SVR

April 15, 2021

Tarea: Máquinas de soporte vectorial

Juaη Uriϵl Lϵgaria $\pi\epsilon$ña

1 Generación de los datos

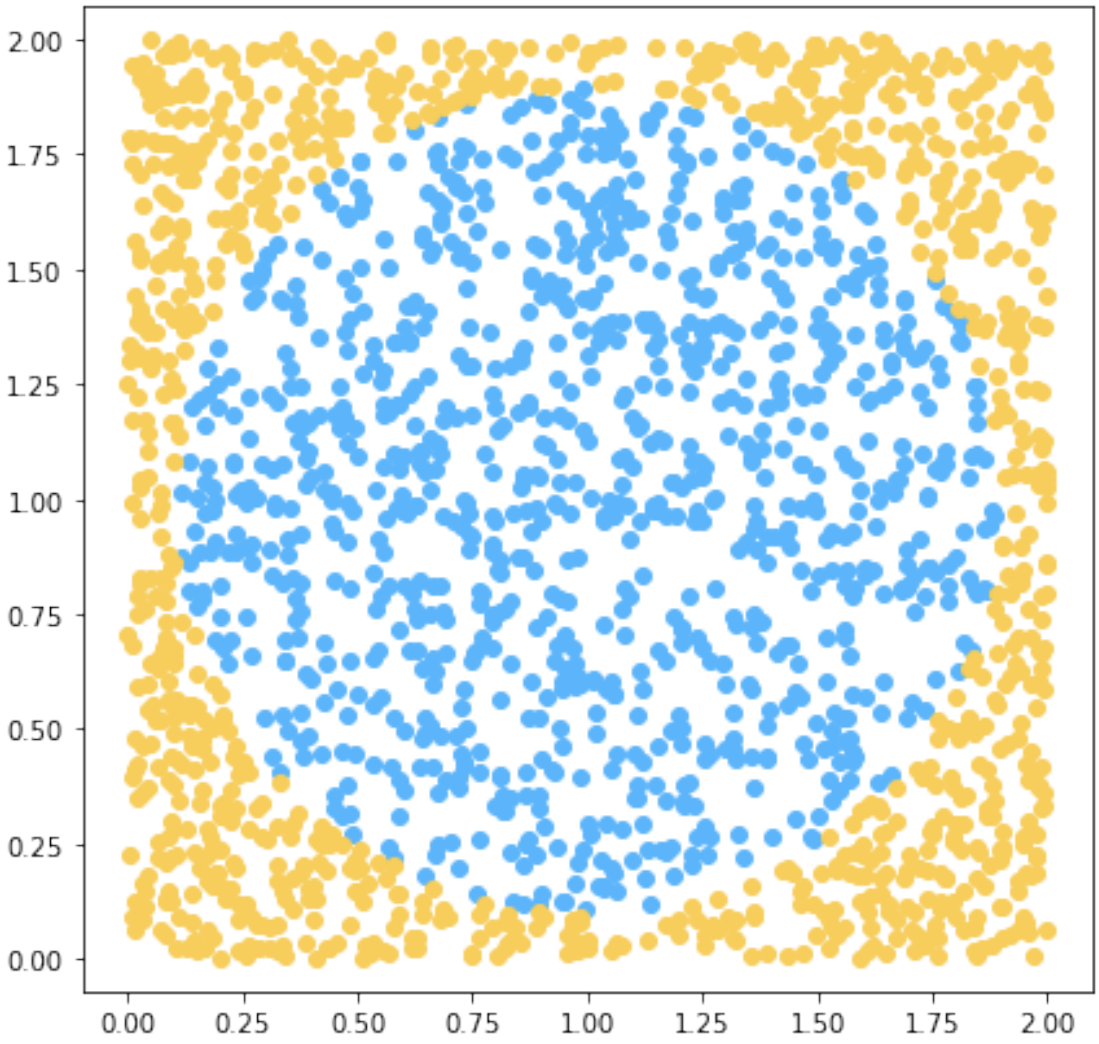
1.1 Generación de los datos del círculo

[14]:
```
import random
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
from IPython.display import clear_output
from sklearn import svm
```

[15]:
```
def generateRandomCirclePoints(numberOfPoints):
    inCircleArray = []
    outsideCircle = []
    #No se si esta sea la forma mas sencilla de llenar pero no son demasiados
    #puntos, así que ebería funcionar
    L = 1
    centerX = L
    centerY = L
    while(len(inCircleArray) < numberOfPoints or len(outsideCircle) <␣
 ↪numberOfPoints):
        newX = 2*L*random.random()
        newY = 2*L*random.random()
        norm = np.sqrt((newX-centerX)**2 + (newY - centerY)**2)
        if(norm < 0.9):
            if(len(inCircleArray)< numberOfPoints):
                inCircleArray.append([newX, newY])

        else:
            if(len(outsideCircle) < numberOfPoints):
                outsideCircle.append([newX, newY])

    xCircle = np.zeros((2*numberOfPoints, 2))
```

```
    yCircle = np.zeros((2*numberOfPoints))
    for i in range(0,len(inCircleArray)):
        xCircle[i,:] = inCircleArray[i]
        #1 va a simbolizar que se encuentra dentro de la figura
        yCircle[i] = 1

    for i in range(len(outsideCircle),2*len(outsideCircle)):
        xCircle[i,:] = outsideCircle[i -len(outsideCircle)]
        #0 va a simbolizar que se encuentra fuera de la figura
        yCircle[i] = 0

    return xCircle, yCircle
```

