Memoria P7.2. Detección Spam

Aarón Nauzet Moreno Sosa (aarmor01@ucm.es)
Tomás López Antón (tomalope@ucm.es)

Procesamiento de datos

Todas las funciones de procesado de datos (emails) están localizados dentro del archivo ***utils.py***.

Se obtiene los datos crudos mediante la función **obtainAllDirectoryFiles**, que adquiere todos los archivos dentro de un directorio.

```
def obtainAllDirectoryFiles(dir):
    return [os.path.join(dir, dir_file) for dir_file in os.listdir(dir) if
os.path.isfile(os.path.join(dir, dir_file))]
```

Estos correos son recibidos por el método **transcribe_emails**, que se encarga de procesarlo para su uso en el entrenamiento, y asignarles un resultado a cada uno [en este caso, 0 si no son spam y 1 si lo son].

En primer lugar, se genera una lista de correos procesados, donde cada elemento se corresponde con cada palabra recogida en el diccionario de vocabulario; y se rellenan con 0s (indicando esto que no está presenta la palabra en el correo).

Tras ello, se recorre cada correo crudo, convirtiéndolo a una lista de *tokens* para facilitar su lectura, utilizando para ello el método **email2TokenList**. Una vez los emails han sido convertidos, se recorre cada *token* (palabra) dentro de la lista, buscándola dentro del diccionario de vocabulario, y, si está presente, se marca como 1.

Por último, se devuelve la lista de correos transcritos, junto con la lista de resultados asignados.

```
def transcribe_emails(emails, default_value=0):
    vocab_list = getVocabDict()
    transcribed_emails = np.zeros((len(emails), len(vocab_list)))

    for email in range(len(emails)):
        # ignore non utf-8 characters
        email_contents = open(emails[email], 'r', encoding='utf-8',
errors='ignore').read()
        curr_email = email2TokenList(email_contents)
        # sys.exit("Error message")
```

```
for word in curr_email:
            if word in vocab_list:
                word_id = vocab_list[word] - 1
                transcribed_emails[email, word_id] = 1

    return transcribed_emails, np.full(transcribed_emails.shape[0],
default_value)
```

La función **preprocess_data** aplica la función **transcribe_emails** sobre los datos aportados en easy_ham (correos no spam fáciles de detectar), hard_ham (correos no spam difíciles de detectar) y spam (correos de spam); asignando 0 o 1 a sus resultados según son o no spam.

Posteriormente, se divide cada uno de estos datos en conjunto de **entrenamiento**, **validación** y **test** (60%, 20% y 20% para cada paquete de datos) y, finalmente, se combinan cada conjunto de cada paquete de datos específico en un conjunto general homónimo (**entrenamiento**, **validación** y **test**), para aplicar en cada sistema.

```
def preprocess_data():
    # adapt emails to what we need for our training (words -> list(0,1))
    easy_ham_x, easy_ham_y =
transcribe_emails(obtainAllDirectoryFiles('./data_spam/easy_ham/'))
    hard_ham_x, hard_ham_y =
transcribe_emails(obtainAllDirectoryFiles('./data_spam/hard_ham/'))
    spam_x, spam_y =
transcribe_emails(obtainAllDirectoryFiles('./data_spam/spam/'), 1)

    # 60% train, 20% validation, 20% test for each type
    eh_x_train, eh_x_temp, eh_y_train, eh_y_temp =
skm.train_test_split(easy_ham_x, easy_ham_y, test_size=0.4, random_state=1)
    eh_x_test, eh_x_val, eh_y_test, eh_y_val = skm.train_test_split(eh_x_temp,
eh_y_temp, test_size=0.5, random_state=1)

    hh_x_train, hh_x_temp, hh_y_train, hh_y_temp =
skm.train_test_split(hard_ham_x, hard_ham_y, test_size=0.4, random_state=1)
    hh_x_test, hh_x_val, hh_y_test, hh_y_val = skm.train_test_split(hh_x_temp,
hh_y_temp, test_size=0.5, random_state=1)

    s_x_train, s_x_temp, s_y_train, s_y_temp = skm.train_test_split(spam_x,
spam_y, test_size=0.4, random_state=1)
    s_x_test, s_x_val, s_y_test, s_y_val = skm.train_test_split(s_x_temp,
s_y_temp, test_size=0.5, random_state=1)
```

```
    # we mix the three data sets for each step of the process
    x_train, y_train = np.vstack((eh_x_train, hh_x_train, s_x_train)),
np.hstack((eh_y_train, hh_y_train, s_y_train))
    x_test, y_test = np.vstack((eh_x_test, hh_x_test, s_x_test)),
np.hstack((eh_y_test, hh_y_test, s_y_test))
    x_val, y_val = np.vstack((eh_x_val, hh_x_val, s_x_val)),
np.hstack((eh_y_val, hh_y_val, s_y_val))

    return x_train, y_train, x_val, y_val, x_test, y_test
```

Entrenamiento de modelos

Con carácter general, para cada sistema de entrenamiento se ha seguido la misma estructura que en la obtención de hiper parámetros de la prácticas, de tal forma que:

1. Se recorren todos los valores posibles de los parámetros a obtener (**log_reg** → lambda; **nn** → lambda; **svm** → c & sigma).
2. Entrenamos cada sistema con cada valor, guardando los resultados obtenidos de cada sistema (**log_reg** → w & b; **nn** → theta1 & theta2; **svm** → svm).
3. Se obtiene el porcentaje de acierto de cada valor utilizando el conjunto de validación; y se guarda cada uno en una lista.
4. Tras recorrer todos los valores, se obtiene el resultado obtenido por aquel valor con el mayor porcentaje de acierto.
5. Por último, se prueba el porcentaje de acierto del sistema seleccionado con el conjunto de datos de test, para certificar su veracidad.

