به نام خدا

**گزارش مینی پروژه 2**

امیر ارشام بهشتی طهرانی مبانی سیستم های هوشمند شماره دانشجویی:9820293

**سوال‏1)**

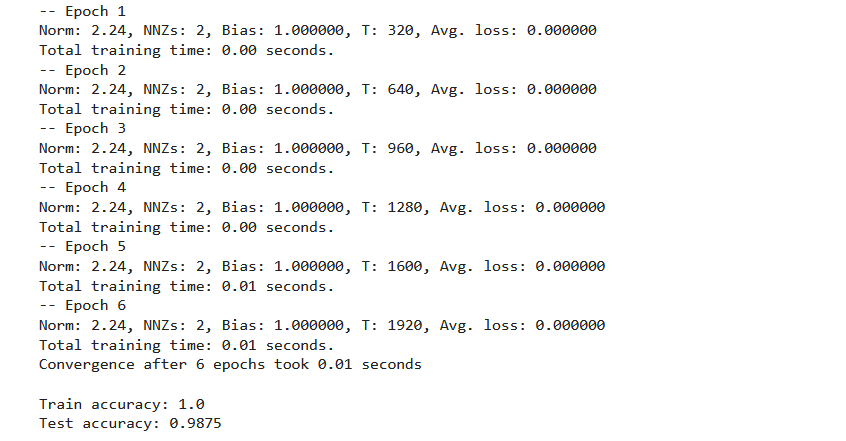
1\_ : ابتدا با دستور dgown فایل دیتاست را در google colab بارگذاری کرده و سپس کتابخانه های مربوطه را import می کنیم و در ادامه با دستور pd.read\_csv دیتاست را در یک دیتافریم پاندا قرار می دهیم.

با دستور .iloc[:,:-1] تمام داده ها در سطر ها و ستون ها به جز ستون آخر که مربوط به کلاس است را در x قرار می دهیم و به طور مشابه فقط داده های ستون آخر را در y قرار می دهیم.

با استفاده از دستور np.where( y == -1, 0, 1) تمام y هایی که -1 هستند را به صفر تبدیل می کنیم تا از لحاظ بصری کار با دیتاست راحت‏تر شود.

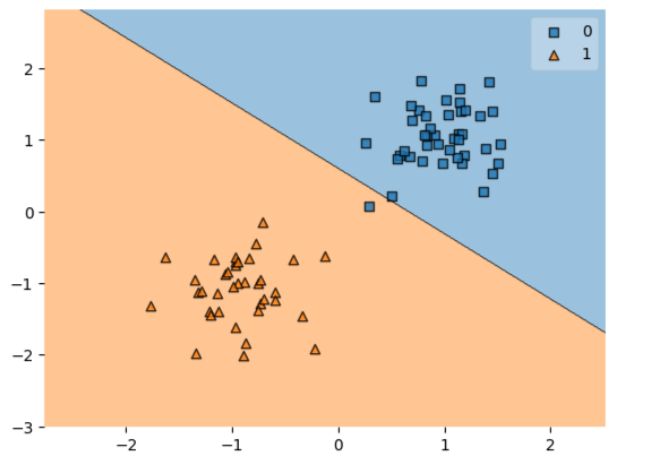
به کمک train\_test\_split مشابه تمرین قبل داده ها را با نسبت 80 درصد به train و test (test\_size = 0.2) تقسیم می‏کنیم و random\_state =93 قرار می دهیم تا در هر حالت داده ها را به طور مشخص جدا کند.

در ادامه مدل کلاس بندی مان را با Perceptron( ) داخل کتابخانه sklearn.Linear\_model تعریف می‏کنیم و با verbose = True می توانیم در هر دوره مشخصات مدل (تابع اتلاف و ...) را مشاهده می‏کنیم. در ادامه با متد .fit( ) مدل را بر داده های train آموزش می دهیم و با متد .predict( ) ، y خروجی متناظر با ورودی داده های test را مدل به عنوان خروجی پیش بینی می کند. با متد .score( ) دقت مدل را روی داده های train و داده های test می سنجیم.



با توجه به نتایج متوجه می شویم که بعد از دوره ششم (epoch=6) تابع اتلاف همگرا شده و در نتیجه روند آموزش متوقف شده است. دقت داده های آموزش و ارزیابی هم مشاهده می شود.

