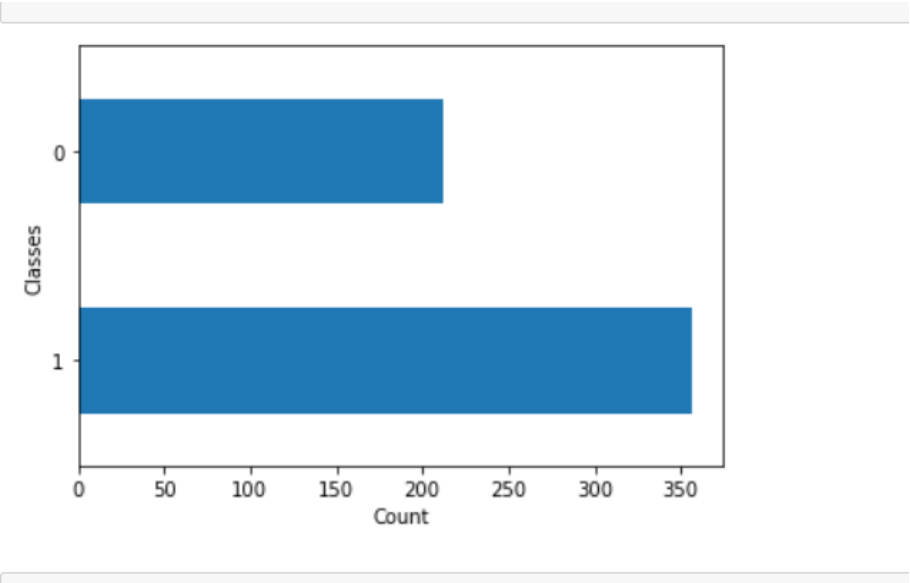
گزارش تمرین چهارم داده کاوی

فاطمه غلام زاده – ۹۵۳۱۰۶۰

- **سوال اول :**

بخش اول : پیاده سازی الگوریتم پرسپترون

ابتدا داده های breast cancer را لود می کنیم. این داده ها شامل ۵۶۹ سطر و ۳۰ فیچر است و یک مجموعه unbalance است چون داده های کلاس ۰ کمتر از کلاس ۱ هستند که در شکل زیر هم مشخص شده است :



اما سعی نداریم که آن ها را متعادل کنیم چون تفاوت زیادی ندارند و اینکه ما سعی داریم پرسپترون ساده را مدل کنیم .

پیاده سازی مدل به این صورت است که هر نمونه در وکتور وزن ها ضرب نقطه ای می شود و اگر بیش تر از threshold بود 1 در غیر اینصورت ۰ بر می گرداند.

تابعی داریم که بهترین وکتور وزن و بهترین threshold را learn میکند . در هر دور مقدار ستون متغیر هدف و مقادیر پیش بینی شده را مقایسه می کند و وزن ها را آپدیت می کند تا به هدف نزدیک شود.

```
    # find best weights and b
    def fit(self, X, Y, epochs, lr):
        self.w = np.ones(X.shape[1])
        self.b = 0
        accuracy = {}
        max_accuracy = 0
        wt_matrix = []
        # for all epochs
        for i in range(epochs):
            # for each (x,y)
            for count in range(X.shape[0]):
                if np.dot(self.w, X.iloc[count] ) >= self.b:
                    y_pred = 1
                else:
                    y_pred=0
                # compare real class value and prediction and update weig
                if Y.iloc[count] == 1 and y_pred == 0:
                    self.w = self.w + lr * X.iloc[count]
                    self.b = self.b - lr * 1

                elif Y.iloc[count] == 0 and y_pred == 1:
                    self.w = self.w - lr * X.iloc[count]
                    self.b = self.b + lr * 1

            wt_matrix.append(self.w)
            accuracy[i] = accuracy_score(self.predict(X), Y)
            if (accuracy[i] > max_accuracy):
                max_accuracy = accuracy[i]
                chkptw = self.w
                chkptb = self.b
```

در هر epoch دقت را نگه می داریم و در آخر بیشترین دقت را اعلام می کنیم. که هر چقدر تعداد epoch ها بیشتر باشد دقت هم بالاتر می رود مثلا برای ۱۰۰۰۰ تا epoch دقت حدود ۹۹ درصد می شود .

به دست آوردن دقت بیشتر :

۱- W را رندوم مقدار دهی کنیم (یعنی به جای تابع fit که وزن هارا آپدیت میکند وزن های اولیه را رندوم بدهیم) : این روش دقت را بالاتر نمی برد و آن را به حدود ۶۰ یا ۷۰ درصد می رساند .

