Proyecto Final Aprendizaje Automático y Minería de Datos - *¿Abeja o Avispa?*

Gonzalo Cidoncha y Guillermo Sánchez de Lamadrid

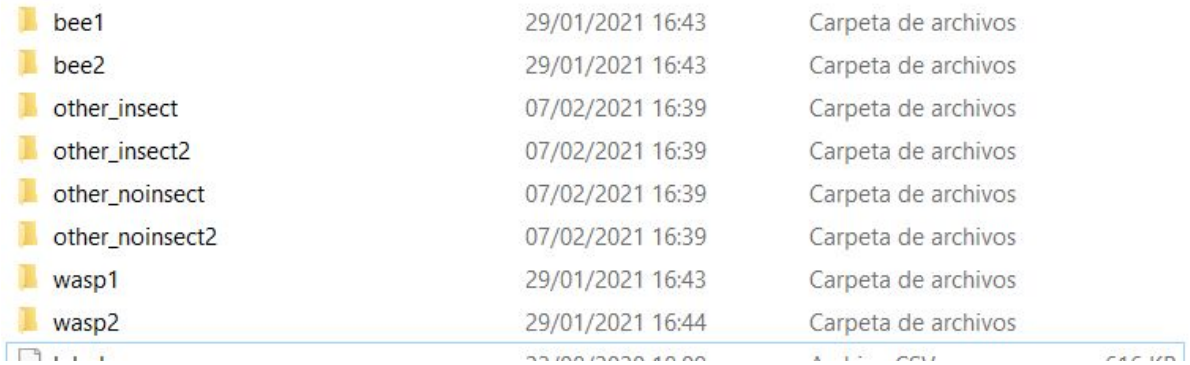
1. **Introducción:**

Para nuestro proyecto final hemos decidido hacer uso de una base de datos obtenida en *Kaggle.com (* https://www.kaggle.com/jerzydziewierz/bee-vs-wasp) que proporciona carpetas con imágenes de avispas, abejas, otros insectos e imágenes aleatorias que no contienen nada de lo anterior.
El objetivo será realizar un programa de Python que permita discernir si la imagen obtenida refleja una abeja, una avispa o ninguna de las anteriores haciendo uso de una red neuronal y una regresión logística tal y como hemos realizado en las prácticas anteriormente realizadas en clase siguiendo las indicaciones del profesor.

(Los scripts utilizados los adjuntamos junto con la entrega para que puedan ser analizados con más detenimiento de lo que permite la memoria)

2. **Estructura y conversión de los datos:**

Las carpetas con las imágenes obtenidas tendrán la siguiente estructura:



Siendo bee1 y wasp1 las de las imágenes que usaremos de entrenamiento, las demás carpetas para ser utilizadas de test y validación.

Para poder hacer uso de las imágenes aquí contenidas en formato JPG es necesario primero redimensionarlas para que todas posean el mismo tamaño, etiquetarlas para saber a cuál de los tipos pertenece (Abeja, Avispa, Otro Insecto u Otro no Insecto) y por último añadirlo todo a un archivo .mat (“dataMat.mat”). Para encargarse de este proceso hemos creado el script adjuntado en la entrega “generaMat.py”.

El tratamiento aplicado a cada imagen para transformarlo a la matriz es el siguiente (para mayor detalle, revisar el script adjunto en el cual hemos descrito paso a paso el proceso por medio de comentarios):

```
for i in insect_train:
    if os.path.isfile(carpeta + "/other_insect/" + i):
        aux = Image.open(carpeta + "/other_insect/" + i).convert('P')
        aux = aux.resize((100,100), Image.ANTIALIAS)
        aux = np.asarray(aux)/255.0
        resizedinsect.append(aux)
        etiqueta.append(2)
```

3. **Realización de la Red Neuronal:**

En primer lugar, desarrollamos un entrenamiento para una red neuronal que nos ayudase a clasificar las imágenes de test (archivo “redNeuronal.py”). Partiendo de la matriz generada anteriormente, preparamos las iteraciones de entrenamiento de la siguiente manera:

Primeramente, extraemos la información de la matriz ya generada, siendo la X los ejemplos de entrenamiento que hemos guardado, y la Y sus etiquetas. Además concretamos la cantidad de capas ocultas, las diferentes lambdas e iteraciones que vamos a probar, y el número de entradas y etiquetas.

```
data = loadmat("dataMat.mat")

X = data["X"]
y = data["y"].ravel()

entradas = X.shape[1]
etiquetas = len(np.unique(y))
ocultas = [25, 50]
landas = [0.01, 0.1, 1, 10]
iteraciones = [50]
```

Pasamos al bucle principal de entrenamiento, que consiste en tres bucles anidados que recorrerán todos los valores de Lambdas, iteraciones y capas ocultas que queramos probar.

