۱۳۰۷
دانشگاه صنعتی خواجه نصیرالدین طوسی
دانشکده مهندسی برق

مینی پروژه دوم

نام و نام خانوادگی:

فاطمه پاکدامن

شماره دانشجویی:

۹۹۲۴۷۵۳

استاد درس:

دکتر مهدی علیاری

درس:

مبانی سیستم‌های هوشمند

آذر ۱۴۰۲


بسم الله الرحمن الرحيم

فهرست مطالب

عنوان | صفحه

چکیده

این گزارش به بررسی داده‌ها و مدل‌های مختلف در تحلیل‌های یادگیری ماشین می‌پردازد و شامل مجموعه‌ای از مینی پروژه‌ها است. پروژه‌ها متناسب با چالش‌های متنوعی تعریف شده‌اند که از پیش‌پردازش داده‌های خام گرفته تا پیاده‌سازی و آموزش مدل‌ها و ارزیابی عملکرد آن‌ها می‌پردازد.

در بخش اول، با استفاده از داده‌های زیستی، یک مدل پرسپترون ساده آموزش داده شده و تأثیر تغییر آستانه بر روی عملکرد آن بررسی گردیده است. در بخش دوم، با به‌کارگیری نورون‌های McCulloch-Pitts، یک ضرب کننده باینری پایه پیاده‌سازی شده که توانایی انجام محاسبات پایه را دارد. سپس در بخش سوم، یک شبکه همینگ برای شناسایی الگو و استخراج ویژگی‌های کلیدی بررسی شده است.

بخش چهارم به پیش‌بینی قیمت خانه‌ها با استفاده از مدل Multi-Layer Perceptron (MLP) اختصاص یافته که به ویژه تأکید بر پیش‌پردازش داده‌ها، بررسی هم‌بستگی ویژگی‌ها و تأثیر آن‌ها بر قیمت خانه‌ها دارد. نتایج بیانگر بالاترین همبستگی برخی ویژگی‌ها با قیمت و ارائه راهبردهای بهینه‌سازی مدل است.

در بخش پنجم، داده‌های مربوط به گل‌های زنبق (مجموعه داده‌ای Iris) تجزیه‌وتحلیل شده‌اند. سه روش مختلف برای طبقه‌بندی شامل رگرسیون لجستیک، MLP و شبکه‌های عصبی پایه شعاعی (RBF) استفاده شده و نتایج تحلیلی ارائه شده‌اند. تجزیه و تحلیل‌ها شامل ماتریس‌های درهم‌ریختگی و ارزیابی‌های دقیق با شاخص‌های استانداردی مانند Precision، Recall و F1-Score هستند. این مجموعه بررسی‌ها به درک بهتر و توسعه مدل‌های پیچیده‌تر در آینده کمک خواهد کرد و پیش‌بینی‌های دقیق‌تری را ممکن می‌سازد.

بخش اول

مقدمه

در ابتدا با استفاده داده‌های داده شده دانلود شده و تعدادی از داده‌ها نمایش داده می‌شود سپس پیش پردازش لازم برای آموزش و همچنین تقسیم داده‌ها به دسته آموزش و تست انجام می‌شود. پس از آن یک نورون بر روی آن آموزش داده شده و سپس نتایج حاصل نمایش داده شده‌اند. در نهایت تاثیر آستانه بر روی نورون برسی شده است.

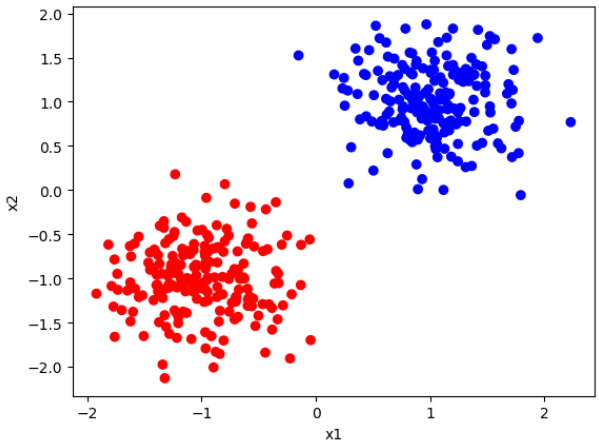
۱. تقسیم داده‌ها و آموزش یک نورون

ابتدا با استفاده از دستور os.makedirs یک فولدر مربوط به داده‌های این پروژه در colab ایجاد شده و سپس با استفاده از دستور gdown! داده‌ها دانلود شده و به این پوشه منتقل شده‌اند. در نهایت فایل csv با استفاده از کتابخانه pandas خوانده شده و ۵ داده اول نمایش داده شده است.

```
if not os.path.exists("Mp2_1"):
    os.makedirs("Mp2_1")
#download dataset and move data to mp2 folder
!gdown 1-KWQDy-8MkXX7s5JHyzkYUhEpN0YI1m1
!mv '/content/Perceptron.csv' '/content/Mp2_1'
#import data and display first 5
perceptron_data = pd.read_csv("/content/Mp2_1/Perceptron.csv")
perceptron_data.head()
```

	x1	x2	y
0	1.028503	0.973218	-1.0
1	0.252505	0.955872	-1.0
2	1.508085	0.672058	-1.0
3	1.940002	1.721370	-1.0
4	-1.048819	-0.844999	1.0

در این مجموعه داده مشاهده می‌شود که دو برچسب ۱ و ۱- وجود دارد و مسئله طبقه‌بندی دو کلاسه است. در این میان داده ها دارای ۲ ویژگی هستند. به منظور آموزش راحت تر برچسب ها به ۱ و ۰ تغییر کرده‌اند. نمایشی از این داده‌ها در نمودار ۱ آمده است.



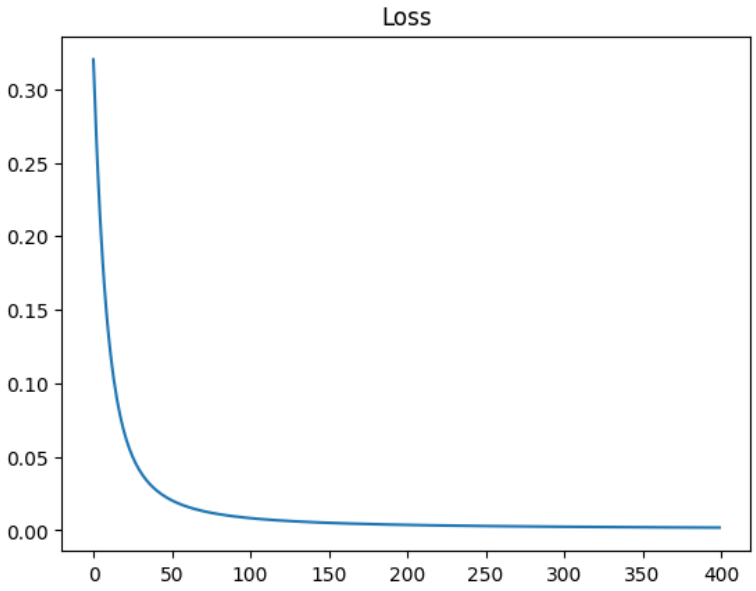
نمودار ۱) نمایشی از داده‌ها بر حسب ویژگی آن‌ها

به منظور تقسیم داده‌ها به آموزش و تست از تابع آماده train_test_split استفاده شده است. تعداد هر کدام از این دو دسته در زیر نمایش داده شده ست.

```
#train =  320
#test =  80
persentage of instaneces with label 1 in train set =  50.0
persentage of instaneces with label 1 in test set =  50.0
```

پس از اماده سازی داده‌ها برای آموزش، به منظور ساده کردم فرایند آموزش از کلاس استفاده شده است. در این کلاس تابع init به منظور رجیستر کردن پارامترهای ورودی، predict برای انجام پیش بینی بر روی داده های وارد شده، decision_function به منظور رسم ناحیه تصمیم‌گیری، fit برای آموزش، gradient وgradient_descent برای گرفتن مشتق و اپدیت وزن‌ها و بایاس و در نهایت دو فانکشن برای نمایش شی ساخته شده از این کلاس(__repr__) و نمایش به وزن‌ها (parameters) نوشته شده است.

پس از این کار یک نرون از این کلاس ساخته شده و با استفاده از داده‌های آموزش، آموزش دیده است. نتایج حاصل در نمودار ۲ که نمودار loss حاصل نمایش داده شده است.

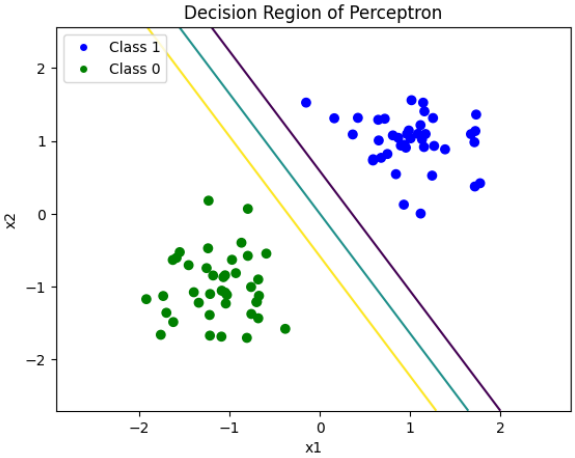


نمودار ۲) نتایج آموزش نرون پرسپترون

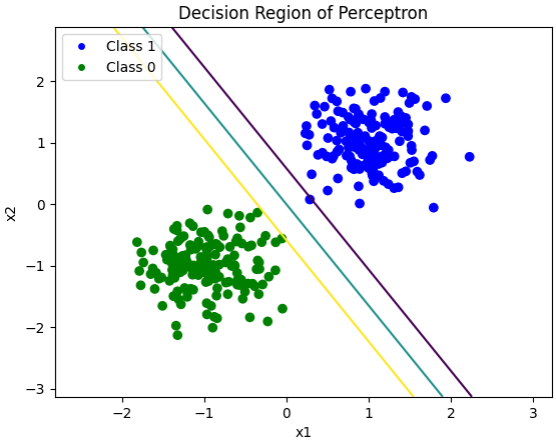
۲. ناحیه تصمیم گیری

حال برای نمایش ناحیه تصمیم‌گیری بدست آمده حاصل از این آموزش این نورون، تابعی به منظور رسم آن و نمایش داده‌ها نوشته شده است. این تابع بردار ویژگی و پرچسب آن و همچنین نرون آموزش دیده را به عنوان ورودی گرفته و با پیدا کردن حداقل ئ حداکثر هر ویژگی و تولید داده در این میان و در نهایت اعمال پیش بینی بر این نقاط، ناحیه تصمیم گیری پیدا و رسم می‌شود.

