Aprendizaje automático y minería de datos: Práctica 0

Jorge Rodríguez García y Gonzalo Sanz Lastra

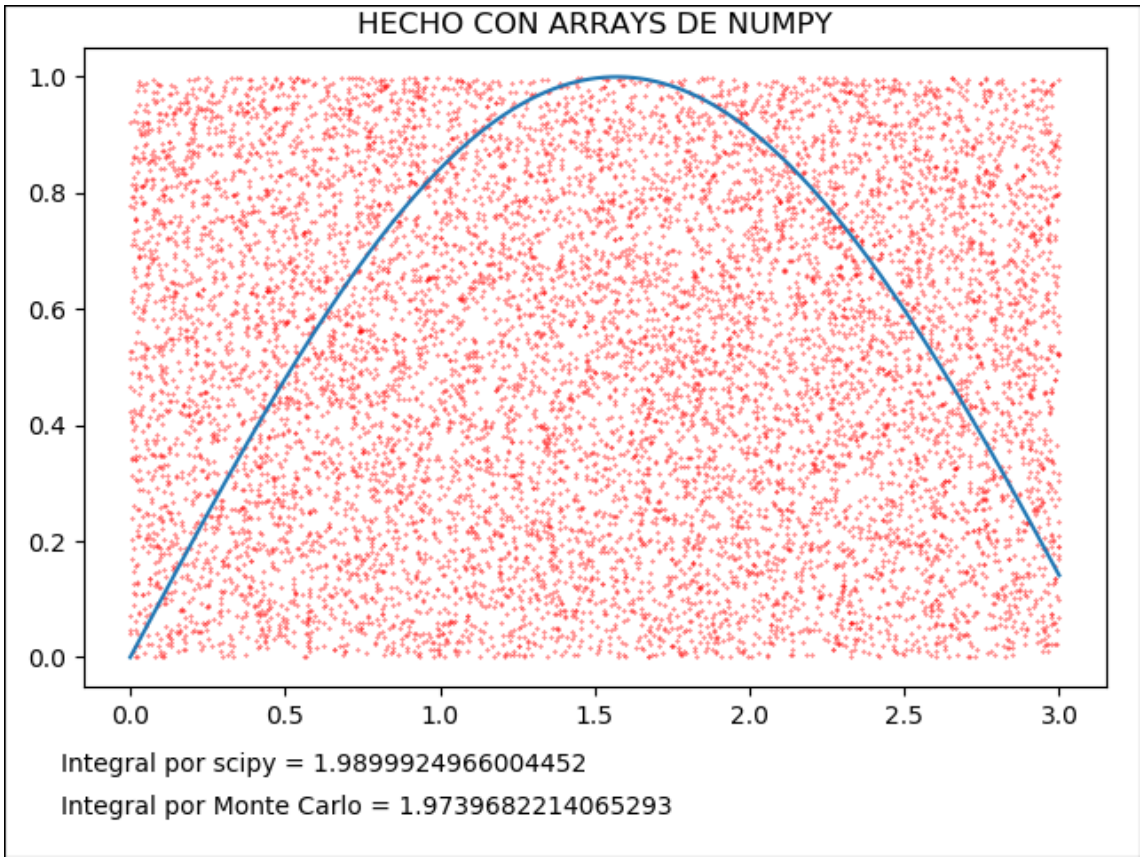
Código de la práctica:

