گزارش تمرین سوم داده‌کاوی

علیرضا آزادبخت ۹۹۴۲۲۰۱۹

سوال ۱:

بعضی از دیتاست هایی که ما با آن ها سر و کار داریم به صورت خطی از هم قابل تفکیک نیستند اما طبق تعوری cover در موضوع جداسازی الگو ها اینگونه بیان میکند که مسئله طبقه بندی الگو ها با احتمال زیادی به صورت خطی قابل حل هستند اگر به کمک یک تابع غیر خطی به فضای با ابعاد زیاد منتقل شود، یکی از دلایل موفقیت شبکه های عضبی هم بر پایه همین تعوری میباشد و svm نیز از این موضوع مستقل نیست این مدل مسئله را به صورت خطی حل میکند و اگر داده ورودی آن به صورت خطی قابل جدا سازی نباشد میتوان به کمک اعمال یک تابع غیر خطی این مسئله را حل کرد، برای این کار میتوان به صورت دستی دیتا ست را تولید کرد که از ارتباط فیچر های مختلف ایجاد میشود مصلا حاصل ضرب دو به دو فیچر ها یا توان دو های آن ها این تبدیل های دستی هزینه محاسباتی زیادی دارند و ممکن از دیتا ست بعد از این تبدیل دیگر نتوان با زیر ساخت قبلی از نظر رم و سخت افزاری مهار شود لذا در روش svm میتوان از کرنل های مختلف استفاده کرد که بدون اضافه کردن دستی این فیچر های جدید یک تبدیل غیر خطی روی آن ها اعمال کند این روش مشکل دیتا ست های غیر خطی جدا پذیر را حل میکند و بهینه تر از حالت قبلی است، از کرنل های معروف میتوان به rbf و poly و linear یاد کرد، کرنل خطی همان svm ساده است که دیتا ست را کاملا خطی جدا میکند،

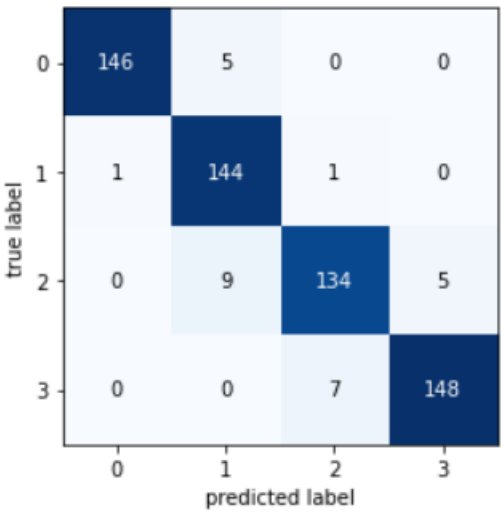
کرنل rbf با فرمول $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$ که صورت آن فاصله اقلیسی و مخرج آن یک هایپر است

و کرنل poly با فرمول $K(x, y) = (x^\mathsf{T} y + c)^d$ یک رابطه با درجه های مختلف بین فیچر ها ایجاد میکند به طور کلی نمیتوان حکم کلی در رابطه با استفاده از کرنل ها داد چون در خیلی از موارد ما شکل دیتا را نمیدانیم وبهتر است به کمک کراس ولیدیشن کرل مناسب را پیدا کنیم و حتی همین کرنل ها هم باید تیون شوند تا هایپر پارامتر مناسب آن ها پیدا شود و این کار هم به کمک کراس ولیدیشن انجام میشود.
اما میتوان به صورت کلی گفت اگر کلاس ها در دل یک دیگر باشند مانند دایره ای درون دایره دیگر از rbfاستفاده میکنیم و اگر کلاس ها به صورت کلی خطی جدا پذیر باشند اما در بعضی نواحی به صورت غیر خطی وارد یک دیگر شده اند مانند دو حلال که در یک دیگر رفته اند بهتر است از poly استفاده کنیم و از کرنل خطی هم در دیتا ست هاییکه کاملا خطی جدا میشوند استفاده میشود.

سوال ۲:

بر روی دیتاست موبایل ها یک svm اجرا کردیم و نتایج را بر روی داده های تست که که ۳۰ درصد داده هستند گزارش میکنیم:

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

           0       0.99      0.97      0.98       151
           1       0.91      0.99      0.95       146
           2       0.94      0.91      0.92       148
           3       0.97      0.95      0.96       155

    accuracy                           0.95       600
   macro avg       0.95      0.95      0.95       600
weighted avg       0.95      0.95      0.95       600
```



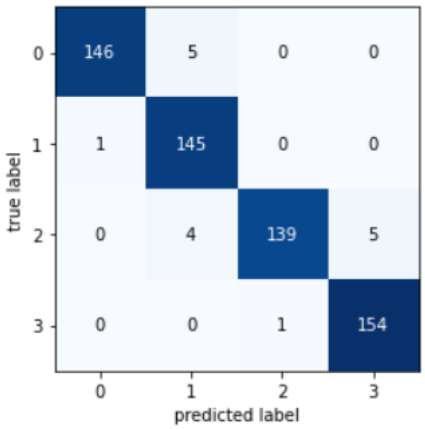
سوال ۳:

نتایج مرحله قبل را باکانفیگ های مختلف اجرا گرفتیم:

Svm با کرنل linear

Classification Report:

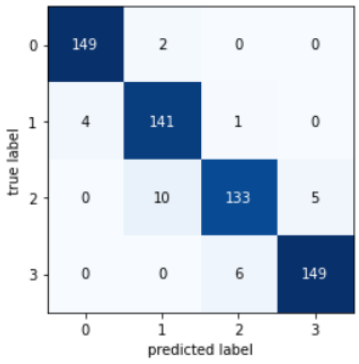
	precision	recall	f1-score	support
0	0.99	0.97	0.98	151
1	0.94	0.99	0.97	146
2	0.99	0.94	0.97	148
3	0.97	0.99	0.98	155
accuracy			0.97	600
macro avg	0.97	0.97	0.97	600
weighted avg	0.97	0.97	0.97	600



Svm با کرنل poly

Classification Report:

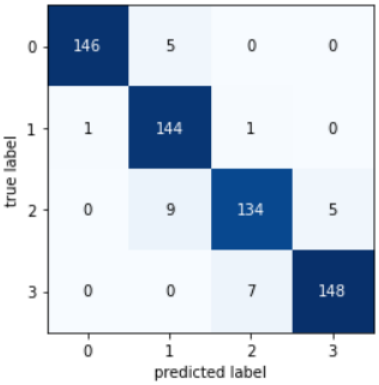
	precision	recall	f1-score	support
0	0.97	0.99	0.98	151
1	0.92	0.97	0.94	146
2	0.95	0.90	0.92	148
3	0.97	0.96	0.96	155
accuracy			0.95	600
macro avg	0.95	0.95	0.95	600
weighted avg	0.95	0.95	0.95	600



Svmبدون shrinking

Classification Report:

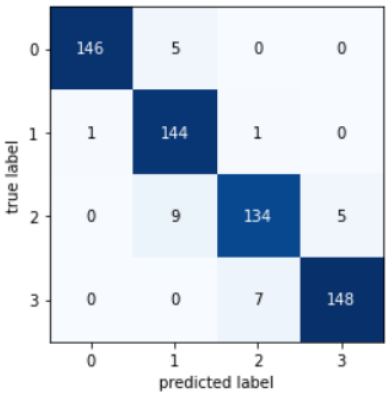
	precision	recall	f1-score	support
0	0.99	0.97	0.98	151
1	0.91	0.99	0.95	146
2	0.94	0.91	0.92	148
3	0.97	0.95	0.96	155
accuracy			0.95	600
macro avg	0.95	0.95	0.95	600
weighted avg	0.95	0.95	0.95	600



Svm با روش جدا سازی یکی در برابر همه

Classification Report:

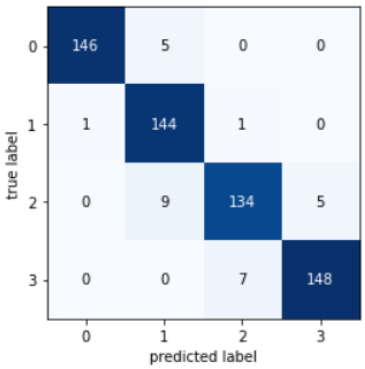
	precision	recall	f1-score	support
0	0.99	0.97	0.98	151
1	0.91	0.99	0.95	146
2	0.94	0.91	0.92	148
3	0.97	0.95	0.96	155
accuracy			0.95	600
macro avg	0.95	0.95	0.95	600
weighted avg	0.95	0.95	0.95	600



Svmبا روش جدا سازی یکی در برابر یکی(دو به دو)

Classification Report:

	precision	recall	f1-score	support
0	0.99	0.97	0.98	151
1	0.91	0.99	0.95	146
2	0.94	0.91	0.92	148
3	0.97	0.95	0.96	155
accuracy			0.95	600
macro avg	0.95	0.95	0.95	600
weighted avg	0.95	0.95	0.95	600



نتیجه طبق یافته ها دیتا ست به صورت خطی جدا پذیر است و بیشترین دقت با کرنل خطی بدست آمد و لازم به ذکر است که نتیجه بدست آمده در سوال قبل با کرنل rbf بود.

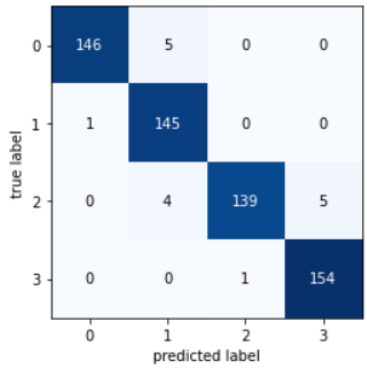
سوال ۴:

در این بخش اثر سافت و هارد مارجین را بررسی میکنیم:

Svm با هارد مارجین (c=1)

Classification Report:

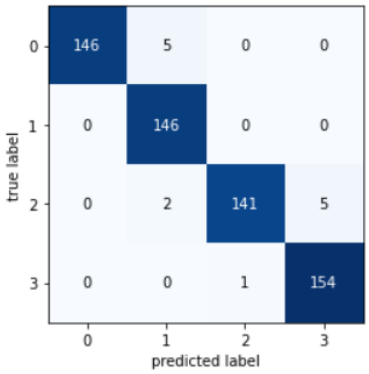
	precision	recall	f1-score	support
0	0.99	0.97	0.98	151
1	0.94	0.99	0.97	146
2	0.99	0.94	0.97	148
3	0.97	0.99	0.98	155
accuracy			0.97	600
macro avg	0.97	0.97	0.97	600
weighted avg	0.97	0.97	0.97	600



Svm با سافت مارجین (c=0.05)

Classification Report:

	precision	recall	f1-score	support
0	1.00	0.97	0.98	151
1	0.95	1.00	0.98	146
2	0.99	0.95	0.97	148
3	0.97	0.99	0.98	155
accuracy			0.98	600
macro avg	0.98	0.98	0.98	600
weighted avg	0.98	0.98	0.98	600



نتیجه: در قسمت قبل دیدیم که دیتا ست به صورت خطی جدا پذیر است و در این بخش مشاهده کردیم که با سافت مرجین میتوانیم حتی دقت خیلی بالاتری نیز کسب کنیم و نتیجه بهبود یک درصدی داشته باشد.

سوال ۵:

بر روی فیچر ظرفیت باتری به ۳ روش بین بندی کردیم یکبار به ۴ بین مختلف یک بار به ۶ بین مختلف و یک بار هم ب صورت نامساوی به دو بین کوچکتر از ۱۰۰۰ و بزرگتر از ۱۰۰۰ تبدیل کردیم.

این دیتا ست دارای فیچر های کتگوریکال نبود و استفاده از روش وان هات بر روی کتگوریکال های دو کلاسه صحیح نیست و فقط ریداندنسی به دیتا اضافه میکند اما در کیس هایی که کتگوری های یک فیچر ترتیب معنایی ندارند مانند گروه خونی افراد برای اینکه این ترتیب نداشتن را به مدل بفهمانیم از روش وان هات استفاده میکنیم.

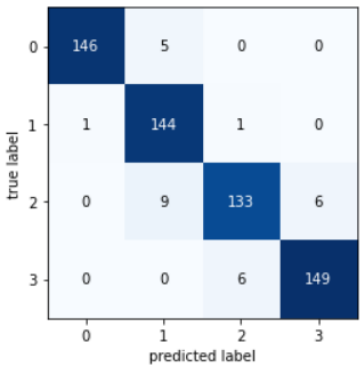
طب توزیع هایی که از فیچر ها در نوتبوک و گزارش مربوطه تمرین قبلی دیدم فیچر های " ,"fc", "px_height","sc_w","clock_speed"" دارای چولگی هستند هستند و نرمال نیستند در این حالت بهتر است از تبدیل log استفاده کنیم که تویع داده ها نرمال شود، فیچری نداشتیم که نیاز به تبدیل نمایی داشته باشد و تنها همین ۴ فیچر را log میزنیم. به طور کلی اگر توزیع های داده ها مناسب مدل هایی که میخواهیم استفاده کنیم نباشد و یا داده های پرت زیاد باشند یا مقادیر اشتباه د آن ها وجود داشته باشد از تبدیلات

مختلف میتوانیم استفاده کنیم مثلا معروف ترین آن ها گوسین هست که توزیع هارا نرمال میکند. و یا تبدیل نمایی که توزیع هارا به سمت راست حرکت میدهد.
فیچر جدیدی به اسم مساحت تولید میکنیم که از حاصل ضرب sc_w و sc_h بدست میآید.

سوال ۶:

Svm ای بر روی تمام فیچر های سوال قبل اجرا کردیم و نتیجه زیر را بدست آوردیم:

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

           0       0.99      0.97      0.98       151
           1       0.91      0.99      0.95       146
           2       0.95      0.90      0.92       148
           3       0.96      0.96      0.96       155

    accuracy                           0.95       600
   macro avg       0.95      0.95      0.95       600
weighted avg       0.95      0.95      0.95       600
```