شکل ۱ ناحیه تصمیم گیری برای داده تست و شکل دو ناحیه تصمیم گیری داده‌های تست را نشان



شکل ۱) ناحیه تصمیم گیری برای داده تست

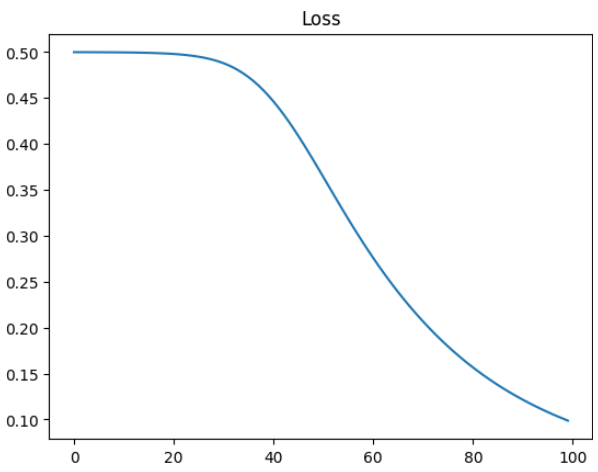


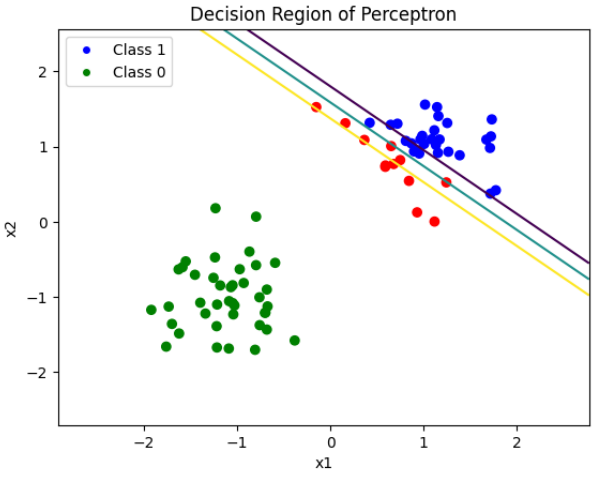
شکل ۲) ناحیه تصمیم گیری برای داده آموزشی

می‌دهد.

۳. تاثیر تغییر آستانه

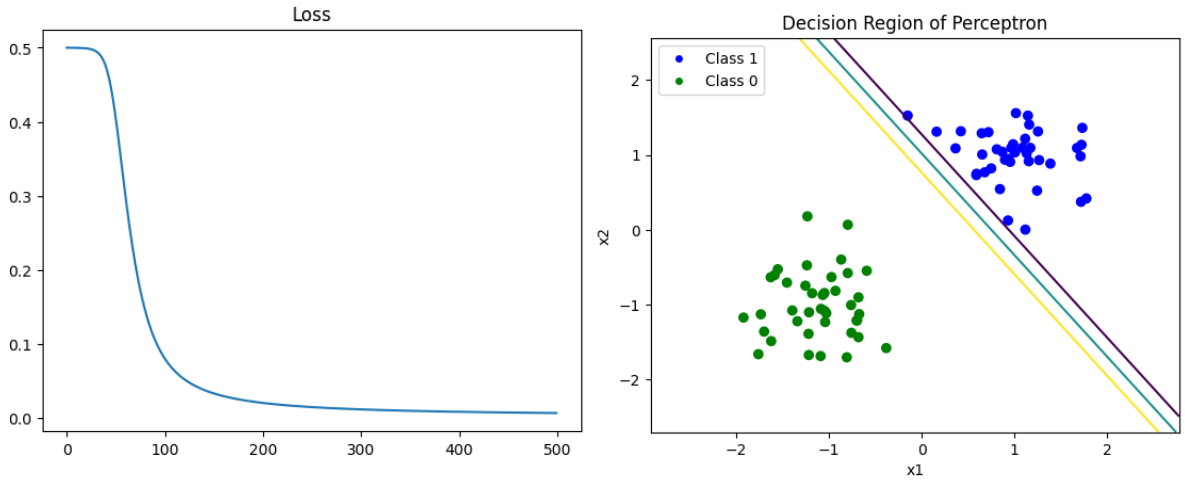
به منظور بررسی تغییر آستانه بر آموزش، پارامتر دیگری به عنوان threshold به نورون اضافه شده است. این آستانه هنگام پیش بینی با پیش بینی انجام شده جمع می‌شود. در شکل ۳ نتایج حاصل با آستانه ۱۰ قابل مشاهده است. همان طور که انتظار می‌رفت، با اعمال آستانه ناحیه تصمیم گیری به یک سمت منحرف شده است که دلیل آن این است که در پیشبینی، مقدار ثابتی با آن جمع می‌شود که ناحیه تصمیم گیری را به سمت یک کلاس هل می‌دهد. با توجه به این که نورون مورد بررسی دارای بایاس و وجود آستانه با این پارامتر قابل جبران است، در شکل ۴ که نرون با همان آستانه و در تعداد دور های بیشتر آموزش دیده است،





شکل ۳) نمودار خطا و ناحیه تصمیم گیری با اعمال آستانه

مقداری از خطای ایجاد شده به علت وجود آستانه کم شده است. در صورت حذف بایاس این آستانه اعمال شده غیر قابل جبران خواهد بود به عبارت دیگر اگر آستانه‌ای وجود داشته باشد نورون نمی‌تواند این پارامتر را یاد بگیرد.



شکل ۴) با افزایش تعداد دوره آموزش آستانه اعمال شده جبران شده است

مشاهده می‌شود که پس از گذشت حدود ۲۰۰ دوره آموزش، آستانه اعمال شده تا حدی جبران شده و داده‌های تست همگی به درستی طبقه بندی شده‌اند.

بخش دوم

ضرب کننده باینری با استفاده از نورون McCulloch-Pitts

مقدمه

در این بخش به کمک نورون McCulloch-Pitts یک ضرب کننده‌ی باینری پیاده سازی شده است.

۱. رسم جدول ورودی-خروجی

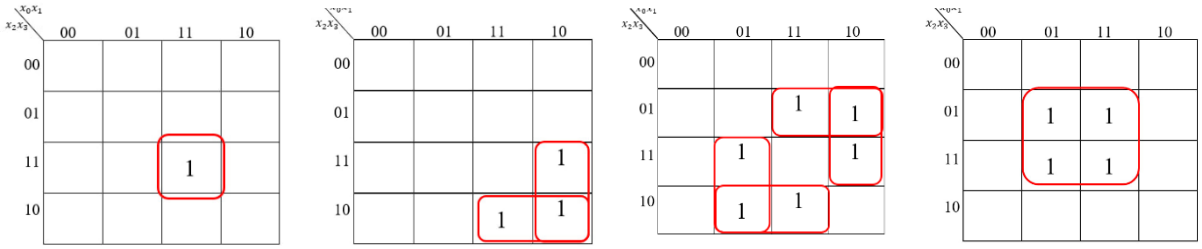
جدول ورودی و خروجی یک ضرب کننده باینری در جدول ۱ آمده است بدین صورت که با ضرب دو ورودی اول در دو ورودی دوم حاصل بدست می‌آید.

در جدول ۲، جدول کارنو به ازای هر خروجی رسم شده و هر خروجی با کمترین تعداد and و or بدست آمده است. با توجه به آنکه می‌توان هر کدام از این گیت‌ها را با یک نورون مدل کرد، پس تعداد مود نیاز این شبکه با تعداد گیت‌های مورد نیاز برای پیاده سازی برابری می‌کند.

جدول ۱) ورودی و خروجی

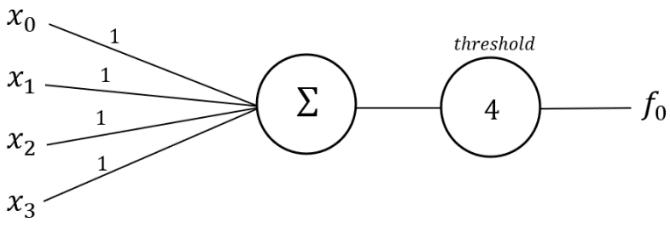
x_0	x_1	x_2	x_3	f_0	f_1	f_2	f_3
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

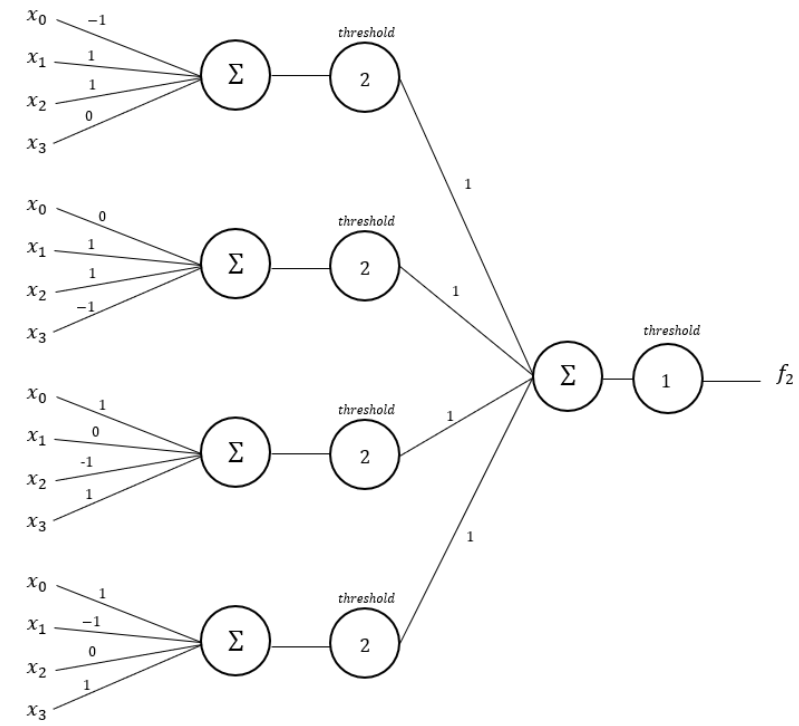
جدول ۲) جداول کارنو ضرب کننده باینری برای طراحی با کمترین تعداد نورون

