به نام خدا

پروژه‌ی درس یادگیری ماشین

استاد:

دکتر عباس حسینی

اعضای گروه:

امیرحسین باقری-98105621

ahbagheri01@gmail.com

علیرضا دهقانپور فراشاه-98101555

alirezefarashah@gmail.com

تحلیل اکتشافی داده و مهندسی ویژگی‌ها

تمامی کارهای این بخش در فایل "EDA & Feature engeniering.ipynb" قرار دارد.

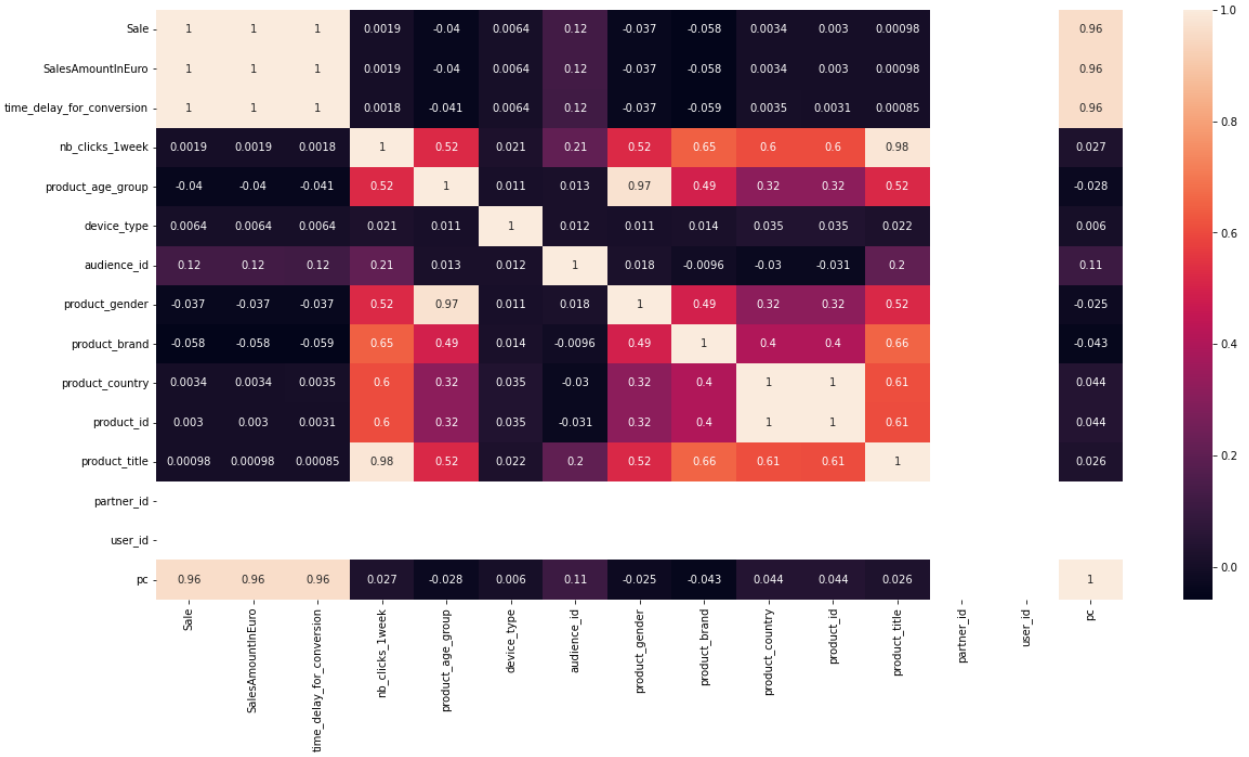
در ابتدای مسئله می‌بایست به بررسی ویژگی‌های دیتاست پرداخته‌شود. برای این منظور در ابتدا برای هر ستون تعداد مقادیر یکتای آن و تعداد مقادیر نامشخص آن را بدست آورده می‌شود.

```
Number of unique values :
Sale                          2
SalesAmountInEuro          8931
time_delay_for_conversion  9207
click_timestamp           67045
nb_clicks_1week            1137
product_price              4495
product_age_group             8
device_type                   3
audience_id                3181
product_gender               10
product_brand              4769
product_category(1)          21
product_category(2)         144
product_category(3)         698
product_category(4)         909
product_category(5)         441
product_category(6)          88
product_category(7)           0
product_country              16
product_id                45446
product_title             27694
partner_id                  183
user_id                   96766
dtype: int64
```

unique values1 Figure

ارتباط موجود بودن ستون‌ها یعنی nan بودن یا نبودن آنها با بدست اوردن correlation میان ستون‌های nan و غیرآن قابل مشاهده‌است. در تصویر مشخص است که دو فیچر SalesAmountInEuro و time_delay_for_conversion در واقع همان لیبل‌ما هستند. یعنی مقدار داشتن آنها معادل با 0 بودن مقدار Sale است. همچنین ستون product_price زمانی که ناموجود باشد حتما مقدار Sale صفر می‌شود و این فیچر طبق نکات گفته شده در پیاتزا مناسب نخواهد بود.

در میان فیچرهای مختلف audience_id دارای همبستگی بیشتری با ستون هدف است و این یعنی می‌تواند برای یادگیری مناسب باشد.



corr-heatmap2 Figure

با توجه به هیت‌مپ در ابتدا مشخص می‌شود که سه ستون اول یعنی SalesAmountInEuro و time_delay_for_conversion و product_price را حذف باید کرد.

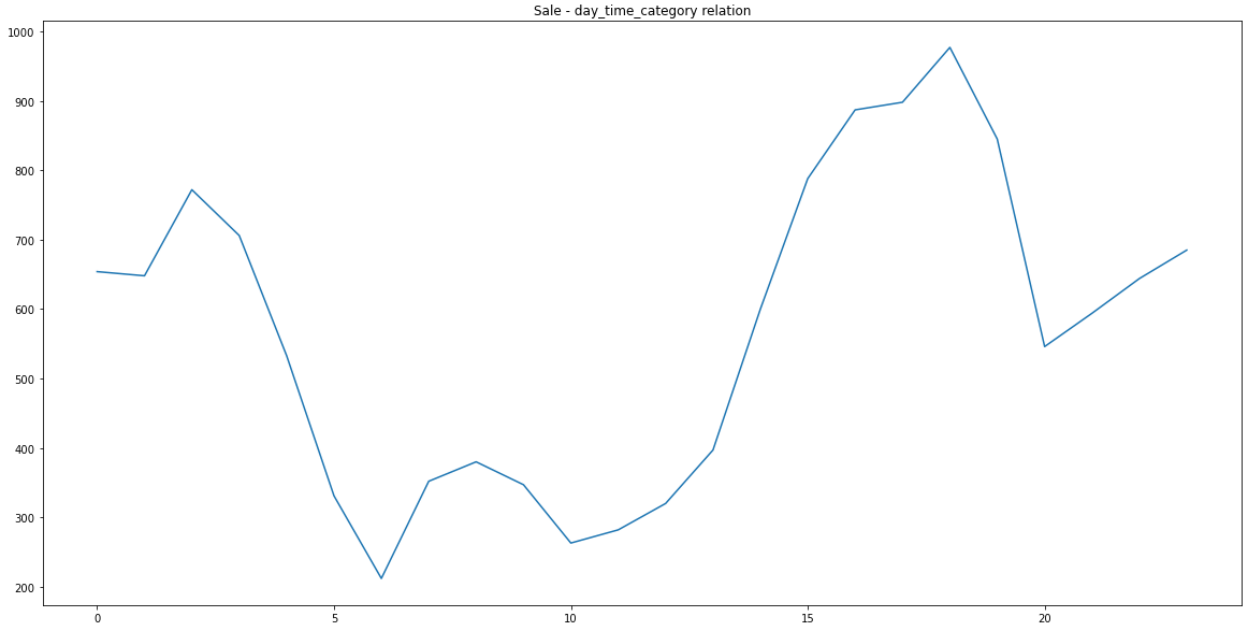
در ادامه تک تک فیچرها بررسی می‌شود.

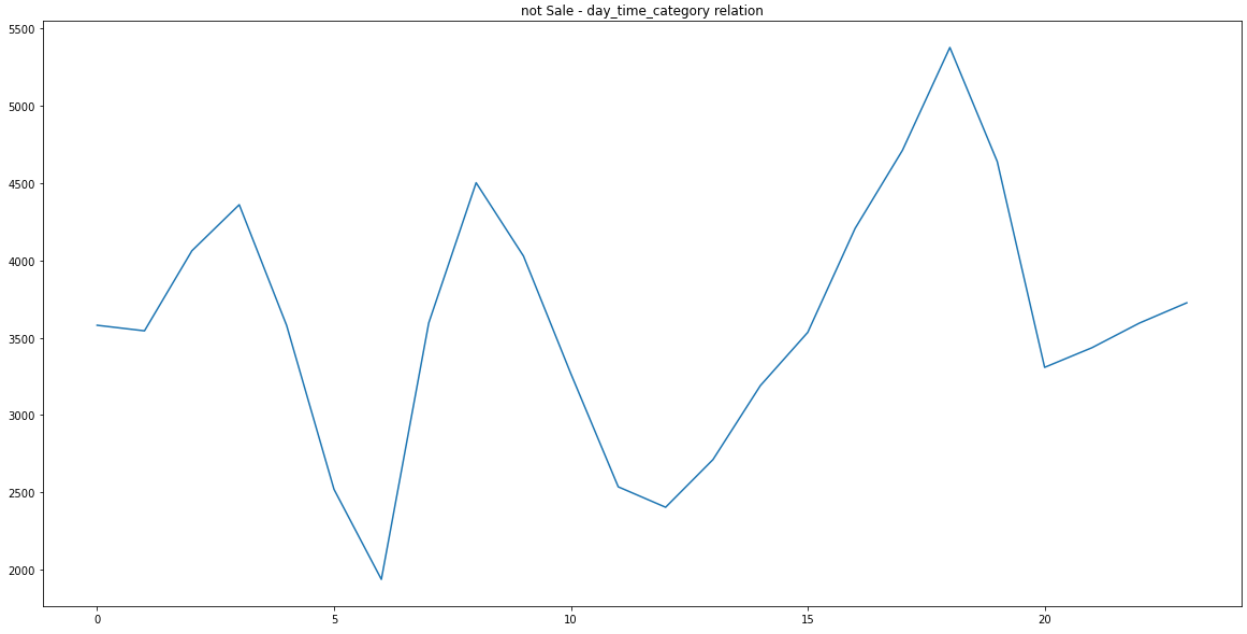
click_time_stamp

در click_time_stamp زمان وجود دارد که اطلاعات روز و ماه آن مربوط به دو روز خاص است و برای یادگیری مناسب نیست به همین‌منظور بهتر است که فقط ساعت آن درنظر گرفته‌شود و نمودار ساعت بر اساس تعداد خرید در شکل زیر قابل مشاهده است.