En primer lugar, generamos pesos aleatorios y se los asignamos a las Thetas que vamos a optimizar, uniéndolas en un mismo array. Después, utilizamos la función “minimize” de la librería Optimize de Scipy para ejecutar la función de propagación hacia atrás sobre este array de pesos, obteniendo así las Thetas optimizadas con unos valores determinados de Lambdas, iteraciones y capas ocultas.

```
theta1 = pesosAleatorios(len(X[0]), t)
theta2 = pesosAleatorios(t, etiquetas)

thetas = [theta1, theta2]

pesos = np.concatenate([thetas[i].ravel() for i,_ in enumerate(thetas)])

thetasguays = opt.minimize(fun=backprop, x0=pesos, args=(entradas, t, etiquetas, X, y_onehot, i), method='TNC', jac=True, options={'maxiter':j})

theta1opt = np.reshape(thetasguays.x[:t * (entradas + 1)], (t, (entradas + 1)))
theta2opt = np.reshape(thetasguays.x[t * (entradas + 1):], (etiquetas, (t + 1)))
```

Una vez obtenidas las Thetas óptimas, calculamos el porcentaje de acierto utilizando la técnica de propagación hacia adelante, y si es el mejor resultado obtenido hasta entonces, guardamos los pesos obtenidos para su posterior uso.

```
a1, x2, a2, z3, h = forward_prop(X, theta1opt, theta2opt)

res = calcAciertos(y, h)

if(res > porcentaje):
    porcentaje = res
    mejores_thetas = [theta1opt, theta2opt]
```

Ya habiendo pasado por todas las Lambdas, todas las capas ocultas, y todas las iteraciones determinadas, tendremos guardados los pesos que hayan obtenido el mejor porcentaje de acierto en el entrenamiento, por lo que guardaremos en un archivo .mat dichos pesos, y podremos pasar a la fase de Testing.

En la fase de test de la red neuronal (archivo “testRedNeuronal.py”), accederemos a la matriz de datos “dataMat.mat” y a la matriz de pesos que acabamos de obtener “pesos.mat” para realizar la propagación hacia adelante y obtener el porcentaje de aciertos de los pesos óptimos con las imágenes de test.

```
data = loadmat("dataMat.mat")

X = data["Xtest"]
y = data["ytest"].ravel()

pesos = loadmat("pesos.mat")

theta1 = pesos["theta1"]
theta2 = pesos["theta2"]

a1, z2, a2, z3, a3 = forward_prop(X, theta1, theta2)

res = calcAciertos(y, a3)
```

- **Resultados:**

Para entrenar la red neuronal, hicimos pruebas con dos matrices distintas (sólo abejas y avispas/abejas, avispas, insectos y no-insectos), y los resultados son bastante diferentes.

1) Matriz “dataMat.mat” (sólo abejas y avispas)

```
--------------------------------------------------------------------------
Resultado con lambda:0.1, iteraciones 20, capas ocultas 25
40.30769
Tiempo: 181.82
--------------------------------------------------------------------------
Resultado con lambda:0.1, iteraciones 20, capas ocultas 50
60.73846
Tiempo: 354.48
--------------------------------------------------------------------------
Resultado con lambda:1, iteraciones 5, capas ocultas 25
60.36923
Tiempo: 51.82
--------------------------------------------------------------------------
```

Los resultados varían con las diferentes cantidades de capas ocultas e iteraciones y lambdas, pero generalmente, los mejores resultados no pasaban el 62% de aciertos.

2) Matriz “dataMat2.mat” (4 etiquetas)

```
--------------------------------------------------------------------------
Resultado con lambda:0.1, iteraciones 1, capas ocultas 50
6.917688266199649
Tiempo: 23.864241099999994
--------------------------------------------------------------------------
Resultado con lambda:1, iteraciones 1, capas ocultas 25
27.670753064798596
Tiempo: 11.107253
--------------------------------------------------------------------------
Resultado con lambda:1, iteraciones 1, capas ocultas 50
44.176882661996494
Tiempo: 21.7630147
--------------------------------------------------------------------------
```

Como podemos ver, los resultados son bastante peores, habiendo máximos de 45% de acierto.

