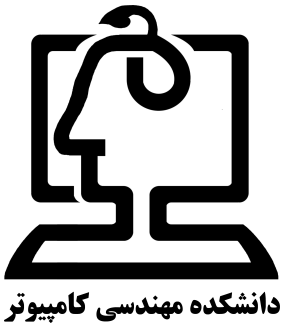
به نام خدا



دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر

استاد درس: دکتر امین ناظرفرد

پاییز ۱۳۹۹

درس یادگیری ماشین

سروش مهدی
شماره دانشجویی: ۹۹۱۳۱۰۵۰

سری ۳ تمرین‌ها

فهرست مطالب

۱ سوالات تشریحی

۱.۱ Smoothing

در بیز ساده در شرایطی که برای مثال یکی از ویژگی ها برای یک مقدار خاص مقدار likelihood صفر برای یکی از کلاس ها داشته باشد این موضوع باعث میشود که کل احتمال ثانویه ان کلاس برای دسته از مقادیری که این مقدار خاص از این ویژگی خاص را دارند صفر شود و در نتیجه باعث میشود اطلاعات زیادی در مورد سایر ویژگی ها در این حالت نادیده گرفته شود. برای حل این مشکل ما از smoothing استفاده میکنیم و احتمالات likelihood را طوری تغییر میدهیم که هیچ کدام صفر نباشند. برای این منظور از فرمول زیر برای بدست اوردن این احتمالات استفاده میکنیم.

$$P(X_i = x_{ik}|Y = y_i) = \frac{Count(X_i = x_{ik}, Y = y_i) + L}{Count(Y = y_j) + LK} \tag{1}$$

در این فرمول K تعداد مقادیر ممکن ویژگی i ام میباشد و L ضریب smoothing است.

۲.۱ LR vs NBC

بیز ساده یک مدل Generative و رگرسیون لاجستیک یک مدل discriminative هست. هنگامی که مقدار داده های اموزش زیاد باشد رگرسیون لاجستیک عملکرد بهتری دارد و یکی از دلایل ان این است که بیز ساده فرض مستقل بودن ویژگی ها از یکدیگر را دارد. همچنین در نتیجه این فرض اگر بین ویژگی ها همبستگی داشته باشیم دسته بند بیز ساده عملکرد خوبی نخواهد داشت. در صورت داشتن داده های کم اما عملکرد بیز ساده بهتر است چون این مدل سریع تر از مدل رگرسیون لاجستیک به تقریب قابل قبولی میرسد. اما رگرسیون لاجتستیک در صورت کم بودن داده ها دچار بیش برازش میشود که البته میتوان به وسیله رگولاریزیشن جلوی این موضوع را گرفت. اگر تعداد داده ها خیلی خیلی زیاد باشپ اما رگرسیون لاجستیک و دسته بند بیز ساده گاوسی نتیجه یکسانی خواهند داشت.
یکی از برتری های های بیز ساده این است که مرحله بهینه سازی ندارد. در رگرسیون لاجستیک میتوانیم با استفاده از رگولاریزیشن ویژگی های با درجه بالاتر از ویژگی حال حاضر استخراج کنیم و به مدل اضافه کنیم اما این کار در بیز ساده خراب شدن مدل میشود. هر دو مدل پارامتریک هستند.

۳.۱

باید بین دو عبارت زیر عبارتی که مقدار بیشتری دارد انتخاب شود.

