Bài tập về nhà 1 - Logistic Regression

Ngày 28 tháng 9 năm 2017

Tóm tắt nội dung

Trong bài tập này, các bạn sẽ sử dụng kiến thức đã học về logistic regression để giải quyết bài toán phân lớp.

Mục lục

1 Giới thiệu

Để có thể hoàn tất bài tập này, các bạn cần nắm rõ những kiến thức sau:

- Logistic regression là gì, nguyên tắc hoạt động ra sao.
- Cách lấy đạo hàm cho các tham số trong mô hình logistic regression.
- Giải thuật gradient descent.

Các bạn có thể tham khảo lại bài giảng của lớp để nắm vững các nội dung này. Ngoài ra, các bạn có thể đặt câu hỏi cho đội ngũ giảng dạy nếu có thắc mắc.

Bài tập này sẽ gồm có hai bài chính:

- Bài 1: phân loại hai lớp dùng logistic regression.
- Bài 2: phân loại ba lớp dùng logistic regression.

Yêu cầu dành cho các bạn trong là giải quyết hai bài trên bằng numpy và TensorFlow.

2 Hướng dẫn làm bài và nộp bài

2.1 Cấu trúc file

Bài tập lớn này được đi kèm với các file python sau:

- **data_util.py**: cung cấp các hàm để đọc dữ liệu trong thư mục data thành các ma trận numpy. Bạn không cần chỉnh sửa file này.
- **bin_classify_np.py**: cung cấp các hàm dựng sẵn để giải quyết bài 1, dùng numpy.
- **multi_classify_np.py**: dành cho bài 2, dùng numpy.
- **bin_classify_tf.py**: dành cho bài 1, nhưng code phải dùng TensorFlow thay vì numpy.
- **multi_classify_tf.py**: dành cho bài 2, nhưng code phải dùng TensorFlow thay vì numpy.
- **bin_classify_np_unittest.py**: dành cho bài 1, các bạn có thể thực thi file này để kiểm tra xem code của mình trong bin_classify_np.py có chính xác chưa.

- **eclipse-data.npz**: dữ liệu của bài 1.
- **iris.dat**: dữ liệu của bài 2.

2.2 Hướng dẫn nộp bài

Để nộp bài, các bạn nén thư mục gốc lại (gồm các file .py và thư mục con data) thành file zip. Định dạng là [Tên_không_dấu].zip.

Ví dụ: [HUYNH_CHI_KIEN].zip

Sau đó, các bạn gửi file này đến mail phat.hoang@med2lab.com với tiêu đề là [AI COURSE MED2LAB 2017] Assignment 1 - Tên Bạn.

Ví dụ: [AI COURSE MED2LAB 2017] Assignment 1 - Huỳnh Chí Kiên.

3 Bài 1 - Phân loại hai lớp

3.1 Dữ liệu Eclipse

Tập dữ liệu Eclipse là tập gồm có 2 lớp, được gán nhãn lớp 0 và 1. Ta có thể đọc tập dữ liệu này bằng hàm get_eclipse_data():

```
train_x, train_y, test_x, test_y = get_eclipse_data()
```

Ở đây, train_x là biến mảng numpy có kích thước 3200×2 (ý nghĩa: tập dữ liệu huấn luyện train_x có 3200 mẫu, mỗi mẫu có 2 chiều). train_y là ma trận chứa nhãn ứng với mỗi hàng trong train_x. Tương tự, test_x có kích thước 800×2, mỗi hàng trong test_y biểu diễn cho nhãn của mỗi hàng trong test_x. Hai mảng train_x và train_y được dùng cho việc huấn luyện mô hình phân loại; hai mảng test_x và test_y được dùng cho quá trình đánh giá (test).