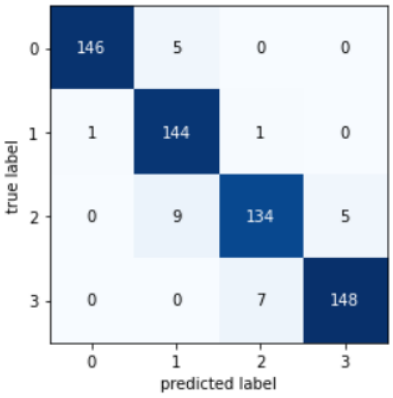


نتیجه بهتری کسب نکردیم از حالت خام

Svm ای بر روی فیچر های لاگ به تنهایی از سوال قبل اجرا کردیم و نتیجه زیر را بدست آوردیم:

Classification Report:

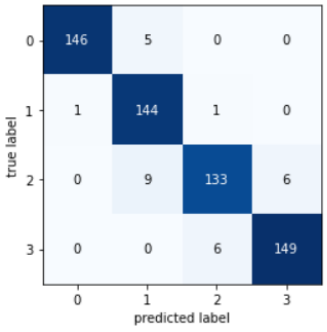
	precision	recall	f1-score	support
0	0.99	0.97	0.98	151
1	0.91	0.99	0.95	146
2	0.94	0.91	0.92	148
3	0.97	0.95	0.96	155
accuracy			0.95	600
macro avg	0.95	0.95	0.95	600
weighted avg	0.95	0.95	0.95	600



Svm ای بر روی فیچر های مساحت سوال قبل اجرا کردیم و نتیجه زیر را بدست آوردیم:

Classification Report:

	precision	recall	f1-score	support
0	0.99	0.97	0.98	151
1	0.91	0.99	0.95	146
2	0.95	0.90	0.92	148
3	0.96	0.96	0.96	155
accuracy			0.95	600
macro avg	0.95	0.95	0.95	600
weighted avg	0.95	0.95	0.95	600



نتیجه بهتری کسب نکردیم و لازم به ذکر است که فیچر های بین را در حالت یک در نظر گرفتیم و به طور کلی هیچ یک از فیچر های جدید در دقت مدل کمکی نکردند و فقط دیسیژن باندری های مختلفی پیدا کردیم.

سوال ۷:

سه الگوریتم معروف برای تولید درخت های تصمیم داریم:

الگوریتم ID3:

این الگوریتم یکی از ساده ترین الگوریتم های درخت تصمیم است. در این الگوریتم درخت تصمیم از بالا به پایین ساخته می‌شود. این الگوریتم با این سوال شروع می‌شود: کدام ویژگی باید در ریشه درخت مورد آزمایش، قرار بگیرد؟ برای یافتن جواب از معیار بهره اطلاعات استفاده می‌شود. با انتخاب این ویژگی، برای هر یک از مقادیر ممکن آن یک شاخه ایجاد شده و نمونه های آموزشی بر اساس ویژگی هر شاخه مرتب می‌شوند. سپس عملیات فوق برای نمونه های قرار گرفته در هر شاخه تکرار می شوند تا بهترین ویژگی برای گره بعدی انتخاب شود.

الگوریتمC4.5 :

این الگوریتم یکی از تعمیم های الگوریتم ID3 است که از معیار نسبت بهره (Gain ratio)استفاده می کند. الگوریتم هنگامی متوقف می‌شود که تعداد نمونه ها کمتر از مقدار مشخص شده‌ای باشد. این الگوریتم از تکنیک پس هرس استفاده می‌کند و همانند الگوریتم قبلی داده‌های عددی را نیز می‌پذیرد.

از نقاطِ ضعف الگوریتم ID3 که در C4.5 رفع شده است می‌توان به موارد زیر اشاره کرد:

الگوریتم C4.5 می‌تواند مقادیر گسسته یا پیوسته را در ویژگی‌ها درک کندو

الگوریتم C4.5قادر است با وجود مقادیر گمشده نیز درخت تصمیم خود را بسازد، در حالی که الگوریتمی مانند ID3 و بسیاری دیگر از الگوریتم‌های طبقه‌بندی نمی‌توانند با وجود مقادیر گمشده، مدلِ خود را بسازند.

سومین موردی که باعث بهینه شدن الگوریتم C4.5 نسبت به ID3 می‌شود، عملیاتِ هرس کردن جهت جلوگیری از بیش برازش می‌باشد. الگوریتم‌هایی مانند ID3 به خاطر اینکه سعی دارند تا حد امکان شاخه و برگ داشته باشند (تا به نتیجه مورد نظر برسند) با احتمال بالاتری دارای پیچیدگی در ساخت مدل و این پیچیدگی در بسیاری از موارد الگوریتم را دچار بیش برازش و خطای بالا می‌کند.اما باعملیات هرس کردن درخت که در الگوریتم 5 انجام می‌شود، می‌توان مدل را به یک نقطه بهینه رساند که زیاد پیچیده نباشد (و البته زیاد هم ساده نباشد) و بیش برازش یا کم برازش رخ ندهد.