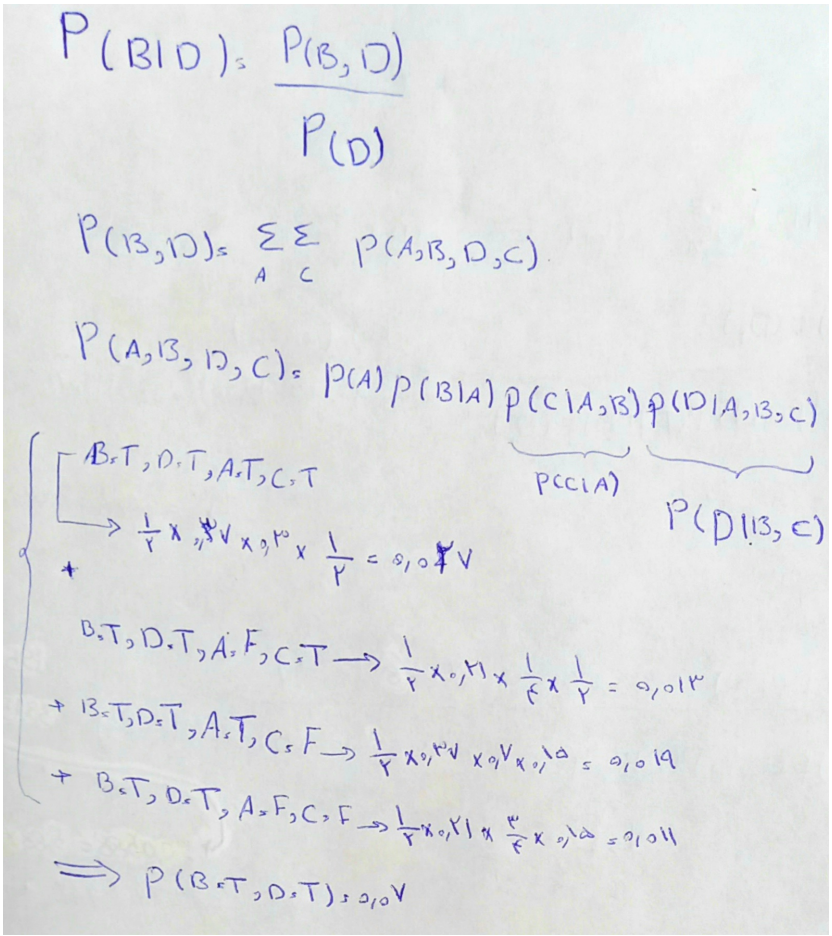
$$P(X_1|B)P(X_2|B)P(B) \tag{2}$$

$$P(X_1|A)P(X_2|A)P(A) \tag{3}$$

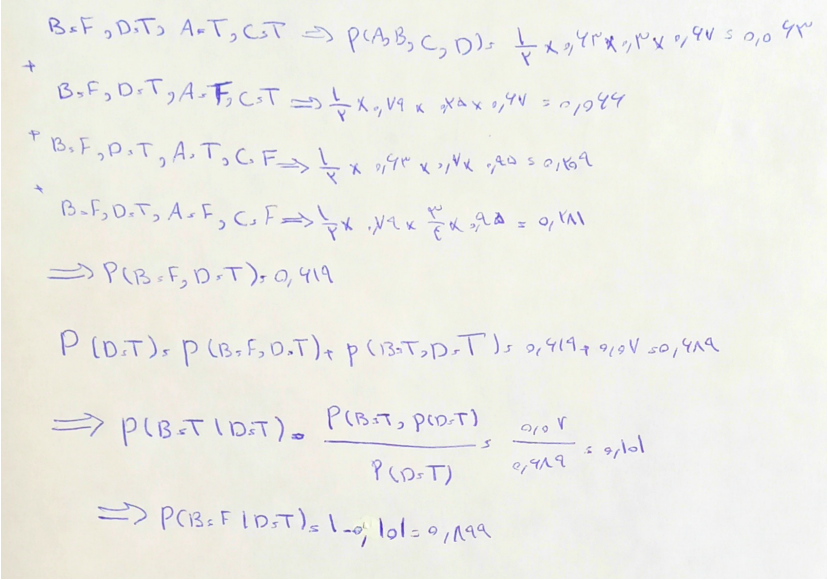
از طریق شمارش متوجه میشویم که احتمال اولیه کلاس B از کلاس A بیشتر است. همچنین روی محور عمودی اگر توزیع نرمال برای داده ها در نظر بگیریم هر دو کلاس میانگین تقریبا یکسانی دارند که تقریبا برابر با ویژگی X2 داده تست است اما کلاس A واریانس و پراکندگی بیشتری دارد و کلاس دیگر پراکندگی کمتر دارد و در نتیجه احتمال بیشتری نزدیک به میانگین خود دارد پس ترم دوم در عبارت مربوط به کلاس B مقدار بیشتری

