ĐẠI HỌC BÁCH KHOA HÀ NỘI

Trường Công Nghệ Thông Tin Và Truyền Thông

PROJECT I

Giảng viên hướng dẫn: TS. BÙI QUỐC TRUNG

Sinh viên: Nguyễn Duy Thành

Mã số sinh viên: 20204691

Lớp: Khoa học máy tính 02-K65

Hà Nội, 02-2023

MỞ ĐẦU

Machine Learning (Học máy) là một nhánh của **AI (trí tuệ nhân tạo)** và khoa học máy tính, tập trung vào việc sử dụng dữ liệu và thuật toán để bắt chước cách học của con người, dần dần cải thiện độ chính xác của nó.

Trong vài thập kỷ qua, những tiến bộ công nghệ về khả năng lưu trữ và xử lý đã cho phép một số sản phẩm sáng tạo dựa trên máy học, chẳng hạn như công cụ đề xuất của Netflix và ô tô tự lái.

Học máy là một thành phần quan trọng trong lĩnh vực khoa học dữ liệu đang phát triển. Thông qua việc sử dụng các phương pháp thống kê, các thuật toán được đào tạo để phân loại hoặc dự đoán và khám phá những hiểu biết chính trong các dự án khai thác dữ liệu. Những thông tin chi tiết này sau đó sẽ thúc đẩy quá trình ra quyết định trong các ứng dụng và doanh nghiệp, tác động lý tưởng đến các chỉ số tăng trưởng chính. Khi dữ liệu lớn tiếp tục mở rộng và phát triển, nhu cầu thị trường đối với các nhà khoa học dữ liệu sẽ tăng lên. Họ sẽ được yêu cầu giúp xác định các câu hỏi kinh doanh phù hợp nhất và dữ liệu để trả lời chúng.

Những năm gần đây, khi mà khả năng tính toán của các máy tính được nâng lên một tầm cao mới và lượng dữ liệu khổng lồ được thu thập bởi các hãng công nghệ lớn, Machine Learning đã tiến thêm một bước dài và một lĩnh vực mới được ra đời gọi là Deep Learning (Học Sâu). Deep Learning đã giúp máy tính thực thi những việc tưởng chừng như không thể vào 10 năm trước: phân loại cả ngàn vật thể khác nhau trong các bức ảnh, tự tạo chú thích cho ảnh, bắt chước giọng nói và chữ viết của con người, giao tiếp với con người, hay thậm chí cả sáng tác văn hay âm nhạc.

Trong báo cáo này, tôi bắt đầu tìm hiểu những thuật toán cơ bản nhất của Machine Learning để từng bước tiếp cận những kiến thức liên quan tới lĩnh vực này!

MỤC LỤC

1 Linear Regression 1
1.1 Giới thiệu 1
1.2 Phân tích toán học 2
1.2.1 Mean Square Error Loss (MSE) 2
1.2.2 Tối ưu hàm mất mát 2
1.2.3 Giải thuật Gradient Descent 3
1.3 Triển khai với Numpy 4
1.3.1 Cài đặt mô hình Linear regression 4
1.3.2 Chạy thử 5
1.3.2.1 Khởi tạo dữ liệu 5
1.3.2.2 Khởi tạo mô hình và training 5
1.3.2.3 Kết quả thu được 6
1.4 Kết luận 7
2 Logistic Regression 8
2.1 Giới thiệu 8
2.2 Phân tích toán học 9
2.2.1 Binary Cross Entropy Loss (BCE) 9
2.2.2 Tối ưu hàm mất mát 10
2.3 Triển khai với Numpy 11
2.3.1 Cài đặt mô hình Logistic Regression 11
2.3.2 Chạy thử 12
2.3.2.1 Nạp dữ liệu 12
2.3.2.2 Training và so sánh với SkLearn 12
2.3.2.3 Kết quả thu được 14
2.4 Kết luận 15

1 Linear Regression

1.1 Giới thiệu

Đầu tiên, ta sẽ đi tìm hiểu về một trong những thuật toán cơ bản nhất của Machine Learning (Học máy) nhưng lại rất hiệu quả trong nhiều bài toán của Học máy, đó là thuật toán **Linear Regression** (Hồi quy tuyến tính). Các mô hình hồi quy tuyến tính tương đối đơn giản và cung cấp một công thức toán học dễ hiểu để có thể tạo ra các dự đoán. Hồi quy tuyến tính có thể được áp dụng cho các lĩnh vực khác nhau trong kinh doanh và nghiên cứu học thuật. Các mô hình hồi quy tuyến tính đã được chứng minh để dự đoán tương lai một cách khoa học và đáng tin cậy. Sau đây ta sẽ đi tìm hiểu về một ví dụ đơn giản ứng dụng hồi quy tuyến tính!

Ví dụ về dự đoán doanh thu

Giả sử một công ty có doanh thu là *y* và có các thuộc tính đại diện cho chi phí quảng cáo, tiền thu mặt bằng, chi phí nhân sự,… lần lượt là $x_1, x_2, x_3, ...$ Chúng ta mong muốn tìm được một **hàm tuyến tính** biểu diễn mối tương quan này như sau:

$$y \approx f(x) = w_1x_1 + w_2x_2 + w_3x_3 + w_0$$

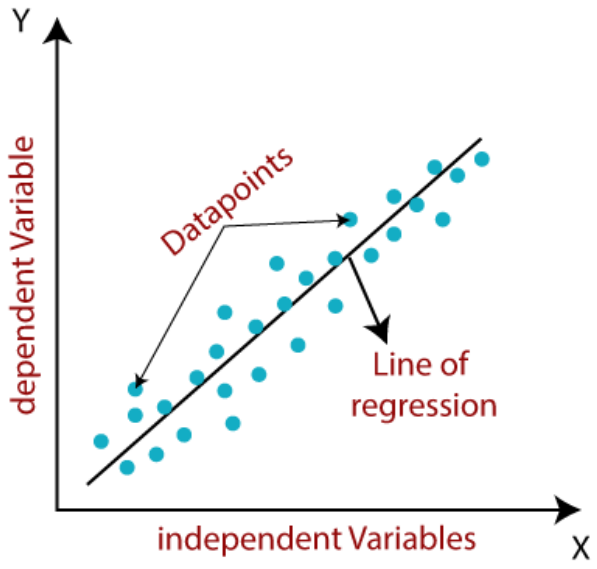
trong đó w_1, w_2, w_3 là các hằng số, w_0 được gọi là bias

