بسمه تعالی

دانشکده مهندسی کامپیوتر

داده کاوی

نام استاد: دکتر حسین رحمانی

پروژه دوم

آرمان حیدری

شماره دانشجویی: ۹۷۵۲۱۲۵۲

فروردین ۱۴۰۱

فهرست

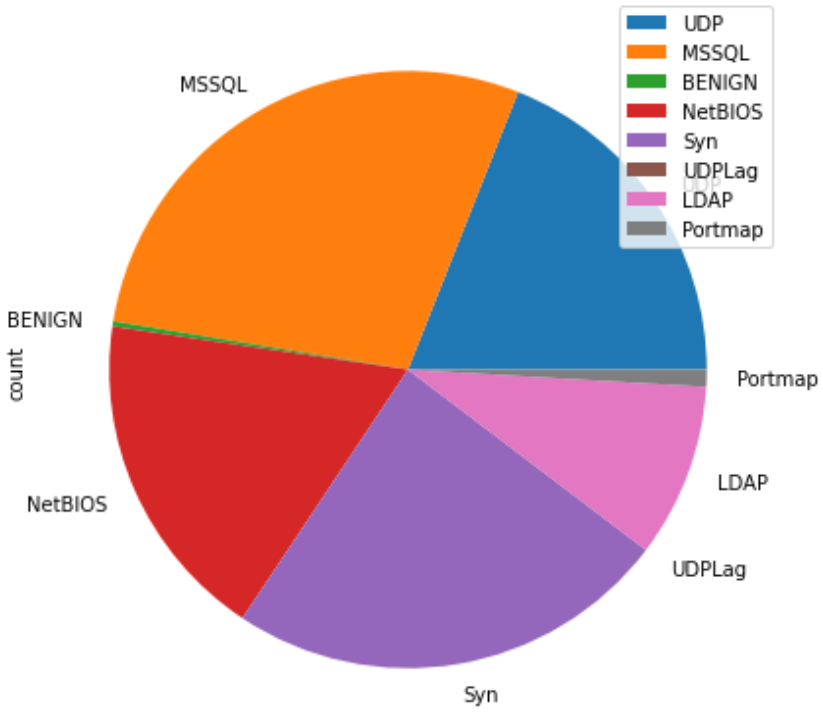
۱. بارگذاری داده و نمونه برداری

ابتدا داده ها را از url داده شده دانلود میکنیم و از حالت zip خارج میکنیم. سپس مسیر مربوط به ۷ فایل csv. را در یک آرایه نگه میداریم تا بتوانیم آن ها را بخوانیم. اگر همه را در ابتدا بارگذاری کنیم به دلیل حجم زیاد داده ها دچار مشکل کمبود ram میشویم. این تمام فایل ها و داده های ماست:

```
UDP.csv founded. it has 3782207 records.
NetBIOS.csv founded. it has 3455900 records.
UDPLag.csv founded. it has 725166 records.
LDAP.csv founded. it has 2113235 records.
Portmap.csv founded. it has 191695 records.
MSSQL.csv founded. it has 5775787 records.
Syn.csv founded. it has 4320542 records.
```

در این پروژه به دلیل زیاد بودن حجم دیتا (حدودا ۲ گیگ داده خام) نیاز به نمونه برداری داریم. در کلاس روش های مختلف آن یعنی random sampling و stratified sampling را خواندیم که دومی نسبت کلاس ها را در نمونه برداری نگه میداشت.

مجموعه داده ما بسیار imbalanced است. در ابتدا با chunk های 1000000 تایی داده ها را از فایل ها خوانده ایم و کلاس های آن ها میشماریم و با هم جمع میزنیم تا ببینیم در کل چه داده ای در اختیار داریم. به چنین نموداری بین توزیع کلاس های مختلف آن میرسیم:



که میبینیم حالت BENIGN که در واقع کلاس ۰ ماست داده های خیلی خیلی کمتری از حالت کلاس ۱ ما دارد. به همین خاطر نوع خاصی از نمونه برداری را در تابع Sampling خود پیاده سازی کردم.

این تابع به عنوان ورودی یک dataframe میگیرد و تمام سطرهایی که کلاس BENIGN دارند را انتخاب میکند. هدف از این تابع balance کردن دیتاست است، پس یک threshold برای آن تعریف کرده ایم که مشخص میکند از هر نوع حمله حداکثر چقدر نمونه برداری کند. و این نمونه برداری را به صورت رندوم انجام می دهد.

نکته: برای مسئله تشخیص حمله، این threshold را یک ششم تعداد BENIGN ها قرار داده ایم که مجموع حملات مختلف برابر موارد بدون حمله باشد و از همه حملات نمونه داشته باشیم. ولی برای مسئله تعیین نوع حمله به داده بیشتری نیاز داشتیم و این threshold را متفاوت انتخاب میکنیم که جلوتر توضیح میدهیم.