En la fase de testing de los pesos con mejores resultados en el entrenamiento, el porcentaje final de aciertos era algo mayor, pero tampoco superaban el 70% en el mejor de los casos.

```
------------------------------------------
Se ha obtenido un resultado del 61.01% de aciertos
------------------------------------------
```

4. Realización de la Regresión Logística:

Tras obtener la matriz con los datos de todas las imágenes, haremos uso del script (“regresionLogistica.py”), el cual usará las imágenes clasificadas como de entrenamiento para así, usando la técnica de ***oneVsAll*** para conseguir la matriz de thetas óptimas y guardarla tal y como hicimos en la práctica 2.

```
def oneVsAll(X, y, num_etiquetas, reg, maxIt, landa):

    ThetasMatriz = np.zeros((num_etiquetas, X.shape[1]))

    i = 0
    while i < num_etiquetas:

        print("Numero de etiquetas procesadas: ", i + 1, " Etiquetas: ", num_etiquetas, " Lamda: ", landa, " y ", " Iteraciones: " ,maxIt)
        auxY = (y == i).astype(int)
        ThetasMatriz[i, :] = calcOptTheta(X, auxY, maxIt, landa)
        i += 1

    return ThetasMatriz
```

```
def calcOptTheta(X, Y, maxIt, landa):
    result = opt.minimize(
        fun=coste_y_gradiente,
        x0=np.zeros(X.shape[1]),
        args=(X, Y, landa),
        method='TNC',
        jac=True,
        options={'maxiter': maxIt})

    return result.x


def coste_y_gradiente(x0, X, Y, landa):
    return coste(x0,X,Y,landa), gradiente(x0, X, Y, landa)
```

```
def sigmoide(z):
    return 1/(1+np.exp(-z))

def coste(theta, X, Y, lmbda):
    H = sigmoide(np.matmul(X, theta))

    cost = (-1 / (len(X))) * (np.dot(Y, np.log(H)) + np.dot((1 - Y), np.log(1 - H)))

    ret = cost + np.sum(np.square(theta)) * lmbda / 2 * m

    return ret

def gradiente(theta, X, Y, lmbda):

    H = sigmoide(np.matmul(X, theta))

    grad = (1 / len(Y)) * np.matmul(X.T, H - Y)

    aux = np.r_[[0], theta[1:]]

    ret = grad +(lmbda*aux/m)

    return ret
```

Realizaremos esta operación con una serie de lambdas y números de iteraciones máximas distintos:

```
lamdas = [0.001, 0.01, 0.1, 1, 10, 50, 100, 250, 500]
maxIterations = [50, 100, 150, 200, 250, 300]
num_labels = len(np.unique(yaux))

for i in lamdas:
    for r in maxIterations:
        landa = i
        one = (oneVsAll(X, yaux, num_labels, i, r, landa))
        testedThetasValues[p] = np.mean(one)
        testedThetas.append(one)

        myLandas.append(i)
        myIter.append(r)
        aciertos.append(calcAciertos(X, yaux, one))
        p += 1

val =  aciertos.index(max(aciertos)) #Lo mismo que max(testedThetasValues)

dict1 = {
    "thetas" : testedThetas
}

out.savemat("regresionMatTrainedBW.mat", dict1)
```

También calcularemos el porcentaje de aciertos con los ejemplos de entrenamiento:

```
def calcAciertos(X, Y, t):
    cont = 0
    aciertos = 0
    totales = len(Y)
    dimThetas = len(t)
    valores = np.zeros(dimThetas)

    for i in X:
        p = 0
        for x in range(dimThetas):
            valores[p] = sigmoide(np.dot(i, t[x]))
            p+=1

        r = np.argmax(valores)

        if(r==Y[cont]):
            aciertos+=1

        cont += 1

    porcentaje = aciertos / totales * 100
    return porcentaje
```

Tras realizar el entrenamiento usaremos el script (“testRegresionLineal.py”) para calcular el porcentaje de acierto para los casos de test y validación.

- **Resultados:**

La regresión logística la hemos realizado con 4 matrices de datos distintos, para así poder comprobar su funcionamiento y comparar resultados:

1) La matriz de imágines en color (Image.convert(“P”), con dos etiquetas (Avispa o abeja), para que sea capaz de distinguir entre ambas.

```
Mejor porcentaje de acierto para entrenamiento:  68.59487179487179% de acierto. Lambda =  500  iteraciones 50
```

```
Mejor porcentaje de acierto con los datos de validacion:  67.56923076923077%
Mejor porcentaje de acierto con los datos de test:  68.38868388683886%
```

