شناسایی اماری الگو

تمرین 2

بهاره غلامی

40033626

مهسا ملایم

40032718

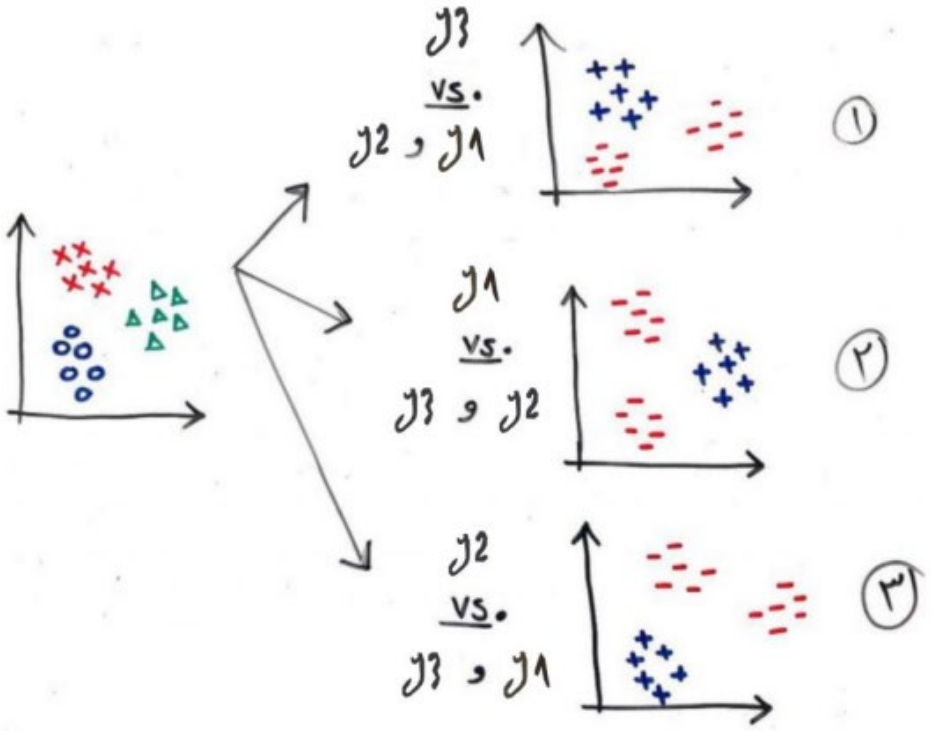
1401/9/28

بخش اول :

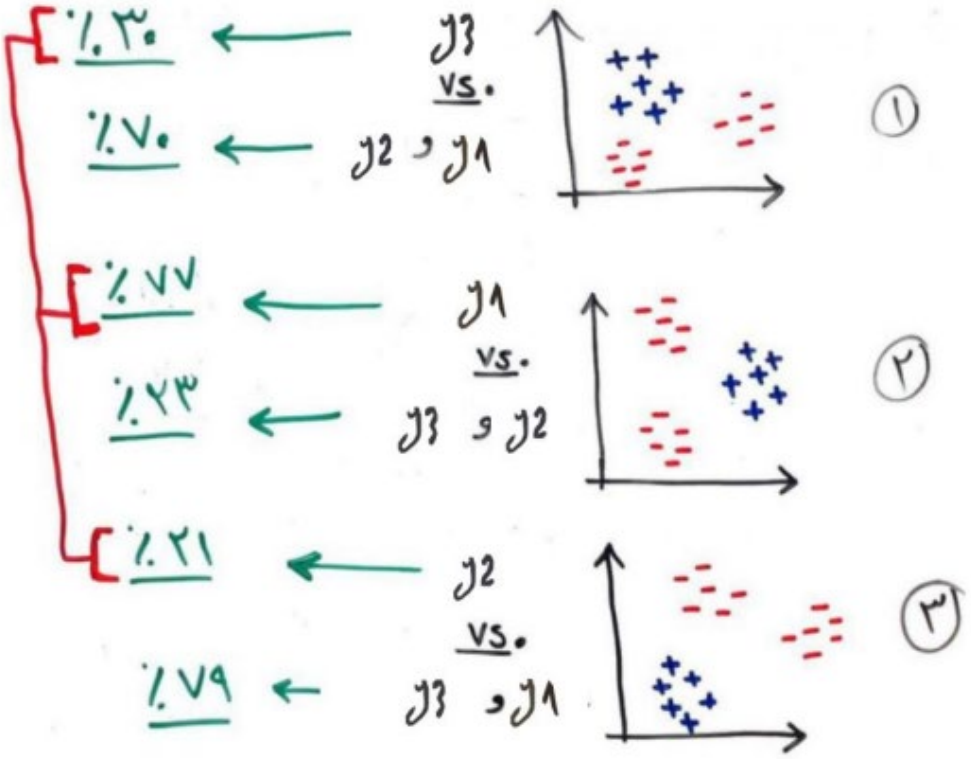
در این قسمت از روش one vs all برای کلاس بندی داده ها استفاده میکنیم .

