

Proyecto Final

David Godoy Ruiz
Eva Lucas Leiro



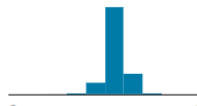



Para este proyecto hemos usado un dataset con datos sobre empresas de Taiwan entre los años 1999 y 2009 para entrenar un modelo que pueda predecir si entraron o no en bancarrota

Inicialización de los datos:

El archivo csv contiene 95 datos de 6819 empresas distintas. Hemos separado los datos en dos mitades: una para entrenar el modelo y otra para probar su eficacia para asegurarnos de no hacer sobreajuste sobre una sola muestra. Además hemos eliminado una columna (Net income flag) que no influye en el resultado al ser todas iguales.

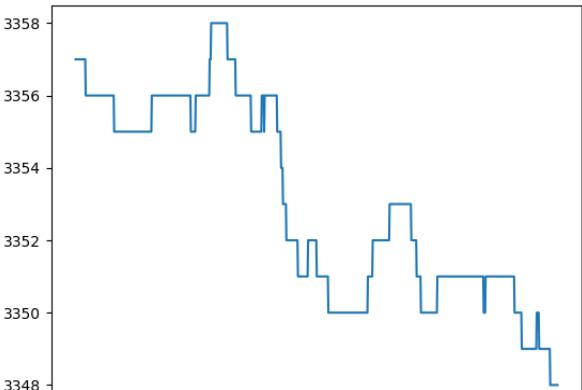
Por último hemos normalizado cada columna ya que las magnitudes de los datos varían enormemente.

Nota: las gráficas evalúan el número de aciertos sobre el conjunto de prueba, consistente en 3410 datos

# Bankrupt?	# ROA(C) before int...	# ROA(A) before int...	# ROA(B) before int...	# Operating Gross ...	# Realized Sales Gr...
Class label
					
0	0	0	0	0	0
1	0.370594257300249	0.424389446140427	0.40574977247176	0.601457213277793	0.601457213277793
1	0.464290937454297	0.53821412996075	0.516730017666899	0.610235085544617	0.610235085544617
1	0.426071271876371	0.499018752725687	0.472295090743616	0.601450006486113	0.601363524985947
1	0.399844001364988	0.451264718709115	0.457733283366347	0.583541129160121	0.583541129160121
1	0.465022181055916	0.538432184910597	0.522297767546443	0.598783493564335	0.598783493564335
1	0.388680349046946	0.415176624509376	0.419133786605279	0.590171377506162	0.590250652214647
0	0.390922829425243	0.445704317488007	0.436158252583115	0.619949840729904	0.619949840729904
0	0.508360551845171	0.570922372437854	0.559077038385353	0.601738278153332	0.60171665777829
0	0.48851947545459	0.545137374618404	0.5432839017078	0.603612043990256	0.603612043990256

Regresión Logística:

Primero hemos probado con la técnica de regresión logística. Para mejorar el modelo hemos probado con varios parámetros de regularización y hemos visto que no tiene ninguna influencia a baja escala (0-10), pero a mayor escala sí varía la precisión del modelo, como se muestra en la gráfica



```

def regresion_logistica_reg(X, Y, Xval, Yval):

    m=np.shape(X)[0]
    n=np.shape(X)[1]

    X_ones = np.hstack([np.ones([m, 1]), X])

    mval = np.shape(Xval)[0]
    Xval_ones = np.hstack([np.ones([mval, 1]), Xval])

    max_correctos = 0
    max_lambda = 0

    plt.figure()
    correctArray = np.zeros(1000)

    for lambd in np.arange(0, 1000, 1):
        theta = np.full(n+1, 0)
        result = opt.fmin_tnc( func=coste_reg , x0=theta , fprime=gradiente_reg
, args =(X_ones,Y, lambd), messages= 0)
        theta_opt = result[0]
        correctos = evaluacion(theta_opt,Xval_ones,Yval)
        correctArray[lambd] = correctos
        if (correctos > max_correctos):
            max_correctos = correctos
            max_lambda = lambd

    plt.plot(np.linspace(1,len(correctArray),len(correctArray),
dtype=int),correctArray)
    plt.savefig("reg_logistica.png")

    return max_correctos/mval, max_lambda