2) La matriz de imágenes en blanco y negro (Image.convert(“L”), con dos etiquetaws (Avispa o abeja).

```
Mejor porcentaje de acierto para entrenamiento:  60.75897435897436% de acierto. Lambda =  0.001  iteraciones 50
```

```
Mejor porcentaje de acierto con los datos de validacion:  60.61538461538461%
Mejor porcentaje de acierto con los datos de test:  57.38007380073801%
```

3) La matriz de imágenes en color (Image.convert(“P”), con cuatro etiquetas (Avispa, abeja, otro insecto y no insecto).

```
Mejor porcentaje de acierto para entrenamiento:  45.19848219497957% de acierto. Lambda =  0.001  iteraciones 50
```

```
Mejor porcentaje de acierto con los datos de validacion:  47.8984238178634%
Mejor porcentaje de acierto con los datos de test:  49.14660831509847%
```

4) La matriz de imágenes en blanco y negro (Image.convert("L"), con cuatro etiquetas (Avispa, abeja, otro insecto y no insecto).

```
Mejor porcentaje de acierto para entrenamiento:  43.096906012842965% de acierto. Lambda =  0.001  iteraciones 50
```

```
Mejor porcentaje de acierto con los datos de validacion:  43.30122591943958%
Mejor porcentaje de acierto con los datos de test:  43.7636761487965%
```

5. Realización de las SVMs

Partimos de la misma base que los otros métodos, teniendo en posesión la matriz de datos de entrenamiento, validación y test de las imágenes. Además, vamos a crear un pequeño array de valores para nuestros parámetros C y sigma, que utilizaremos para saber cuáles de ellos son los óptimos en cuanto a resultados.

```
data = loadmat('dataMat.mat')
X = data['X']
y = data['y'].ravel()
Xval = data['Xval']
yval = data['yval']
Xtest = data['Xtest']
ytest = data['ytest']

vals = np.array([0.01, 0.1, 1, 10, 30])

svms = []
mejorsvm = 0

c = 0
sigma = 0

porcentaje = 0

cont = 0
```

Hemos utilizado sólo 5 valores diferentes para C y sigma puesto que cuantos más valores haya, más tiempo de ejecución supone, y tampoco queremos que nuestros ordenadores se fundan demasiado tiempo.

Ya declarados los datos, pasamos al bucle principal, en el que se irán probando valores de C y sigma y se irán creando SVCs con diferentes resultados. También, al igual que en la red neuronal, nos quedamos con el modelo que mejor porcentaje de acierto haya obtenido.

```
for i in range(len(vals)):
    for j in range(len(vals)):
        svms.append(SVC(kernel='rbf', C=vals[i], gamma=1/(2*(vals[j]**2))))
        svms[cont].fit(X, y)

        h = svms[cont].predict(Xval)
        cosa = calcula_porcentaje(yval.ravel(), h)

        if(cosa > porcentaje):
            porcentaje = cosa
            c = vals[i]
            sigma = vals[j]
            mejorsvm = cont

        cont = cont+1

        print(cosa)
        print(cont)
```

```
60.36923
1
60.36923
2
60.36923
3
60.36923
4
60.36923
5
```

```
60.36923
6
60.36923
7
60.36923
8
70.33846
9
67.56923
10
```

```
60.36923
11
60.36923
12
60.36923
13
74.58462
14
70.58462
15
```

```
60.36923
16
60.36923
17
60.36923
18
74.83077
19
72.18462
20
```

Una vez obtenido el modelo con mejor resultado, creamos una última SVC y la comparamos con los valores de la matriz de testing, obteniendo el resultado final.

```
svm = SVC(kernel='rbf', C=c, gamma=1/(2*(sigma**2)))
svm.fit(X, y)

h = svm.predict(Xtest)
resultado = calcula_porcentaje(ytest.ravel(), h, 3)

print(resultado)
```

```
74.78475
```