[16]:
```
colorInside = "#5cb5fa"
colorOutside = "#f7ce5c"
xCircle, yCircle = generateRandomCirclePoints(1000)
plt.figure(figsize = (7,7))
plt.gca().axis('equal')
for i in range(0,np.size(xCircle, 0)):
    point = xCircle[i,:]
    pointClass = yCircle[i]
    if(pointClass == 0):
        plt.plot(point[0], point[1], 'o', color = colorOutside)
    elif(pointClass == 1):
        plt.plot(point[0], point[1], 'o', color = colorInside)
```

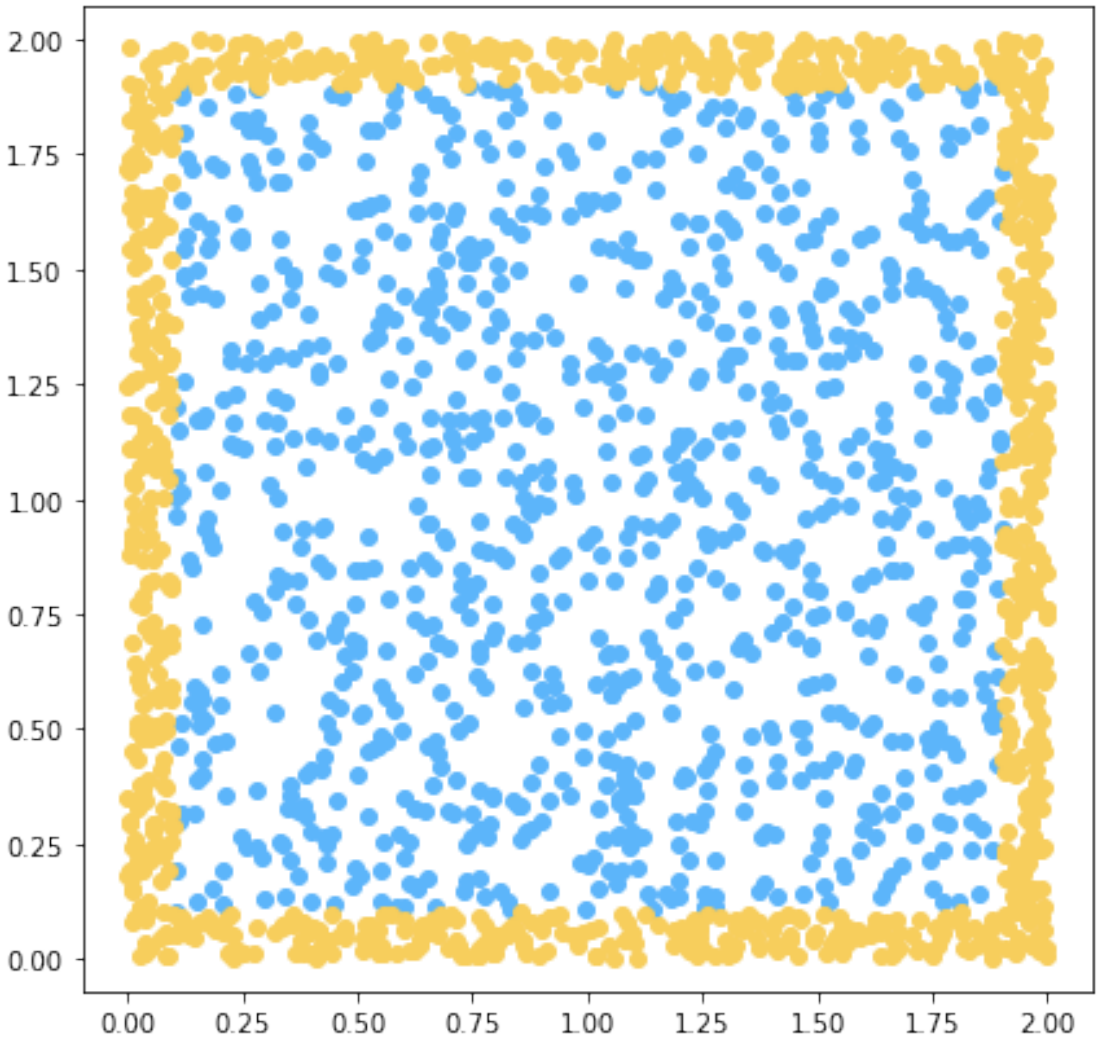


1.2 Generación de los datos de un cuadrado

[17]:
```
def generateRandomSquarePoints(numberOfPoints):
    inSquare = []
    outsideSquare = []
    L = 1
    delta = (L - 0.9)
    while(len(inSquare) < numberOfPoints or len(outsideSquare) < numberOfPoints):
        newX = 2*L*random.random()
        newY = 2*L*random.random()
        if(newX >= delta and newX <= 2*L - delta and newY >= delta and newY <=␣
↪2*L -delta):
            if(len(inSquare) < numberOfPoints):
                inSquare.append([newX, newY])
```

```
        else:
            if(len(outsideSquare) < numberOfPoints):
