], c='purple', label='Etiqueta -1.0')
plt.scatter(et_pos[:, 0], et_pos[:, 1], c='orange', label='Etiqueta 1.0')


# - Dibujo la recta divisora que me divide en la gráfica las dos clases, la nube
#   de puntos con etiqueta negativa de la nube de puntos con etiqueta positiva.

#para pintar la recta, necesitamos 2 puntos, uno en el que la x = -50, y otro en el que
# x=50
#para sacar el valor de la y, despejamos en la función inicial:
#f(x,y) = y - ax - b -> -y = -ax - b ; y = ax + b
#como para x = 50, la y > 50, he ido probando hasta encontrar un valor de x en el
#que y =< 50
y_m50 = a*-50 + b
y_50 = a*50 + b

#se dibuja la recta: valores de x, valores de y, color y etiqueta
plt.plot(intervalo, [y_m50, y_50], c='green', label='Recta divisoria')

plt.xlabel("Intervalo del eje x [-50, 50]")
plt.ylabel("Intervalo del eje y [-50, 50]")

plt.legend()
plt.show()
```

```
---------------------------------
|                               |
|          Apartado a)          |
|                               |
---------------------------------
```

```
input(" -- apartado b)")
print("——————————————")

#averiguamos cuántos elementos forman el 10% de la lista de etiquetas negativas y
positivas
u_pc_n = int(len(et_neg) * 0.1) #negativas
u_pc_p = int(len(et_pos) * 0.1) #positivas

et_neg_ruido = et_neg
et_pos_ruido = et_pos

#las desordenamos
np.random.shuffle(et_neg_ruido)
np.random.shuffle(et_pos_ruido)

#a los u_pc_n/p elementos primeros les cambiamos al etiqueta, esto es, los sacamos
#de su correspondiente lista y los almacenamos en la de su otra clase

#almacenamos en variables auxiliares los valores que queremos sacar
val_n = et_neg_ruido[0:u_pc_n]    #negativos
val_p = et_pos_ruido[0:u_pc_p]    #positivos

#los eliminamos de sus correspondientes listas
et_neg_ruido = et_neg_ruido[u_pc_n:]
et_pos_ruido = et_pos_ruido[u_pc_p:]

#y los almacenamos en la lista de la otra clase
#en la lista de etiquetas positivas, voy a meter los valores negativos
et_neg_ruido = np.concatenate((et_neg_ruido, val_p), axis=0)
et_pos_ruido = np.concatenate((et_pos_ruido, val_n), axis=0)

plt.title("2.b Gráfica de puntos según la etiqueta")
plt.scatter(et_neg_ruido[:, 0], et_neg_ruido[:, 1], c='purple', label='Etiqueta -1.0')
plt.scatter(et_pos_ruido[:, 0], et_pos_ruido[:, 1], c='orange', label='Etiqueta 1.0')

#se dibuja la recta: valores de x, valores de y, color y etiqueta
plt.plot([-50,50], [y_m50, y_50], c='green', label='Recta divisoria')

plt.xlabel("Intervalo del eje x [-50, 50]")
plt.ylabel("Intervalo del eje y [-50, 50]")

plt.legend()
plt.show()
```

```
############################
#                          #
#       Ejercicio 3        #
#                          #
############################

print("\n\n")
print("********************\n")
print('*** EJERCICIO 3 ***')
input('********************')

#la función plot_datos_cuad, necesita como argumentos las etiquetas con ruido de dichos
#puntos a la lista de puntos con etiqueta negativa, le añado una columna de -1
et_neg_ruido = np.c_[et_neg_ruido, np.ones(et_neg_ruido.shape[0])*-1]

#a la lista de puntos con etiqueta positiva, le añado una columna de 1
et_pos_ruido = np.c_[et_pos_ruido, np.ones(et_pos_ruido.shape[0])]

#uno ambas listas en una sola
et_ruido = np.concatenate((et_neg_ruido, et_pos_ruido), axis=0)

#llamamos a la función plot_datos_cuad para cada función distinta
# - 1er argumento: las coordenadas de los puntos, que se corresponde con las dos
#   primeras columnas de et_ruido -> et_ruido[:, :2]
# - 2o argumento:las etiquetas de dichos puntos, que es la tercera columna de
#   et_ruido -> et_ruido[:, 2]

# - 3er argumento: la función
input("Circunferencia con el etiquetado del 2b")
plot_datos_cuad(et_ruido[:, :2], et_ruido[:, 2], funcion_0, 'f(x, y) = (x - 10)² + (y -
20)² - 400, con el etiquetado del 2b')

print('\n')
input("Elipse con el etiquetado del 2b")
plot_datos_cuad(et_ruido[:, :2], et_ruido[:, 2], funcion_1, 'f(x, y) = 0,5(x + 10)² +
(y - 20)² - 400, con el etiquetado del 2b')

print('\n')
input("Hipérbola con el etiquetado del 2b")
plot_datos_cuad(et_ruido[:, :2], et_ruido[:, 2], funcion_2, 'f(x, y) = 0,5(x - 10)² -
(y + 20)² - 400, con el etiquetado del 2b')

print('\n')
input("Parábola con el etiquetado del 2b")
plot_datos_cuad(et_ruido[:, :2], et_ruido[:, 2], funcion_3, 'f(x, y) = y - 20x² - 5x +
3, con el etiquetado del 2b')


print('\n')
input("Circunferencia")
plot_datos_cuad(et_ruido[:, :2], np.sign(funcion_0(et_ruido[:, :2])), funcion_0, 'f(x,
y) = (x - 10)² + (y - 20)² - 400')

print('\n')
input("Elipse")
plot_datos_cuad(et_ruido[:, :2], np.sign(funcion_1(et_ruido[:, :2])), funcion_1, 'f(x,
y) = 0,5(x + 10)² + (y - 20)² - 400')

print('\n')
input("Hipérbola")
plot_datos_cuad(et_ruido[:, :2], np.sign(funcion_2(et_ruido[:, :2])), funcion_2, 'f(x,
y) = 0,5(x - 10)² - (y + 20)² - 400')

print('\n')
input("Parábola")
plot_datos_cuad(et_ruido[:, :2], np.sign(funcion_3(et_ruido[:, :2])), funcion_3, 'f(x,
y) = y - 20x² - 5x + 3')
```