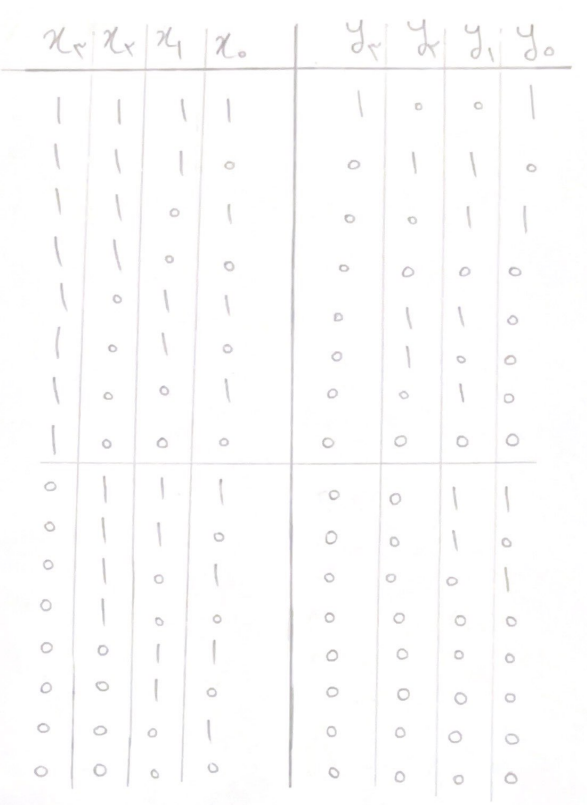
به کمک دستور .scatter ، داده های دو کلاس را با رنگ مختلف نشان داده و سپس به کمک mlxtend ، دو ناحیه موجود در دو کلاس را از هم جدا می‏کنیم (شکل برای داده های test نشان داده شده است.)



**سوال 2)**

در ضرب کننده دودویی باینری ورودی ما شامل x3x2 به عنوان دو بیت عدد اول و x1x0 به عنوان دو بیت عدد دوم است. چون ورودی ها دودویی هستند حداکثر تا عدد 3 مقدار دارند و در نتیجه در خروجی طبق همه حالات ضرب اعداد از صفر تا 3، اعداد 9,6,4,3,2,1 را پوشش می دهند و در نتیجه با توجه به عدد 9 در خروجی نیاز به 4 بیت y3y2y1y0 خواهیم داشت.

ابتدا جدول صحت ورودی و خروجی را رسم می کنیم.



سپس با توجه به حالات ورودی برای هر بیت خروجی، می توان هر بیت خروجی را با در نظر گرفتن گیت های مناسب منطقی به دست آورد. برای این کار ابتدا به کمک جدول کارنو گیت‏های مناسب AND و OR را برای کمترین حالت دست می آوریم. سپس در صورت امکان با ساده تر کردن جواب به دست آمده، تعداد گیت ها را کاهش می دهیم.

در نهایت چون هر گیت را می توان با یک نورون McCulloch-Pits مدل کرد، شبکه حاصل را برای ضرب کننده دودویی باینری به دست می‏آوریم.

شبکه خروجی y3 :

با توجه به جدول درستی مشاهده می شود این خروجی تنها هنگامی که تمام ورودی ها برابر 1 هستند فعال می شود در نتیجه می توان آن را با یک AND با چهار ورودی مدل کرد. در این صورت می توان نورون را طوری تعریف کرد که 4 ورودی با وزن 1 دارند و آستانه برابر 3- است.

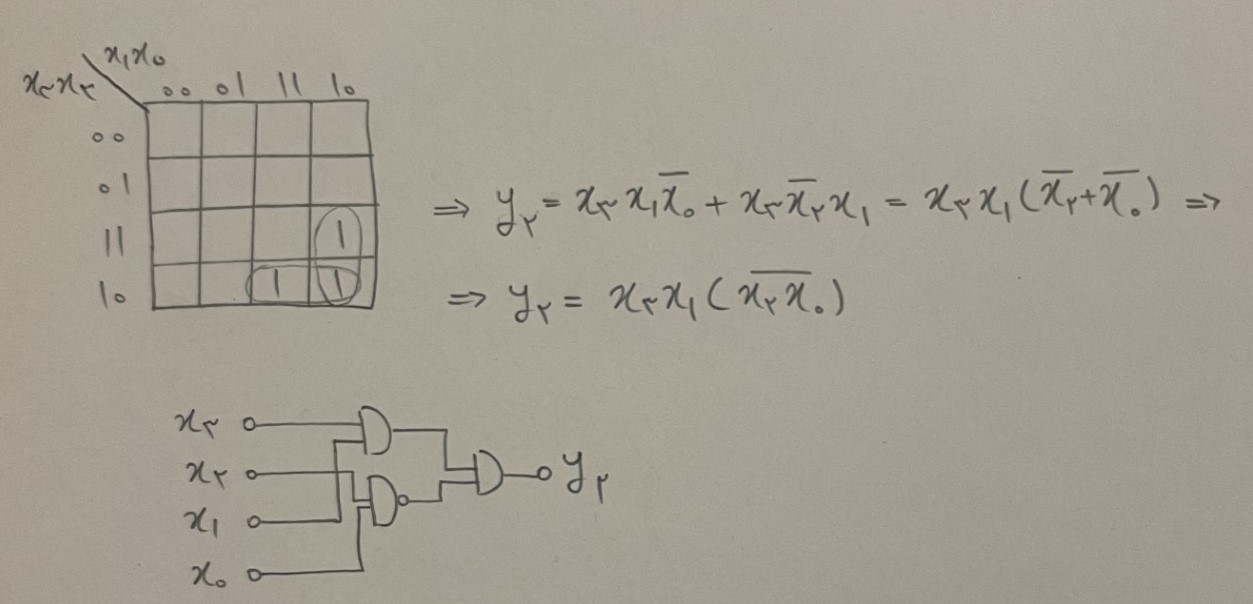
در این صورت تنها در حالتی که همه ورودی ها برابر 1 هستند، خروجی نورون 1 می‏شود.