                outsideSquare.append([newX, newY])


    xSquare = np.zeros((2*numberOfPoints, 2))
    ySquare = np.zeros((2*numberOfPoints))

    for i in range(0,len(inSquare)):
        xSquare[i,:] = inSquare[i]
        #1 va a simbolizar que se encuentra dentro de la figura
        ySquare[i] = 1

    for i in range(len(outsideSquare),2*len(outsideSquare)):
        xSquare[i,:] = outsideSquare[i -len(outsideSquare)]
        #0 va a simbolizar que se encuentra fuera de la figura
        ySquare[i] = 0

    return xSquare, ySquare
```

[18]:
```
colorInside = "#5cb5fa"
colorOutside = "#f7ce5c"
xSquare, ySquare = generateRandomSquarePoints(1000)
plt.figure(figsize = (7,7))
plt.gca().axis('equal')
for i in range(0,np.size(xCircle, 0)):
    point = xSquare[i,:]
    pointClass = ySquare[i]
    if(pointClass == 0):
        plt.plot(point[0], point[1], 'o', color = colorOutside)
    elif(pointClass == 1):
        plt.plot(point[0], point[1], 'o', color = colorInside)
```



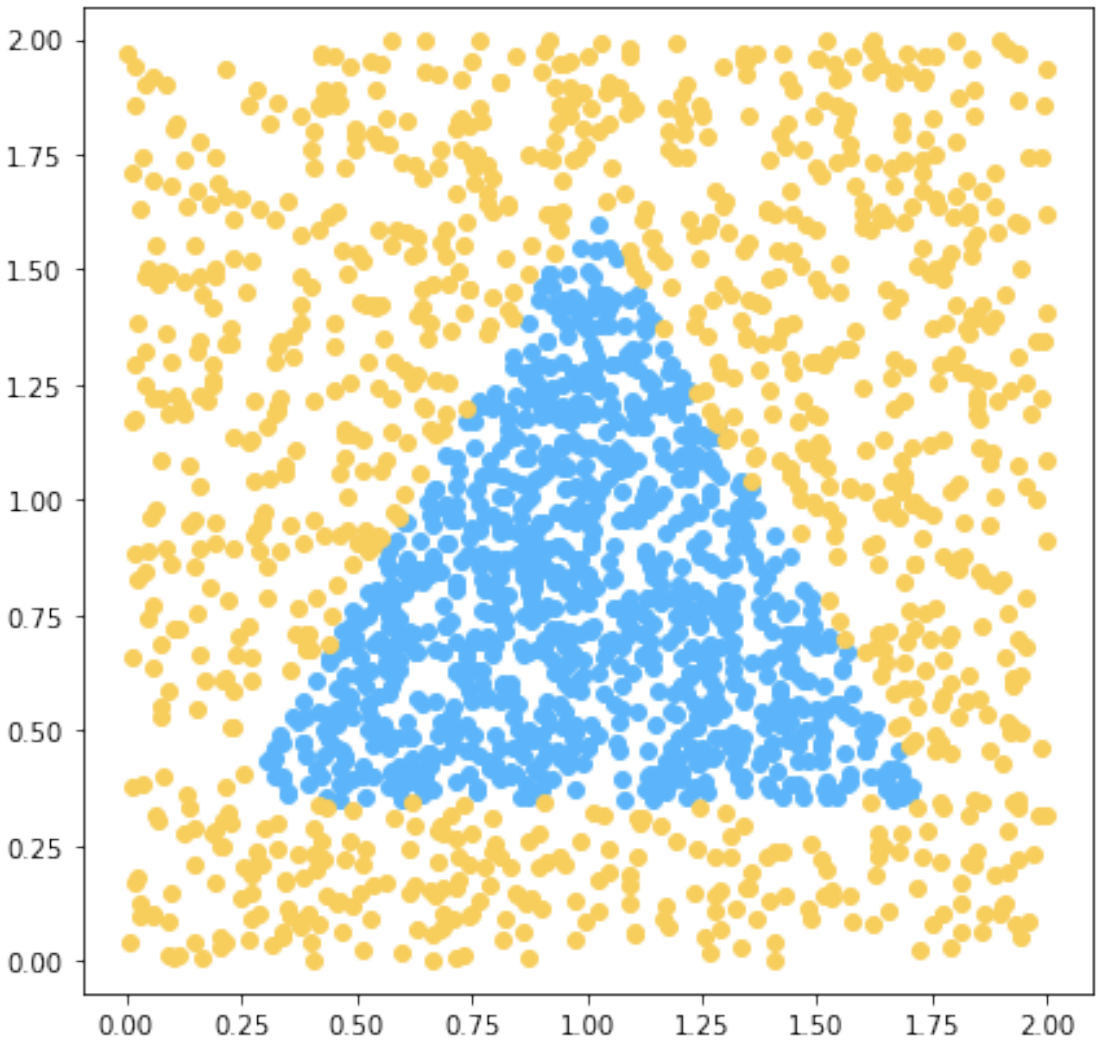
1.3 Generación de los datos del triángulo