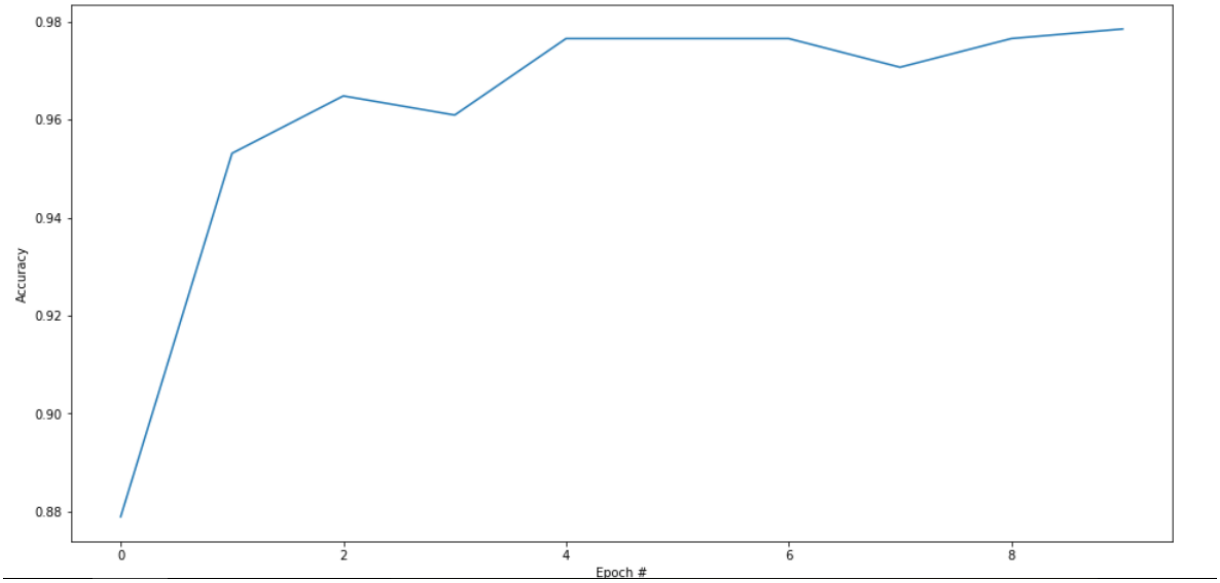
۲- اینکه نسبت سایز تست به train را تغییر بدهیم . برای نسبت های 0.1 و 0.2 و 0.3 و 0.4 امتحان می کنیم و مشاهده می شود که 0.2 بیشترین دقت را می دهد :

```
test size: 0.100000
max accuracy in train:
0.982421875
test accuracy:
0.9473684210526315

test size: 0.200000
max accuracy in train:
0.9824175824175824
test accuracy:
0.9736842105263158

test size: 0.300000
max accuracy in train:
0.9824120603015075
test accuracy:
0.9473684210526315

test size: 0.400000
max accuracy in train:
0.9853372434017595
test accuracy:
0.9605263157894737
```

نمودار دقت در epoch های مختلف :

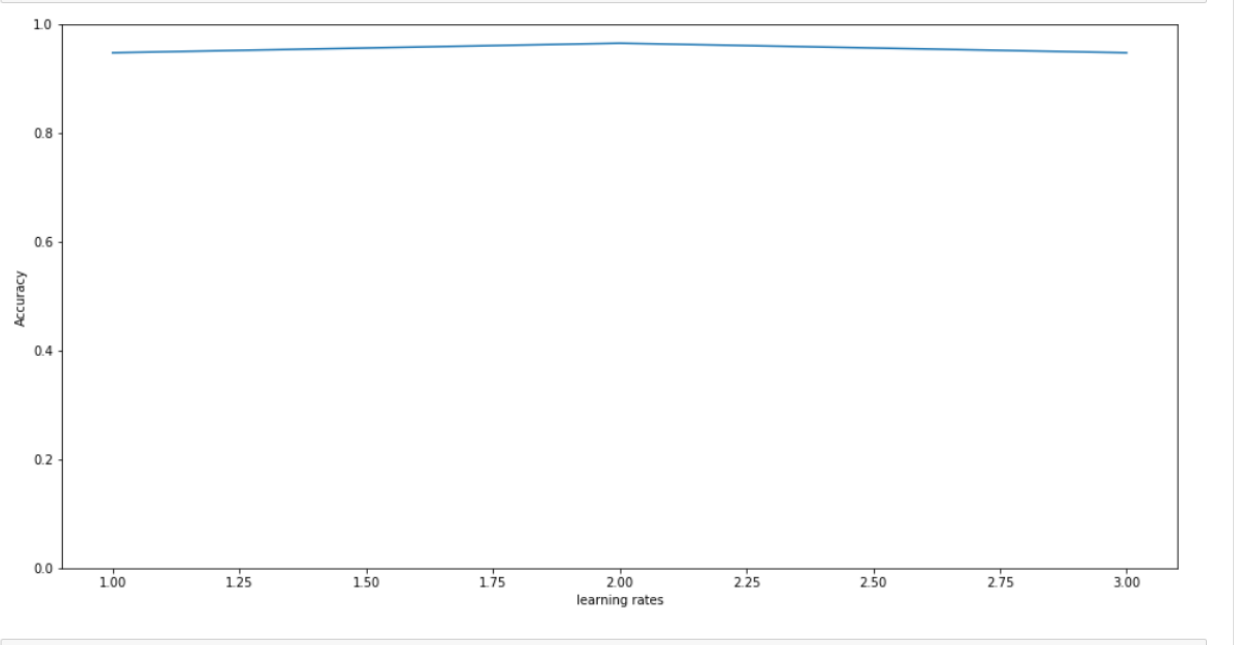
همان طور که مشاهده می شود با افزایش تعداد epoch ها دقت افزایش می یابد .

```
test accuracy :
0.9473684210526315
lr= 0.200000
max accuracy in train:
0.978515625
```



نمودار زیر هم نشان دهنده تاثیر تغییرات learning rate در دقت است :

```
max accuracy in train:
0.982421875
test accuracy :
0.9473684210526315
lr= 0.100000
max accuracy in train:
0.978515625
test accuracy :
0.9473684210526315
lr= 0.200000
max accuracy in train:
0.978515625
test accuracy :
0.9649122807017544
lr= 0.300000
max accuracy in train:
0.982421875
test accuracy :
0.9473684210526315
```