شبکه خروجی y0 :

این خروجی تنها زمانی فعال می شود که x0 و x­­2 هر دو 1 باشند و x3 و x2 هر مقداری می توانند داشته باشند. در نتیجه فقط گیت And میان این دو ورودی نیاز است. و در نورون هم ضرایب x3 و x2 صفر خواهند بود و سایر ضرایب مانند نورون برای گیت And خواهد بود.

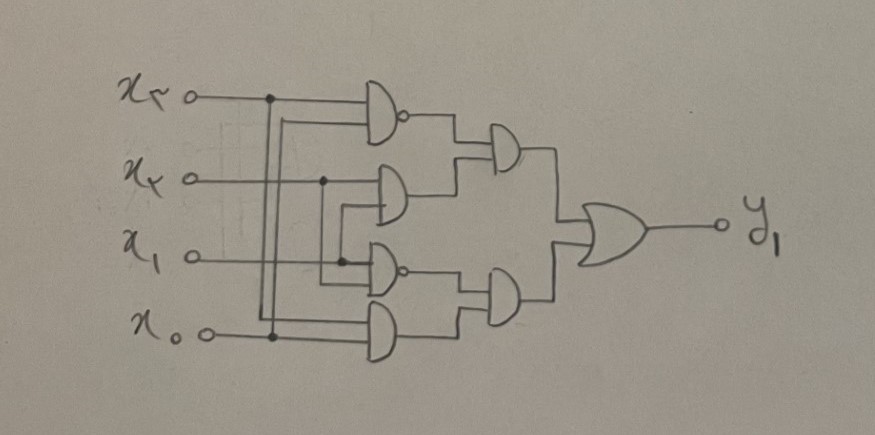
شبکه خروجی y2 :

برای به دست آوردن گیت های مناسب برای این خروجی باید با استفاده از جدول کارنو و حاصل SoP آن گیت های میان ورودی ها را به دست بیاوریم. در شکل زیر نحوه به دست آوردن شبکه مشخص شده است.



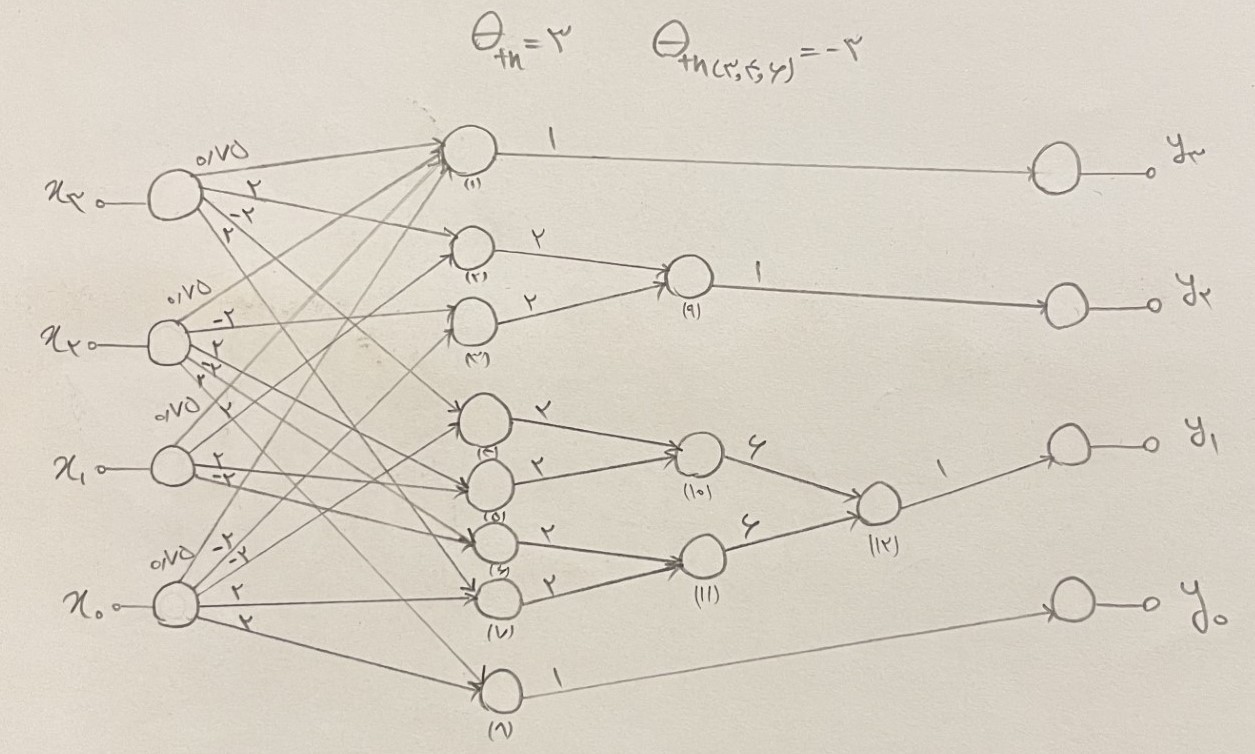
شبکه خروجی y1 :