[19]:
```
def generateRandomTrianglePoints(numberOfPoints):
    numberOfPoints = 1000
    L = 1
    triangleCenterX = L
    triangleCenterY = L
    insideTriangle = []
    outsideTriangle = []
    triangleSide = 1.5
    deltaX = L- triangleSide/2
    h = np.sqrt(triangleSide**2 - (triangleSide/2)**2)
    deltaY = L - h/2
```

```
    #Vertice 1 del triangulo
    v1 = [deltaX, deltaY]
    #Vertice 2 del triángulo
    v2 = [deltaX + triangleSide, deltaY]
    #Vertice 3 del triángulo
    v3 = [deltaX + triangleSide/2, deltaY + h]
    #Pendiente de la primera recta del triangulo
    angleRadians = (np.pi/180)*60
    m1 = np.tan(angleRadians)
    m2 = -np.tan(angleRadians)
    b1 = v1[1] - m1*v1[0]
    b2 = v2[1] - m2*v2[0]

    while(len(insideTriangle) < numberOfPoints or len(outsideTriangle) <␣
↪numberOfPoints):

        newX = 2*L*random.random()
        newY = 2*L*random.random()

        if(newX >= v1[0] and newY >= deltaY and newY <= b1 + m1*newX and newX <=␣
↪v2[0] and newY <= b2 +m2*newX):
            if(len(insideTriangle) < numberOfPoints):
                insideTriangle.append([newX, newY])
        else:
            if(len(outsideTriangle) < numberOfPoints):
                outsideTriangle.append([newX, newY])

    xTriangle = np.zeros((2*numberOfPoints, 2))
    yTriangle = np.zeros((2*numberOfPoints))

    for i in range(0,len(insideTriangle)):
        xTriangle[i,:] = insideTriangle[i]
        #1 va a simbolizar que se encuentra dentro de la figura
        yTriangle[i] = 1

    for i in range(len(outsideTriangle),2*len(outsideTriangle)):
        xTriangle[i,:] = outsideTriangle[i -len(outsideTriangle)]
        #0 va a simbolizar que se encuentra fuera de la figura
        yTriangle[i] = 0


    return xTriangle, yTriangle
```

[20]:
```
#Ahora graficamos los puntos.
colorInside = "#5cb5fa"
colorOutside = "#f7ce5c"
xTriangle, yTriangle = generateRandomTrianglePoints(1000)
```

```
plt.figure(figsize = (7,7))
plt.gca().axis('equal')

for i in range(0,np.size(xCircle, 0)):

    point = xTriangle[i,:]
    pointClass = yTriangle[i]

    if(pointClass == 0):
        plt.plot(point[0], point[1], 'o', color = colorOutside)
    elif(pointClass == 1):
        plt.plot(point[0], point[1], 'o', color = colorInside)
```