6. **Apéndice (Código):**

- **generaMat2.py** (generaMat.py es igual pero sin las imagenes de insectos y no-insectos)

```
carpeta = "kaggle_bee_vs_wasp"

# Las imagenes buenas
bee_train = os.listdir(carpeta+"/bee1")
wasp_train = os.listdir(carpeta+"/wasp1")
insect_train = os.listdir(carpeta+"/other_insect")
noinsect_train = os.listdir(carpeta+"/other_noinsect")

# Las imagenes no tan buenas
bee_test = os.listdir(carpeta+"/bee2")
wasp_test = os.listdir(carpeta+"/wasp2")
insect_test = os.listdir(carpeta+"/other_insect2")
noinsect_test = os.listdir(carpeta+"/other_noinsect2")

################################################## TRAIN
resizedbees = []
resizedwasps = []
resizedinsect = []
resizednoinsect = []
etiqueta = [] # 0 para abeja, 1 para avispa, 2 para insecto, 3 para otro
```

```
# Generar imagenes resizeadas
for i in bee_train:
    if os.path.isfile(carpeta + "/bee1/" + i):
        aux = Image.open(carpeta + "/bee1/" + i).convert('P')
        aux = aux.resize((100,100), Image.ANTIALIAS)
        aux = np.asarray(aux)/255.0
        resizedbees.append(aux)
        etiqueta.append(0)

for i in wasp_train:
    if os.path.isfile(carpeta + "/wasp1/" + i):
        aux = Image.open(carpeta + "/wasp1/" + i).convert('P')
        aux = aux.resize((100,100), Image.ANTIALIAS)
        aux = np.asarray(aux)/255.0
        resizedwasps.append(aux)
        etiqueta.append(1)

for i in insect_train:
    if os.path.isfile(carpeta + "/other_insect/" + i):
        aux = Image.open(carpeta + "/other_insect/" + i).convert('P')
        aux = aux.resize((100,100), Image.ANTIALIAS)
        aux = np.asarray(aux)/255.0
        resizedinsect.append(aux)
        etiqueta.append(2)

for i in noinsect_train:
    if os.path.isfile(carpeta + "/other_noinsect/" + i):
        aux = Image.open(carpeta + "/other_noinsect/" + i).convert('P')
        aux = aux.resize((100,100), Image.ANTIALIAS)
        aux = np.asarray(aux)/255.0
        resizednoinsect.append(aux)
        etiqueta.append(3)
```

```
x_train = np.concatenate((resizedbees, resizedwasps, resizedinsect, resizednoinsect), axis=0)
x_train_etiqueta = np.asarray(etiqueta)
x_train_etiqueta = x_train_etiqueta.reshape(x_train_etiqueta.shape[0], 1)

################################################ TEST
resizedbees = []
resizedwasps = []
resizedinsect = []
resizednoinsect = []
etiqueta = [] # 0 para abeja, 1 para avispa, 2 para insecto, 3 para otro
```

```
# Generar imagenes resizeadas
for i in bee_test:
    if os.path.isfile(carpeta + "/bee2/" + i):
        aux = Image.open(carpeta + "/bee2/" + i).convert('P')
        aux = aux.resize((100,100), Image.ANTIALIAS)
        aux = np.asarray(aux)/255.0
        resizedbees.append(aux)
        etiqueta.append(0)

for i in wasp_test:
    if os.path.isfile(carpeta + "/wasp2/" + i):
        aux = Image.open(carpeta + "/wasp2/" + i).convert('P')
        aux = aux.resize((100,100), Image.ANTIALIAS)
        aux = np.asarray(aux)/255.0
        resizedwasps.append(aux)
        etiqueta.append(1)

for i in insect_test:
    if os.path.isfile(carpeta + "/other_insect2/" + i):
        aux = Image.open(carpeta + "/other_insect2/" + i).convert('P')
        aux = aux.resize((100,100), Image.ANTIALIAS)
        aux = np.asarray(aux)/255.0
        resizedinsect.append(aux)
        etiqueta.append(2)

for i in noinsect_test:
    if os.path.isfile(carpeta + "/other_noinsect2/" + i):
        aux = Image.open(carpeta + "/other_noinsect2/" + i).convert('P')
        aux = aux.resize((100,100), Image.ANTIALIAS)
        aux = np.asarray(aux)/255.0
        resizednoinsect.append(aux)
        etiqueta.append(3)
```

```
x_test = np.concatenate((resizedbees, resizedwasps, resizedinsect, resizednoinsect), axis=0)
x_test_etiqueta = np.asarray(etiqueta)
x_test_etiqueta = x_test_etiqueta.reshape(x_test_etiqueta.shape[0], 1)


################################################


X = np.concatenate((x_train, x_test), axis=0)
y = np.concatenate((x_train_etiqueta, x_test_etiqueta), axis=0)
X = X.reshape(X.shape[0], X.shape[1]*X.shape[2])


# Separar en entrenamiento, test y validacion
Xtrain, Xtest, ytrain, ytest = train_test_split(np.asarray(X), np.asarray(y), test_size=0.2)
Xtrain, Xval, ytrain, yval = train_test_split(np.asarray(Xtrain), np.asarray(ytrain), test_size=0.25)


# Guardar en un .mat
matX = np.array(Xtrain)
maty = np.array(ytrain)

dicc = {
    "X": matX,
    "y": maty,
    "Xval": Xval,
    "yval": yval,
    "Xtest": Xtest,
    "ytest": ytest
}

out.savemat("dataMat2.mat", dicc)
```