الگوریتم C4.5 این قابلیت را دارد که وزن‌های مختلف و غیر یکسانی را به برخی از ویژگی‌ها بدهد.

الگوریتمCHAID :

محققان آمار کاربردی، الگوریتم‌هایی را جهت تولید و ساخت درخت تصمیم توسعه دادند .الگوریتم CHAID در ابتدا برای متغیرهای اسمی طراحی شده بود. این الگوریتم با توجه به نوع برچسب کلاس از آزمون‌های مختلف آماری استفاده می‌کند. این الگوریتم هرگاه به حداکثر عمق تعریف شده‌ای برسد و یا تعداد نمونه‌ها در

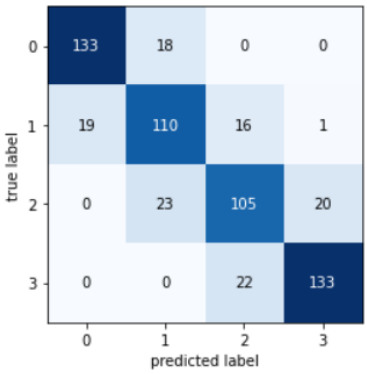
گره جاری از مقدار تعریف شده‌ای کمتر باشد، متوقف می‌شود. الگوریتم CHAID هیچگونه روش هرسی را اجرا نمی‌کند.

سوال ۸:

یک درخت تصمیم بر روی مجموعه داده موبایل با همان نسبت ۳۰ درصد جرا کردیم و بهنتایج زیر رسیدیم:

Classification Report:

	precision	recall	f1-score	support
0	0.88	0.88	0.88	151
1	0.73	0.75	0.74	146
2	0.73	0.71	0.72	148
3	0.86	0.86	0.86	155
accuracy			0.80	600
macro avg	0.80	0.80	0.80	600
weighted avg	0.80	0.80	0.80	600



سوال ۹:

پارامتر های عمق درخت و مینیمم تعداد برگ را به صورت جست و جوی جدولی بررسی کردیم و به نتایج زیر رسیدیم:

```
max_depth: 8    min_sample_leaf: 1    score: 0.8133333333333334
max_depth: 8    min_sample_leaf: 5    score: 0.8283333333333334
max_depth: 8    min_sample_leaf: 20   score: 0.81
max_depth: 8    min_sample_leaf: 40   score: 0.76
max_depth: 20   min_sample_leaf: 1    score: 0.8183333333333334
max_depth: 20   min_sample_leaf: 5    score: 0.8166666666666667
max_depth: 20   min_sample_leaf: 20   score: 0.81
max_depth: 20   min_sample_leaf: 40   score: 0.76
max_depth: 40   min_sample_leaf: 1    score: 0.8066666666666666
max_depth: 40   min_sample_leaf: 5    score: 0.8233333333333334
max_depth: 40   min_sample_leaf: 20   score: 0.81
max_depth: 40   min_sample_leaf: 40   score: 0.76
max_depth: 100  min_sample_leaf: 1    score: 0.8133333333333334
max_depth: 100  min_sample_leaf: 5    score: 0.8266666666666667
max_depth: 100  min_sample_leaf: 20   score: 0.8016666666666666
```

```
max_depth: 100  min_sample_leaf: 40    score: 0.76
```

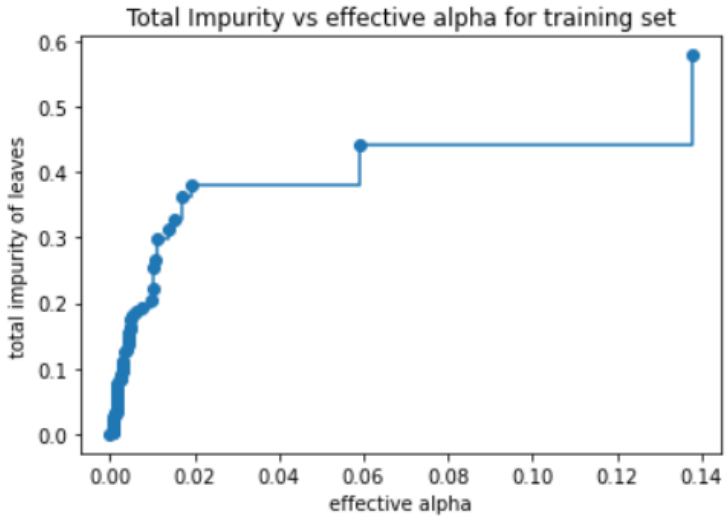
وقتی عمل درخت زیاد میشود دقت هم به نسبتی افزایش میباد به طوری که بیشترین دقت را در عمق ۱۰۰ بدست آورده ایم اما وقتی که تعداد برگ ها زیاد میشود لزوما اتفاق خوبی برای دقت نمیافتد مقدار بهینه برای آن برابر با ۵ است.

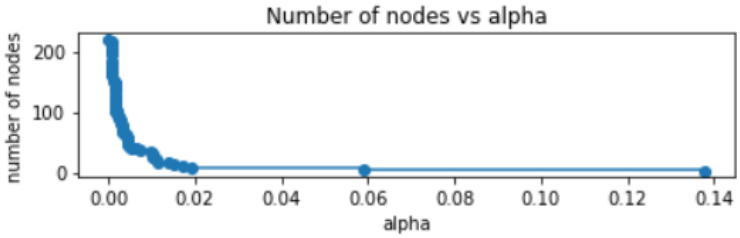
سوال ۱۰:

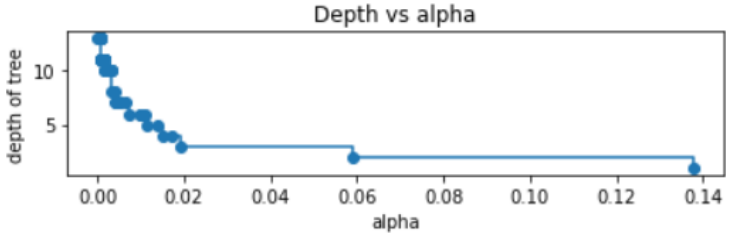
به صورت پیش فرض درخت های تصمیم با جایی پیش میروند که در هر نود برگ آن ها یک داده باقی بماند یا در بعضی روش ها به تعداد مینیممی که کاربر مشخص میکند این امر در درخت های تصمیم و خانواده های آن ها رایج است و به سرعت بر روی مجموعه داده آموزش اوور فیت میشوند و دقت تست آن ها به شدت افت میکند و این مدل ها مشکل جنرالایزیشن دارند و نمیتوانند در کاربرد های جدی مفید باشند البته میتوان به طور دقیق تر بیان کرد بر داده هایی که ماهیت همگن دارند مفید نیستند این داده ها دارای یک الگوی ساختاری هستند مانند دیتای صوت یا تصویر در این داده ها مدلهای دیگر یا بر پایه شبکه های عصبی بهتر عمل میکنند چون مدل های درختی به جای پیدا کردن الگو داده ها سعی میکنند تفاوت بین داده هارا پیدا کنند و لزما نمیتوانند جنرالایز باشند اما از طرف دیگر بر روی داده های نا همگن که از انواع مختلفی جم آوری میشود مانند دیتا های مالی یا اب و هوا یا داده های مبتنی بر اینترنت اشیا بهتر عمل میکنند
اما به طور کلی برای داده های همگن و ناهمگن خوب است که درخت پایانی را هرس کنیم و مقداری خطا بر روی داده اموزش را وارد مدل کنیم اما در عوض دقت آن بر داده تست را افزایش دهیم در این روش میاییم شاخه هایی که اینفورمیشن کمی به ما میدهند را مرج میکنیم و فرزندان آن هارا در همان نود ادغامی قرار میدهیم و درخت های کوچتر اما جنرال تری میسازیم.

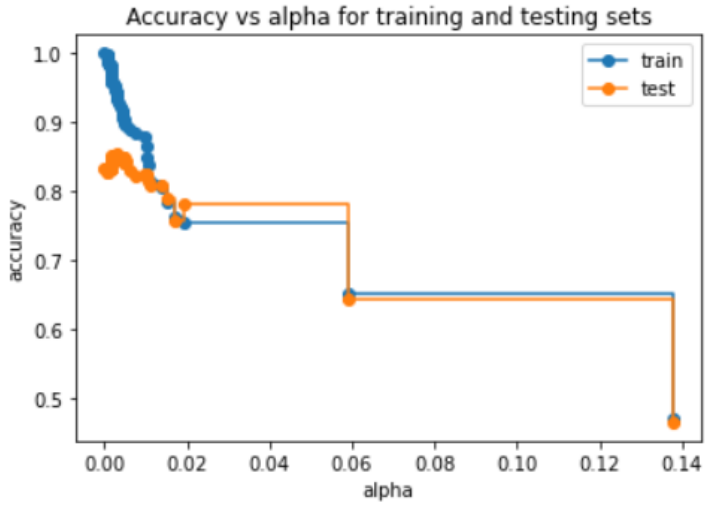
سوال ۱۱:

به کمک الگوریتم cost complexity درخت تصمیمی که در سوال قبلی ساختیم را هرس میکنیم.
این الگوریتم یک مقدار الفا دارد که به کمک آن عمل هرس کردن را انجام میدهیم در این روش از کراس ولیدیشن دیگری از داده های آموزش استفاده میکنیم.