برای هر کلاس Sale=1 و Sale=0 نمودار جدا رسم شده است.

با توجه به عدم تفاوت زیاد ساعت‌های نزدیک به هم زمان روز به چهاردسته تقسیم می‌شود و هر بازه‌ی 6 ساعته در یک دسته قرار می‌گیرد. با این کار correlation بین این ویژگی و متغیر هدف بیشتر می‌شود.



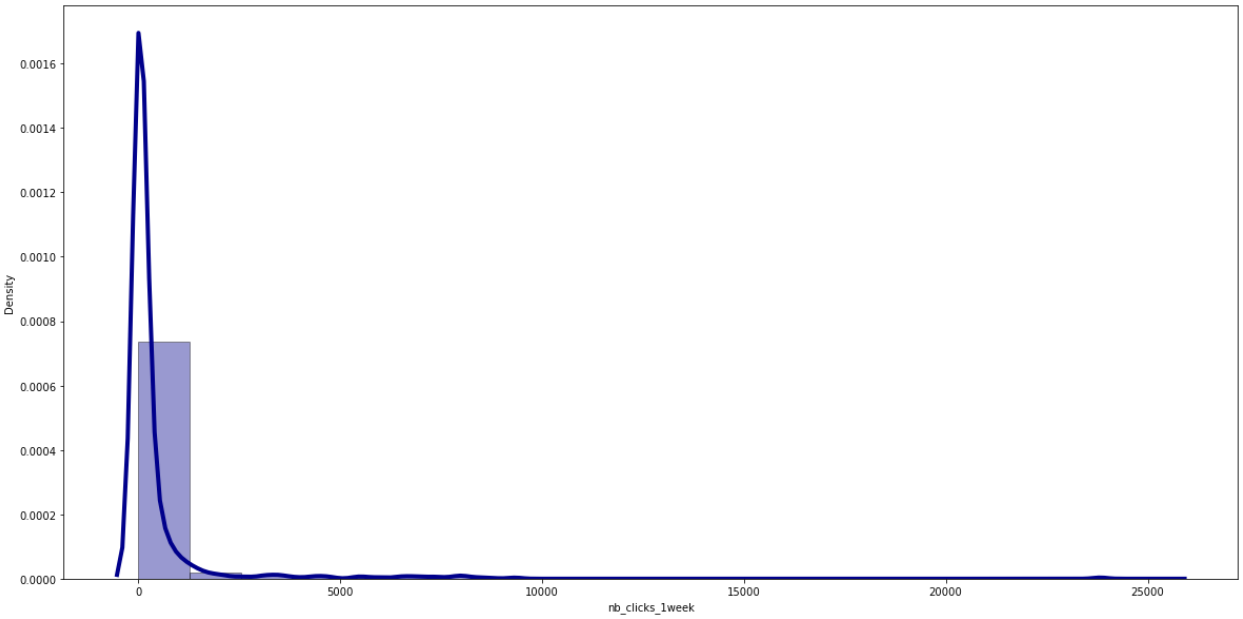


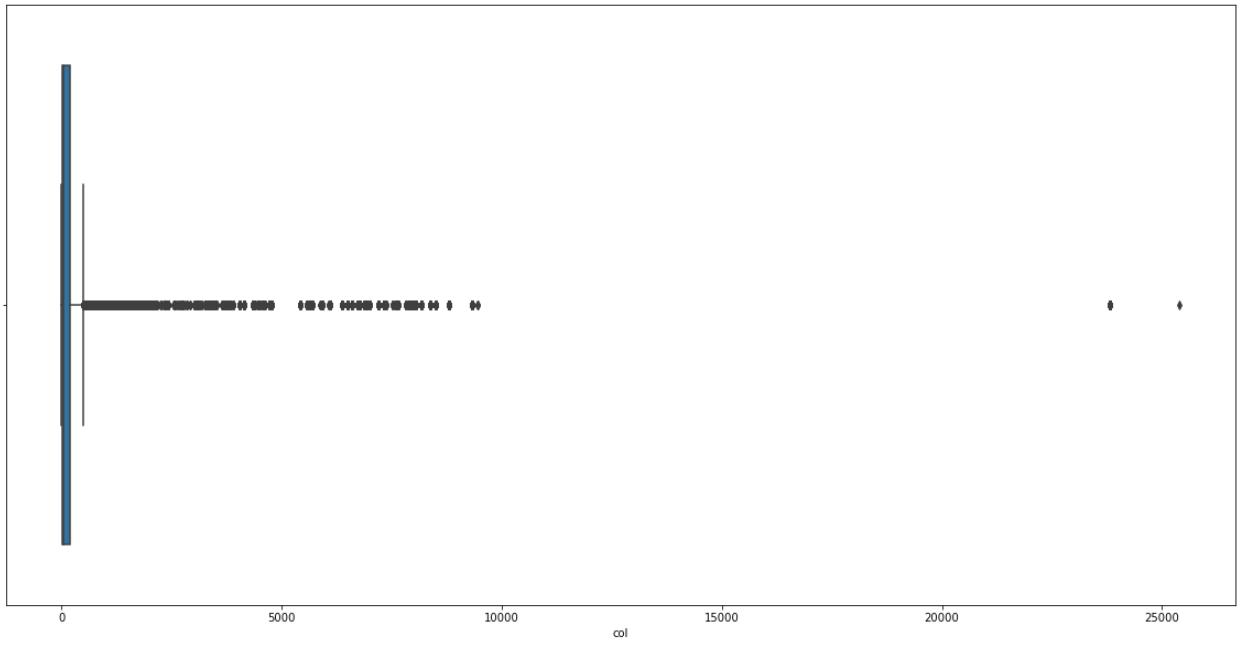
nb_clicks_1week

این ستون دارای تعداد زیادی مقدار خالی است که برای آن‌ها از میانه استفاده می‌شود زیرا دارای تعداد زیادی داده Outlier است.

Box-plot این ویژگی و هیستوگرام آن در ادامه رسم شده‌است.

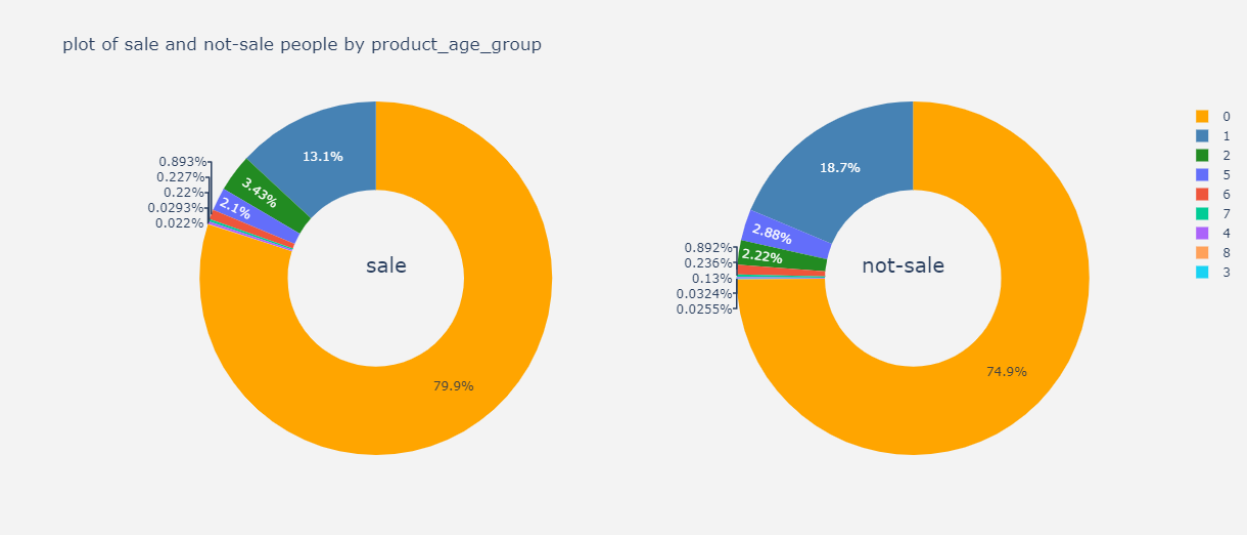
این ویژگی تنها ویژگی غیرکتگوریکالی است که در شبکه عصبی استفاده خواهیم کرد.