این روش برای طبقه بندی داده های چند کلاسه استفاده میشود .

فرض کنید ما سه کلاس داریم به صورت شکل زیر . پس نیاز به سه مجموعه داده داریم که در هر یک نمونه های یک کلاس در مقابل نمونه های بقیه کلاس ها قرار بگیرد (در شکل کلاس موردنظر با + و کلاس بقیه که در مقابل ان قرار میگیرد را با – نمایش داده)و سپس باید روی هر مجموعه یک الگوریتم طبقه بندی باینرطراحی کنیم که از رگرسیون لجستیک استفاده میکنیم در این تمرین .



یک نمونه جدید به این سه مدل که یه دست اورده ایم میدهیم نتایج به صورت زیر است :



احتمال کلاس y1 بیشتر از بقیه احتمال است پس این نمونه مربوط به کلاس y1 است و برای مقایسه باید فقط کلاس هاس + را با هم مقایسه کنیم .

پیاده سازی :

: Normalize

def Normalize()

ورودی تابع x یا همون متغیر های مستقل هست و مقدار x به بازه بین صفر تا یک میبرد.

: Relabel

def relabel(y_train)

این تابع مقدار y_train را میگیرد و سه تا مقدار yبه ما برمیگرداند یکی مربوط به زمانی که کلاس یک و بقیه صفر یکی مربوط به زمانی که کلاس 2 مقدار یک داشته باشد و برچسب بقیه کلاس ها صفر و یکی مربوط به زمانی که کلاس سه مقدار یک داشته باشد و بقیه کلاس ها صفر .

: Sigmoid

def sigmoid(x,theta,theta0)

این تابع برای محاسبه سیگموید استفاده میشود .

:cost(y_hat,y,m)

مقدرا هزینه را در هر تکراربا استفاده از این تابع به دست می اوریم .

:LogisticRegression

def LogisticRegression(x, y,alpha,itr)

برای اموزش مدل از این تابع استفاده میکنیم .که ورودی دیتای اموزش هست و خروجی ان مجموعه تتا ها .

:Predict

def predict(x_test,theta,theta0)

برای مرحله تست از این تابع استفاده میکنیم که ورودی ان مجموعه تتاهای به دست امده در مرحله اموزش و خروجی ان برچسبی هست که به دست می اوریم برای هر دیتای تست .

از این تابع استفاده میکنیم و سپس طبق توضیحات بالا برچسب یک دیتای تست را احتمال بیشتر قرار میدهیم .