Total Impurity vs effective alpha for training set
total impurity of leaves
effective alpha
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14


Number of nodes vs alpha
number of nodes
alpha
0
100
200
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14

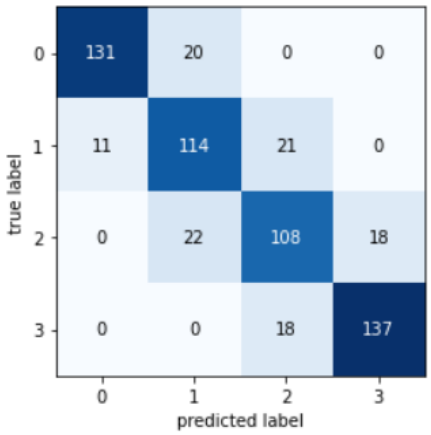

Depth vs alpha
depth of tree
alpha
5
10
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14


Accuracy vs alpha for training and testing sets
accuracy
alpha
train
test
0.5
0.6
0.7
0.8
0.9
1.0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14

همانطور که در نمودار اخر میبینیم بیشترین دقت تست ساختگی از آموزش بر روی مقدار ۰.۰۰۳ بیشترین مقدار را دارد پس درخت تمیمی با الفای برابر این مقدار میسازیم و دو درصد افزایش دقت مشاهده میکنیم:

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.87	0.89	151
1	0.73	0.78	0.75	146
2	0.73	0.73	0.73	148
3	0.88	0.88	0.88	155
accuracy			0.82	600
macro avg	0.82	0.82	0.82	600
weighted avg	0.82	0.82	0.82	600

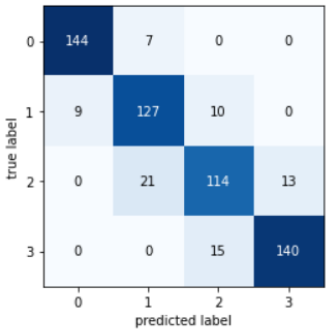


سوال ۱۲:

یک جنگل تصادفی بر روی داده ها آموزش میدهیم و به دقت های زیر میرسیم:

Classification Report:

	precision	recall	f1-score	support
0	0.94	0.95	0.95	151
1	0.82	0.87	0.84	146
2	0.82	0.77	0.79	148
3	0.92	0.90	0.91	155
accuracy			0.88	600
macro avg	0.87	0.87	0.87	600
weighted avg	0.87	0.88	0.87	600



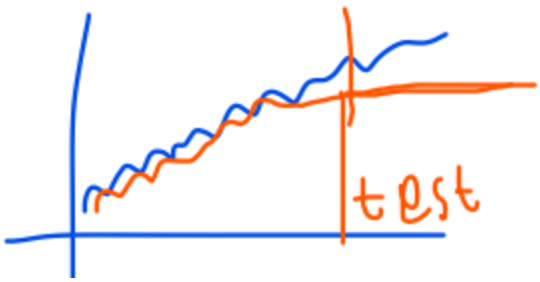
۸ درصد دقت بهتری نسبت به درخت تصمیم معمولی کسب میکنیم و این بدلیل ماهیت جنگل تصادفی است چون در این الگوریتم به جای یک درخت تصمیم چندین درخت تصمیم آموزش داده میشود و نتایج کلی مدل به کمک رای گیری بین این درخت های کوچک که هر کدام به صورت رندومکات های متفاوتی زده اند ساخته میشود و دقت بهتری کسب میشود.

سوال ۱۳:

همانطور که به صورت کامل در سوال ۱۰ اشاره شد درخت های تصمیم بر روی داده های نا همگن خیلی خوب عمل میکنند و در بعضی مسائل از این قبیل در رقبای آن ها مانند شبکه های عصبی بهتر عمل میکنند مثلا در مبحث ریکامندر سیستم ها مدل های درختی سال هاست که رتبه های برتر مسابقات را کسب میکنند این مدل ها به دلیل سادگی تسفیر پذیری و مقاوم بودن به داده های غیر بالانس کاربرد های زیادی دارند این مدل ها اکثرا خیلی سریع تر از شبکه های عصبی آموزیش میبینند و این امکان تست های بیشتری را به طراحان مدل ها میدهند پ این مدل ها ساختار درونی روشنی دارند و بیشتر شبکه های عصبی به صورت بلک باکس استفاده میشوند اما این مدل ها روشن تر هستند و میتوان به خوبی درک کرد که چه اتفاقی در آن ها می افتد میتوان بر اساس نیاز دیباگ راحتتری داشت و تغیرات لازم را داد
تفسیر پذیری این مدل ها کمک میکند ک بتوان از داده ها اینسایت استخراج کرد مثلا میتوان روابط مخفی بین فیچر ها را کشف کرد که تاثیر زیادی بر تصمیم گیری میکند.

سوال۱۵ :

بسته به نوع نگاه به مسئله میتوان استفاده کرد و نکرد.
اگر به صورت کلاسیک و ساده بخواهیم قیمت بعدی را پیشبینی کنیم در این روش خیر نمیشود چون تغیرات قیمت در کندل های بعد در نزدیکی کندل فعلی است و مدل به شدت اورفیت میشود و به صورت رندوم اگر کدل جدیدی بایند قیمت فعلی را نگاه میکند و رندوم اندکی بالاتر از آن یا پایین تر از آن را پیشبینی میکند و دقت همیشه روی 50 درصد باقی میماند، از طرف دیگر اگر ترند روبه بالایی داشته باشیم مثلا میانگین قیمت های آموزش در 100 دلار باشد اما در تست به 1000 برسد در این سناریو مدل های درختی شکست میخورند چون عدد هایی بازه 1000 را ندیده اند و پیشبینی رندومی مانند زیر میدهند



برای حل این مشکل باید داده هارا نرمال کرد و این امر باعث دیتا لیکیج میشود و چالش اصلی نوعی نرمال کردن بدون دیتا لیکیج میباشد.

اما اگر بتوان نوعی مسئله را نسبی کرد مثلا درصد رشد یا متغیر های هدف از این قبیل که مشکل اورفیتینگ ویژگی قیمت را نداشته باشند مدل های درختی به شدت خوب جواب میدهند.

سوال ۱۶:

داده های بیتکوین دانلود شد و به کمک کد زی تمیز سازی شدند و ویژگی های هدف از آن ها استخراج شد:

```
data = pd.read_csv(path+'Bitcoin Historical Data - Investing.com.csv',
thousands=',')
data['Date'] = pd.to_datetime(data.Date)
data['target'] = data['Price'].shift(1)
data = data.dropna()
data['label'] = data['target'] - data['Price']
data['label'] = (data['label'] / abs(data['label'])) + 1
data['label'] = data['label'] /2
data['Vol.'] = pd.to_numeric(data['Vol.'].apply(lambda x: x[:-1]))
data['Change %'] = pd.to_numeric(data['Change %'].apply(lambda x: x[:-1]))
data = data.fillna(0)
train = data[data.Date < '2020-01-02' ]
test = data[data.Date >= '2020-01-02' ]
feat = ['Price', 'Open', 'High', 'Low', 'Vol.', 'Change %']
print(train.shape)
print(test.shape)
```

۳۴۵۵ داده آموزش و ۴۸۵ داده تست داریم.

سوال ۱۷:

در این بخش ده مدل مختلف و کلاسیک و جدید را مورد بررسی قرار دادیم و بر روی مقدار قیمت استپ بعدی تست گرفتیم (تارگت در هر سطر قیمت کندل بعدی است. نتایج زیر را بدست آوردیم:

Model	Train RMSE	Test RMSE	Train Accuracy	Test Accuracy
LinearRegression	204.181	1096.603	0.679595	0.979381
Ridge	204.181	1096.603	0.679595	0.979381
Lasso	204.1818	1096.353	0.680174	0.979381
BayesianRidge()	204.1839	1096.564	0.679884	0.979381
MLPRegressor	206.6573	1120.955	0.813314	0.979381
RandomForestRegressor	89.02811	15665.66	0.98466	0.694845
AdaBoostRegressor	309.7614	15885.35	0.309407	0.686598
GradientBoostingRegressor	128.167	15692.1	0.692041	0.698969
DecisionTreeRegressor	0.01553	15494.76	1	0.651546
CatBoostRegressor	93.68759	16436.54	0.673227	0.674227

سوال ۱۸:

سه مدل برتر از مرحله قبل lassoوBayesianRidgeو MLPRegressor را انتخاب کردیم

به کمک کد زیر این ۳ مدل را انسمبل کردیم و به نتایج زیر رسیدیم:

روش voting:

```
model = VotingRegressor([('Lasso', Lasso()),
                         ('BayesianRidge', BayesianRidge()),
                         ('MLPRegressor', MLPRegressor())])
model.fit(train[feat], train['target'])
train_pred = model.predict(train[feat])
pred = model.predict(test[feat])
train_rmse, train_acc, test_rmse, test_acc = eval_report(y_train = train['target'],
                                                         pred_train = train_pred,
                                                         y_test = test['target'],
                                                         pred_test = pred,
                                                        is_print = True)
```

```
Train Results :

RMSE : 205.96305347824543
Accuracy with 5% : 0.685383502170767

Test Results :

RMSE : 1132.6780972525842
Accuracy with 5% : 0.9752577319587629
```

در این روش دقت مدل انسمبل افت کرد.

