

Entrega 4: clasificación multi-clase

Aprendizaje Automatico y Big Data- Alejandro Barrachina Argudo

Introducción

En este documento se explicará el código del entregable 4 y el proceso del clasificador multi-clase. Esta práctica se divide en 2 apartados: clasificador multi-clase y redes neuronales. El *dataset* consiste de imágenes de números escritos a mano como los vistos en la figura 0.3

Para esta práctica se usarán los siguientes *imports* vistos en la figura 0.1. Parte del código se reutiliza de la práctica anterior. A los *strings* estáticos anteriores añadimos uno para guardar el modelo entrenado (figura 0.2)

```
1 import csv
2 import numpy as np
3 import scipy.io as sio
4 import concurrent.futures
5 import matplotlib.pyplot as plt
6 import os
7 import logistic_reg
```

Figura 0.1: Código de las bibliotecas usadas

```
1 model_folder = "./models/"
```

Figura 0.2: Código de los nuevos strings estáticos



Figura 0.3: Ejemplo de los dígitos del *dataset*

1. Parte A: clasificación multi-clase

Para esta sección vamos a reutiliza todo lo referente a la regresión logística de la práctica anterior. Los cambios implementados son la distinción entre clases distintas y la concurrencia para acortar los tiempos de entrenamiento.

Para la concurrencia usaremos 3 funciones distintas: *train_model*, *train_all* y *oneVsAll* (figuras 1.4, 1.5 y 1.6 respectivamente). La función *train_model* es la misma que la de la práctica anterior, pero ahora se le pasa un parámetro extra, *class_num*, que indica la clase que se está entrenando. La función *train_all* se encarga de generar los hilos para entrenar cada etiqueta y juntar todos los resultados. La función *train_model* se encarga de generar los datos de entrenamiento (α , λ y el número de iteraciones.) La función *oneVsAll* hace de wrapper para las dos funciones anteriores. Finalmente, la función *run_one_vs_all* (figure 1.3) se encarga de hacer de driver para esta sección.

Tras esto, utilizamos la función *predictOneVsAll* (figura 1.7) para predecir la clase de un conjunto de datos. Esta función es muy similar a la de la práctica anterior, pero ahora diferencia entre los distintos tipos de clases dadas según el *dataset*. La función *run_one_vs_all* será la función principal de este apartado. Cargaremos los datos del *dataset* desde aquí y daremos valores al número de clases y a λ , así como mostraremos los distintos gráficos y predicciones.

Para la elaboración de los gráficos usaremos las funciones *plot_confusion_matrix* (figura 1.8) para hacer la matriz de confusión (figura 1.1) y *print_predictions* (figura 1.9) para mostrar las predicciones a lo largo del entrenamiento (figura 1.2).

Tras una sesión de entrenamiento con $\alpha = 0,2$, $\lambda = 0,01$ y 2500 iteraciones, obtenemos una predicción del 94.7%, con estos gráficos (matriz de confusión 1.1 y predicciones 1.2) para ver mejor los resultados. Con la función *save_model* (figura 1.10) guardamos el modelo para futuras predicciones y para evitar nuevos entrenamientos. Las funciones *save_predict* y *save_last_predictions* (figura 1.11 y 1.12) guardan las predicciones en distintos tipos de archivo para poder usarlos en otras ocasiones para hacer gráficos.

Las últimas predicciones de cada label son:

0	1	2	3	4	5	6	7	8	9
0.981	0.9976	0.9878	0.9894	0.9818	0.9932	0.9762	0.9694	0.995	0.9816