برای مشخص کردن برچسب یک دیتای تست از این قطعه کد استفاده کردیم :

```python
#predictlabel
y_hat1=predict(x_test,theta_m1,theta0_m1)
y_hat2=predict(x_test,theta_m2,theta0_m2)
y_hat3=predict(x_test,theta_m3,theta0_m3)
label=[]
for i in range(y_hat1.shape[0]):
    max_pro=max(y_hat1[i],y_hat2[i],y_hat3[i])
    if y_hat1[i]==max_pro and y_hat2[i]==max_pro  :
        label.append(random.randint(1,2))
        continue
    if y_hat2[i]==max_pro and y_hat3[i]==max_pro :
        label.append(random.randint(2,3))
        continue

    if y_hat1[i]==max_pro and y_hat3[i]==max_pro :
        label_t=random.randint(1,3)
        while(label_t== 2):
            label_t=random.randint(1,3)
        label.append(label_t)
        continue
    if y_hat1[i]==max_pro and y_hat3[i]==max_pro and y_hat2[i]==max_pro :
        label.append(random.randint(1,3))
        continue
    if y_hat1[i]==max_pro :
        label.append(1)
    if y_hat2[i]==max_pro :
        label.append(2)
    if y_hat3[i]==max_pro :
        label.append(3)

label=np.array(label)
accuracy_test = np.sum(y_test == label) / (y_test.shape[0])
print('Test accuracy:', accuracy_test)
```

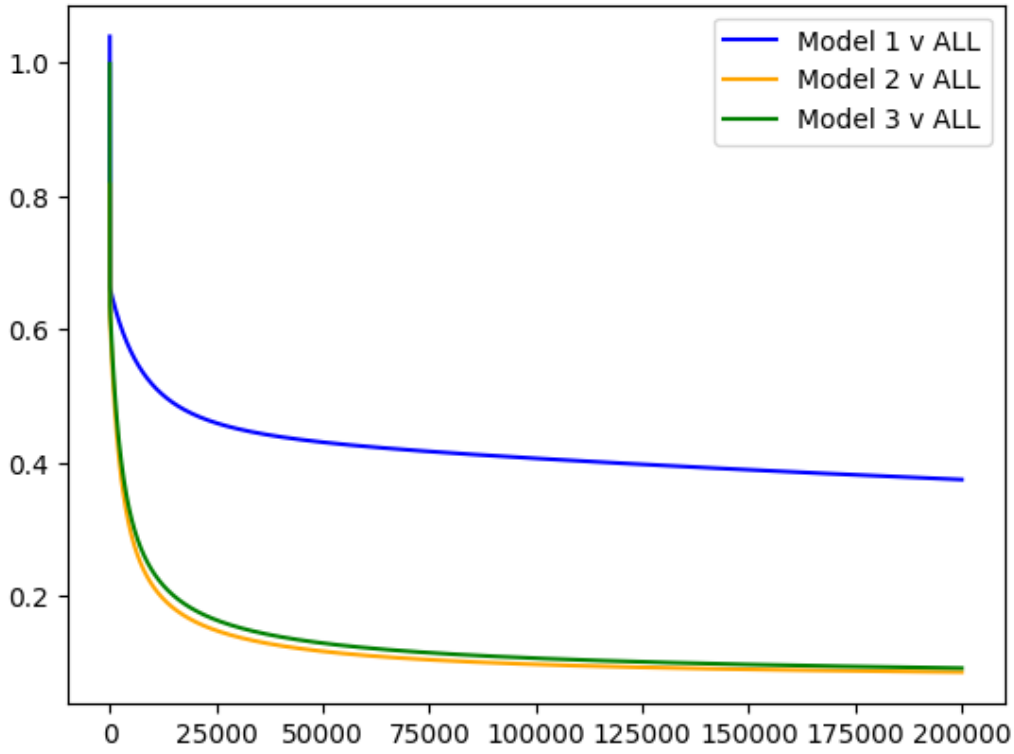
دقت اموزش :

```
Train accuracy: 0.8809523809523809
```