روش bagging:

در این روش ابتدا هر سه مدل پیشبینی خود را انجام میدهند و جواب نهایی میانگین این مقادیر است:

```
model1 = Lasso()
model2 = BayesianRidge()
model3 = MLPRegressor()

model1.fit(train[feat], train['target'])
train_pred1 = model1.predict(train[feat])
pred1 = model1.predict(test[feat])

model2.fit(train[feat], train['target'])
train_pred2 = model2.predict(train[feat])
pred2 = model2.predict(test[feat])

model3.fit(train[feat], train['target'])
train_pred3 = model3.predict(train[feat])
pred3 = model3.predict(test[feat])

pred = (pred1+pred2+pred3)/3
train_pred = (train_pred1+train_pred2+train_pred3)/3
train_rmse, train_acc, test_rmse, test_acc = eval_report(y_train = train['target'],
                                                         pred_train = train_pred,
                                                         y_test = test['target'],
                                                         pred_test = pred,
                                                        is_print = True)
```

```
Train Results :

RMSE : 205.0645017748052
Accuracy with 5% : 0.6850940665701881

Test Results :

RMSE : 1086.3938782976395
Accuracy with 5% : 0.9835051546391752
```

دقت یک رصد افزایش پیدا کرد.

روش boosting:

در این روش مدل ها به ترتیب نظر خود را برای پیشبینی میدهند و مدل بعدی پیشبینی مدل قبلی را هم به عنوان ورودی میگیرد و بر روی فیچر های اصلی و نظر مدل های قبل از خود نظر میدهند.

```
boost_train_x = train[feat].copy()
boost_test_x = test[feat].copy()
boost_train_y = train['target'].copy()
boost_test_y = test['target'].copy()


model1 = Lasso()
model2 = BayesianRidge()
```

```
model3 = MLPRegressor()

model1.fit(boost_train_x, boost_train_y)
boost_train_x['pred1'] = model1.predict(boost_train_x)
boost_test_x['pred1'] = model1.predict(boost_test_x)

model2.fit(boost_train_x, boost_train_y)
boost_train_x['pred2'] = model2.predict(boost_train_x)
boost_test_x['pred2'] = model2.predict(boost_test_x)

model3.fit(boost_train_x, boost_train_y)
train_pred = model3.predict(boost_train_x)
pred = model3.predict(boost_test_x)

pred = (pred1+pred2+pred3)/3
train_pred = (train_pred1+train_pred2+train_pred3)/3
train_rmse, train_acc, test_rmse, test_acc = eval_report(y_train = train['target'],
                                                         pred_train = train_pred,
                                                         y_test = test['target'],
                                                         pred_test = pred,
                                                        is_print = True)
```

```
Train Results :

RMSE : 204.99367317582505
Accuracy with 5% : 0.6879884225759768

Test Results :

RMSE : 1102.0326468190078
Accuracy with 5% : 0.9814432989690721
```

دقت از رای گیری بهتر شد اما از بگینگ کمتر شد مقداری.

سوال ۱۹:

در روش adaboostسه هایپر پارامتر lossو learning_rate و n_estimatorsرا انتخاب کردیم و به صورت گیرید سرچ سعی کردیم مقدار های بهینه را برای آن ها ازمایش کنیم و به نتایج زیر رسیدیم

loss	learning_rate	n_estimators	Train RMSE	Test RMSE	Train Accuracy	Test Accuracy
linear	0.2	50	277.0023	16241.63	0.31809	0.684536
linear	0.5	100	291.516	15845.11	0.294356	0.684536
square	0.9	50	310.0312	15579.58	0.427786	0.684536
square	0.5	50	297.3003	15595.78	0.37424	0.686598
linear	0.9	50	290.5518	16011.13	0.389291	0.68866
linear	0.9	150	313.0941	15844.61	0.375109	0.68866
square	0.2	50	291.018	15791.05	0.304776	0.68866
exponential	0.2	50	276.2881	16087.27	0.327062	0.690722

square	0.2	150	308.3506	15714.44	0.360926	0.690722
exponential	0.5	100	313.5398	16008.14	0.382923	0.692784
exponential	0.9	50	327.8406	15958.29	0.308828	0.692784
square	0.9	150	367.8154	15534.45	0.298698	0.692784
square	0.9	100	315.195	15584.59	0.336903	0.692784
exponential	0.9	150	538.435	16000.23	0.307959	0.692784
square	0.2	100	319.0803	15723.89	0.278726	0.692784
linear	0.9	100	306.8101	15843.3	0.413893	0.692784
exponential	0.5	150	403.7361	15930.12	0.309696	0.692784
linear	0.2	150	272.2116	16218.31	0.317221	0.694845
linear	0.2	100	272.3681	16001.07	0.319537	0.694845
exponential	0.5	50	304.7049	15937.35	0.363531	0.696907
exponential	0.2	100	297.2219	15924.45	0.288567	0.696907
linear	0.5	50	309.3212	15943.24	0.297829	0.696907
exponential	0.2	150	299.0701	15910.09	0.397974	0.696907
square	0.5	100	312.4441	15600.43	0.326773	0.696907
linear	0.5	150	301.3945	15915.36	0.336903	0.698969
square	0.5	150	321.2077	15622.43	0.30275	0.698969
exponential	0.9	100	476.6882	15945.71	0.301013	0.698969

بهترین مدل با تابع خطای نمایی و لرنینگ ریت ۰.۹ و ۱۰۰ مدل کوچک میباشد.

سوال ۲۰:

برای مدل جنگل تصادفی مراحل سوال قبل را با سه هایپر پارامتر max_depth وn_estimatorو criterion انجام دادیم و به نتایج زیر رسیدیم:

max_depth	criterion	n_estimators	Train RMSE	Test RMSE	Train Accuracy	Test Accuracy
150	mae	50	91.82992	15717.49	0.98437	0.68866
50	mse	50	92.08983	15668.67	0.985239	0.690722
50	mae	50	92.24422	15650.85	0.98437	0.690722
50	mse	100	90.51432	15712.35	0.985528	0.690722
50	mae	100	89.61587	15663.44	0.98437	0.690722
100	mae	50	88.72511	15632.73	0.986397	0.690722
150	mse	150	89.81668	15721	0.984949	0.690722
150	mae	100	88.83817	15712.94	0.984949	0.692784
150	mse	100	92.85508	15607.56	0.98466	0.692784
150	mse	50	91.42234	15647.13	0.985239	0.692784
100	mse	100	89.63912	15704.23	0.986107	0.692784

100	mse	50	90.60269	15739.61	0.98466	0.692784
50	mae	150	88.90039	15684.42	0.984081	0.692784
50	mse	150	87.87277	15692.3	0.985239	0.692784
100	mse	150	86.49656	15675.72	0.984949	0.692784
100	mae	150	88.89796	15709.36	0.985528	0.694845
150	mae	150	88.40289	15728.81	0.985528	0.694845
100	mae	100	91.00276	15636.85	0.984949	0.698969

دقت مدل های جنگل تصادفی خیلی به این سه پارامتر وابسطه نیست و به کلی مدل نمیتواند دقت بیشتری از خود نشان دهد به دلیل نوعی که مسئله را تعریف کردیم این مشکل رخ داده است، در سوال ۱۵ کامل توضیح داده شد.

سوال ۲۱:

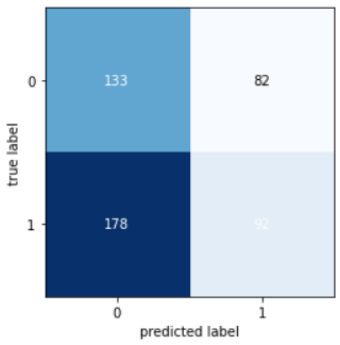
تمامی نتایج سوالات قبل در اساس rmse و accuracy 0.05% محاسبه و گزارش شدند.

سوال ۲۲:

مسئله سوال قبل را به صورت طبقه بندی حل میکنیم به این صورت که اگر قیمت کاهش یابد کلاس ۰ اگر افزایش یابد کلاس ۱ و به نتایج زیر با دو مدل random forest و catboost رسیدیم

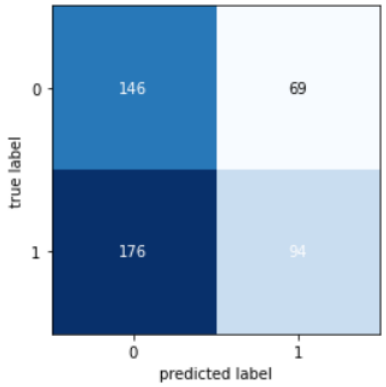
جنگل تصادف:

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

         0.0       0.43      0.62      0.51       215
         1.0       0.53      0.34      0.41       270

    accuracy                           0.46       485
   macro avg       0.48      0.48      0.46       485
weighted avg       0.48      0.46      0.45       485
```



Catboost:

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

         0.0       0.45      0.68      0.54       215
         1.0       0.58      0.35      0.43       270

    accuracy                           0.49       485
   macro avg       0.52      0.51      0.49       485
weighted avg       0.52      0.49      0.48       485
```

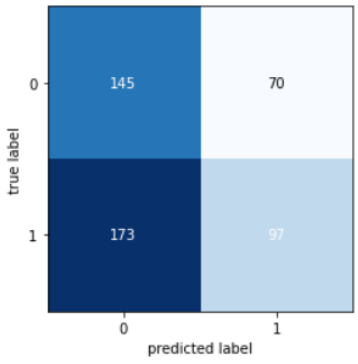


متاسفانه با فیچر های فعلی و ماهیت نوسانی بازار های مالی نمیتوانیم این بازار را با این رویکرد حل کنیم چون دقت مدل ها مانند سکه انداخت هستند و نمیتوان روی آن ها حساب کرد.