دارد . همچنین در ارتباط با محور افقی نیز در این محور واریانس کلاس A نزدیک به صفر است که در توزیع نرمال باعث میشود در نقاطی به غیر از میانگین احتمال بسیار کوچک و نزدیک به صفر باشد پس در رابطه با ترم اول دو عبارتی که میخواهیم مقایسه کنیم نیز ترم مربوط به کلاس B بیشتر است پس در نتیجه عبارت اول بزرگ تر و داده مربوط به کلاس B دسته بندی میشود.

۴.۱



شکل ۱: حل سوال چهار



شکل ۲: حل سوال چهار

۵.۱

نحوه انتخاب این نقطه بستگی به این دارد که کدام نوع خطا برای ما هزینه بیشتری دارد و نسبت این هزینه ها به چه صورت است ما نسبت این خطاها را میتوانیم به ازای نقاط مختلف cut-off در نمودار ROC مشاهده کنیم. مثلا اگر هزینه خطای FN بیشتر باشد ما ترجیح میدهیم نقطه cut-off را پایین تر بیاوریم تا میزان این نوع خطا کمتر شود اما این مقدار را تا جایی پایین می اوریم که دیگر خطا با توجه به هزینه ای که دارد نیز مقدار قابل قبولی داشته باشد.

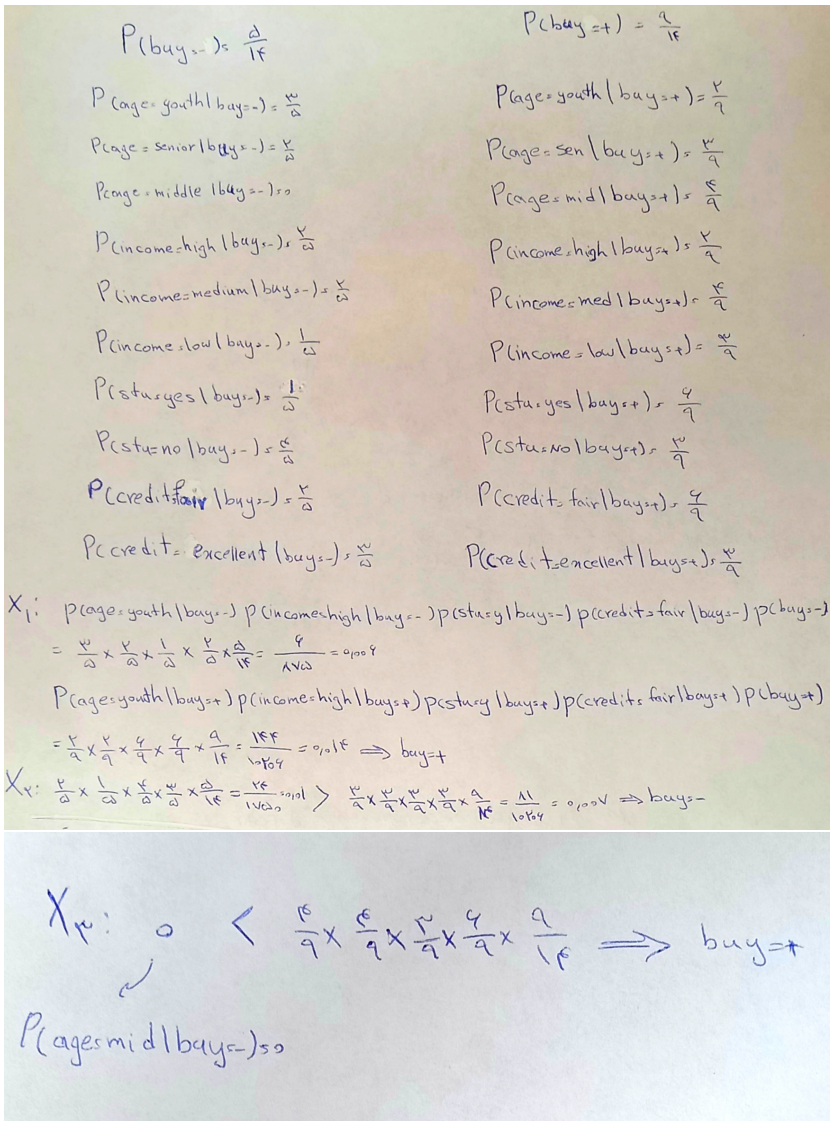
۶.۱

بخت یک رویداد برابر است با نسبت احتمال رخ دادن ان رویداد به احتمال رخ ندادن ان رویداد.نسبت بخت یک معیار برای اندازه گیری تاثیرگذار بودن یا نبودن یک متغیر تصادفی بر متغیر تصادفی دیگر است. این نسبت برابر است با نسبت بخت یک رویداد در حالتی که مقدار متغیر تصادفی که میخواهیم تاثیر ان را اندازه بگیریم برابر با مقدار اول باشد به بخت همان رویداد هنگامی که این مقدار برابر با مقدار دوم باشد. در رگرسیون لاجستیک نسبت بخت برای دو مقدار مختلف x برابر

$$e^{w(x_1-x_2)} \tag{4}$$

میباشد که میتوانیم بررسی کنیم تغییرات x چه تاثیری روی کلاس خروجی دارد.

۷.۱



شکل ۳: حل سوال هفت

۲ سوالات پیاده سازی

۱.۲ NBC

۱.۱.۲

کد این بخش با نام 2a.py ذخیره شده

```
#loading data
data = pd.read_csv('car.data', sep=",", header=None)
data.columns = ['buying' , 'maint' ,'doors'  , 'persons' ,'lug_boot','safety','class']
#shuffling
data = data.sample(frac = 1).reset_index(drop=True)
data = data.astype('category')