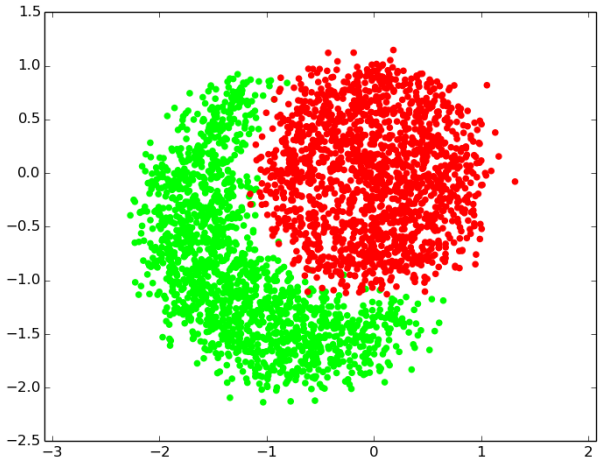
Để hiển thị các điểm dữ liệu huấn luyện, ta có thể dùng đoạn code sau:

```
fig, ax = plt.subplots()
rgb = np.eye(3)
ax.scatter(train_x[:,0], train_x[:,1],
color = rgb[train_y[:,0].astype(np.int32), :])
plt.axis('equal')
plt.ion()
plt.show()
```

Tập dữ liệu được phân bố như trong Hình 1.

3.2 Class logistic_classifier

Để tiện cho việc lập trình, trong file có cung cấp sẵn (python) class logistic_classifier. (Lưu ý: để tiện cho việc phân biệt giữa lớp python và lớp trong



Hình 1: Các điểm dữ liệu trong tập Eclipse. Đỏ: lớp 1, Xanh: lớp 0.

bài toán phân loại, người viết sẽ qui ước rằng khi viết **class** nghĩa là đang nói về python class, khi viết **lớp** nghĩa là đang ám chỉ lớp của dữ liệu cần phân loại)

Một trong các thành phần chính của class logistic_classifier là biến w, chứa tham số mà ta cần tìm khi huấn luyện. Tham số này là một mảng 3×1, được khởi tạo ngẫu nhiên trong hàm __init__(w_shape=(3,1)). Để truy xuất w từ bên trong class, ta dùng self.w:

```
class LogisticClassifier(object):
        def feed_forward(self, x):
                print(self.w)
```

Để truy xuất w từ bên ngoài class, ta cần có một thực thể của class và gọi thông qua thực thể này:

```
if __name__ == "__main__":
        num_feature = train_x.shape[1]
        bin_classifier = LogisticClassifier((num_feature, 1))
        print(bin_classifier.w)
```

Đối với các hàm thuộc class logistic_classifier, việc truy xuất cũng hoàn toàn giống với w. Chúng sẽ được mô tả chi tiết trong mục tiếp theo.

3.3 Thực hiện bài 1 với numpy

Trong phần này, bạn thực hiện các bước trình bày bên dưới vào trong file tạo sẵn bin_classify_np.py.

3.3.1 Bước 1 - Thêm feature vào x [YC1.1]

Để tính tích vô hướng dễ dàng, nối thêm một cột có giá trị bằng 1 vào train_x và test_x (concatenate có axis=1). Trong file có sẵn hàm add_one và ta nên thực hiện code trong hàm này.

3.3.2 Bước 2 - Tính các giá trị phân loại [YC1.2]

Các giá trị phân loại, $\hat{y}$ / y_hat, sẽ được tính trong hàm feed_forward của class logistic_classifier. Công thức tính như sau:

$$z = x^T w \tag{1}$$

$$\hat{y} = \frac{1}{1 + e^{-z}} \tag{2}$$

Ở đây, $w = [w_0, w_1, w_2]^T$ là các tham số cần học (lưu trong biến self.w trong class logistic_classifier).

3.3.3 Bước 3 - Tính độ lỗi [YC1.3]

Việc tính độ lỗi được thực hiện trong hàm compute_loss của class logistic_classifier. Công thức tính độ lỗi như sau:

$$J(w) = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})) \tag{3}$$

Trong đó:

- $y^{(i)}$ là nhãn của mẫu thứ i, mẫu thuộc lớp 0 sẽ có $y^{(i)} = 0$, mẫu thuộc lớp 1 sẽ có $y^{(i)} = 1$. Ta có thể truy cập các nhãn này tông qua biến train_y và test_y.
- $\hat{y}^{(i)}$ là phần tử thứ i trong vector $\hat{y}$.
- $m = 3200$ là tổng số mẫu huấn luyện.