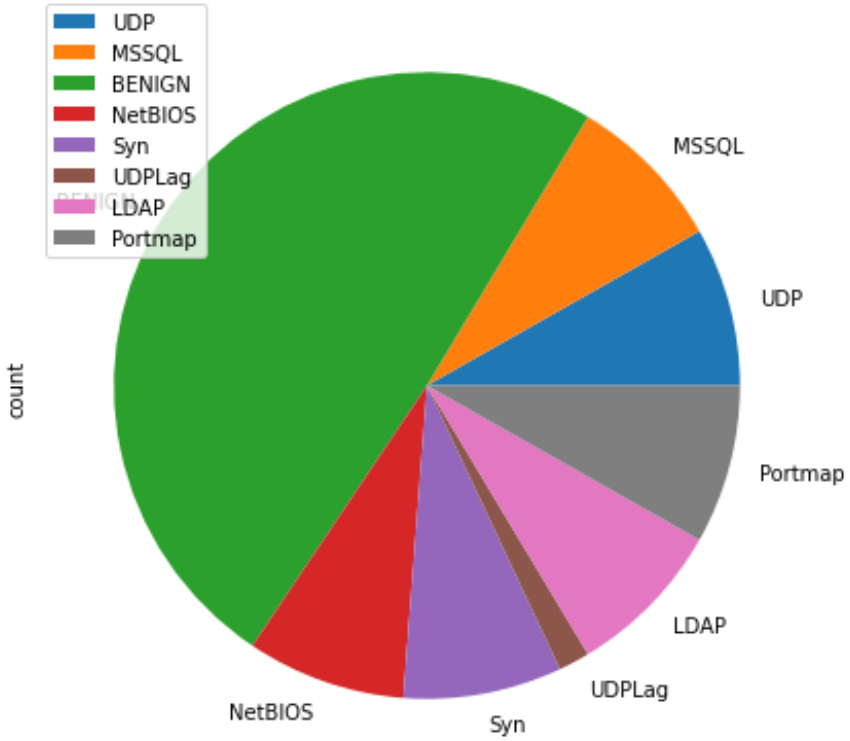
برای اجرای این تابع، مجددا تک تک فایل های csv را میخوانیم و در chunk هایی به طول 10**6، dataframe میسازیم و به تابع sampling پاس میدهیم. البته یک دیکشنری هم داریم که نشان میدهد تاکنون از هر کلاس چقدر انتخاب کرده ایم و تابع کنترل میکند که مقدار انتخاب هایمان از threshold بیشتر نشود.

با همه ی این ملاحظات به چنین مجموعه داده ای میرسیم که مناسب شروع کار است:

	Unnamed: 0	Flow ID	Source IP	Source Port	Destination IP	Destination Port	Protocol	Timestamp	Flow Duration	Total Fwd Packets	...	Active Std	Active Max	Active Min	Idle Mean	Idle Std	Idle Max	I
0	107045	172.16.0.5-192.168.50.4-52672-12336-17	172.16.0.5	52672	192.168.50.4	12336	17	2018-11-03 10:54:04.137085	1	2	...	0.0	0.0	0.0	0.0	0.0	0.0	
1	29735	172.16.0.5-192.168.50.4-34556-28115-17	172.16.0.5	34556	192.168.50.4	28115	17	2018-11-03 10:55:50.580705	1	2	...	0.0	0.0	0.0	0.0	0.0	0.0	
2	35771	172.16.0.5-192.168.50.4-59385-33778-17	172.16.0.5	59385	192.168.50.4	33778	17	2018-11-03 10:55:34.903826	1	2	...	0.0	0.0	0.0	0.0	0.0	0.0	
3	94473	172.16.0.5-192.168.50.4-36880-33621-17	172.16.0.5	36880	192.168.50.4	33621	17	2018-11-03 10:55:56.837498	214006	6	...	0.0	0.0	0.0	0.0	0.0	0.0	
4	34252	172.16.0.5-192.168.50.4-38146-20840-17	172.16.0.5	38146	192.168.50.4	20840	17	2018-11-03 10:54:17.081870	1	2	...	0.0	0.0	0.0	0.0	0.0	0.0	

که 115802 سطر و 88 ستون دارد.

و توزیع کلاس های آن به این صورت هستند که به خوبی balance است:



نکته: از نوع حمله UDPLag داده های بسیار کمی داریم. و اگر آن را ملاک میگذاشتیم باید داده کمی انتخاب میکردیم و به نتیجه خوبی نمیرسیدیم. پس تمام رکوردهای این کلاس را هم نمونه برداری میکنیم.

در ابتدای این پروژه تحلیل ها را بدون نمونه برداری انجام دادم و مدل ها کاملا روی کلاس های پر تکرار overfit میشدند!

۲. پیش پردازش داده

حذف null

گام اول حذف داده هایی خالی یا Nan است. که ابتدا بررسی میکنیم و میبینیم مقدار آن بسیار کم است و فقط یک ستون شامل ۴۲ رکورد است. پس با حذف رکورد ها چیزی از دست نمیدهیم و آن را حذف میکنیم.

```
null_columns=sampled_df.columns[sampled_df.isna().any()]
sampled_df[null_columns].isna().sum().sort_values(ascending=False)
```

```
Flow Bytes/s    42
dtype: int64
```

```
print("before drop nulls:", sampled_df.shape)
preprocessed_df = sampled_df.dropna()
print("after drop nulls:", preprocessed_df.shape)
```

```
before drop nulls: (115802, 88)
after drop nulls: (115760, 88)
```

در صورت پروژه تشخیص outlier در این بخش خواسته شده اما چون پس از نرمال سازی راحت تر است، در گام پنجم این مار را انجام داده ام.