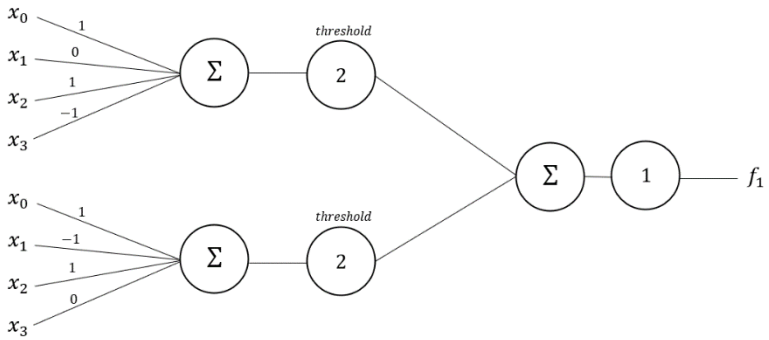


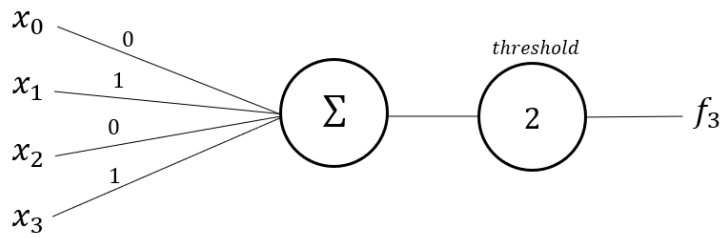
$$f_0 = x_0x_1x_2x_3 \quad f_1 = x_0x_2x_3' + x_0x_1'x_2 \quad f_2 = x_1x_2x_3' + x_0x_2'x_3 \quad f_3 = x_1x_3$$
$$x_0'x_1x_2 + x_0x_1'x_3$$

با توجه روابط نوشته شده بر اساس جدول کارنو، نورون‌های زیر پیاده سازی می‌شود.









همچنین می‌توان با حذف کردن ورودی‌هایی که وزن صفر دارند شبکه را ساده‌تر کرد.

۳. پیاده سازی شبکه

برای پیاده سازی این شبکه با استفاده از نورون McCulloch-Pitts در زبان پایتون، ابتدا یک کلاس برای این نوون تعریف شده است که شامل دو تابع است. در تابع مدل از این کلاس ضرب وزن‌ها در ورودی انجام شده و با آستانه مقایسه می‌شود و در صورت بزرگتر یا مساوی بودن حاصل از آستانه مقدار یک برگردانده می‌شود و در غیر این صورت مقدار صفر را برمی‌گرداند.

در ادامه برای پیاده سازی ضرب کننده، مشابه با طراحی انجام شده در قسمت قبل، با استفاده از ۱۰ نورون پیاده سازی شده است. این فرایند برای امکان تکرار پذیری در یک تابع نوشته شده است. ابتدا نورون‌ها ساخته شده و وزن آن‌ها داده شده است. پس از آن مدل بر روی آن فراخوانی شده و ورودی به آن داده می‌شود. در قسمت‌هایی که طراحی شامل دو لایه است خروجی گروهی از مدل‌ها به عنوان ورودی نورون نهایی داده شده است و در نهایت لیستی شامل ۴ خروجی مشخص شده توسط این تابع برگردانده می‌شود.

به منظور برسی عملکرد این تابع بر روی همه ورودی‌های تعریف شده، لازم است این ورودی‌ها تولید شوند که این کار با استفاده از itertools.product صورت گرفته است. در نهایت با طی یک حلقه for بر این ورودی‌ها، همه خروجی‌ها تولید شده و پرینت می‌شوند. خروجی حاصل در زیر آمده است.

```
x =  (1, 1, 1, 1) multply [1, 0, 0, 1]
x =  (1, 1, 1, 0) multply [0, 1, 1, 0]
x =  (1, 1, 0, 1) multply [0, 0, 1, 1]
x =  (1, 1, 0, 0) multply [0, 0, 0, 0]
x =  (1, 0, 1, 1) multply [0, 1, 1, 0]
x =  (1, 0, 1, 0) multply [0, 1, 0, 0]
x =  (1, 0, 0, 1) multply [0, 0, 1, 0]
x =  (1, 0, 0, 0) multply [0, 0, 0, 0]
x =  (0, 1, 1, 1) multply [0, 0, 1, 1]
x =  (0, 1, 1, 0) multply [0, 0, 1, 0]
x =  (0, 1, 0, 1) multply [0, 0, 0, 1]
x =  (0, 1, 0, 0) multply [0, 0, 0, 0]
x =  (0, 0, 1, 1) multply [0, 0, 0, 0]
x =  (0, 0, 1, 0) multply [0, 0, 0, 0]
x =  (0, 0, 0, 1) multply [0, 0, 0, 0]
x =  (0, 0, 0, 0) multply [0, 0, 0, 0]
```

همانطور که پیشتر گفته شد، می‌توان با حذف کردن ورودی‌هایی که وزن صفر دارند شبکه را ساده‌تر شده است و خروجی مشابه حاصل شده است.

بخش سوم

شبکهٔ عصبی همینگ

۱. داده‌های مورد بررسی

داده‌های مورد بررسی شامل ۵ تصویر حروف الفبای فارسی است. دو تابع نوشته شده در ادامه توضیح داده شده است.

۱.۱. تبدیل تصویر به دودویی

تابع convertImageToBinary که در کد داده شده است، طراحی شده تا یک فایل تصویری را از مسیر مشخص شده دریافت کند و آن را بر اساس شدت هر پیکسل به نمایشی دودویی تبدیل کند.

در گام اول با استفاده از دستور Image.open از کتابخانه PIL (Python Imaging Library) برای باز کردن یک تصویر از مسیر فایل داده شده در ورودی تابع استفاده شده است. سپس یک شی ImageDraw.Draw ایجاد شوده، که امکان ویرایش تصویر را فراهم می کند. پس از آن عرض و ارتفاع تصویر (به پیکسل) تصویر باز شده در width و height زخیره می‌شود و متغیر pix برای دسترسی برای تغییر مقادیر پیکسل تصویر ایجاد شده است. یک آستانه تعریف شده است تا شدت تفاوت بین سفید و سیاه را تعیین کند. برای نمایش دودویی یک لیست خالی به نام binary_representation برای نگه داشتن فرم دودویی تصویر که در آن ۱- نشان دهنده یک پیکسل سفید و ۱ نشان دهنده یک پیکسل سیاه است، ایجاد شده و دو حلقه تودرتو هر پیکسل در تصویر را برسی می‌کنند. مقادیر RGB هر پیکسل بیرون کشیده می شود و total_intensity با مجموع مقادیر RGB محاسبه می شود و در نهایت یک آستانه شدت برای تعیین اینکه آیا پیکسل باید سفید یا سیاه در نظر گرفته شود اعمال می شود. بر اساس شدت کلی در مقایسه با آستانه، رنگ پیکسل به سفید با مقادیر RGB (۲۵۵, ۲۵۵, ۲۵۵) تعیین می‌شود و به لیست دودویی به عنوان ۱- اضافه می‌شود، یا به سیاه با مقدار RGB (۰, ۰, ۰) و به عنوان ۱ اضافه می‌شود. پس از آن رنگ هر پیکسل با استفاده از روش draw.point به رنگ سیاه یا سفید به روز می‌شود. لیست binary_representation که نسخه دودویی تصویر است توسط این تابع برگردانده می‌شود.

پیاده سازی ساده‌تر این تابع با استفاده از کتابخانه opencv انجام شده است. ابتدا تصویر خوانده شده، به تصویر سیاه و سفید تبدیل می‌شود. پس از آن با اعمال یک آستانه به نمایش دودویی تبدیل می‌شود. به منطور ایجاد لیست با مقادیر صفر و یک از کتابخانه numpy استفاده شده، بین صورت که اگر مقادیر بیشتر از مقدار آستانه باشد ۱- و در غیر این صورت ۱ در بردار زخیره می‌شود. تفاوت در این است که به جای برسی تک پیکسل، تمام عملیات بر روی همه پیکسل ها در یک دستور صورت می‌گیرد.

۲.۱. افزودن نویز به داده‌ها

کد مربوط به این قسمت یک مجموعه از تصاویر را گرفته و با افزودن نویز به هر پیکسل، نسخه‌های دارای نویز از آن‌ها را ایجاد و ذخیره می‌کند.

در تابع generateNoisyImages لیستی از مسیرهای فایل تصاویر را تعریف شده است و روی این لیست حلقه زده و برای هر تصویر، تابع getNoisyBinaryImage را فراخوانی می‌کند. در نهایت پس از ایجاد هر تصویر نویز دار، یک پیغام چاپ می‌شود که اعلام می‌کند تصویر با نویز برای هر مسیر تصویر ایجاد و ذخیره شده است.

تابع getNoisyBinaryImage تابع اصلی افزودن نویز به تصویر است. مشابه قسمت قبل در گام اول اقدام به باز کردن تصویر ورودی بر اساس مسیر داده شده، ایجاد ابزار ویرایش و زخیره طول و عرض تصویر به منظور ایجاد حلقه for و برسی همه پیکسل ها انجام شده است. پس از آن یک استانه برای اعمال نویز تعیین شده است. سپس برای هر پیکسل در عرض و ارتفاع تصویر، یک مقدار تصادفی نویز به مقادیر رنگ قرمز، سبز و آبی اضافه می‌شود. از یک مقدار معین برای اطمینان از اینکه مقادیر RGB پیکسل‌ها از حداقل و حداکثر مجاز (۰ تا ۲۵۵) فراتر نروند، استفاده شده است. درنهایت رنگ هر پیکسل به روز شده و تغییر می‌کند. و تصویر با نویز به یک فایل جدید با فرمت JPEG ذخیره می‌شود.