Khi đó, Linear Regression có dạng:

$$\hat{y} = \bar{x}w$$

trong đó, $\hat{y}$ là giá trị Linear Regression dự đoán được, $\bar{x}=[1, x_1, x_2, x_3]$ là vector hàng chứa thông tin input của training data, $w=[w_0, w_1, w_2, w_3]^T$ là vector cột hệ số cần tối ưu.

Biểu diễn hình học:



1.2 Phân tích toán học

1.2.1 Mean Square Error Loss (MSE)

Chúng ta mong muốn sự sai khác giữa giá trị thực y và giá trị dự đoán $\hat{y}$ là nhỏ nhất. Điều đó đồng nghĩa với việc tìm vector hệ số w sao cho giá trị hàm Mean Square Error Loss (MSE) càng nhỏ càng tốt:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Hay theo chuẩn Euclid, ta có thể viết lại:

$$MSE = J(w) = \frac{1}{N}\sum_{i=1}^{N}(y_i - (\bar{x}_i w))^2 = \frac{1}{N}\|y - \bar{X}w\|_2^2$$

Trong đó, $y = [y_1, y_2, y_3]$ là vector cột chứa output của training data, $\bar{X} = [x_1, x_2, x_3]$ là ma trận dữ liệu đầu vào mà mỗi hàng là một điểm dữ liệu.

1.2.2 Tối ưu hàm mất mát

Chúng ta cần **cực tiểu hoá hàm lỗi này** với tham số cần tìm là *w*. Để tìm cực trị của hàm này, ta cần phải xét các **điểm dừng** tức là giải phương trình đạo hàm bằng 0. Các **điểm dừng** này được gọi là **local minimum** (Cực trị địa phương) để phân biệt với **global minimum** (là điểm mà tại đó hàm số đại giá trị nhỏ nhất). Có thể thấy **global minimum** là một trường hợp đặc biệt của **local minimum** và chính là bộ tham số mà chúng ta đang kì vọng tìm được.

Đạo hàm theo *w* của hàm mất mát là:

$$\frac{\partial J(w)}{\partial w} = \frac{2}{N}\bar{X}^T(\bar{X}w - y)$$

Phương trình đạo hàm bằng 0 tương đương với:

$$\bar{X}^T\bar{X}w = \bar{X}^T y$$

có nghiệm: $w = (\bar{X}^T\bar{X})^{-1}\bar{X}^T y$

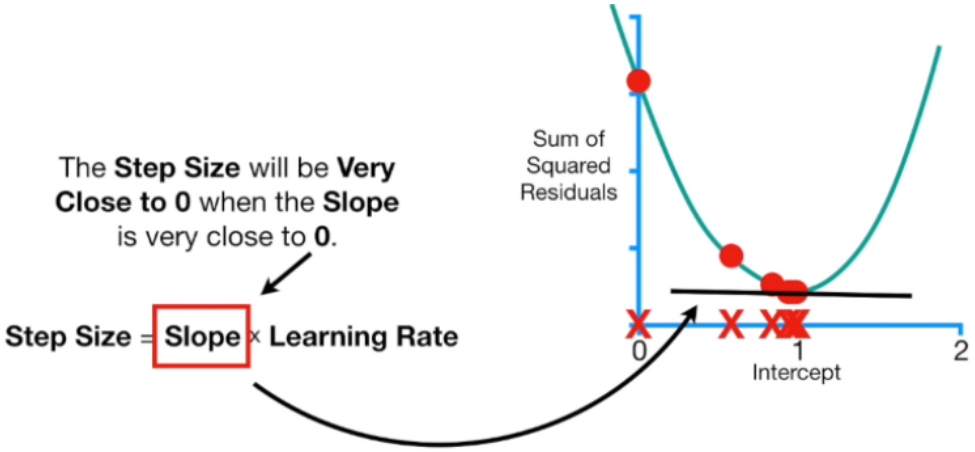
Nhận xét:

Việc đi tìm các điểm **local minimum** bằng cách giải phương trình đạo hàm bằng 0 không phải lúc nào cũng đơn giản và có thể bất khả thi khi số lượng điểm dữ liệu (*x*, *y*), số chiều quá lớn hoặc do việc tính toán đạo hàm quá phức tạp. Điều này khiến việc trực tiếp giải phương trình đạo hàm bằng 0 một cách thủ công trở nên khó khăn. Thay vào đó, ta sẽ sử dụng giải thuật **Gradient Descent** để tìm các điểm **local minimum** và ở một mức độ chấp nhận được nào đó và coi như là **global minimum** của bài toán.