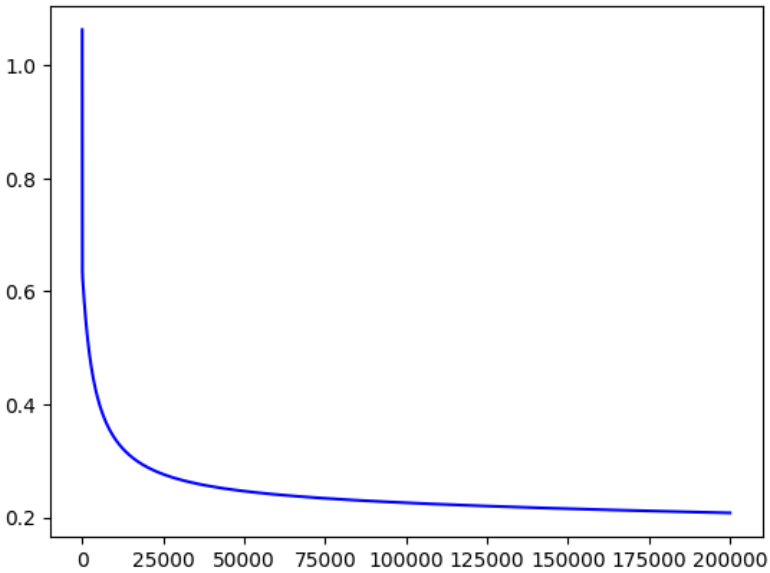
دقت تست :

```
Test accuracy: 0.9761904761904762
```

نمودار هزینه برای سه مدلی که اموزش دیده :



میانگین هزینه :