#splitting data to train and test
train = data[:int(len(data)*0.7)]
test = data[int(len(data)*0.7):]
```

Listing :۱

در این قسمت از کد داده ها را بارگزاری کردیم و چون داده ها دارای ترتیب بودند ابتدا ان ها را شافل کردیم و سپس به دو دسته اموزش و ازمون تقسیم کردیم
در صفحه بعد کد دو تابع امده است

تابع اول برای محاسبه احتمالات است که در این تابع ابتدا احتمال های اولیه را حساب میکنیم سپس احتمالات لایکلیهود را برای هر ویژگی و بر اساس مقادیر ان مشخص میکنیم و در نهایت این احتمالات را در یک ساختمان داده دیکشنری ذخیره میکنیم.

تابع دوم برای پیش بینی کلاس یک نمونه است که با استفاده از فرمول احتمال ثانویه و با استفاده از احتمالات اولیه و لایکلیهود که توسط بیز ساده محاسبه کردیم توسط این تابع کلاس مربوط به هر داده را پیش بینی میکنیم.

```python
def NBC(train , smoothing = 0):

    """calculate priors and likelihoods of training setfor naive bayes

    Returns:
        [dict]: [contains probabilities for priors and likelihoods]
    """
    probs = {}
    tables = {}
    #calculating priors
    for class_ in train['class'].unique():
        probs[class_] = train['class'].value_counts()[class_]/train.shape[0]
        #we use this tables for calculating likelihoods
        tables[class_] = train[train['class'] == class_]

    #calculating likelihoods for each feature and its value and target class
    for feature in train.columns[:-1]:
        probs[feature]={}
        k = len(train[feature].unique())
        for value in train[feature].unique():
            probs[feature][value]={}
            for class_ in train['class'].unique():
                a = (tables[class_][feature].value_counts()[value] + smoothing)
                b = (tables[class_].shape[0] + smoothing*k )
                probs[feature][value][class_] = a/b

    return probs
```