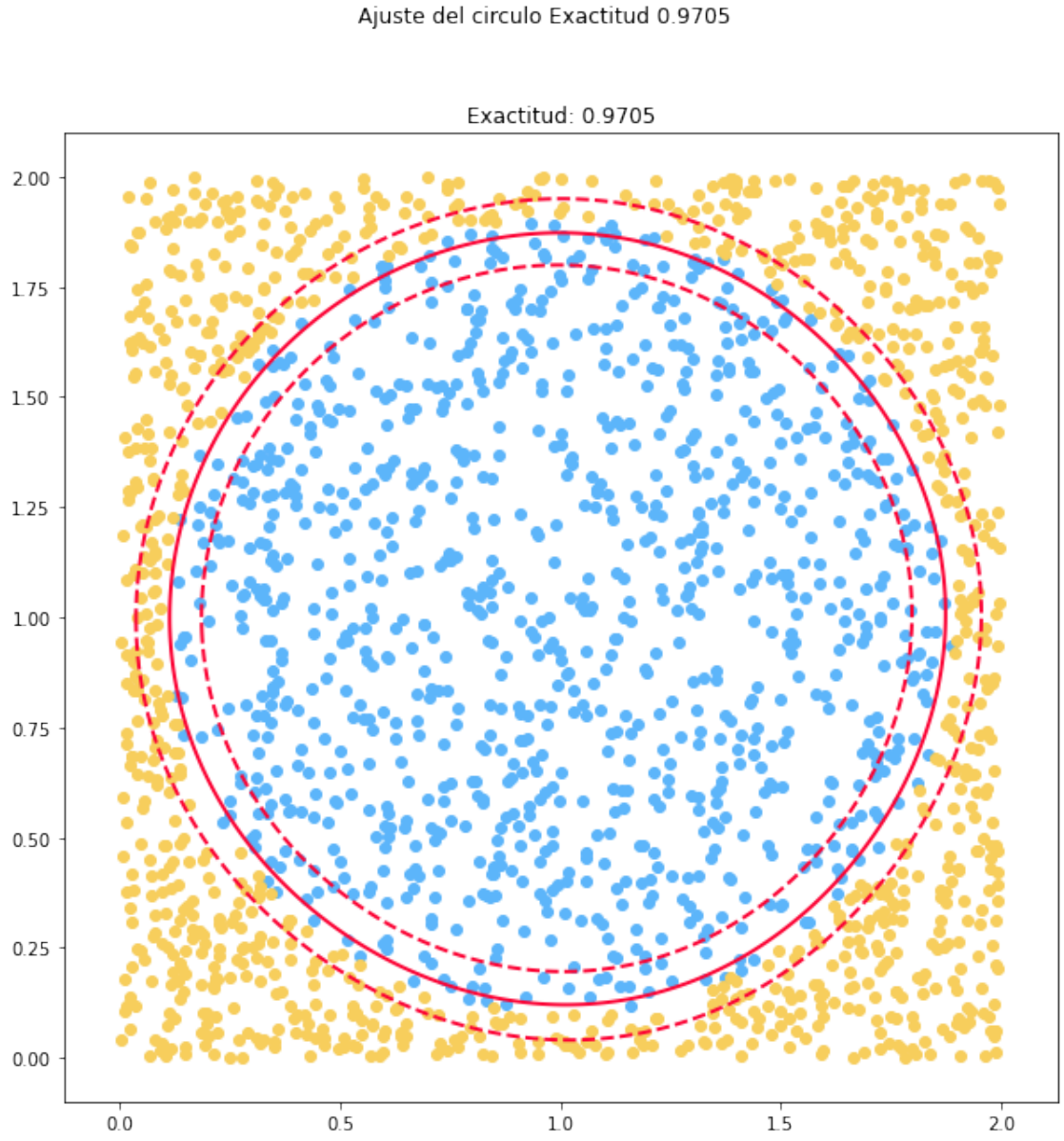
2 Clasificaciones

2.1 Clasificación del círculo

[26]:
```
def analyzeContingencyTable(TP, FP, FN, TN):
    sensitivity = TP/(TP + FN)
    specificity = TN/(TN + FP)
    accuracy = (TP + TN)/(TP + TN + FP + FN)

    return sensitivity, specificity, accuracy
```

[69]:
```
#Creamos el clasificador
clf = svm.SVC()
clf.fit(xCircle, yCircle)
#Generamos nuevos datos
xCircleNew, yCircleNew = generateRandomCirclePoints(1000)
predictedY = []
truePositives = 0
falsePositives = 0
trueNegatives = 0
falseNegatives = 0
for i in range(0, np.size(xCircleNew, 0)):
    point = xCircleNew[i,:].reshape(1,-1)
    result = clf.predict(point)[0]
    predictedY.append(result)
    if(result == yCircleNew[i] and yCircleNew[i] == 0):
        trueNegatives += 1
    elif(result != yCircleNew[i] and yCircleNew[i] == 0):
        falsePositives += 1
    elif(result == yCircleNew[i] and yCircleNew[i] == 1):
        truePositives += 1
    elif(result != yCircleNew[i] and yCircleNew[i] == 1):
        falseNegatives += 1
sensitivity, specificity, accuracy = analyzeContingencyTable(truePositives,␣
 ↪falsePositives, falseNegatives, trueNegatives)
xInterval = np.linspace(np.min(xCircle[:,0]), np.max(xCircle[:,0]), 1000)
yInterval = np.linspace(np.min(xCircle[:,1]), np.max(xCircle[:,1]), 1000)
yGrid, xGrid = np.meshgrid(yInterval, xInterval)
xy = np.vstack([xGrid.ravel(), yGrid.ravel()]).T
z = clf.decision_function(xy).reshape(xGrid.shape)
#Veamos que pasa si hacemos esto


#Ahora simplemente graficamos los datos del circulo
fig,ax = plt.subplots(figsize=(10,10))
fig.suptitle("Ajuste del circulo"+" Exactitud "+str(accuracy))
ax.axis('equal')
```