Để tính trung bình trên ma trận theo hàng hoặc cột, ta có thể sử dụng hàm np.mean() với tham số axis tương ứng.

3.3.4 Bước 4 - Tính gradient [YC1.4]

Để tính đạo hàm riêng cho thành phần w_j trong w trong hàm get_grad, ta dùng công thức sau:

$$\frac{\partial J(w_j)}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)} \tag{4}$$

Trong trường hợp này, sau khi thêm 1 vào train_x thì ta sẽ có $j \in \{0, 1, 2\}$.

3.3.5 Bước 5 - Cập nhật w [YC1.5]

Để huấn luyện được mô hình phân loại trong hàm update_weight, ta cần cập nhật w theo công thức sau:

$$w = w - \alpha \times \frac{\partial J(w)}{\partial w} \tag{5}$$

Với α là hệ số học (learning_rate).

3.3.6 Bước 6 - Đánh giá mô hình phân loại [YC1.6]

Để đánh giá mô hình phân loại trên tập kiểm thử (test_x và test_y), trước tiên, ta cần thực hiện tính các giá trị phân loại trên test_x. Sau khi đã có các giá trị này, ta sử dụng các tiêu chí sau để đánh giá mô hình:

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{P} \tag{7}$$

$$F_1 - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{8}$$

Với lớp positive là lớp có giá trị y = 1, TP là tổng số các mẫu mà mô hình dự đoán là positive ($\hat{y} = 1$) và thực sự có nhãn là positive ($y = 1$), FP là tổng số các mẫu mô hình dự đoán là positive ($\hat{y} = 1$) nhưng thực chất có nhãn là negative ($y = 0$), P là tổng số mẫu positive trong tập test.

Nhiệm vụ của bạn trong bước này là tính các thông số trên trong hàm test. Trong hàm này, bạn chỉ cần in các kết quả ra bằng hàm print, không cần return.

3.3.7 Vòng lặp huấn luyện

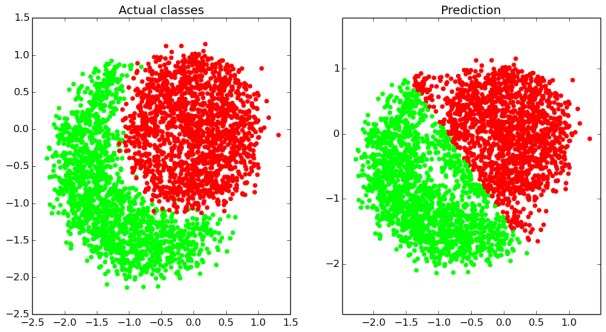
Vòng lặp của quá trình huấn luyện được xây dựng trong đoạn code main của file. Tất cả khung sườn cho việc thực thi đã được lập trình sẵn. Ta có thể thay đổi hai tham số tác động đến quá trình huấn luyện như sau:

- num_epoch: số lượng vòng lặp cho quá trình huấn luyện.
- learning_rate: hệ số học.

3.3.8 Các hàm hỗ trợ

Nhằm hỗ trợ việc lập trình, hai hàm hỗ trợ đã được hoàn tất sẵn gồm có:

- numerical_check (trong class logistic_classifier): hàm này nhằm kiểm tra xem gradient ta đã tính có chính xác không. Sau khi tính thử gradient bằng hàm get_grad, ta có thể truyền nó vào hàm numerical_check. Hàm numerical_check sẽ tính gradient dựa vào các phương pháp tính xấp xỉ và so nó với gradient ta đã tính được. Cả hai lượng gradient chỉ nên lệch nhau trong hoảng $\pm 10^{-6}$. Ví dụ: 1.2345678 và 1.2345683.



Hình 2: Hiển thị dữ liệu dựa theo nhãn bằng hàm visualize_data. Trái: phân bố lớp thực sự của dữ liệu. Phải: phân bố lớp được dự đoán bởi mô hình phân loại.