برای این خروجی هم مطابق با حالت قبل می توان خروجی را محاسبه کرد که در نهایت شبکه به صورت زیر خواهد شد.



در نهایت با جایگذاری نورون های McCulloch-Pitts به جای گیت ها با مشخص کردن آستانه مربوط به هر گیت می توان شبکه این مدار را رسم کرد.

طبق محاسبات مشخص است که نورون ها با آستانه مشخص برای هر گیت تعریف می شوند تا مانند آن گیت عمل کنند. برای گیت And با 4 ورودی، آستانه را برابر 4 در نظر می گیریم تا تنها در زمانی که ورودی ها برابر 1 هستند خروجی آن 1 شود. برای گیت And با 2 ورودی آستانه برابر 5/1 ، برای گیت Or برابر 5/0 و برای گیت Nand برابر 5/1- است. چون همه نورون ها آستانه برابر باید داشته باشند می توانیم آستانه را برابر 3 برای همه نورون ها (به جز نورون مربوط به Nand که 3- است) در نظر بگیریم و در این صورت وزن ها به همین نسبت تغییر میکنند. در نتیجه وزن گیت And با 4 ورودی برابر 75/0، گیت Or برابر 6، گیت And با 2 ورودی برابر 2 و گیت Nand برابر 2- می‏شود.



2\_ : ابتدا نورون McCulloch-Pitts را با یک کلاس و ورودی های وزن ها و آستانه هر نورون تعریف می کنیم. سپس با تعریف متد .model که ورودی آن x و هر نمونه کلاس است، عملیت صورت گرفته در هر نورون را پیاده سازی می کنیم.(مقایسه ضرب وزن ها در ورودی با آستانه)

سپس با تعریف تابع BM به عنوان ضرب کننده باینری، نورون های McCulloch-Pitts را با وزن ها و آستانه تعریف شده در شبکه‏ای که در بالا طراحی کردیم، ایجاد می کنیم (12 نورون). سپس با استفاده از متد .model فعالیت نورون ها را پیاده سازی می کنیم و z1 تا z12 خروجی هر یک از نورون ها هستند. در .model ورودی ها آرایه‏ای از input که در ادامه تعریف کرده می کنیم، می باشند.

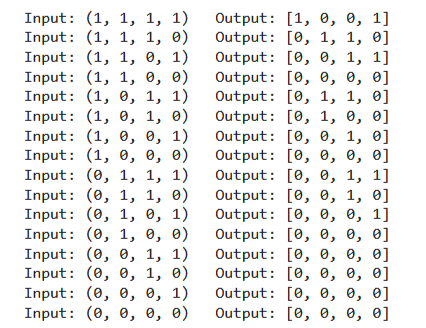
با توجه به شبکه نورون ها ورودی نورون 9 ، خروجی نورون 2 و 3 بوده (z2 , z3) در نتیجه به جای input، z2 و z3 را در ورودی نورون 9 قرار می دهیم. به همین صورت ورودی های نورون 10، 11 و 12 را نیز با توجه به خروجی نورون های مربوط به آن ها مشخص می کنیم.

در نهایت خروجی این تابع یک لیست از خروجی نورون های 1، 9، 12 و 8 است زیرا در شبکه این نورون‏ها به ترتیب به خروجی های y3، y2، y1 و y0 متصل هستند.

در ادامه با تعریف ورودی باینری 0,1 در input\_b و استفاده از itertools.product تمام حالت ها برای 4 ورودی (16 حالت) را ایجاد می کنیم و داخل آرایه X قرار می‏دهیم.

با نوشتن یک حلقه for برای هر سطر X که هر حالت ورودی ماست، آن را (i) به عنوان ورودی ضرب کننده باینری (BM) اعمال می کنیم و خروجی هر حالت (result) را مشاهده می‏کنیم.

