Aprendizaje automático y minería de datos

Práctica 7: Detección de spam

Juan Diego Mendoza Reyes

```python
#Lectura de los mail proporcioandos por un path comun
def open_mails(path, Spam):
    #Diccionario
    dicc = utils.getVocabDict()
    docs = glob.glob(path)

    #Inicializamos los datos
    x = np.zeros((len(docs), len(dicc)))
    y = np.full((len(docs)), Spam)

    #Recorremos todos los emails
    for i in range(len(docs)):
        v = np.zeros(len(dicc))
        email_contents = codecs.open(docs[i], 'r', encoding='utf-8', errors='ignore').read()
        email = utils.email2TokenList(email_contents)
        #Obtenemos el vector de las palabras coincidentes con el diccionario
        for j in range(len(email)):
            try:
                indexDicc = dicc[email[j]]
                v[indexDicc] = 1
            except:
                continue

        x[i] = v

    return x,y
```

```python
#-------------------------------------------------------------
#Cálculo de la fiabilidad con Regresion Logistica
#División de los datos :60% train, 20% test, 20% val
#Además de calcular el coste, aplicamos la mejor lambda con los ejemplos de validación
def proccessWithLogReg(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)

    #posibles lambdas de la practica 6
    lambdas = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 300, 600, 900]

    best_lambda = -1
    success = -1
    best_w = np.zeros(len(x_train[0]))
    best_b = 0

    for l in lambdas:
        s, w, b = lgr.train(x_train, y_train, x_val, y_val, l, 1000)    #Mas porcentaje de exito cuantas mas iteraciones

        if(best_lambda == -1 or success< s):
            best_lambda = l
            success = s
            best_w = w
            best_b = b

    testing = lgr.test(x_test, y_test, best_w, best_b)
    print("Log_reg :", testing)

    return testing
```

```python
#-----------------------------------------------------------------
#Cálculo de la fiabilidad con Redes Neuronales
#División de los datos :60% train, 20% test, 20% val
def proccessWithNN(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)

    #lambdas de la practica 6
    lambdas = [1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 300, 600, 900]

    #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

    #thetas inicializadas random según el enunciado de la practica 5
    EPSILON = 0.12
    theta1 = np.random.uniform(-EPSILON, EPSILON, (25, len(x_train[0]) + 1))
    theta2 = np.random.uniform(-EPSILON, EPSILON, (2, 26))

    thetas = np.concatenate([theta1.ravel(), theta2.ravel()])

    bestTheta1 = []
    bestTheta2 = []
    success = -1

    for l in lambdas:
        s, th1, th2 = nn.train(x_train, y_hot, x_val, y_val, thetas, l, 1000)    #Mas porcentaje de exito cuantas mas iteraciones

        if(success< s):
            best_lambda = l
            success = s
            bestTheta1 = th1
            bestTheta2 = th2


    testing = nn.test(x_test, y_test, bestTheta1, bestTheta2)
    print("NN :", testing)

    return testing
```

```python
#-----------------------------------------------------------------
#Cálculo de la fiabilidad con Redes Neuronales
#División de los datos :60% train, 20% test, 20% val
def proccessWithSVM(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)

    #Entrenamiento de los datos
    train = svm.train(x_train, y_train, x_val, y_val)

    #Prediccion de los datos entrenados
    testing = svm.test(x_test, y_test, train)
    print("SVM :", testing)

    return testing
```

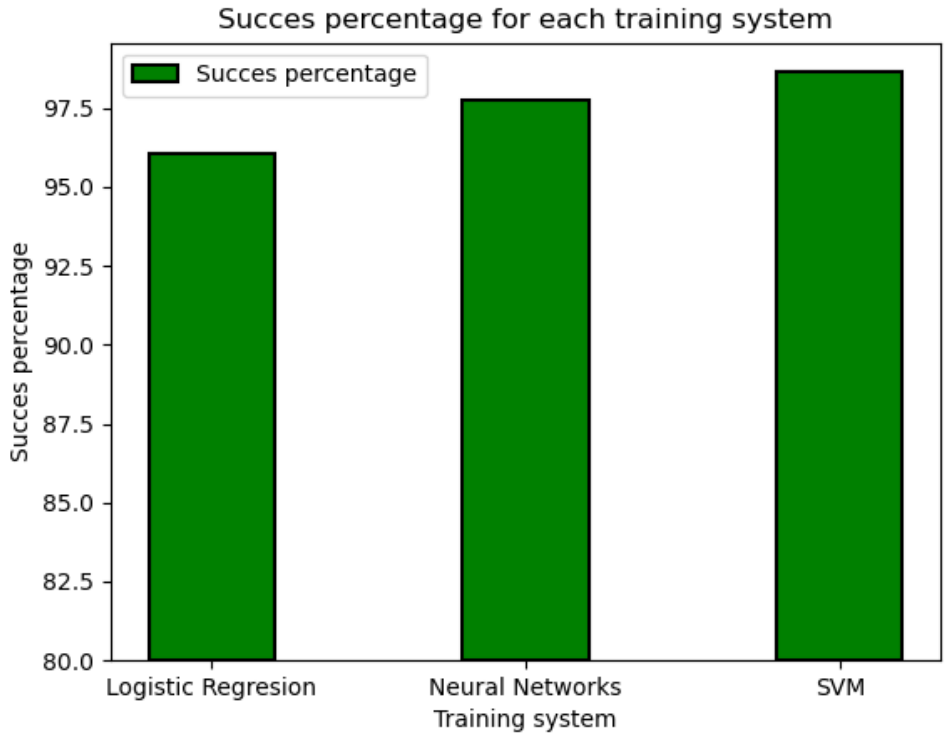
```python
def main():
    X_spam, y_spam = open_mails('data_spam/spam/*.txt', 1)
    X_easy, y_easy = open_mails('data_spam/easy_ham/*.txt', 0)
    X_hard, y_hard = open_mails('data_spam/hard_ham/*.txt', 0)

    #Almacenamos todos los ejemplos en X e y
    x = np.concatenate((X_spam, X_easy, X_hard), axis=0)
    y = np.concatenate((y_spam, y_easy, y_hard), axis=0)


    inicio = time.time()
    #LogisticReg
    aciertoLogReg = proccessWithLogReg(x, y)
    timeLogReg = time.time() - inicio
    #NN
    aciertoNN = proccessWithNN(x, y)
    timeNN = time.time() - inicio
    #SVM
    aciertoSVM = proccessWithSVM(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, color='red',  linewidth=1.5)
    plt.xticks(X_axis, Xplot)
    plt.xlabel("Training system")
    plt.ylabel("Success percentage")
    plt.title("Success percentage for each training system")
    plt.legend()
    plt.show()
    plt.close("all")

    plt.bar(X_axis, yplotTime, 0.4, label = 'Time training', color='green',  linewidth=1.5)
    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()
```