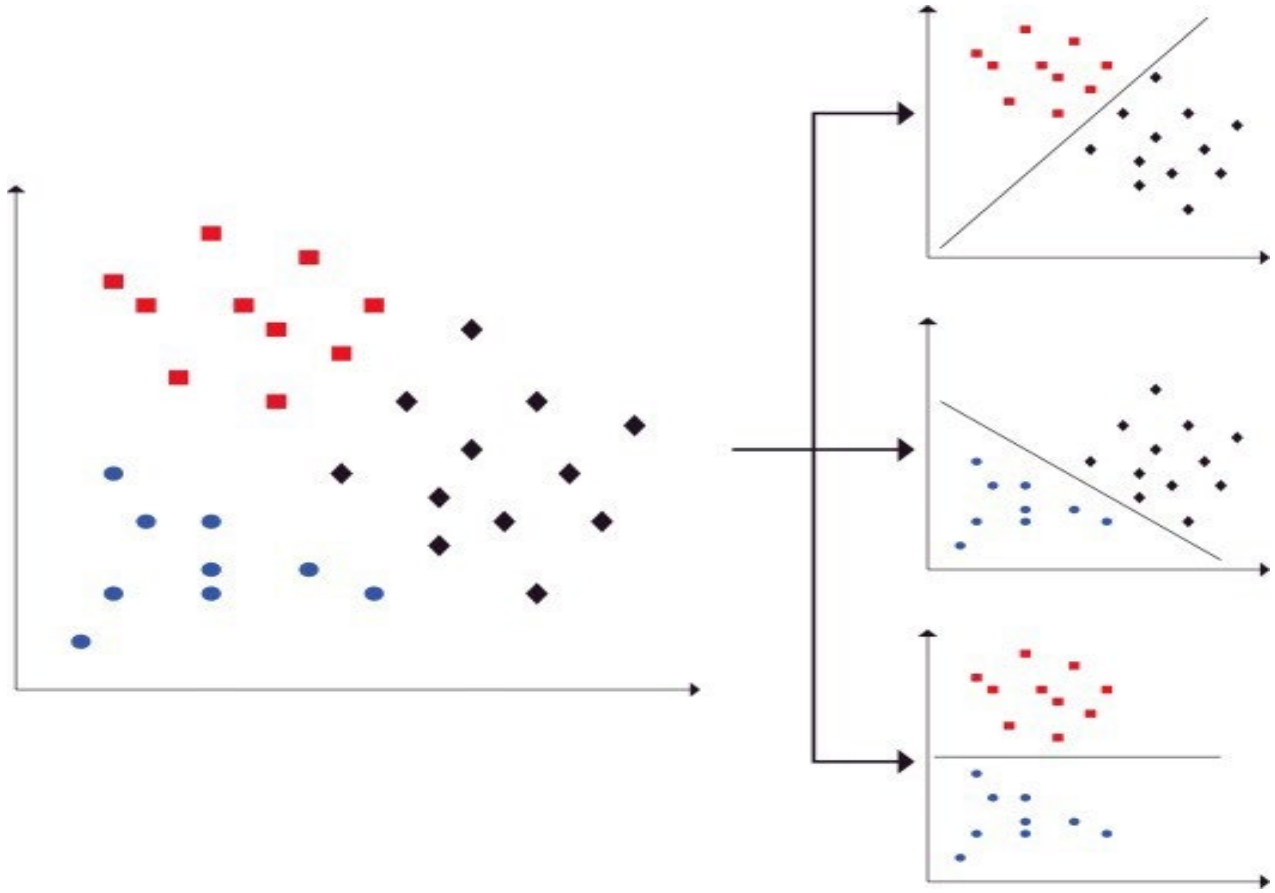


سه بار باید طبقه بندی باینری را در این روش صدا بزنیم.در این روش در صورتی که n کلاس داشته باشیم باید nبار طبقه بندی باینری را صدا بزنیم .

تعداد ایپاک را 200000 را برای هر مدل در نظر گرفتیم و مقدار الفا را 0.07 .

روش one vs one :

این روش هم برای طبقه بندی چند کلاسه استفاده میشود د ر این روش اگر سه کلاس داریم یکبار کلاس اول و دوم را به رگرسیون لاجستیک میدهیم یکبار کلاس اول و سوم را به رگرسیون لجستیک میدهیم و یکبار هم کلاس دوم و سوم را و سه مدل به دست می اوریم زمانی که دیتای جدید داریم به هر سه این مدل ها میدهیم و برچسبی که از همه بیشتر رای اورد را به عنوان برچسب در نظر میگیریم .



پیاده سازی :

(توابعی که در قسمت قبل توضیح دادم و مشترک هستند را دیگر در این قسمت نمی اورم)

:load_data

def load_data()

از این تابع استفاده میکنم برای اماده سازی دیتا با استفاده از این تابع دیتای اموزش را سه مجموعه دیتا به دست می اورم یک مجموعه مربوط به کلاس یک با دو ،یکی مربوط به کلاس یک با سه و یکی هم مربوط به کلاس دو با سه .

def predict(x,y):

از این تابع برای به دست اوردن y_hat استفاده میکنیم که ورودی ان متغیر مستقلx و متغیر وابسته y و دیتا را به هر سه مدل میدهیم و اون کلاسی که از همه بیشتر پیش بینی شده را به عنوان کلاس برنده در نظر میگیریم .

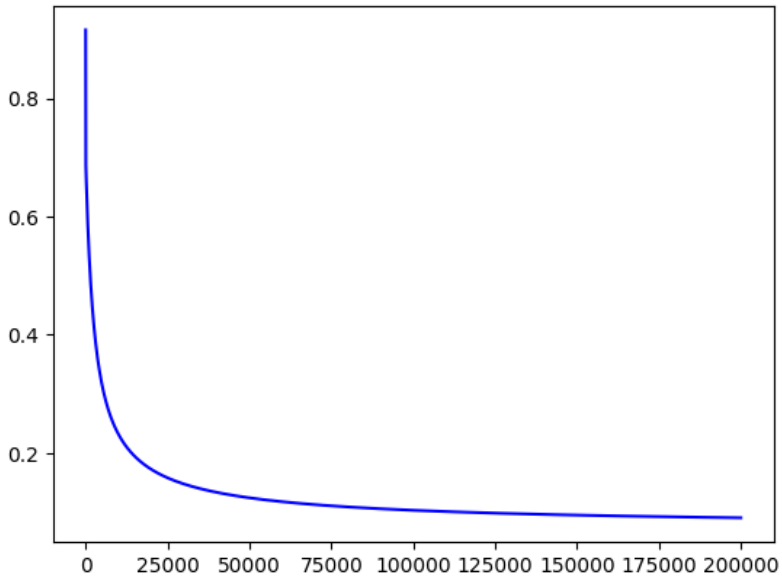
دقت اموزش :

```
Train accuracy: 0.8154761904761905
```