به منظور پیاده سازی ساده تر با کتابخانه opencv برای افزودن نویز به تصویر یک ماتریس نویز به اندازه تصویر ایجاد و سپس اعدادی رندم بین قدرمطلق آستانه تعریف شده در این ماتریس زخیره می‌شود و در نهایت به تصویر اصلی افزوده می‌شود. مشاهده می‌شود که خروجی در دو مورد یکسان است. نمونه ای از خروجی نویزی در تصویر زیر قابل مشاهده است.

```
image = Image.open("/content/noisy1.jpg")
image
```

۴. شبکه همینگ

به منظور پیاده سازی شبکه ابتدا توابعی پیاده سازی شده تا روند کار راحت تر صورت بگیرد. شامل چند تابع که برای انجام عملیات روی بردارها و ماتریس‌ها پیاده سازی شده. در ادامه توضیح هر کدام از توابع ارائه شده‌است.

show(matrix) یک ماتریس را می‌گیرد و عناصر آن را به صورت قالب‌بندی شده چاپ می‌کند. هر عنصر به گونه‌ای فرمت داده شده‌است تا تا سه رقم اعشار نشان داده شود. print(sep="") برای چاپ یک خط جدید پس از هر ردیف ماتریس استفاده می‌شود، بنابراین هر ردیف ماتریس روی یک خط جداگانه چاپ می‌شود. change(vector, a, b) طوری طراحی شده‌است تا یک لیست ساده (بردار) را به یک ماتریس با ابعاد a در b تبدیل کند. product(matrix, vector, T) یک ماتریس را در یک بردار ضرب می‌کند. این ضرب به صورت ضرب سطر به سطر با بردار است. `T` یک پارامتر آستانه است که به هر نتیجه ضرب سطری اضافه می‌شود. و در نهایت بردار نتیجه از این عملیات برگردانده می‌شود. action(vector, T, Emax) یک تابع فعال‌ساز را بر هر عنصر بردار ورودی اعمال می‌کند. عناصر با قوانین زیر تغییر می‌کنند.

- اگر مقدار کمتر یا مساوی ۰ باشد، به ۰ تنظیم می‌شود.
- اگر مقدار مثبت و کمتر از یا مساوی با T باشد، در Emax ضرب می‌شود.
- اگر مقدار از T بیشتر باشد، در سطح T محدود می‌شود.
- بردار پردازش‌شده پس از آن برگردانده می‌شود.

mysum(vector, j) مجموع همه عناصر را در یک بردار به جز عنصر در ایندکس j محاسبه می‌کند.

norm(vector, p) فاصله اقلیدسی اختلاف بین دو بردار را محاسبه می‌کند، که اساساً فاصله بین آن‌ها در این فضای است.

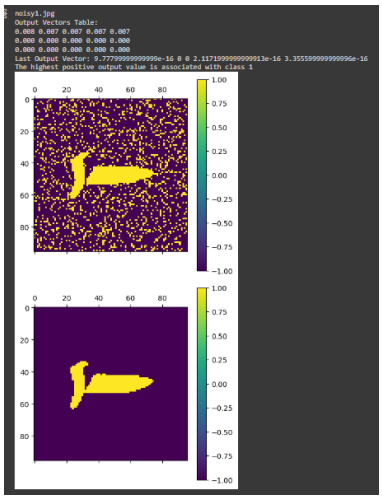
در یک نمونه تصویر ۵ انتخاب شده و در چندین مرحله تصاویر را پردازش می‌کنند، آنها را به فرم باینری (دودویی) تبدیل می‌کنند، برای دسته‌بندی از آن ها برای ایجاد یک ماتریس وزن استفاده می‌کنند، و با استفاده از یک تابع فعالساز از آن می‌گذرند تا دسته‌بندی را اصلاح کنند.

به طور خلاصه نمایش باینری تصویر ورودی IMAGE_PATH در y ذخیره می‌شود. سپس نمایش باینری تصویر ورودی به یک ماتریس با استفاده از تابع change تبدیل می‌شود. این ماتریس با استفاده از matshow از کتابخانه matplotlib نمایش داده شده و با استفاده از colorbar یک نوار رنگ اضافه می‌شود.

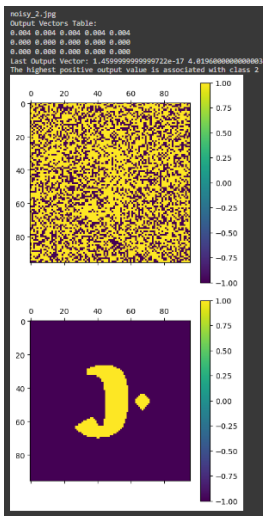
پس از آن یک ماتریس وزن W برای تصاویر نمونه ایجاد شده، با هر مقدار در ماتریس نصف مقدار باینری مرتبط با تصویر باشد. E به عنوان ماتریس اتصال سیناپسی تعریف شده، با ورودی‌های قطری تنظیم شده بر ۱ و سایر مقادیر به مقدار کوچک منفی e.

بردار خروجی s به عنوان نتیجه تابع محصول اولیه شروع شده و تابع action روی آن اجرا می‌شود. کد شامل فرآیند تکراری است که در آن نتیجه y به طور مداوم به روز می‌شود تا زمانی که تفاوت، که توسط تابع norm محاسبه می‌شود، کمتر از Emax باشد.

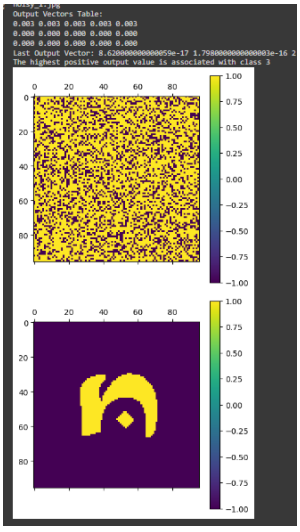
پس از حلقه، کد بردار خروجی، وزن‌ها و خطای هر تکرار را چاپ می‌کند. طبقه نهایی با گرفتن شاخص حداکثر مقدار در آخرین بردار خروجی تعیین می‌شود. متناظر نمایش باینری تصویر نمونه مربوطه به یک ماتریس برای تصویرسازی تبدیل می‌شود و سپس نمایش داده می‌شود. در نهایت اگر خروجی تعیین شده باشد، ماتریس نتیجه را به شکل تصویر نمایش داده می‌شود، که نشان دهنده طبقه‌ای با بیشترین مقدار خروجی است. تعدادی از این خروجی‌ها در زیر آمده است.



Noise factor = 100



Noise factor = 800

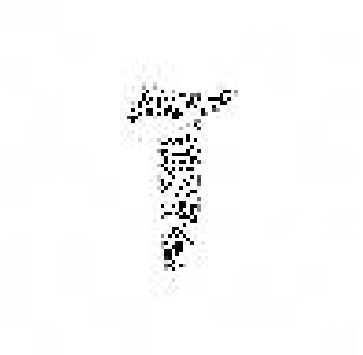


Noise factor = 1000

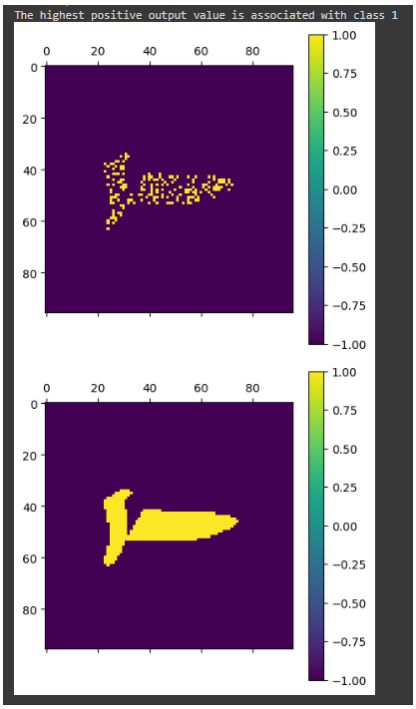
مشاهده می‌شود که مدل با افزایش نویز تا ۱۰۰۰ عملکرد خوبی نخواهد داشت.

۵. خروجی‌های دارای Missing Point

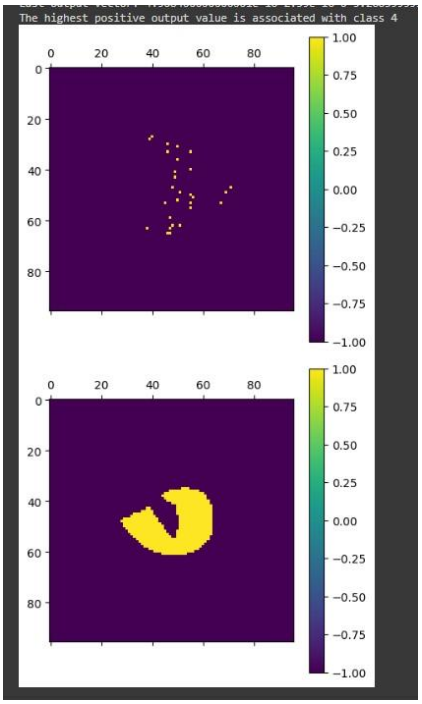
برای ایجاد خروجی‌های دارای Missing Point از تابعی که جهت نویزی کردن تصاویر نوشته شده بود استفاده شد با این تفاوت که این تابع دو ورودی با عنوان تعداد نقاط گمشده و درصد نقاط مشکی که به سفید تبدیل می‌شوند را داراست. پس از اعمال نویز به تصویر به تعداد مشخص شده نقطه رندم در تصویر انتخاب شده و سپس این نقاط به رنگ تغییر می‌کنند. نمونه ای از خروجی در زیر آمده است.



