به نام خدا

مباحث ویژه 1 | تمرین دوم | رضا علیدوست 96461334

جواب 1)

الف) برای مدل logistic regression یک نورون داریم که به ازای هر feature یک وزن داریم و یک bias داریم و پس از ترکیب خطی featureها با وزن هاییشان و bias ، مقدار به دست آمده را در داخل یک activation function که اینجا از تابع sigmoid استفاده کرده ایم قرار میدهیم و با مرز تصمیم 0.5 دیتا ها را کلاس بندی میکنیم. برای یادگیری این مدل نیاز هست این پارامترها را به گونه ای بدست آوریم که مقدار loss function به کمینه برسد. در صورت سوال اشاره شده از روش mini-batch gradient descent(batch-size = 2) برای بهینه سازی مدل استفاده کنیم. در زیر مراحل یادگیری مدل و نتایج بدست آمده را بررسی میکنیم.

قبل از شروع مراحل دو ماتریس X و Y را که دیتاست ما را تشکیل میدهند میسازیم.

```
X =
[[22 25 47 52 46 56 55 60]
 [ 1  0  1  0  1  1  0  0]]
```

```
Y =
[[0 0 1 0 1 1 0 1]]
```

برای اینکه مقدار ها رنج درستی داشته باشند آنها را normalize میکنیم. (ماکسیمم مقدار سن را 100 در نظر میگیریم.)

```
X[0] = X[0] / 100
```

```
X =
[[0.22 0.25 0.47 0.52 0.46 0.56 0.55 0.6 ]
 [1.   0.   1.   0.   1.   1.   0.   0.  ]]
```

X : هر ستون شامل یک sample از dataset میباشد. که مقدار دو feature سن و داشتن کار را نشان میدهد.

Y : هر ستون شامل target برای هر sample میباشد یا همان جواب مطلوب ما برای شبکه.

مراحل:

۱) ابتدا پارامتر های مدل را با 1 مقدار دهی میکنیم.

```
w =
[[1.]
 [1.]]
```

```
b =
1.0
```

w : همان وزن featureها میباشد که وزن سن و داشتن کار را نشان میدهد.

b : همان bias میباشد.

۲) Forward pass: در این مرحله ما مقدار شبکه را با پارامتر هایی که تا این مرحله بدست آمده است predict میکنیم. نحوه انجام prediction به این صورت است که ابتدا یک batch از dataset انتخاب میکنیم. Z = (w.T).x_batch + b) که در اینجا منظور از x_batch دو ستون از X میباشد چون batch_size = 2 و w.T ترانهاده w میباشد.) و سپس A = sigmoid(Z) را بدست می آوریم. برای مثال در مرحله اول داریم :

```
w = [[1.]
 [1.]] ---- > w' = [[1. 1.]], b = 1.0

 x_batch = [[0.22 0.25]
 [1.   0.  ]]

 --------> w' . x_batch + b = [[2.22 1.25]]

 --------> Z = [[2.22 1.25]]]

 --------> A = sigmoid(Z) = [[0.9020312  0.77729986]]
```

۳) Backpropagation: در این مرحله مشتقات جزیی تابع loss function را نسبت به پارامتر ها محاسبه میکنیم که با توجه به loss function و ریاضیات مشتقات بدست آمده برابر خواهند بود : dw =1/m(x_batch . (A – y_batch)') و db =1/m(np.sum(A-y_batch)) که محاسبات برای مرحله اول داریم :

```
A = [[0.9020312  0.77729986]]
y_batch = [[0. 0.]]
x_batch = [[0.22 0.25]
 [1.   0.  ]]

dZ = A - y_batch = [[0.9020312  0.77729986]]

db = (1/2) np.sum(dZ) = 0.8396655284385686

dw =1/2 (x_batch . dZ = ') [[0.19638591]
 [0.4510156 ]]
```