Cabe destacar que, para cada sistema de entrenamiento, se ha reutilizado código de aquellas prácticas en las que se programaron originalmente (**log_reg** → p3; **nn** → p5; **svm** → p7.1).

Entrenamiento regresión logística

- **Parámetros a seleccionar:** lambda.
- **Valores devueltos:** w & b.

```
def check_spam_reg_log(x_train, y_train, x_val, y_val, x_test, y_test):
    tic = time.process_time()
    w, b = obtain_reg_log(x_train, y_train, x_val, y_val)
    predict = lgr.predict(x_test, w, b)
    accuracy = np.sum(y_test == predict) / y_test.shape[0] * 100
    toc = time.process_time()
    process_time = toc - tic

    return accuracy, process_time
```

```
def obtain_reg_log(x_train, y_train, x_val, y_val):
    # initial values
    values = np.array([1e-6, 1e-5, 1e-4, 0.001, 0.01, 0.1, 1, 10, 50, 100,
500])
    w_in, b_in = np.zeros(x_train.shape[1]), 1
    num_iters = 2000
    alpha = 0.01

    # check for best parameters
    iteration = 0
    accuracies, lambda_list = [], []
    for lambda_value in values:
        new_w, new_b, _ = lgr.gradient_descent(x_train, y_train, w_in, b_in,
lgr.compute_cost_reg, lgr.compute_gradient_reg, alpha, num_iters,
lambda_value)

        predict = lgr.predict(x_val, new_w, new_b)
        accuracy = np.sum(y_val == predict) / y_val.shape[0]

        accuracies.append(accuracy)
        lambda_list.append((lambda_value, new_w, new_b))
        iteration += 1

    best_accuracy = np.argmax(accuracies)
    lambda_, w, b = lambda_list[best_accuracy]

    return w, b
```

Entrenamiento red neuronal

- **Parámetros a seleccionar:** lambda.
- **Valores devueltos:** theta1 & theta2.

```
def check_spam_neural_network(x_train, y_train, x_val, y_val, x_test, y_test):
    tic = time.process_time()
    theta1, theta2 = obtain_neural_network(x_train, y_train, x_val, y_val)
    p = nn.predict(theta1, theta2, x_test)
    accuracy = np.sum(y_test == p) / y_test.shape[0] * 100
    toc = time.process_time()
    process_time = toc - tic

    return accuracy, process_time
```

```
def obtain_neural_network(x_train, y_train, x_val, y_val):
    # initial values
    # values = np.array([1e-6, 1e-5, 1e-4, 0.001, 0.01, 0.1, 1, 10, 50, 100,
500])
    values = [1e-5, 1e-4, 0.001, 0.01, 0.1, 1, 10, 50]

    output_labels = 2               # spam or not spam
    input_labels = len(x_train[0])  # nº words email
    hidden_labels = 25              # because I can

    theta1, theta2 = nn.random_thetas(input_labels, hidden_labels,
output_labels)
    y_onehot = nn.onehot(y_train, output_labels)
    num_iters = 200

    # check for best parameters
    iteration = 0
    accuracies, nn_list = [], []
    for lambda_value in values:
        theta_1, theta_2 = nn.minimize(x_train, y_onehot, theta1, theta2,
num_iters, lambda_value)

        predict = nn.predict(theta_1, theta_2, x_val)
        accuracy = np.sum(y_val == predict) / y_val.shape[0]

        accuracies.append(accuracy)
        nn_list.append((lambda_value, theta_1, theta_2))
        iteration += 1

    best_accuracy = np.argmax(accuracies)
    lambda_, theta1, theta2 = nn_list[best_accuracy]

    return theta1, theta2
```

Entrenamiento SVM

- **Parámetros a seleccionar:** c & sigma.
- **Valores devueltos:** svm_.

```
def check_spam_svm(x_train, y_train, x_val, y_val, x_test, y_test):
    tic = time.process_time()
    svm = obtain_svm(x_train, y_train, x_val, y_val)
    accuracy = skme.accuracy_score(y_test, svm.predict(x_test)) * 100
    toc = time.process_time()
    process_time = toc - tic

    return accuracy, process_time
```

```
def obtain_svm(x_train, y_train, x_val, y_val):
    # initial values
    values = [0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30]

    # check for best parameters
    iteration = 0
    accuracies, svm_list = [], []
    for c_value in values:
        for sigma_value in values:
            new_svm, _, accuracy = svm.svm(x_train, y_train, x_val, y_val,
'rbf', c_value, sigma_value)

            accuracies.append(accuracy)
            svm_list.append((c_value, sigma_value, new_svm))
            iteration += 1

    best_accuracy = np.argmax(accuracies)
    c, sigma, svm_ = svm_list[best_accuracy]

    return svm_
```

Resultados

Tras correr todas las pruebas, se obtienen los siguientes resultados:

```
Logistic regression best parameters: Lambda --> 10.0
Logistic regression accuracy: 96.52%%
Logistic regression process time: 598.91 seconds

Neural network best parameters: Lambda --> 0.001
Neural network accuracy: 97.12%%
Neural network process time: 2377.78 seconds

SVM best parameters: C --> 10; Sigma --> 10
SVM accuracy: 98.03%%
SVM process time: 444.64 seconds
```