ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM
KHOA CÔNG NGHỆ THÔNG TIN

THỐNG KÊ MÁY TÍNH VÀ ỨNG DỤNG - CQ2018-22

BÁO CÁO ĐỒ ÁN THỰC HÀNH
ĐỒ ÁN 2: SỬ DỤNG LINEAR REGESSION TRÊN TẬP DỮ LIỆU MNIST

Thông tin nhóm

STT	MSSV	Họ tên	Email
1	18120009	Vương Gia Bảo	18120009@student.hcmus.edu.vn
2	18120374	Nguyễn Minh Hiếu	18120374@student.hcmus.edu.vn

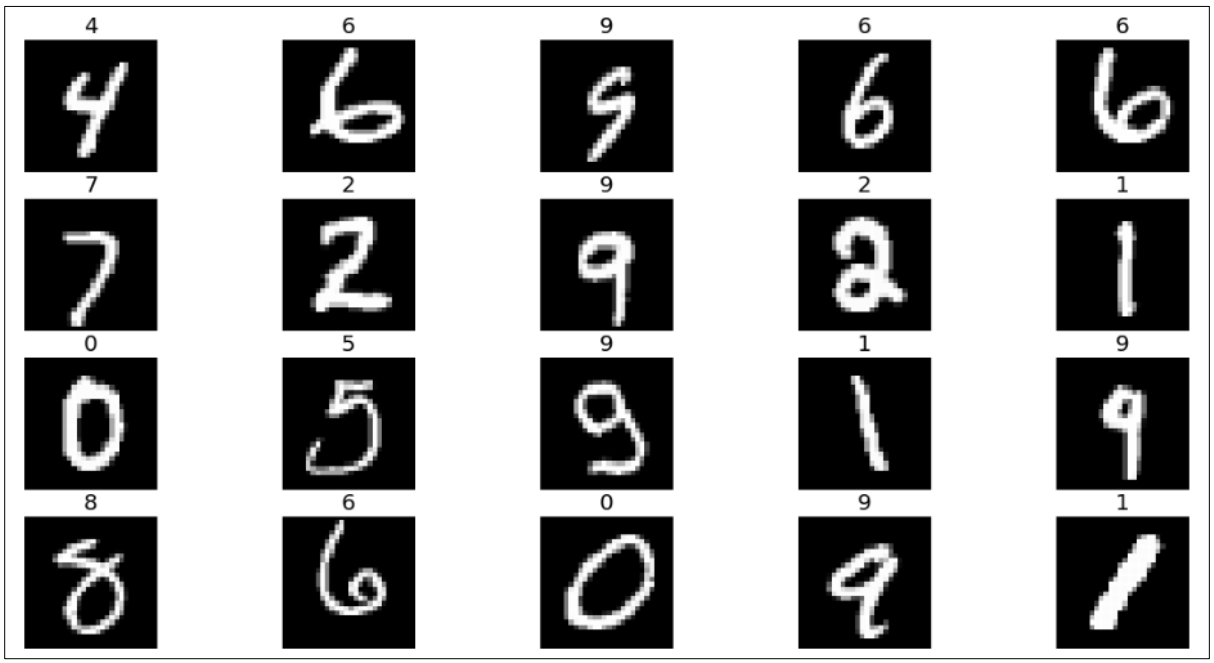
Phân công công việc

STT	MSSV	Họ tên	Nội dung công việc	Hoàn thành
1	18120009	Vương Gia Bảo	Tiền xử lý, chuẩn hóa dữ liệu, thử nghiệm các model khác nhau để chọn ra model cuối cùng, viết báo cáo mục 2	100%
2	18120374	Nguyễn Minh Hiếu	Viết chương trình cho phép sử dụng chuột để viết một chữ số để thực hiện dự đoán. Tìm hiểu lý thuyết, tổng hợp báo cáo.	100%

Contents

1. Tập dữ liệu

MNIST là tập dữ liệu bao gồm các chữ số từ 0-9 viết tay.



- Tập dữ liệu bao gồm tập Train (60000 ảnh) và Test (10000 ảnh).
- Mỗi ảnh là ảnh xám (1 channel) và kích thước 28x28.
- Giá trị mỗi pixels trong ản nằm trong đoạn [0:255], càng nhỏ thì càng tối và ngược lại.
- Dữ liệu có dáng chuẩn, không bị xoay ngang dọc.

2. Tiền xử lý

2.1. Cách biến đổi dữ liệu đầu ra

- Thử dự đoán với 5 dòng dữ liệu đầu tiên, xem kết quả dự đoán và vectơ xác suất
- Dòng đầu tiên dự đoán là số 6 nên xác suất ở cột 6 là lớn nhất, và tương tự đối với các dòng còn lại.

```
logreg_model.predict(X_train[:5])
```

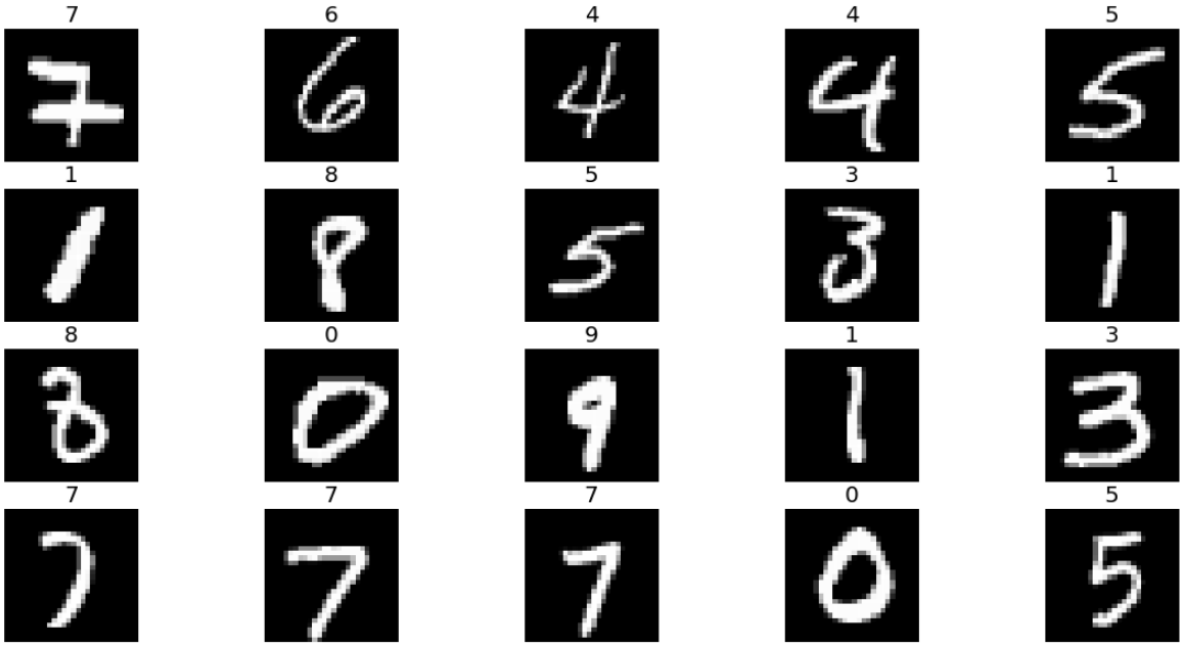
```
array(['6', '1', '6', '5', '5'], dtype=object)
```

```
pd.DataFrame(logreg_model.predict_proba(X_train[:5]))
```

	0	1	2	3	4	5	6	7	8	9
0	4.595096e-07	1.068848e-12	0.000411	7.336572e-07	4.337255e-06	0.000006	0.999574	9.775111e-12	0.000003	9.618304e-07
1	6.687473e-11	9.995744e-01	0.000016	2.990997e-04	5.582501e-08	0.000013	0.000005	1.891784e-07	0.000074	1.758373e-05
2	1.069751e-06	3.876379e-09	0.000157	4.131085e-05	5.030343e-04	0.000334	0.998925	4.091316e-09	0.000038	3.388586e-07
3	2.399429e-04	1.764806e-07	0.007721	2.418825e-02	1.300587e-05	0.866058	0.000002	1.572215e-10	0.101757	2.086752e-05
4	9.189433e-08	1.441968e-04	0.086461	5.580044e-02	8.151391e-03	0.555151	0.107789	9.219598e-12	0.186503	2.056157e-08

2.2. Loại bỏ điểm ảnh ở biên

Do liệu ảnh ban đầu

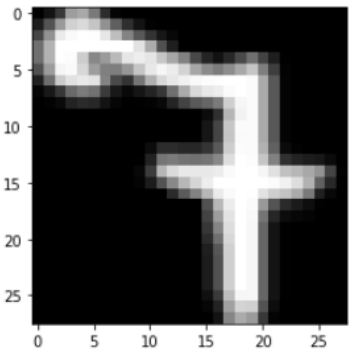


Khi quan sát một số mẫu, một số điểm ảnh có thể không có ý nghĩa trong việc phân loại. Chẳng hạn, các điểm ảnh ở biên luôn mang giá trị 0. Khi đó, ta loại bỏ các điểm ảnh như vậy ra khỏi mô hình

```
# Loại bỏ các điểm ảnh ở biên (giá trị 0)
def crop(data):
    data = data.reshape(28,28)
    r = data[~np.all(data == 0, axis=1)] # loại bỏ hàng toàn 0
    idx = np.argwhere(np.all(r[..., :] == 0, axis=0)) # vị trí cột toàn 0
    c = np.delete(r, idx, axis=1) # loại bỏ cột toàn 0
    res = cv2.resize(c, dsize=(28, 28)) # Trả lại về size 28x28
    res = res.flatten() # Chuyển lại thành array 1 chiều như ban đầu
    return res
```

Quan sát ảnh sau khi xử lý

```
# Xem thử một ảnh sau khi xử lý
data = X_train[0]
res = crop(data)
plt.imshow(res.reshape(28,28), cmap=plt.cm.gray);
```



Kết quả khi train dữ liệu đã qua bước loại bỏ các điểm ảnh ở biên đã tốt hơn khi chưa tiền xử lý.

```
X_train_crop = np.apply_along_axis(crop, 1, X_train)
X_test_crop = np.apply_along_axis(crop, 1, X_val)
```

```
logreg_crop_model = LogisticRegression(solver='lbfgs').fit(X_train_crop, y_train)
preds_train_crop = logreg_crop_model.predict(X_train_crop)
preds_test_crop = logreg_crop_model.predict(X_test_crop)

# Evaluation
print('Train Error (before): {} %'.format(np.mean(preds_train != y_train)*100))
print('Test Error (before): {} %\n'.format(np.mean(preds_test != y_val)*100))

print('Train Error: {} %'.format(np.mean(preds_train_crop != y_train)*100))
print('Test Error: {} %\n'.format(np.mean(preds_test_crop != y_val)*100))
```

```
/home/bao/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs fail
ed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Train Error (before): 6.4079999999999995 %
Test Error (before): 8.260000000000002 %

Train Error: 5.6579999999999995 %
Test Error: 6.63 %
```

2.3. Đa cộng tuyến

Định nghĩa: Đa cộng tuyến là hiện tượng các biến độc lập có mối tương quan cao với nhau (có mối quan hệ tuyến tính)

Hậu quả: Mô hình hồi quy xảy ra hiện tượng đa cộng tuyến sẽ khiến nhiều chỉ số bị sai lệch, dẫn đến kết quả của việc phân tích định lượng không còn mang lại nhiều ý nghĩa.

Cách xử lý: Dùng phương pháp Principal Component Analysis – PCA (Phân tích thành phần chính). Theo cách này, chúng ta sẽ tạo ra các principal components để thay thế cho các biến độc lập. Cái hay của phương pháp này là nó sẽ giúp loại bỏ hoàn toàn hiện tượng đa cộng tuyến vì các principal component được đảm bảo không có mối quan hệ với nhau

2.4. Giảm chiều dữ liệu bằng PCA

PCA để tăng tốc độ phân loại Chữ số Viết tay

Huấn luyện bằng model Logistic Regression trên tập dữ liệu thô ban đầu

```
start = time.time()
logreg_model = LogisticRegression(solver='lbfgs').fit(X_train, y_train)
end = time.time()

full_logreg_time = end-start
print('Time to fit: {}s'.format(full_logreg_time))

preds_train = logreg_model.predict(X_train)
preds_test = logreg_model.predict(X_val)

# Evaluation
print('Train Error: {} %'.format(np.mean(preds_train != y_train)*100))
print('Test Error: {} %'.format(np.mean(preds_test != y_val)*100))
```

```
/home/bao/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs fail
ed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Time to fit: 40.32598543167114s
Train Error: 6.4079999999999995 %
Test Error: 8.260000000000002 %
```

Confusion matrix

```
pd.crosstab(y_val, preds_test, margins=True, rownames=['Actual'], colnames=['Predicted'])
```

Predicted	0	1	2	3	4	5	6	7	8	9	All
Actual											
0	981	0	3	4	3	7	8	2	10	1	1019
1	0	1092	1	3	2	7	2	3	12	2	1124
2	12	10	886	27	8	4	12	15	27	2	1003
3	5	6	16	890	1	36	3	10	24	7	998
4	1	8	5	3	878	1	11	5	10	43	965
5	6	6	3	27	10	815	15	1	39	6	928
6	7	4	8	1	9	16	966	4	5	0	1020
7	1	5	8	5	8	2	1	942	4	33	1009
8	8	9	10	21	3	32	5	5	851	16	960
9	4	6	3	16	25	10	0	29	8	873	974
All	1025	1146	943	997	947	930	1023	1016	990	983	10000

Dùng PCA

```
start = time.time()
logreg_model_pca = LogisticRegression(solver='lbfgs').fit(x_train_pca, y_train)
end = time.time()

print('Time to fit model (100 PCs): {}s'.format(end-start))
print('Time to fit model (full dataset): {}s\n'.format(full_logreg_time))

preds_train_pca = logreg_model_pca.predict(x_train_pca)
preds_test_pca = logreg_model_pca.predict(x_test_pca)

# Evaluation
print('Train Error (100 PCs): {} %'.format(np.mean(preds_train_pca != y_train)*100))
print('Test Error (100 PCs): {} %\n'.format(np.mean(preds_test_pca != y_val)*100))

print('Train Error (full dataset): {} %'.format(np.mean(preds_train != y_train)*100))
print('Test Error (full dataset): {} %'.format(np.mean(preds_test != y_val)*100))

fig, ax = plt.subplots(figsize=(10, 5))
ax.bar(0, full_logreg_time, width=0.5)
ax.bar(1, end-start, width=0.5)
ax.set_xlabel('Model')
ax.set_xticks([0,1])
ax.set_xticklabels(['Full Dataset', '100 PCs'])
ax.set_ylabel('Time to Fit Model (s)')
ax.set_title('Time taken to fit different models (s)');
```

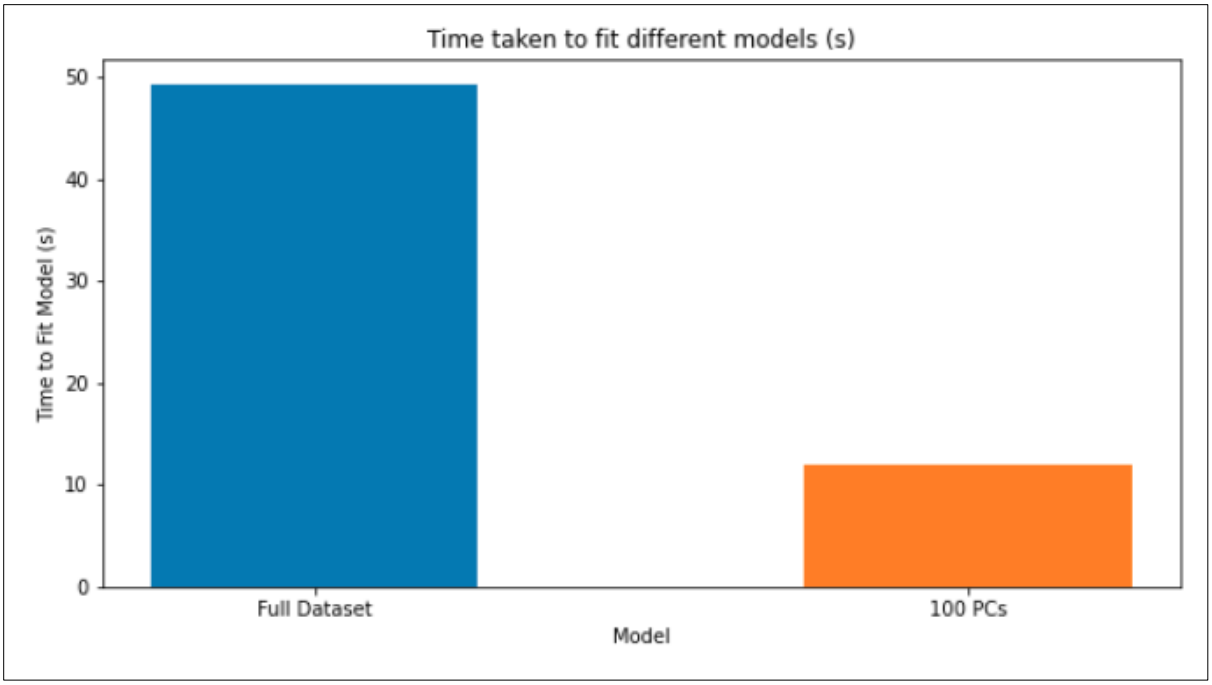
```
/home/bao/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:763: ConvergenceWarning: lbfgs fail
ed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
Time to fit model (100 PCs): 12.028300762176514s
Time to fit model (full dataset): 49.24395036697388s

Train Error (100 PCs): 8.352 %
Test Error (100 PCs): 8.39 %

Train Error (full dataset): 6.4079999999999995 %
Test Error (full dataset): 8.260000000000002 %
```



Confusion matrix

```
pd.crosstab(y_val, preds_test_pca, margins=True, rownames=['Actual'], colnames=['Predicted'])
```

Predicted / Actual	0	1	2	3	4	5	6	7	8	9	All
0	985	0	3	1	5	11	4	0	8	2	1019
1	0	1087	2	4	2	7	2	4	15	1	1124
2	16	14	883	18	14	4	16	16	19	3	1003
3	6	10	15	886	1	32	7	10	19	12	998
4	2	9	5	1	890	0	14	2	6	36	965
5	14	6	7	19	11	822	14	1	26	8	928
6	6	6	3	0	8	10	981	0	6	0	1020
7	2	14	7	3	9	2	1	940	1	30	1009
8	9	21	7	35	8	24	6	8	817	25	960
9	9	8	0	13	24	12	0	32	6	870	974
All	1049	1175	932	980	972	924	1045	1013	923	987	10000

Công dụng của PCA

- Đẩy nhanh quá trình đào tạo mà không làm giảm đáng kể khả năng dự đoán của mô hình
- Giảm đa cộng tuyến và do đó có thể cải thiện thời gian tính toán của các mô hình phù hợp.
- Giảm kích thước trong cài đặt

2.5. Chuẩn hóa dữ liệu

Sử dụng StandardScaler của Scikit-Learn để tăng tốc độ hội tụ. Tạo một pipeline gồm hai bước cropData và StandardScaler. Tăng max_iter lên 5000 và dự đoán trên tập train (train + val). Ta không còn thấy thông báo ConvergenceWarning, chứng tỏ thuật toán đã hội tụ.

```
# Loại bỏ các điểm ảnh ở biên (giá trị 0)
def crop(data):
    data = data.reshape(28,28)
    r = data[~np.all(data == 0, axis=1)] # loại bỏ hàng toàn 0
    idx = np.argwhere(np.all(r[..., :] == 0, axis=0)) # vị trí cột toàn 0
    c = np.delete(r, idx, axis=1) # loại bỏ cột toàn 0
    res = cv2.resize(c, dsize=(28, 28)) # Trả lại về size 28x28
    res = res.flatten() # Chuyển lại thành array 1 chiều như ban đầu
    return res
```

```
def cropData(data):
    return np.apply_along_axis(crop, 1, data)
```

```
preprocess_pipeline = make_pipeline(FunctionTransformer(cropData),
                                    StandardScaler())
```

```
X_train_crop = preprocess_pipeline.fit_transform(X_train)
X_test_crop = preprocess_pipeline.fit_transform(X_test)
```

```
start = time.time()
logreg_crop_model = LogisticRegression(solver='lbfgs',max_iter= 5000).fit(X_train_crop, y_train)
end = time.time()

print('Time to fit model: {}s'.format(end-start))
```

```
Time to fit model: 669.5262906551361s
```

Kết quả cuối cùng

```
# save the model to disk
import pickle
filename = 'final_model.sav'
pickle.dump(logreg_crop_model, open(filename, 'wb'))
```

```
# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))

preds_train_crop = loaded_model.predict(X_train_crop)
preds_test_crop = loaded_model.predict(X_test_crop)

# Evaluation
print('Train Error: {} %'.format(np.mean(preds_train_crop != y_train)*100))
print('Test Error: {} %\n'.format(np.mean(preds_test_crop != y_test)*100))
```

```
Train Error: 4.35 %
Test Error: 6.78 %
```

3. Áp dụng Logistic Regression

Phần trước ta đã thực hiện vài biến đổi trên các ảnh, tại đây các kết quả đấy sẽ là input cho mô hình nhằm giúp mô hình học được và gán nhãn chính xác.

Logistic Regression thuộc mô hình học máy có giám sát, là mô hình giúp nhận biết được dữ liệu đã cho có "nằm ở" lớp đó hay không.

Các thành phần:

- Các features của đầu vào **X.** Với mỗi $\mathbf{x^i} = [x_1, x_{2,...,} x_n]$
- Hàm phân lớp để tính $\hat{y}$ = **P(y|x)**. Thường thì người ta dùng sẽ dùng **sigmoid** cho 2 lớp và **softmax** cho nhiều lớp.
- Hàm đặc biệt để tính toán tham số phù hợp cho mô hình, mục tiêu là cần phải giảm thiểu lỗi nhiều nhất có thể khi huấn luyện. Ở đây sẽ giới thiệu về **cross-entrophy loss function**.
- Cách tối ưu hóa hàm trên.

Vậy tổng thể cần 2 bước:

- **Training:** Xác định tham số **w** và bias thích hợp cho mô hình đựa vào cross-entropy loss function và tối hưu hàm đó.
- **Test:** Đưa một vào **x** để kiểm tra. Ta sẽ tính P(y|**x**) là trả về giá trị **P** cao hơn với y = 1 hoặc y = 0.

a. Sigmoid

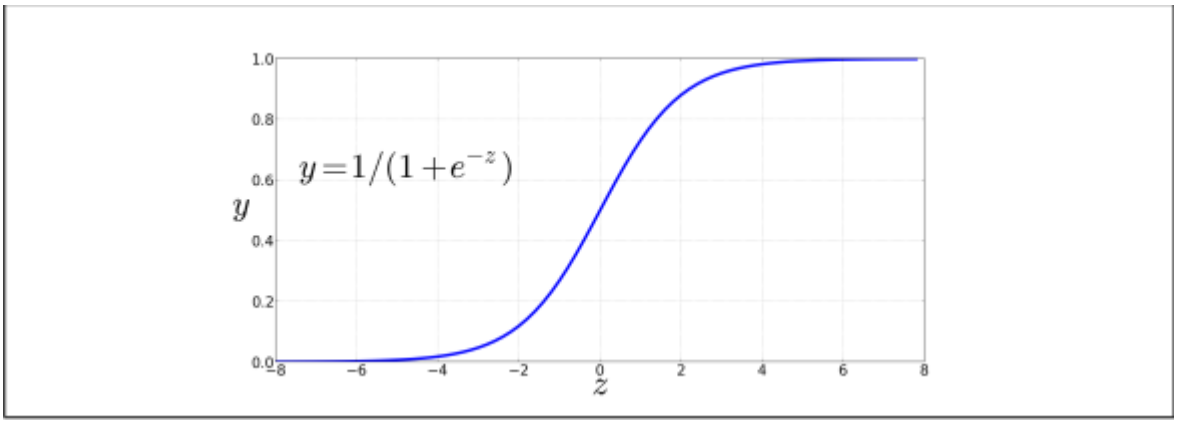
Giả sử ta có một vector feature **x** = $[x_1, x_{2, ..., } x_n]$. Giờ ta muốn xác định P(y=1|**x**) và P(y=0|**x**). Giống như là xác định 1 comment có phải tích cực hay không. Logistic Regression sẽ sử dụng một vector gồm bias và weight. Weight đại diện cho mức quan trọng của một feature còn bias là tham số giúp phân lớp chính xác hơn. Cuối cùng ta cần tính:

$$\boldsymbol{z} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{wixi} + b$$

hay

$$z = \mathbf{w} \cdot \mathbf{x} + b$$

Sau đó đưa z vào hàm sigmoid, hàm này biến mọi giá trị x số thực thành giá trị trong đoạn [0,1]



Sigmoid function:

$$y = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+\exp(-z)}$$

Giá trị của y có thể dùng làm xác suất:

$$\begin{aligned} P(y=1) &= \sigma(w \cdot x + b) \\ &= \frac{1}{1+\exp(-(w \cdot x + b))} \\ P(y=0) &= 1 - \sigma(w \cdot x + b) \\ &= 1 - \frac{1}{1+\exp(-(w \cdot x + b))} \\ &= \frac{\exp(-(w \cdot x + b))}{1+\exp(-(w \cdot x + b))} \end{aligned}$$

Giờ ta tính được P(y = 1 | **x**). Ta sẽ đưa ra quyết định như sau

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

0.5 ở trên coi như là một mức độ, vượt qua nó thì ŷ sẽ là 1.

b. Cross Entropy loss function

Ta muốn dánh giá được được ước lượng ŷ = σ(**w** · **x** + b) gần sát với y thực tế (y = 1 hoặc 0) như thế nào thì có thể sử dụng **Cross-Entropy loss**.

Đầu tiên sử dụng hàm:

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

y = 1 -> p(y | x) = ŷ
y = 0 -> p(y | x) = 1 - ŷ

Lấy log 2 vế:

$$\begin{aligned} \log p(y|x) &= \log\left[\hat{y}^y (1-\hat{y})^{1-y}\right] \\ &= y \log \hat{y} + (1-y)\log(1-\hat{y}) \end{aligned}$$

Đổi dấu để chuyển thành **loss function** (một hàm mà ta cần cực tiểu hóa)

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

Vậy cuối cùng hàm **Cross-Entropy loss function** sẽ là:

$$L_{CE}(\hat{y}, y) = -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$

c. Tối ưu hàm Cross Entropy bằng thuật toán Gradient Descent

Mục tiêu là cần tìm weight để cực tiểu hóa hàm loss. Ta có dạng sau:

Với θ = w, b

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{\mathrm{CE}}(f(x^{(i)};\theta), y^{(i)})$$

Dùng thuật toán Gradient Descent để giải quyết bài toán này. Thuật toán sẽ tìm Gradient của hàm loss tại 1 điểm và di chuyển đến vị trí đối nghịch.

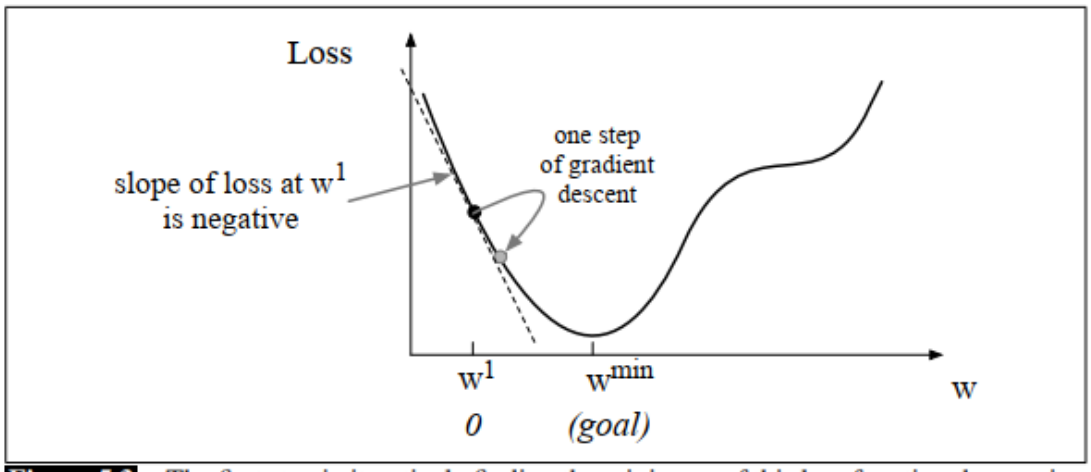


Figure 5.3 The first step in iteratively finding the minimum of this loss function, by moving w in the reverse direction from the slope of the function. Since the slope is negative, we need to move w in a positive direction, to the right. Here superscripts are used for learning steps, so w^1 means the initial value of w (which is 0), w^2 at the second step, and so on.