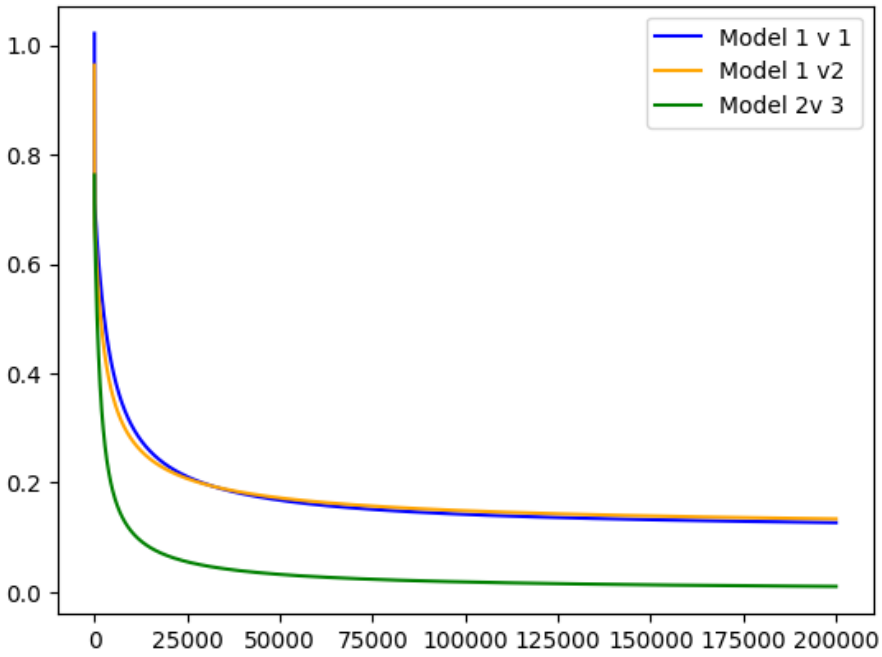
دقت تست :

```
Test accuracy: 0.9047619047619048
```

میانگین هزینه :



نمودار هزینه برای هر سه مدل :



سه بار کلاسیفایر باینری را صدا میزنیم در این روش از در صورتی که n کلاس داشته باشیم باید n(n-1)/2 کلاس باینری را انجام دهیم .

تعداد ایپاک را 200000 را برای هر مدل در نظر گرفتیم و مقدار الفا را 0.07 .

: Softmax

تابع سافت مکس زمانی استفاده میشود که قصد داریم روش های طبقه بندی چند کلاسه استفاده کنیم که فرمول ان به صورت زیر است .

$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

دلیل استفاده از سافت مکس این است که خروجی تابع به ما این اجازه را می دهد یک توزیع احتمال را روی نتایج مختلف و متنوع انجام دهیم .

پیاده سازی :

def softmax(z)

تابع سافت مکس که در بالا توضیح دادیم را پیاده سازی کردیم.

def LogisticRegression(x, y,alpha,itr)

برای اموزش مدل از تابع بالا استفاده میکنیم .

def predict(X,theta)

برای مشخص کردن برچسب دیتای جدید از تابع بالا استفاده میکنیم .