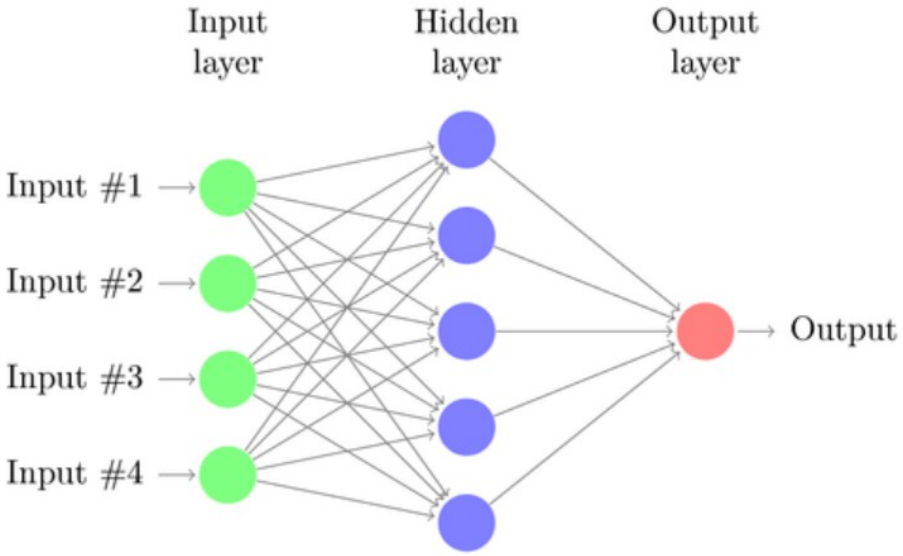


بخش دوم :

در این قسمت میخواهیم یک شبکه عصبی با یک لایه پنهان طراحی کنیم و آن را با پرسپترون مقایسه کنیم.

مقدار b یعنی bias را ۰ در نظر میگیریم .داده ها را لود کرده و به دو بخش test و train تقسیم می کنیم.

یک شبکه عصبی با یک لایه پنهان به صورت زیر است :



در این نوع از شبکه عصبی دو سری وزن داریم : یک سری بین لایه ورودی و لایه پنهان و یک سری بین لایه پنهان و لایه خروجی .

تولید اولیه ی هر ۲ سری این وزن ها به صورت رندوم صورت می گیرد .در اینجا هم مثل قسمت قبل یک تابع fit داریم که بهترین w را به داده ها fit می کند و آن را در self.w قرار می دهد تا مدل برای پیش بینی داده های جدید از این wاستفاده کند. هدف اصلی یافتن بهترین w و b است که کمترین loss را داشته باشیم.

در یک فانکشن دیگر هر نمونه را در w ضرب نقطه ای کرده و تابع activation را روی نتیجه اعمال می کنیم و مقدار prediction به دست می آید و سپس مقدار loss را حساب می کنیم :

The output $\hat{y}$ of a simple 2-layer Neural Network is:

$$\hat{y} = \sigma(W_2\sigma(W_1x + b_1) + b_2)$$

بعد از یافتن error یا همان loss باید وزن ها را آپدیت کنیم بنابراین نیاز است که Backpropagation انجام بدهیم که از مشتق تابع loss برحسبw استفاده می کنیم . (روش گرادیان نزولی)

برای مشتق گیری از قانون زنجیره ای استفاده می کنیم :

$$Loss(y, \hat{y}) = \sum_{i=1}^{n} (y - \hat{y})^2$$

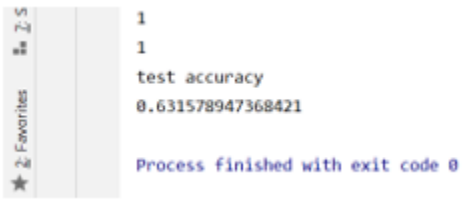
$$\frac{\partial\, Loss(y,\hat{y})}{\partial W} = \frac{\partial Loss(y,\hat{y})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial W} \quad \text{where } z = Wx + b$$

$$= 2(y - \hat{y}) * \text{derivative of sigmoid function} * x$$

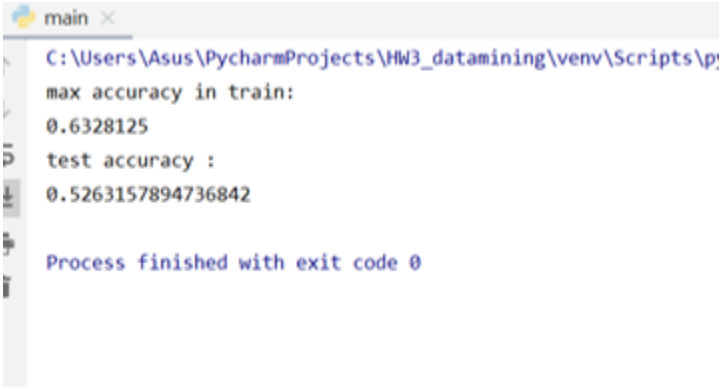
$$= 2(y - \hat{y}) * z(1\text{-}z) * x$$

یک تابع برای پیش بینی کردن هم داریم که اگر مقدار پیش بینی شده بیش تر از 0.5 باشد آن نمونه در کلاس ۱ و در غیر این صورت آن نمونه در کلاس ۰ قرار دارد.

دقت مدل برای داده های تست برابر 63 درصد است :



در بخش اول اگر b را برابر ۰ در نظر می گرفتیم دقت داده های تست برابر ۵۲ درصد می شد :