Listing :۲

```python
def predict(row , probs):
    """predict target class for row

    Args:
        row (pandas series): a sample in dataset
        probs (dict): [probabilities for NBC]

    Returns:
        [int]: [predicted class (0 to 3 which 0 is acc , 1 is unacc , 2 is vgood , 3 is good)]
    """
    posteriors = []
    #calculating each postrior
    for i in['acc' , 'unacc' , 'vgood' , 'good']:
        p = probs[i]
        for feature in ['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety']:
            p *= probs[feature][row[feature]][i]
        posteriors.append(p)

    return posteriors.index(max(posteriors))
    #acc = 0  , unacc = 1  , vgood = 2   , good = 3
```

Listing :۳

```python
def confusion(y_true , y_pred):
    """a function for calculating confusion matrix for multiclass problems


    """
    conf = np.zeros((4,4))
    for tr_la in range(4):
        for pr_la in range(4):
            true = y_true == tr_la
            pre = y_pred[true]
            conf[tr_la,pr_la]=np.count_nonzero(pre==pr_la)

    return conf.astype('int64')
```

Listing :۴

از این تابع برای محاسبه ماتریس پریشانی چند کلاسه استفاده میکنیم که سطر های ماتریس مقادیر واقعی و ستون های ان مقادیر پیش بینی شده میباشند و هر سطر و ستون از صفر شروع میشود.
ما دز این سوال کلاس acc را به عنوان کلاس صفر و کلاس unacc را به عنوان کلاس یک و کلاس vgood را به عنوان کلاس دو و کلاس good را به عنوان کلاس سه در نظر گرفتیم.

```python
#calculating prior and likelihoods of train data
probs = NBC(train)

#adding predict column and a column for changing categorical target calss to numerical values
test = test.assign (predict  = test.apply(lambda row:predict(row , probs),axis = 1).values)
test =test.assign(true =  test.apply(lambda row:true(row) , axis = 1).values)

train = train.assign (predict  = train.apply(lambda row:predict(row , probs),axis = 1).values)
train =train.assign(true =  train.apply(lambda row:true(row) , axis = 1).values)

#changing predict column and true column type to numpt
y_test_true = test['true'].to_numpy()
y_test_pred = test['predict'].to_numpy()

y_train_true = train['true'].to_numpy()
y_train_pred = train['predict'].to_numpy()

#calculating confusion matrix
conf_test = confusion(y_test_true , y_test_pred)
conf_train = confusion(y_train_true , y_train_pred)
```

Listing :۵

کد محاسبه مقادیر خواسته شده در صفحه بعد امده است توجه کنید که این مقادیر به صورت برداری محاسبه شده اند و هر یک برداری به طول ۴ هستند که برای هر کلاس به صورت جداگانه مقادیر را دارا میباشند

```python
#calculating FN , FP , sensitivity , specificity
#column sums  - diagonal
fp_test = conf_test.sum(axis = 0) - np.diag(conf_test)
fp_train = conf_train.sum(axis = 0) - np.diag(conf_train)
#row sums - diagonal
fn_test = conf_test.sum(axis = 1) - np.diag(conf_test)
fn_train = conf_train.sum(axis = 1) - np.diag(conf_train)
sens_test = np.diag(conf_test) / (np.diag(conf_test) + fn_test)
sens_train = np.diag(conf_train) / (np.diag(conf_train) + fn_train)
tn_test = -(fp_test + fn_test + np.diag(conf_test)) + conf_test.sum()
tn_train = - (fp_train + fn_train + np.diag(conf_train)) + conf_train.sum()
spec_test = (tn_test / (tn_test + fp_test)).reshape(4,)
spec_train = (tn_train / (tn_train + fp_train)).reshape(4,)
```