شبکه همینگ که در قسمت قبل توضیح داده شد، بر روی این تصاویر دارای نقاط گمشده اعمال شد. دو تصویر داده شده یکی با ۱۰۰۰ و یکی با ۳۰۰۰ نقطه گمشده هستند که هر دو دارای عدد نویز ۵ هستند. با افزایش تعداد نقاط گمشده تا ۳۰۰۰ می‌بینیم که شبکه دیگر قادر به پیشبینی صحیح نیست.



#mising points =1000



#mising points =3000

بخش چهارم

پیش بینی قیمت خانه

مقدمه

در این قسمت تلاش بر پیش‌بینی داده‌های قیمت خانه ارائه شده است. ابتدا داده‌ها دانلود، طبقه‌بندی و پیش‌پردازش شده‌اند تا برای استفاده در مدل‌های یادگیری ماشین آماده‌باشند. مشاهدات ابتدایی نشان داده‌اند که متغیرهای مختلف چطور با قیمت خانه‌ها در ارتباط هستند. از این رو، ماتریس همبستگی برای درک بهتر این روابط ترسیم شده است. آموزش مدل Multi-Layer Perceptron با استفاده از TensorFlow و Keras ساخته و آموزش داده شد. مدل‌ها با توابع اتلاف و بهینه‌سازهای مختلف ارزیابی شدند و در نتیجه، داده‌ها به صورت دسته‌های آموزشی و آزمایشی تفکیک و مورد بررسی اتلاف قرار گرفتند.

۱. خواندن فایل داده‌ها

در ابتدا فایل داده‌ها با که در درایو قرار داده شده است، با استفاده از gdown دانلود و سپس به فولدر مربوط به تمرین با استفاده از mv منتقل شده است. پس از آن فایل از حالت فشرده خارج شده و فایل اکسل مرتبط با داده‌ها خوانده شده است و ۵ نمونه از این داده‌ها نمایش داده شده است.

با فراخوانی تابع info. بر روی مجموعه دیتایی که خوانده شد، خلاصه‌ای از اطلاعات این داده‌ها قابل برسی است. نتیجه فراخوانی این تابع بر روی مجموعه داده قیمت خانه در زیر آمده است. می‌توان مشاهده کرد که این مجموعه شامل ۴۶۰۰ داده است که ۱۸ ستون دارد. با توجه به آن که هدف پیش بینی قیمت است، یک ستون به عنوان برچست و ۱۷ ستون باقی مانده را به عنوان ویژگی در نظر گرفته می‌شود. قابل مشاهده است که برخی از این ویژگی‌ها مانند شهر یا کشور یا تاریخ غیر عددی هستند. تعداد داده نال در این مجموعه با توجه به تعداد ۴۶۰۰ داده در هر ستون صفر است.

در گام دوم تعداد داده‌های نال بر حسب ستون با فراخوانی دستور isnull().sum(). نمایش داده شده است که نتایج آن حاکی از آن است که هیچ داده نالی در این مجموعه داده وجود ندارد.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           4600 non-null   object
 1   price          4600 non-null   float64
 2   bedrooms       4600 non-null   float64
 3   bathrooms      4600 non-null   float64
 4   sqft_living    4600 non-null   int64
 5   sqft_lot       4600 non-null   int64
 6   floors         4600 non-null   float64
 7   waterfront     4600 non-null   int64
 8   view           4600 non-null   int64
 9   condition      4600 non-null   int64
 10  sqft_above     4600 non-null   int64
 11  sqft_basement  4600 non-null   int64
 12  yr_built       4600 non-null   int64
 13  yr_renovated   4600 non-null   int64
 14  street         4600 non-null   object
 15  city           4600 non-null   object
 16  statezip       4600 non-null   object
 17  country        4600 non-null   object
dtypes: float64(4), int64(9), object(5)
memory usage: 647.0+ KB
```

```
[ ] housedata.isnull().sum()

date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
street           0
city             0
statezip         0
country          0
dtype: int64
```

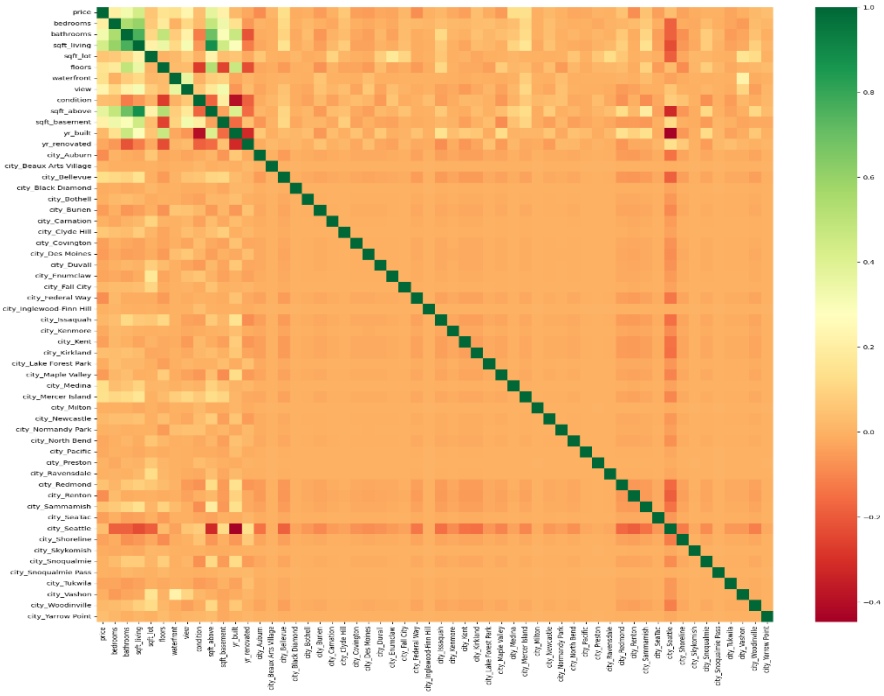
۲. ماتریس هم بستگی

با توجه به وجود داده غیر عددی در مجموعه، پیش از رسم ماتریس همبستگی پیش پردازشی بر داده‌ها انجام شده است. به این منظور ابتدا ستون کشور حذف شده است چرا که در همه داده‌ها یکسان است. پس از آن اقدام به حذف ستون خیابان نیز شده به دلیل آن که نسبت به تعداد داده‌ها تنوع تعداد خیابان ها زیاد بوده و یادگیری این مورد برای مدل حتی در صورت وجود همبستگی با قیمت دشوار است.

در قسمت قبل مشاهده شد که ستون شامل کد پستی از جنس عدد نیست، با مشاهده داده متوجه می‌شویم که شامل یه قسمت از جنس رشته و قسمتی عددی است که قسمت اول در همه داده‌ها مشترک است. با توجه به این که این دوقسمت با یک فاصله از هم جدا شده‌اند، ما نیز با برسی همه این داده‌ها در یک حلقه قسمت عددی را جایگزین این ستون کرده‌ایم.

ستون شهر نیز ستون دیگر بود که غیر عددی بود. با توجه به تعداد نسبتا قابل قبول داده در هر شهر علارغم عدم وجود تعادل این ستون به صورت one-hot کد گذاری شده و در تعداد ستون به تعداد شهرها ثبت شده است.

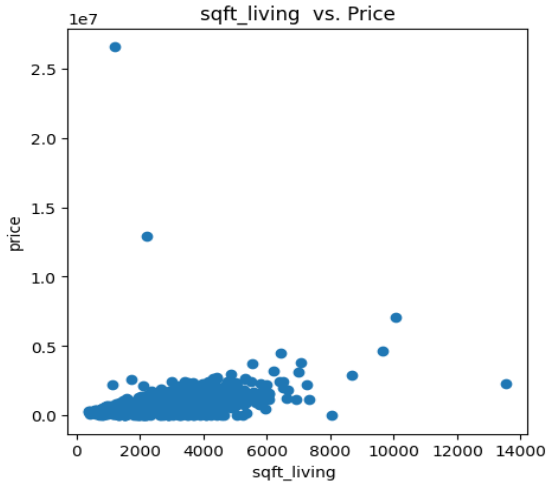
پس از انجام این پیش پردازش، میزان همبستگی قیمت با همه ستون ها برسی و چاپ شده است و نشان می‌دهد که قیمت با ویژگی sqft_living، sqft_above، و تعداد اتاق خواب به ترتیب بیشترین همبستگی را دارد. در نهایت برای دیداری کردن این مورد. ماتریس همبستگی رسم شده است و مطابق شکل ۵ است. نتایج حاصل مشابه قبل است.



شکل ۵) نمایی از ماتریس همبستگی

۳. نمودار توزیع قیمت

برای رسم نمودار توضیع قیمت بر حسب متراژ خانه، این دو ستون به ترتیب به عنوان محور عمودی و افقی در نظر گرفته شده و داده‌های مربوطه رسم شده‌اند که در نمودار آمده است. مشاهده می‌شود که در اکثر موارد با افزایش متراژ خانه قیمت نیز افزایش می‌یابد.



نمودار ۳) نمودار توزیع قیمت بر حسب متراژ

۴. مرتب کردن داده‌ها

خواسته شده ستون ماه و سال از ستون تارخ استخراج شود. با نگاهی بن ستون تاریخ مشاهده می‌شود که اگر با در نظر گرفتن خط تیره (-) به عنوان عامل جدا کننده بر خورد کنیم، قسمت اول جدا شده سال و قسمت دوم ماه را تشکیل می‌دهد. با اعمال یک حلقه بر وی داده‌های این ستون این مقادیر به ترتیب در یک لیست زخیره می‌شود و در نهایت به عنوان ستون یک و دو به مجموعه داده اضافه می‌شوند. پس از آن ستون مربوط به تاریخ حذف می‌شود. نیم نگاهی به این مجموعه داده پس از پردازش در زیر آمده است.

	Year	Month	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	city	Zipcode
0	2014	05	2014-05-02 00:00:00	313000.0	3.0	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	Shoreline	98133
1	2014	05	2014-05-02 00:00:00	2384000.0	5.0	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	Seattle	98119
2	2014	05	2014-05-02 00:00:00	342000.0	3.0	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	Kent	98042
3	2014	05	2014-05-02 00:00:00	420000.0	3.0	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	Bellevue	98008

۵. تقسیم و نرمالایز داده‌ها

با توجه به این که در این قسمت داده‌ها برای آموزش آماده می‌شوند، با توجه به ماتریس همبستگی و عدم وجود همبستگی زیاد بین شهر و قیمت، ستون شهر از مجموعه داده حذف شده است. سپس استفاده از LabelEncoder داده‌هایی از جنس شی به داده عددی تبدیل شده‌اند تا در فرایند آموزش مشکلی ایجاد نشود.

پس از آن قیمت و باقی ستون ها جدا شده و به خروجی و ورودی جدا شده است. پس از آن تقسیم به تست و آموزش انجام شده است. و ۲۰ درصد در تست قرار گرفته است. مشاهده می‌شود که تعداد داده‌ها در قسمت آموزش برابر ۳۶۸۰ و در تست برابر با ۹۲۰ است. حداقل و حداکثر قیمت نیز استخراج شده تا پس از نرمالایز کردن قابلیت برگشت به مقادیر اصلی وجود داشته باشد.