- **beesPlot.py** (para generar un cuadro de imagenes, solo usado en la presentación)

```
carpeta = "kaggle_bee_vs_wasp"

# Las imagenes no tan buenas
bee_test = os.listdir(carpeta+"/bee1")

resizedbees = []
etiqueta = [] # 0 para abeja, 1 para avispa

# Generar imagenes resizeadas
for i in bee_test:
    if os.path.isfile(carpeta + "/bee1/" + i):
        aux = Image.open(carpeta + "/bee1/" + i).convert('P')
        aux = aux.resize((100,100), Image.ANTIALIAS)
        aux = np.asarray(aux)/255.0
        resizedbees.append(aux)
        etiqueta.append(0)

imgbees = []
imgwasps = []
```

```
for i in range(1, len(resizedbees)):
    imgbees.append(resizedbees[i])

w=10
h=10
fig=plt.figure(figsize=(15,15))
columns = 4
rows = 5

for i in range(1,21):

    img = np.random.randint(10, size=(h,w))
    fig.add_subplot(rows, columns, i)
    plt.imshow(resizedbees[random.randint(1,len(resizedbees))])
    converted_num = str(i)
    plt.title("bee - "+converted_num)
    i=int(i)

plt.show()
```

- **redNeuronal.py**

```
data = loadmat("dataMat.mat")

X = data["X"]
y = data["y"].ravel()
X2 = data["Xval"]
y2 = data["yval"].ravel()

entradas = X.shape[1]
etiquetas = len(np.unique(y))
ocultas = [25, 50]
landas = [0.01, 0.1, 1, 10]
iteraciones = [5, 20]

y_onehot = np.zeros((len(y), etiquetas))
for i in range(len(y)):
    y_onehot[i][y[i]] = 1

mejores_thetas = []
porcentaje = 0
```

```
for i in landas:
    for j in iteraciones:
        for t in ocultas:

            tic = timeit.default_timer()
            theta1 = pesosAleatorios(entradas, t)
            theta2 = pesosAleatorios(t, etiquetas)

            thetas = [theta1, theta2]

            pesos = np.concatenate([thetas[i].ravel() for i,_ in enumerate(thetas)])

            thetasguays = opt.minimize(fun=backprop, x0=pesos, args=(entradas, t, etiquetas, X, y_onehot, i),
                method='TNC', jac=True, options={'maxiter':j})

            theta1opt = np.reshape(thetasguays.x[:t * (entradas + 1)], (t, (entradas + 1)))
            theta2opt = np.reshape(thetasguays.x[t * (entradas + 1):], (etiquetas, (t + 1)))

            a1, x2, a2, z3, h = forward_prop(X2, theta1opt, theta2opt)

            res = calcAciertos(y2, h)

            if(res > porcentaje):
                porcentaje = res
                mejores_thetas = [theta1opt, theta2opt]

            toc = timeit.default_timer()
            print("------------------------------------------------------------------------")
            print("Resultado con lambda:" + str(i) + ", iteraciones " + str(j) + ", capas ocultas " + str(t))
            print(res)
            print("Tiempo: " + str(round((toc-tic), 2)))
```

```
dicc = {
    "theta1" : mejores_thetas[0],
    "theta2" : mejores_thetas[1]
}

out.savemat("pesos.mat", dicc)
```

- **testRedNeuronal.py**

```
data = loadmat("dataMat.mat")

X = data["Xtest"]
y = data["ytest"].ravel()

pesos = loadmat("pesos.mat")

theta1 = pesos["theta1"]
theta2 = pesos["theta2"]

a1, z2, a2, z3, a3 = forward_prop(X, theta1, theta2)

res = calcAciertos(y, a3)

print("-------------------------------------------")
print("Se ha obtenido un resultado del " + str(round(res, 2)) + "% de aciertos")
print("-------------------------------------------")
```