1.2.3 Giải thuật Gradient Descent

Tóm tắt các bước của giải thuật Gradient Descent:

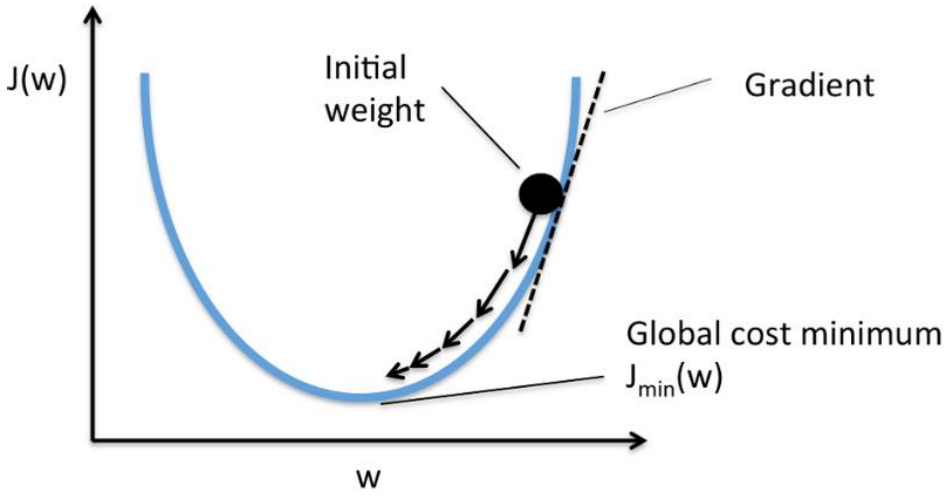
- **Bước 1:** Chọn giá trị khởi tạo cho tham số w
- **Bước 2:** Thay giá trị tham số w vào đạo hàm $J'(w)$ (được gọi là **Slope** – Độ dốc)
- **Bước 3:** Tính kích thước bước: Kích thước bước (**Step size**) = Độ dốc*Tốc độ học
- **Bước 4:** Tính toán tham số mới: Tham số mới = Tham số cũ – Kích thước bước
- **Bước 5:** Quay lại bước 2 và lặp lại cho đến khi Kích thước dốc gần bằng 0 hoặc đạt đến Số bước tối đa. Lúc này Kích thước bước rất gần bằng 0 và ta coi như tìm được w sao cho hàm mất mát đạt giá trị cực tiểu.



Ta có công thức cập nhật đạo hàm:

$$W_{t+1} = W_t - \alpha J'(W_t)$$

Trong đó hệ số α được gọi là tốc độ học hay **learning rate** và dấu trừ thể hiện việc di chuyển ngược theo hướng của đạo hàm.



1.3 Triển khai với Numpy

1.3.1 Cài đặt mô hình Linear regression

```python
class LinearRegression:
    def __init__(self, lr=0.01, epochs=1000):   # lr - Learnint rate: tốc độ học, epochs: số bước lặp tối đa
        self.epochs = epochs
        self.lr = lr
        self.w = 0

    # Hàm khởi tạo tham số w
    def initialize(self, n_features):
        self.w = np.zeros((n_features))

    # Hàm tính độ dốc J'(w)
    def gradient(self, X, y, n_samples):
        y_pred = self.predict(X)
        d_w = (2 / n_samples) * np.dot(X.T, (y_pred - y))
        return d_w

    def fit(self, X, y):
        # Lấy khuôn dạng X (số hàng, số cột)
        n_samples, n_features = X.shape
        # Khởi tạo tham số
        self.initialize(n_features)
        # Tính đạo hàm sau mỗi bước
        for _ in tqdm(range(self.epochs)):
            # Cập nhật tham số
            d_w = self.gradient(X, y, n_samples)
            self.w -= self.lr * d_w
        return self.w

    # Hàm dự đoán kết quả
    def predict(self, X):
        return np.dot(X, self.w)
```

(Ảnh code class Linear Regression)

Trong đó chúng ta thấy hàm gradient() được triển khai từ công thức trong phần cực tiểu hoá hàm loss

$$\frac{\partial J(w)}{\partial w} = \frac{2}{N}\bar{X}^T(\bar{X}w - y)$$

Trong hàm fit() chúng ta cài đặt giải thuật **Gradient Descent** lặp lại theo từng bước đến khi đạt được số lượng bước nhất định.

1.3.2 Chạy thử

1.3.2.1 Khởi tạo dữ liệu

Phân chia dữ liệu thành 2 phần training và testing

```
# Hàm nạp dữ liệu
def Loadtxt(path):
    try:
        raw = np.loadtxt(path,delimiter = ',')
        X = np.zeros((np.size(raw,0),np.size(raw,1)))
        X[:,0] = 1
        X[:,1:] = raw[:,:-1]
        y = raw[:,-1]
        yield X
        yield y
    except:
        return 0
```

```
[X, y] = Loadtxt('data_linear.txt')

# Phân chia dữ liệu tập Train - Test
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2)
```

(Ảnh code phần khởi tạo dữ liệu)

1.3.2.2 Khởi tạo mô hình và training

Ta sẽ train và so sánh mô hình với thư viện SKLearn

```
# Khớp mô hình LinearRegression
model = LinearRegression(epochs=1000000)
theta = model.fit(X_train, y_train)

# Đánh giá mô hình của tôi
y_pred = model.predict(X_test)
mse = mean_squared_error(y_pred, y_test)

# Khớp mô hình dùng thư viện scikit-learn
sklearn_model = SkLearnLN(fit_intercept=False) # fit_intercept = False for calculating the bias
sklearn_model.fit(X_train, y_train)

# Đánh giá mô hình khi dùng thư viện scikit-learn
sk_y_pred = sklearn_model.predict(X_test)
sk_mse = mean_squared_error(sk_y_pred, y_test)
```