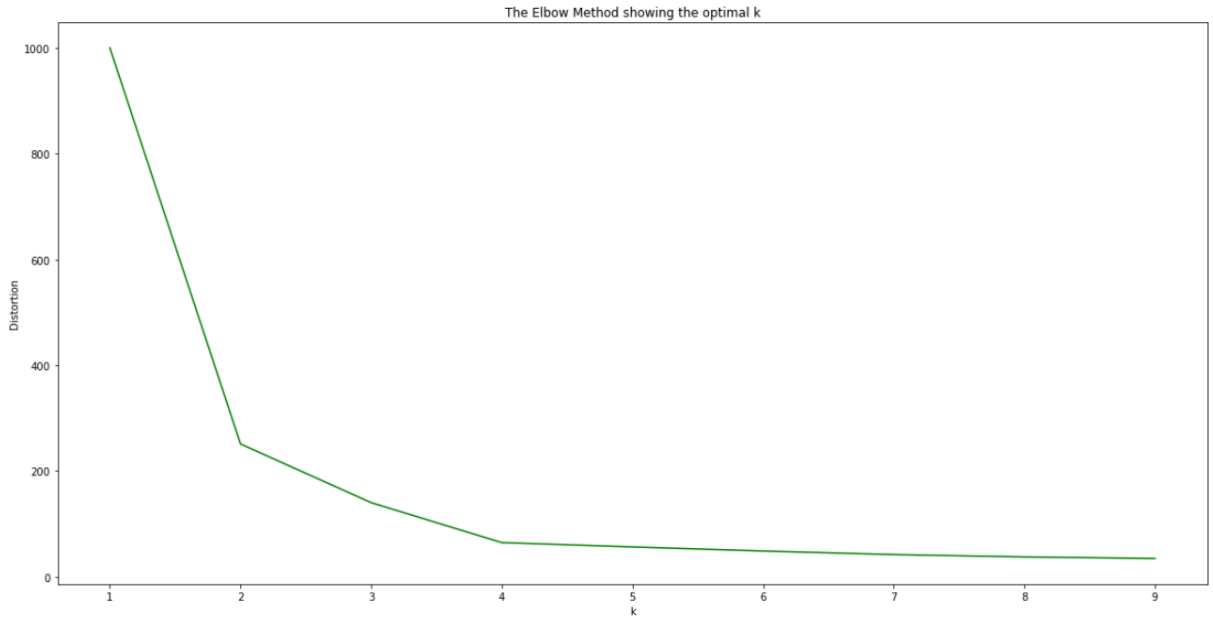
نتیجه :

همان طور که مشاهده می شود دقت شبکه عصبی با یک لایه پنهان بیشتر از پرسپترون است . در پرسپترون تابع تصمیم گیری یک تابع پله ای است و فقط برای داده هایی که به صورت خطی جداپذیرند مناسب است اما در شبکه عصبی از توابع activation استفاده می کنیم که قدرتمند تر هستند و دیتاهایی که به صورت خطی جداپذیر نیستند یا به وسیله ابرصفحه جدا می شوند را هم تشخیص می دهد.علاوه بر آن پرسپترون کارایی و سرعت کمتری هم دارد چون دارای افزونگی است.

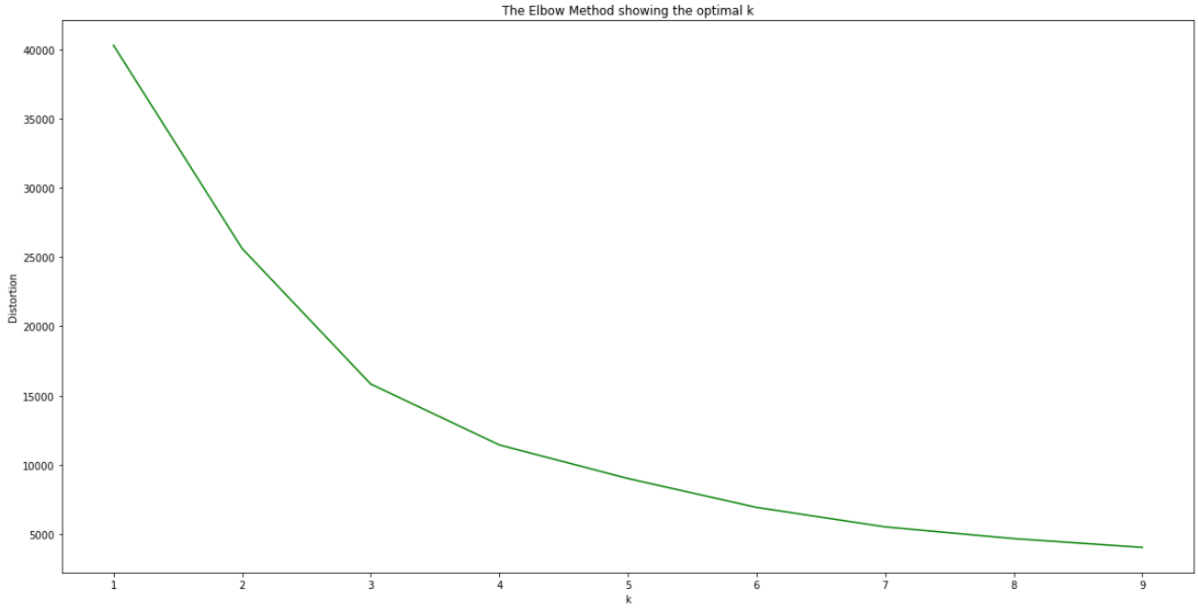
➢ سوال دوم

بخش اول :

برای اجرای الگوریتم k-means ابتدا باید تعداد خوشه ها (k) را تعیین نماییم . برای تعیین k از روش elbow استفاده می کنیم که نمودار آن برای دیتاست ۱ به شکل زیر است :



بنابراین برای دیتاست ۱ عدد k=4 مناسب است چون بعد از عدد ۴ دیگر شکستگی نداریم . نمودار حاصل برای دیتاست دوم به شکل زیر است :

