Practica 7: Detección de spam

Alberto Muñoz Fernández

Óscar García Castro

Parte A: Support Vector Machines

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.svm
import scipy.io as sc

def get_data(path):
    data = sc.loadmat(path, squeeze_me=True)
    X = data['X']
    y = data['y']

    return X, y

def draw(X, y, x1, x2, yp):
    plt.figure()
    positive = y == 1
    negative = y == 0
    plt.plot(X[positive, 0], X[positive, 1], 'k+')
    plt.plot(X[negative, 0], X[negative, 1], 'yo')
    plt.contour(x1, x2, yp)
    plt.show()

def kernel_lineal(X, y):
    svm = sklearn.svm.SVC(kernel='linear', C=1.0)
    svm.fit(X, y)

    x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    x1, x2 = np.meshgrid(x1, x2)
    yp = svm.predict(np.array([x1.ravel(),
           x2.ravel()]).T).reshape(x1.shape)

    draw(X, y, x1, x2, yp)

def kernell_gausiano(X, y, c = 1, sigma = 0.1):
    svm = sklearn.svm.SVC(kernel='rbf', C = c, gamma=1 / (2 * sigma**2))
    svm.fit(X, y)

    x1 = np.linspace(X[:, 0].min(), X[:, 0].max(), 100)
    x2 = np.linspace(X[:, 1].min(), X[:, 1].max(), 100)
    x1, x2 = np.meshgrid(x1, x2)
```

```
    #da error aqui (X has 2 features, but SVC is expecting 1899 features
as input)
    yp = svm.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape)

    draw(X, y, x1, x2, yp)
    return svm

def elec_params(X, y, Xval, yval, c, sigma):
    svm = sklearn.svm.SVC(kernel='rbf', C = c, gamma=1 / (2 * sigma**2))
    svm.fit(X, y)
    yp = svm.predict(Xval)

    cont = 0
    for i in range (len(yval)):
        if (yval[i] == yp[i]):
            cont += 1
    return cont/len(yval) * 100

def bestCandSigma():
    data = sc.loadmat('data/ex6data3.mat', squeeze_me=True)
    X = data['X']
    y = data['y']
    Xval = data['Xval']
    yval = data['yval']

    vals = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

    acierto = -1
    bestC = -1
    bestSigma = -1
    for c in vals:
        for sig in vals:
            aux = elec_params(X, y, Xval, yval, c, sig)
            if (acierto == -1 or aux > acierto):
                acierto = aux
                bestSigma = sig
                bestC = c

    kernell_gausiano(X, y, bestC, bestSigma)

def calcSVM(X, y, Xval, yval):
    vals = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

    acierto = -1
    bestC = -1
    bestSigma = -1
    for c in vals:
        for sig in vals:
```

```
            aux = elec_params(X, y, Xval, yval, c, sig)
            if (acierto == -1 or aux > acierto):
                acierto = aux
                bestSigma = sig
                bestC = c
    svm = sklearn.svm.SVC(kernel='rbf', C = bestC, gamma=1 / (2 *
         bestSigma**2))
    svm.fit(X, y)
    return svm
```

Parte B: Detección de spam

```
import codecs
import glob
from matplotlib import pyplot as plt
import numpy as np
import utils
import svm
import sklearn.model_selection as sms
import logistic_reg as lr
import nn
import time

#Lectura de casos de ejemplo de la carpeta path
def readMail(path, dicc):
    vec = np.zeros(len(dicc))
    email_contents = codecs.open(path, 'r', encoding='utf-8',
errors='ignore').read()
    email = utils.email2TokenList(email_contents)
    for i in range (len(email)):
        try:
            num = dicc[email[i]]
            vec[num] = 1
        except:
            continue
    return vec

#Convierte los mails a la estructura necesaria
#X es de tamaño NxM, siendo N nº de casos y M tamaño del diccionario
#Y es de tamaño Nx1, indicando con 0s y 1s si es o no spam
def readFolder(dicc, folderPath, isSpam):
    docs = glob.glob(folderPath)
    X = np.zeros((len(docs), len(dicc)))
    y = np.full((len(docs)), isSpam)
    for i in range (len(docs)):
        X[i] = readMail(docs[i], dicc)

    return X, y
```

```python
#Support Vector Machines
#Cálculo de la fiabilidad del modelo SVM
#División de casos de prueba:
#60% train, 20% test, 20% val
def withSVM(X, y):
    x_train, x_test, y_train, y_test = sms.train_test_split(X, y,
test_size = 0.2, random_state = 1)
    x_train, x_val, y_train, y_val = sms.train_test_split(x_train,
y_train, test_size = 0.25, random_state = 1)