- visualize_data: ta có thể truyền vào hàm này train_x, train_y và y_hat để quan sát khả năng hoạt động của bộ phân loại trên chính tập dữ liệu huấn luyện (Hình 2). Bạn có thể điều khiển tần số hiển thị bằng biến draw_frequency.
- unit_test: nằm trong file bin_classify_np.py, bạn có thể gọi hàm này để kiểm tra các hàm chính của bài 1. Để gọi hàm này, trong thư mục data cần có file unittest.npy.

3.3.9 Phần làm thêm

Nếu hoàn tất được các phần này, bạn sẽ được điểm cộng.

Tăng thêm đặc trưng cho x [YC1.7]: với dữ liệu hiện tại, x chỉ có hai đặc trưng là x_1 và x_2. Với hai đặc trưng này, mô hình phân loại chỉ có thể phân loại tốt các tập dữ liệu có sự tách biệt rõ rệt. Vì vậy, đối với tập Eclipse, việc phân loại vẫn còn một số sai sót (Hình 2). Để cải thiện, ta có thể thêm các đặc trưng đa thức vào x. Công việc này được thực hiện trong hàm add_poly_feature. Tham số degree trong hàm này chỉ định bậc đa thức tối đa của các đặc trưng mới. Giả sử, với degree=2 ta sẽ có:

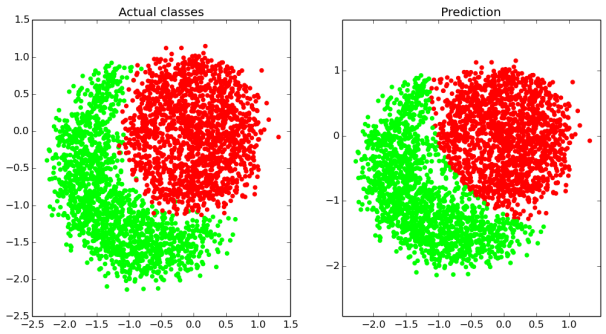
$$x = [x_1,\ x_2,\ x_1^2,\ x_1x_2,\ x_2^2] \tag{9}$$

Với degree = 3:

$$x = [x_1,\ x_2,\ x_1^2,\ x_1x_2,\ x_2^2,\ x_1^3,\ x_1^2x_2,\ x_1x_2^2,\ x_2^3] \tag{10}$$

Nếu thành công, chỉ cần với degree=2, ta sẽ có được kết quả như Hình 3.

Sử dụng giải thuật cập nhật w có quán tính [YC1.8]: giải thuật cập nhật trình bày trong 3.3.5 có điểm yếu là chậm và dễ rơi vào tối ưu cục bộ. Tuy trong bài này, giải thuật đó cũng đủ để giải quyết bài toán của ta, nhưng ta vẫn có thể sử dụng giải thuật có quán tính để việc huấn luyện diễn ra nhanh hơn.



Hình 3: Kết quả phân loại sau khi thêm đặc trưng đa thức bậc 2 vào x.

Khởi tạo quán tính trước khi vào vòng lặp chính:

$$\Delta w = 0 \tag{11}$$

Ở đây, Δw là ma trận có kích thước bằng chính kích thước của w. Quá trình cập nhật w sẽ được diễn ra như sau:

$$\Delta w = \gamma \Delta w + \alpha \frac{\partial J(w)}{\partial w} \tag{12}$$

$$w = w - \Delta w \tag{13}$$

Với γ là hệ số quán tính (thường được đặt là 0.9).

3.3.10 Kết quả kỳ vọng

Nếu hoàn thành đúng các bước trên (mà không thêm đặc trưng đa thức), các độ đo của bạn sẽ như sau:

- $0.9 \leq Precision \leq 0.95$
- $0.9 \leq Recall \leq 0.95$
- $0.9 \leq F1 - score \leq 0.95$

Sau khi thêm đặc trưng đa thức bậc hai, các độ đo sẽ tốt hơn:

- $Precision \geq 0.95$
- $Recall \geq 0.95$
- $F1 - score \geq 0.95$

Cụ thể, người viết bài đã kiểm thử với đặc trưng đa thức và đạt mốc $F1 - score = 0.98$.