Listing :۶

```
test data:

confusion matrix:
[[ 91  20   0   1]
 [ 20 338   0   1]
 [ 13   0  12   0]
 [ 18   0   0   5]]
positive class:  0  (acc )
FN :  21
FP :  51
 sensitivity:  0.8125
 specificity:  0.8746928746928747

positive class:  1  (unacc )
FN :  21
FP :  20
 sensitivity:  0.9415041782729805
 specificity:  0.875

positive class:  2  (vgood )
FN :  13
FP :  0
 sensitivity:  0.48
 specificity:  1.0

positive class:  3  (good )
FN :  18
FP :  2
 sensitivity:  0.21739130434782608
 specificity:  0.9959677419354839
```

```
train data:

confusion matrix:
[[211  55   0   8]
 [ 36 800   0   0]
 [ 18   0  29   0]
 [ 35   0   1  16]]
positive class:  0  (acc )
FN :  63
FP :  89
 sensitivity:  0.7700729927007299
 specificity:  0.9048128342245989

positive class:  1  (unacc )
FN :  36
FP :  55
 sensitivity:  0.9569377990430622
 specificity:  0.8525469168900804

positive class:  2  (vgood )
FN :  18
FP :  1
 sensitivity:  0.6170212765957447
 specificity:  0.9991394148020654

positive class:  3  (good )
FN :  36
FP :  8
 sensitivity:  0.3076923076923077
 specificity:  0.993085566119274
```

شکل ۴: نتایج خواسته شده

۲.۱.۲

کد این بخش با نام 2b.py ذخیره شده

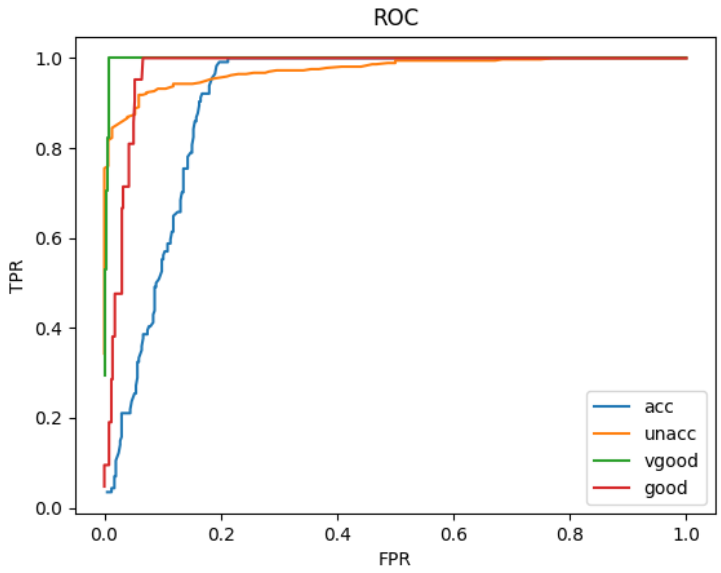
```
test data:

confusion matrix:
[[ 73  28   0   5]
 [ 23 350   0   2]
 [ 14   0   7   0]
 [ 11   0   2   4]]
positive class:  0  (acc )
FN :  33
FP :  48
 sensitivity:  0.6886792452830188
 specificity:  0.8837772397094431

positive class:  1  (unacc )
FN :  25
FP :  28
 sensitivity:  0.9333333333333333
 specificity:  0.8055555555555556

positive class:  2  (vgood )
FN :  14
FP :  2
 sensitivity:  0.3333333333333333
 specificity:  0.9959839357429718

positive class:  3  (good )
FN :  13
FP :  7
 sensitivity:  0.23529411764705882
 specificity:  0.9860557768924303
```

```
train data:

confusion matrix:
[[195  71   0   9]
 [ 31 811   0   1]
 [ 24   0  18   0]
 [ 34   0   0  15]]
positive class:  0  (acc )
FN :  80
FP :  89
 sensitivity:  0.7090909090909091
 specificity:  0.9047109207708779

positive class:  1  (unacc )
FN :  32
FP :  71
 sensitivity:  0.9620403321470937
 specificity:  0.8060109289617486

positive class:  2  (vgood )
FN :  24
FP :  0
 sensitivity:  0.42857142857142855
 specificity:  1.0

positive class:  3  (good )
FN :  34
FP :  10
 sensitivity:  0.30612244897959184
 specificity:  0.9913793103448276
```