حذف ستون‌ها

برخی ستون ها دارای مقادیر کاملا متمایز هستند مانند timestamp و idها، که به خاطر آنتروپی زیاد مدلی مانند درخت تصمیم را دچار مشکل کرده و از لحاظ منطقی هم ارزشی در تحلیلمان ندارند.

و برخی هم که آدرس های پورت یا پروتکل http هستند و در تحلیل اهمیتی ندارند. حتی چون مقادیر غیر عددی دارند(این موضوع را با بررسی ستون های با dtype = object هندل کردم) مدل ها را دچار مشکل هم میکنند. پس این ستون ها را حذف کرده ایم:

```
preprocessed_df = preprocessed_df.drop(columns=[
                                                ' Timestamp',
                                                'Flow ID',
                                                'Unnamed: 0',
                                                ' Destination Port',
                                                ' Source Port',
                                                ' Source IP',
                                                ' Destination IP',
                                                'SimillarHTTP',
])
```

همچنین ستون هایی داریم که فقط مقدار ۰ را دارند، در واقع تک مقداره هستند. آن ها را با کمک تابع dataframe.nunique() پیدا کرده ایم و حدود ۱۲ ستون را حذف میکنیم:

```
preprocessed_df.nunique().sort_values(ascending=True).head(15)
```

```
 Bwd PSH Flags           1
Bwd Avg Bulk Rate        1
 Bwd Avg Packets/Bulk    1
 Bwd Avg Bytes/Bulk      1
 Fwd Avg Bulk Rate       1
 Fwd Avg Packets/Bulk    1
 Bwd URG Flags           1
Fwd Avg Bytes/Bulk       1
 ECE Flag Count          1
 Fwd URG Flags           1
FIN Flag Count           1
 PSH Flag Count          1
 Inbound                 2
 SYN Flag Count          2
 CWE Flag Count          2
dtype: int64
```

```
temp = preprocessed_df.nunique()
single_value_columns = [c for c in preprocessed_df.columns if temp[c] == 1]
preprocessed_df = preprocessed_df.drop(columns=single_value_columns)
```

پس از حذف این ها به ۶۷ ستون (ویژگی) میرسیم.

جایگذاری برچسب با اعداد

برای مسئله اصلی، تمام برچسب ها به جز BENIGN را با ۱ و برچسب های BENIGN که فاقد حمله است را با ۰ نمایش میدهیم.

همچنین در انتها برای بخش امتیازی، هر کدام از حملات را با عددی مشخص کرده ایم و از ۱ تا ۷ آن ها را نامگذاری کرده ایم.

```
preprocessed_df[' Label'] = preprocessed_df[' Label'].replace(
    ['NetBIOS', 'BENIGN', 'MSSQL', 'LDAP', 'Portmap', 'Syn', 'UDP','UDPLag'],
     [1, 0, 1, 1, 1, 1, 1, 1])
```

```
preprocessed_df.dtypes.unique()
```

```
array([dtype('int64'), dtype('float64')], dtype=object)
```

پس از این مرحله میبینیم که به کل داده های عددی داریم و همگی float با int هستند.

نرمال سازی

برای نرمال کردن داده ها با توجه به تاکید همیشگی استاد درس، از z_score استفاده کرده ام. نرمال سازی z_score به فرآیند نرمال سازی هر مقدار در یک مجموعه داده اشاره دارد به طوری که میانگین همه مقادیر ۰ و انحراف استاندارد ۱ باشد. این به مدیر داده اجازه می دهد تا احتمال وقوع یک امتیاز در توزیع عادی داده ها را درک کند. z-score یک مدیر داده را قادر می‌سازد تا دو امتیاز مختلف را که از توزیع‌های نرمال متفاوت داده‌ها هستند، مقایسه کند.

در اینجا تابعی که پیاده سازی کرده ایم یک dataframe را میگیرد، تمام ستون های iterate میکند و فرمول این نرمال سازی را برروی هرکدام از ستون ها اعمال می کند. در نهایت dataframe جدید را باز می گرداند.

$$Z = \frac{x - \mu}{\sigma}$$

نکته: وقتی این نرمال سازی را پیاده کردم، دچار مشکلی به دلیل inf بودن برخی میانگین ها و nan بودن انحراف معیارها شدم. که با بررسی متوجه شدم به دلیل وجود چندین مقادیر بی نهایت در جدول است. به جای این مقادیر، حداکثر مقدار ممکن در ستون خودشان را قرار میدهیم و در نتیجه مشکل حل می شود. البته این مرحله را باید قبل از نرمال سازی انجام میدادم که دچار مشکل در فرمول نشویم. ستون هایی که این مقادیر را داشتند به همراه مقدار ماکسیمم آن ها میبینم:

```
for c in list(preprocessed_df.columns):
  temp = np.isinf(preprocessed_df[c]).values.sum()
  if temp>0:
    print(c, temp)
```

```
Flow Bytes/s 3089
 Flow Packets/s 3089
```

```
infinite_columns = ['Flow Bytes/s', ' Flow Packets/s']
for c in infinite_columns:
  temp = preprocessed_df[preprocessed_df[c] != float('inf')]
  c_max = np.max(temp[c])
  print(c_max)
  preprocessed_df.loc[preprocessed_df[c] == float('inf'), c] = c_max
```

```
2944000000.0
3000000.0
```

پس از حل این مشکل و نرمال کردن داده ها با تابع، میتوانیم داده های outlier را هم به راحتی پیدا کنیم. طبق نرمال سازی z_score، معمولا حدود ۹۷ درصد مقادیر موجود در جدول، بین −۳ تا +۳ قرار میگیرند. من اعداد بالای ۱۰ یا کمتر از −۱۰ را به عنوان outlier در نظر گرفتم چون کاملا از حدود ستون خود خارج هستند.