3.4 Thực hiện bài 1 với TensorFlow [YC1.9]

Để thực hiện công việc này, bạn lập trình trong file tạo sẵn bin_classify_tf.py. Các bước để hoàn tất bài 1 hoàn toàn tương tự như đã trình bày ở phần trước.

4 Bài 2 - Phân loại ba lớp

4.1 Dữ liệu hoa Iris

Tập dữ liệu Iris là tập gồm có 3 lớp, được gán nhãn lớp 1, 2 và 3 tương ứng với 3 loại hoa IRIS - setosa, versicolor, virginica. Tập dữ liệu này do Fisher xây dựng với 150 mẫu dữ liệu (50 mẫu dữ liệu cho mỗi loại hoa). Mỗi mẫu có 4 thông số đầu vào và 1 thông số đầu ra chỉ ra loại hoa tương ứng. Các thông số đầu vào gồm chiều dài và chiều rộng của đài hoa (sepal width and sepal length), chiều dài và chiều rộng của cánh hoa (petal length and petal width). Mục tiêu của bài toán là từ input features ta có thể xây dựng được bộ phân loại để biết đó là loài hoa gì - setosa, versicolor, virginica.

Hình 4: Iris flower - Setosa

Hình 5: Iris flower - Versicolor

Ta có thể đọc tập dữ liệu này bằng hàm get_iris_data():

Hình 6: Iris flower - Virginica

```
train_x, train_y, test_x, test_y = get_iris_data()
```

Lưu ý là giá trị của train_y và test_y sẽ có thể là 1, 2 hoặc 3 thay vì 0 và 1 như bài 1.

4.2 Sử dụng lại class logistic_classifier

Để hoàn tất bài tập này, ta có thể dùng ba mô hình phân loại hai lớp:

- Bộ phân loại một sẽ xem lớp 1 là positive (1), lớp 2 và 3 như negative (0).
- Bộ phân loại hai sẽ xem lớp 2 là positive (1), lớp 1 và 3 như negative (0).
- Bộ phân loại hai sẽ xem lớp 3 là positive (1), lớp 1 và 2 như negative (0).

Giả sử như có một mẫu thử x^{test} cần được phân loại, ta phải đưa mẫu nào vào cả ba mô hình. Nếu mô hình 1 cho ra kết quả cao nhất, ta sẽ xem x^{test} thuộc lớp thứ nhất, tương tự cho mô hình 2 và 3.

4.3 Thực hiện bài 2 với numpy

Sau khi đã hoàn tất bài 1 với numpy thì ta có thể sử dụng lại class logistic_regression vào bài này để tiết kiệm thời gian. Mục tiêu của bài 2 gần giống với bài 1, chỉ khác ở hai điểm chính là xây dựng dữ liệu huấn luyện và đánh giá mô hình. Bạn cần hoàn tất các mục TODO trong file multi_classify_np.py

4.3.1 Xây dựng dữ liệu huấn luyện [YC2.1]

Biến train_y có các giá trị 1, 2 và 3; nhưng hàm lỗi của logistic regression chỉ có thể nhận đầu vào cho các y là 0 hoặc 1. Vì vậy để huấn luyện cho từng bộ phân loại, ta phải biến đổi giá trị của train_y về 0, 1. Cách biến đổi tương tự như đã trình bày trong mục 4.2.