```
#train =  3680
#test =  920
```

در نهایت یک شی MinMaxScaler() ساخته شده و همه داده‌ها نرمال شده است و بین صفر و یک قرار گرفته‌اند. نمونه ای از این داده‌ها در زیر آمده است.

```
(array([[0.        , 0.5       , 0.375     , ..., 0.54385965, 0.        ,
         0.07894737],
        [0.        , 1.        , 0.5       , ..., 0.21052632, 0.        ,
         0.72368421],
        [0.        , 0.5       , 0.75      , ..., 0.50877193, 0.97914598,
         0.17105263],
        ...,
        [0.        , 0.5       , 0.5       , ..., 0.50877193, 0.99503476,
         0.61842105],
        [0.        , 0.5       , 0.5       , ..., 0.87719298, 0.        ,
         0.48684211],
        [0.        , 0.        , 0.375     , ..., 0.90350877, 0.        ,
         0.22368421]]),
 array([[0.01504325],
        [0.02068447],
        [0.01413689],
        ...,
        [0.0319669 ],
        [0.02726589],
        [0.02329823]]))
```

۶. مدل Multi-Layer Perceptron

مدل در نظر گرفته شده برای آموزش شامل ۳ لایه پنهان به ترتیب با ۴۰، ۲۵ و ۱۰ نورون و یک لایه خروجی با ۱ نرون است. با توجه به این که خروجی یک عدد است، خروجی باید تنها یک نورون داشته باشد. خلاصه‌ای از مدل در زیر آمده است. که تعداد لایه، نوع آن و تعداد نورون در هر لایه و تعداد پارامترهای قابل آموزش و غیر قابل آموزش (ثابت) قابل مشاهده است. این مدل به کمک تنسورفلو ساخته شده است. مدل Sequential یک ترکیب خطی از لایه‌ها است، و می‌توان لایه‌ها را به صورت تکی اضافه کرد. به این صورت سه لایه گفته شده به مدل اضافه شده است. برای لایه‌های پنهان تابع فعال ساز relu و برای لایه خروجی تابع فعال ساز خطی در نظر گرفته شده است چرا که خروجی قیمت است و نیاز است که همه بازه را شامل شود. با توجه به شکل بردار ویژگی‌ها، ورودی لایه اول تعیین شده است.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 40)                640

 dense_9 (Dense)             (None, 25)                1025

 dense_10 (Dense)            (None, 10)                260

 dense_11 (Dense)            (None, 1)                 11

=================================================================
Total params: 1936 (7.56 KB)
Trainable params: 1936 (7.56 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

پس از آن این مدل با بهینه ساز آدام و تابع اتلاف mse در ۵۰ دسته ۵۰ تایی آموزش داده شده است. همچین قسمت اعتبار سنجی به میزان ۲۰ درصد این داده‌ها در نظر گرفته شده است. r2_score و اتلاف آن پس از آموزش مطابق زیر است.

```
29/29 [==============================] - 0s 999us/step - loss: 1.0355e-04
29/29 [==============================] - 0s 855us/step
(0.00010354510595789179, 0.43683250309907595)
```

بهینه ساز Adam یک الگوریتم بهینه سازی انباشت گرادیان ونسخهٔ تعمیم یافتهٔالگوریتم SGDاست که در شبکه های عصبی عمیق به خوبی کارمی کند. این الگوریتم درمقایسه با برخی بهینه سازهای دیگرسرعت هم‌گرایی بالا ودقت بیش تری در پیش بینی‌داده ها دارد.

تابع اتلاف میانگین مربعات یکی از معروف ترین توابع اتلاف در مسائل رگرسیون است. این تابع برای اندازه گیری خطا و اختلاف پیش بینی ها و مقادیر واقعی استفاده می شود. در این تابع، فاصله بین پیش بینی شده توسط مدل و مقدار واقعی داده با استفاده از مربع این فاصله محاسبه می شود. سپس میانگین این مربعات برای تمامی داده ها محاسبه می شود. برای n داده، تابع اتلاف میانگین مربعات به صورت زیر تعریف می شود.

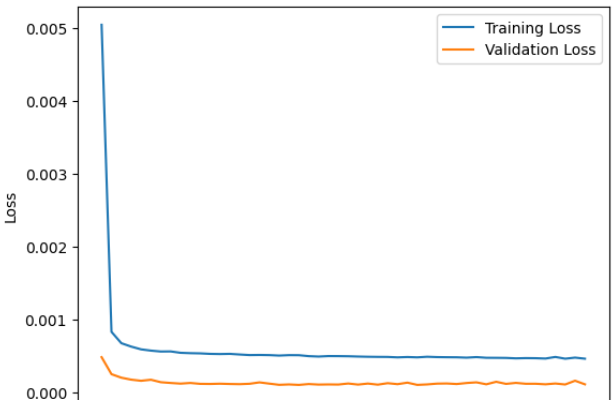
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

معیار R2 Score یکی از معیارهای مهم است که در تحلیل رگرسیون استفاده می شود. این معیار، نشان می دهد که چه میزان از تغییرات متغیر وابسته توسط متغیرهای مستقل قابل پیش بینی است. در واقع، R2 Score بین ۰تا ۱ قرار می گیرد و نشان می دهد که چه میزان از تغییرات متغیر وابسته توسط متغیرهای مستقل مورد پوشش قرار گرفته است. عدد صفر به معنای عدم توانایی مدل در پیش بینی و عدد یک به معنای پیش بینی دقیق تمامی مقادیر است.

هر چه مقدار این معیار بیشتر باشد بهتر است و به این معنی است که مدل با ویژگی‌های در نظر گرفته شده، مناسب است و اگر مقدار آن کم باشد باید ویژگی های در نظر گرفته شده دوباره برسی شود.

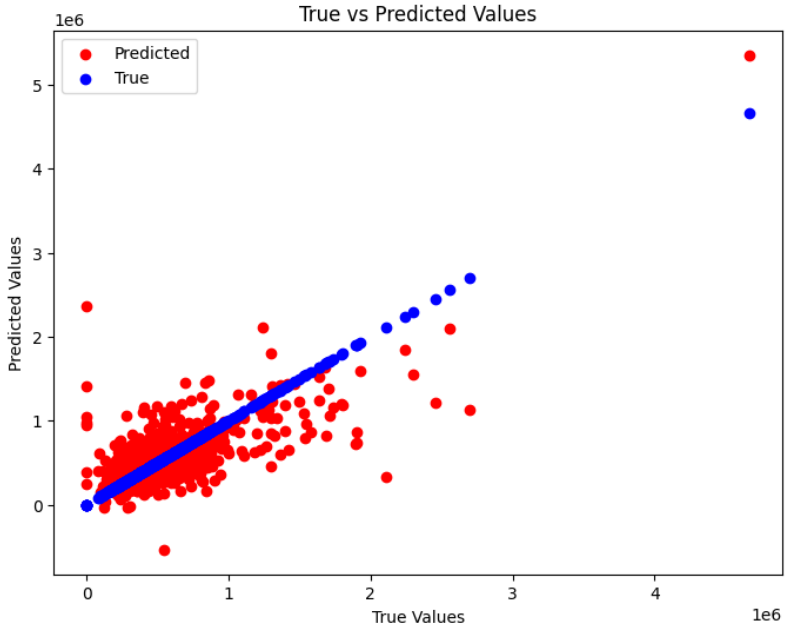
نتایج حاصل از آموزش R2 Score برابر ۰.۴۴ را گزارش می‌دهد که به نشان می‌دهد وابستگی قیمت به این ویژگی ها آنقدر نیست که بتوان پیشبینی دقیقی داشت ولی پیش بینی انجام شده تا حدی نزدیک خواهد بود.

نمودار نمودار اتلاف حاصل از آموزش است. اتلاف در ابتدای آموزش زیاد است و با آموزش مدل کاهش یافته و ثابت می‌شود. داده‌های اعتبار سنجی نیز وضعیت مشابهی دارند که نشان دهنده روند صحیح آموزش است.



نمودار ۴) نمودار اتلاف

حال داده‌های تست به مدل داده شده است. شکل ۶ نشان دهنده پیش بینی و مقادیر صحیح این داده‌ها است.



شکل ۶) داده‌های تست و پیش بینی آن‌ها

۷. تغییر بهینه ساز و تابع اتلاف

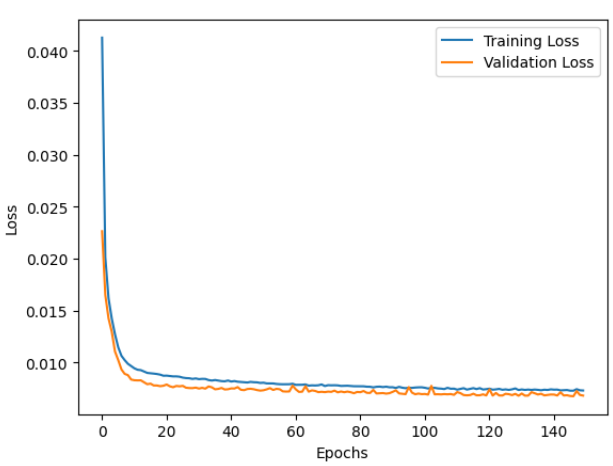
Mean Absolute Error Loss نیز یکی دیگر از توابع اتلاف معمول در مسائل رگرسیون است. در این تابع، فاصله بین پیش بینی شده توسط مدل و مقدار واقعی داده با استفاده از مقدار مطلق این فاصله محاسبه می شود. سپس میانگین این فواصل مطلق برای تمامی داده‌ها محاسبه می شود. برای n داده، تابع این اتلاف به صورت زیر تعریف می شود.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

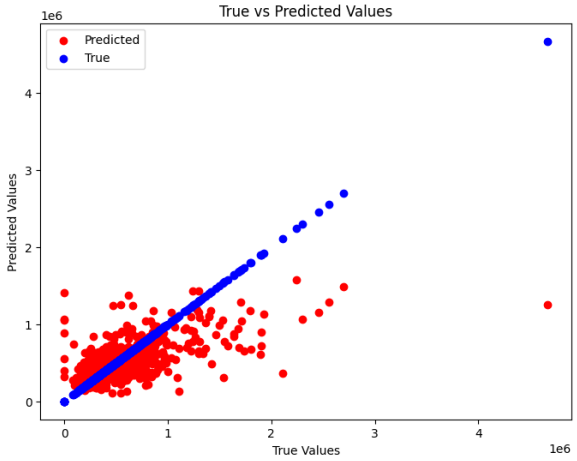
یکی از بهینه سازهایی که با داشتن mini-batch می توانیم از آن استفاده کنیم، گرادیان کاهشی تصادفی (SGD) است. روش محاسبه گرادیان در این الگوریتم دقیقا مشابه گرادیان نزولی است. اما بر خلاف روش گرادیان کاهشی که کل دیتاست را یک باره به شبکه می دهد، الگوریتم گرادیان کاهشی تصادفی دسته های کوچک داده را برای هر مرحله از بهینه سازی در نظر می گیرد. مدل ۲ مشابه با مدل ۱ تعریف شده است با این تفاوت که بهینه ساز و تابع اتلاف به موارد بالا تغییر یافته است. آموزش انجام شده و نتایج مطابق زیر

است. در نمودار ۵ نمودار اتلاف و در شکل ۷ نتایج پیش بینی و داده تست نشان داده شده است. همانطور که انتظار می‌فت مقدار اتلاف با وجود ایپاک بیشتر افزایش یافته و R2 Score به ۰.۳۵ کاهش یافته است.