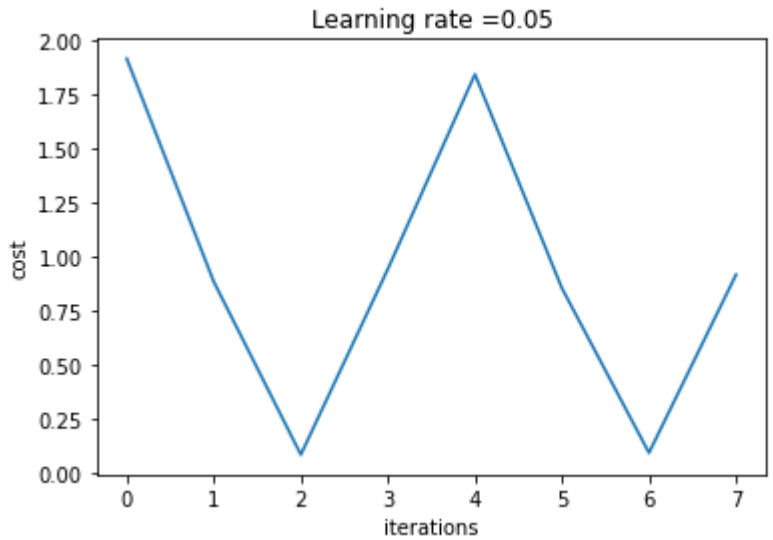
۴) Optimization : با توجه به الگوریتم gradient descent در این مرحله گرادیان های بدست امده را از مقدار های قبلی w و b کم میکنیم تا در مسیر بهینه ای این مقادیر آپدیت شده و از مقدار loss function کاسته شود. که داریم: w = w – learning_rate*dw و b = b – learning_rate * db .

```
current w = [[1.]
 [1.]]

after update w = w - 0.05 * [[0.19638591]
 [0.4510156 ]] =  [[0.9901807 ]
 [0.97744922]]

current b = 1.0

after update b = b - 0.05 * 0.8396655284385686 = 0.9580167235780715
```

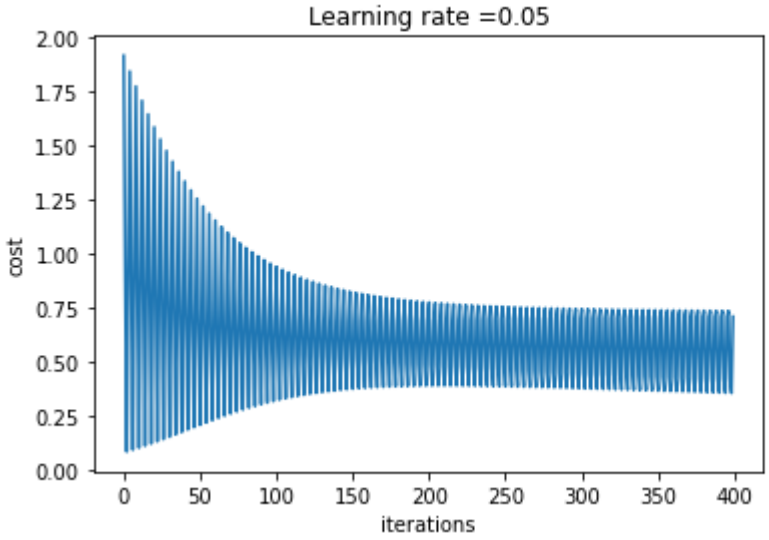
بعد از آپدیت مقدار loss function را هم حساب میکنیم که داریم :

```
cost in epoch 1 and batch [array([[0.22, 0.25],
       [1.  , 0.  ]]), array([[0., 0.]])] ----> 1.912517627913481
```

همه این مراحل برای یک batch انجام شده است. باید این مراحل را برای همه batch ها انجام دهیم تا epoch 1 انجام شود. که در صورت سوال epoch 2 خواسته شده است. می توان نتیجه گرفت که در stochastic gradient descent البته اینجا mini-batch هست همواره cost function کاهش پیدا نمیکند و حالت نویزی دارد ولی در مجموع به سمت کاهش میرود. در زیر نمودار cost function را پس از هر مرحله اپدیت مشاهده میکنیم :



که با زیاد کردن تعداد اپدیت ها مشاهده میکنیم :