Giả sử như train_y của ta là:

$$train\_y = [1, 1, 2, 2, 3, 3]^T \tag{14}$$

Thì đối với bộ phân loại 1, ta cần biến đổi nó thành:

$$train\_y = [1, 1, 0, 0, 0, 0]^T \tag{15}$$

Đối với bộ phân loại 2, ta cần biến đổi nó thành:

$$train\_y = [0, 0, 1, 1, 0, 0]^T \tag{16}$$

Và tương tự cho bộ phân loại 3.
Lưu ý:

- NÊN thực hiện copy mảng train_y vào một biến khác bằng np.copy() rồi thực hiện biển đổi trên mảng mới đó. Việc này là bởi vì phép gán train_y_new = train_y chỉ mang ý nghĩa là train_y_new là một biến tham khảo đến train_y. Khi đó, mọi sự thay đổi trên train_y_new đều diễn ra trên train_y. Đây là điều không mong muốn khi ta cần bảo toàn các giá trị của train_y.
- Để việc biến đổi giá trị của mảng sang nhị phân được nhanh chóng, ta có thể sử dụng chỉ mục boolean trong numpy. Tham khảo thêm ở đây: Numpy basic indexing - boolean index arrays.

4.3.2 Đánh giá mô hình phân loại [YC2.2]

Để đánh giá mô hình phân loại 3 lớp, ta cần một khái niệm gọi là confusion matrix.

	Lớp 1	Lớp 2	Lớp 3
Lớp 1	N_{11}	N_{12}	N_{13}
Lớp 2	N_{21}	N_{22}	N_{23}
Lớp 3	N_{31}	N_{32}	N_{33}

Trong đó, N_{kl} là tổng số mẫu thực chất thuộc lớp k và bộ phân loại phân loại thành lớp l. Bộ phân loại càng tốt thì các ô trên đường chéo chính sẽ càng cao hơn so với các ô xung quanh.

4.3.3 Kết quả kỳ vọng

Trong bài này, nếu làm đúng thì kết quả confusion matrix sẽ ra như sau:

	Lớp 1	Lớp 2	Lớp 3
Lớp 1	10	0	0
Lớp 2	0	10	0
Lớp 3	0	0	10

4.4 Thực hiện bài 2 với TensorFlow [YC2.3]

Bạn lập trình trong file tạo sẵn multi_classify_tf.py. Các bước để hoàn tất bài 2 hoàn toàn tương tự như đã trình bày ở các phần trước.

5 Cách thực thi

Để thực thi các file python trong bài tập này, các bạn có thể thêm thư mục chứa các file code vào trong IDE (pycharm, spyder, v.v) và chạy thử. Ngoài ra, bạn có thể sử dụng command line (Windows) hoặc terminal (Ubuntu) để thực thi. Cú pháp là python [tên file]. Ví dụ, để thực thi file bin_classify_np.py, ta gọi lệnh sau:

```
python bin_classify_np.py
```

6 Chấm điểm

Việc chấm điểm cho các bạn sẽ dựa vào thang sau:

Bài 1 (numpy)	
Hiện thực add_one	5
Hiện thực feed_forward	10
Hiện thực compute_loss	10
Hiện thực get_grad	10
Hiện thực update_weight	10
Hiện thực test	5
Hiện thực add_poly_feature	10
Sử dụng giải thuật GD với momentum	10
Bài 1 (TensorFlow)	
Hiện thực được các bước chính bằng TF	20
Bài 2 (numpy)	
Xử lý data và huấn luyện đúng	10
Tính được confusion matrix	10
Bài 2 (TensorFlow)	
Hiện thực được bước huấn luyện ba bộ phân loại bằng TF	10
Tổng điểm	120

Tính trung thực trong khi làm bài tập là điều quan trọng trong bài tập này. Nếu bất kì thành viên nào sao chép mã chương trình (code) hoặc báo cáo (report) và bị phát hiện thì sẽ không được tính điểm cho bài tập này. Điểm bài tập sẽ được tính vào cuối khóa học nhằm giúp các bạn có chứng nhận về việc học tập tại khóa PilotAI class. Ngoài ra việc nộp bài tập muộn vẫn chấm nhân nhưng sẽ bị trừ điểm theo số ngày nộp muộn (nộp muộn 1 ngày bị trừ 10 điểm, 2 ngày sẽ bị trừ 30 điểm, 3 ngày bị trừ 50 điểm, quá 4 ngày sẽ không tính điểm).