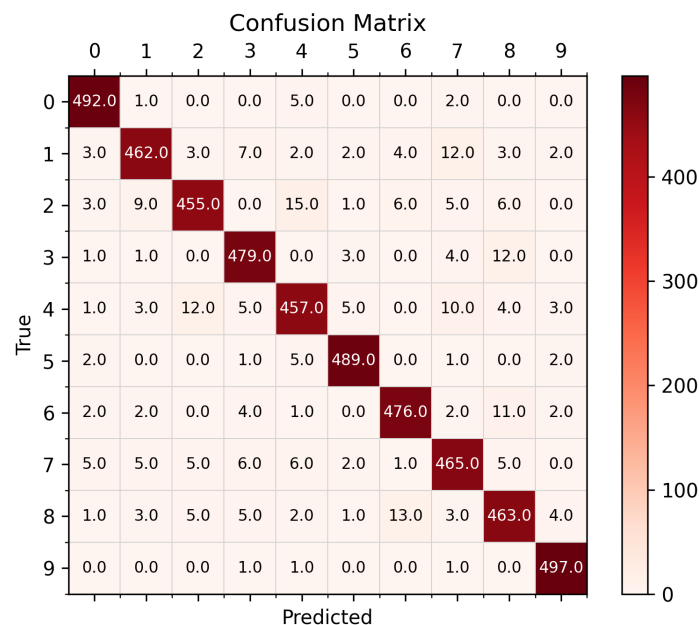


Figura 1.1: Matriz de confusión

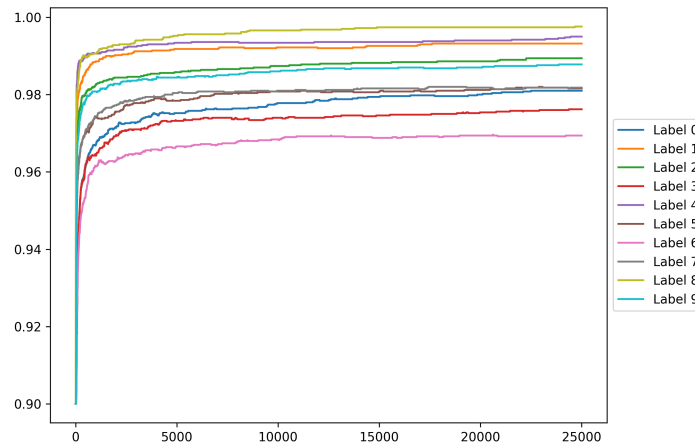


Figura 1.2: Predicciones

```

1 def run_one_vs_all() -> None:
2     """Runs the oneVsAll simulation
3     """
4     data = sio.loadmat('./data/ex3data1.mat', squeeze_me=True)
5     X = data['X']
6     y = data['y']
7     m, n = X.shape
8     num_labels = 10
9     lambda_ = 0.01
10    all_theta = []
11    predict_all = []
12
13    if not os.path.isfile(model_folder + "one_vs_all_train.mat"):
14        print("Training one vs all...")
15        all_theta, predict_all = oneVsAll(X, y, num_labels, lambda_)
16        print(f'saving model to {model_folder}...')
17        save_model(all_theta, model_folder + "one_vs_all_train.mat")
18        print(f'saving predictions to {model_folder}...')
19        save_predict(predict_all, model_folder + "one_vs_all_predict.mat")
20        print(f'saving last predictions to {model_folder}...')
21        save_last_predictions(predict_all)
22    else:
23        print("Loading model from file...")
24        aux = sio.loadmat(
25            f'{model_folder}one_vs_all_train.mat', squeeze_me=True)
26        all_theta = np.array(aux['all_theta'])
27        print('Loading predictions from file...')
28        predict_all = np.array(sio.loadmat(
29            f'{model_folder}one_vs_all_predict.mat', squeeze_me=True)['predict'])
30
31    print('Plotting predictions...')
32    print_predictions(predict_all, 25001)
33
34    p = predictOneVsAll(all_theta, X)
35    print(f'Prediction for A: {(np.sum(np.array(p) == y) / m) * 100}\'')
36
37    print('Plotting confusion matrix...')
38    plot_confusion_matrix(y, p, logistic_reg.plot_folder +
39        "confusion_matrix_one_vs_all.png")