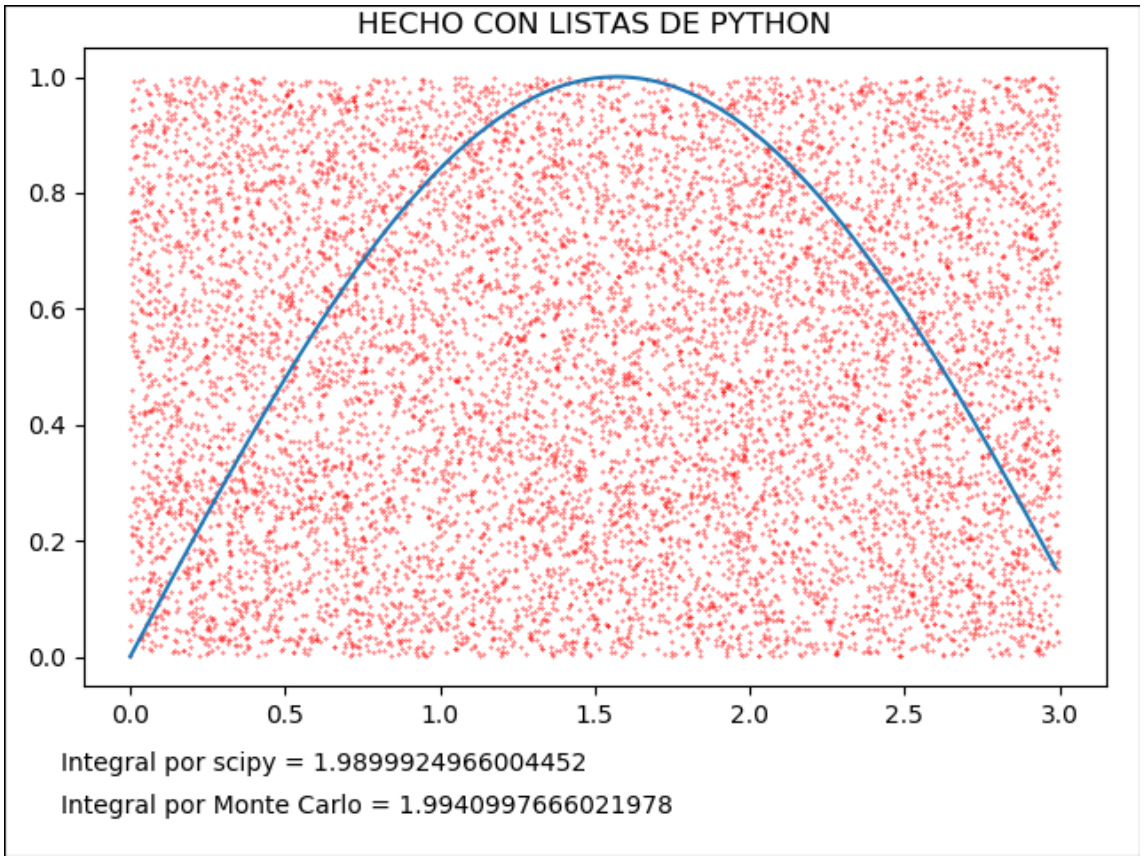
```python
import numpy as np
import scipy.integrate
from matplotlib import pyplot as plt
import random
import time

# integral de una funcion usando arrays de numpy
def integra_mc_npy(fun = np.sin, a = 0, b = 3, num_puntos = 10000, showFigure = True):
    tic = time.process_time() # tiempo inicial

    # generacion de la curva de la funcion
    X = np.linspace(a, b, 256, endpoint=True)
    Y = fun(X)

    maxValue = np.amax(Y) # maximo valor de la funcion

    #generacion de numeros aleatorios
    xRand = np.random.uniform(a, b, num_puntos)
    yRand = np.random.uniform(0, maxValue, num_puntos)

    nDebajo = (fun(xRand) > yRand) # los que quedan bajo la funcion

    # integrales por monte carlo y scipy
    IMonteCarlo = (nDebajo.sum()/num_puntos)*(b-a)*maxValue
    IIntegrate = (scipy.integrate.quad(fun, a, b))[0]

    #pintamos grafica y datos
    if(showFigure):
        plt.axes([0, 0, 1, 1])
        plt.text(0.05,0.05, "Integral por Monte Carlo = " + str(IMonteCarlo))
        plt.text(0.05,0.1, "Integral por scipy = " + str(IIntegrate))

        plt.axes([.07, .2, .9, .75])
        plt.plot(X, Y)
        plt.scatter(xRand, yRand, 0.1, 'red')

        plt.title("HECHO CON ARRAYS DE NUMPY")

        plt.savefig('IntegralMonteCarloNumPY.png')
        plt.show()

    toc = time.process_time() # tiempo final
    return 1000 * (toc - tic)