```
29/29 [==============================] - 0s 2ms/step - loss: 0.0067
29/29 [==============================] - 0s 1ms/step
loss 0.006731478497385979 r2score 0.35335789079527735
```



نمودار ۵) نمودار اتلاف بر حسب تعداد ایپاک



شکل ۷) نمایش قیمت واقعی و قیمت پیش بینی شده

مشابه قسمت قبل تابع اتلاف نشان دهنده آموزش صحیح مدل است. با توجه به تغییر بهینه ساز مشاهده می‌شود که نوسانات اتلاف افزایش یافته است. نتایج حاصل نسبت به قسمت قبل به پراکنده تر است و عملکرد ضعیف تری دارد.

۸. ارزیابی

حال ۵ داده به صورت تصادفی انتخاب و نتایج اصلی آن‌ها با نتایج حاصل از پیش بینی نمایش داده شده است. به منظور ارزیابی ساده تر اختلاف این دو عدد بر مقدار واقعی تقسیم شده و مشاهده می‌شود که بین ۱۲ تا ۶۶ درصد خطا با مقدار واقعی وجود دارد که نشان دهنده آن است که مدل به خوبی عمل نمی‌کند و مناسب نیست.

```
[['% of errore']
 ['-0.16194145683453237']
 ['-0.12129533226976097']
 ['-0.2678764607679466']
 ['0.37369647749510765']
 ['-0.6660032762096776']]

[['prediction' 'label']
 ['695000.0' '807549.3']
 ['308830.769231' '346290.5']
 ['599000.0' '759458.0']
 ['2555000.0' '1600205.5']
 ['247999.99999999997' '413168.8']]
```

برای بهبود مدل می‌توان ویژگی‌های بهتری انتخاب کرد یا تعداد لایه‌های پنهان را افزایش داد با این وجود به علت همبستگی کم قیمت با دیگر ویژگی‌ها به احتمال قوی مدل خوبی قابل استخراج نخواهد بود.

بخش پنجم

مجموعه دادهٔ Iris

مقدمه

در این قسمت مجموعه دادهٔ Iris برسی شده است و با سه روش برسی شده و با شاخص‌های مختلف برسی و نتایج آن تحلیل شده است.

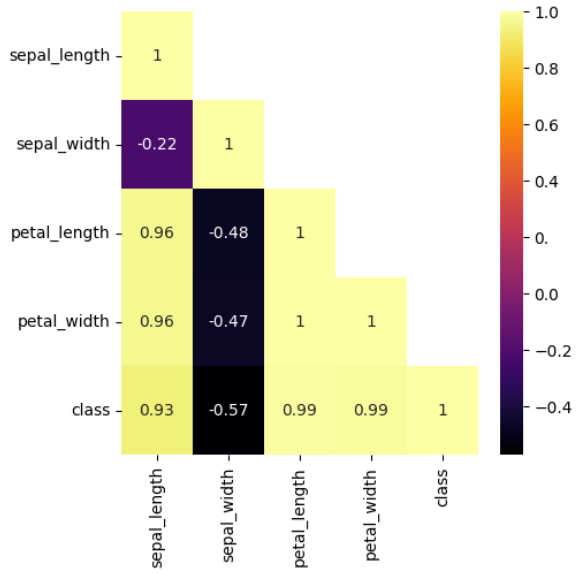
۱. داده‌های مورد بررسی

این مجموعه داده شامل ۱۵۰ داده مربوط به گل زنبق بوده که طول کاسبرگ، عرض کاسبرگ، طول و عرض گلبرگ به عنوان ویژگی و ۳ کلاس ستوسا، ویرجینیکا و ورسی‌کالر وجود دارد.

این مجموعه داده از مجموعه داده‌های سایکیت لرن به عنوان یک شی ساخته شده و داده های آن در بردار X و کلاس‌های آن در بردار y ریخته شده است. سپس این دو برای نمایش به همه چسبانده شده و با نمایش داده شده است. ۵ داده رندم از این مجموعه داده در زیر نامایش داده شده است.

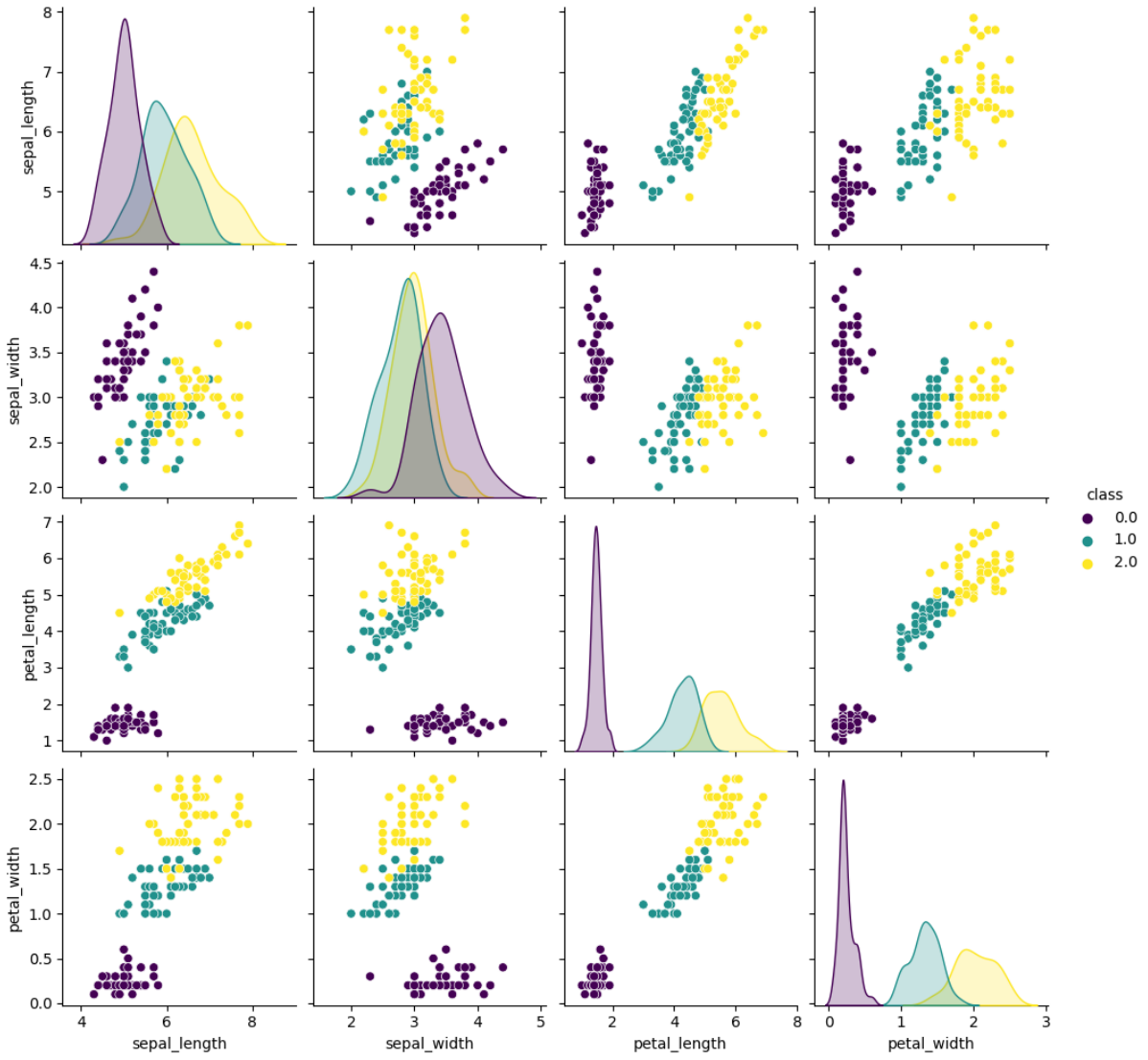
	sepal_length	sepal_width	petal_length	petal_width	class
0	4.4	2.9	1.4	0.2	1.0
1	5.1	3.3	1.7	0.5	1.0
2	7.7	2.6	6.9	2.3	3.0
3	5.5	2.3	4.0	1.3	2.0
4	6.7	3.1	4.7	1.5	2.0

سپس به منظور آموزش، داده‌ها به دو قسمت تست و آموزش به نسبت ۲ به ۸ تقسیم شده است. با توجه به این که کل دیتاست ۱۵۰ داده است، ۱۲۰ داده آموزشی و ۳۰ داده تست داریم. ماتریس همبستگی این داده‌ها در شکل ۸ نمایش داده شده است که بیانگر آن است که عرض کاسبرگ همبستگی زیادی با کلاس ندارد و می‌توان این ویژگی را در آموزش حذف کرد.



شکل ۸) ماتریس همبستگی

برای مشاهده بهتر روابط کلاس‌ها و ویژگی‌ها، داده‌ها بر حسب هر دو ویژگی به صورت جفت جفت پلات شده‌اند. نتیجه حاصل در شکل ۹ آمده است. مشابه تابع همبستگی مشاهده می‌شوند که عرض کاسبرگ ویژگی نیست که در طبقه بندی موثر باشد. همچنین مشاهده می‌شود که کلاس صفر به راحتی قابل جدا سازی از دو کلاس دیگر است اما دو کلاس ۲ و ۱ در همه موارد مقداری همپوشانی دارند و جدا کردن آن ها حتی با استفاده از یک تابع غیر خطی امکان پذیر نیست.



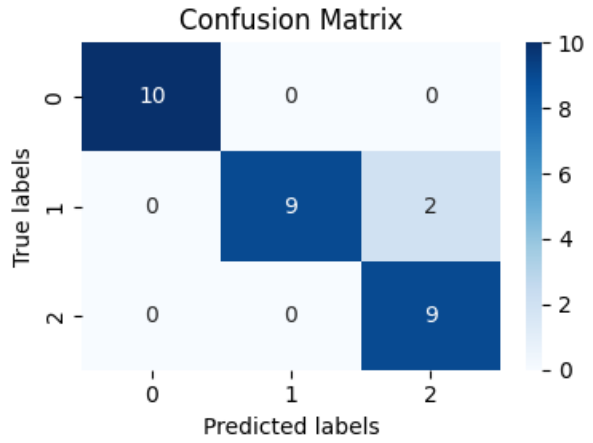
شکل ۹) روابط دو به دو ویژگی‌ها با هم

۲. آموزش شبکه‌ها

به منظور طبقه بندی از سه مدل رگرسیون لجیستیک، پرسپترون چند لایه و شبکه‌های عصبی پایه شعاعی استفاده شده است. در ادامه نتایج حاصل از این موارد آمده است. این مدل ها ابتدا با استفاده از کتابخانه اماده پیاده شده و بعدا تلاش بر پیاده‌سازی آن‌ها از پایه شده است.

در استفاده از کتابخانه آماده ۳ مدل به صورت تقریبا مشابه تعریف می‌شوند بدین صورت که یک شی از این کلاس‌ها که توسط کتابخانه سایکیت لرن در دسترس هستند ساخته می‌شود. قابل ذکر است که در مدل MLP تعداد لایه‌ها نیز در تعریف اولیه تعیین شده است. سپس داده‌های آموزشی به هر کدام از این مدل‌ها داده می‌شود. پس از آن به منظور ارزیابی مدل، داده‌تست به آن داده شده و با مقایسه خروجی مدل و برچسب مربوط به آن، صحت مدل بدست آمده است. در ادامه ماتریس در هم ریختگی مربوط به آن رسم شده است و در انتها به کمک classification_report مقادیر شاخص‌های مختلف نمایش داده شده است که در ادامه تحلیل خواهند شد.

با توجه به سادگی دیتاست، نتایج هر ۳ مدل یکسان شده است ولی در حالتی که تعداد لایه‌ای مدل MLP کاهش یافته است، مدل همگرا نشده و صحت کمتری دارد. نتایج حاصل به شرح زیر است. ماتریس درهم‌ریختگی نیز در شکل ۱۰ رسم شده است.



شکل ۱۰) ماتریس درهم‌ریختگی

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.82      0.90        11
           2       0.82      1.00      0.90         9

    accuracy                           0.93        30
   macro avg       0.94      0.94      0.93        30
weighted avg       0.95      0.93      0.93        30
```