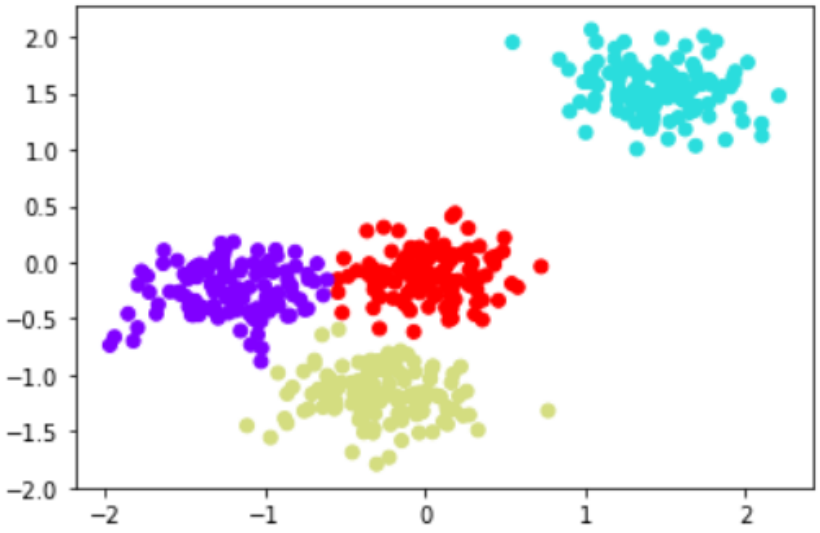


برای دیتاست دوم هیچ k مناسبی یافت نمی شود چون در نمودار آن شکستگی نمی بینیم بنابراین برای این دیتاست الگوریتم k-means روش مناسبی برای خوشه بندی نمی باشد.

سپس الگوریتم k-means را با k به دست آمده روی دیتاست اول اجرا می کنیم :

```python
# run Kmeans clustering algorithm
data1=pd.read_csv('Dataset1.csv')
k=4
kmeans = KMeans(n_clusters=k, random_state=0).fit(data1)
print(kmeans.labels_)
print(kmeans.cluster_centers_)
plt.scatter(data1['X'],data1['Y'], c=kmeans.labels_, cmap='rainbow')
plt.show()
```

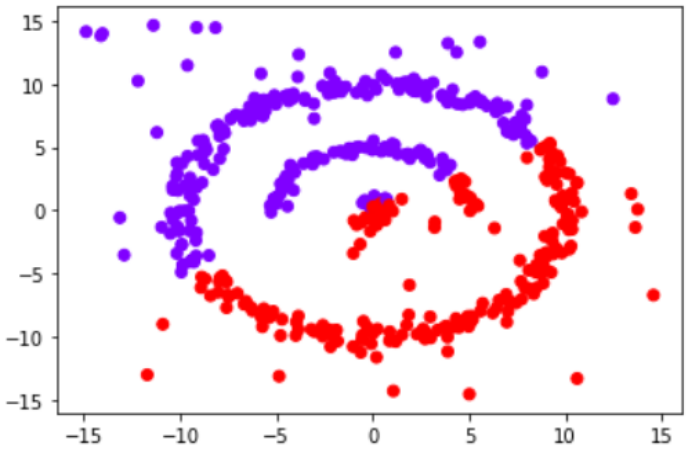
نمودار حاصل اجرای خوشه بندی به روش k-means بر روی دیتاست ۱ :



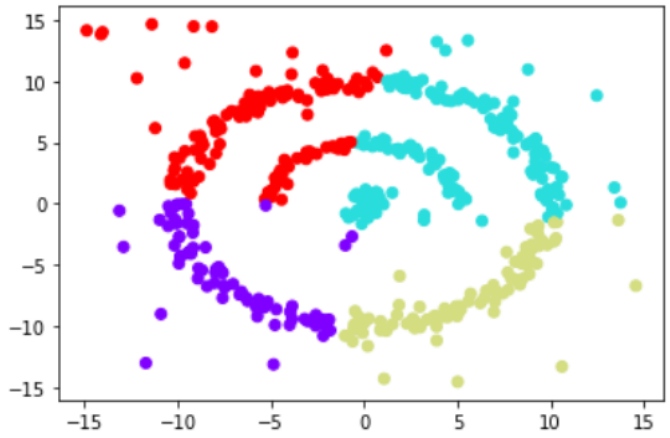
مراکز خوشه ها :

```
[[-1.20371678 -0.25345176]
 [ 1.45785578  1.53881252]
 [-0.27709885 -1.17219561]
 [ 0.0328524  -0.1120338 ]]
```

همان طور که گفتیم الگوریتم k-means الگوریتم مناسبی برای دیتاست دوم نمی باشد و با روش elbow هم نتوانستیم k مناسبی برای آن بیابیم . اگر برای مثال با k = 2 این الگوریتم را بر روی دیتاست دوم اجرا کنیم خروجی به شکل زیر است :



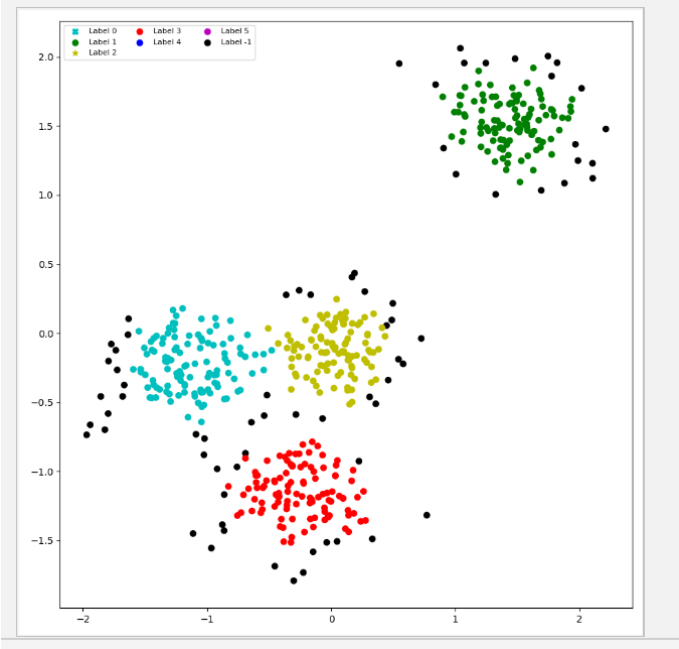
و یا مثلا برای k=4 خروجی به شکل زیر است :



علت اینکه k-means برای دیتاست دوم مناسب نیست این است که الگوریتم k-means برای دیتاست هایی که اشکال non-convex دارند مناسب نیست و دیتاست دوم هم به همین شکل است.