```

Figura 1.3: Código de la función *run_one_vs_all*

```

1 def train_model(c: int, X: np.ndarray, y: np.ndarray, lambda_: float) -> tuple[int, (np.
  ndarray, np.ndarray)]:
2     """Trains a single label using logistic regression.
3
4     Args
5         c (int): The label to train
6         X (np.ndarray): The input data
7         y (np.ndarray): The values of the data
8         lambda_ (float): The regularization parameter
9     Returns:
10         tuple[int, (np.ndarray, np.ndarray)]: The label, the trained parameters and the
  predictions
11     """
12     print(f"Training {c}...")
13     initial_theta = np.zeros(X.shape[1] + 1)
14     y_i = np.array([1 if label == c else 0 for label in y])
15     alpha = 0.2
16     lambda_ = 0.01
17     num_iters = 25000
18     (w, b, _, predict) = logistic_reg.gradient_descent(X, y_i, initial_theta[1:],
  initial_theta[0],
19                                     logistic_reg.compute_cost_reg,
20                                     logistic_reg.compute_gradient_reg, alpha, num_iters, lambda_)
21     print(f"Training {c}... Done")
22     return (c, ([b] + w.tolist(), predict))

```

Figura 1.4: Código de la función *train_model*

```

1 def train_all(X: np.ndarray, y: np.ndarray, n_labels: int, lambda_: float) -> tuple[list[np.
  ndarray], list[np.ndarray]]:
2     """Trains all the labels concurrently using logistic regression.
3
4     Args:
5         X (np.ndarray): input data
6         y (np.ndarray): expected values
7         n_labels (int): number of labels to train
8         lambda_ (float): regularization parameter
9
10    Returns:
11        tuple[list[np.ndarray], list[np.ndarray]]: all theta for all labels and the
  predictions for each training
12    """
13    all_theta = [0 for _ in range(n_labels)]
14    predict_history = []
15    with concurrent.futures.ThreadPoolExecutor() as executor:
16        futures = [executor.submit(train_model, c, X, y, lambda_)
17                    for c in range(n_labels)]
18        for future in concurrent.futures.as_completed(futures):
19            c, (result, predict) = future.result()
20            all_theta[c] = result
21            predict_history.append(predict)
22    return all_theta, predict_history

```

Figura 1.5: Código de la función *train_all*

```

1 def oneVsAll(X: np.ndarray, y: np.ndarray, n_labels: int, lambda_: float):
2     """
3     Trains n_labels logistic regression classifiers and returns
4     each of these classifiers in a matrix all_theta, where the i-th
5     row of all_theta corresponds to the classifier for label i.
6
7     Parameters
8     -----
9     X : array_like
10         The input dataset of shape (m x n). m is the number of
11         data points, and n is the number of features.
12
13     y : array_like
14         The data labels. A vector of shape (m, ).
15
16     n_labels : int
17         Number of possible labels.
18
19     lambda_ : float
20         The logistic regularization parameter.
21
22     Returns
23     -----
24     all_theta : array_like
25         The trained parameters for logistic regression for each class.
26         This is a matrix of shape (K x n+1) where K is number of classes
27         (ie. 'n_labels') and n is number of features without the bias.
28     """
29     all_theta, predict_history = train_all(X, y, n_labels, lambda_)
30
31     return all_theta, predict_history

```

Figura 1.6: Código de la función *oneVsAll*

```

1 def predictOneVsAll(all_theta: np.ndarray, X: np.ndarray) -> np.ndarray:
2     """
3     Return a vector of predictions for each example in the matrix X.
4     Note that X contains the examples in rows. all_theta is a matrix where
5     the i-th row is a trained logistic regression theta vector for the
6     i-th class. You should set p to a vector of values from 0..K-1
7     (e.g., p = [0, 2, 0, 1] predicts classes 0, 2, 0, 1 for 4 examples) .
8
9     Parameters
10    -----
11    all_theta : array_like
12        The trained parameters for logistic regression for each class.
13        This is a matrix of shape (K x n+1) where K is number of classes
14        and n is number of features without the bias.
15
16    X : array_like
17        Data points to predict their labels. This is a matrix of shape
18        (m x n) where m is number of data points to predict, and n is number
19        of features without the bias term. Note we add the bias term for X in
20        this function.
21
22    Returns
23    -----
24    p : array_like
25        The predictions for each data point in X. This is a vector of shape (m, ).
26    """
27    p = []
28
29    for c in range(len(all_theta)):
30        theta = all_theta[c]
31        p.append(logistic_reg.function(X, theta[1:], theta[0]))
32    p = np.argmax(p, axis=0)
33    return p