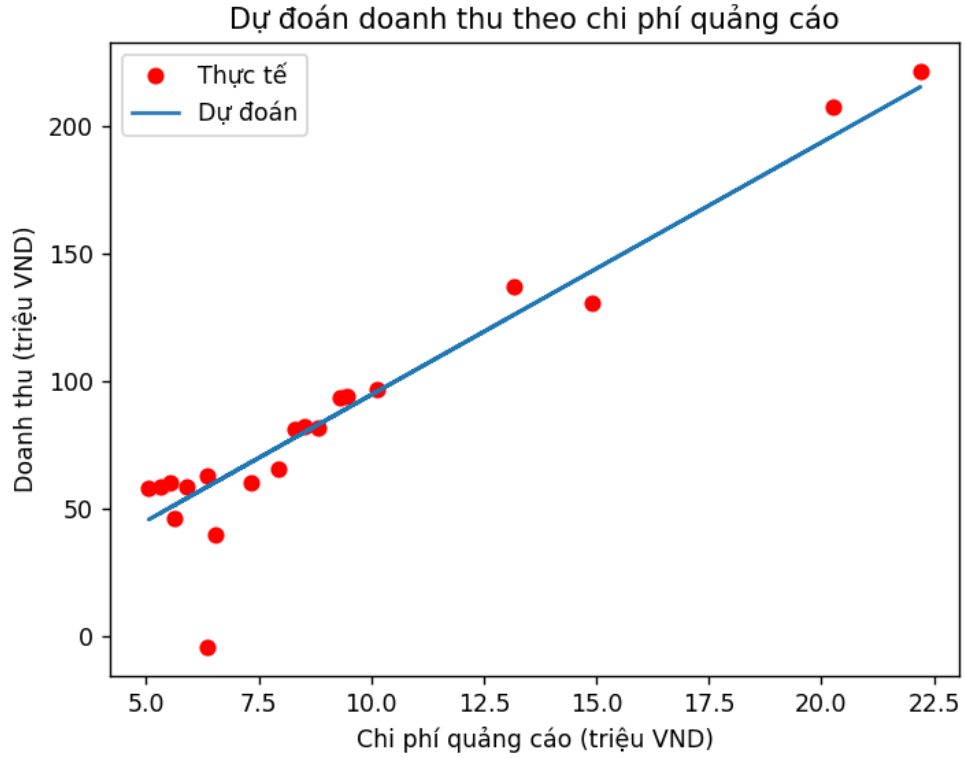
(Ảnh code train mô hình)

```python
# So sánh 2 kết quả thu được
print('Kết quả tìm bởi mô hình của tôi           ', theta)
print('Kết quả tìm bởi mô hình của scikit-learn: ', sklearn_model.coef_)

# So sánh giá trị hàm mất mát
print('Giá trị hàm mất mát khi dùng mô hình của tôi:    ', mse)
print('Giá trị hàm mất mát khi dùng scikit-learn:       ', sk_mse)

plt.plot(X_test[:, 1:], y_test, 'ro', label='Thực tế')
plt.plot(X_test[:, 1:], y_pred, label='Dự đoán')
plt.title('Dự đoán doanh thu theo chi phí quảng cáo')
plt.xlabel('Chi phí quảng cáo (triệu VND)')
plt.ylabel('Doanh thu (triệu VND)')
plt.legend()
plt.show()
```

(Ảnh code phần xuất ra kết quả thu được)

1.3.2.3 Kết quả thu được

```
100%|██████████| 200000/200000 [00:01<00:00, 192595.88it/s]
Kết quả tìm bởi mô hình của tôi           [-3.23411254  9.78438948]
Kết quả tìm bởi mô hình của scikit-learn:  [-3.23411254  9.78438948]
Độ lệch bình phương trung bình giữa giá trị dự đoán và giá trị thực tế khi dùng mô hình của tôi:
229.475410362559
Độ lệch bình phương trung bình giữa giá trị dự đoán và giá trị thực tế khi dùng scikit-learn:
229.4754103625587
```



(Ảnh kết quả thu được)

Nhận xét:

- Trọng số *w* sau khi được tối ưu bởi phương pháp của tôi cho kết quả tương đồng với khi dùng thư viện SKLearn.
- Lỗi trên tập test của SKLearn thấp hơn phương pháp của tôi một chút nhưng không quá đáng kể.
- Quan sát biểu đồ, ta thấy đường dự đoán cho kết quả khá sát với dữ liệu trong tập test. Tuy vậy giá trị hàm mất mát còn lớn do còn nhiều yếu tố khác Chi phí quảng cáo ảnh hưởng tới Doanh thu như: Tình hình biến động thị trường, Các sản phẩm, thương hiệu cạnh tranh, … Qua đó cho thấy sự chưa hiệu quả của mô hình Linear Regression!
- Ngoài ra, việc sang lọc dữ liệu đầu vào, thay đổi các giá trị của tốc độ học, giá trị khởi tạo của tham số cũng sẽ làm thay đổi tính hiệu quả của thuật toán.

1.4 Kết luận

Trên đây là những phân tích, triển khai, đánh giá về mô hình Linear Regression. Bên cạnh những hạn chế thì Hồi quy tuyến tính vẫn được sử dụng rộng rãi trong khoa học sinh học, hành vi và xã hội để mô tả các mối quan hệ có thể có giữa các biến và nó được xếp hạng là một trong những công cụ quan trọng nhất được sử dụng trong các ngành này.

2 Logistic Regression

2.1 Giới thiệu

Trong Linear Regression chúng ta thấy nhãn của chúng ta là một **biến liên tục** (giá, chiều cao, tuổi, khoảng cách, …) và vì thế nên đầu ra của chúng ta sẽ là một số thực cụ thể. Tuy nhiên trên thực tế có những bài toán yêu cầu đầu ra là **dạng phân loại** (có hay không, nam hay nữ, thắng hay không thắng, …) tức là các lớp cụ thể. Đây được gọi là bài toán **phân loại** dựa trên dữ liệu đầu vào. Trong trường hợp số lượng class đầu ra của mô hình phân loại là 2 lớp thì chúng ta có bài toán **phân lớp nhị phân**. Logistic Regression chính là một mô hình cơ bản được sinh ra để giải quyết bài toán phân lớp nhị phân đó. Đầu ra của mô hình được thể hiện dưới dạng xác suất và bị chặn trong khoảng (0,1).