در کلاس ۰ (ستوسا) Precision طبقه‌بندی در پیش‌بینی ۱۰۰٪ است، به این معنی که هر نمونه‌ای که به عنوان کلاس ۰ پیش‌بینی شده، درست بوده است. Recall نیز برای این کلاس۱۰۰٪ است که نشان می‌دهد طبقه‌بندی همه موارد کلاس ۰ را به درستی شناسایی کرده است. F1-score، میانگین Precision و Recall است و در این مورد کامل (۱.۰۰) است که با توجه به Precision و Recall کامل، انتظار می‌رود.

در کلاس ۱ (ورسیکالر) Precision ٪۱۰۰ برای کلاس ۱ است که نشان می‌دهد وقتی طبقه‌بندی نمونه‌ای را به عنوان کلاس ۱ پیش‌بینی می‌کند، همیشه صحیح است. Recall ٪۸۲ برای کلاس ۱ است، به این معنا که حدود ۱۸٪ از موارد واقعی کلاس ۱ را از دست داده است. F1-score ۰.۹ است، که به دلیل Recall کامل نیست.

کلاس ۲ (ویرجینیکا) Precision ٪۸۲ برای کلاس ۲ است، که نشان می‌دهد مواردی وجود دارد که طبقه‌بندی به اشتباه کلاس ۲ را پیش‌بینی می‌کند. Recall ٪۱۰۰ برای کلاس ۲ است، به این معنی که مدل همه موارد واقعی کلاس ۲ را به درستی شناسایی کرده است. F1-score نیز ۰.۹ است، که تحت تأثیر Precision پایین است.

دقت کلی ٪۹۳ است، که نشان می‌دهد طبقه‌بندی تا چه اندازه در تمام کلاس‌ها صحیح است. میانگین کلی برای Precision، Recall، و F1-score همگی ۰.۹۴ هستند، که عملکرد را بین کلاس‌ها بدون توجه به پشتیبانی (تعداد نمونه‌ها برای هر کلاس) متوسط می‌کند. میانگین وزنی، تعداد نمونه‌ها برای هر کلاس را برای هر کلاس در نظر می‌گیرد. برای Precision، Recall، و F1-score به ترتیب ۰.۹۵، ۰.۹۳ و ۰.۹۳ است، که نشان‌دهنده دقت زیاد مدل روی مجموعه داده متعادل مجموعه داده‌های Iris است.

به طور کلی مدل عملکرد فوق‌العاده‌ای در کلاس ۰ و عملکرد بسیار خوبی در کلاس‌های ۱ و ۲ دارد. با این حال اندکی جای بهبود در دقت برای کلاس ۲ (ویرجینیکا) و بازیابی برای کلاس ۱ (ورسیکالر) وجود دارد.

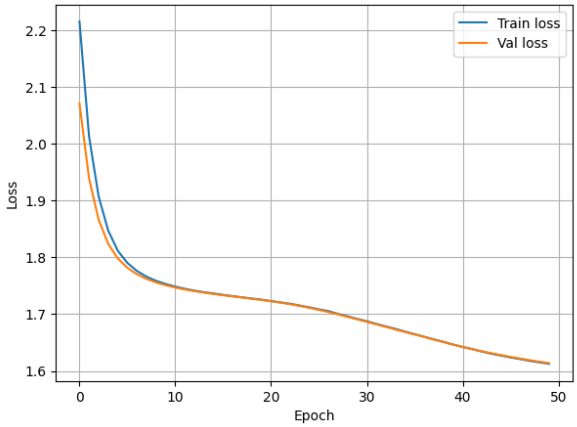
۳. پیاده سازی از پایه

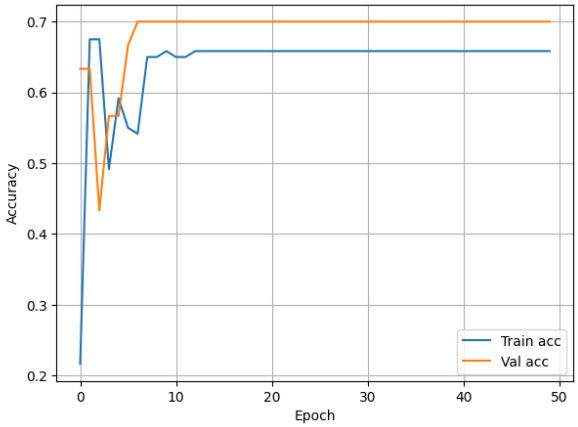
در این قسمت تلاش بر این شده است که شبکه‌های عصبی استفاده شده در بخش قبل پیاده سازی شوند.

برای پیاده سازی رگرسیون لجیستیک یک کلاس با این نام ساخته شده و ۴ تابعinit ، سیگموید، فیت و پیش بینی برای آن تعریف شده است که تا حد زیادی مشابه با نورون تعریف شده در قسمت اول تمرین است. پس از پیاده سازی کلاس، مجموعه داده به یک مدل ساخته شده از این کلاس داده می‌شود و بعد از فیت شدن بر روی داده‌های آموزشی پیش بینی داده تست و ارزیابی مدل صورت می‌گیرد. نتایج بدست آمده عبارتند از:

```
Logistic Regression Metrics:
Accuracy: 0.3667
Precision: 0.1344
Recall: 0.3667
F1 Score: 0.1967
```

در پیاده سازی MLP نیز کلاسی با همین نام نوشته شده و در ۱۲ لایه با یک نورون در هر لایه نتایج زیر حاصل شده است که نشان می‌دهد آموزش تا حد قابل قبولی صورت گرفته است.





مراجع

[1]. https://github.com/MJAHMADEE/MachineLearning2023