Memoria P4. Clasificación Multiclase & Redes Neuronales

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

Parte A

Clasificación uno contra todos

La función **oneVsAll** entrena varios clasificadores de regresión logística. Recibe los siguientes parámetros:

- **X**: conjunto de datos de tamaño **m***n, siendo **m** el número de puntos de datos y **n** el número de características.
- **y**: conjunto de etiquetas resultado.
- **n_labels**: número de etiquetas posibles.
- **lambda_**: constante de regularización.

Comenzamos definiendo una variable **n** del tamaño de la segunda dimensión del conjunto **X**, que corresponde al número de características. Inicializamos las variables **w_in** y **b_in**, la tasa de aprendizaje **alpha** y el número de iteraciones **n_iters**. También definimos el tamaño del conjunto de parámetros entrenados **all_theta**.

Para entrenar cada clasificador, se itera sobre cada una de las **n_labels** posibles etiquetas. Se crea una nueva variable **classif,** que es True para todas las filas de **y** que tienen el valor de la etiqueta actual y False para el resto.

Luego se llama a la función **gradient_descent** para entrenar un clasificador de regresión logística para esta clase específica. Los parámetros entrenados se guardan en **all_theta**.

```
def oneVsAll(X, y, n_labels, lambda_):
    m = X.shape[0]
    n = X.shape[1]

    w_in = np.zeros(n)
    b_in = 0
    alpha = 1
    n_iters = 1500

    all_theta = np.zeros([n_labels, n + 1])
```

```
for i in range(n_labels):
        classif = (y == i)
        w, b, _ = lgr.gradient_descent(X, classif, w_in, b_in,
lgr.compute_cost_reg, lgr.compute_gradient_reg,  alpha, n_iters, lambda_)
        all_theta[i] = np.append(b, w)

    return all_theta
```

Predicción uno contra todos

La función **predictOneVsAll** devuelve un vector de predicciones por cada ejemplo del conjunto **X**. Recibe los siguientes parámetros:

- **all_theta**: conjunto de parámetros de regresión logística entrenados.
- **X**: conjunto de ejemplos.

La función comienza definiendo una variable **m** del tamaño del conjunto **X** y un array de predicciones de tamaño **m**, en el que todos los valores son inicializados a 0.

Para predecir la etiqueta de cada ejemplo, iteramos en el conjunto **X**, extrayendo las probabilidades de cada etiqueta utilizando el sigmoide y asignando como predicción a la etiqueta con mayor probabilidad. Por último la función devuelve el conjunto de predicciones.

```
def predictOneVsAll(all_theta, X):
    m = X.shape[0]

    predict = np.zeros(m)
    for i in range(m):
        predict[i] = np.argmax(lgr.sigmoid(np.append(1, X[i]) @ all_theta.T))

    return predict
```

Parte B

Propagación prealimentada y predicción

La función **predict** devuelve un vector de predicciones por cada ejemplo del conjunto **X** utilizando una red neuronal prealimentada (**feedforward**). Recibe los siguientes parámetros:

- **theta1**: conjunto de pesos entre la primera y segunda capa de la red.
- **theta2**: conjunto de pesos entre la segunda y tercera capa de la red.
- **X**: conjunto de ejemplos.

La función comienza definiendo una variable **m** del tamaño del conjunto **X**. A continuación se asignan los valores de la capa de entrada **a1** como los valores del conjunto **X**. La capa intermedia **a2** se calcula haciendo el sigmoide de los valores de la capa de entrada multiplicados por los pesos del conjunto **theta1**.

A continuación se repite el proceso para la capa de salida **a3**, haciendo el sigmoide de los valores de la capa intermedia multiplicados por los pesos del conjunto **theta2**. Este proceso se conoce como **feedforward propagation** o propagación prealimentada.

Por último se calcula la predicción como el mayor de los valores de probabilidad de la capa de salida **a3**.

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

    # first layer (input)
    a1 = np.column_stack([np.ones((m, 1)), X])
    z2 = a1 @ theta1.T

    # second layer (hidden)
    a2 = lgr.sigmoid(z2)
    a2 = np.column_stack([np.ones((m, 1)), a2])
    z3 = a2 @ theta2.T

    # third layer (output)
    a3 = lgr.sigmoid(z3)
    predict = np.argmax(a3, axis=1)

    return predict
```

Resultado

Tras cargar los datos y hacer el cómputo y predicción del oneVsAll; y la predicción de la red neuronal pre-entrenada, obtenemos los siguientes resultados:

Parte A

```
Accuracy oneVsAll: 93.54
```

Parte B

```
Accuracy Neural Network: 97.52
```