شکل ۵: نتایج خواسته شده

۳.۱.۲

کد این قسمت با نام 2c.py ذخیره شده



شکل ۶: نمودار خواسته شده

اگر حد استانه را صفر قرار دهیم همه داده ها مثبت دسته بندی میشوند و در این نمودار نقطه ۱ و ۱ بدست می اید اما با افزایش تدریجی حد استانه FPR کمتر میشود زیرا تعداد داده هایی که مثبت دسته بندی میکنیم کمتر میشود همچنین میبینیم کلاس هایی که داده کمتری در دیتاست دارند زودتر به TPR برابر یک میرسند و دسته بندی کننده برای این کلاس ها عملکرد بهتری دارد.

۲.۲ Regression Logistic

۱.۲.۲

کد این بخش و بخش بعدی با نام 3.py ذخیره شده است

ما از این دو تابع که در صفحه بعد امده اند برای الگوریتم one vs all استفاده میکنیم در تابع اول برای هر کلاس به صورت جداگانه یک مدل اموزش میدهیم به این صورت که کلاس مورد نظر را به عنوان کلاس مثبت و بقیه را منفی در نظر میگیریم. از تابع دوم نیز برای پیش بینی به این صورت که کدام کلاس با توجه به مدل احتمال بیشتری دارد استفاده میشود.

```python
def one_vs_all(x_train , y_train):
    models = []
    for positive_class in range(10):

        #seperaing positive labels
        y_train_positive = (y_train == positive_class).astype('uint8')
        #initializing model
        model = LogisticRegression( multi_class = 'ovr' , solver = 'lbfgs' , max_iter =500)
        print('starting training for positive class = ' , positive_class)
        #fitting model
        model.fit(x_train, y_train_positive)
        models.append(model)
        print('training for positive class = ' , positive_class , ' has ended')
    return models

def predict(models , x):
    """this function calculates which class has the biggest probability

    Args:
        models ([type]): [description]
        x ([type]): [description]

    Returns:
        [type]: [description]
    """
    x = x.reshape(1 , 784)
    max_prob = 0
    max_index = 0

    for i in range(10):
        if models[i].predict_proba(x)[0,1]>max_prob:
            max_prob = models[i].predict_proba(x)[0,1]
            max_index = i


    return max_index
```

Listing :۷

```python
#fetching mnist dataset
mnist = fetch_openml('mnist_784')
x = mnist.data
y = mnist.target

#scaling
x = x/255
y = y.astype('uint8')
x = x.astype('float32')

#splitting dataset into test and train
x_train = x[:60000]
y_train = y[:60000]
x_test = x[60000:]
y_test = y[60000:]
```

Listing ۸:

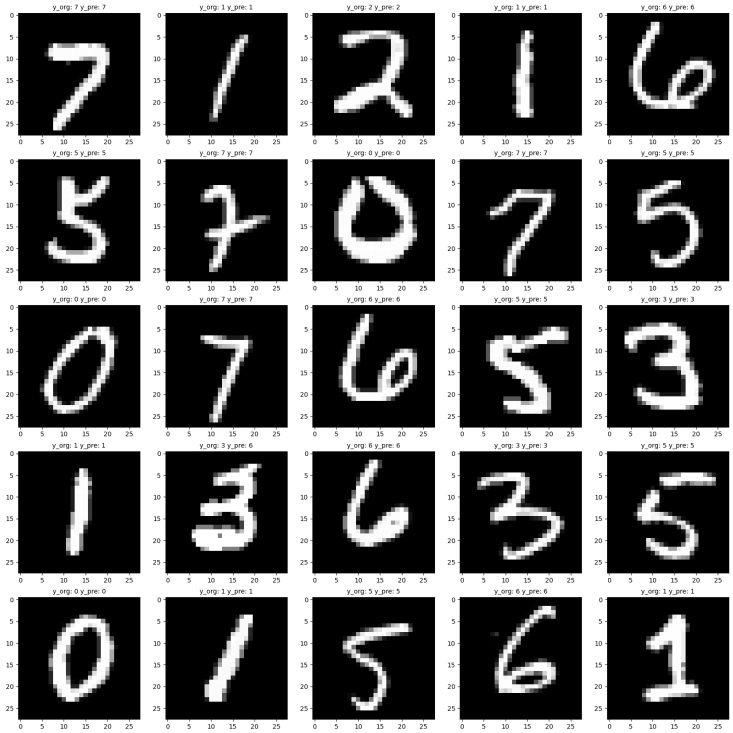
برای بارگزاری دیتاست از کتابخانه sklearn کمک گرفتیم و همچنین داده ها را نرمال کردیم و ۶۰۰۰۰ داده به عنوان اموزش و ۱۰۰۰۰ به عنوان تست جدا کردیم.
نتایج خواسته شده در صفحه بعد امده اند.

```
test accuracy:  0.9198
train accuracy:  0.9278833333333333
confusion matrix for train data:
[[5801    1   12    8    7   17   28    7   36    6]
 [   1 6584   32   14    4   24    5   12   58    8]
 [  39   55 5424   82   58   20   54   66  143   17]
 [  20   28  147 5499   10  167   24   52  122   62]
 [   9   25   29    6 5491    9   32   13   54  174]
 [  49   25   36  162   58 4802   98   16  119   56]
 [  28   13   31    2   24   80 5708    3   27    2]
 [  14   23   60   12   55   11    3 5906   18  163]
 [  41  133   63  133   40  142   44   25 5142   88]
 [  31   25   21  101  179   42    2  186   46 5316]]

confusion matrix for test data:
[[ 959    0    1    2    0    5    6    4    1    2]
 [   0 1112    3    1    0    1    5    1   12    0]
 [   9    8  920   20    9    4   10   11   37    4]
 [   4    0   17  918    2   23    4   12   21    9]
 [   1    2    5    3  915    0   10    2    6   38]
 [  10    2    0   42   10  769   17    7   28    7]
 [   9    3    7    2    6   20  907    1    3    0]
 [   2    7   22    5    8    1    1  951    5   26]
 [  10   14    5   21   15   28    8   11  850   12]
 [   8    8    2   13   31   14    0   24   12  897]]
```

شکل ۷: مقادیر خواسته شده

در ماتریس های پریشانی ردیف ها نشان دهنده مقدار واقعی و ستون ها نشان دهنده مقدار پیشبینی شده میباشند که شماره هر ردیف و ستون از صفر شروع شده و هر شماره نشان دهنده کلاس مورد نظر میباشد.

۲.۲.۲



شکل ۸: مقادیر پیشبینی شده برای ۲۵ داده تصادفی

۳.۲.۲

رگرسیون لاجستیک یک جدا کننده خطی است و برای شرایطی که کلاس ها خطی جدا پذیر نیستند مناسب نیست اما روش KNN برای شرایطی که کلاس ها خطی جداپذیر نیستند نیز ممکن است عملکرد خوبی داشته باشد. KNN یک مدل غیر پارامتریک است اما مدل رگرسیون لاجستیک پارامتریک است روش رگرسیون لاجستیک سرعت بیشتری دارد و میتواند احتمال پیش بینی کند. پس اگر کلاس ها خطی جدا پذیر باشند احتمالا روش بهتری میباشد.