Ví dụ dự đoán xác suất thi đỗ

Một nhóm sinh viên dành từ 0 đến 7 giờ để học cho một kỳ thi. Làm thế nào để số giờ học ảnh hưởng đến xác suất của sinh viên vượt qua kỳ thi?

Bảng hiển thị số giờ mỗi học sinh dành cho việc học và liệu họ đạt (1) hay không đạt (0).

Số giờ học	Kết quả	Số giờ học	Kết quả
0,5	0	3,25	1
0,75	0	3,5	0
1,00	0	4,00	1
1,25	0	4,25	1
1,75	1	4,5	1
2,00	0	4,75	1
2,25	1	5,00	1
2,5	0	5,50	1
2,75	1	1,50	0
3,00	0	1,75	0

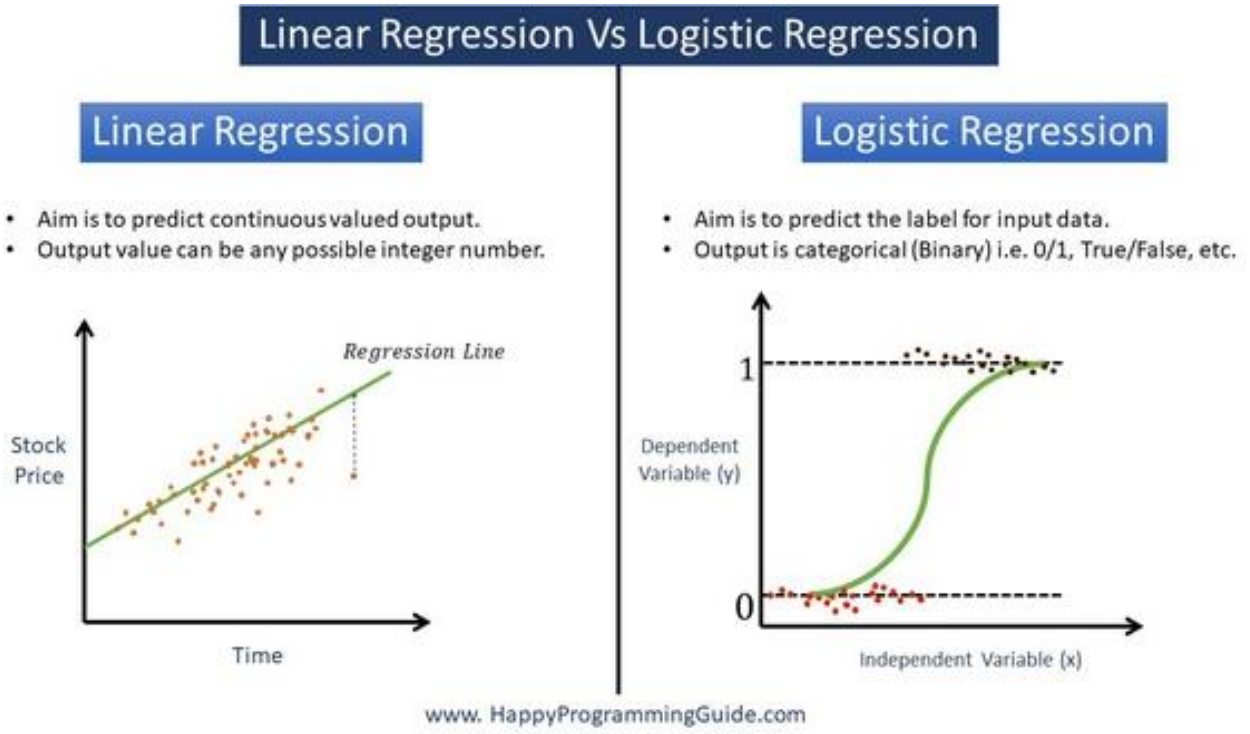
Ví dụ này thích hợp để ta dùng Logistic Regression với một biến x (số giờ học) và biến phân loại y (kết quả bài kiểm tra) gồm hai loại: đạt (1) hoặc không đạt (0)

Nếu như trong Linear regression, *f(s) = s* là activation function và sử dụng tích vô hướng $w^T x$ để dự đoán kết quả thì trong Logistic regression, để đưa đầu ra về trong khoảng (0,1) - thể hiện xác suất của đầu ra trên mỗi class, ta sử dụng hàm sigmoid làm activation function. Khi đó đầu ra dự đoán của Logistic regression có dạng:

$$f(s) = \frac{1}{1+e^{-s}}$$

trong đó $s = w^T x$

Nếu đầu ra sau khi đi qua hàm sigmoid lớn hơn một **ngưỡng cứng** nhất định thì chúng ta sẽ gắn giá trị cho class là 1, ngược lại sẽ là 0. Minh hoạ đầu ra của hàm sigmoid trong hình sau.



2.2 Phân tích toán học

2.2.1 Binary Cross Entropy Loss (BCE)

Giả sử xác suất dữ liệu *x* rơi vào lớp 1 là $\hat{y}$. Khi đó xác suất dữ liệu *x* rơi vào lớp 0 là 1 - $\hat{y}$

Ta có biểu thức xác suất cho 1 điểm dữ liệu x_i là:

$$P(y_i|x_i; w) = \hat{y}(1 - \hat{y})^{1-y_i}$$

Ta luôn muốn mô hình gần nhất với dữ liệu đầu vào hay xác suất trên đạt giá trị cao nhất. Khi đó ta cần tìm *w* để biểu thức $P(y|X; w)$ đạt giá trị lớn nhất.