Product_age_group

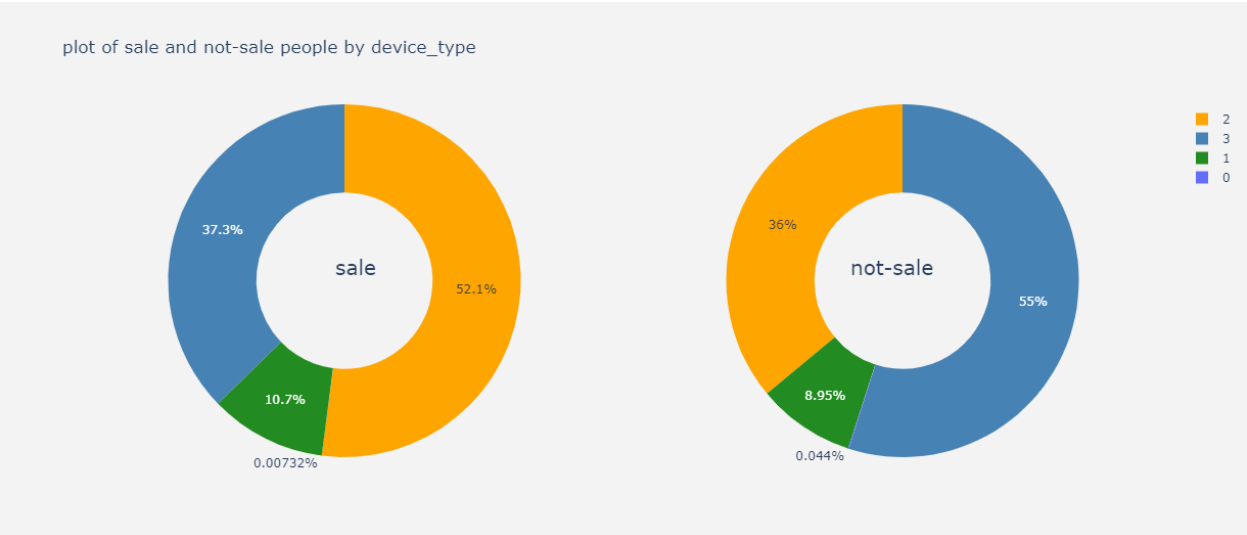
نمودار دایره‌ای این ویژگی در ادامه کشیده‌شده است که مشخص می‌کند هر مقدار از این ویژگی چه میزان sale شده‌است که با توجه به تفاوت توزیع احتمال دو نمودار می‌توان گفت که ویژگی مناسبی برای یادگیری است. برای ستون‌های غیرعددی خانه‌های nan را یک دسته جدا در نظرگرفته‌ایم. همانطور که مشخص است بخش زیادی از داده‌های این ستون دارای مقدار nan هستند.

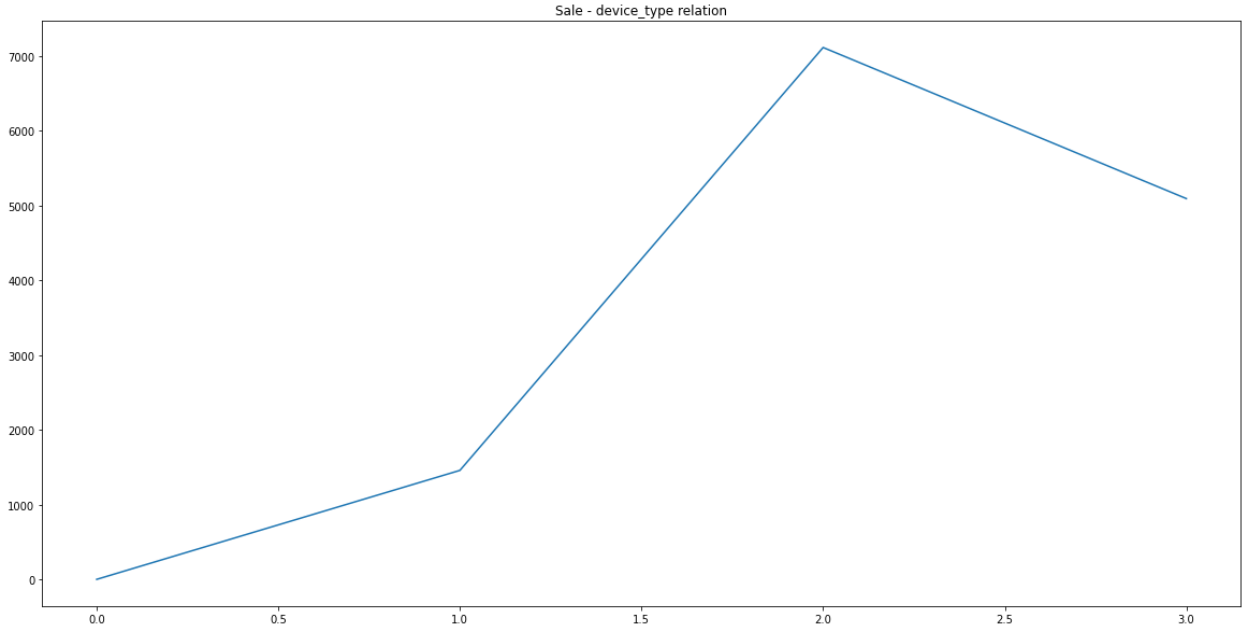


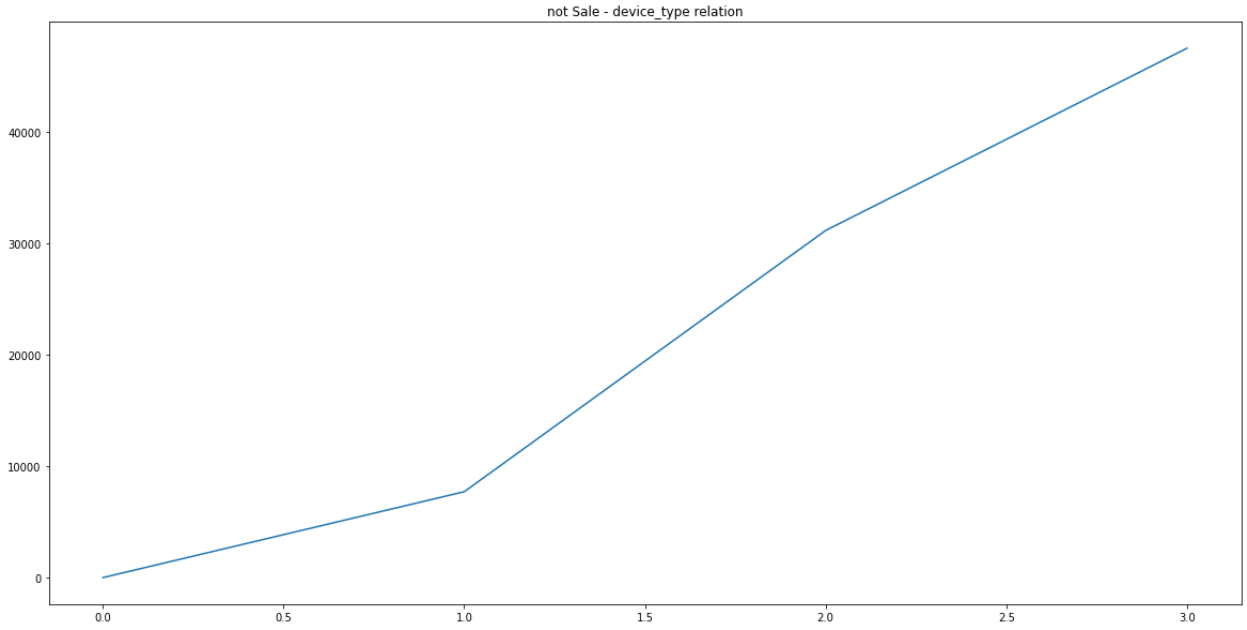
Device_type

در این ویژگی که دارای سه دسته به اضافه‌ی یک دسته برای مقادیر nan است

به مانند ویژگی قبل با مشاهده تفاوت نمودار دایره‌ای و خطی برای دو مقدار هدف متوجه می‌شویم این ویژگی برای یادگیری مناسب است.