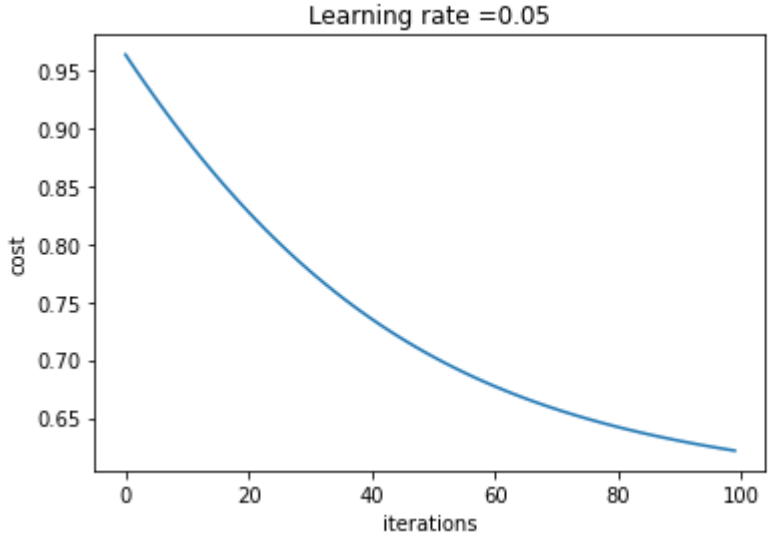
که همان نتیجه نویزی بودن و اکیدا نزولی نبودن را میگیریم ولی در کل کاسته میشود زیرا هر بار batch تغییر میکند و cost(loss) روی batch جدید train میشود.

با انجام مراحل به تعداد epoch 2 به پارامتر های زیر خواهیم رسید که با پیاده سازی logistic regression در فایل mini_batch.ipynb بدست آمده است:

```
w =  [[0.9901807 ]
 [0.97744922]]

b = 0.9580167235780715

cost in epoch 1 and batch [array([[0.22, 0.25],
       [1.  , 0.  ]]), array([[0., 0.]])] ----> 1.912517627913481

w =  [[0.98058173]
 [0.97952692]]

b = 0.9397569346426939

cost in epoch 1 and batch [array([[0.47, 0.52],
       [1.  , 0.  ]]), array([[1., 0.]])] ----> 0.8830430989624943

w =  [[0.98265798]
 [0.98361602]]

b = 0.943846043069526

cost in epoch 1 and batch [array([[0.46, 0.56],
       [1.  , 1.  ]]), array([[1., 1.]])] ----> 0.08532865629509048
```

```
w =  [[0.97411103]
 [0.98361602]]

b = 0.9279027536376557

cost in epoch 1 and batch [array([[0.55, 0.6 ],
       [0.  , 0.  ]]), array([[0., 1.]])] ----> 0.9419989440314482

w =  [[0.96442613]
 [0.96128133]]

b = 0.886482970427297

cost in epoch 2 and batch [array([[0.22, 0.25],
       [1.  , 0.  ]]), array([[0., 0.]])] ----> 1.839978382876785

w =  [[0.95509229]
 [0.96355724]]

b = 0.8687521252396183

cost in epoch 2 and batch [array([[0.47, 0.52],
       [1.  , 0.  ]]), array([[1., 0.]])] ----> 0.8531203105894013


 w =  [[0.95736748]
  [0.96803748]]

 b = 0.8732323619491512

 cost in epoch 2 and batch [array([[0.46, 0.56],
        [1.  , 1.  ]]), array([[1., 1.]])] ----> 0.09388556041516327

 w =  [[0.94919337]
  [0.96803748]]

 b = 0.8579376962516984

 cost in epoch 2 and batch [array([[0.55, 0.6 ],
        [0.  , 0.  ]]), array([[0., 1.]])] ----> 0.9157069369050892
```

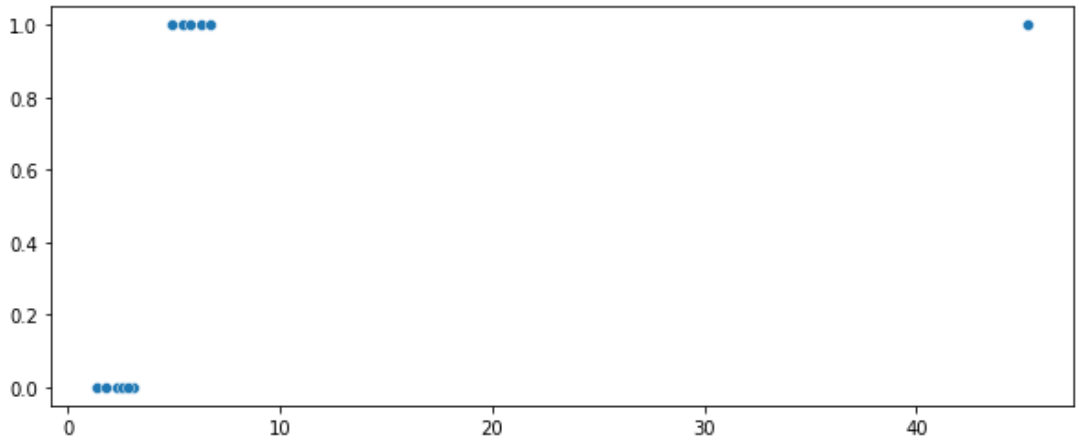
ب) اگر سوال بالا را در batch gradient descent حل میکردیم داشتیم :