```

Redes Neuronales:

A continuación hemos probado a entrenar una red neuronal. Tras probar con múltiples opciones sobre número de nodos en la capa oculta hemos decidido dejarlo en 25, y para el parámetro de regularización hemos hecho algo similar que con la regresión logística, probando con varios, solo que el parámetro tiene más influencia a escalas menores. Hemos visto una ligera variación de la precisión entre pruebas debido a la inicialización aleatoria de los pesos en la red neuronal.

```

def red_neuronal(X, y ,XVal, YVal):
    X, y, yval = load(X, y ,XVal, YVal)

    num_entradas = np.shape(X)[1]
    num_ocultas = 25
    num_etiquetas = 2

    ini = 0.12
    i=100

    plt.figure()
    lambdArray = np.zeros(10)

    max_correctos = 0
    max_lambda = 0

    for lambd in np.arange(0, 10, 1):
        pesos=
np.random.uniform(-ini,ini,(num_entradas+1)*num_ocultas+(num_ocultas+1)*num_etiq
uetas)
        sol = opt.minimize(fun=backprop, x0=pesos,
args=(num_entradas,num_ocultas, num_etiquetas,X, y ,lambd), jac= True,method =
'TNC', options ={'maxiter' :i})

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

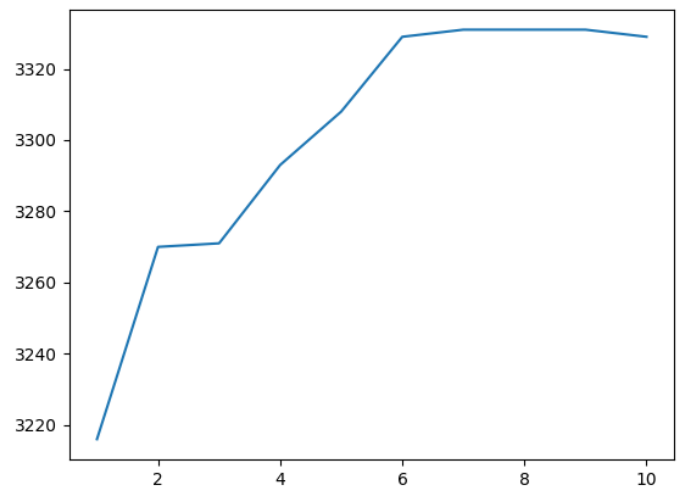
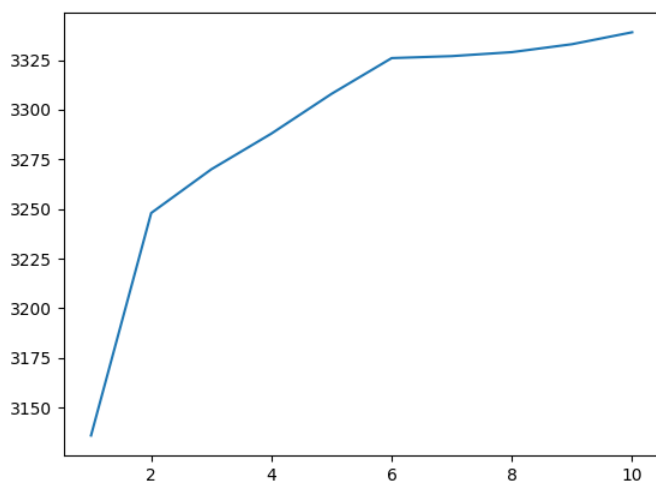
        m = np.shape(X)[0]

        A1, A2, H = propagacion(XVal, theta1, theta2)

        maxChance = H.argmax(axis= 1)
        res = yval.argmax(axis= 1)
        correctos = np.sum(maxChance == res)
        lambdArray[lambd] = correctos
        if (correctos > max_correctos):
            max_correctos = correctos
            max_lambda = lambd
        plt.plot(np.linspace(1,len(lambdArray),len(lambdArray),
dtype=int),lambdArray)
        plt.savefig("redes_neuronales.png")

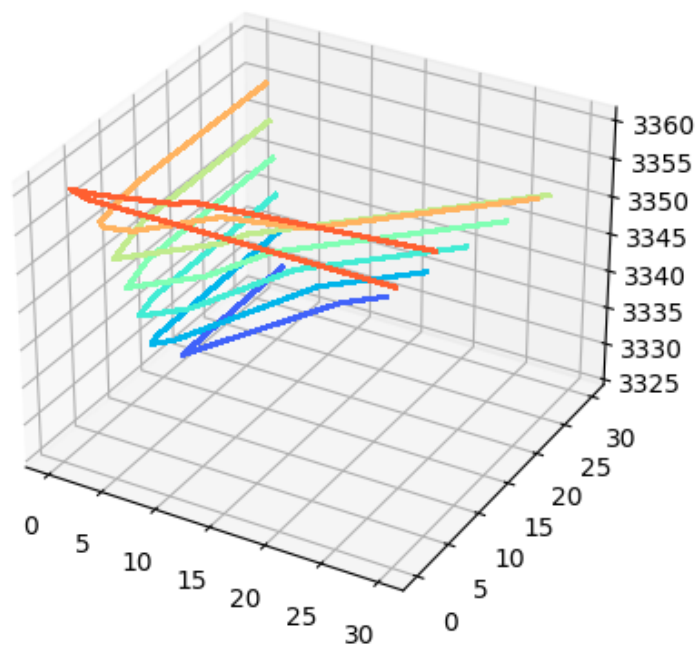
    return correctos/m, max_lambda