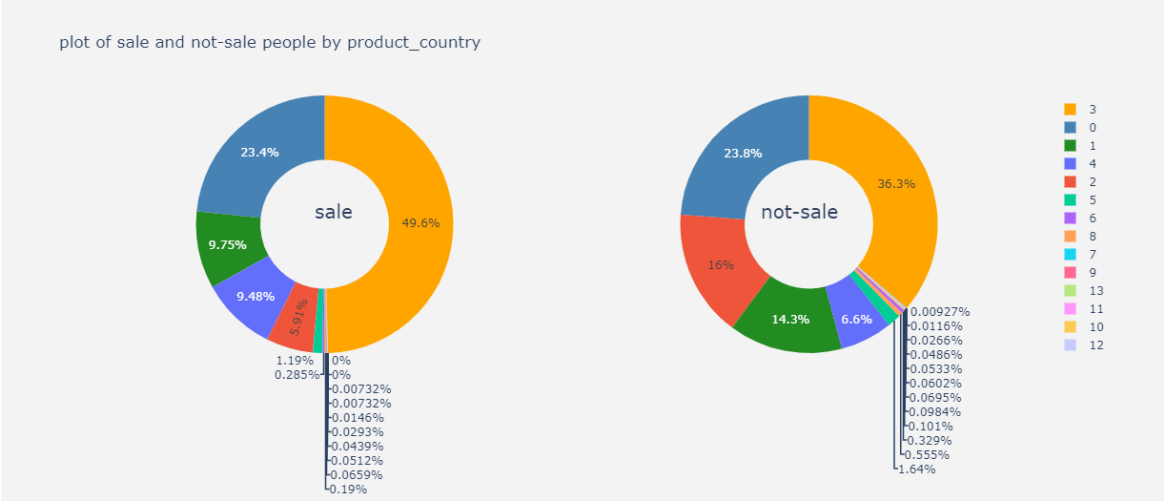


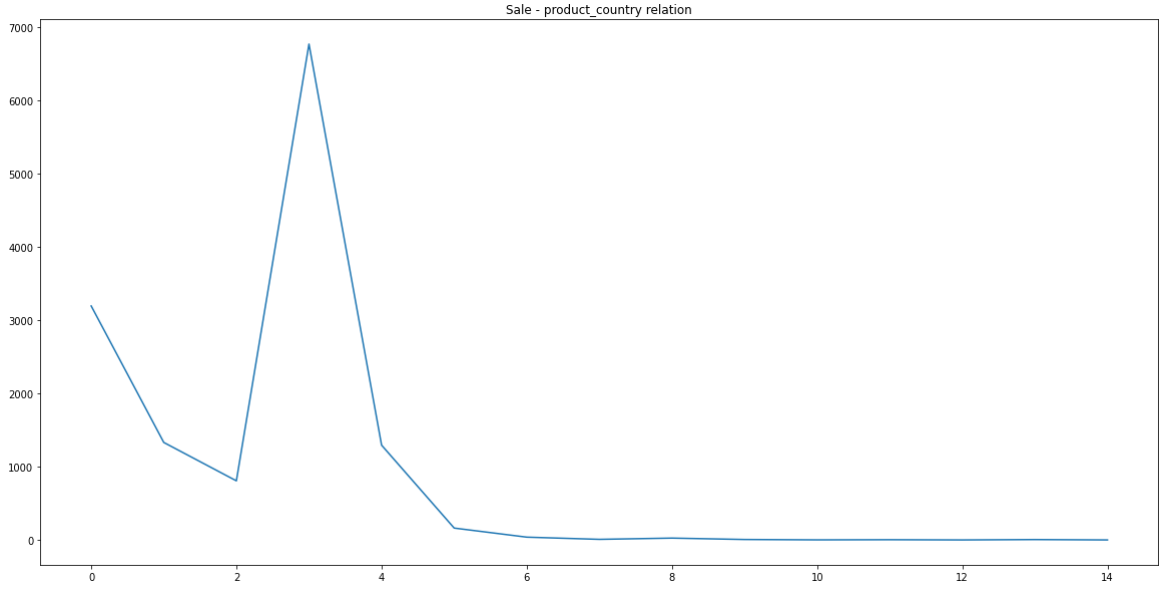


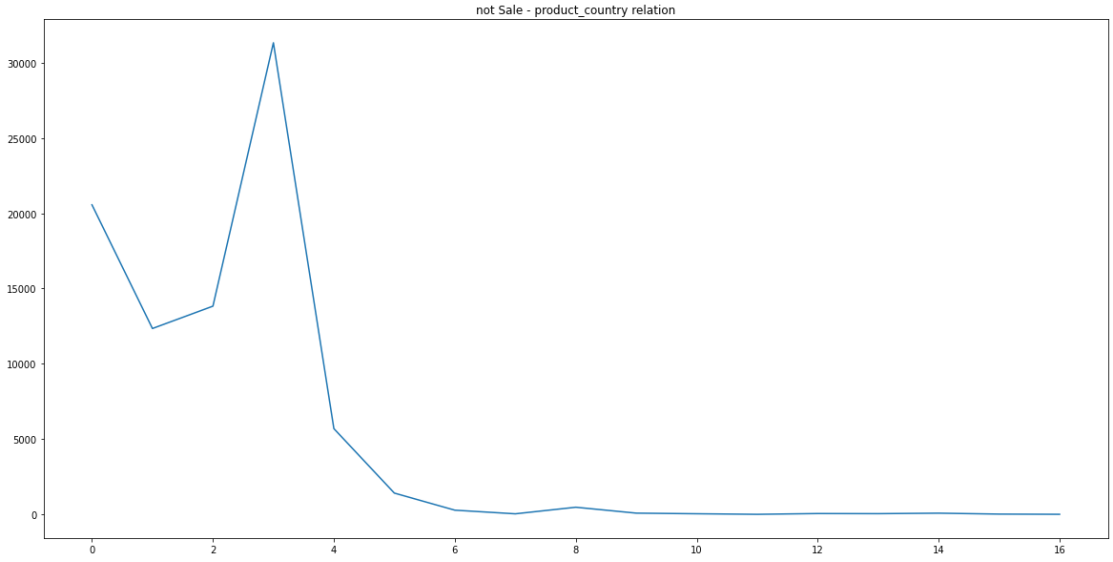


Product_country

در نمودار دایره‌ای مشخص است که تعداد خریدها در 4 کشور خاص بیشتر است و برای بقیه صفر است. می‌توان برای این 4 کشور یک دسته خاص گرفت و برای سایر کشورها یک دسته‌ی دیگر اما چون تعداد دسته‌ها کم است می‌توان برای هر دسته یک متغیر صفر و یک گرفت و آن را one-hot کرد.


plot of sale and not-sale people by product_country
sale
23.4%
49.6%
9.75%
9.48%
5.91%
1.19%
0.285%
0%
0%
0.00732%
0.00732%
0.0146%
0.0293%
0.0439%
0.0512%
0.0659%
0.19%
not-sale
23.8%
36.3%
16%
14.3%
6.6%
0.00927%
0.0116%
0.0266%
0.0486%
0.0533%
0.0602%
0.0695%
0.0984%
0.101%
0.329%
0.555%
1.64%
3
0
1
4
2
5
6
8
7
9
13
11
10
12


Sale - product_country relation
7000
6000
5000
4000
3000
2000
1000
0
0
2
4
6
8
10
12
14


not Sale - product_country relation
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14
16

از چهار ویژگی product_id, partner_id, user_id, product_title به دلیل ابعاد بسیار بالا و همچنین correlation خیلی کم هیچ استفاده‌ای نکردیم و این فیچرها هیچ information gainی با متغیر هدف نداشتند.

توجه کنید که علت استفاده نکردن از سه ویژگی گفته شده در بالا در نمودارهای زیر قابل مشاهده‌است.

توجه کنید که با داشتن product_price یعنی فقط صرف این که مقدار داشته باشد یا نه می‌توان به f1_score با مقدار 0.99 رسید. در کد زیر میتوانید مشاهده کنید:

```
In [67]: col = df["product_price_category"]
         col2 = col > 0
         col2 = col2.astype(int)
         s = df["Sale"] == col2
         sum(s)/len(s)

Out[67]: 0.99123
```

Product_category

دیتاست دارای 7 نوع کتگوری مختلف است که طبق نکات گفته شده هر کدام مستقل از دیگری ویژگی‌هایی را مشخص می‌کند. کتگوری هفتم تمامی مقادیرش صفر است و باید آن را دور ریخت. برای باقی کتگوری‌ها مقادیر nan را به عنوان یک کتگوری در نظر می‌گیریم.

کتگوری 1 و 2 دارای تعداد معقولی مقدار مختلف است ولی از 3 به بعد این تعداد زیاد می‌شود و چون دارای correlation کمی هستند خیلی مناسب نیستند. با این حال در train کردن مدل از آن‌ها استفاده شد ولی به این دلیل که تاثیری در افزایش معیار سنجش نداشتند دیگر مورد بررسی قرار نگرفتند.

مسیر منتهی به مدل نهایی

در ابتدا یک سری مدل غیرشبکه عصبی بر روی دادگان آموزش داده می‌شود تا نحوه‌ی عملکرد آن‌ها را بتوان با شبکه عصبی مقایسه کرد.

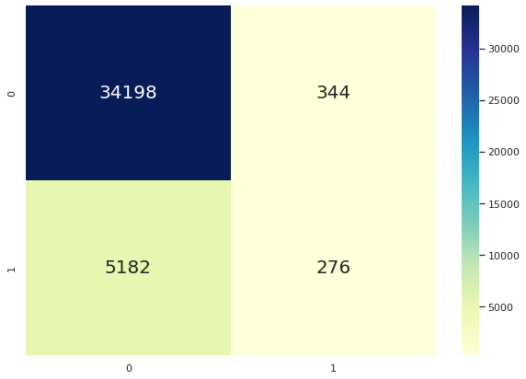
XGBoost

در ابتدا با استفاده از کتابخانه‌ی XGBoost یک classifier روی دادگان آموزش داده شده‌است. داداگان طبق توضحیات بخش قبل تمیز می‌شوند و داده‌های کتگوریکال به صورت عددی به وسیله‌ی sklearn.preprocessing.LabelEncoder انکود می‌شوند. 40 درصد از دادگان را به عنوان داده‌ی تست در نظر و سپس مدل را روی داده‌ی ترین آموزش داده می‌شود.

نتایج این مدل را در ادامه مشاهده می‌کنید.

```
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
visualize_result(y_test,predictions)
```

```
f1_score  is :0.09081934846989141%
recall_score  is :0.050567973616709415%
precision_score  is :0.44516129032258067%
              precision    recall  f1-score   support

     class 0       0.87      0.99      0.93     34542
     class 1       0.45      0.05      0.09      5458

    accuracy                           0.86     40000
   macro avg       0.66      0.52      0.51     40000
weighted avg       0.81      0.86      0.81     40000
```



به وسیله‌ی plot_importance اهمیت هر فیچر نمایش داده می‌شود و بیشترین تاثیر در میان فیچرها برای audience_id و nbclicks است.

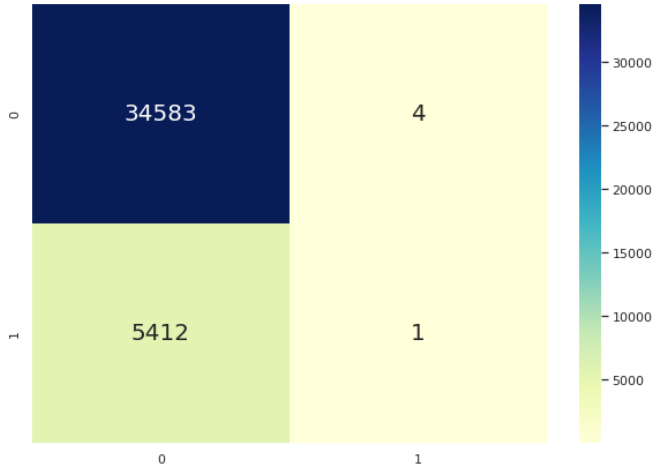
محتویات این مدل و اجرای آن در فایل XGBoost.ipynb قرار دارد.

LogisticRegression

در این بخش از کتابخانه sklearn استفاده کردیم. به وسیله همان دیتاست تمیز شده در مراحل قبل این مدل را فیت کردیم و نتایج آن اصلا قابل قبول نیست و دلیل آن بالانس نبودن دیتاست است.

همانطور که در عکس واضح است مدل به این سمت حرکت کرده است که تمامی خروجی‌ها را صفر بدهد.