که نشان میدهید همواره کاهشی میباشد.(مگر اینکه learning_rate درست انتخاب نشده باشد) با توجه به اینکه sgd حالت نویزی دارد. Sgd برا مسایلی که dataset بزرگی داریم مناسب است زیرا نیاز نیست که همه دیتاست را ببیند تا یک مرحله آپدیت شود در نتیجه سریعتر از gd همگرا میشود و به محاسبات کمتری نیازمند است. معمولا sgd نسبت به gd بهتر از local minimum میتواند فرار کند (به دلیل stochastic بودنش) ولی gd میتواند به راحتی گیر کند. پس مشکل gd زیاد بودن محاسبات و طولانی بودن همگرایی و گیر کردن در local optimum میباشد. نکته مثبت gd این است که در محاسبات از vectorizing به بهترین شل ممکن استفاده میشود و در iterationهای برابر gd بهتر و سریعتر عمل میکند چون کاملا vectorized است و کتابخانه ها به خوبی از parallelism استفاده کردن برای پیاده سازی vectorها که این نکته در sgd نادیده گرفته میشود و به ازای هر input جدا گانه محاسبه میکنم و از vectorizing به خوبی استفاده نمیشود که باعث میشود کند تر شود . پس برای اینکه هم مشکلات gd را کمتر داشته باشیم و هم مشکلات sgd را میتوانیم از mini-batch gd استفاده کنیم که هم از vectorzing تا حدودی بهره ببریم و هم از ویژگی های sgd غافل نشویم. و بهتر است batch_size را توانی از 2 انتخاب کنیم (به علت ساختار GPU) تا سرعت بهتری حاصل شود.

جواب 2)

ابتدا دیتای داده شده را در صفحه مختصات رسم میکنیم که دیتای مورد نظر را در زیر میبینیم.

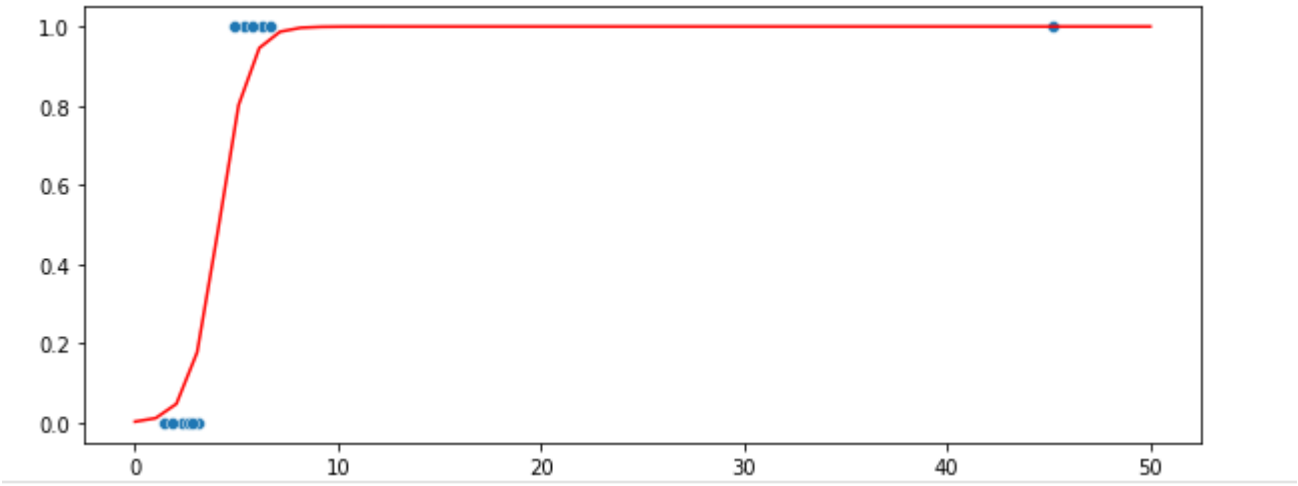


حال به کمک دو روش logistic regression و linear regression دیتای مورد نظر را روی مدل ها fit خواهیم کرد. در روش logistic regression ما از activation function که اینجا از sigmoid استفاده کردیم و همچنین loss function ما از نوع log میباشد که با fit کردن دیتا مدل زیر را داریم :