```



SVM:

Por último hemos utilizado un clasificador SVM. Para ello hemos usado el kernel gaussiano variando los parámetros C y sigma para ver cuál se ajusta mejor al modelo.



```
def eleccionParams(X, y, Xval, yval):
    C_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    sigma_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    scores = np.zeros((len(C_vec), len(sigma_vec)))
```

```

plt.figure()
for i in range(len(C_vec)):
    for j in range(len(sigma_vec)):
        svm = SVC(kernel= 'rbf' , C=C_vec[i], gamma = 1/(2*sigma_vec[j]**2))
        svm.fit(X, y)
        scores[i][j] = calculaScore(svm, Xval, yval)

axes = plt.axes(projection= '3d')
axes.contour(C_vec, sigma_vec, scores, cmap='rainbow',linewidth=0,
antialiased=False)
plt.savefig("svm.png")
index = np.unravel_index(np.argmax(scores), scores.shape)

svm = SVC(kernel= 'rbf' , C=C_vec[index[0]], gamma =
1/(2*sigma_vec[index[1]]**2))
svm.fit(X, y)
return svm

#-----

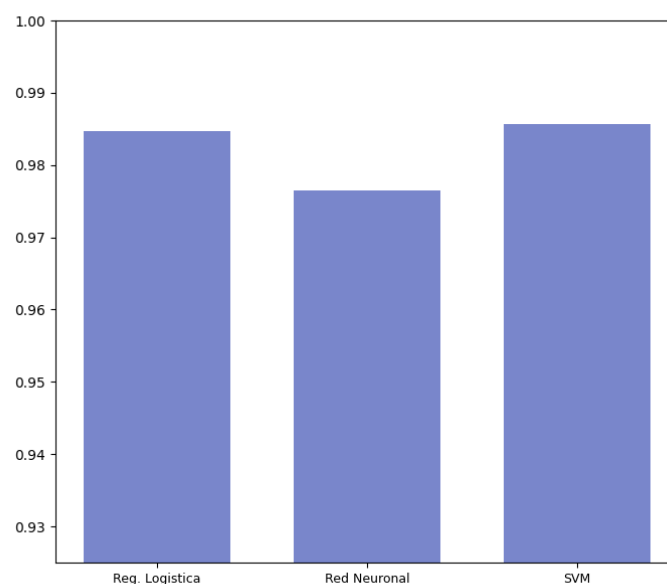
def svm_proyecto( X, y, Xval, yval):

    svm = eleccionParams(X, y, Xval, yval)
    return calculaScore(svm,Xval, yval)/np.shape(Xval)[0]