```

Figura 1.7: Código de la función *predictOneVsAll*

```

1 def plot_confusion_matrix(y: np.ndarray, p: np.ndarray, filename: str) -> None:
2     """Plots the confusion matrix for a given prediction.
3
4     Args:
5         y (np.ndarray): expected values
6         p (np.ndarray): predicted values
7         filename (str): file to store the plot
8     """
9     fig, ax = plt.subplots()
10    ax.set_title("Confusion Matrix")
11    ax.set_xlabel("Predicted")
12    ax.set_xticks(np.arange(0, 10))
13    ax.set_yticks(np.arange(0, 10))
14    ax.set_ylabel("True")
15    cm = np.zeros((10, 10))
16    for i in range(len(y)):
17        cm[y[i] - 1][p[i] - 1] += 1
18    cax = ax.matshow(cm, cmap='Reds')
19    ax.set_xticks(np.arange(0, 10))
20    ax.set_yticks(np.arange(0, 10))
21    ax.set_yticks(np.arange(0.5, 10.5), minor='True')
22    ax.set_xticks(np.arange(0.5, 10.5), minor='True')
23    plt.grid(which='minor', color='lightgrey', linestyle='--', linewidth=0.5)
24    fig.colorbar(cax)
25    for (i, j), z in np.ndenumerate(cm):
26        if i == j:
27            ax.text(j, i, '{:0.1f}'.format(z),
28                    ha='center', va='center', fontsize=8, color='white')
29        else:
30            ax.text(j, i, '{:0.1f}'.format(z),
31                    ha='center', va='center', fontsize=8)
32    plt.savefig(filename, dpi=300)
33    plt.clf()

```

Figura 1.8: Código de la función *plot_confusion_matrix*

```

1 def print_predictions(predict: np.ndarray, num_iters: np.ndarray) -> None:
2     """Plots the prediction progress of a model.
3
4     Args:
5         predict (np.ndarray): predictions of each label
6         num_iters (np.ndarray): number of iterations of the training
7     """
8     plt.clf()
9     i = 0
10    plt.figure(figsize=(10, 6))
11    ax = plt.subplot(111)
12    for prediction in predict:
13        ax.plot(range(num_iters), prediction,
14                label=f'Label {i}')
15        i += 1
16    box = ax.get_position()
17    ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
18
19    # Put a legend to the right of the current axis
20    ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))
21    plt.savefig(logistic_reg.plot_folder +
22                "predictions_one_vs_all.png", dpi=300)
23    plt.clf()

```

Figura 1.9: Código de la función *print_predictions*

```

1 def save_model(all_theta: np.ndarray, filename: str) -> None:
2     """Saves a trained model to a matlab binary file.
3
4     Args:
5         all_theta (np.ndarray): trained model
6         filename (str): file to store the model
7     """
8     sio.savemat(filename, {'all_theta': all_theta})

```

Figura 1.10: Código de la función *save_model*

```

1 def save_predict(predict: np.ndarray, filename: str) -> None:
2     """Saves the predictions of a model in a matlab binary file.
3
4     Args:
5         predict (np.ndarray): predictions of a model
6         filename (str): file to store the predictions
7     """
8     sio.savemat(filename, {'predict': predict})

```

Figura 1.11: Código de la función *save_predict*

```

1 def save_last_predictions(predict_all: np.ndarray) -> None:
2     """Saves the last predictions of a model to a csv file.
3
4     Args:
5         predict_all (np.ndarray): last predictions of each file
6     """
7     with open(model_folder + "one_vs_all_predict.csv", "w") as f:
8         csv_writer = csv.writer(f)
9         headers = [str(i) for i in range(10)]
10        csv_writer.writerow(headers)
11        data = []
12        for i in range(10):
13            data.append(str(predict_all[i][-1]))
14
15        csv_writer.writerow(data)

```

Figura 1.12: Código de la función *save_last_predictions*

2. Parte B: Regresión lógica regularizada

En el apartado B usaremos el *dataset* anterior, pero esta vez nos dan un modelo ya entrenado para hacer la red neuronal. Para ello usaremos la función *predict* (figura 2.2) para hacer la predicción basándonos en el modelo dado. Usaremos la función de matriz de antes (1.8) para hacer la matriz de confusión (figura 2.1) y la función *run_neural_network* (figura 2.3) para como driver de este apartado.