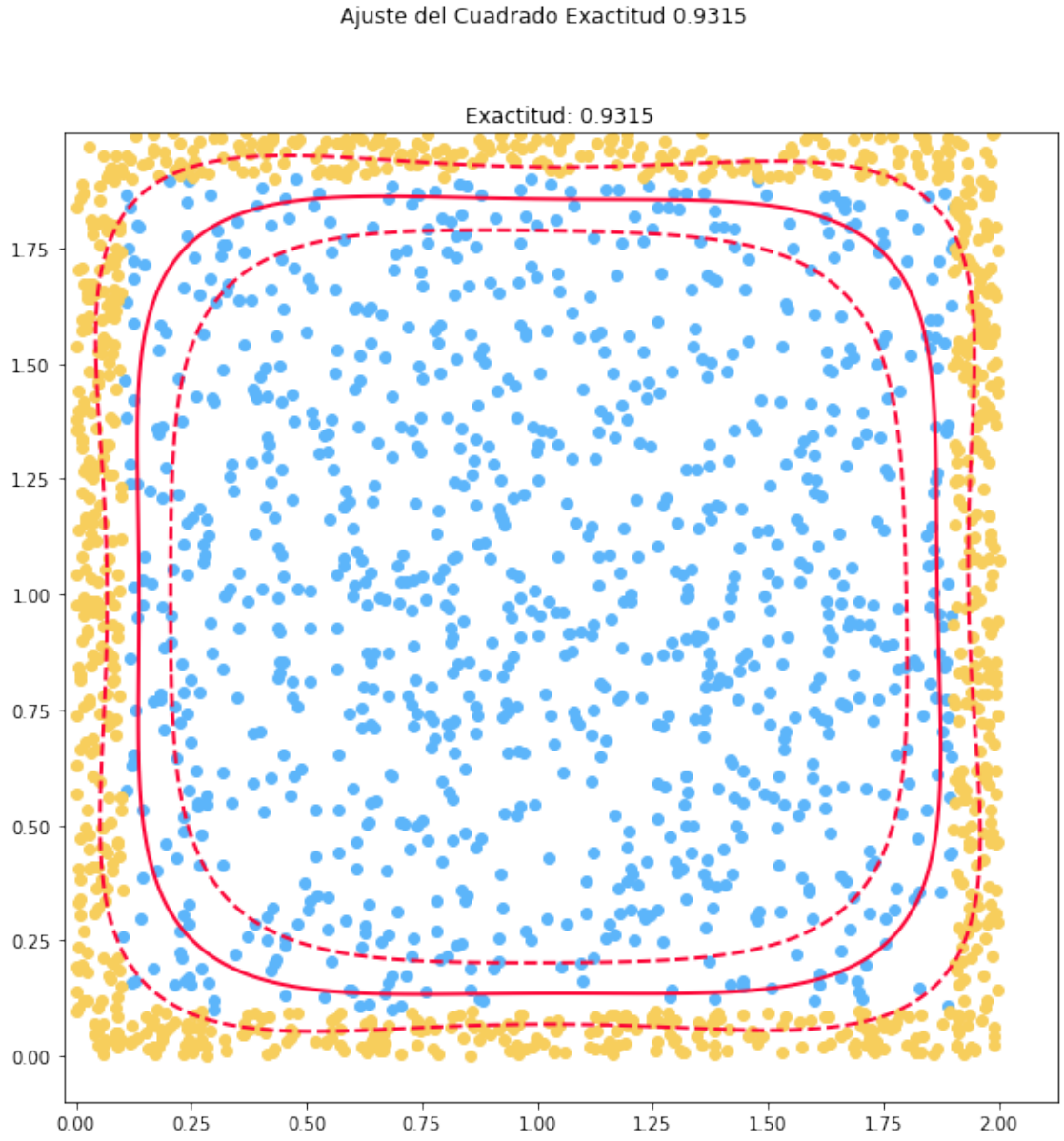
```
ax.set_title("Exactitud: "+str(accuracy))
#ax.pcolormesh(xGrid, yGrid, z, cmap=plt.cm.RdYlBu)
for i in range(0,np.size(xCircleNew,0)):
    point = xCircleNew[i,:]
    #Si el punto está fuera del circulo
    if(yCircleNew[i] == 0):
        ax.plot(point[0], point[1], 'o', color = colorOutside)
    #Si el punto esta dentro del círculo
    elif(yCircleNew[i] == 1):
        ax.plot(point[0],point[1], 'o',color = colorInside)



ax.contour(xGrid, yGrid, z, levels = [-1,0,1], alpha=1, colors =␣
 ↪["#fc0335","#fc0335","#fc0335"],linewidths = 2, linestyles =␣
 ↪['--','-','--'],zorder = 300)
plt.show()
```



2.2 Clasificación del cuadrado

[72]:
```
#Creamos el clasificador
clf = svm.SVC()
clf.fit(xSquare, ySquare)
#Generamos nuevos datos
xSquareNew, ySquareNew = generateRandomSquarePoints(1000)
predictedY = []
truePositives = 0
falsePositives = 0
```