```

Comparativa:

Para concluir hemos comparado los tres métodos y hemos visto que la técnica que mejor predice las bancarrotas de las empresas en nuestro caso es el uso de SVM, seguido muy de cerca por la regresión logística, esto último al ser una variable de salida booleana a la que se ajusta muy bien el modelo.



Código utilizado:

- regresion_logistica.py:

```
import numpy as np
import scipy.optimize as opt
import matplotlib.pyplot as plt

#-----

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

#-----

def cost(theta, X, Y):
    H = sigmoid(np.matmul(X, theta))
    cost = (- 1 / (len(X))) * (np.dot(Y, np.log(H)) + np.dot((1 - Y), np.log(1 - H)))
    return cost

#-----

def gradient(theta, XX, Y):
    H = sigmoid( np.matmul(XX,theta))
    grad =(1/len(Y))* np.matmul(XX.T, H - Y)
    return grad

#-----

def coste_reg(theta, X, Y, lambd):
    coste = cost(theta, X, Y) + lambd/(2*len(X)) * np.sum(theta**2)
    return coste

#-----

def gradiente_reg(theta, XX, Y, lambd):
    grad =gradient(theta, XX, Y) + lambd/(len(XX)) * theta
    return grad

#-----

def evaluacion(theta, X, Y):
    correctos = np.sum((sigmoid(np.dot(X, theta))>=0.5)==Y)
    return correctos

#-----
```

```

def regresion_logistica_reg(X, Y, Xval, Yval):

    m=np.shape(X)[0]
    n=np.shape(X)[1]

    X_ones = np.hstack([np.ones([m, 1]), X])

    mval = np.shape(Xval)[0]
    Xval_ones = np.hstack([np.ones([mval, 1]), Xval])

    max_correctos = 0
    max_lambda = 0

    plt.figure()
    correctArray = np.zeros(1000)

    for lambd in np.arange(0, 1000, 1):
        theta = np.full(n+1, 0)
        result = opt.fmin_tnc( func=coste_reg , x0=theta , fprime=gradiente_reg
, args =(X_ones,Y, lambd), messages= 0)
        theta_opt = result[0]
        correctos = evaluacion(theta_opt,Xval_ones,Yval)
        correctArray[lambd] = correctos
        if (correctos > max_correctos):
            max_correctos = correctos
            max_lambda = lambd

    plt.plot(np.linspace(1,len(correctArray),len(correctArray),
dtype=int),correctArray)
    plt.savefig("reg_logistica.png")

    return max_correctos/mval, max_lambda

#-----

```

- redes_neuronales.py:

```

import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
import scipy.optimize as opt

#-----

def load(X, y, XVal,Yval):