Trong hình trên, nếu ta muốn thuật toán đi nhanh hay chậm thì sẽ sử dụng tham số learning rate: η

$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x;w)$$

Áp dụng:

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta), y)$$

$$\nabla_\theta L(f(x;\theta), y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta), y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta), y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta), y) \end{bmatrix}$$

Ta tính được đạo hàm của hàm log:

$$\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

```
    # where: L is the loss function
    #        f is a function parameterized by θ
    #        x is the set of training inputs x(1), x(2),..., x(m)
    #        y is the set of training outputs (labels) y(1), y(2),..., y(m)

θ ← 0
repeat til done   # see caption
  For each training tuple (x(i), y(i)) (in random order)
    1. Optional (for reporting):        # How are we doing on this tuple?
       Compute ŷ(i) = f(x(i); θ)        # What is our estimated output ŷ?
       Compute the loss L(ŷ(i), y(i))   # How far off is ŷ(i) from the true output y(i)?
    2. g ← ∇θ L(f(x(i); θ), y(i))       # How should we move θ to maximize loss?
    3. θ ← θ − η g                      # Go the other way instead
return θ
```

Figure 5.5 The stochastic gradient descent algorithm. Step 1 (computing the loss) is used to report how well we are doing on the current tuple. The algorithm can terminate when it converges (or when the gradient norm $< \epsilon$), or when progress halts (for example when the loss starts going up on a held-out set).

Trong model được xây dựng với Sklearn, sử dụng Logistic Regression. Đặt các hypermeter **multi_class** = ***multinomial,*** **solver** = ***lbfgs***

- **multi_class:** Nếu data set gồm 2 label thì sẽ chọn 'OVR' còn nhiều label thì chọn 'multinominal'. Khi ta đặt là multinominal thì hàm training sẽ là cross-entropy loss.

  Hàm cross-entropy loss cho data nhiều label có dạng:

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_k, \quad \text{(where } k \text{ is the correct class)}$$
$$= -\log \frac{\exp(w_k \cdot x + b_k)}{\sum_{j=1}^{K} \exp(w_j \cdot x + b_j)} \quad \text{(where } k \text{ is the correct class)}$$

$$\frac{\partial L_{CE}}{\partial w_{k,i}} = -(\mathbb{1}\{y = k\} - p(y = k|x))x_i$$
$$= -\left(\mathbb{1}\{y = k\} - \frac{\exp(w_k \cdot x + b_k)}{\sum_{j=1}^{K} \exp(w_j \cdot x + b_j)}\right) x_i$$

  Với **y** ở dạng one-hot vector: $y_i = 1$, $y_j = 0$ $\forall j \neq i$

- **solver:** Các thuật toán để tối ưu vấn đề, tùy data thì sẽ có thuật toán thích hợp. Nó sẽ giúp giảm thời gian so với thuật toán "tầm thường", điển hình là ta không cần phải chọn learning rate.

4. Kết quả

a. Sử dụng thẳng data gốc ban đầu, mô hình logistic regression với các tham số mặc định

- Kết quả:

  Train Error: 6.41 %

  Test Error: 8.26 %

- Nhận xét:

  Mặc dù chưa được xử lý nhưng kết quả của mô hình khá là tốt

b. Sử dụng data đã qua xử lý, thay đổi các tham số của mô hình logistic regression

- Kết quả:

  Train Error: 4.35 %

  Test Error: 6.78 %

- Nhận xét:

  Chất lượng của mô hình đã được cải thiện

5. Tham khảo

https://harvard-iacs.github.io/2019-CS109A/sections/sec_6/

https://www.kaggle.com/ronnierulz/decoding-digits-using-pca-and-logistic-regression

https://stataguide.wordpress.com/2020/04/30/hien-tuong-da-cong-tuyen-multicollinearity/