بنابراین برای خوشه بندی چنین دیتاستی به سراغ روش dbscan می رویم .

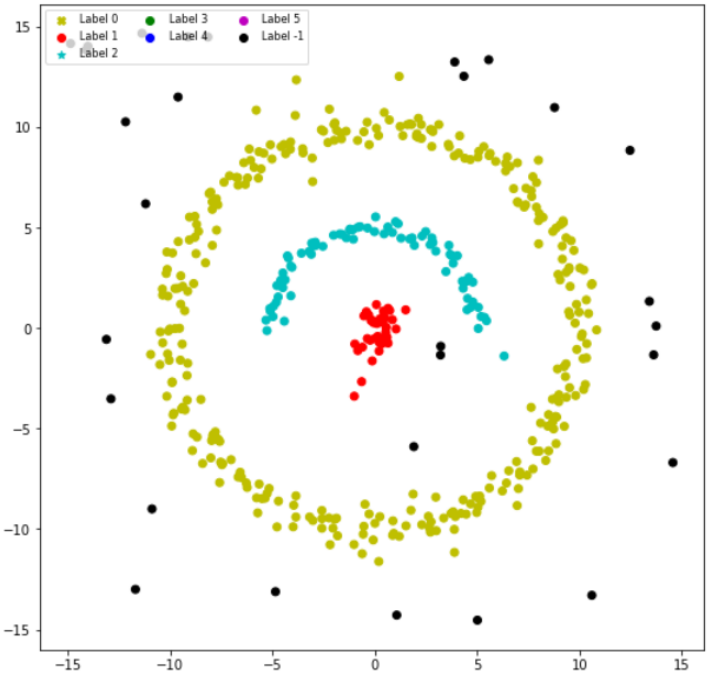
نمودار اجرای الگوریتم خوش بندی dbscan بر روی دیتاست ۱ :



داده های پرت با رنگ سیاه نشان داده شده اند . پارامترهای مناسب برای دیتاست اول :

```
# for data1
data1=pd.read_csv('Dataset1.csv')
db = DBSCAN(eps = 0.15, min_samples =8).fit(data1)
```

نمودار اجرای الگوریتم خوش بندی dbscan بر روی دیتاست ۲ :



پارامترهای مناسب برای دیتاست دوم :

```
# # for data2
data1=pd.read_csv('Dataset2.csv')
db = DBSCAN(eps = 2, min_samples =9).fit(data1)
```

همان طور که مشاهده می شود الگوریتم dbscan دیتاست دوم را به درستی خوشه بندی می کند .