Để tránh kết quả là quá nhỏ ta lấy logarit biến phép nhân thành phép cộng và lấy ngược dấu để đưa bài toán thành tìm giá trị nhỏ nhất của hàm mất mát. Ta thu được

$$J(w) = -\log P(y|X; w) = -\sum_{i=1}^{N}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$$

trong đó $\hat{y} = f(w^T x)$

Biểu thức vế phải có tên gọi là **cross entropy**, thường được sử dụng để đo **khoảng cách** giữa hai phân phối. Trong bài toán đang xét, một phân phối là dữ liệu được cho, với xác suất chỉ là 0 hoặc 1; phân phối còn lại được tính theo mô hình logistic regression. Khi khoảng cách giữa hai phân phối nhỏ đồng nghĩa với việc hai phân phối đó rất gần nhau.

Như vậy, giống như trong **Linear Regression**, ta cần phải cập nhật cho tham số *w*. Tuy nhiên **Logistic regression** có một điểm khác, do label của ta là một số cụ thể tức là class 0 hay class 1 thế nên chúng ta không nên sử dụng **Mean Square Error** để làm hàm mất mát

được mà cần phải có một hàm mất mát khác. Thay vào đó chúng ta sử dụng **Binary Cross Entropy Loss** để làm hàm loss cho bài toán này.

$$BCE(yi, \hat{y}_i) = \frac{-1}{N}\sum_{i=1}^{N}(y_i\, log(\,\hat{y}_i) + (1 - y_i) log(\,1 - \hat{y}_i))$$

2.2.2 Tối ưu hàm mất mát

Ta mong muốn **cực tiểu hoá giá trị** của hàm loss BCE nhưng trong công thức có chứa hàm sigmoid với trọng số *w*, nên muốn tính được đạo hàm theo *w* thì sẽ cần phải áp dụng đạo hàm của hàm hợp.

Bước 1: Tính $\frac{\partial J}{\partial \hat{y}}$.

Ta xét hàm mất mát với chỉ một điểm dữ liệu: $J = -y.log(\hat{y}) - (1-y).log(1-\hat{y})$

Lấy đạo hàm logarit ta được:

$$\frac{\partial J}{\partial \hat{y}} = [-y.log(\hat{y})]'_{\hat{y}} + [-(1-y).log(1-\hat{y})]'_{\hat{y}} = \frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

Bước 2: Tính $\frac{\partial J}{\partial s}$.

Ta có: $y = f(s) = \frac{1}{1+e^{-s}}$

Lấy đạo hàm $\hat{y}$ theo s ta được:

$$\frac{\partial \hat{y}}{\partial s} = \left[\frac{1}{1+e^{-s}}\right]'_s = \frac{e^{-s}}{(1+e^{-s})^2} = \frac{1}{1+e^{-s}} * \frac{(1+e^{-s}) - 1}{1+e^{-s}} = \hat{y} * (1-\hat{y})$$

Tổng hợp bước 1 và 2 ta được:

$$\frac{\partial J}{\partial s} = \frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial s} = -y(1-\hat{y}) + \hat{y}(1-y) = \hat{y} - y$$

Bước 3: Tính $\frac{\partial J}{\partial w}$.

Ta có: $s = w^T x$

Lấy đạo hàm *s* theo *w* ta được: $\frac{\partial s}{\partial w} = x_i$

Tổng hợp 3 bước trên ta được: $\frac{\partial J}{\partial w} = \frac{\partial J}{\partial \hat{y}}.\frac{\partial \hat{y}}{\partial s}.\frac{\partial s}{\partial w} = x_i.(\hat{y} - y)$

Xét với toàn tập điểm dữ liệu: $\frac{\partial J(w)}{\partial w} = \frac{2}{N}\bar{X}^T(\bar{X}w - y)$

2.3 Triển khai với Numpy

2.3.1 Cài đặt mô hình Logistic Regression

```python
class LogisticRegression:
    def __init__(self, lr=0.01, epochs=1000):
        self.epochs = epochs
        self.lr = lr
        self.w = 0

    def initialize(self, n_features):
        self.w = np.zeros((n_features))

    def sigmoid(self, s):
        return 1 / (1 + np.exp(-s))

    #   Hàm tính độ dốc J'(w)
    def gradient(self, X, y, n_samples):
        y_pred = self.sigmoid(np.dot(X, self.w))
        d_w = np.dot(X.T, y_pred - y) / n_samples
        return d_w

    def fit(self, X, y):
        # Lấy khuôn dạng X (số hàng, số cột)
        n_samples, n_features = X.shape
        # Khởi tạo tham số
        self.initialize(n_features)
        # TÍnh đạo hàm sau mỗi bước
        for _ in tqdm(range(self.epochs)):
            # Cập nhật tham số
            d_w = self.gradient(X, y, n_samples)
            self.w -= self.lr * d_w
        return self.w

    # Hàm dự đoán kết quả
    def predict(self, X):
        return [1 if i > 0.5 else 0 for i in self.sigmoid(np.dot(X, self.w))]
```

(Ảnh code class Logistic Regression)

Cập nhật đạo hàm *J'(w)* theo giải thuật Gradient Descent đã được trình bày ở mục 1.2.3, trong đó hàm gradient được tính toán theo công thức:

$$\frac{\partial J(w)}{\partial w} = \frac{2}{N}\bar{X}^T(\bar{X}w - y)$$

2.3.2 Chạy thử

2.3.2.1 Nạp dữ liệu

```
# Hàm nạp dữ liệu
def Loadtxt(path):
    try:
        raw = np.loadtxt(path,delimiter = ',')
        X = np.zeros((np.size(raw,0),np.size(raw,1)))
        X[:,0] = 1
        X[:,1:] = raw[:,:-1]
        y = raw[:,-1]
        yield X
        yield y
    except:
        return 0
```

```
[X, Y] = Loadtxt('data_logistic.txt')
```

(Ảnh code load dữ liệu)