خروجی هر حالت با توجه به ورودی در شکل پایین آمده که مطابق جدول درستی ضرب کننده باینری دو بیتی است.



**سوال 3)**

1\_ : در این دو تابع از ماژول PIL، با استفاده از Image و ImageDraw می توانیم انجام عملیات خواندن و ذخیره کردن فایل های تصویر را به راحتی انجام دهیم.

تابع اول مربوط به تبدیل تصویر به اعداد باینری است که باعث می‏شود نویز را در ادامه به آن راحت تر اضافه کنیم. این تابع به عنوان آرگومان ورودی، مسیر تصویر ذخیره شده را دریافت می‏کند که با متد .open(path) ابتدا تصویر را دریافت می‏کنیم.

با استفاده از متد .Draw() می توانیم بر روی تصویر تغییراتی اعمال کنیم و نمایش دهیم. در ادامه عرض و طول تصویر را در دو لیست مجزا تعریف می کنیم تا به هر پیکسل دسترسی داشته باشیم. سپس با متد .load() پیکسل ها را در یک لیست دیگر (pix) قرار می دهیم.

برای بیان باینری تصویر باید مقادیر هر پیکسل را با آستانه‏ای بسنجیم تا با توجه به آن مقدار سیاه (1) یا سفید (-1) را برای آن در نظر بگیریم. در نتیجه آستانه factor را تعریف کرده و برای مقدار نهایی تصویر باینری binary\_representation را به صورت یک لیست خالی تعریف می‏کنیم تا مقدار جدید هر پیکسل را داخل آن قرار دهیم.

سپس به کمک دو حلقه تو در تو مقدار RGB داخل هر پیکسل را فراخوانی می کنیم و شدت کل آن ها را به دست می آوریم. می دانیم هر چه مقادیر هر سه رنگ بیشتر باشد تصویر بیشتر سفید و روشن خواهد بود. در نتیجه برای نسبت دادن رنگ سفید یا سیاه به هر پیکسل، شدت کل را با میانگین آستانه (factor) و 255 که در 3 ضرب شده (چون برای هر رنگ باید در نظر گرفته شود) می‏سنجیم و اگر بیشتر از آن بود پیکسل را سفید در نظر گرفته و مقدار -1 را در لیست binary\_representation قرار می دهیم و در غیر این صورت مقدار 1 را در آن قرار می‏دهیم تا سیاه در نظر گرفته شود. با متد .point() هم پیکسل‏های تصویر را سیاه و سفید تبدیل می‏کنیم.

در تابع دوم ابتدا قسمتی را برای ذخیره تصویر نویزی شده با استفاده از دستور enumerate که به هر تصویر یک index نسبت می دهد و تصویر نویزی معادل با تصویر ساده را جایگزین و ذخیره می‏کند، در نظر می گیریم.

این تابع در آرگومان خود مسیر تصویر ورودی و خروجی را دریافت می کند. مانند قبل تصویر را فراخوانی کرده و مقادیر پیکسل های آن را همراه طول و عرض جدا می‏کنیم.

برای تولید نویز یک بازه به کمک noise\_factorاز منفی تا مثبت آن را مشخص می کنیم تا با تابع random\_randint یک عدد رندوم در آن را به عنوان نویز تولید کند. این نویز تولید شده را به هر رنگ در پیکسل اضافه کرده و سپس چون مقادیر هر پیکسل رنگی میان 0-255 در هر رنگ می باشد، مقادیر کمتر را به صفر و مقادیر بیشتر از این بازه را به 255 تبدیل می‏کنیم.

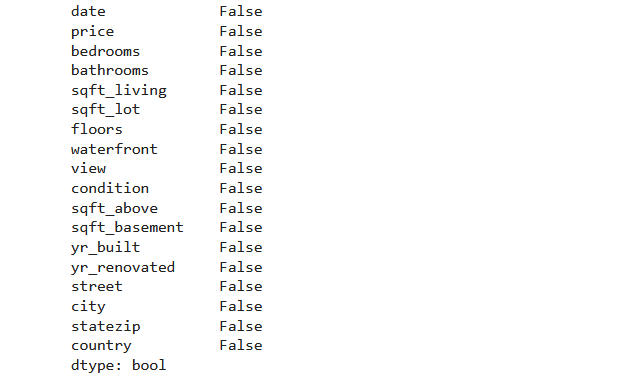
به کمک متد .point() در draw پیکسل های تصویر را به مقدار های جدید تبدیل می کنیم و سپس تصویر حاصل را به عنوان تصویر نویزی تصویر متناظر ذخیره می کنیم.