$$z = w^T . x + b$$

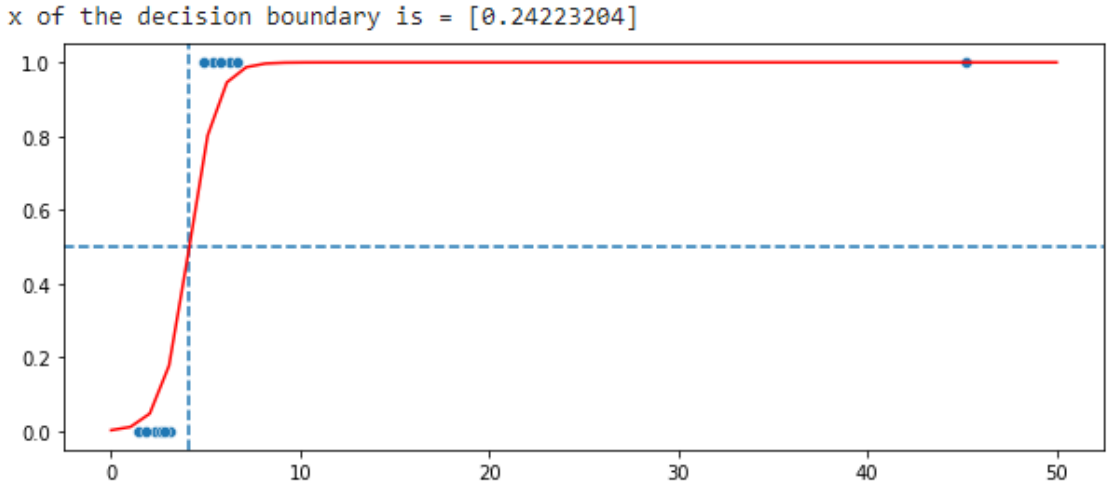
$$A = sigmoid(z) = \frac{1}{1 + e^{-z}}$$

که اگر رسم کنیم A را به این شکل میرسیم.



برای بدست اوردن decision boundary با توجه به اینکه threshold = 0.5 میباشد باید معادله sigmoid(z) = 0.5 را حل کنیم که به جواب z = 0 میرسیم و از آن هم به جواب $x = \frac{-b}{w}$ میرسیم. که برای مدل بالا جواب برابر است با : $x = 0.24$

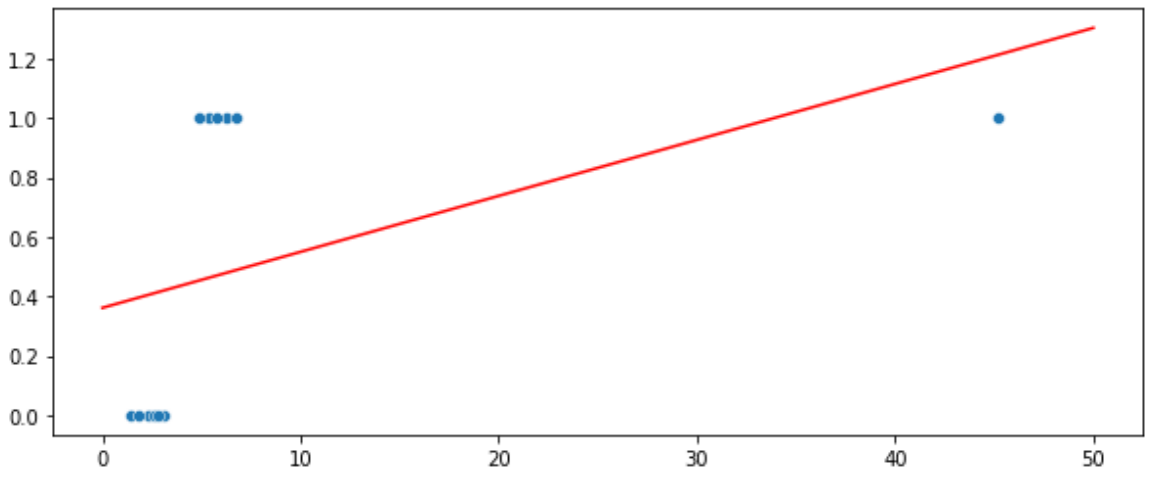
شکل زیر نشان دهنده decision boundary میباشد که نشان میدهد این مدل دیتا را به خوبی کلاس بندی کرده است:



در روش linear regression ما loss function را معمولا از نوع mean square error function میباشد که با fit کردن دیتا مدل زیر را داریم :

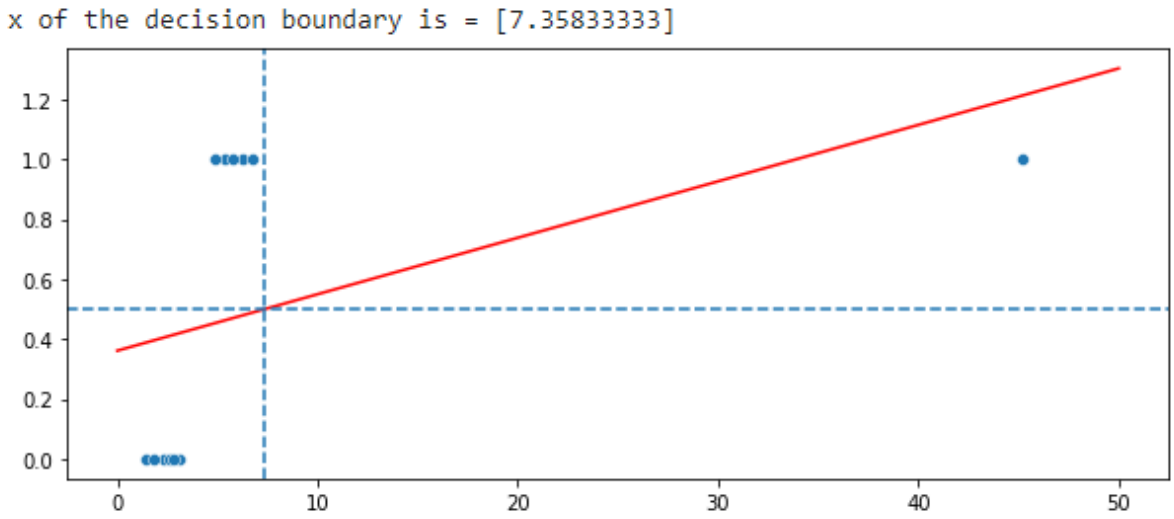
$$z = w^T . x + b$$

که اگر رسم کنیم z را به این شکل میرسیم.



برای بدست اوردن decision boundary با توجه به اینکه threshold = 0.5 میباشد باید معادله z = 0.5 را حل کنیم که به جواب $x = \frac{-b+0.5}{w}$ میرسیم. که برای مدل بالا جواب برابر است با :

$$x = 7.36$$



با توجه به شکل بالا میتوانیم متوجه شویم که مدل بالا بخاطر وجود داشتن یک دیتای پرت نتوانسته به خوبی همه دیتا را کلاس بندی کند.

در فایل logreg_vs_linreg.ipynb دو مدل با استفاده از کتابخانه sklearn پیاده سازی شده است.

با مقایسه نتایج بدست امده میتوان نتیجه گرفت که مدل logistic regression برای مسایل کلاس بندی بسیار مناسب بوده و میتواند decision boundary مناسبی ارایه دهد. ولی در linear regression بیشتر مناسب برای مسایل regression بوده و برای fit کردن یک فانکشن پیوسته مناسب است و در مسایل کلاس بندی به خوبی عمل نمیکند. در این مسله به علت وجود داده پرت مدل سعی کرده است که این دیتا را fit کند و باعث شده دیتا های دیگری fit نشوند.

جواب 3) در این سوال به کمک کتابخانه sklearn مدل logistic regression را روی دیتای iris یادگیری خواهیم کرد (فایل multi_class_logistic_regression.ipynb) و کلاس بندی میکنیم این دیتا را.