2.3.2.2 Training và so sánh với SkLearn

```
# Khớp mô hình LogisticRegression
model = LogisticRegression(lr=0.1, epochs=200000)
theta = model.fit(X, Y)

# Đánh giá mô hình của tôi
Y_pred = model.predict(X)
print(classification_report(Y, Y_pred))
bce = log_loss(Y, Y_pred)

# Khớp mô hình dùng thư viện scikit-learn
sklearn_model = SkLearnLG(penalty=None, fit_intercept=False)
sklearn_model.fit(X, Y)

# Đánh giá mô hình khi dùng thư viện scikit-learn
sk_y_pred = sklearn_model.predict(X)
print(classification_report(Y, sk_y_pred))
sk_bce = log_loss(Y, sk_y_pred)
```

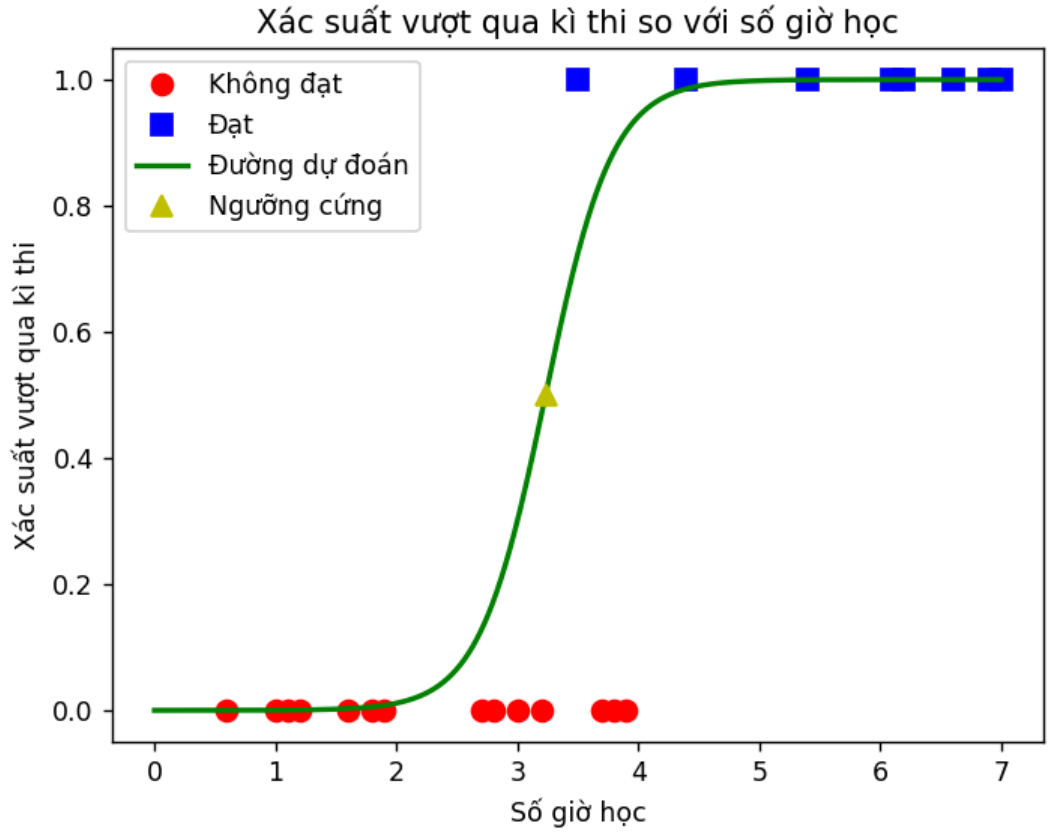
(Ảnh code train dữ liệu)

```
# So sánh 2 kết quả thu được
print('Kết quả tìm bởi mô hình của tôi           ', theta)
print('Kết quả tìm bởi mô hình của scikit-learn: ', sklearn_model.coef_)

# So sánh giá trị hàm mất mát
print('Giá trị hàm mất mát khi dùng mô hình của tôi:    ', bce)
print('Giá trị hàm mất mát khi dùng scikit-learn:       ', sk_bce)

# Vẽ biểu đồ
X0 = X.T[1, np.where(Y == 0)][0]
y0 = Y[np.where(Y == 0)]
X1 = X.T[1, np.where(Y == 1)][0]
y1 = Y[np.where(Y == 1)]

plt.plot(X0, y0, 'ro', markersize = 8, label = 'Không đạt')
plt.plot(X1, y1, 'bs', markersize = 8, label = 'Đạt')

xx = np.linspace(0, 6, 1000)
w0 = theta[0]
w1 = theta[1]
threshold = -w0/w1
yy = LogisticRegression().sigmoid(w0 + w1*xx)

plt.plot( xx, yy, 'g-', linewidth = 2, label = 'Đường dự đoán')
plt.plot(threshold, .5, 'y^', markersize = 8, label = 'Ngưỡng cứng')
plt.xlabel('Số giờ học')
plt.ylabel('Xác suất vượt qua kì thi')
plt.title('Xác suất vượt qua kì thi so với số giờ học')
plt.legend()
plt.show()
```

(Ảnh code xuất ra kết quả)

2.3.2.3 Kết quả thu được

```
100%|| 200000/200000 [00:02<00:00, 81379.60it/s]
Đánh giá mô hình của tôi
              precision    recall  f1-score   support

         0.0       1.00      0.83      0.91        12
         1.0       0.83      1.00      0.91        10

    accuracy                           0.91        22
   macro avg       0.92      0.92      0.91        22
weighted avg       0.92      0.91      0.91        22

Đánh giá mô hình khi dùng SkLearn
              precision    recall  f1-score   support

         0.0       1.00      0.83      0.91        12
         1.0       0.83      1.00      0.91        10

    accuracy                           0.91        22
   macro avg       0.92      0.92      0.91        22
weighted avg       0.92      0.91      0.91        22

Kết quả tìm bởi mô hình của tôi            [-10.37984502   3.11604401]
Kết quả tìm bởi mô hình của scikit-learn:  [[-10.37984281   3.11604333]]
Giá trị hàm mất mát khi dùng mô hình của tôi:     3.276695762647014
Giá trị hàm mất mát khi dùng scikit-learn:        3.276695762647014
```