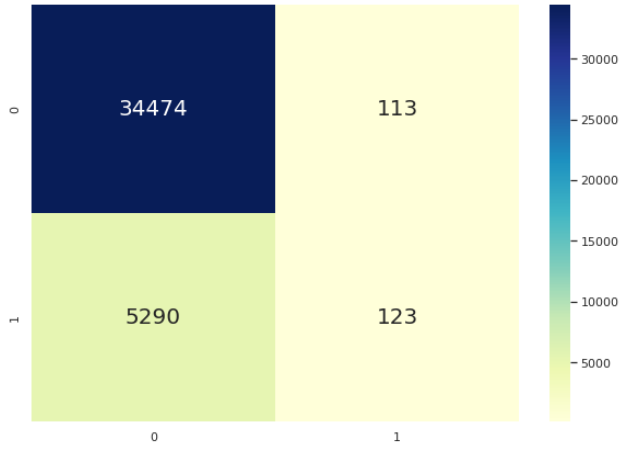
```
f1_score  is :0.00036913990402362 5%
recall_score  is :0.0001847404396822 4645%
precision_score  is :0.2%
              precision    recall  f1-score   support

     class 0       0.86      1.00      0.93     34587
     class 1       0.20      0.00      0.00      5413

    accuracy                           0.86     40000
   macro avg       0.53      0.50      0.46     40000
weighted avg       0.77      0.86      0.80     40000
```



Decision Tree

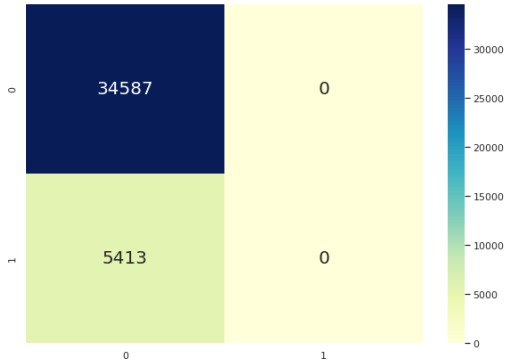
به وسیله‌ی مدل آماده sklearn یک درخت تصمیم به دادگان فیت کردیم که مدل باز به دلیل بالانس نبودن دیتاست مانند بخش قبل نتایج ضعیفی داشت. یکی از راه‌های بهبود مدل‌های غیر شبکه عصبی برای دیتاست‌های غیربالانس استفاده از sklearn.utils.resample است که از کلاس کوچک‌تر داده تولید می‌کند و سعی می‌کند دیتا بالانس‌تر شود.

```
f1_score  is :0.043547530536378116%
recall_score  is :0.02272307408091631%
precision_score  is :0.5211864406779662%
              precision    recall  f1-score   support

     class 0       0.87      1.00      0.93     34587
     class 1       0.52      0.02      0.04      5413

    accuracy                           0.86     40000
   macro avg       0.69      0.51      0.49     40000
weighted avg       0.82      0.86      0.81     40000
```



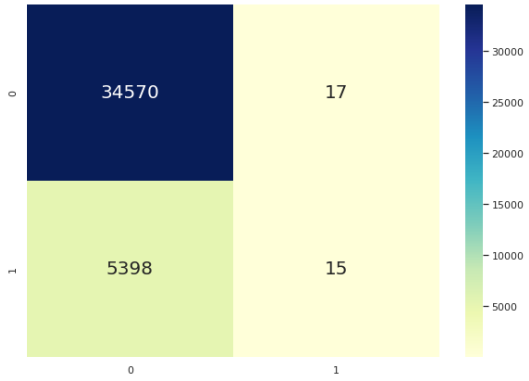
Random Forest

این دسته‌بند نیز همانند دسته‌بندهای قبلی اصلا نتیجه خوبی نداشت و تماما مقدار صفر را خروجی داد.



AdaBoost

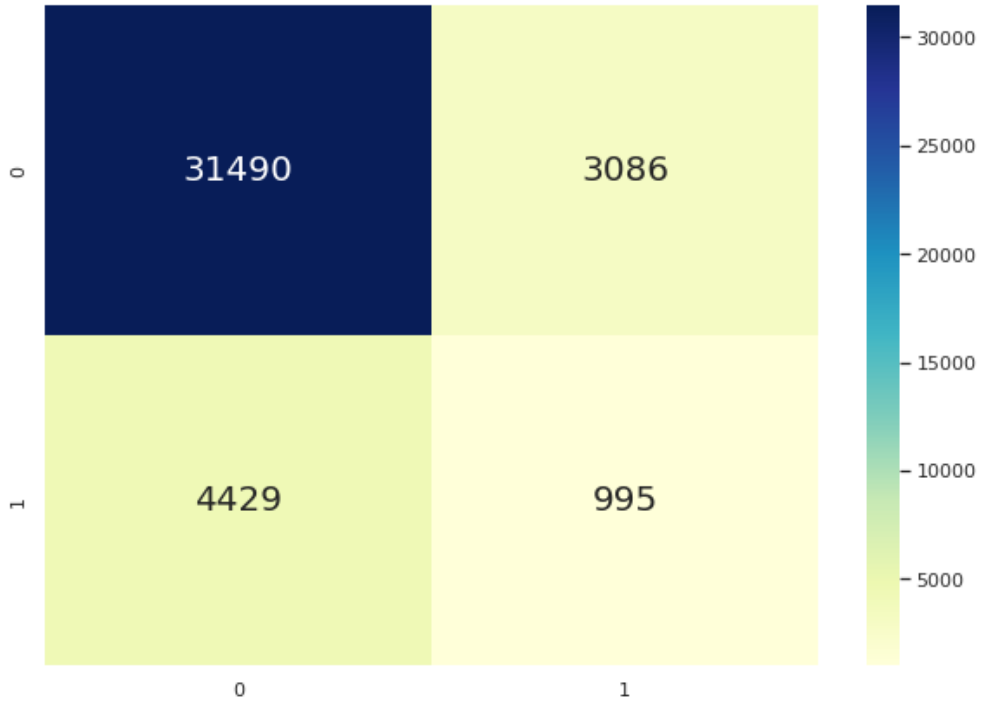
این دسته بند نیز عملکرد مشابه به مدل‌های قبلی دارد و قابل قبول نیست.



Gaussian Naïve Bayes

در میان تمام دسته‌بندهای بالا بهترین نتیجه به وسیله دسته‌بند نیوبیز بدست آمد و دلیل اصلی آن این است که به طور مستقیم احتمال هر کلاس را در نظر می‌گیرد و آن را در خروجی مدل خود تاثیر می‌دهد. همانطور که در تصویر زیر مشاهده می‌کنید مقدار f1 به 0.20 رسیده است که نسبت به سایر مدل‌ها عملکرد بهتری دارد.

```
f1_score  is :0.2093634928984745%
recall_score  is :0.1834439528023599%
precision_score  is :0.2438127909826023%
              precision    recall  f1-score   support

     class 0       0.88      0.91      0.89     34576
     class 1       0.24      0.18      0.21      5424

    accuracy                           0.81     40000
   macro avg       0.56      0.55      0.55     40000
weighted avg       0.79      0.81      0.80     40000
```



تمامی مدل‌های بالا در نوت بوکی به اسم Non Deep Models.ipynb قرار دارد.

در ادامه به بررسی مدل‌های شبکه عصبی می‌پردازیم.

مدل‌های شبکه عصبی

در ابتدا یک مدل ساده‌ی چندلایه شبکه عصبی را به وسیله‌ی pytorch پیاده‌سازی شده‌است. در اینجا برای سادگی داده‌های کتگوریکال به صورت one-hot داده نمی‌شوند. گرچه این کار خوبی نیست ولی برای اجرای اولیه از آن استفاده می‌کنیم.

معماری مدل را در تصویر زیر مشاهده می‌کنید.

```
model = nn.Sequential(
        nn.Linear(19,128),
        nn.ReLU(),
        nn.Linear(128,64),
        nn.ReLU(),
        nn.Linear(64,16),
        nn.ReLU(),
        nn.Linear(16,8),
        nn.ReLU(),
        nn.Linear(8,1),
        nn.Sigmoid()
    )
```

برای loss نیز از کراس انتروپی برای binary classification استفاده کردیم که به صورت آماده در تورچ با نام BCELoss موجود است.

نتیجه‌ای که از این بخش حاصل شد شبکه‌ای بود که تماما صفر خروجی می‌دهد و اصلا ترین نمی‌شود و در یک مینیمم محلی گیر افتاده‌است. بخشی از مراحل ترین را مشاهده می‌کنید که دقت همواره ثابت است و f1 صفر می‌شود.

کد این مدل نیز در فایل NN_Project.ipynb قرار دارد.

```
epoch 13 - train Loss: 8.868e+01 - train Acc: 86.43%: 100%|| 469/469 [00:03<00:
epoch 13 - val Loss: 6.931e-01 - val Acc: 86.20%: 100%|| 313/313 [00:01<00:00,
epoch 14 - train Loss: 8.868e+01 - train Acc: 86.43%: 100%|| 469/469 [00:03<00:
epoch 14 - val Loss: 6.931e-01 - val Acc: 86.20%: 100%|| 313/313 [00:01<00:00,
epoch 15 - train Loss: 8.868e+01 - train Acc: 86.43%: 100%|| 469/469 [00:03<00:
epoch 15 - val Loss: 6.931e-01 - val Acc: 86.20%: 100%|| 313/313 [00:01<00:00,
epoch 16 - train Loss: 8.868e+01 - train Acc: 86.43%: 100%|| 469/469 [00:04<00:
epoch 16 - val Loss: 6.931e-01 - val Acc: 86.20%: 100%|| 313/313 [00:02<00:00,
epoch 17 - train Loss: 8.868e+01 - train Acc: 86.43%: 100%|| 469/469 [00:04<00:
epoch 17 - val Loss: 6.931e-01 - val Acc: 86.20%: 100%|| 313/313 [00:01<00:00,
epoch 18 - train Loss: 8.868e+01 - train Acc: 86.43%: 100%|| 469/469 [00:03<00:
epoch 18 - val Loss: 6.931e-01 - val Acc: 86.20%: 100%|| 313/313 [00:01<00:00,
epoch 19 - train Loss: 8.868e+01 - train Acc: 86.43%: 100%|| 469/469 [00:03<00:
epoch 19 - val Loss: 6.931e-01 - val Acc: 86.20%: 100%|| 313/313 [00:01<00:00,
```

برای رفع مشکل بالا که به دلیل نامتوازن بودن دیتاست رخ داده بود از RandomWeightedSampler استفاده شد که به هر داده وزنی را نسبت که می‌دهد که به تعداد آن در دیتاست بستگی دارد.