سوال ۲۴:

مدل xgboostرا بر روی مسئله رگرسون و طبقه بندی اجرا کردیم و به نتایج زیر رسیدیم:

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

         0.0       0.46      0.67      0.54       215
         1.0       0.58      0.36      0.44       270

    accuracy                           0.50       485
   macro avg       0.52      0.52      0.49       485
weighted avg       0.53      0.50      0.49       485
```




```
Train Results :

RMSE : 31.322374853637452
Accuracy with 5% : 0.8057887120115774

Test Results :

RMSE : 15692.485235521754
Accuracy with 5% : 0.6927835051546392
```

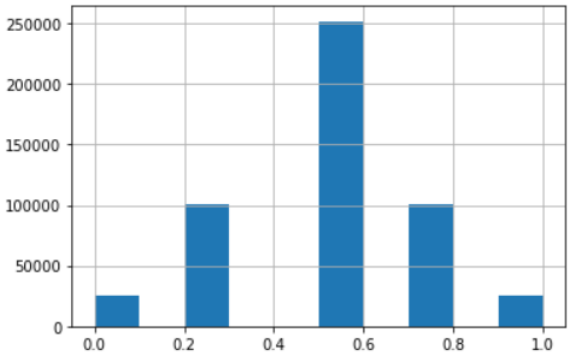
این مدل از نسل های بعدی مدل درخت تصمیم هستند که با بهره گیری از ایده های گرادیان و نرخ یادگیری سعی میکنند فضای مسئله را به مناطق بهتر و جنرال تری تقسیم بندی کنند

همانطور که میبینیم دقت رگرسون آن بد نیست اما مدل طبقه بند همچنان مشابه سکه انداختن میباشد.

سوال ۲۶:

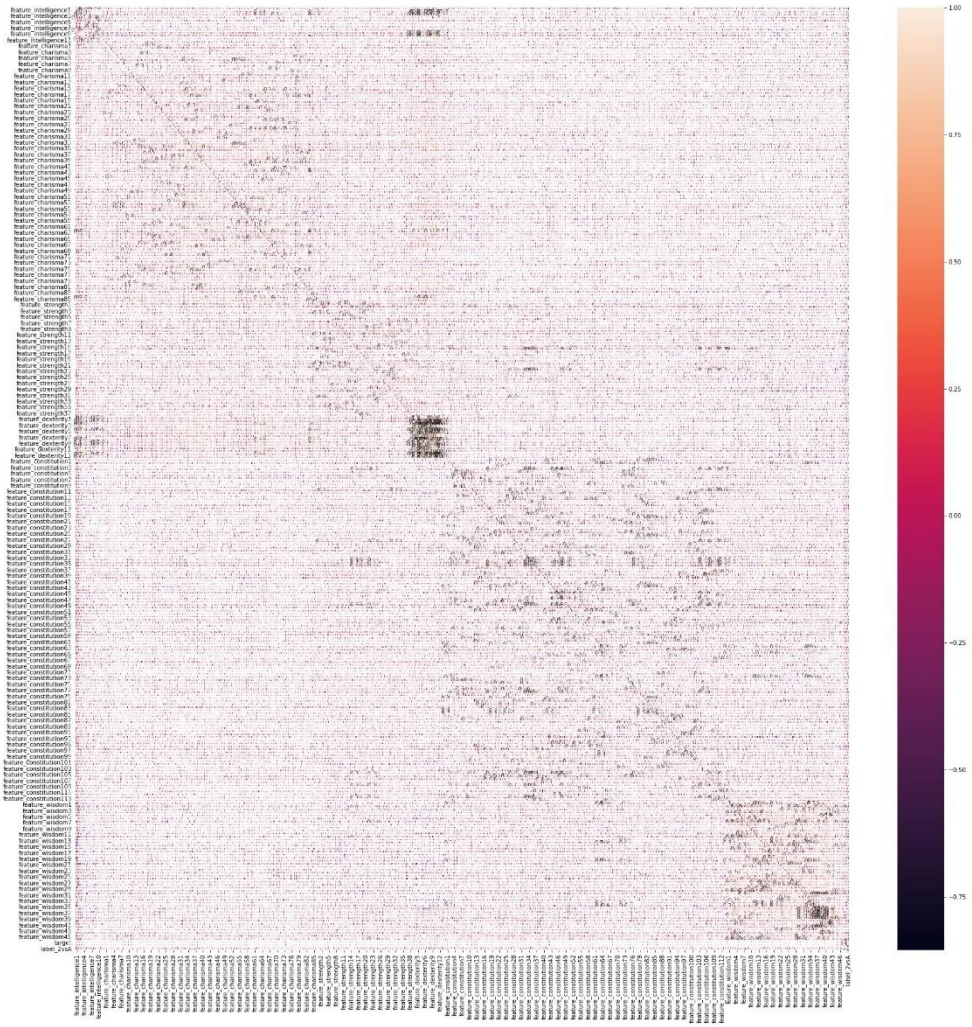
در این تمرین ما 501808 داده آموزش و 137779 داده ست داریم که تمامی آن ها داری 314 فیچر مختلف هستند و ما هیچ ایده ای در رابطه با اینکه فیچر ها چی هستند نداریم، و به طرز عجیبی تمامی توزیع هایی که میبینیم نرمال هستند و بین بازه ۰ و ۱ پیشبینی من این است که ' ,'feature_dexterity 'feature_constitution', 'feature_intelligence', 'feature_strength', 'feature_wisdom', 'feature_charisma'' ۶ مدل مختلف آماده شده بوده و هر یک ازآن ها وظیفه پیشبینی روند افزایش یا کاهش بازار در گام بعدی را به صورت احتمالاتی برمیگردانند و از آن جاییکه این مدل ها اصلا موفق به انجام درست این کار نشده اند اکثر توزیع ها توزیع نرمال با میانگین ۰.۵ میباشد.

اولین کاری که ما کردیم توزیع برچسب های کلاس ها رصد کردیم:

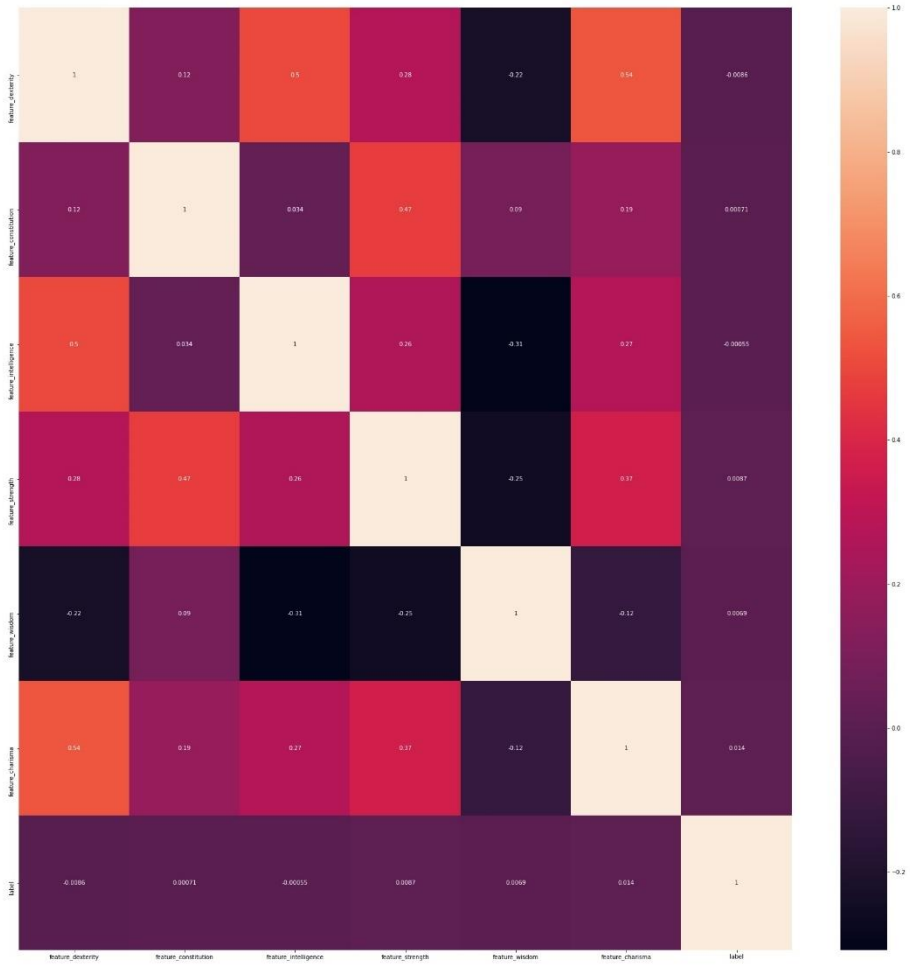


تسک را به صورت طبقه بندی نگاه کردیم و سعی کردیم داده ها را در ۵ کلاس مختلف دسته بندی کنیم،

ایده اولی که اکسپلور کردیم این بود که آیا بین فیچر ها کورولیشن وجود دارد یا نه



در این پلات نواحی مربع شکل با کورولینشن منفی بالا میبینیم که اکثرا مربوط به یک ست از ویژگی ها هستند ابتدا هر ست از ویژگی ها را که از یک نوع با عدد متفاوت هستند را بر میداریم و میانگین آن ها را به عنوان نمایده آن ها قرار میدهیم سپس ماتریس کورولیشن را مجددا چک میکنیم



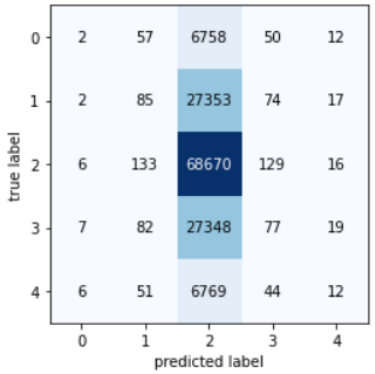
پترن کلی در پلات قبل تر را همچنان مشاهده میکنیم لازم به ذکر است که هیچ یک از فیچر ها با کلاس کورولیشن زیادی نداشتند کرولیشین با لیبل کلاس