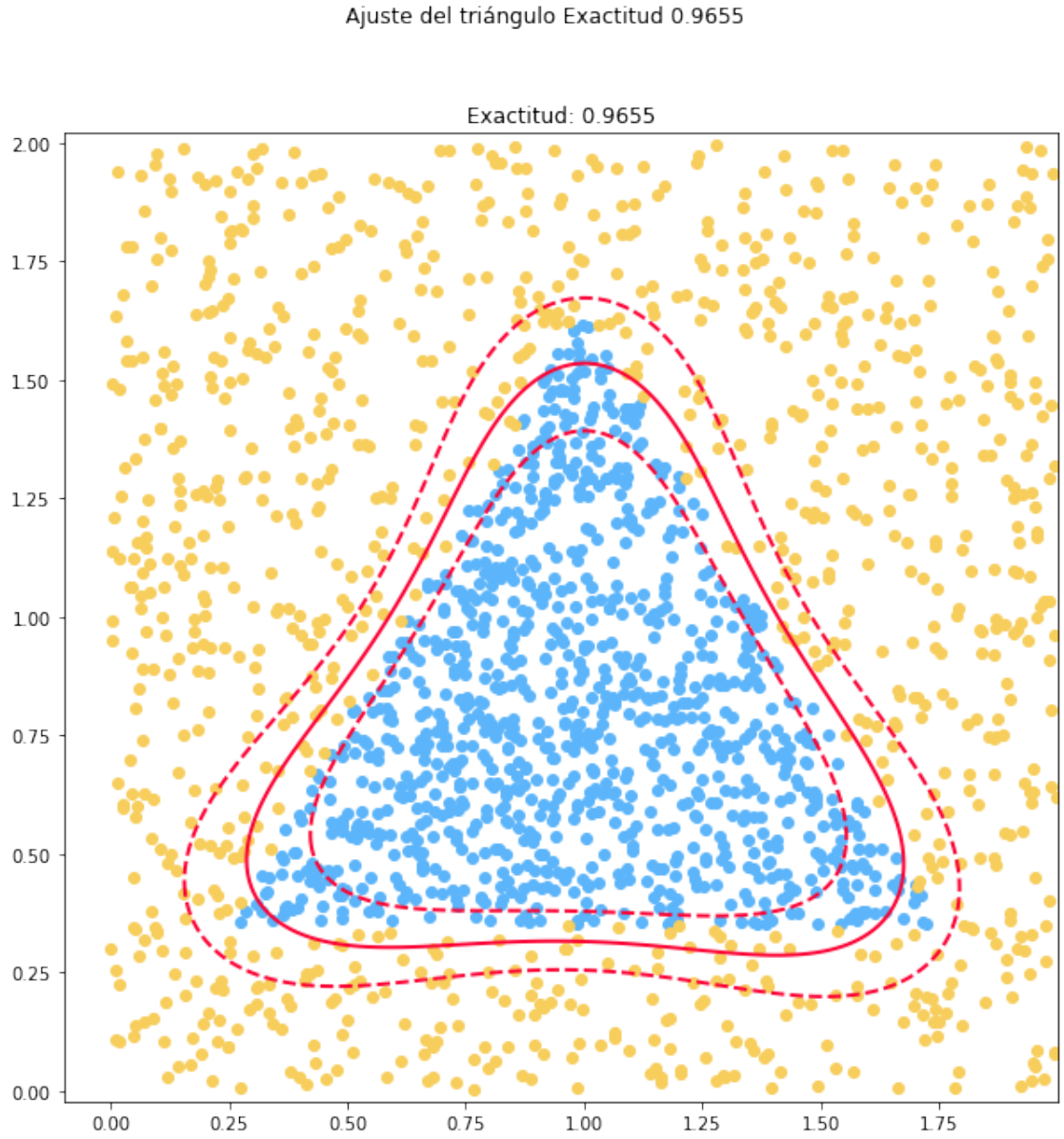
```
trueNegatives = 0
falseNegatives = 0
for i in range(0, np.size(xSquareNew, 0)):
    point = xSquareNew[i,:].reshape(1,-1)
    result = clf.predict(point)[0]
    predictedY.append(result)
    if(result == ySquareNew[i] and ySquareNew[i] == 0):
        trueNegatives += 1
    elif(result != ySquareNew[i] and ySquareNew[i] == 0):
        falsePositives += 1
    elif(result == ySquareNew[i] and ySquareNew[i] == 1):
        truePositives += 1
    elif(result != ySquareNew[i] and ySquareNew[i] == 1):
        falseNegatives += 1
sensitivity, specificity, accuracy = analyzeContingencyTable(truePositives,␣
 ↪falsePositives, falseNegatives, trueNegatives)
xInterval = np.linspace(np.min(xSquare[:,0]), np.max(xSquare[:,0]), 1000)
yInterval = np.linspace(np.min(xSquare[:,1]), np.max(xSquare[:,1]), 1000)
yGrid, xGrid = np.meshgrid(yInterval, xInterval)
xy = np.vstack([xGrid.ravel(), yGrid.ravel()]).T
z = clf.decision_function(xy).reshape(xGrid.shape)
#Veamos que pasa si hacemos esto


#Ahora simplemente graficamos los datos del circulo
fig,ax = plt.subplots(figsize=(10,10))
fig.suptitle("Ajuste del Cuadrado"+" Exactitud "+str(accuracy))
ax.axis('equal')
ax.set_title("Exactitud: "+str(accuracy))
#ax.pcolormesh(xGrid, yGrid, z, cmap=plt.cm.RdYlBu)
for i in range(0,np.size(xSquareNew,0)):
    point = xSquareNew[i,:]
    #Si el punto está fuera del circulo
    if(ySquareNew[i] == 0):
        ax.plot(point[0], point[1], 'o', color = colorOutside)
    #Si el punto esta dentro del círculo
    elif(ySquareNew[i] == 1):
        ax.plot(point[0],point[1], 'o',color = colorInside)



ax.contour(xGrid, yGrid, z, levels = [-1,0,1], alpha=1, colors =␣
 ↪["#fc0335","#fc0335","#fc0335"],linewidths = 2, linestyles =␣
 ↪['--','-','--'],zorder = 300)
plt.show()
```