بعد از تعریف sampler آن را به DataLoader ورودی می‌دهیم و در حین آموزش شبکه تاثیر وزن را بر ورودی می‌گذارد.

```
from collections import Counter
count=Counter(y_train)
print(count)
class_count=np.array([count[0],count[1]])
print(class_count)
weight=1./class_count
print(weight)
samples_weight = np.array([weight[int(t)] for t in y_train])
samples_weight = torch.from_numpy(samples_weight)
```

```
Counter({0.0: 64699, 1.0: 10301})
[64699 10301]
[1.54561894e-05 9.70779536e-05]
```

وزن داده‌های کلاس 1 تقریبا 6 برابر وزن داده‌های کلاس 0 خواهد شد.

کمی هم مدل را تغییر دادیم و به حالت زیر رسیدیم و خروجی شبکه را دو نورون کردیم و به جای BCELoss از CrossEntropyLoss استفاده کردیم.

```
s_nn = nn.Sequential(
        nn.Linear(17,128),
        nn.LeakyReLU(),
        nn.Linear(128,128),
        nn.LeakyReLU(),
        nn.Linear(128,64),
        nn.LeakyReLU(),
        nn.Linear(64,32),
        nn.LeakyReLU(),
        nn.Linear(32,16),
        nn.LeakyReLU(),
        nn.Linear(16,8),
        nn.Linear(8,2),
        nn.LeakyReLU(),

    )
```

با این تغییر شبکه شروع به یادگیری می‌کند و دقت accuracy به حدود 70 درصد می‌رسد ولی مقدار f1 تقریبا برابر 0.27 می‌شود برای دادگان تست. نتایج مدل را در تصویر زیر مشاهده می‌کنید.

```
from sklearn.metrics import f1_score, confusion_matrix,
print(confusion_matrix(predicted.cpu(), y_test))
print(f1_score(predicted.cpu(), y_test))
```

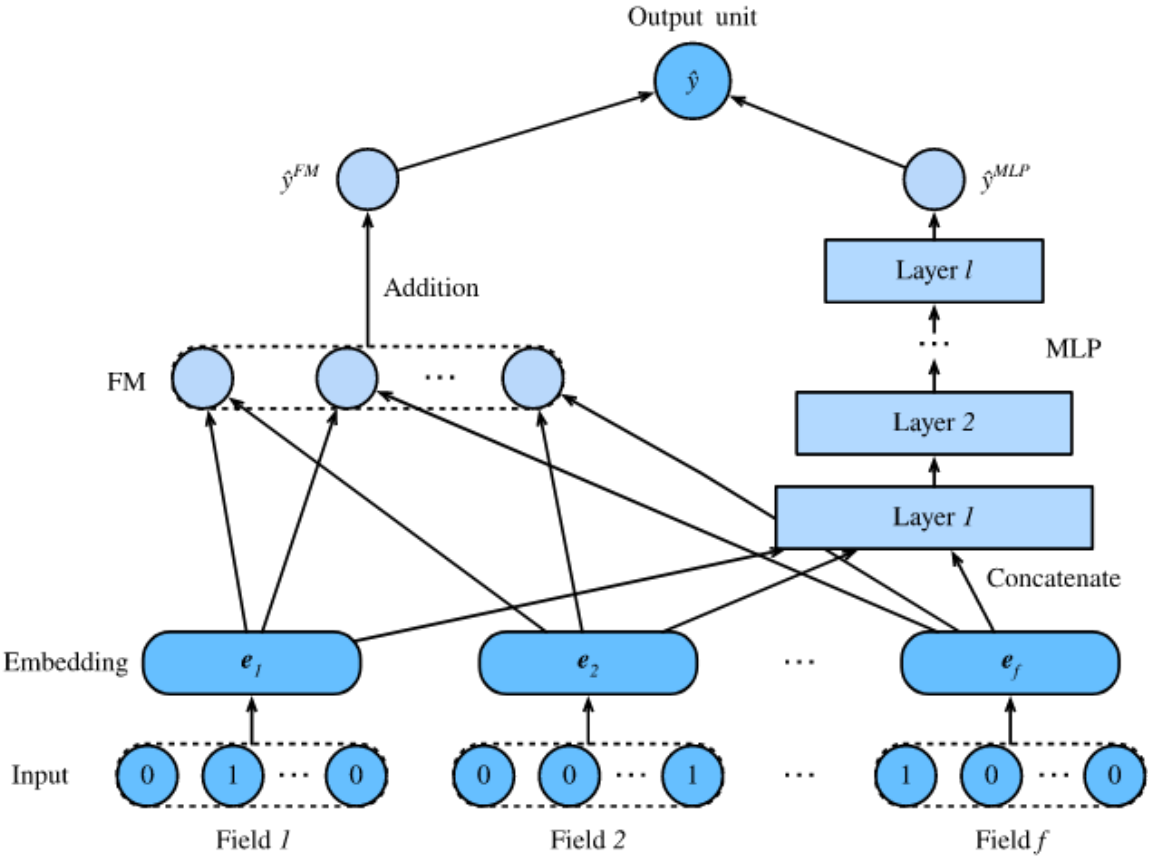
```
[[15665  1903]
 [ 5975  1457]]
0.27001482579688657
```

مدل این بخش نیز در فایل NN_Project_Weighted.ipynb قرار دارد.

در ادامه از مدل‌های DeepFM و Deep and Wide استفاده خواهیم کرد که در انتهای داک معرفی شده بودند.

DeepFM

در ابتدا مدلی پیشنهادی deepFM مورد مطالعه قرار گرفت که معماری آن را در تصویر زیر مشاهده خواهید کرد.



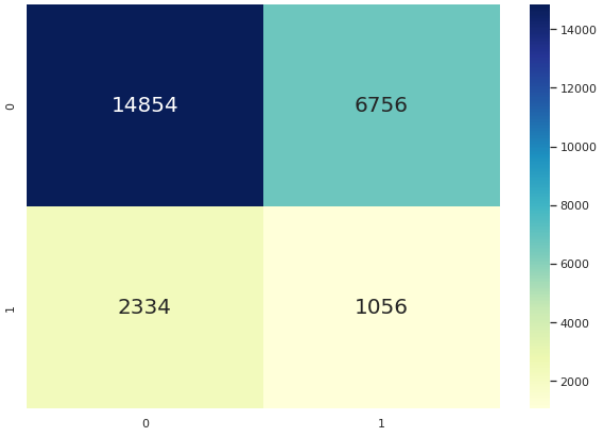
کد این مدل در فایل Depp & Wide.ipynb قرار دارد. برای آموزش این شبکه نیز از RandomWeightedSampler استفاده شده است. کد زیر برگرفته از کد داده شده در لینک مدل‌های پیشنهادی داک پروژه است.

```
class DeepWide(nn.Module):
    def __init__(self,cat_F_dims,emmbeded_hyper,MLP_dims,numerical_dim):
        super(DeepWide,self).__init__()
        cat_dims = sum(cat_F_dims)
        self.embeding = nn.Embedding(cat_dims, emmbeded_hyper)
        input_dims = numerical_dim + len(cat_F_dims) * emmbeded_hyper
        self.FXlayer = nn.Sequential(nn.Linear(input_dims, 1),nn.ReLU())
        modules = []
        for output in MLP_dims:
            modules.append(nn.Linear(input_dims, output))
            modules.append(nn.ReLU())
            modules.append(nn.Dropout(0.1))
            input_dims = output
        self.MLP = nn.Sequential(*modules)
        self.Flatten = nn.Flatten()
        self.sigmoid = nn.Sigmoid()
    def forward(self,x1,x2):
        embedded_output = torch.Tensor(self.embeding(x1.to(torch.int64)))
        square_of_sum = torch.sum(embedded_output, axis=1) ** 2
        sum_of_square = torch.sum(embedded_output ** 2, axis=1)
        embedded_output2 = self.Flatten(embedded_output)
        cated_input = torch.cat((embedded_output2, x2), -1 )
        z = self.FXlayer(cated_input)
        z+=self.MLP(cated_input)
        z+=0.5 * (square_of_sum - sum_of_square).sum(1, keepdims=True)
        return self.sigmoid(z).reshape(-1)
```

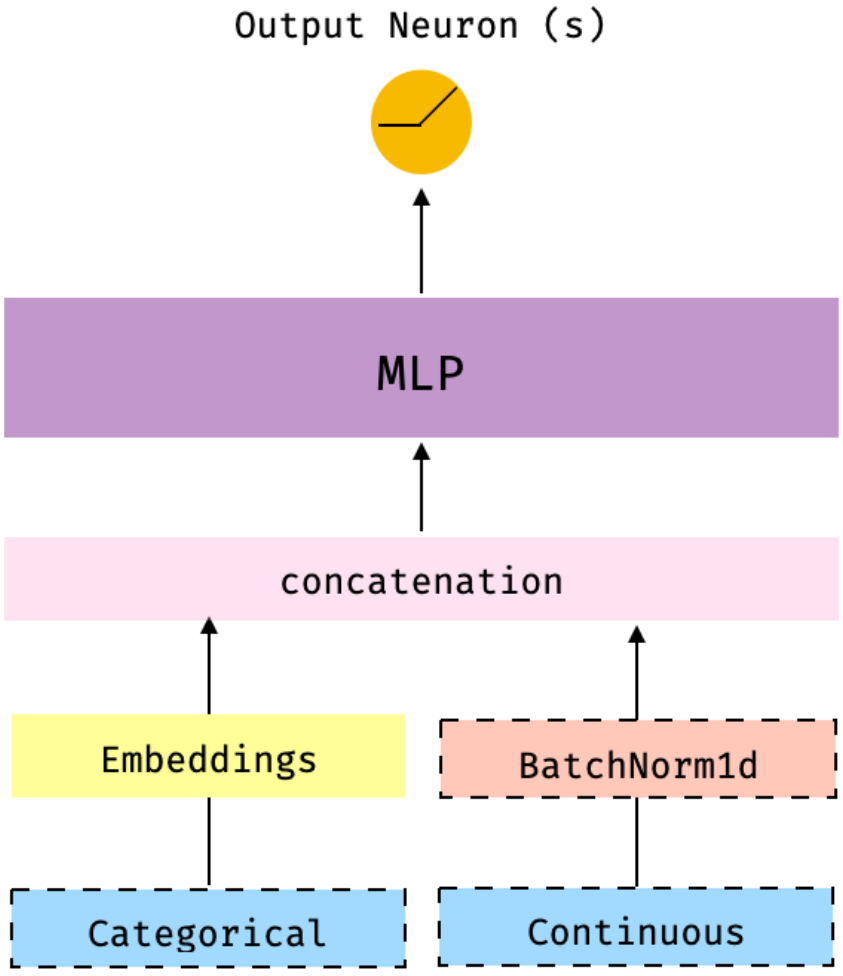
نتایج مدل بعد از 40 ایپاک ترینیگ به شرح زیر می‌باشد:

```
epoch 35 - train Loss: 4.039e+02 - train Acc: 63.39%: 100%|| 293/293 [00:21<00:
epoch 35 - val Loss: 1.579e+00 - val Acc: 63.74%: 100%|| 98/98 [00:02<00:00, 33
epoch 36 - train Loss: 3.966e+02 - train Acc: 63.52%: 100%|| 293/293 [00:21<00:
epoch 36 - val Loss: 1.555e+00 - val Acc: 63.36%: 100%|| 98/98 [00:02<00:00, 47
epoch 37 - train Loss: 3.894e+02 - train Acc: 63.64%: 100%|| 293/293 [00:22<00:
epoch 37 - val Loss: 1.532e+00 - val Acc: 63.50%: 100%|| 98/98 [00:02<00:00, 41
epoch 38 - train Loss: 3.836e+02 - train Acc: 63.75%: 100%|| 293/293 [00:20<00:
epoch 38 - val Loss: 1.512e+00 - val Acc: 63.55%: 100%|| 98/98 [00:03<00:00, 31
epoch 39 - train Loss: 3.784e+02 - train Acc: 63.87%: 100%|| 293/293 [00:23<00:
epoch 39 - val Loss: 1.494e+00 - val Acc: 63.64%: 100%|| 98/98 [00:02<00:00, 46
```

```
f1_score  is :0.1885377611140868%
recall_score  is :0.31150442477876106%
precision_score  is :0.13517665130568357%
              precision    recall  f1-score   support

     class 0       0.86      0.69      0.77     21610
     class 1       0.14      0.31      0.19      3390

    accuracy                           0.64     25000
   macro avg       0.50      0.50      0.48     25000
weighted avg       0.77      0.64      0.69     25000
```



در نهایت مدل نهایی که بسیار شبیه به مدل بالا است با این تفاوت که برای هر فیچر یک embedding درنظر می‌گیرد را مشاهده می‌کنید.



TabMlp

از تفاوت‌های این مدل و مدل قبلی باید به این اشاره کرد که در اینجا برای هر فیچر که به صورت کتگوریکال است یک nn.Embedding قرار می‌دهیم که ابعاد ورودی آن تعداد بعد (فیچر+1) خواهد بود و به بعد خروجی یک هایپر پارامتر است که برای داده‌هایی که تعداد مقادیر یکتای کمی دارند بعد خروجی همان بعد ورودی می‌گذاریم ولی اگر بعد فیچر بزرگتر از 50 بود بعد خروجی را 50 در نظر می‌گیریم تا از بزرگ شدن ورودی لایه MLP جلوگیری کنیم.

از دیگر تفاوت‌های این مدل وجود لایه‌ی BatchNorm است که در مدل قبلی وجود نداشت.

در این مدل نام ستون‌های کتگوریکال و نام ستون‌های عددی همچنین بعد هر ستون کتگوریکال ورودی داده می‌شود. در ضمن یک دیکشنری از نام ستون به شماره نمایه آن در ورودی باید در تعریف مدل ورودی داده شود.

ابعاد لایه MLP نیز به صورت یک لیست ورودی داده می‌شود همچنین می‌توان میزان احتمال dropout در انتهای هر لایه‌ی MLP را نیز ورودی داد.

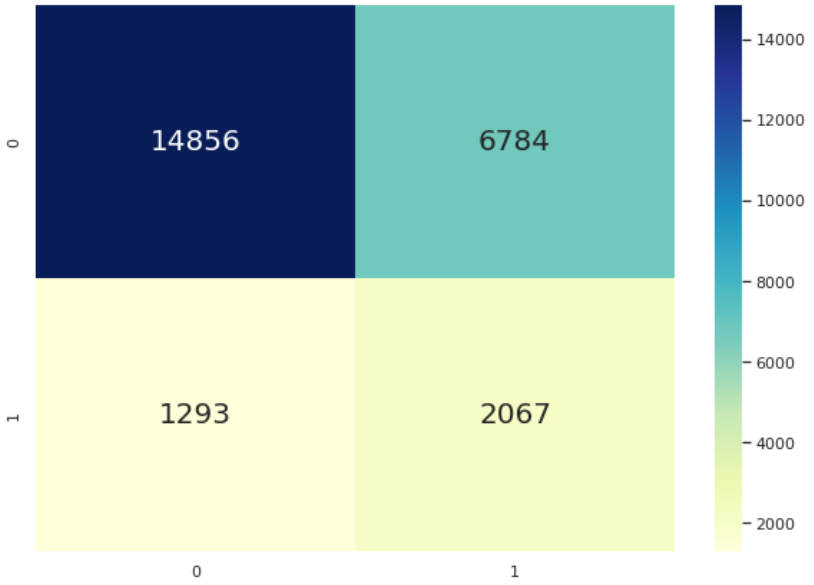
پارامترهای ورودی مدل در تصویر زیر قابل مشاهده است.

```
model = MD.TabMlp(
    mlp_hidden_dims=[300, 200, 100],
    column_idx=column_idx,
    embed_input=embeddings_input,
    mlp_dropout=[0.2,0.2,0.2],
    continuous_cols=cont_cols,

    mlp_batchnorm=True,
    pred_dim = 2,
)
```

سپس شروع به آموزش این مدل می‌کنیم. برای learning rate از 0.01 شروع می‌کنیم و با Scheduler بعد از هر 10 ایپاک آن را 0.1 می‌کنیم.

نتایج به شرح زیر است که f1 برابر با 0.338 است که از تمامی مدل‌های قبلی بهتر است.

```
f1_score  is :0.33854721153058714%
recall_score  is :0.6151785714285715%
precision_score  is :0.23353293413173654%
              precision    recall  f1-score   support

     class 0       0.92      0.69      0.79     21640
     class 1       0.23      0.62      0.34      3360

    accuracy                           0.68     25000
   macro avg       0.58      0.65      0.56     25000
weighted avg       0.83      0.68      0.73     25000
```



حال به بررسی اهمیت هر فیچر می‌پردازیم به این صورت که هر بار یک فیچر را حذف کردیم و به جای آن صفر قرار دادیم و عملکرد مدل را روی دیتاست تست بررسی کردیم و عملکرد مدل با حذف هر فیچر در تصویر زیر مشخص است.

همانطور که در تصویر مشخص است partner_id مهمترین فیچر و تاثیرگذارترین است. یعنی هرچقدر مدل دقت کمتری بگیرد اهمیت فیچری که حذف شده است بالاتر بوده است.

```
product_age_group : 0.3391403451644415
device_type : 0.3107904642409034
partner_id : 0.28240091754810753
audience_id : 0.32421909565525175
product_gender : 0.33806515139082627
product_category(1) : 0.3396414342629482
product_country : 0.33819195539625646
day_time_category : 0.34256375725948995
nb_clicks_1week : 0.33776091081593923
```

تمامی کدهای مربوط به کلاس TabMLP در فایل TabMLP.ipynb قرار دارد.

Deployment

ابتدا مدل نهایی به وسیله‌ی mlflow با گرفتن runهای مختلف با لاگ کردن پارامترهای مختلف بهترین مقدار هر پارامتر را انتخاب می‌کنیم.

							Metrics			
	Start Time	Duration	Run Name	User	Source	Version	Models	acc	acct	↓ f1	f
	20 hours ago	8.9min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.672	-	0.345	-
	17 hours ago	6.6min	-	amirhoosein	trainDEP.p	f98ddf	pytorch	0.667	0.698	0.342	0
	18 hours ago	4.5min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.671	0.703	0.341	0
	19 hours ago	6.4min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.668	0.699	0.341	0
	17 hours ago	4.9min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.664	0.697	0.341	0
	19 hours ago	6.4min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.673	0.706	0.339	0
	18 hours ago	5.3min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.679	0.728	0.336	0
	20 hours ago	5.2min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.679	-	0.335	-
	12 hours ago	7.3min	-	amirhoosein	trainDEP.p	8e2879	pytorch	0.674	0.718	0.335	0
	19 hours ago	8.5min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.677	0.725	0.335	0
	16 hours ago	7.4min	-	amirhoosein	trainDEP.p	f98ddf	pytorch	0.668	0.709	0.335	0
	20 hours ago	12.7s	-	amirhoosein	trainDEP.p	50d569	pytorch	0.643	-	0.335	-
	19 hours ago	6.9min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.671	0.718	0.333	0
	17 hours ago	4.3min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.674	0.725	0.331	0
	20 hours ago	7.5min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.657	0.706	0.331	0
	17 hours ago	5.8min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.681	0.74	0.327	0
	17 hours ago	5.0min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.668	0.724	0.325	0
	18 hours ago	5.8min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.671	0.736	0.324	0
	18 hours ago	6.0min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.692	0.763	0.323	0
	20 hours ago	5.9min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.721	0.737	0.315	0
	18 hours ago	3.8min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.489	0.518	0.302	0
	18 hours ago	7.1min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.729	0.822	0.297	0
	18 hours ago	7.0min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.717	0.812	0.291	0
	20 hours ago	9.5min	-	amirhoosein	trainDEP.p	50d569	pytorch	0.134	0.137	0.237	0

برای دیپلوی کردن مدل در کامپیوتر شخصی از mlflow.pyfunc استفاده می‌کنیم که دارای دو تابع load_context و predict است که باید در کلاس فرزند پیاده‌سازی شوند.

برای این منظور یک کلاس به نام WrapperModule پیاده‌سازی کردیم.

سپس مدل wrapper را saveو log کردیم تا بتوان برای تست آن را serve کرد.

توجه کنید که فایل‌های مورد نیاز برای انجام عملیات preprocess مانند column_idx به صورت فایل json ذخیره شده اند و ورودی مدل wrapper هستند.

در تصویر زیر کد پایپلاین preprocess و test کردن مدل را مشاهده می‌کنید.

```python
class ModelWrapper(mlflow.pyfunc.PythonModel):

  def load_context(self, context):
    import DWModels as MD
    import Preprocess as ps
    import torch
    self._p = ps.preprocess()

    # Load in and deserialize the embeddings
    print(context.artifacts)
    with open(context.artifacts["column_idx"], 'rb') as handle:
      self._column_idx = pickle.load(handle)

    # load in and deserialize the model tokenizer
    with open(context.artifacts["embeddings_input"], 'rb') as handle:
      self._embeddings_input = pickle.load(handle)

    with open(context.artifacts["cont_cols"], 'rb') as handle:
      self._cont_cols = pickle.load(handle)

    model = MD.TabMlp(
    mlp_hidden_dims=[500,400,300,200, 100, 100],
    column_idx=self._column_idx,
    embed_input=self._embeddings_input,
    mlp_dropout=[0.2,0.3,0.2,0.2,0.2,0.2],
    continuous_cols=self._cont_cols,
    mlp_batchnorm=True,
    pred_dim = 2)

    self._model = model
    self._model.load_state_dict(torch.load(context.artifacts["state_dict_model"]))
    self._model.eval()

  def predict(self, context, input_model):
    input_m = torch.Tensor(self._p.prepro_test(input_model))
    output = self._model(input_m)
    predicted = torch.max(output.data,1)[1]
    return predicted.numpy()
```

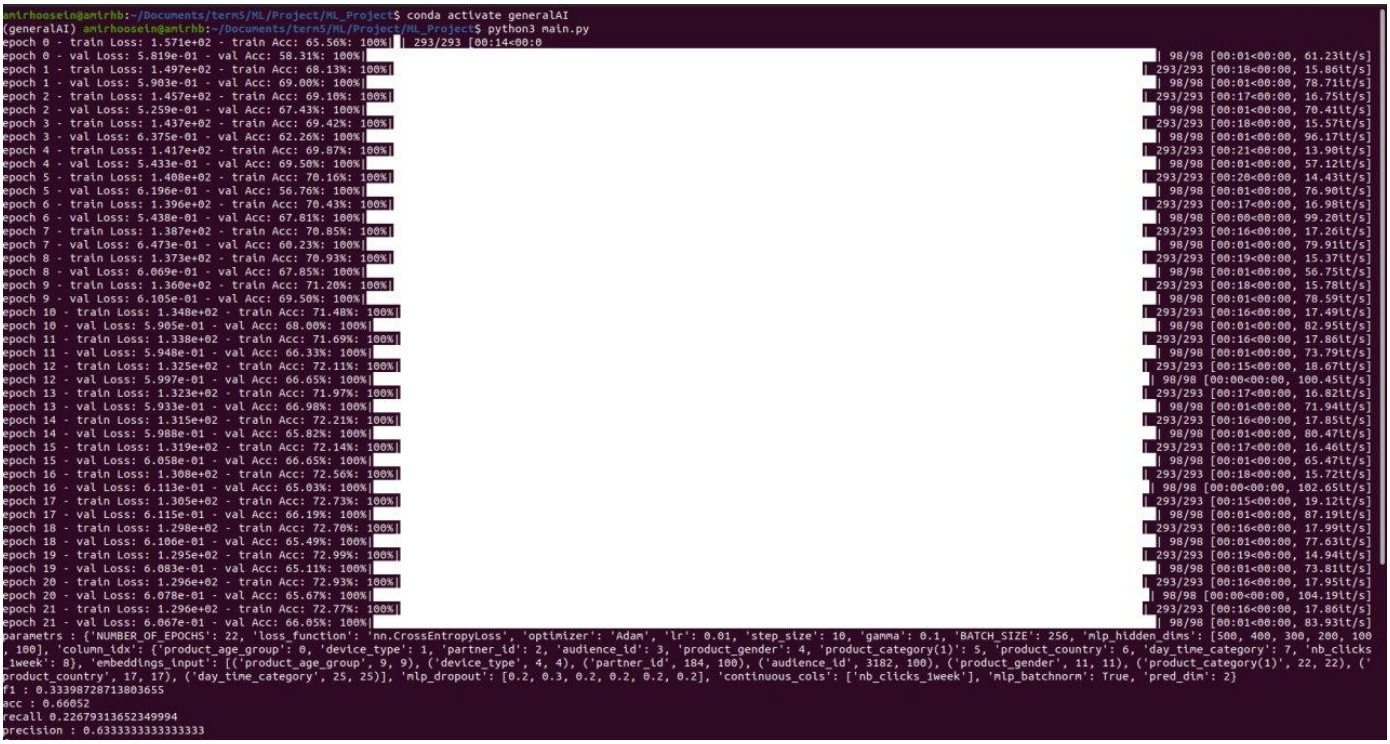
مدل ابتدا عملیات preprocess را بر روی داده‌ی input انجام می‌دهد و سپس آن را به مدل شبکه عصبی که وزن‌های آن در state_dict_model لود شده است می‌دهد و خروجی را که تنسورهای دوتایی از احتمال هر کلاس است را دریافت و کلاس با احتمال بیشتر را خروجی می‌دهد.

فرآیند تست کردن مدل

برای تست کردن مدل فایل main.py را اجرا کنید. از آنجا که artifact آدرس مطلق را خودش ذخیره می‌کند برای اولین ران نیاز است تا مدل یک بار ترین شود و این فرآیند قبل از serve کردن مدل به صورت خودکار انجام می‌شود. بنابراین اگر بار اول مدل را اجرا کنید باید چند دقیقه صبر کنید تا مدل ترین شود که بخش ترین نیز توسط mlflow بدون انجام کاری توسط شما صورت می‌گیرد و تنها باید چند دقیقه صبر کرد. البته ما مدل را برای روی کامپیوتر شخصی و بر روی cpu آن را آموزش دادیم.

اگر config کدها با کامپیوتر شما سازگاری نداشت باید config کد را به صورت دستی تغییر دهید.

در تصاویر زیر شما یک نمونه از ران کردن فایل main برای بار اول را مشاهده می‌کنید.



در تصویر زیر یک نمونه از ران کردن فایل main پس از یک بار ترین شدن را مشاهده می‌کنید. در این تصویر مدل یک بار قبلا train و ذخیره شده است و در این مرحله serve می‌شود و آماده برای دریافت request و خروجی دادن می‌شود.

در این بخش برای تست کردن مدل serve شده یک نمونه 10000 تایی از دیتاست داده شده سمپل کردیم و دیتای پیش-پردازش نشده را به وسیله‌ی request در پایتون به APIی دیپلوی شده ارسال کردیم و خروجی که یک لیست 10000 تایی از کلاس‌های خروجی را دریافت کردیم و به وسیله target مقدار f1_score را محاسبه کردیم. این مقدار در تست برابر 0.40 است که علت زیاد شدن آن نسبت تست‌های قبلی این است که بخشی از داده‌ی آموزش درهنگام سمپل کردن آورده شده است.

اگر پورت اشغال بود با دستور زیر آن را آزاد کنید:

fuser -i -TERM -k 8080/tcp

توجه کنید که f1_score ادعا شده همان 0.33 است و در بخش‌های قبلی مدل هرگز داده‌ی تست را مشاهده نمی‌کند.

```
host = '127.0.0.1'

port = 8080

url = f'http://{host}:{port}/invocations'

headers = {
    'Content-Type': 'application/json',
}

data = pd.read_csv('PreModule/datasets/train_dataset.csv')
d = data.to_numpy()
for i in range(20):
    d = shuffle(d, random_state=i)
test_data = pd.DataFrame()
test_data[data.columns] = d[1:20000]
target = test_data["Sale"].to_numpy()

http_data = test_data.to_json(orient='split')

r = requests.post(url=url, headers=headers, data=http_data)
predict = r.text
predict = predict.replace("[","")
predict = predict.replace("]","")
l = predict.split(",")
pred = []
for ind in l:
    pred.append(int(ind))
visualize_result(target.astype(int),pred)
```

```
amirhoosein@amirhb:~/Documents/term5/ML/Project/ML_Project$ conda activate generalAI
(generalAI) amirhoosein@amirhb:~/Documents/term5/ML/Project/ML_Project$ python3 request_test.py
f1_score  is :0.40472730795311435%
recall_score  is :0.76464128843338 21%
precision_score  is :0.275194309050191%
              precision    recall  f1-score   support

     class 0       0.95      0.68      0.79     17267
     class 1       0.28      0.76      0.40      2732

    accuracy                           0.69     19999
   macro avg       0.61      0.72      0.60     19999
weighted avg       0.86      0.69      0.74     19999

(generalAI) amirhoosein@amirhb:~/Documents/term5/ML/Project/ML_Project$
```

منابع

https://github.com/jonad/pytorch_mlflow/blob/master/textclassification_with_mlflow.ipynb

https://towardsdatascience.com/pytorch-widedeep-deep-learning-for-tabular-data-9cd1c48eb40d

https://d2l.ai/chapter_recommender-systems/deepfm.html

https://github.com/jrzaurin/pytorch-widedeep