```
feature_dexterity7    -0.012175
feature_dexterity6    -0.011807
feature_dexterity4    -0.011706
feature_charisma69    -0.010858
feature_dexterity11   -0.010820
                         ...
feature_charisma18     0.010509
feature_charisma19     0.010543
feature_charisma37     0.010719
feature_strength14     0.011353
```

```
feature_strength34      0.012310
```

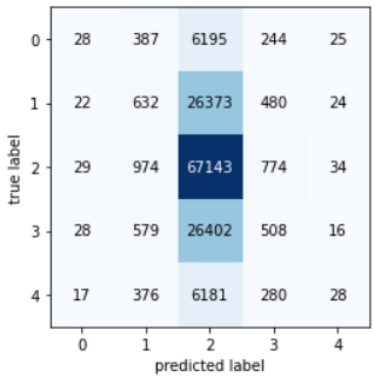
دیتا ست حاصل از میانگین را به همراه ۶ فچیر و مدل catboost طبقه بندی میکنیم:

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

         0.0       0.09      0.00      0.00      6879
         1.0       0.21      0.00      0.01     27531
         2.0       0.50      1.00      0.67     68954
         3.0       0.21      0.00      0.01     27533
         4.0       0.16      0.00      0.00      6882

    accuracy                           0.50    137779
   macro avg       0.23      0.20      0.14    137779
weighted avg       0.35      0.50      0.34    137779
```



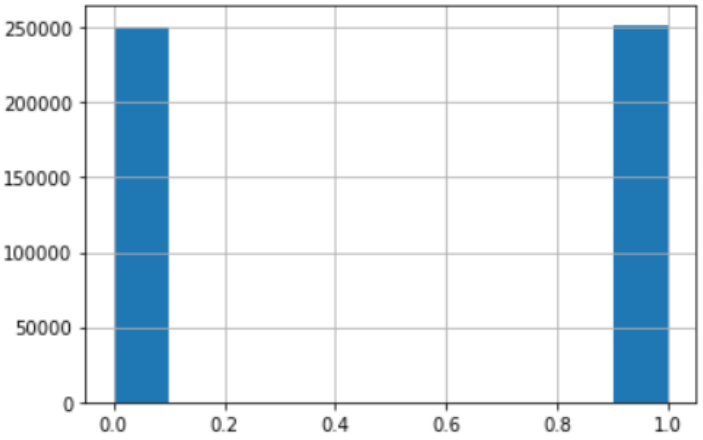
ایده بعدی به صورت خیلی معقول کلیه فیچر ها را به کمک مدل جنگل تصادفی طبقه بندی میکنیم و به نتیجه زیر میرسیم:

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

         0.0       0.23      0.00      0.01      6879
         1.0       0.21      0.02      0.04     27531
         2.0       0.51      0.97      0.67     68954
         3.0       0.22      0.02      0.03     27533
         4.0       0.22      0.00      0.01      6882

    accuracy                           0.50    137779
   macro avg       0.28      0.20      0.15    137779
weighted avg       0.36      0.50      0.35    137779
```



در مرحله بعدی سعی میکنیم بالانس نبودن داده ها را به کمکروش های زیر هندل کنیم:

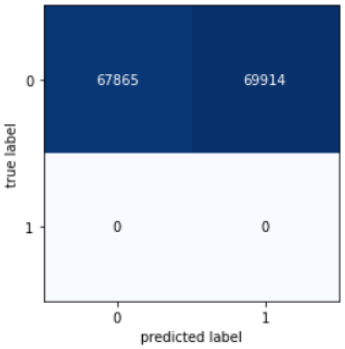
طبه بندی تعریف میکنیم که کلاس ۲ که بیشترین داده هارا دارد از باقی کلاس ها جدا کند، توزیع کلاس ها ۲ در مقابل باقی به شکل زیر است:



به کمک catboost اقدام به طبقه بند میکنیم:

	precision	recall	f1-score	support
0	1.00	0.49	0.66	137779
1	0.00	0.00	0.00	0
accuracy			0.49	137779
macro avg	0.50	0.25	0.33	137779
weighted avg	1.00	0.49	0.66	137779

```
E:\ProgramData\lib\site-packages\mlxtend\plotting\plot_
RuntimeWarning: invalid value encountered in true_divi
  normed_conf_mat = conf_mat.astype('float') / total_s
```

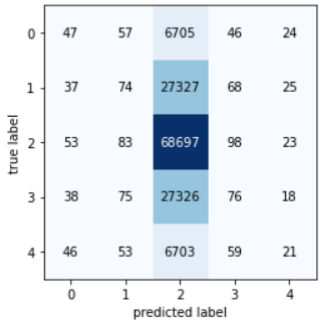


نتایج به هیچ وجه مورد قبول نیستند.

در این مرحله 20 فیچر مهمی که مدل قسمت قبل از طبقه بندی دو کلاسه پیدا کرده بود را به عنوان فیچر های ورودی بر روی مسئله ۵ کلاسه انتخاب میکنیم و به کمک catboost طبقه بندی میکنیم:

Classification Report:

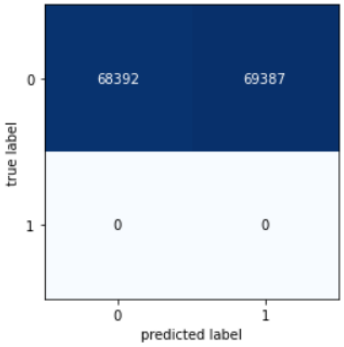
	precision	recall	f1-score	support
0.0	0.21	0.01	0.01	6879
1.0	0.22	0.00	0.01	27531
2.0	0.50	1.00	0.67	68954
3.0	0.22	0.00	0.01	27533
4.0	0.19	0.00	0.01	6882
accuracy			0.50	137779
macro avg	0.27	0.20	0.14	137779
weighted avg	0.36	0.50	0.34	137779

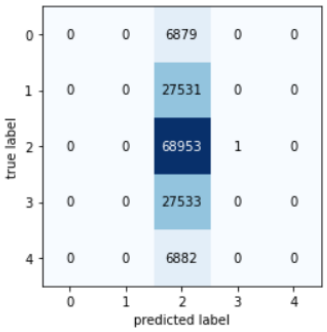


همچنان دقت بالا نمیآید و البته منطقی است که خطای مرحله قبل به صورت انباشه به این مرحله وارد شود.

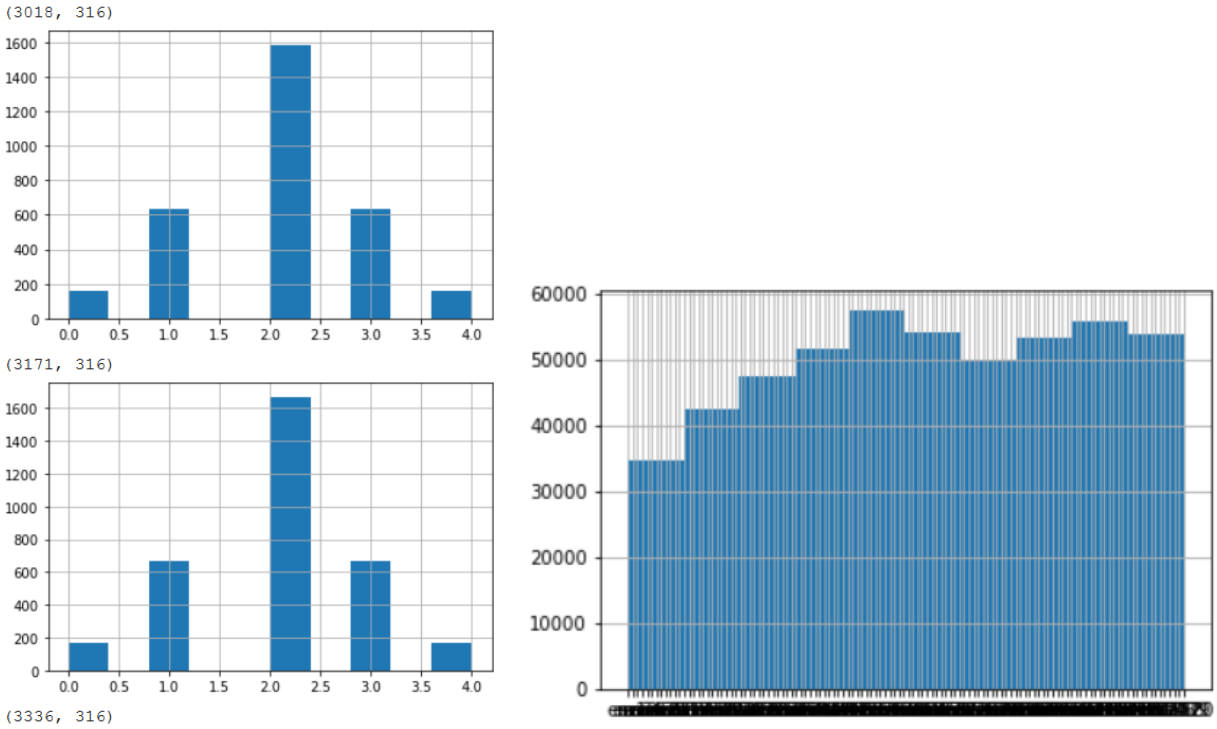
یک لاجستیک رگرسیون بر روی فیچر های ۲۰ اصلی و مسئله دو کلاسه نیز تست میکنیم:

	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	6879
1.0	0.00	0.00	0.00	27531
2.0	0.50	1.00	0.67	68954
3.0	0.00	0.00	0.00	27533
4.0	0.00	0.00	0.00	6882
accuracy			0.50	137779
macro avg	0.10	0.20	0.13	137779
weighted avg	0.25	0.50	0.33	137779





در مرحله بعدی چک میکنیم که آیا در era ها مختلف توزیع کلاس ها و یا تعداد آن ها متفاوت است که مشاهده کردیم از نظر تعداد تقریبا یک اندازه هستند و توزیع همه آن ها نرمال میباشد:

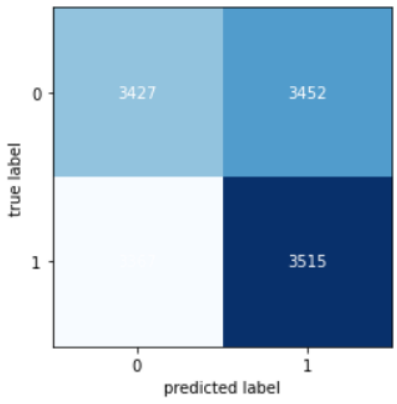


در گام های بعدی سعی کردیم کلاس هایی که در گوشه های توزیع نرمال بودند را دو به دو طبقه بندی کنیم

کلاس 0 و 4:

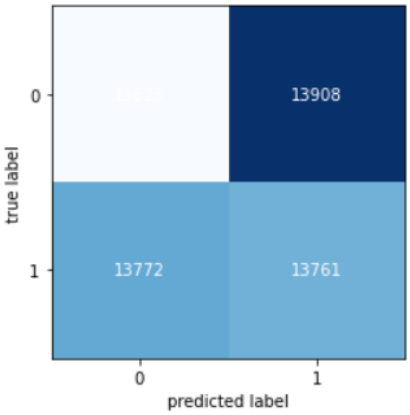
Classification Report:

	precision	recall	f1-score	support
0.0	0.50	0.50	0.50	6879
4.0	0.50	0.51	0.51	6882
accuracy			0.50	13761
macro avg	0.50	0.50	0.50	13761
weighted avg	0.50	0.50	0.50	13761



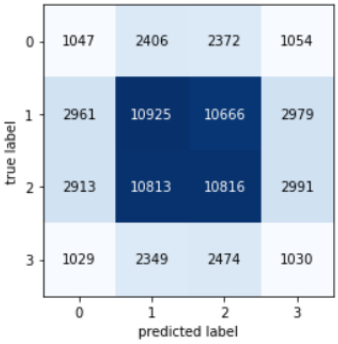
کلاس ۱و ۳:

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

         1.0       0.50      0.49      0.50     27531
         3.0       0.50      0.50      0.50     27533

    accuracy                           0.50     55064
   macro avg       0.50      0.50      0.50     55064
weighted avg       0.50      0.50      0.50     55064
```