2.3 Clasificación del triángulo

[73]:
```
#Creamos el clasificador
clf = svm.SVC()
clf.fit(xTriangle, yTriangle)
#Generamos nuevos datos
xTriangleNew, yTriangleNew = generateRandomTrianglePoints(1000)
predictedY = []
truePositives = 0
falsePositives = 0
```

```
trueNegatives = 0
falseNegatives = 0
for i in range(0, np.size(xTriangleNew, 0)):
    point = xTriangleNew[i,:].reshape(1,-1)
    result = clf.predict(point)[0]
    predictedY.append(result)
    if(result == yTriangleNew[i] and yTriangleNew[i] == 0):
        trueNegatives += 1
    elif(result != yTriangleNew[i] and yTriangleNew[i] == 0):
        falsePositives += 1
    elif(result == yTriangleNew[i] and yTriangleNew[i] == 1):
        truePositives += 1
    elif(result != yTriangleNew[i] and yTriangleNew[i] == 1):
        falseNegatives += 1
sensitivity, specificity, accuracy = analyzeContingencyTable(truePositives,␣
 ↪falsePositives, falseNegatives, trueNegatives)
xInterval = np.linspace(np.min(xTriangle[:,0]), np.max(xTriangle[:,0]), 1000)
yInterval = np.linspace(np.min(xTriangle[:,1]), np.max(xTriangle[:,1]), 1000)
yGrid, xGrid = np.meshgrid(yInterval, xInterval)
xy = np.vstack([xGrid.ravel(), yGrid.ravel()]).T
z = clf.decision_function(xy).reshape(xGrid.shape)
#Veamos que pasa si hacemos esto

#Ahora simplemente graficamos los datos del circulo
fig,ax = plt.subplots(figsize=(10,10))
fig.suptitle("Ajuste del triángulo"+" Exactitud "+str(accuracy))
ax.axis('equal')
ax.set_title("Exactitud: "+str(accuracy))
#ax.pcolormesh(xGrid, yGrid, z, cmap=plt.cm.RdYlBu)
for i in range(0,np.size(xTriangleNew,0)):
    point = xTriangleNew[i,:]
    #Si el punto está fuera del circulo
    if(yTriangleNew[i] == 0):
        ax.plot(point[0], point[1], 'o', color = colorOutside)
    #Si el punto esta dentro del círculo
    elif(yTriangleNew[i] == 1):
        ax.plot(point[0],point[1], 'o',color = colorInside)



ax.contour(xGrid, yGrid, z, levels = [-1,0,1], alpha=1, colors =␣
 ↪["#fc0335","#fc0335","#fc0335"],linewidths = 2, linestyles =␣
 ↪['--','-','--'],zorder = 300)
plt.show()
```



3 Observaciones

Uno de los aspectos interesantes de los modelos de clasificación generados con SVM es que los Kernels que se usan son diferenciables (son polinomios, gaussianas, sigmoides etc.) con lo cual las fronteras de decisión que se generan tienden a ser curvas suaves que no pueden tener esquinas. Esto hace que se tenga un caso un tanto contrario al obtenido con árboles aleatorios. En estos últimos, se había encontrado que los bordes se iban trazando con lineas rígidas horizontales y verticales, por tanto la clasificación de puntos dentro de un cuadrado presentaba la exactitud mas alta, mientras que para el círculo se trazaba esta especie de frontera rígida pixelada circular, que presentaba la peor exactitud de entre las 3 figuras examinadas. Con SVM sin embargo el uso

de un Kernel diferenciable (gaussiano en este caso) hace que la frontera de decisión circular se ajuste bastante bien, y es de hecho para la que se consigue la mejor exactitud con este modelo. Sin embargo, para el cuadrado, las esquinas del contorno deben necesariamente suavizarse y es por ello que se logra la exactitud mas baja en esa figura.

El triángulo aquí vuelve a jugar un papel de caso intermedio, se trata de una figura con menos esquinas que el cuadrado, pero igualmente el modelo con SVM termina suavisandolas, lo cual resulta en una exactitud menor a la del círculo.

En resumen las relaciones de exactitud para árboles aleatorios y SVM son:

Árboles aleatorios: Cuadrado > Triángulo > Círculo

SVM: Círculo > Tríangulo >Cuadrado

[ ]: