



## Assignment NO.2 Solutions

Deep learning | winter 1400 | Dr.Mohammadi

Teacher Assistant:

Amirreza Fateh

---

Student name : **Amin Fathi**

Student id : **400722102**

## Problem 1

نویسندگان این مقاله دادگان آموزش (train) خود را که تحت عنوان gallery images تعریف کرده اند ، ابتدا به این صورت معرفی می کنند :

Gallery images شامل یک یا چند مجموعه عکس برای هر کلاس منحصر به فرد است که هر مجموعه عکس ها شامل چند عکس می باشد . همچنین داده تست (test) هم شامل تعدادی تصاویر از همان کلاس ها و موضوع هاست که با محاسبه برخی معیار های شباهت ، کلاس آن ها تخمین زده خواهد شد . نویسندگان مقاله ، روش پیشنهادی خود را از شاخه روش های non-parametric معرفی کرده اند که نقطه مقابل روش های parametric می باشد ، در روش های parametric که برای طبقه بندی مجموعه عکس ها به کار می روند اساس کار طبقه بندی ، شباهت بین توزیع های آماری بین تصاویر است . در حالی که روش های non-parametric مجموعه های تصویر را به صورت زیرفضاهای خطی یا غیرخطی نشان می دهند و در این زیر فضا ها فاصله را مقایسه می کنند . تکنیک پیشنهادی در این مقاله مبتنی بر مفهوم بازسازی تصویر با استفاده از طبقه بندی رگرسیون خطی (LRC) و طبقه بندی کنند نزدیک ترین زیرفضا (NS) می باشد .

تکنیک اعمالی :

فرض کنید تعداد عکس های موجود برای هر کلاس معین عضو مجموعه C برابر با N باشد و مجموعه عکس های هر کلاس C عضو مجموعه C را با نماد  $K_C$  نشان دهیم . همه تصاویر را به صورت خاکستری تک کاناله (grayscale) در آورده و به ابعاد  $a*b$  تغییر اندازه بدهیم ، در این صورت تصاویر کلاس C به صورت ماتریس به ابعاد  $a*b$  می توان نشان داد ، با ترکیب کردن سطر ها و ستون ها با هم و تبدیل ماتریس دو بعدی به بردار تک بعدی به صورت  $T*1$  که در آن  $T=a*b$  است ، میتوان تصاویر را به صورت یک بردار در آورد و آن را به صورت مجموعه  $q_c^n \in \mathbb{R}^{T \times 1}$  تعریف کرد که C در آن بیانگر کلاس تصاویر و n هم بیانگر شماره عکس پروتوتایپ موجود در آن کلاس است  $n = 1, 2, 3, \dots, N$ .

چنانچه همه تصاویر موجود در یک کلاس را به یک بردار زیر فضای خطی نگاشت کنیم ، حاصل مجموعه  $Q_C$  خواهد بود که در واقع حاصل ترکیب (به صورت افقی) تمام تصاویر موجود در یک کلاس در کنار هم است .

$$Q_c = [q_c^1 q_c^2 q_c^3 \dots q_c^N] \in \mathbb{R}^{T \times N}, \quad c = 1, 2, 3, \dots, C \quad (1)$$

N بیانگر تعداد تصاویر هر کلاس است و T همان  $a*b$  است .

در این حالت در واقع ما توانستیم برای هر کلاس یک بردار زیر فضا QC مختص آن کلاس تخصیص بدهیم که در واقع هر کلاس فضای اقلیدسی مختص به خود را داراست. برای جا افتادن بحث مثال زیر را در نظر بگیرید:

فرض کنید در کلاس شماره ۱، دو عکس داریم ( $N=2$ ) و این دو تصویر را به صورت خاکستری و به ابعاد ۲ پیکسل در ۲ پیکسل تبدیل کرده ایم مقادیر پیکسل ها را در تصویر زیر مشاهده می کنید.

$$\begin{bmatrix} 25 & 50 \\ 75 & 100 \end{bmatrix} \quad \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

عکس شماره ۱                      عکس شماره ۲

در این صورت بردار ها  $q$  برای هر تصویر به شکل زیر خواهد بود

$$q_1^1 = \begin{bmatrix} 25 \\ 50 \\ 75 \\ 100 \end{bmatrix}$$

$$q_1^2 = \begin{bmatrix} 10 \\ 20 \\ 30 \\ 40 \end{bmatrix}$$

و در نتیجه، رگرسیون کلاس ۱ به شکل زیر خواهد بود:

$$Q_1 = [q_1^1 \ q_1^2] = \begin{bmatrix} 25 & 10 \\ 50 & 20 \\ 75 & 30 \\ 100 & 40 \end{bmatrix}_{4 \times 2}$$

داده تست ( که قصد تشخیص کلاس آن را داریم ) را هم به همین منوال به رگسور تبدیل می کنیم ، داده تست همان مجموعه تصاویر تست ( با M عضو ) است که به صورت خاکستری و به همان ابعاد  $a*b$  تبدیل کرده و ادامه روند هم همانند دادگان آموزشی (train) است ؛

فقط در اینجا بردار های پایه زیر فضا ( رگسور ) را به جای نشان دادن با  $q_c^n$  با  $x_\mu^m$  نشان می دهیم و رگسور را نیز به جای نشان دادن با نماد Q با نماد X نشان می دهیم .

زیر فضای حاصل برای داده آزمایش (test)  $X_\mu$  نام دارد

$$X_\mu = [x_\mu^1 x_\mu^2 x_\mu^3 \dots x_\mu^M] \in \mathbb{R}^{T \times M}, \quad (2)$$

M بیانگر تعداد مجموعه تصاویر داده test که قصد تعیین کلاس آن ها را داریم است و T همان  $a*b$  است .

$\mu$  هم بیانگر کلاسی است که داده تست به آن تعلق دارد و هنوز برایمان مشخص نیست و هدف به دست آوردن آن است .

چنانچه  $X_\mu$  متعلق به کلاس C ام باشد ، بنابر این می توان آن را بر اساس ترکیب خطی عکس های کلاس C ام به دست آورد. به عبارت دیگر  $x_\mu$  را برای تصویر m ام میتوان با استفاده از فرمول زیر به دست آورد .

$$x_\mu^m = Q_c \gamma_c^m, \quad m = 1, 2, \dots, M, \quad c = 1, 2, \dots, C \quad (3)$$

$\gamma_c^m \in \mathbb{R}^{N \times 1}$  همان بردار پارامتر است

معادله شماره (3) با در نظر گرفتن شرط  $T > N$  را با استفاده از least square method می توان بردار پارامتر را به شکل زیر به دست آورد :

$$\gamma_c^m = (Q_c' Q_c)^{-1} Q_c' x_\mu^m, \quad m = 1, 2, \dots, M \quad c = 1, 2, \dots, C \quad (4)$$

که در آن Q پریم همان ترانهاده Q است .

با جاگذاری فرمول شماره 4 در فرمول شماره 3 به نتایج زیر می رسیم .

$$\hat{x}_c^m = Q_c \gamma_c^m, \quad m = 1, 2, \dots, M \quad c = 1, 2, \dots, C \quad (5)$$

$$\hat{x}_c^m = Q_c (Q_c' Q_c)^{-1} Q_c' x_\mu^m \quad (6)$$

$\hat{x}_c^m$  در واقع بردار تصویر بازسازی شده ی تصویر شماره  $m$  ام دادگان آزمایش (test) با استفاده از رگرسور کلاس  $c$  ام دادگان آموزش (train) است .

نویسندگان به جای محاسبه فرمول شماره 6 برای تک تک تصاویر ، تصاویر را در یک ماتریس ترکیب کرده و از  $X_\mu$  به جای  $x_\mu$  و از ماتریس پارامترها یا همان  $\Gamma_c \in \mathbb{R}^{N \times M}$  به جای استفاده از بردار پارامترها یا همان  $\gamma_c^m \in \mathbb{R}^{N \times 1}$  استفاده کرده اند که در آن صورت فرمول ها در نهایت به شکل زیر تبدیل خواهند شد .

$$X_\mu = Q_c \Gamma_c, \quad c = 1, 2, \dots, C \quad (7)$$

$$\Gamma_c = (Q_c' Q_c)^{-1} Q_c' X_\mu, \quad c = 1, 2, \dots, C \quad (8)$$

$$\hat{X}_c = Q_c \Gamma_c, \quad c = 1, 2, \dots, C \quad (9)$$

$$\hat{X}_c = Q_c (Q_c' Q_c)^{-1} Q_c' X_\mu \quad (10)$$

$\hat{X}_c \in \mathbb{R}^{T \times M}$  همان ماتریس تصویر باز سازی شده ی دادگان آزمایش (test) (یا همان  $X_\mu$ ) با استفاده از رگرسور کلاس  $c$  ام دادگان آموزش (train) ( یا همان  $Q_c$ ) است .

حال به مقایسه معیار اختلاف بین تصاویر ساخته شده با استفاده از رگرسور های هر کلاس با تصویر تست می پردازیم ، فاصله اقلیدسی بین تصویر ساخته شده  $m$  ام با استفاده از رگرسور کلاس  $c$  ام و تصویر تست  $m$  ام را به صورت زیر و با نماد  $d_c^m$  نشان می دهیم :

$$d_c^m = \|x_\mu^m - \hat{x}_c^m\|_2, \quad c = 1, 2, \dots, C, \quad m = 1, 2, \dots, M \quad (11)$$

نگارندگان مقاله در ادامه اشاره کرده اند که بر اساس تجربه آن ها چنانچه برای معیار شباهت از فرمول (11) فاصله اقلیدسی به صورت نمایی استفاده کنند نتایج بهتری به دست خواهند آورد ، به این صورت که مقدار خروجی فرمول 13 برای تصویر تست به ازای هر کدام از کلاس ها که بیشتر باشد ، تصویر تست متعلق به آن کلاس است .

$$\theta_c^m = e^{-\alpha d_c^m}, c = 1, 2, \dots, C, m = 1, 2, \dots, M \quad (12)$$

$$\Theta_c = \sum_{m=1}^M \theta_c^m, c = 1, 2, \dots, C \quad (13)$$

$$\mu = \arg \max_c (\Theta_c) c = 1, 2, \dots, C \quad (14)$$

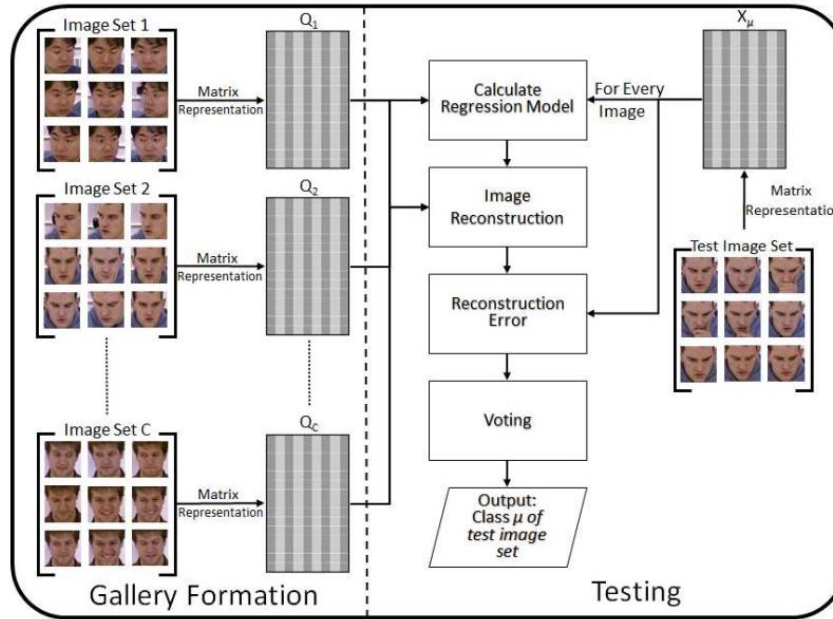


Figure 1. A block diagram of the proposed technique.

## Problem 1.b

ما در مسایل تشخیص چهره معمولا با 3 موضوع اصلی سر و کار داریم :

۱ - استخراج ویژگی

۲- تجزیه و تحلیل تشخیص ( مثلا تحلیل تشخیص خطی خطی فیشر و مساله کاهش بعد )

۳- قوانین طبقه بندی

برای بهبود مقاله سوال ۱ ، می توان در هر کدام از این بخش ها از نو آوری های اخیر استفاده کرد ؛ به طور مثال در مورد موضوع سوم ( قوانین طبقه بندی ) می توان از ایده ای که نگارندان [مقاله ۲](#) به کار برده اند استفاده کرد

در این مقاله علاوه بر استفاده از روش NS که در مقاله صورت سوال به کار رفته بود و در قسمت قبل آن را توضیح دادیم ، از FS (Farthest subspace) هم استفاده شده است و با ترکیب NS و FS ، طبقه بندی کننده NFS (nearest-farthest subspace) طراحی شده است .

ایده FS به این صورت است که به ازای هر کلاس ، یک زیر فضای "leave-one-class-out" تشکیل می دهیم که بردار پایه های این زیر فضا در واقع از تصاویر تمامی کلاس ها به غیر از کلاس مد نظر تشکیل شده است . ( شیوه تشکیل رگرسور مربوط به زیر فضای جدید همانند روش NS می باشد ) ، در اینجا سیستم رای گیری بر خلاف روش NS ( که تصویر تست مربوط به کلاسی بود که تصویر ساخته شده توسط زیر فضای آن کلاس کمترین اختلاف را با تصویر تست داشت ) می باشد و در واقع تصویر تست مربوط به کلاسی است که تصویر ساخته شده توسط زیر فضای "leave-one-class-out" آن کلاس بیشترین اختلاف را با تصویر تست داشته باشد ، چرا که از تصاویر آن کلاس در ساخت زیر فضای مذکور استفاده نشده و بنابراین منطقا باید اختلاف بیشتر در این حالت بیانگر بالا بودن احتمال تعلق تصویر تست به آن کلاس باشد.

در نهایت برای ترکیب NS و FS هم روش هایی ذکر شده است ، به طور مثال چنانچه فاصله اقلیدسی تصویر تست با زیر فضای کلاس  $i$  ام در روش NS را  $di$  بنامیم و فاصله اقلیدسی به دست آمده در روش FS بین تصویر تست و زیر فضای "leave - one - class - out" مربوط به همان کلاس  $i$  ام را  $Li$  بنامیم ، با تقسیم  $di$  بر  $Li$  متغیر جدیدی به دست می آید که هرچه مقدار آن کوچکتر باشد ، احتمال تعلق تصویر تست به آن کلاس بیشتر است .

$$j_i = \frac{d_i}{l_i}.$$

نگارندان مقاله در ادامه با ذکر مثالی نشان داده اند که FS چگونه می تواند برخی اشتباهات NS را برطرف می کند؛ در اینجا نیز می توان برای بهبود عملکرد روش پیشنهادی در مقاله صورت سوال از ترکیب روش FS با روش NS ذکر شده در مقاله استفاده کرد تا بتوان عملکرد بهتری را ثبت کرد.



## Problem 2.a

فرض کنید به هر کدام از ورودی ها مقدار ثابت  $\alpha$  اضافه کنیم ، در این صورت مقدار  $\text{Softmax}(o + \alpha)$  برابر است با :

$$\begin{aligned} \text{Softmax}(o + \alpha) &= \frac{e^{o_i + \alpha}}{\sum_j e^{o_j + \alpha}} = \frac{e^{o_i} \cdot e^{\alpha}}{\sum_j e^{o_j} \cdot e^{\alpha}} = \frac{e^{o_i} \cdot e^{\alpha}}{e^{\alpha} \cdot \sum_j e^{o_j}} \\ &= \frac{e^{o_i}}{\sum_j e^{o_j}} = \text{softmax}(o) \end{aligned}$$

که همانطور که در خط دوم مشاهده می شود ، این مقدار برابر همان  $\text{Softmax}(o)$  می باشد.

## Problem 2.b

ابتدا تابع ضرر Hinge را ساده سازی می کنیم ، این تابع در واقع یک تابع دو ضابطه ای است و چنانچه امتیاز پیش بینی شده برای کلاس مورد نظر از جمع امتیاز پیش بینی شده برای کلاس دیگر با ۱ بیشتر باشد ، مقدار خروجی تابع ضرر برابر صفر می باشد و در غیر این صورت خروجی برابر  $o_j - o_i + 1$  خواهد بود .

$$\begin{cases} 0 & o_i \leq o_j + 1 \\ o_j - o_i + 1 & o_i > o_j + 1 \end{cases}$$

چنانچه از رابطه دو ضابطه ای بالا نسبت به  $o_i$  یا خروجی نسبت به کلاس مدنظر گرادینان بگیریم مقدار برابر است با :

$$\begin{cases} 0 & o_i \leq o_j + 1 \\ -1 & o_i > o_j + 1 \end{cases}$$

که با توجه به حضور  $\sum$  در فرمول ، نتیجه نهایی گرادیان به صورت زیر خواهد شد

$$\frac{\partial L}{\partial o_i} = - \sum_{j \neq i} 1(o_i > o_j + 1)$$

فرمول بالا در واقع تعداد کلاس هایی که خروجی آن ها از کلاس مد نظر ما فاصله ای کمتر از ۱ دارند را بر می شمارد ( و ضرب در ضریب -۱ می کند )

همانطور که در فرمول مشتق چند ضابطه ای مشاهده شد ، مقدار گرادیان برای کلاس های دیگر هم برابر خواهد بود با :

$$\frac{\partial L}{\partial o_j} = +1$$

در مورد تفاوت دو تابع ضرر hinge و CE میتوان گفت که CE با احتمالات ناشی از Softmax سر و کار دارد به دنبال بهینه کردن وزن ها و پارامتر هاست و تا پیدا کردن بهینه ترین پارامتر ها کارش را ادامه میدهد ولی در hinge همین که مقدار margin مورد نظر رعایت شود کافی است و گرادیان نسبت به آن کلاس برابر ۰ میشود ( همانطور که گفتیم گرادیان در این حالت برابر مجموع تعداد کلاس هایی است که مارجین آن ها رعایت نشده است و اگر همه کلاس ها مارجین شان رعایت شود در این صورت مقدار گرادیان برابر صفر خواهد بود و تغییری در وزن ها رخ نخواهد داد )

منابع: <https://rohanvarma.me/Loss-Functions/>

<https://stats.stackexchange.com/questions/155088/gradient-for-hinge-loss-multiclass>

## Problem 3

ابتدا داده های ورودی را به شکل زیر تعریف می کنیم . دادگان  $X$  را برای اینکه بتوانند مورد استفاده قرار گیرند باید به صورت

$(-1, 1)$  تغییر شکل داد (reshape)

همچنین ، مقادیر متناظر دادگان ورودی را هم تحت عنوان متغیر  $Y$  تعریف می کنیم.

```
X = np.array([1.8, 2.1, 3.3, 2.5 , 1.1 , 0.5 , 5.1 , 5.3 , 6.6 , 7.3 , 8.1 , 55.2 ])
X = np.reshape(X , (-1, 1))
Y = np.array([ 0 , 0, 0 , 0 , 0 , 0 , 1 , 1 , 1 , 1 , 1 , 1])
```

ابعاد خروجی دادگان به صورت زیر خواهد بود .

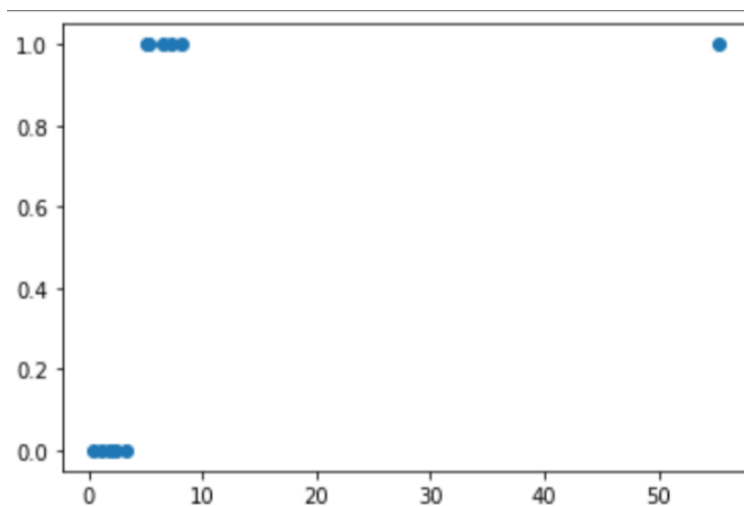
$X.shape$

$(12, 1)$

$Y.shape$

$(12,)$

چنانچه دادگان صورت سوال را رسم کنیم ، شکل زیر حاصل خواهد شد .



```
model = LinearRegression()
model.fit(X, Y)
```

مدل linear Regression را با توجه به دادگان موجود fit میکنیم .

کار این نوع رگرسیون در این مثال ، یافتن خطی است با معادله  $y = b_0 + b_1 * x$  که در آن مقدار مربع خطا بر اساس مقادیر پیش بینی شده توسط این معادله خط و مقادیر اصلی برچسپ دادگان ، مینیمم باشد.

b0 در کتابخانه sklearn همان model.intercept\_ و مقدار b1 همان model.coef\_ خواهد بود .

```
print('intercept:', model.intercept_)

intercept: 0.372948908290032

print('slope:', model.coef_)

slope: [0.0154157]
```

در واقع معادله خط در این مثال برابر است با :

$$Y = 0.015 * X + 0.372$$

چنانچه مقادیر پیشبینی مدل آموزش دیده برای دادگان ورودی را با مقادیر حاصل از این معادل خطی محاسبه کنیم ، نتایج یکی خواهد بود

```
y_pred = model.predict(X)

print('predicted response:', y_pred, sep='\n')

predicted response:
[0.40069718 0.40532189 0.42382073 0.41148817 0.38990618 0.38065676
 0.451569    0.45465214 0.47469255 0.48548355 0.49781611 1.22389576]
```

```
y_pred2 = model.intercept_ + np.sum(model.coef_ * X, axis=1)
print('predicted response:', y_pred2, sep='\n')
```

predicted response:

```
[0.40069718 0.40532189 0.42382073 0.41148817 0.38990618 0.38065676
 0.451569    0.45465214 0.47469255 0.48548355 0.49781611 1.22389576]
```

چنانچه خروجی معادله خط را برابر 0.5 قرار بدهیم ، مختصات (x) نقطه ی مد نظر صورت سوال هم به دست می آید .

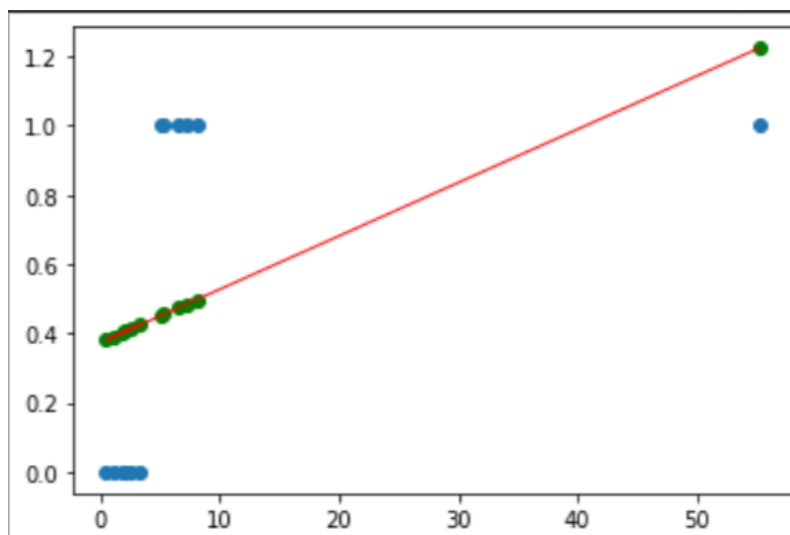
```
0.5 = ( decision_boundary * model.coef_ ) + model.intercept
```

```
[17] decision_boundary = (0.5 - model.intercept_) / model.coef_
```

```
[18] print('decision boundary :', decision_boundary )
```

```
decision boundary : [8.24166667]
```

در نهایت هم دادگان اصلی ( نقاط آبی ) را همراه با خروجی پیش بینی شده برای آن ها توسط linear Regression ( نقاط سبز ) و معادله خط مربوطه ( قرمز ) را رسم میکنیم.



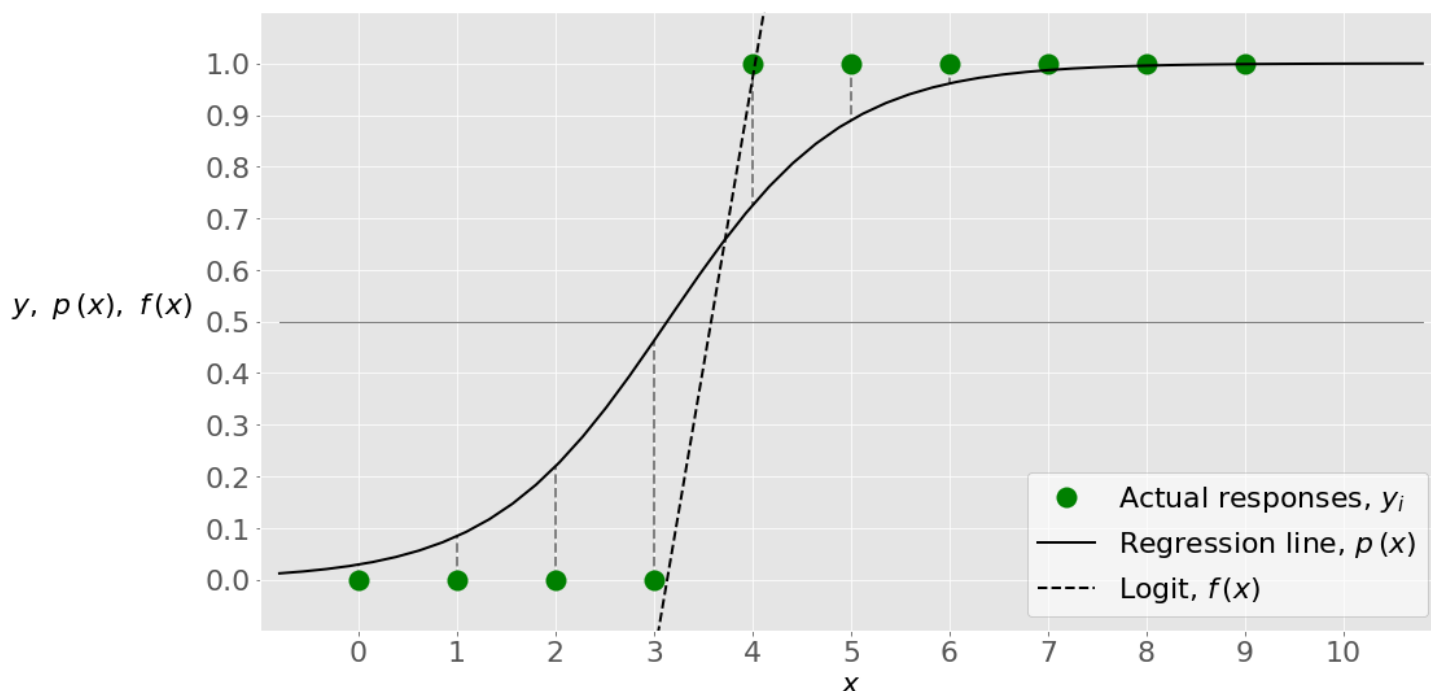
حال همین کار ها را برای logistic Regression انجام می دهیم ، دادگان مشترکند پس دوباره آن ها را تعریف نمی کنیم ، مدل را با استفاده از آن دادگان fit می کنیم . از آنجا که دیتاست کوچکی داریم slover را liblinear ست می کنیم

```
[4] model2 = LogisticRegression(solver='liblinear', C = 10.0 ,random_state=0)
```

```
[5] model2.fit(X, Y)
```

```
LogisticRegression(C=10.0, random_state=0, solver='liblinear')
```

باید در نظر داشت که در logistic Regression با دو تابع  $px$  ,  $fx$  در طرف هستیم ، تابع  $fx$  که یک تابع خطی است و  $px$  هم سیگموئید تابع  $fx$  است



- The logit  $f(x_1, x_2) = b_0 + b_1x_1 + b_2x_2$
- The probabilities  $p(x_1, x_2) = 1 / (1 + \exp(-f(x_1, x_2)))$

برای محاسبه مرز تصمیم ، میبایست مقدار  $x$  ای که در آن  $px$  برابر 0.5 است یا به عبارت دیگر بنابه فرمول  $fx$  برابر 0 است را حساب کنیم .

در این سوال  $fx$  ما برابر است با  $b0 + b1 * x$  و در واقع ما نیاز داریم تا  $b1$  ,  $b0$  را به دست آوریم ؛ که مقدار  $b0$  همان `model.intercept_` و مقدار  $b1$  همان `model.coef_` است .

```
model2.intercept_  
array([-4.27321491])  
  
model2.coef_  
array([[1.12305135]])
```

در واقع  $fx$  در این سوال برابر است با  $1.12x - 4.27$

برای به دست آوردن مرز تصمیم ،  $fx$  را باید با 0 برابر قرار دهیم و  $x$  را به دست آوریم  
که برابر است با 3.8

```
0 = ( decision_boundary * model2.coef_ ) + model2.intercept_  
  
▶ decision_boundary2 = - model2.intercept_ / model2.coef_  
print(decision_boundary2)  
  
📄 [[3.80500403]]
```

نتایج به دست آمده هم از این روش هم گویای این حقیقت است که تماماً مقادیر به درست پیش بینی شده اند .

```
model2.predict_proba(X)  
array([[0.90479901, 0.09520099],  
       [0.87155824, 0.12844176],  
       [0.63810424, 0.36189576],  
       [0.81238563, 0.18761437],  
       [0.95425544, 0.04574456],  
       [0.97614667, 0.02385333],  
       [0.18933346, 0.81066654],  
       [0.1572336 , 0.8427664 ],  
       [0.04152993, 0.95847007],  
       [0.01935879, 0.98064121],  
       [0.00797449, 0.99202551],  
       [0.          , 1.          ]])  
  
YPRED = model2.predict(X)  
print(YPRED)  
[0 0 0 0 0 0 1 1 1 1 1 1]
```

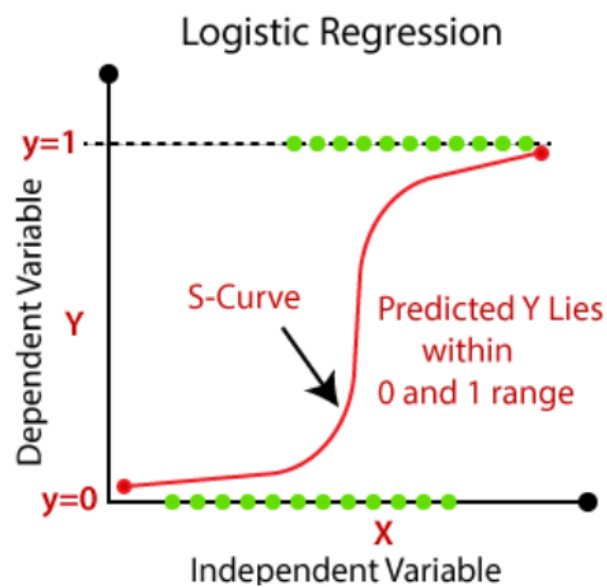
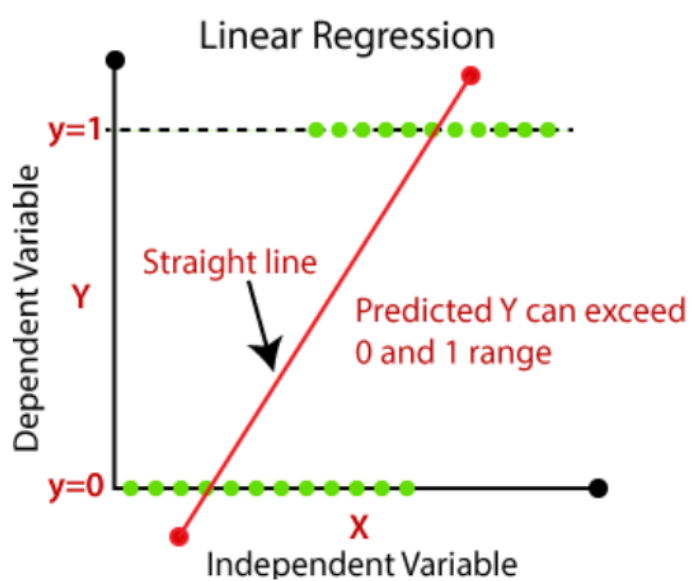
مشاهده می شود چنانچه نتایج را از طریق فرمول ریاضی هم حساب کنیم باز همان نتایج حاصل می شود .

```
Y_pred4 = 1/ (1 + np.exp(- (( X * model2.coef_ ) + model2.intercept_ )))
```

Y\_pred4

```
array([[0.09520099],  
       [0.12844176],  
       [0.36189576],  
       [0.18761437],  
       [0.04574456],  
       [0.02385333],  
       [0.81066654],  
       [0.8427664 ],  
       [0.95847007],  
       [0.98064121],  
       [0.99202551],  
       [1.          ]])
```

نتیجه گیری و مقایسه :



همانطور که مشاهده می شود ، linear Regression برای مسایل رگرسیون و به دست آوردن خروجی به کار می رود در حالی که logistic Regression برای مسایل طبقه بندی به کار می رود .



از آنجا که linear Regression هدف نهایی اش پیدا کردن بهینه ترین خط ممکن برای پیش بینی بهترین مقادیر خروجی است ( با کم ترین خطا ) ، بنابر این فقط بر روی دادگانی عملکرد مناسب دارد که بین متغیر های وابسته و ثابت آن دادگان رابطه خطی برقرار باشد . در مسئله صورت سوال ، به دلیل وجود داده شماره ۱۲ با  $x$  به بزرگی 55.2 که در فاصله نسبتا دور تری نسبت به سایر دادگان قرار دارد ، امکان تعریف رابطه بین ورودی و خروجی به صورت خطی وجود ندارد ، چون همانطور که در نتایج هم مشاهده شد ، مدل این داده را هم در پیدا کردن بهترین خط در نظر گرفته و تاثیر می دهد که این باعث ضعیف تر شدن عملکرد مدل می شود .

اما از آنجا که برجسپ دادگان ما به صورتی است که از یک جا به بعد لیبیل ها یکسان هستند و دادگان حالت پله به خود میگیرند ، می توان با انتخاب خط رگرسیون مناسب ( همان  $fx$  که به وسیله انتخاب C بالاتر ، خطی با شیب بیشتر در نظر گرفته میشود ) تمامی دادگان را به درستی کلاس بندی کرد و حضور یا عدم حضور داده ای همانند داده شماره ۱۲ در این حالت مشکلی ایجاد نمیکند.

از دیگر تفاوت هایی که مشاهده شد این بود که خروجی در linear به صورت پیوسته بود ( مسئله رگرسیون ) در حالی که خروجی در logistic به صورت گسسته ( مسئله کلاس بندی ) بود ، linear در باره پیدا کردن بهترین خط که بتواند بهترین تقریب را در باره خروجی داده ها بدهد است در حالی که logistic به دنبال پیدا کردن بهترین منحنی سیگموئیدی برای تطبیق روی دادگان است و در نهایت دو باره تاکید می شود که اگر رابطه بین  $x$  و  $y$  را میتوان به صورت خطی بیان کرد ، linear گزینه بهتری برای رگرسیون است ، در حالی که چنانچه رابطه بین  $x, y$  را بتوان با استفاده از منحنی سیگموئید تعریف کرد logistic گزینه بهتری خواهد بود .

کد این سوال در فایل پیوست گردیده است .

منابع :

<https://realpython.com/logistic-regression-python/>

<https://realpython.com/linear-regression-in-python/>

[Linear Regression vs Logistic Regression - Javatpoint](#)

## Problem 4

برای حل سوال از لینک زیر استفاده شده است :

<https://www.cs.toronto.edu/~lczhang/321/tut/tut04.html>

ابتدا دیت ست Mnist را لود کرده و ۹ عکس اول آن را نمایش می دهیم .

```
# load the training data
mnist = dt.MNIST('data', train=True, download=True)
mnist = list(mnist)[:2500]

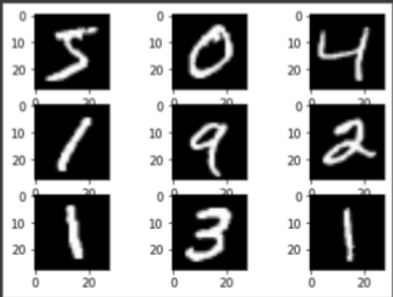
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
9913344/7 [00:00<00:00, 260177]
Extracting data/MNIST/raw/train-images-idx3-ubyte.gz to data/MNIST

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
29696/7 [00:00<00:00, 517100.72]
Extracting data/MNIST/raw/train-labels-idx1-ubyte.gz to data/MNIST

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
1649664/7 [00:00<00:00, 482367]
Extracting data/MNIST/raw/t10k-images-idx3-ubyte.gz to data/MNIST

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
5120/7 [00:00<00:00, 67561.74]
Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST

# plot the first 12 images in the training data
import matplotlib.pyplot as plt
for k, (image, label) in enumerate(mnist[:9]):
    plt.subplot(3, 3, k+1)
    plt.imshow(image, cmap='gray')
```



تصاویر به ابعاد  $28 * 28$  پیکسل هستند ، و ما در این سوال ۲۵۰۰ تصویر اول این دادگان را استفاده می کنیم ، پس از تبدیل تصاویر به تانسور ، از ۲۰۰۰ عکس اول به جهت train استفاده می کنیم .

برای ایجاد مدل خطی در PyTorch ما از کلاس `nn.linear` استفاده می کنیم ، پارامتر اولی که به آن پاس می دهیم تعداد فیچر های ورودی است و پارامتر دوم تعداد فیچر های خروجی است ( که برای کلاس بندی باینری و رگرسیون برابر است با ۱ و برای طبقه بندی چند کلاسه به تعداد کلاس های خروجی است )

در این سوال با توجه به اینکه ابعاد هر تصویر  $28 \times 28$  است پس ما ۷۸۴ فیچر یا ویژگی ورودی داریم و با توجه به اینکه تعداد کلاس های ما ( رقم ها ) برابر است با ۱۰ ، کلاس `nn.linear` به کار رفته به شکل زیر خواهد بود ، PyTorch به صورت خودکار مقادیر اولیه وزن ها و بایاس ها را تنظیم می کند که معمولاً عدد رندم در بازه ۰ هستند و هر بار که دوباره ران کنیم ، این مقادیر تغییر خواهند کرد.

```
model = nn.Linear(784, 10) # 784 = 28*28

# Let's verify that the shapes of the weights and biases are what we expect
weight, bias = list(model.parameters())
print(weight.shape)
print(bias.shape)

torch.Size([10, 784])
torch.Size([10])
```

حال به سراغ تابع `run_gradient_descent` که وظیفه آن اجرای الگوریتم گرادیان کاهشی است برویم ، از آنجا که با مسئله طبقه بندی چند کلاسه روبه رو هستیم از تابع ضرر `crossEntropy` استفاده می کنیم .

```
criterion = nn.CrossEntropyLoss()
```

تابع ضرر خروجی `logit` ( قبل از اعمال softmax ) را به عنوان ورودی میگیرد به همراه مقدار `ground-truth` برای بهینه سازی وزن ها ، از بهینه ساز SGD استفاده می کنیم ، نرخ یادگیری را هم برابر با ۰.۱ قرار می دهیم .

```
optimizer = optim.SGD(model.parameters(), lr=learning_rate, weight_decay=weight_decay)
```

در این قسمت هم تصاویر را به خوشه ها ۶۴ تایی پس از شافل کردن تقسیم می کنیم

```
train_loader = torch.utils.data.DataLoader(
    mnist_train,
    batch_size=batch_size,
    shuffle=True)
```

در شبکه کد زیر ، به ازای هر epoch و در هر epoch به ازای هر خوشه ، ابتدا عکس ها ( مجموعه عکس ۶۴ تایی که هر کدام ۲۸ \* ۲۸ است و XS نام گذاری شده ) و لیبل مربوط به آن را (ts) استخراج می کنیم ، تصاویر را flatten کرده و به حالت 1\* 784 تبدیل میکنیم، سپس مقدار logit را با استفاد از model حساب کرده و با مقایسه آن با لیبل های تصاویر ، مقدار خروجی تابع ضرر را به دست می آوریم ، با استفاده از این مقدار ضرر ، پارامتر های مدل را آپدیت می کنیم و در نهایت مقادیر گرادیان را به جهت استفاد برای خوشه بعدی صفر می کنیم .

```
# training
n = 0 # the number of iterations
for epoch in range(num_epochs):
    for xs, ts in iter(train_loader):
        if len(ts) != batch_size:
            continue
        xs = xs.view(-1, 784)    # flatten the image. The -1 is a wildcard
        zs = model(xs)
        loss = criterion(zs, ts) # compute the total loss
        loss.backward()          # compute updates for each parameter
        optimizer.step()         # make the updates for each parameter
        optimizer.zero_grad()    # a clean up step for PyTorch
```

در ادامه به ازای هر ۱۰ تکرار در چرخه train عملکرد مدل را ثبت و ضبط میکنیم .

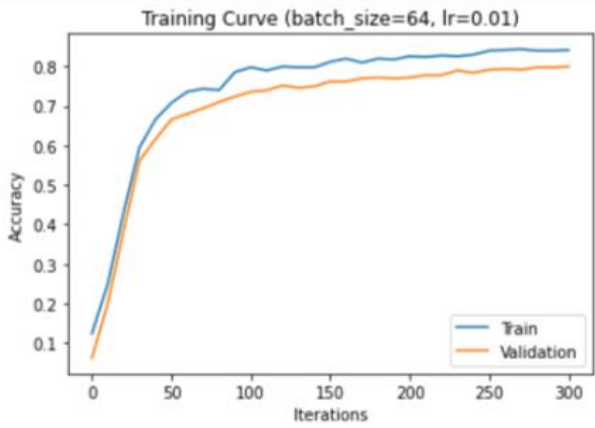
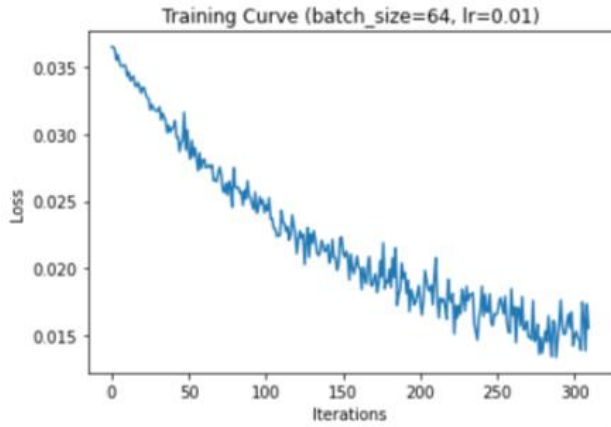
```
# save the current training information
iters.append(n)
losses.append(float(loss)/batch_size) # compute *average* loss

if n % 10 == 0:
    iters_sub.append(n)
    train_acc.append(get_accuracy(model, mnist_train))
    val_acc.append(get_accuracy(model, mnist_val))
# increment the iteration number
n += 1
```

از آنجا که به ازای هر خوشه یک تکرار یا iterarion داریم ، و در کل ۲۰۰۰ عکس و خوشه ها ۶۴ تایی است ، پس در هر epoch ۳۱ تکرار داریم و با توجه به اینکه کلا ۱۰ epoch داریم پس تعداد کل تکرار ها برابر است با ۳۱۰ عدد .

در انتها یک مدل خطی با ۷۸۴ ورودی و ۱۰ خروجی تعریف می کنیم و نتایج را با استفاده از تابعی که هم اکنون نوشتیم مشاهده می کنیم . که به شکل زیر است

```
model = nn.Linear(784, 10)
run_gradient_descent(model, batch_size=64, learning_rate=0.01, num_epochs=10)
```



```
Linear(in_features=784, out_features=10, bias=True)
```