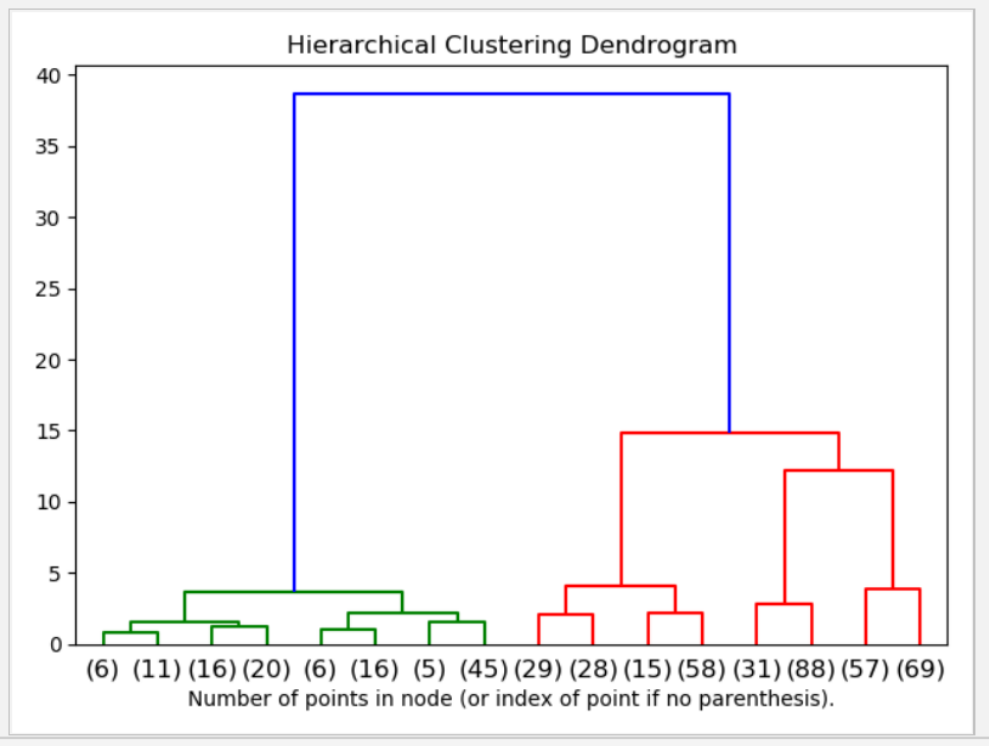
- **بخش دوم :**

از روش سلسله مراتبی agglomerative برای خوشه بندی استفاده می کنیم

treshold اولیه : ۰

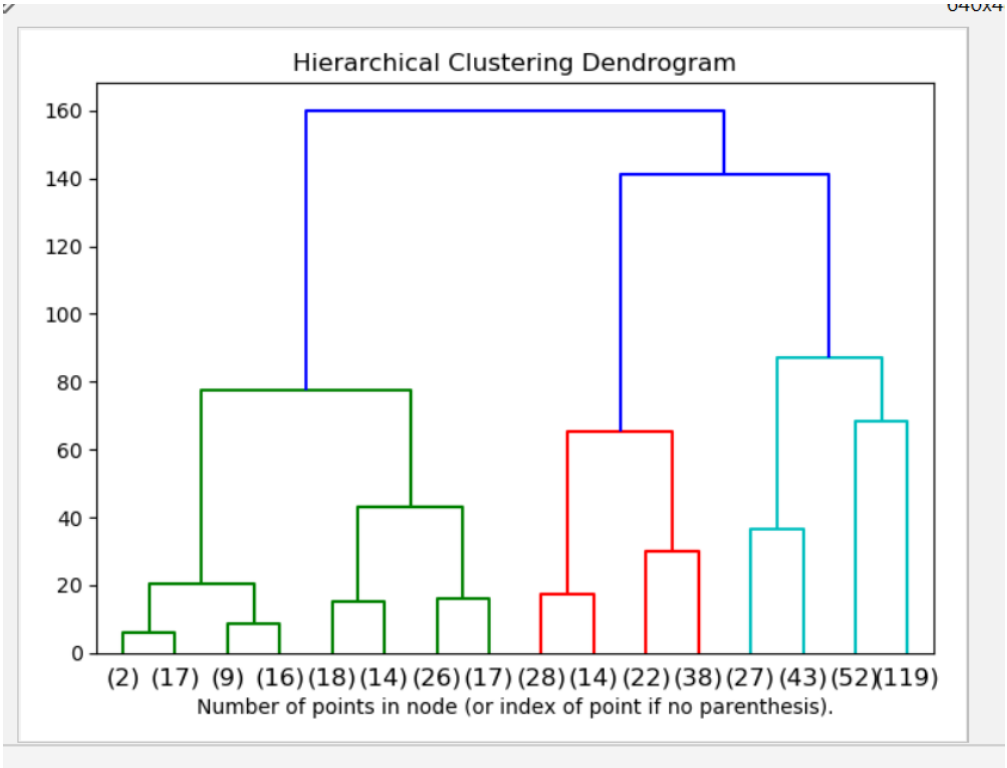
سپس یک treshold تعریف می کنیم و درخت را از آن جا میبریم تا خوشه ها بدست آیند.

نمودار dendrogram برای دیتاست اول :



۳ سطح بالای درخت در نظر گرفته شده است .

نمودار dendrogram برای دیتاست دوم :



۴ سطح بالای درخت در نظر گرفته شده است.

مزایای روش سلسله مراتبی :

در ابتدا لازم نیست که تعداد خوشه ها مشخص باشد.

معایب روش سلسله مراتبی :

۱- نیاز به یک threshold دارد که تعیین مقدار مناسب برای آن کار سختی است.

۲- پیچیدگی زمانی بالایی دارد : حداقل $O(n^2)$.

۳- قادر به undo کردن نیست.

➢ سوال سوم

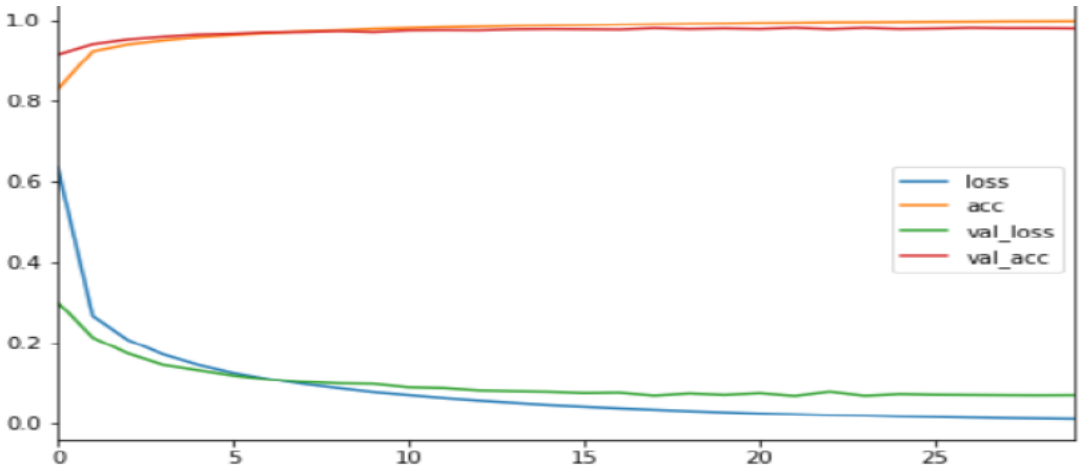
بخش اول:

ابتدا داده های MINIST را لود می کنیم . می خواهیم این داده ها را با یک شبکه ی fully connected دسته بندی کنیم .

از سه لایه استفاده می کنیم که تعداد نورون ها به صورت زیر است :

```
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(50, activation="relu"))
```

تعداد epoch ها 30 در نظر گرفته شده است . بیشترین دقت به دست آمده بر روی داده های validation برابر است با : 0.98



```
55000/55000 [==============================] - 5s 97us/sample - loss:
0.1105 - acc: 0.9677 - val loss: 0.1090 - val acc: 0.9692
Epoch 8/30
55000/55000 [==============================] - 5s 97us/sample - loss:
0.0972 - acc: 0.9722 - val loss: 0.1024 - val acc: 0.9700
Epoch 9/30
55000/55000 [==============================] - 5s 96us/sample - loss:
0.0868 - acc: 0.9744 - val loss: 0.0985 - val acc: 0.9726
Epoch 10/30
55000/55000 [==============================] - 5s 98us/sample - loss:
0.0771 - acc: 0.9779 - val_loss: 0.0973 - val_acc: 0.9700
Epoch 11/30
55000/55000 [==============================] - 5s 100us/sample -
loss: 0.0698 - acc: 0.9800 - val loss: 0.0886 - val acc: 0.9742
Epoch 12/30
55000/55000 [==============================] - 5s 99us/sample - loss:
0.0627 - acc: 0.9825 - val loss: 0.0867 - val acc: 0.9748
Epoch 13/30
55000/55000 [==============================] - 5s 99us/sample - loss:
0.0565 - acc: 0.9841 - val loss: 0.0809 - val acc: 0.9744
Epoch 14/30
55000/55000 [==============================] - 5s 96us/sample - loss:
0.0511 - acc: 0.9862 - val_loss: 0.0791 - val_acc: 0.9768
Epoch 15/30
55000/55000 [==============================] - 5s 96us/sample - loss:
0.0458 - acc: 0.9872 - val loss: 0.0777 - val acc: 0.9776
Epoch 16/30
55000/55000 [==============================] - 5s 97us/sample - loss:
```