- **regresionLogistica.py:**

```python
def sigmoide(z):
    return 1/(1+np.exp(-z))

def coste(theta, X, Y, lmbda):
    H = sigmoide(np.matmul(X, theta))

    cost = (-1 / (len(X))) * (np.dot(Y, np.log(H)) + np.dot((1 - Y), np.log(1 - H)))

    ret = cost + np.sum(np.square(theta)) * lmbda / 2 * m

    return ret

def gradiente(theta, X, Y, lmbda):

    H = sigmoide(np.matmul(X, theta))

    grad = (1 / len(Y)) * np.matmul(X.T, H - Y)

    aux = np.r_[[0], theta[1:]]

    ret = grad +(lmbda*aux/m)

    return ret

def oneVsAll(X, y, num_etiquetas, reg, maxIt, landa):

    ThetasMatriz = np.zeros((num_etiquetas, X.shape[1]))

    i = 0
    while i < num_etiquetas:

        print("Numero de etiquetas procesadas: ", i + 1, " Etiquetas: ", num_etiquetas, " Lamda: ", landa, " y ", " Iteraciones: " ,maxIt)
        auxY = (y == i).astype(int)
        ThetasMatriz[i, :] = calcOptTheta(X, auxY, maxIt, landa)
        i += 1

    return ThetasMatriz
def calcOptTheta(X, Y, maxIt, landa):
    result = opt.minimize(
        fun=coste_y_gradiente,
        x0=np.zeros(X.shape[1]),
        args=(X, Y, landa),
        method='TNC',
        jac=True,
        options={'maxiter': maxIt})

    return result.x
```

```python
def coste_y_gradiente(x0, X, Y, landa):
    return coste(x0,X,Y,landa), gradiente(x0, X, Y, landa)

def calcAciertos(X, Y, t):
    cont = 0
    aciertos = 0
    totales = len(Y)
    dimThetas = len(t)
    valores = np.zeros(dimThetas)

    for i in X:
        p = 0
        for x in range(dimThetas):
            valores[p] = sigmoide(np.dot(i, t[x]))
            p+=1

        r = np.argmax(valores)

        if(r==Y[cont]):
            aciertos+=1

        cont += 1

    porcentaje = aciertos / totales * 100
    return porcentaje



data = loadmat('dataMat.mat')
X = data['X']
y = data['y']

yaux = np.ravel(y)

Xval = data['Xval']
yval = data['yval']

Xtest = data['Xtest']
ytest = data['ytest']

m = X.shape[0]
Xones = np.hstack([np.ones([m, 1]), X])

m2 = Xval.shape[0]
Xvalones = np.hstack([np.ones([m2, 1]), Xval])
```

```python
testedThetasValues = dict()
testedThetas = []
p = 0

aciertos = []
myLandas = []
myIter = []

lamdas = [0.001, 0.01, 0.1, 1, 10, 50, 100, 250, 500]
maxIterations = [50, 100, 150, 200, 250, 300]
num_labels = len(np.unique(yaux))

for i in lamdas:
    for r in maxIterations:
        landa = i
        one = (oneVsAll(X, yaux, num_labels, i, r, landa))
        testedThetasValues[p] = np.mean(one)
        testedThetas.append(one)

        myLandas.append(i)
        myIter.append(r)
        aciertos.append(calcAciertos(X, yaux, one))
        p += 1

val =  aciertos.index(max(aciertos)) #Lo mismo que max(testedThetasValues)

dict1 = {
    "thetas" : testedThetas
}

out.savemat("regresionMatTrained.mat", dict1)

print("Mejor porcentaje de acierto para entrenamiento: " ,str(aciertos[val]) + "% de acierto. Lambda = ", myLandas[val], " iteraciones", myIter[val])
```

- **regresionLogisticaTest.py:**

```
def sigmoide(z):
    return 1/(1+np.exp(-z))


def porcentajeAciertos(X, Y, t):
    cont = 0
    aciertos = 0
    totales = len(Y)
    valores = np.zeros(len(t))

    for i in X:
        p = 0
        for x in range(len(t)):
            valores[p] = sigmoide(np.dot(i, t[x]))
            p+=1

        r = np.argmax(valores)

        if(r==Y[cont]):
            aciertos+=1

        cont += 1

    porcentaje = aciertos / totales * 100
    return porcentaje
```

```
dataTest = loadmat('dataMat2BW.mat')
dataReg = loadmat('regresionMatTrainedCompleteBW.mat')

Xval = dataTest['Xval']
yval = dataTest['yval']
yaux = np.ravel(yval)

Xtest = dataTest['Xtest']
ytest = dataTest['ytest']
ytest = np.ravel(ytest)

m2 = Xval.shape[0]
Xvalones = np.hstack([np.ones([m2, 1]), Xval])

thetas = dataReg["thetas"]

aciertos = []
for i in thetas:
    perc = porcentajeAciertos(Xval, yaux, i)
    aciertos.append(perc)
    print(perc)

maxVal =  aciertos.index(max(aciertos))
print("Mejor porcentaje de acierto con los datos de validacion: ", str(aciertos[maxVal]) + "%")

porcentaje = porcentajeAciertos(Xtest, ytest, thetas[maxVal])

print("Mejor porcentaje de acierto con los datos de test: ",str(porcentaje) + "%")
```