در اینجا حدود ۸۰۰ داده به این شکل داریم که حداقل در یک سطر خود outlier هستند و آن ها را حذف میکنیم.

	Protocol	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std
20761	-1.10926	0.120830	0.102787	0.064357	4.315936	-0.011854	5.450687	-0.588020	2.011868	11.588028
21508	-1.10926	0.140899	0.102787	0.026585	5.113788	-0.011918	5.450687	-0.588020	2.469219	10.289292
21657	-1.10926	0.147228	0.176953	0.102129	7.789020	-0.002555	6.019650	-0.588020	2.711159	11.473568
21671	-1.10926	-0.060328	0.028622	0.064357	2.880332	-0.016409	6.351369	-0.588020	2.336364	14.008778
21713	-1.10926	-0.058776	0.013789	0.045471	1.682556	-0.016472	5.631243	-0.588020	1.487440	11.721079
...	...	...	...	...	...	...	...	...	...	...
110044	-1.10926	4.234543	1.511927	1.329725	3.764547	0.044679	1.585521	-0.602747	-0.364942	2.148984
110142	-1.10926	2.876553	1.630592	1.782991	0.672120	1.431703	0.405606	-0.602747	-0.583050	0.544999
110147	-1.10926	2.866294	0.325283	0.224889	0.482788	0.107716	0.529477	-0.602747	-0.432651	1.358877
110249	-1.10926	3.009995	0.740609	0.649826	1.385604	0.042127	1.073245	-0.602747	-0.419656	1.554595
110330	-1.10926	4.225959	0.592278	0.442079	1.367667	0.024003	1.369274	-0.602747	-0.373153	1.942874

865 rows × 68 columns

جداسازی train و test

با استفاده از تابع آماده کتابخانه scikit-learn، ۱۵ درصد داده ها را به عنوان داده آموزشی در نظر میگیریم. و دو dataframe جدا برای test و train میسازیم که مدل هایمان هرگز داده test را نبینند. ویژگی shuffle=true نشان می دهد که داده کاملا به هم میریزد و بعد رندوم تفکیک میشود.

```
from sklearn.model_selection import train_test_split

train_df, test_df = train_test_split(normalized_df, test_size=0.15, shuffle=True)
train_df.shape
```

```
(97660, 68)
```

سپس آرایه هایی عددی میسازیم که label ها در y_train و y_test قرار میگیرد و سایر ۶۷ ستون در x_train و x_test قرار میگیرند. اکنون داده ما آماده تحلیل شده است.

۳. مدل‌ها

شبکه عصبی:

یکی از مدل های عالی برای رسیدن به دقت مناسب همواره شبکه های عصبی هستند. البته همیشه مشکل تفسرپذیری دارند، یعنی سوال مهم این که کدام متغیر مهم ترین نقش را در برچسب دارد، نمیتوانند جواب دهند. اما برای رسیدن به دقت خوب در دسته بندی قطعا مناسب هستند.

برای پیاده سازی این قسمت از tensorflow.keras استفاده میکنیم که توابع آماده خوبی در اختیارمان میگذارد. ابتدا مدلی ترتیبی تعریف میکنیم که ۳ لایه دارد. تمام این لایه ها از نوع fully connected هستند چون با دیتای عکس یا صوت یا ... سروکار نداریم که بخواهیم از cnn یا rnn استفاده کنیم.

به ترتیب ۶۷ (به تعداد ویژگی های ورودی، چون best practice است) و ۱۰۰ واحد در دو لایه اول میگذاریم. و چون مسئله ۲ کلاسه است لایه آخر را تک نورونه با تابع فعالسازی sigmoid می گذاریم. تابع فعالسازی دو لایه اول را relu میگذاریم، به دلیل سرعت بیشتر محاسباتی و غیر خطی بودن، این هم best practice است. ساختار مدل با توجه به ورودی ۶۷ تایی چنین چیزی می شود:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 67)                4556

 dense_1 (Dense)             (None, 100)               6800

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 11,457
Trainable params: 11,457
Non-trainable params: 0
```

علاوه بر موارد گفته شده، تعدادی هایپرپارامتر دیگر نیز داریم:

- Batch_size=64: که از مزایای SGD به جای GD بهره مند شویم و ram زیادی مصرف نشود.
- Optimizer=Adam: مشهورترین و پراستفاده ترین بهینه ساز است.
- Loss=binary_crossentropy: مسئله دو کلاسه است و این بهترین می باشد.

- Epochs=5: به نظر با همین عدد کم به دقت قابل قبولی میرسیم و بیشتر ادامه دادن فقط باعث overfit می شود.
- Validation_split=0.1: هنگام آموزش، یک دهم داده ها را جدا میکند و به عنوان داده ارزیابی که مدل ندیده است از آن استفاده میکند. تا ببینیم مدل در حال overfit یا underfit نباشد. چون مشکل کمبود داده نداریم میتوانیم این کار را بکنیم.
- Weight decay و لایه dropout قرار ندادیم چون دقت داده آموزشی و آزمایشی نزدیک هم و بالا بود. یعنی مدل دچار high bias یا high variance نشد.

مدل را آموزش میدهیم و روند کاهشی loss و افزایشی دقت مشخص است:

```
Epoch 1/5
1056/1056 [==============================] - 10s 5ms/step - loss: 0.0307 - accuracy: 0.9946
Epoch 2/5
1056/1056 [==============================] - 5s 4ms/step - loss: 0.0102 - accuracy: 0.9980
Epoch 3/5
1056/1056 [==============================] - 5s 4ms/step - loss: 0.0080 - accuracy: 0.9982
Epoch 4/5
1056/1056 [==============================] - 5s 5ms/step - loss: 0.0081 - accuracy: 0.9982
Epoch 5/5
1056/1056 [==============================] - 5s 4ms/step - loss: 0.0073 - accuracy: 0.9982
<keras.callbacks.History at 0x7ff07d5ebcd0>
```

درخت تصمیم

- تابع visualize_tree، درخت را میگیرد و فایل تصویر آن را می سازد. در این سوال چون لزومی نبود، و اجرایش زمان بر بود اجرای آن را کامنت کرده ام اما به راحتی میتوانید آن را اجرا کنید.
- در تابع desicion_tree، یک dataframe را به همراه ستون مقصد و تعدادی ستون که نباید در درخت بررسی کنیم، گرفته ایم.
- این تابع با استفاده از DesicionTree موجود در کتابخانه scikit_learn.tree کار میکند.
- با کنار گذاشتن ستون مقصد یعنی label، ورودی X را میسازیم.
- ورودی y را هم برابر ویژگی مورد بررسی میگذاریم که همان label است.
- به این ترتیب درخت تصمیم را بر اساس x و y تشکیل داده ایم و آموزش داده ایم.

- در نهایت درخت تصمیم را، به همراه ویژگی های موثر (به ترتیب عمق درخت تصمیم) را به صورت یک لیست بازگردانده ایم.

این تابع را با داده آموزشی صدا میکنیم و درخت را روی آن آموزش میدهیم. خروجی بدین صورت است:

```
number of unique values of each column:
Flow Bytes/s       33294
 Flow IAT Std      30620
 Flow Packets/s    27773
 Flow IAT Mean     27693
Fwd Packets/s      27456
                   ...
 RST Flag Count        2
 SYN Flag Count        2
Fwd PSH Flags          2
 Inbound               2
 Label                 2
Length: 68, dtype: int64

the most important features (features on top of the tree):
                            importance
 Min Packet Length            0.708941
 URG Flag Count               0.163024
 Inbound                      0.101902
 Average Packet Size          0.013626
 Fwd IAT Min                  0.005135
...                                ...
 Fwd IAT Std                  0.000000
```

که میبینیم میزان آنتروپی چند متغیر اول بسیار بالاتر از بعدی ها است و این به معنی خیلی مهم بودن این ویژگی ها در کلاس بندی ماست.

SVM

SVM با نگاشت داده ها به یک فضای ویژگی با ابعاد بالا کار می کند تا نقاط داده را بتوان دسته بندی کرد، حتی زمانی که داده ها به طور خطی قابل جداسازی نیستند. یک جداکننده بین دسته‌ها پیدا می‌شود، سپس داده‌ها به گونه‌ای تبدیل می‌شوند که جداکننده را می‌توان به صورت ابر صفحه رسم کرد. در اینجا برای این کار هم از SVC در کتابخانه scikit learn استفاده کرده ایم. و به راحتی آن را روی داده آموزشی، fit کرده ایم.

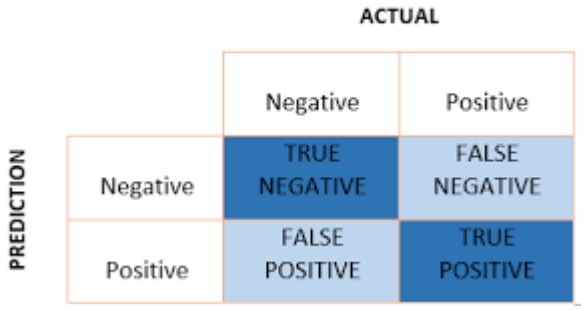
```
from sklearn.svm import SVC

svm_classifier = SVC(random_state=0)
svm_classifier.fit(x_train, y_train)
SVC(random_state=0)
```