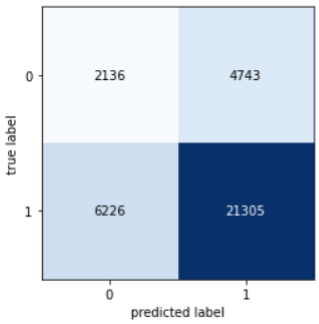
همه کلاس ها به جز ۲:

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

         0.0       0.13      0.15      0.14      6879
         1.0       0.41      0.40      0.40     27531
         3.0       0.41      0.39      0.40     27533
         4.0       0.13      0.15      0.14      6882

    accuracy                           0.35     68825
   macro avg       0.27      0.27      0.27     68825
weighted avg       0.36      0.35      0.35     68825
```



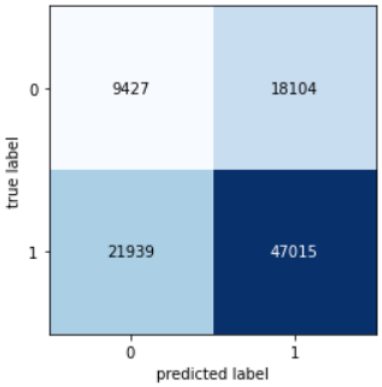
کلاس ۱و۰:

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

         0.0       0.26      0.31      0.28      6879
         1.0       0.82      0.77      0.80     27531

    accuracy                           0.68     34410
   macro avg       0.54      0.54      0.54     34410
weighted avg       0.71      0.68      0.69     34410
```



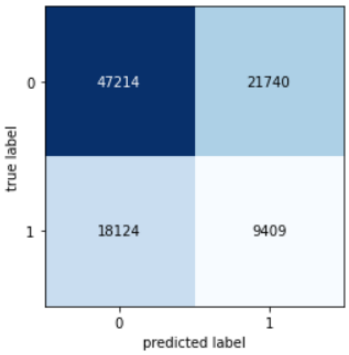
کلاس ۱و۲:

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

         1.0       0.30      0.34      0.32     27531
         2.0       0.72      0.68      0.70     68954

    accuracy                           0.58     96485
   macro avg       0.51      0.51      0.51     96485
weighted avg       0.60      0.58      0.59     96485
```



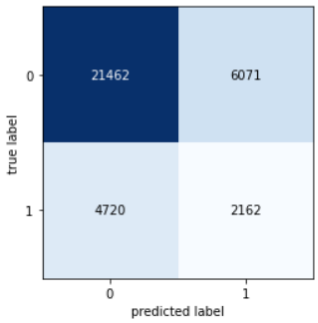
کلاس ۲و ۳:

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

         2.0       0.72      0.68      0.70     68954
         3.0       0.30      0.34      0.32     27533

    accuracy                           0.59     96487
   macro avg       0.51      0.51      0.51     96487
weighted avg       0.60      0.59      0.59     96487
```



کلاس ۳و۴:

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

         3.0       0.82      0.78      0.80     27533
         4.0       0.26      0.31      0.29      6882

    accuracy                           0.69     34415
   macro avg       0.54      0.55      0.54     34415
weighted avg       0.71      0.69      0.70     34415
```



بر روی کلاس هایی که دو به دو نزدیک به هم هستند دقتی بالاتر از ۵۰ گرفتیم اما در باقی حالات دقت بر روی ۵۰ باقی میماند.

در مرحله بعدی یک شبکه عصبی با معماری زیر بر روی داده ها اموزش دادیم:

```
Model(
  (conv): Sequential(
    (0): Conv1d(1, 256, kernel_size=(5,), stride=(1,))
    (1): ReLU()
  )
  (flatten): Sequential(
    (0): AdaptiveMaxPool1d(output_size=1)
    (1): Flatten(start_dim=1, end_dim=-1)
  )
```

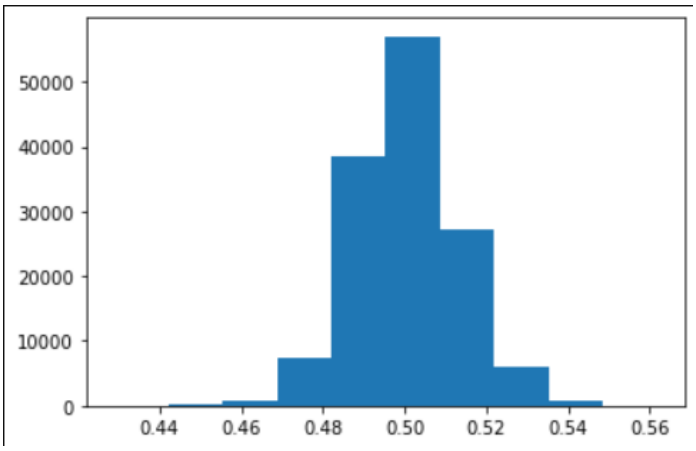
```
  (fc): Sequential(
    (0): ReLU()
    (1): Linear(in_features=256, out_features=5, bias=True)
  )
)
```

اما شبکه بعد از ۱۰ ایپاک همچنان ب روی مقدار اولیه لاس خود باقی مانده بود

```
started!
epoch 1/10 finished with train loss: 1.2806938938313999 and test loss:
1.2683333481397612
epoch 2/10 finished with train loss: 1.2751716581429224 and test loss:
1.271874274305156
epoch 3/10 finished with train loss: 1.2745362805931133 and test loss:
1.2693557801184716
epoch 4/10 finished with train loss: 1.2742652409177602 and test loss:
1.267664607690305
epoch 5/10 finished with train loss: 1.2738433708529397 and test loss:
1.2685914004225014
epoch 6/10 finished with train loss: 1.2739113095705148 and test loss:
1.2685656728903747
epoch 7/10 finished with train loss: 1.2738674873602993 and test loss:
1.267589208809918
epoch 8/10 finished with train loss: 1.273878046470537 and test loss:
1.267319800902386
epoch 9/10 finished with train loss: 1.27393921688465 and test loss:
1.267364300690688
epoch 10/10 finished with train loss: 1.2737680308251038 and test loss:
1.2687691717731708
```

پس عملا چیزی یاد نگرفته است.

در مرحله بعدی طبق گفته صورت تمرین از مدل xgboost در حالت رگرسیون با همان کانفیگ تمرین استفاده کردیم و به نتایج زیر در بخش رگرسیون رسیدیم:



پیشبینی مدل بر روی داده ها توزیع نرمالی با میانگین 0.5 بود که این مقدار به معنی دقت ۵۰ درصدی این مدل میباشد.

با توجه به تست هایی که انجام دادیم بدون دانستن اطلاعات بیشتر در مورد دیتا ست نمیتوانیم به درستی پیشبینی دقیقی از متغیر هدف بدست آوریم.