- **svm.py**

```
data = loadmat('dataMat.mat')
X = data['X']
y = data['y'].ravel()
Xval = data['Xval']
yval = data['yval']
Xtest = data['Xtest']
ytest = data['ytest']

vals = np.array([0.01, 0.1, 1, 10, 30])

svms = []
mejorsvm = 0

c = 0
sigma = 0

porcentaje = 0

cont = 0
```

```
for i in range(len(vals)):
    for j in range(len(vals)):
        svms.append(SVC(kernel='rbf', C=vals[i], gamma=1/(2*(vals[j]**2))))
        svms[cont].fit(X, y)

        h = svms[cont].predict(Xval)
        cosa = calcula_porcentaje(yval.ravel(), h)

        if(cosa > porcentaje):
            porcentaje = cosa
            c = vals[i]
            sigma = vals[j]
            mejorsvm = cont

        cont = cont+1

        print(cosa)
        print(cont)
```

```
svm = SVC(kernel='rbf', C=c, gamma=1/(2*(sigma**2)))
svm.fit(X, y)

h = svm.predict(Xtest)
resultado = calcula_porcentaje(ytest.ravel(), h, 3)

print(resultado)
```

- **Funciones auxiliares:**

```
def sigmoid(z):
    return (1 / (1 + np.exp(-z)))

def coste(X, Y, theta1, theta2):
    m = Y.shape[0]

    a1, z2, a2, z3, a3 = forward_prop(X, theta1, theta2)

    ret = np.sum(np.sum(-Y * np.log(a3) - (1-Y) * np.log(1-a3)))

    return ret/m

def costeReg(X, Y, m, theta1, theta2, K):
    return coste(X, Y, theta1, theta2) + ((K/(2*m)) * np.sum(np.square(theta1[:, 1:]))) + (np.sum(np.square(theta2[:, 1:])))

def forward_prop(X, theta1, theta2):
    m = X.shape[0]

    a1 = np.hstack([np.ones([m, 1]), X])
    z2 = np.dot(a1, theta1.T)
    a2 = np.hstack([np.ones([m, 1]), sigmoid(z2)])
    z3 = np.dot(a2, theta2.T)
    a3 = sigmoid(z3)

    return a1, z2, a2, z3, a3
```

```python
def backprop(params_rn, num_entradas, num_ocultas, num_etiquetas, X, y, K):
    m = X.shape[0]

    theta1 = np.reshape(params_rn[:num_ocultas * (num_entradas + 1)], (num_ocultas, (num_entradas+1)))
    theta2 = np.reshape(params_rn[num_ocultas * (num_entradas + 1): ], (num_etiquetas, (num_ocultas+1)))

    delta1 = np.zeros((num_ocultas, num_entradas+1))
    delta2 = np.zeros((num_etiquetas, num_ocultas + 1))

    a1, z2, a2, z3, a3 = forward_prop(X, theta1, theta2)

    coste = costeReg(X, y, m, theta1, theta2, K)

    for t in range(m):
        a1t = a1[t, :]
        a2t = a2[t, :]
        ht = a3[t, :]
        yt = y[t]

        d3t = ht - yt
        d2t = np.dot(theta2.T, d3t) * (a2t * (1 - a2t))

        delta1 = delta1 + np.dot(d2t[1:,np.newaxis], a1t[np.newaxis,:])
        delta2 = delta2 + np.dot(d3t[:,np.newaxis], a2t[np.newaxis,:])

    delta1 = delta1 / m
    delta2 = delta2 / m

    delta1[:, 1:] = delta1[:, 1:] + (K * theta1[:, 1:]) / m
    delta2[:, 1:] = delta2[:, 1:] + (K * theta2[:, 1:]) / m

    gradiente = np.concatenate((np.ravel(delta1), np.ravel(delta2)))

    return coste, gradiente
```

```python
def pesosAleatorios(tam1, tam2):
    aux = 0.12
    ret = np.zeros((tam2, 1+tam1))
    ret = np.random.rand(tam2, 1+tam1) * (2*aux) - aux
    return ret

def calcAciertos(Y, h):
    m = Y.shape[0]
    res = np.empty(m)

    for i in range(m):
        res[i] = np.argmax(h[i])
    res = res.T

    yes = (Y == res)
    aciertos = np.sum(yes)

    return round((aciertos/m) * 100, 5)
```

- **Enlace al GitHub:**

Si se quieren comprobar los scripts más cómodamente, aquí dejamos el enlace al repositorio donde están todas las imágenes y todo el código:

GitHub