```



```

m = len(y)
input_size = X.shape[1]
num_labels = 2

y = (y-1)
y_onehot = np.zeros((m,num_labels))
for i in range(m):
    y_onehot[i][y[i].__int__()] = 1

n= len(Yval)
Yval = (Yval-1)
yval_onehot = np.zeros((n,num_labels))
for i in range(n):
    yval_onehot[i][Yval[i].__int__()] = 1

return X, y_onehot, yval_onehot

#-----

def loadRed():
    weights = loadmat('ex4weights.mat')
    theta1, theta2 = weights['Theta1'],weights['Theta2']

    return theta1, theta2

#-----

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

#-----

def coste_red(theta, X, Y):
    _, _, H = propagacion(X,theta[0],theta[1])
    cost = (-1 / (len(X))) * np.sum((Y * np.log(H)) + (1 - Y) * np.log(1 - H +
1e-9))
    return cost

#-----

def coste_red_reg(theta, X, Y, lambd):
    a = lambd/(2*(len(X))) * (np.sum(theta[0][1:]**2) + np.sum(theta[1][1:]**2))
    return coste_red(theta, X, Y) + a

#-----

def gradiente_red(theta, XX, Y, lambd):
    H = sigmoid( np.matmul(XX,theta))

```

```

grad =(1/len(Y))* np.matmul(XX.T, H - Y) + lambda/(len(XX)) * theta
return grad

#-----

def propagacion(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)
    H=sigmoid(z3)

    return a1, a2, H

#-----

def termino_reg(g, m, reg, theta):
    columna = g[0]
    g = g + (reg/m)*theta
    g[0] = columna
    return g

#-----

def backprop ( params_rn , num_entradas , num_ocultas , num_etiquetas , X, y ,
reg):

    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 )))

    A1, A2, H = propagacion(X, Theta1, Theta2)
    m = X.shape[0]

    Delta1 = np.zeros_like(Theta1)
    Delta2 = np.zeros_like(Theta2)

    for t in range(m):
        a1t = A1[t, :] # (401,)
        a2t = A2[t, :] # (26,)
        ht = H[t, :] # (10,)
        yt = y[t] # (10,)
        d3t = ht - yt # (10,)

```

```

        d2t = np.dot(Theta2.T, d3t) * (a2t * (1 - a2t)) # (26,)
        Delta1 = Delta1 + np.dot(d2t[1:, np.newaxis], alt[np.newaxis, :])
        Delta2 = Delta2 + np.dot(d3t[:, np.newaxis], a2t[np.newaxis, :])

G1 = Delta1/m
G2 = Delta2/m

G1 = termino_reg(G1, m, reg, Theta1)
G2 = termino_reg(G2, m, reg, Theta2)

    return coste_red_reg(np.array([Theta1, Theta2]), X, y, reg),
np.concatenate([np.ravel(G1), np.ravel(G2)])

#-----

def red_neuronal(X, y ,XVal, YVal):
    X, y, yval = load(X, y ,XVal, YVal)

    num_entradas = np.shape(X)[1]
    num_ocultas = 25
    num_etiquetas = 2

    ini = 0.12
    i=100

    plt.figure()
    lambdArray = np.zeros(10)

    max_correctos = 0
    max_lambda = 0

    for lambd in np.arange(0, 10, 1):
        pesos=
np.random.uniform(-ini,ini,(num_entradas+1)*num_ocultas+(num_ocultas+1)*num_etiq
uetas)
        sol = opt.minimize(fun=backprop, x0=pesos,
args=(num_entradas,num_ocultas, num_etiquetas,X, y ,lambd), jac= True,method =
'TNC', options ={'maxiter' :i})

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

        m = np.shape(X)[0]

```

```

        A1, A2, H = propagacion(XVal, theta1, theta2)

        maxChance = H.argmax(axis= 1)
        res = yval.argmax(axis= 1)
        correctos = np.sum(maxChance == res)
        lambdArray[lambd] = correctos
        if (correctos > max_correctos):
            max_correctos = correctos
            max_lambda = lambd

        plt.plot(np.linspace(1,len(lambdArray),len(lambdArray),
dtype=int),lambdArray)
        plt.savefig("redes_neuronales.png")

        return correctos/m, max_lambda

#-----