الف) ابتدا دیتای iris را معرفی میکنیم :

این دیتا کلا 150 تا sample از زنبق ها میباشد که با توجه به 4 تا فیچر (طول و عرض کاسبرگ و طول و عرض گلبرگ بر اساس سانتی متر) که به 3 کلاس با نام های مختلف(setosa – versicolor- virginicia) target میشوند(هر کلاس 50 sample) . با دستور iris.DESCR میتوان ویژگی های بیشتری را مشاهده کرد.

```
**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988
```

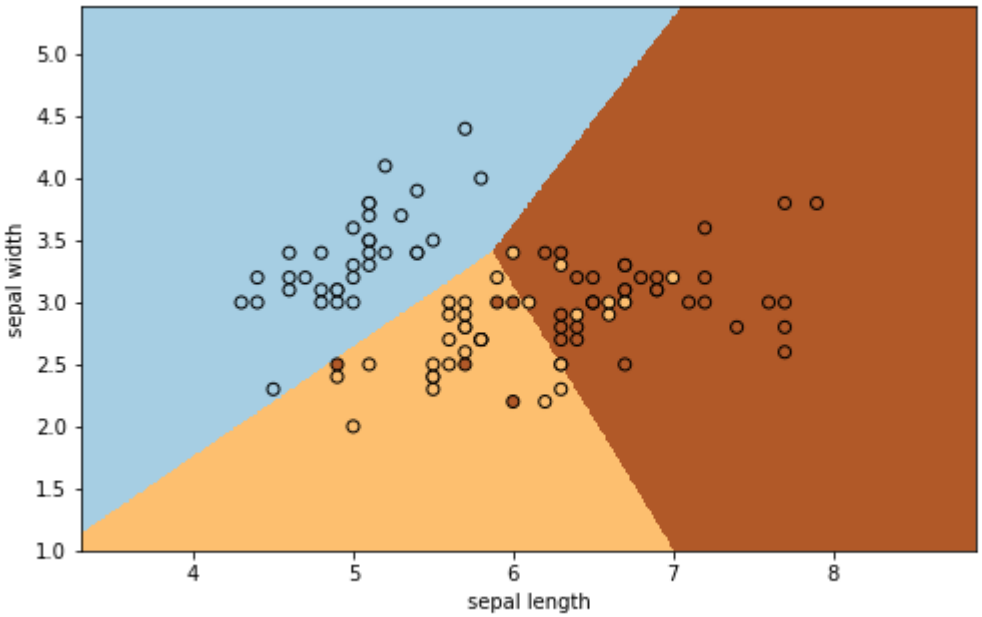
با توجه به این ورودی دیتا shape = (150, 4) میباشد که 150 sample و 4 فیچر
`['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']` میباشد و برای target داریم : shape = (150,) و کلاس های ما برابر است با :
['setosa' , 'versicolor' , 'virginicia']

با توجه به اینکه که گفته شده است فقط از 2 ویژگی اول استفاده کنیم داریم :

```
X = iris.data[:, :2]
```

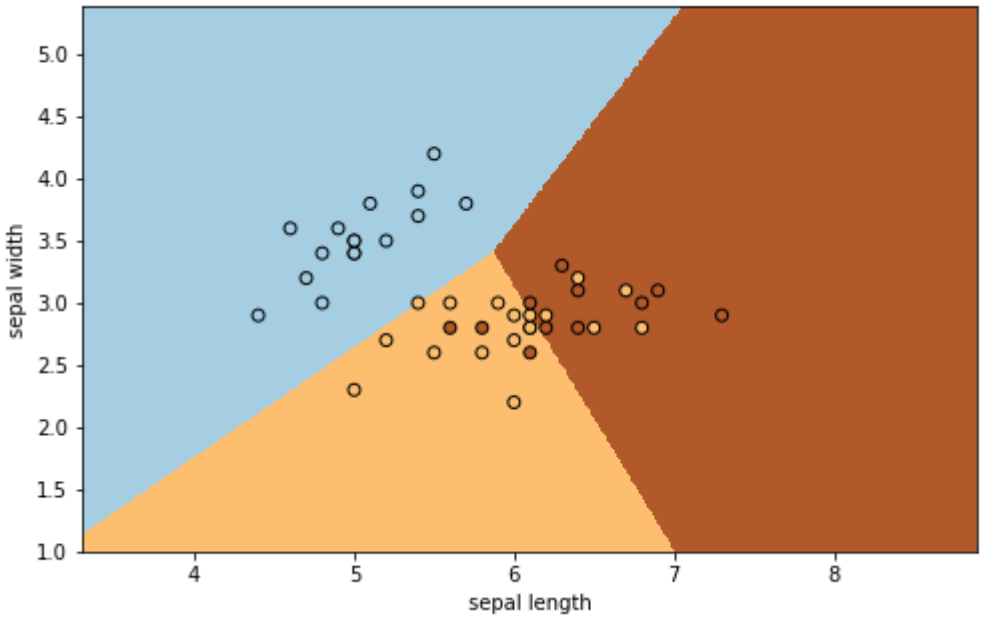
حال فقط از دو فیچر اول برای مدل استفاده خواهیم کرد. سپس 30 درصد دیتا را به عنوان دیتای تست جدا میکنیم تا شبکه بوجود امده را بسنجیم.

ب) حال در این مرحله پس از ساختن شبکه و fit کردن x_train,y_train دیتای train را در صفحه دو بعدی در دستگاه مختصات به شکل کلاس بندی شده رسم میکنیم.



که با توجه به شکل میبینیم که دیتای train در بعضی جا ها در کلاس درستی قرار نگرفته است.

ج) داده های آزمون را هم به شکل مرحله قبل نمایش میدهیم :



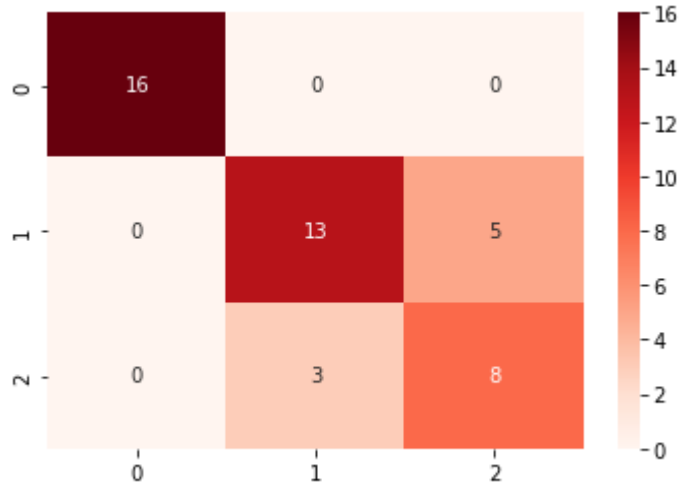
در این طرح هم میتوان مشاهده کرد که خطا در کلاس بندی وجود دارد.

د) برای بدست اوردن دقت فاز اموزش و تست ابتدا باید مقادیر پیش بینی کننده شبکه را به دست اورد و سپس انها را با مقادیر برچسب مقایسه کرد که با این مقایسه به نتایج زیر میرسیم :

```
accuracy on train data :  0.8285714285714286
accuracy on test data :  0.8222222222222222
```

با توجه به اعداد بالا میتوان نتیجه گرفت که شبکه خیلی خوب train نشده است و دارای خطای محسوسی میباشد (باید از فیچر های بیشتری استفاده کنیم تا بتوانیم خطا را کاهش دهیم) ولی میتوان گفت که در شبکه overfit رخ نداده است زیرا دقت فاز اموزش و تست نزدیک به همدیگر هستند و شبکه تا حد خوبی generalize شده است.

ه) در شکل زیر هم confusion matrix را داریم :



Confusion matrix یا error table هم گفته میشود که accuracy را به صورت ساده تری نشان میدهد که سطر ها معرف پیش بینی شبکه و ستون ها معرف واقعیت برچسب ها میباشد. تعداد سطر ها و ستون ها (برابر کلاس ها میباشد) که با توجه به این ماتریس میتوان فهمید که دیتای کلاس 0 با هیچکدام از دیتا های کلاس های دیگر تداخل ندارد ولی دیتای کلاس 1 و 2 اندکی باهم دیگر تداخل دارند و برای جدا سازی ان ها باید feature هایی به شبکه اضافه کرد که این دیتا های تداخل کرده را بتواند از هم متمایز کند.

منابع استفاده شده :

https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

https://medium.com/@kgpvijaybg/logistic-regression-on-iris-dataset-48b2ecdfb6d3