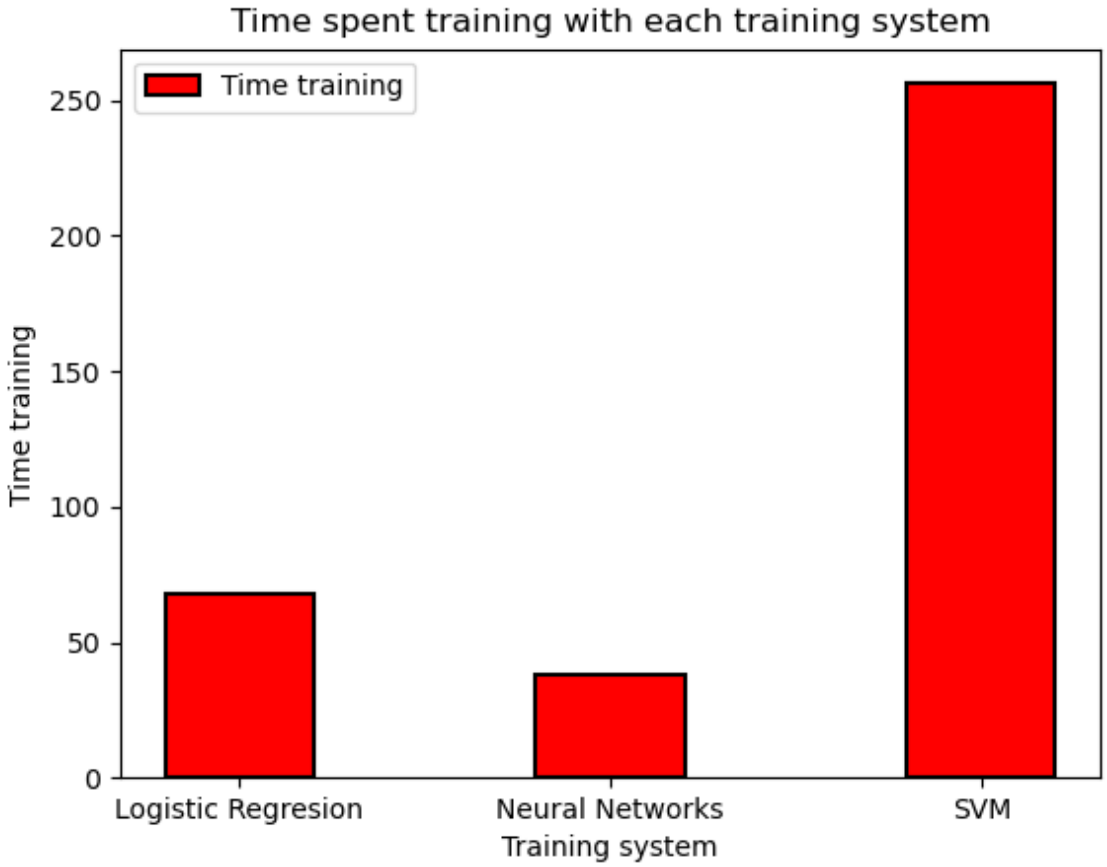
Resultados:

```
Log_reg : 96.06656580937972
Tiempo Regresion Logistica:  67.51099991798401
NN : 98.18456883509833
Tiempo red neuronal:  38.090999603271484
SVM : 98.63842662632375
Tiempo SVM:  256.27700090408325
```

En las gráficas y resultados obtenidos, se puede observar que, tanto NN como SVM tienen un porcentaje de acierto superior al 97%, a diferencia de la Regresión Logística que no llega a ese umbral.
En la comparativa de tiempos se llega a apreciar como SVM decae mucho en términos de eficiencia, y NN se corona como la más rápida de las 3 opciones.

Sin embargo, la Regresión Logística, tiene peores resultados a menor número de iteraciones, aunque aceptables. Por lo que el mayor punto en contra de este sistema de entrenamiento es el número de iteraciones necesarias para tener un ajuste a los datos óptimo, ya que se requieren de 50 veces el número de iteraciones en comparación con NN, pero lo compensa con un tiempo de entrenamiento necesario competente.




Time spent training with each training system
Time training
250
200
150
100
50
0
Time training
Logistic Regresion
Neural Networks
SVM
Training system