**سوال 4)**

­1\_ : ابتدا پس از اینکه فایل را بارگذاری کردیم و کتابخانه های مورد نظر را import کردیم، فایل را به شکل یک دیتافریم مشابه تمرینات قبلی در می آوریم. به کمک دستور .info( ) اطلاعاتی راجع به دیتاست که مربوط به قیمت خانه با توجه به امکانات آن می باشد به دست می آوریم. برای مثال تعداد ویژگی ها و نوع داده های داخل هر کدام را مشاهده می کنیم.

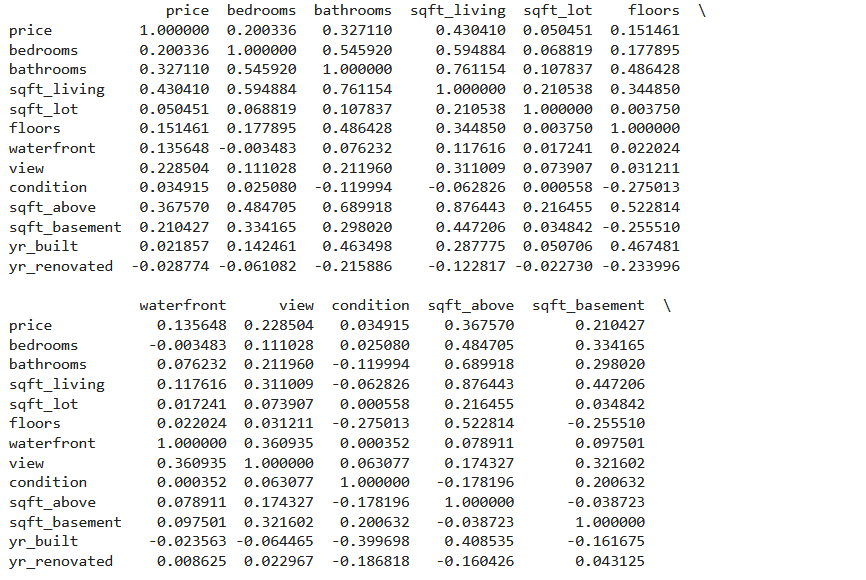
برای بررسی وجود ‘Nan’ داخل هر ستون از دو دستور می توان استفاده کرد.

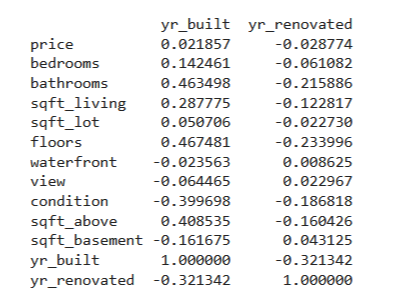
با دستور .isnull( ).any هر ستون بررسی می شود و در هر ستون در صورت وجود داده ‘Nan’ در خروجی ستون، False و در غیر این صورت، True نمایش داده می شود. راه دیگر که تعداد ‘Nan’ های داخل هر ستون را نمایش می دهد استفاده از .isnull( ).sum( ) می باشد که در این مسئله مشاهده می‏شود که داده ای به صورت ‘Nan’ وجود ندارد.



2\_ : برای رسم ماتریس همبستگی که میزان وابستگی ستون ها به یکدیگر را مشخص می کند از دستور df.corr() استفاده می کنیم و سپس با مشاهده میزان اعداد مربوط به سطر قیمت، مشخص می شود که با ستون ‘sqft\_living’ که مساحت خانه است همبستگی بیشتری دارد.

البته این ماتریس فقط همبستگی ستون هایی را نشان می دهد که داده داخل آن به صورت عدد است.





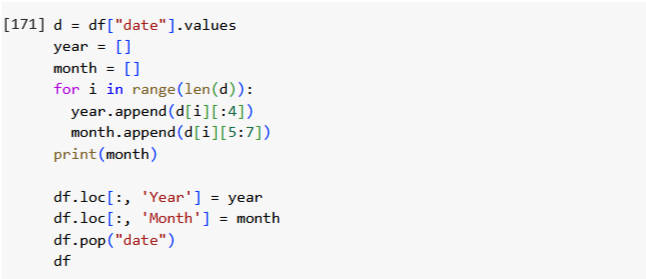
3\_ : برای رسم نمودار قیمت بر حسب sqft\_living، ابتدا به ترتیب price را داخل y و sqft\_living را در x قرار داده و سپس پس از نامگذاری محور های نمودار، حدود آن را با plt.xlim و plt.ylim مشخص می کنیم تا صعود قیمت با توجه به صعود sqft\_living بهتر مشخص شود. در نهایت به کمک plt.scatter نمودار را رسم می کنیم.



4\_ : برای تبدیل ستون date به ماه و سال ابتدا مقادیر داده های داخی این ستون را به کمک متد .values ، داخل لیست d ذخیره می کنیم. دو لیست خالی year و month را تعریف می‏کنیم و سپس به کمک یک حلقه در بازه تعداد داده های داخل ستون date یا همان d ، با توجه به فرمت نوشته شدن تاریخ که 4 المان اول مربوط به سال و المان ششم و هفتم به ماه مربوط است، به ترتیب آن ها را در هر دور حلقه داخل لیست های خالی قرار می دهیم.