    #Esta llamada devuelve el sistema entrenado
    funct = svm.calcSVM(x_train, y_train, x_val, y_val)

    #Calculo de la predicción
    yp = funct.predict(x_test)
    cont = 0
    for i in range (len(y_test)):
        if (y_test[i] == yp[i]):
            cont += 1
    return cont/len(y_test) * 100
```

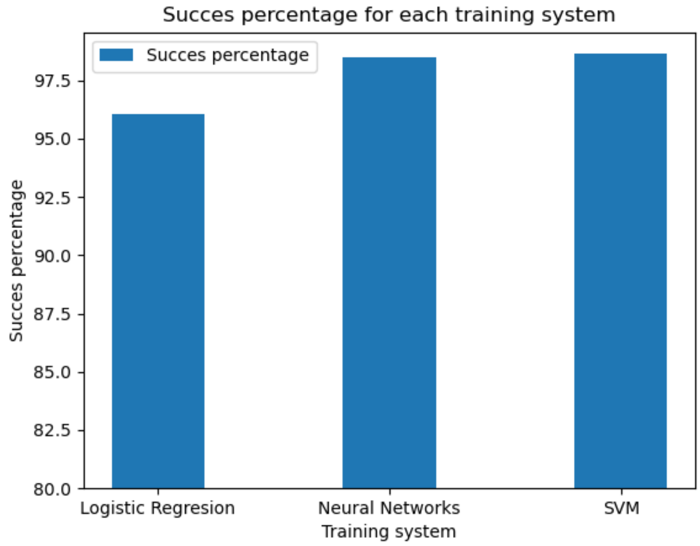
```python
#Logistic Regresion
#Cálculo de la fiabilidad del modelo LogisticReg
#División de casos de prueba:
#60% train, 20% test, 20% val
#Además de calcular el coste, buscamos la mejor lambda con los ejemplos
de validación
def withLogisticRegresion(X, y):
    x_train, x_test, y_train, y_test = sms.train_test_split(X, y,
test_size = 0.2, random_state = 1)
    x_train, x_val, y_train, y_val = sms.train_test_split(x_train,
y_train, test_size = 0.25, random_state = 1)
    #Array posibles lambdas
    lambd = [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100,
300, 600, 900]

    bestLamb = -1
    acierto = -1
    bestW = np.zeros(len(x_train[0]))
    bestB = 0

    for l in lambd:
        aciertoAux, wAux, bAux = lr.calcBest(x_train, y_train, x_val,
y_val, l, 1000)

        if (bestLamb == -1 or acierto < aciertoAux):
            bestLamb = l
            acierto = aciertoAux
```

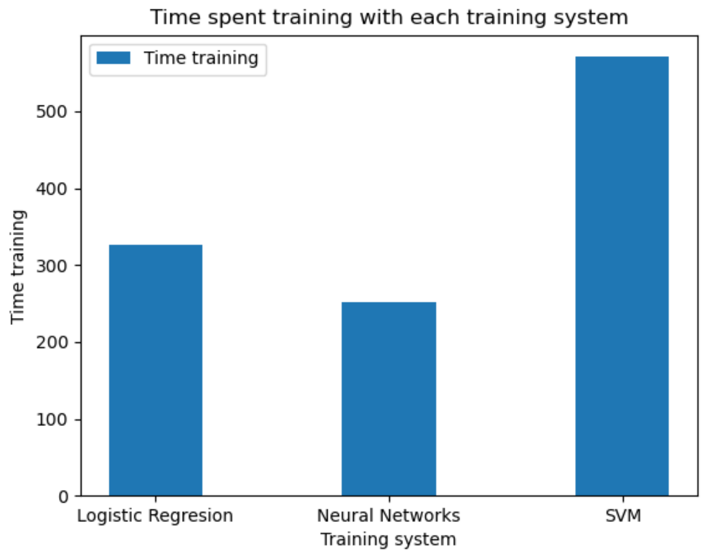
```
            bestW = wAux
            bestB = bAux