(Ảnh kết quả thuật toán)

Nhận xét:

- Trọng số *w* và giá trị hàm mất mát sau khi được tối ưu bởi phương pháp của tôi cho kết quả tương đồng với khi dùng thư viện SKLearn.
- Quan sát **classification_report**:

  - $\boldsymbol{Precision = \frac{True\ positive}{True\ positive + False\ positive}}$

  Xác định tỷ lệ số dự đoán 1 ("Đỗ") thực sự đúng

  - $\boldsymbol{Recall = \frac{True\ positive}{True\ positive + False\ Negative}}$

  Xác định tỷ lệ số dự đoán 1 ("Đỗ") trên toàn bộ các mẫu thuộc nhóm 1 ("Đỗ")

  - $\boldsymbol{F1\text{-}score = \frac{2 * precision * recall}{(precision + recall)}}$

  Kết hợp cả recall và precision, giá trị điểm F1 càng gần 1 thì hiệu suất mong đợi càng cao

  - $\boldsymbol{Accuracy = \frac{True\ positive + True\ Negative}{support\_0 + support\_1}}$
  - $\boldsymbol{Macro\ avg = \frac{f1\_0 + f1\_1}{2}}$

  Xác định tỷ lệ trung bình không có trọng số: trả về giá trị trung bình mà không xem xét tỷ lệ cho từng nhãn trong tập dữ liệu.

  - $\boldsymbol{Weighted\ avg = \frac{f1\_0 * support_0 + f1\_1 * support\_1}{support\_0 + support\_1}}$

  Xác định tỷ lệ trung bình có trọng số: trả về giá trị trung bình khi xem xét tỷ lệ cho từng nhãn trong tập dữ liệu.

  Nhận thấy độ chính xác đạt khoảng **92%**

- Điểm hạn chế của Logistic Regression là nó yêu cầu các điểm dữ liệu được tạo ra một cách *độc lập* với nhau. Trên thực tế, các điểm dữ liệu có thể bị *ảnh hưởng* bởi nhau. Ví dụ: có một nhóm ôn tập với nhau trong 4 giờ, cả nhóm đều thi đỗ (giả sử các bạn này học rất tập trung), nhưng có một sinh viên học một mình cũng trong 4 giờ thì xác suất thi đỗ thấp hơn.

2.4 Kết luận

Logistic Regression mà đặc biệt là Binary Logistic Regression là mô hình phổ biến trong nghiên cứu dùng để ước lượng xác suất một sự kiện sẽ xảy ra. Mô hình được ứng dụng rất mạnh trong việc dự đoán và là một trong những giải pháp hồi quy thông dụng nhất hiện nay!

KẾT LUẬN

- **Các loại biến hay loại dữ liệu của biến mục tiêu** chính là cơ sở chọn lựa phương pháp hồi quy tương ứng.
  - Với biến mục tiêu là **biến định lượng liên tục** thì phương pháp hồi quy đầu tiên mà chúng ta đã tìm hiểu qua chính là hồi quy tuyến tính – Linear regression, ngoài ra còn có các mô hình hồi quy phi tuyến khác.
  - Với biến mục tiêu là **biến định tính**, hay biến thay phiên (hoặc biến rời rạc) thì phương pháp hồi quy chủ yếu được dùng là Logistic regression, với biến thay phiên chúng ta có phương pháp Binary Logistic regression.

⇨ Có thể thấy Logistic regression vận dụng cho những biến tiềm năng không phải là biến định lượng liên tục, là cơ sở chính chỉ ra sự khác biệt giữa hai phương pháp hồi quy này!

- Điểm khác biệt thứ 2 giữa logistic regression và linear regression, chính là **kết quả của biến mục tiêu y** trong linear regression (hay các dạng hồi quy áp dụng cho biến mục tiêu là biến định lượng liên tục) là **giá trị số** (numerical value) còn kết quả dự báo của biến mục tiêu y trong logistic regression sẽ mang **giá trị xác suất** (probability) để phân loại đối tượng nghiên cứu hay quyết định giá trị cuối cùng của biến y trong danh mục các giá trị định tính.

- Điểm khác biệt thứ 3 là đối với các dạng hồi quy áp dụng cho biến mục tiêu là biến định lượng thì nhiệm vụ phân tích sau cùng sẽ là đưa ra **kết quả dự báo chính xác** (value prediction) còn hồi quy logistic sau cùng có cả **kết quả phân loại chính xác** (category classification). Ví dụ sau khi tính toán, phân loại được khách hàng A sẽ được cấp phát thẻ tín dụng khi giá trị y được dự báo = 1.

Qua những kiến thức tổng hợp về hai phương pháp hồi qui Logisstic Regression và Linear Regression, tôi đã có cái nhìn rõ hơn, chi tiết và là bước mở đầu cho tôi tìm hiểu thêm các mô hình khác trong Machine Learning cũng như tìm hiểu them về ý nghĩa, tính ứng dụng của các mô hình đó trong đời sống thực tiễn!

Tài liệu tham khảo

1. https://en.wikipedia.org/wiki/Linear_regression
2. https://scikit-learn.org/stable/modules/linear_model.html
3. https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error
4. https://en.wikipedia.org/wiki/Logistic_regression
5. https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
6. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
7. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html
8. https://machinelearningcoban.com/2017/01/27/logisticregression/
9. https://machinelearningcoban.com/2017/01/27/logisticregression/
10. https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/