در نهایت به کمک دستور .loc[:, ‘Year’] و .loc[:, ‘Month’] دو ستون جدید با این نام ها در دیتافریم ایجاد کرده و مقادیر داخل لیست را در این ستون ها قرار می دهیم. می توانستیم از دستور .insert( ) هم استفاده کنیم ولی با هر بار run کردن این قسمت این ستون ها دوباره به دیتافریم اضافه می شدند.

در ادامه هم به کمک دستور .pop(‘date’) ، ستون تاریخ را حذف می کنیم.

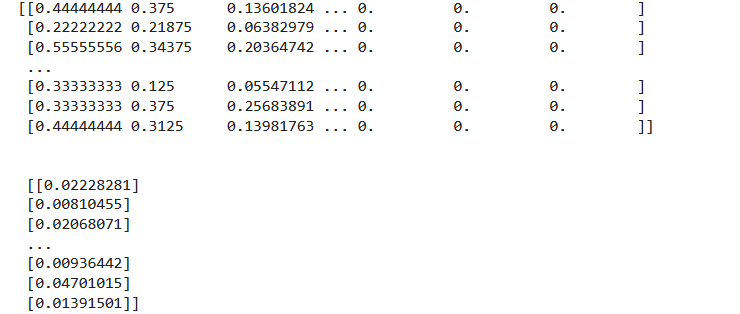


5\_ : برای آموزش مدل ابتدا باید با استفاده از روش one-hot encoding داده هایی را که به صورت کلمه هستند، به صورت اعداد باینری متناظر نشان می دهیم. برای این کار ابتدا در لیست dummy نام ستون هایی که داده شان به صورت عدد نیست را مشخص کرده و سپس به کمک دستور زیر آن ها را به صورت اعداد باینری در می آوریم.



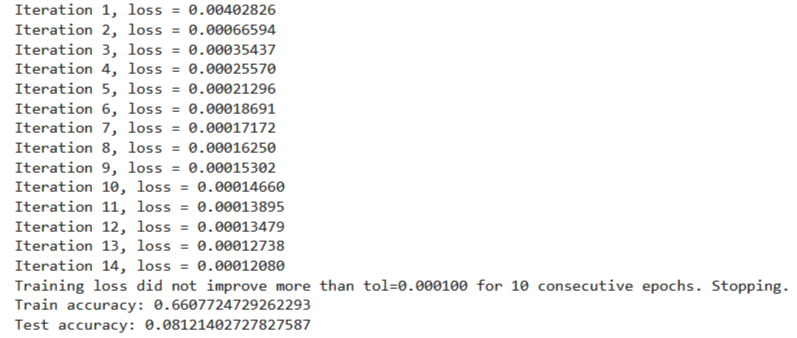
سپس مطابق دستور های قبلی، x و y را با .iloc مشخص می کنیم و به کمک train\_test\_split( ) با نسبت 80 درصد به داده های آموزشی و تست تقسیم می کنیم. در ادامه به کمک MaxMinScaler داده های آموزش و تست را استاندارد می کنیم. با تعریف scaler1 ابتدا با داده های train مربوط به x پارامترهای scaler را روی ان fit می کنیم. ایم کار با دستور .fir\_transform( ) انجام می شود و سپس برای داده های تست فقط نتایج را رو آن ها اعمال می کنیم که این کار با متد .transform( ) انجام می شود. در نهایت با یک scaler2 دیگر که با y\_train ، fit می شود، y\_test را هم استاندارد می کنیم.

در تصویر زیر ابتدا x\_train به صورت آرایه مشخص شده و در زیر آن y\_train. ؛ که مشاهده می شود داده ها استاندارد شده اند.



6\_ :در این مسئله می خواهیم، خروجی قیمت خانه را پیش بینی کنیم در نتیجه با یک مسئله رگرسیون کار داریم.پس مدل MLPRegressor با استفاده از دو لایه پنهان و لایه اول با 100 نورون و لایه دوم با 50 نورون مدل را مورد آزمایش قرار می دهیم. برای تابع بهینه ساز از adam استفاده می کنیم و برای داده های اعتبار سنجی هم ،validation\_fraction =0.1 ، 10 درصد داده های train را کنار می گذاریم. با verbose = true روند فعالیت شبکه را بررسی می کنیم. مانند قسمت های گذشته ابتدا مدل را با داده های آموزش آموزش می دهیم، با .predict خروجی پیش بینی شده شبکه را مشاهده می کنیم و با .score که دقت R2 score مربوط به رگرسیونی که در حال انجام دادن هستیم را می سنجیم.