Tras ejecutar la red, conseguimos una predicción de 97.52%.

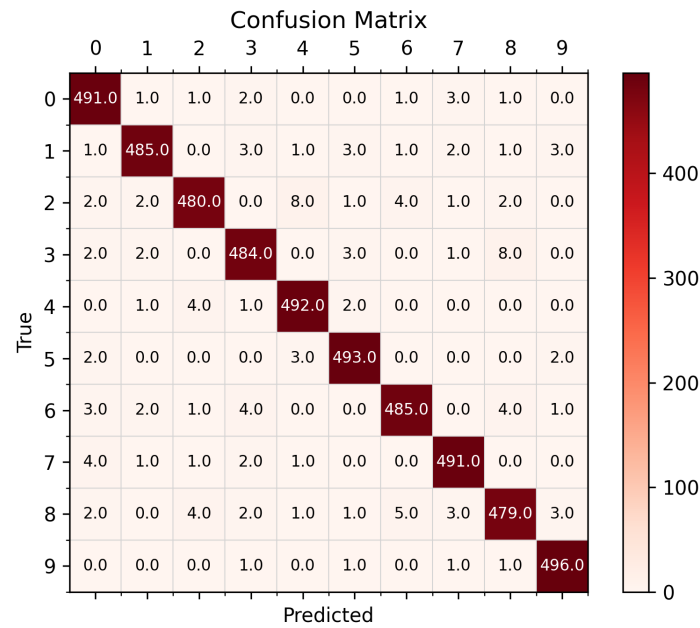


Figura 2.1: Matriz de confusión

```

1 def predict(theta1: np.ndarray, theta2: np.ndarray, X: np.ndarray) -> np.ndarray:
2     """
3     Predict the label of an input given a trained neural network.
4
5     Parameters
6     -----
7     theta1 : array_like
8         Weights for the first layer in the neural network.
9         It has shape (2nd hidden layer size x input size)
10
11     theta2: array_like
12         Weights for the second layer in the neural network.
13         It has shape (output layer size x 2nd hidden layer size)
14
15     X : array_like
16         The image inputs having shape (number of examples x image dimensions).
17
18     Return
19     -----
20     p : array_like
21         Predictions vector containing the predicted label for each example.
22         It has a length equal to the number of examples.
23     """
24
25     a_2 = logistic_reg.function(X, theta1[1:], theta1[0])
26     p = logistic_reg.function(a_2, theta2[1:], theta2[0])
27
28     return np.argmax(p, axis=1)

```

Figura 2.2: Código de la función *predict*


```
1 def run_neural_network() -> None:
2     """Runs the neural network simulation
3     """
4     data = sio.loadmat('./data/ex3data1.mat', squeeze_me=True)
5     X = data['X']
6     y = data['y']
7     m, n = X.shape
8     weights = sio.loadmat('./data/ex3weights.mat', squeeze_me=True)
9     theta1, theta2 = weights['Theta1'], weights['Theta2']
10    p = predict(theta1.T, theta2.T, X)
11
12    print(f'Prediction for B: {(np.sum(np.array(p) == y) / m) * 100}%)')
13    print('Plotting confusion matrix...')
14    plot_confusion_matrix(
15        y, p, logistic_reg.plot_folder + "confusion_matrix_nn.png")
16
17
18 def main() -> None:
19     run_one_vs_all()
20     run_neural_network()
21
22
23 if __name__ == "__main__":
24     main()
```

Figura 2.3: Código de la función *run_neural_network*