```

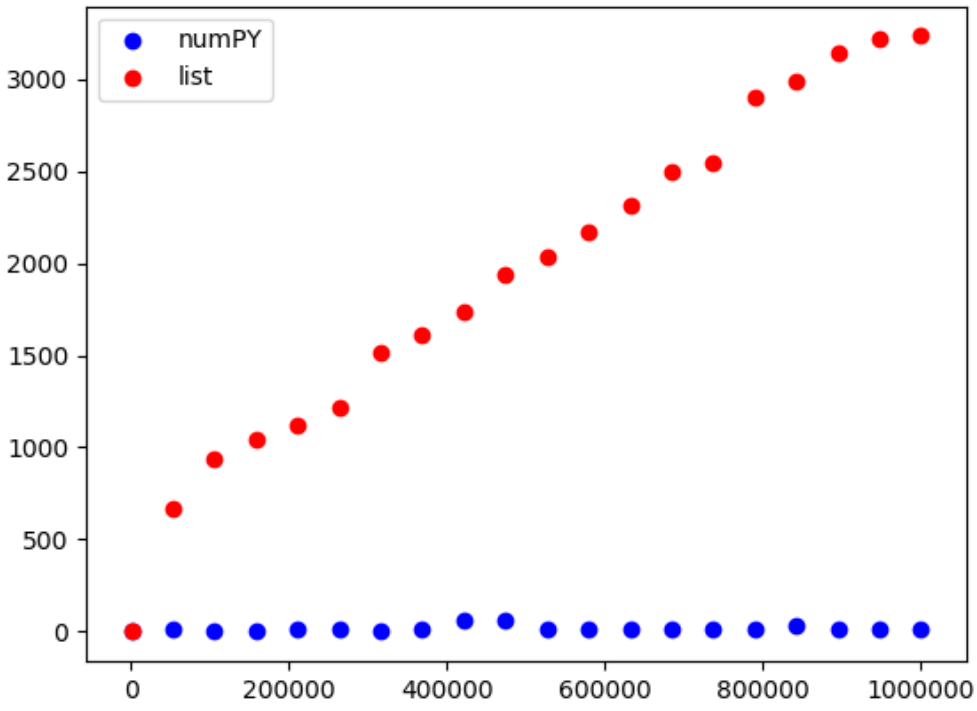
```python

# integral de una funcion usando listas de python con bucles
def integra_mc_list(fun = np.sin, a = 0, b = 3, num_puntos = 10000, showFigure = True):
    tic = time.process_time() # tiempo inicial

    step = 256
    espaciado = (b - a)/step
    X = []
    Y = []
    maxValue = -float("inf")

    # generacion de la curva de la funcion
    for i in range(step):
        X.append(i * espaciado)
        Y.append(fun(X[i]))
        if(Y[i] > maxValue):
            maxValue = Y[i] # maximo valor de la funcion

    xRand = []
    yRand = []
    nDebajo = 0

    #generacion de numeros aleatorios
    for i in range(num_puntos):
        xRand.append(random.uniform(a, b))
        yRand.append(random.uniform(0, maxValue))
        if(fun(xRand[i]) > yRand[i]):
            nDebajo += 1  # los que quedan bajo la funcion

    # integrales por monte carlo y scipy
    IMonteCarlo = (nDebajo/num_puntos)*(b-a)*maxValue
    IIntegrate = (scipy.integrate.quad(fun, a, b))[0]

    #pintamos grafica y datos
    if(showFigure):
        plt.axes([0, 0, 1, 1])
        plt.text(0.05,0.05, "Integral por Monte Carlo = " + str(IMonteCarlo))
        plt.text(0.05,0.1, "Integral por scipy = " + str(IIntegrate))

        plt.axes([.07, .2, .9, .75])
        plt.plot(X, Y)
        plt.scatter(xRand, yRand, 0.1, 'red')

        plt.title("HECHO CON LISTAS DE PYTHON")

        plt.savefig('IntegralMonteCarloLists.png')
        plt.show()
```

```
93
94      toc = time.process_time() # tiempo final
95      return 1000 * (toc - tic)
96
97
98
99  def compara_tiempos():
100     X = np.linspace(100, 300000, 20)
101     timeNumPY = []
102     timeList = []
103
104     for x in X:
105         timeNumPY.append(integra_mc_npy(num_puntos = int(x), showFigure = False))
106         timeList.append(integra_mc_list(num_puntos = int(x), showFigure = False))
107
108     plt.figure()
109     plt.scatter(X, timeNumPY, color = 'blue', label = 'numPY')
110     plt.scatter(X, timeList, color = 'red', label = 'list')
111     plt.legend()
112
113     plt.savefig('ComparadorTiempos.png')
114     plt.show()
115
116
117 def main():
118     timeNumPY = integra_mc_npy() # pinta grafica usando numpy
119     timeList = integra_mc_list() # pinta grafica usando listas y bucles
120     compara_tiempos()            # pinta grafica comparando tiempos entre ellos
121
122
123
124 main()
```

Gráficas resultantes:




HECHO CON LISTAS DE PYTHON
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Integral por scipy = 1.9899924966004452
Integral por Monte Carlo = 1.9940997666021978


numPY
list
3000
2500
2000
1500
1000
500
0
0
200000
400000
600000
800000
1000000