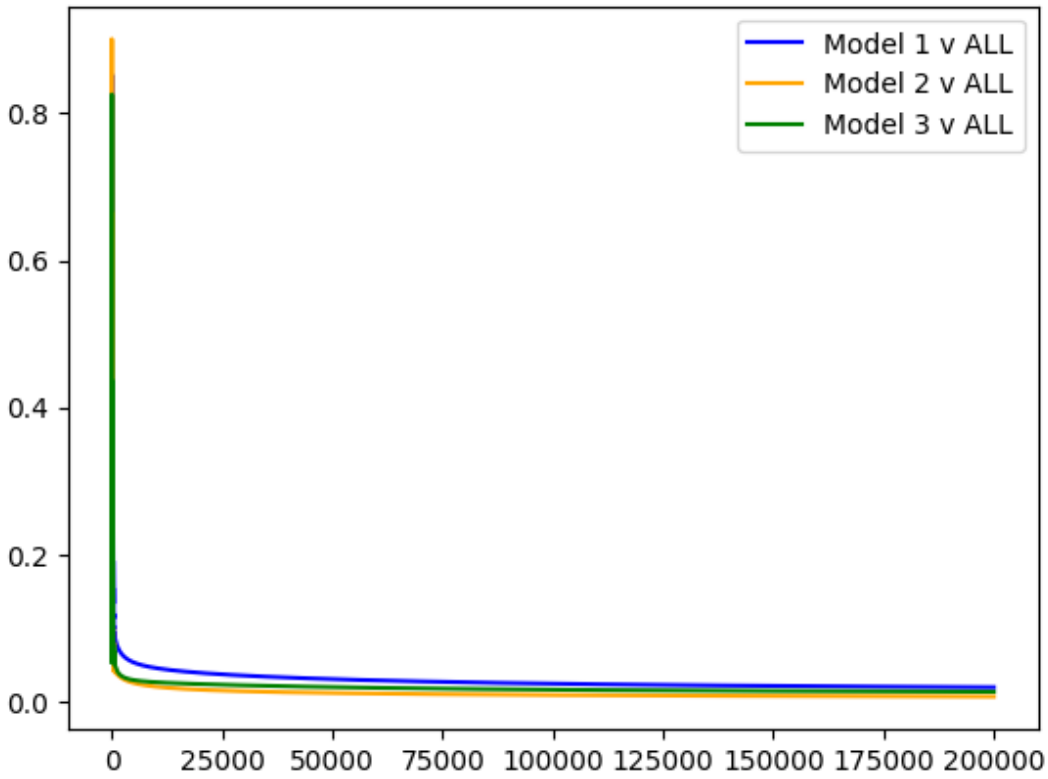
دقت اموزش:

```
0.9761904761904762
```

دقت تست :

```
0.9285714285714286
```

نمودار هزینه :



تعداد ایپاک را 200000 را برای هر مدل در نظر گرفتیم و مقدار الفا را 0.07 .

در مجموع روش سافت مکس بهتر از دو روش دیگر عمل کرده است ودر مجموع نسبت به تست و اموزش دقت بهتری میشه گفت داره .

بخش دوم :

Naïve Bayes

در این قسمت از طبقه بندی نائیو بیز برای تشخیص اعداد که به صورت دست نویس هست استفاده میکنیم.

پیاده سازی :

def PrepareData(x,y)

از این تابع برای خواندن دیتا از فایل استفاده میکنیم و هر دیتا را به صورت یک ماتریس 28*28 در می اوریم .

def train()

از این تابع برای اموزش مدل استفاده میکنیم و طبق فرمول زیر عمل میکنیم .

Train:

$$P(F_{i,j} = f | class = c) = \frac{\#\text{ of times } F_{i,j} = f \text{ when } class = c}{\text{Total number of training examples where } class = c}$$

$$P(class = c) = \frac{\#\text{ of training examples where } class = c}{\#\text{ of training examples}}$$

برای هر کلاس دوتا ماتریس در نظر میگیریم یکی در صورتی که f(i,j) برابر صفر باشد یکی زمانی که برابر یک است .به عنوان مثال ده تا صفر داریم که در 6 تای انها f(0,0) برابر یک پس مقدار ماترس یک را در درایه f(0,0) قرار میدهیم 6/10 و به همین صورت ادامه میدهیم در مجموع 28*28*10*2 احتمال پیدا میکنیم .

def predict(list_matrix0,list_matrix1,count_class)

در فاز تست از این تابع استفاده میکنیم که ورودی ان احتمال هایی هست که پیدا کردیم در فاز اموزش و از فرمول زیر استفاده میکنیم و برای هر دیتا ده تا احتمال پیدا میکنیم و مقدار برچسب را به ان احتمالی که از همه بیشتر هست میدهیم.