    return lr.test(x_test, y_test, bestW, bestB)
```

```
#Neuronal Network
#Cálculo de la fiabilidad del modelo NN
#División de casos de prueba:
#60% train, 20% test, 20% val
def withNN(X, y):
    x_train, x_test, y_train, y_test = sms.train_test_split(X, y,
test_size = 0.2, random_state = 1)
    x_train, x_val, y_train, y_val = sms.train_test_split(x_train,
y_train, test_size = 0.25, random_state = 1)

    #Codificamos y con el método "one-hot" para diferenciar entre spam/no
spam
    y_hot = np.zeros([len(y_train), 2])
    for i in range(len(y_train)):
        y_hot[i][y_train[i]] = 1
    #Posibles lambdas
    lambd = [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100,
300, 600, 900]

    #Thetas random para iniciar
    theta1 = np.random.random((25, len(x_train[0]) + 1)) * (2*0.12)  -
0.12
    theta2 = np.random.random((2, 26))  * (2*0.12)  - 0.12
    arr = np.concatenate([theta1.ravel(), theta2.ravel()])

    bestLamb = -1
    acierto = -1

    for l in lambd:
        aciertoAux, theta1Aux, theta2Aux = nn.calcBest(x_train, y_hot,
x_val, y_val, arr, l, 100)
        if (bestLamb == -1 or aciertoAux > acierto):
            bestLamb = l
            acierto = aciertoAux
            theta1 = theta1Aux
            theta2 = theta2Aux

    return nn.test(x_test, y_test, theta1, theta2)
```

```python
def main():
    #Lectura del diccionario y de todos los ejemplos
    dicc = utils.getVocabDict()
    XSpam, ySpam = readFolder(dicc, 'data_spam/spam/*.txt', 1)
    XEasy, yEasy = readFolder(dicc, 'data_spam/easy_ham/*.txt', 0)
    XHard, yHard = readFolder(dicc, 'data_spam/hard_ham/*.txt', 0)

    #Almacenamos todos los ejemplos en X e y
    X = np.concatenate((XSpam, XEasy, XHard), axis=0)
    y = np.concatenate((ySpam, yEasy, yHard), axis=0)

    inicio = time.time()
    #NN
    aciertoNN = withNN(X, y)
    timeNN = time.time() - inicio
    #LogisticReg
    aciertoLogReg = withLogisticRegresion(X, y)
    timeLogReg = time.time() - inicio
    #SVM
    aciertoSVM = withSVM(X,y)
    timeSVM = time.time() - inicio

    plotOffset = 80

    Xplot = ['Logistic Regresion','Neural Networks','SVM']
    yplot = [aciertoLogReg - plotOffset,aciertoNN - plotOffset,
aciertoSVM - plotOffset]
    yplotTime = [timeLogReg ,timeNN, timeSVM ]

    X_axis = np.arange(len(Xplot))

    plt.bar(X_axis, yplot, 0.4, label = 'Succes percentage',
bottom=plotOffset)
    plt.xticks(X_axis, Xplot)
    plt.xlabel("Training system")
    plt.ylabel("Succes percentage")
    plt.title("Succes percentage for each training system")
    plt.legend()
    plt.show()
    plt.close("all")

    plt.bar(X_axis, yplotTime, 0.4, label = 'Time training')
    plt.xticks(X_axis, Xplot)
    plt.xlabel("Training system")
    plt.ylabel("Time training")
    plt.title("Time spent training with each training system")
    plt.legend()
    plt.show()
```

Comparativa resultados





Podemos observar que, tanto NN como SVM tienen un porcentaje de acierto superior al 97.5%, la comparativa de tiempo deja claro que SVM es mucho peor en términos de eficiencia, y NN la más rápida de las 3 opciones. LogisticReg sin embargo, tiene peores resultados, aunque aún así aceptables, y un tiempo de ejecución no demasiado malo en comparación. El punto negativo de este sistema de entrenamiento es la cantidad de itreraciones que necesita para ajustarse a los datos, ya que en comparación con NN, son 10 veces más para obtener resultados similares, haciéndolo así más lento.