اما نتایج به دست آمده مطلوب نمی باشد و دقت پایینی نشان می دهد در نتیجه طراحی شبکه با مشکل مواجهه شده است.



با توجه به نتایج بعد از دوره چهاردهم آموزش، فرایند یادگیری به دلیل نرخ یادگیری پایین در 10 دوره قبلی که با tol مشخص می شود، متوقف شدهکه نشان دهنده مشکل طراحی شبکه است. در اثر تغییر تعداد لایه ها و نعداد نورون های هر لایه نیز نتایج بهتری حاصل نشد.

همان طور که مشاهده می شود دقت بسیار کمی برای داده های تست به دست آمده است. در نتیجه نمودار های به دست آمده و ادامه مسئله نیز با مشکل رو به رو خواهد شد.

**سوال 5)**

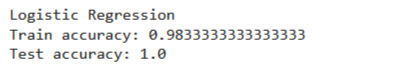
ابتدا با دستور load\_Iris(return X,y = True) دیتاست Iris که مربوط به گل iris با سه نوع کلاس است را در X وy قرار می دهیم. این دیتاست دارای 3 کلاس و 50 نمونه برای هر کلاس است که مجموعا 150 نمونه دارد. تعداد feature های هر نمونه نیز 4 است.

سپس داده ها را با نسبت 80 به 20 به train و test تقسیم می کنیم.

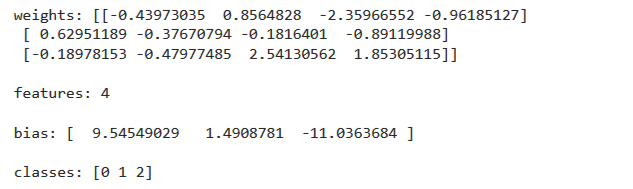
و 3 مدل Logistic Regression ، MLP Classifier و RBF را تعریف کرده و در هر مدل داده های آموزشی را روی مدل ها fit کرده و سپس با predict خروجی مدل به ورودی داده test را حساب می کنیم تا در ماتریس confusion به عنوان ورودی آن را لحاظ کنیم.

با استفاده از .score هم می توان دقت مدل روی داده های آموزشی و تست را محاسبه کرد.

Logistic Regression :

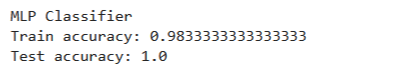


برای شاخص های مدل با دستور .coef و .intercept می توان به ترتیب وزن ها و بایاس مدل را مشاهده کرد. همچنین با دستور .features\_in تعداد feature ها و .classes تعداد کلاس های خروجی را مشاهده می کنیم.



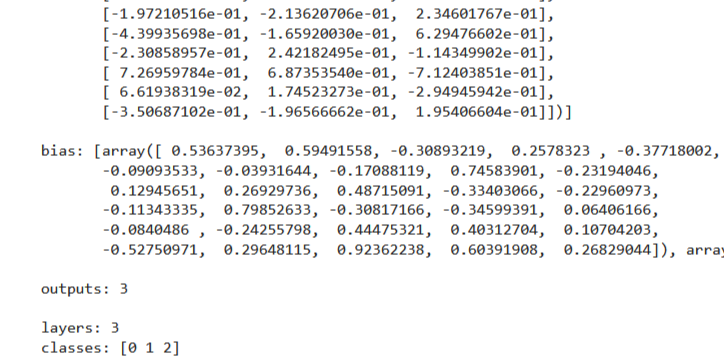
MLP Classifier:

برای این مدل یک لایه مخفی با 30 نورون در نظر می گیریم. تعداد دوره ها برای همگرایی کامل بهینه ساز، 800 و بهینه ساز را ‘adam’ انتخاب می کنیم.



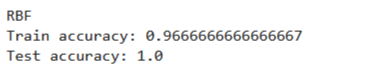
وزن ها و بایاس مشابه قبل مشاهده می کنیم. با دستور .n\_outputs و .n\_layers تعداد خروجی ها و تعداد کل لایه ها را مشاهده می کنیم. تعداد کلاس ها نیز مشابه قبل خواهد بود.

که تنها قسمتی از وزن ها در تصویر مشاهده می شود.

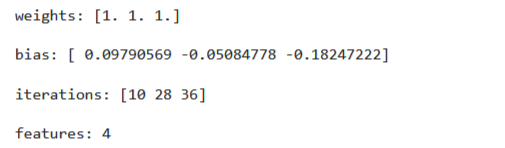


RBF :

پس از تعریف مدل مشابه با قبل دقت را محاسبه می کنیم.



وزن ها، بایاس و تعداد feature ها را مشابه قبل محاسبه کرده و با n\_iter تعداد دوره ها برای مدل را مشاهده می کنیم.



ماتریس confusion برای هر سه حالت مشابه به دست آمد.