```

- SVM.py

```

import numpy as np
from sklearn.svm import SVC
import matplotlib.pyplot as plt

#-----

def calculaScore(svm, Xval, yval):
    yp = svm.predict(Xval).reshape(yval.shape)
    return sum(yp == yval)

#-----

def eleccionParams(X, y, Xval, yval):
    C_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    sigma_vec = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]
    scores = np.zeros((len(C_vec), len(sigma_vec)))

    plt.figure()
    for i in range(len(C_vec)):
        for j in range(len(sigma_vec)):
            svm = SVC(kernel= 'rbf' , C=C_vec[i], gamma = 1/(2*sigma_vec[j]**2))
            svm.fit(X, y)
            scores[i][j] = calculaScore(svm, Xval, yval)

    axes = plt.axes(projection= '3d')
    axes.contour(C_vec, sigma_vec, scores, cmap='rainbow',linewidth=0,
antialiased=False)

```

```

    axes.set_xlabel('C')
    axes.set_ylabel('sigma')
    axes.set_zlabel('n° aciertos')
    plt.savefig("svm.png")
    index = np.unravel_index(np.argmax(scores), scores.shape)

    svm = SVC(kernel= 'rbf' , C=C_vec[index[0]], gamma =
1/(2*sigma_vec[index[1]]**2))
    svm.fit(X, y)
    return svm

#-----

def svm_proyecto( X, y, Xval, yval):

    svm = eleccionParams(X, y, Xval, yval)
    return calculaScore(svm,Xval, yval)/np.shape(Xval)[0]

#-----

```

- proyecto.py:

```

from pandas.io.parsers import read_csv
from regresion_logistica import regresion_logistica_reg
from redes_neuronales import red_neuronal
from SVM import svm_proyecto
import numpy as np
import matplotlib.pyplot as plt

#-----

def carga_csv(file_name):

    valores = read_csv(file_name, header=0).to_numpy()
    return valores.astype(float)

#-----

def normalizar_mat(X):
    mu = np.mean(X, 0)
    sigma = np.std(X, 0)
    X_norm = (X-mu)/sigma
    X_norm[93] += X[93]
    return X_norm, mu, sigma

#-----

```

```

data = carga_csv('data.csv')
data = np.delete(data, 94, 1)
size = (data.shape[0]/2).__int__()

X = data[0:size, 1:]
y = data[0:size, 0]

Xval = data[size:, 1:]
yval = data[size:, 0]

XvalNor, _, _ = normalizar_mat(Xval)
XNor, _, _ = normalizar_mat(X)

_reg, _ = (regresion_logistica_reg(XNor, y, XvalNor, yval))
_red, _ = (red_neuronal(XNor, y, XvalNor, yval))
_svm = (svm_proyecto(XNor, y, XvalNor, yval))
print(_reg)
print(_red)
print(_svm)

xBars = ['Reg. Logistica', 'Red Neuronal', 'SVM']
ancho = 0.7
fig, aux = plt.subplots(figsize=(8,7))
index = np.arange(len(xBars))
plt.bar(index, [float(_reg), float(_red), float(_svm)], ancho, color =
'#7986CB')
plt.xticks(index, xBars, fontsize=9)
plt.ylim((0,1))
plt.savefig('Comparacion1.png')

xBars = ['Reg. Logistica', 'Red Neuronal', 'SVM']
ancho = 0.7
fig, aux = plt.subplots(figsize=(8,7))
index = np.arange(len(xBars))
plt.bar(index, [float(_reg), float(_red), float(_svm)], ancho, color =
'#7986CB')
plt.xticks(index, xBars, fontsize=9)
plt.ylim((0.8,1))
plt.savefig('Comparacion2.png')

xBars = ['Reg. Logistica', 'Red Neuronal', 'SVM']
ancho = 0.7
fig, aux = plt.subplots(figsize=(8,7))
index = np.arange(len(xBars))
plt.bar(index, [float(_reg), float(_red), float(_svm)], ancho, color =
'#7986CB')
plt.xticks(index, xBars, fontsize=9)

```

```
plt.ylim((0.925,1))  
plt.savefig('Comparacion3.png')
```