Test:

$$\log(P(class)) + \log(P(f_{1,1}|class)) + \log(P(f_{1,2}|class)) + ... + \log(P(f_{28,28}|class))$$

def predicttrn(list_matrix0,list_matrix1,count_class)

مشابه تابع قبل است برای به دست اورددن دقت اموزش .

def confusion_matrix(y,y_hat)

برای به دست اوردن ماتریس کانفیوژن از تابع بالا استفاده میکنیم .

def precision(label, confusion_matrix)

از تابع بالا برای به دست اوردن precision استفاده میکنیم که فرمول ان به صورت زیر است .

$$precision : \frac{TP}{FP + TP}$$

def recall(label, confusion_matrix)

از فرمول زیر استفاده کردیم :

$$recall = \frac{TP}{FN + TP}$$

برای به دست اوردن f1 هم از فرمول زیر استفاده میکنیم :

```
F1 = 2 * (precision * recall) / (precision + recall)
```

دقت اموزش :

```
accuracy_trn: 0.8442
```

دقت تست :

```
accuracy_test: 0.774
```

ماتریس کانفیوژن :

```
array([[ 76.,   0.,   1.,   0.,   1.,   5.,   3.,   0.,   4.,   0.],
       [  0., 104.,   1.,   0.,   0.,   2.,   1.,   0.,   0.,   0.],
       [  1.,   3.,  81.,   4.,   2.,   0.,   6.,   1.,   5.,   0.],
       [  0.,   1.,   0.,  80.,   0.,   3.,   2.,   7.,   1.,   6.],
       [  0.,   0.,   1.,   0.,  80.,   1.,   4.,   1.,   2.,  18.],
       [  2.,   1.,   1.,  12.,   3.,  63.,   1.,   1.,   2.,   6.],
       [  1.,   4.,   5.,   0.,   3.,   6.,  70.,   0.,   2.,   0.],
       [  0.,   5.,   4.,   0.,   3.,   0.,   0.,  77.,   3.,  14.],
       [  1.,   1.,   3.,  14.,   3.,   8.,   0.,   1.,  62.,  10.],
       [  1.,   1.,   0.,   3.,   9.,   2.,   0.,   2.,   1.,  81.]])
```

مقادیر precision recall F1:

label	precision	recall	F1
0	0.927	0.844	0.884
1	0.867	0.963	0.912
2	0.835	0.786	0.810
3	0.708	0.800	0.751
4	0.769	0.748	0.758
5	0.700	0.685	0.692
6	0.805	0.769	0.787
7	0.856	0.726	0.786
8	0.756	0.602	0.670
9	0.600	0.810	0.689

If you multiply many small probabilities you may run into problems with numeric precision, what is the problem? To handle it, we recommend that you compute the logarithms of the probabilities instead of the probabilities. Explain why this approach can help?

در ضرب تعدادی زیادی احتمال که اعدادی بین صفر و یک هستند، ممکن است جواب نهایی خیلی کوچک شود و به سمت صفر میل کند .در این صورت در محاسبات دچار مشکل خواهیم شد. به همین دلیل برای حل این مشکل از تکنیک لگاریتم گیری استفاده میکنیم تا اعمال ضرب به جمع تبدیل شده و حاصل به صفر میل نکند.

زمانی که از لگاریتم استفاده میکنیم ان احتمال هایی که مقدار صفر هستند مشکل ساز هستند برای جلوگیری از این مشکل باید مقداری به تمام احتمال ها اضاف کنیم که در زمان حساب کردن احتمال در صورت یک k و در مخرج nk اضاف میکنیم که ما در این تمرین مقدار k را برابر با 0.001و مقدار n را 2 قرار میدهیم که به این روش لاپلاس اسموتینگ میگویند .

What is the base assumption of the Naïve Bayes classifier? Why is it important?

فرض اصلی کلاسیفایر bayes naïve مستقل بودن ویژگی ها از هم دیگر و عدم تاثیر یک ویژگی بر ویژگی دیگر است. مزیت این فرض این است که این کلاسیفایر از دیگر مدل ها مانند Regression Logistic بهتر عمل کرده، دیتای کمتری نیاز خواهد داشت و سریع تر همگرا خواهد شد.