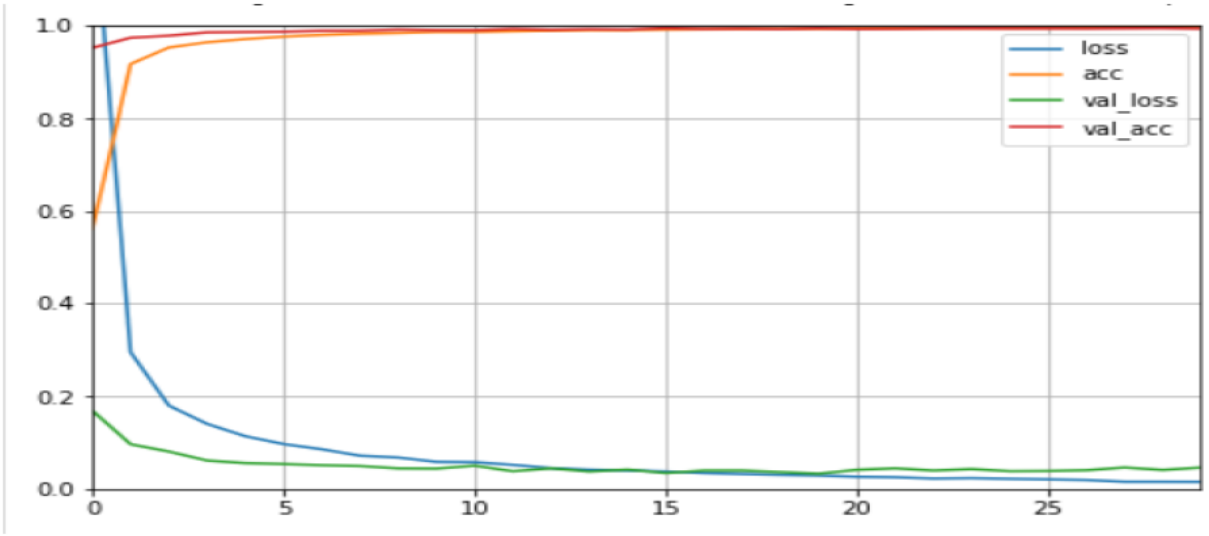
بخش دوم :

در این داده های قسمت قبل را با استفاده از شبکه کانوولوشنی دسته بندی می کنیم . معماری استفاده شده :

Cond2d -> maxpool - > Cond2d -> Cond2d -> maxpool ->dense -> dense-> dense

مدل را به وسیله ی alexnet می سازیم .

Epoch را برابر 30 در نظر میگیریم و بیشترین دقت برای داده های validation برابر 0.9940 به دست می آید .




```
Epoch 20/30
55000/55000 [==============================] - 773s 14ms/sample -
loss: 0.0288 - acc: 0.9927 - val_loss: 0.0328 - val_acc: 0.9934
Epoch 21/30
55000/55000 [==============================] - 774s 14ms/sample -
loss: 0.0260 - acc: 0.9932 - val loss: 0.0411 - val acc: 0.9920
Epoch 22/30
55000/55000 [==============================] - 778s 14ms/sample -
loss: 0.0250 - acc: 0.9936 - val loss: 0.0445 - val acc: 0.9924
Epoch 23/30
55000/55000 [==============================] - 784s 14ms/sample -
loss: 0.0222 - acc: 0.9943 - val loss: 0.0396 - val acc: 0.9932
Epoch 24/30
55000/55000 [==============================] - 780s 14ms/sample -
loss: 0.0231 - acc: 0.9941 - val loss: 0.0428 - val acc: 0.9936
Epoch 25/30
55000/55000 [==============================] - 772s 14ms/sample -
loss: 0.0216 - acc: 0.9945 - val_loss: 0.0381 - val_acc: 0.9932
Epoch 26/30
55000/55000 [==============================] - 771s 14ms/sample -
loss: 0.0204 - acc: 0.9948 - val loss: 0.0386 - val acc: 0.9932
Epoch 27/30
55000/55000 [==============================] - 769s 14ms/sample -
loss: 0.0190 - acc: 0.9952 - val loss: 0.0402 - val acc: 0.9930
Epoch 28/30
55000/55000 [==============================] - 768s 14ms/sample -
loss: 0.0153 - acc: 0.9959 - val loss: 0.0462 - val acc: 0.9934
Epoch 29/30
```

نتیجه :

دقت روش دوم بالاتر است (حدود ۱.۳ درصد). علت :

۱- در شبکه fully connected تعداد پارامترهای کل می تواند بسیار زیاد شود که برابر حاصلضرب تعداد لایه ها در تعداد پرسپترون هاست و این امر در ابعاد بالا افزونگی زیادی را ایجاد می کند .

۲- روش fully connected به اطلاعات مکانی توجهی ندارد و ورودی های آن بردارهای مسطح است نه بردارهای دو بعدی بنابراین دقت آن کم است. یک پرسپترون چند لایه با تعداد لایه های کم می تواند برای این داده ها دقت بالایی بیاید اما CNN از آنجایی که دوبعدی است و اطلاعات را کامل نگه می دارد (و افزونگی هم ندارد) می تواند با تعداد لایه های بیشتر هم به دقت بالا برسد .