```
SVC(random_state=0)
```

۴. ارزیابی روش‌ها

ابتدا مقادیر tp، fp،tn، fn را طبق برچسب های حقیقی و برچسب های پیشبینی شده، در تابع calculate_metrics حساب میکنیم.



سپس طبق فرمول های presicion و recall و f1_score در هر کدام از توابع توسعه داده شده ان ها را به دست میآوریم. تابع f1_score نیاز به ضریب بتا هم دارد که میزان اهمین presicion و recall را نشان میدهد، که در اینجا به دلیل اهمیت برابر، من برابر ۰.۵ قرار دادم.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

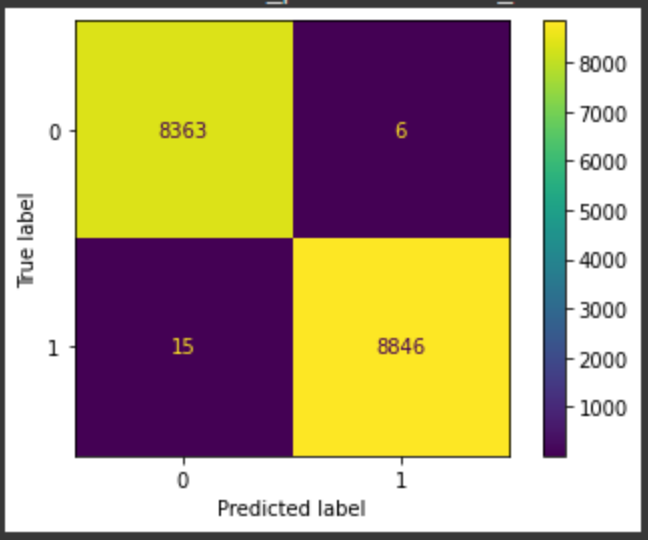
$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

در هر کدام از سلول ها برای یکی از روشها تابع predict را روی داده آزمایشی صدا میزنیم، و پیشبینی های مدل را به این توابع ارزیابی میدهیم تا انواع دقت را بیابیم. به چنین نتایج عالی ای میرسیم:

```
test accuracy of neural network model is: 99.88 %
test presicion of neural network model is: 99.93 %
test recall of neural network model is: 99.84 %
test f1_score of neural network model is: 99.91 %
```

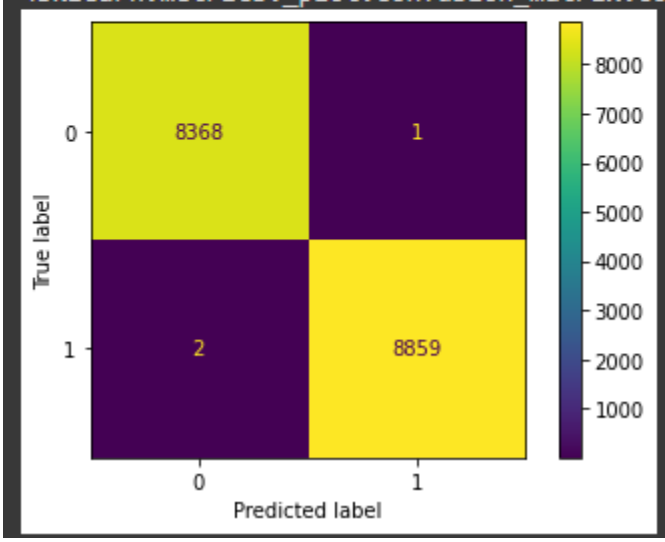
برای امتحان توابع نوسعه داده شده توسط خودم، آن ها را با توبع tensorflow چک کردم و جواب یکسان بود!

```
test accuracy of svm model is: 99.88 %
test presicion of svm model is: 99.93 %
test recall of svm model is: 99.83 %
test f1_score of svm model is: 99.88 %
/usr/local/lib/python3.7/dist-packages/sklearn/utils
  warnings.warn(msg, category=FutureWarning)
<sklearn.metrics._plot.confusion_matrix.ConfusionMat
```



میبینیم که svm هم خیلی خوب بوده و فقط ۶ تا از داده هایی که حمله نبودند را حمله تشخیص داده و ۱۵ تا را برعکس. و چون این روی داده تست است ککه مدل ندیده، نتیجه ای عالی حساب می شود.

```
test accuracy of decision tree model is: 99.98 %
test presicion of decision tree model is: 99.99 %
test recall of decision tree model is: 99.98 %
test f1_score of decision tree model is: 99.98 %
/usr/local/lib/python3.7/dist-packages/sklearn/ut
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/ba
  "X does not have valid feature names, but"
<sklearn.metrics._plot.confusion_matrix.Confusion
```



اوضاع در درخت تصمیم فوق العاده است و مدل تقریبا اشتباهی ندارد. ۳ نمونه در ۱۷۰۰۰ ورودی! البته این که داده ها را balance انتخاب کردیم و پیش پردازش های خوب باعث چنین نتیجه ای شدند.

در اینجا مدل اصلیمان را درخت تصمیم انتخاب میکنیم. و در ادامه برای بهتر سنجیدن آن از p-value هم استفاده میکنیم.

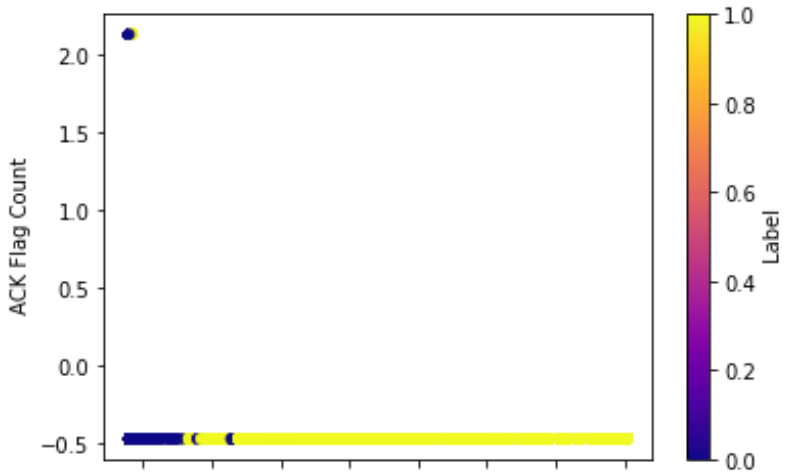
۵. یافتن مهم‌ترین ویژگی‌ها

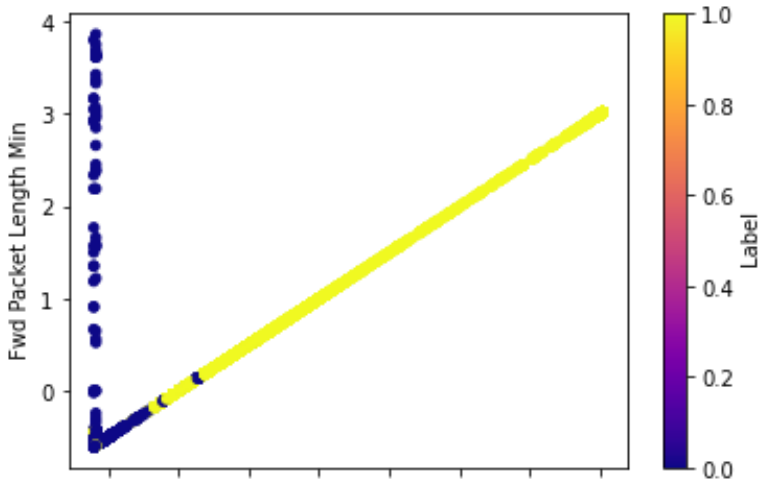
چون هم متغیرها عددی هستند، میتوانیمم از correlation استفاده کنیم. البته عده زیادی از آن ها nnominal هستند که correlation اشتباه میکند. باید دقت کنیم که اعداد منفی و مثبت بازگردانده شده کاملا معنی دار هستند و درواقع چون به دنبال میزان وابستکی هستیم، قدر مطلق این اعداد هرچقدر بیشتر باشد به معنی وابستگی بیشتر است. و علامت عدد وابستگی مستقیم یا معکوس را به ما نشان میدهد.

با حذف nominal ها از خروجی ها، و کنار هم قرار دادن خروجی بهترین متغیرهای درخت تصمیم، به تعدادی ویژگی مهم میرسیم:

```
[' Inbound', ' Min Packet Length', ' ACK Flag Count', ' Fwd IAT Min',
' URG Flag Count', ' Down/Up Ratio']
```

نمودارهای آن ها را رسم کرده ایم و همچنین میانگین مقادیر هرکدام را در کلاس ۰ و کلاس ۱ نوشتیم. میبینیم که همگی کاملا در دو کلاس متمایز ظاهر شده اند.

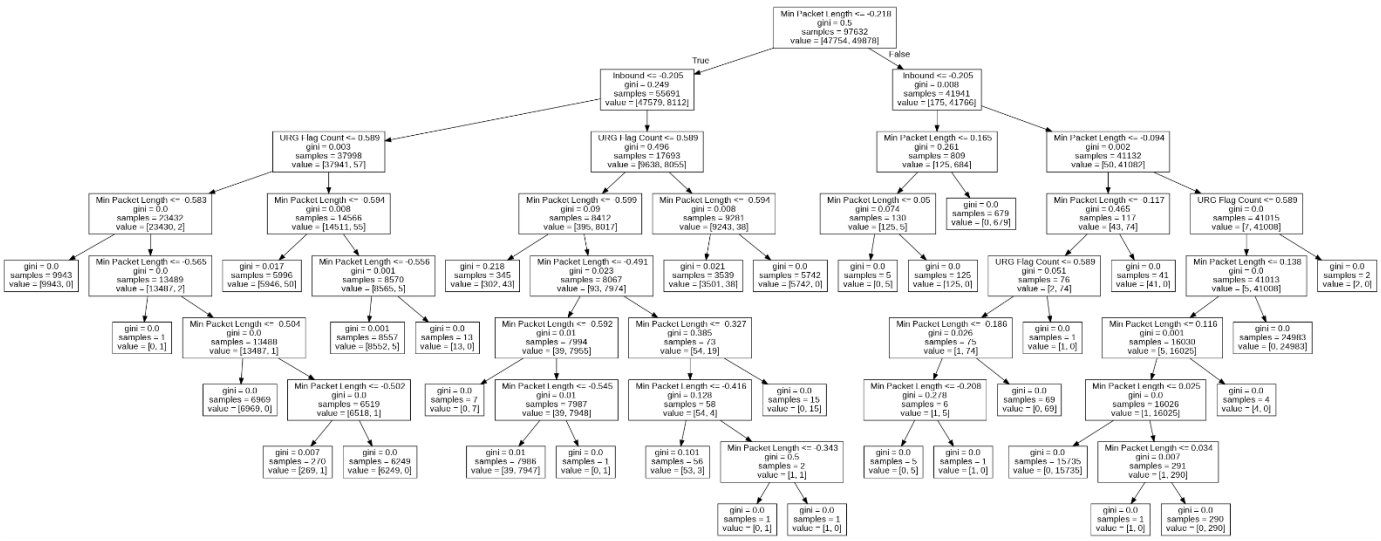




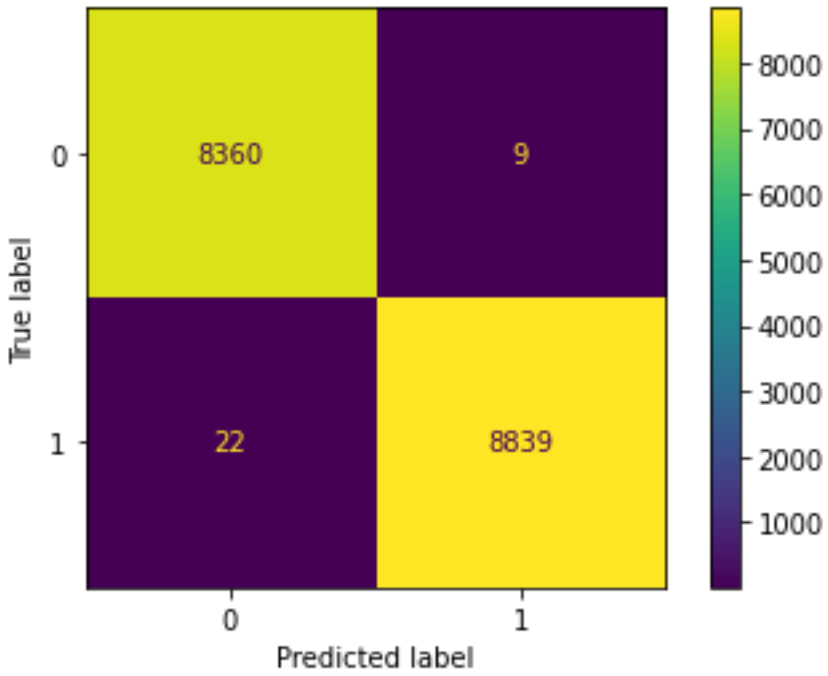
Label	Inbound	Min Packet Length	ACK Flag Count	Fwd IAT Min	URG Flag Count	Down/Up Ratio
0	-0.812109	-0.567156	0.045313	0.027404	0.583605	0.470347
1	0.785103	0.551241	-0.051562	-0.022803	-0.567021	-0.464006

در ادامه فقط سه متغیر را بررسی کرده ایم: `Inbound،URG Flag Count ، Min Packet Length`

چون به نظر تفاوت بیشتری دارند، هم با میانگین هم انتروپی درخت تصمیم و هم شکل نمودار گویاست. در نهایت فقط همین ۳ متغیر و برچسب را نگه داشتیم، و درخت تصمیمی را روی داده آموزشی fit کردیم وبه دقت ۹۹.۸ رسیدیم! یعنی فقط کمی کمتر از قبل، و این یعنی با همین ۳ متغیر به خوبی حمله بودن یا نبودن قابل توصیف است. که به ترتیب اهمیت میتوان طول کوتاه ترین بسته، تعداد بسته های دارای URG، و جهت ترافیک را گفت که مکمل هم هستند. درخت تصمیم با این ۳ متغیر به چنین شکلی است: (فایل dt.png پیوست شده است، برای کیفیت بهتر)



که وقتی آن را روی داده تست هم مشابه قبل بررسی میکنیم به این confusion matrix میرسیم:



کمی خطا زیاد شده اما مدل دقت بالای ۹۹ را میدهد، هم روی تست و هم ترین.

۶. P-value و t-test:

p-value عددی است که از یک آزمون آماری محاسبه می‌شود و نشان می‌دهد که اگر فرضیه ای درست باشد، چقدر احتمال دارد که مجموعه‌ای از مشاهدات را پیدا کنید.

اگر دقت مدل را با روش k-fold بارها اندازه بگیریم، که این کار را در تابع k_fold با k=10 برای یک درخت تصمیم دلخواه انجام داده ایم. که داده آموزشی را ورودی میگیرد. حالا میتوانیم با روش هایی مثل p-value با t-test، ببینیم که چقدر مدل برای مجموعه هههای مختلف train و test مشابه عمل کرده است و درواقع چقدر مدل قابل اتکاست.

اگر مقدار p-value کمتر از ۰.۰۵ باشد به معنی قابل اتکایی بیشتر مدل میباشد. در اینجا درخت تصمیم اول (۶۷ ویژگی) را با تابع k_fold ابتدا برای ۱۰ تقسیم بندی مختلف سنجیدیم و سپس با استفاده از scipy.stats.ttest_1samp آن ها را میسنجیم. به چنین مقادیری میرسد:

```
fold 0 done with accuracy: 99.98 %
fold 1 done with accuracy: 99.97 %
fold 2 done with accuracy: 99.96 %
fold 3 done with accuracy: 99.97 %
fold 4 done with accuracy: 99.96 %
fold 5 done with accuracy: 99.98 %
fold 6 done with accuracy: 99.95 %
fold 7 done with accuracy: 99.95 %
fold 8 done with accuracy: 99.96 %
fold 9 done with accuracy: 99.97 %
Ttest_1sampResult(statistic=29266.75523768695, pvalue=3.2322104724211677e-37)
```

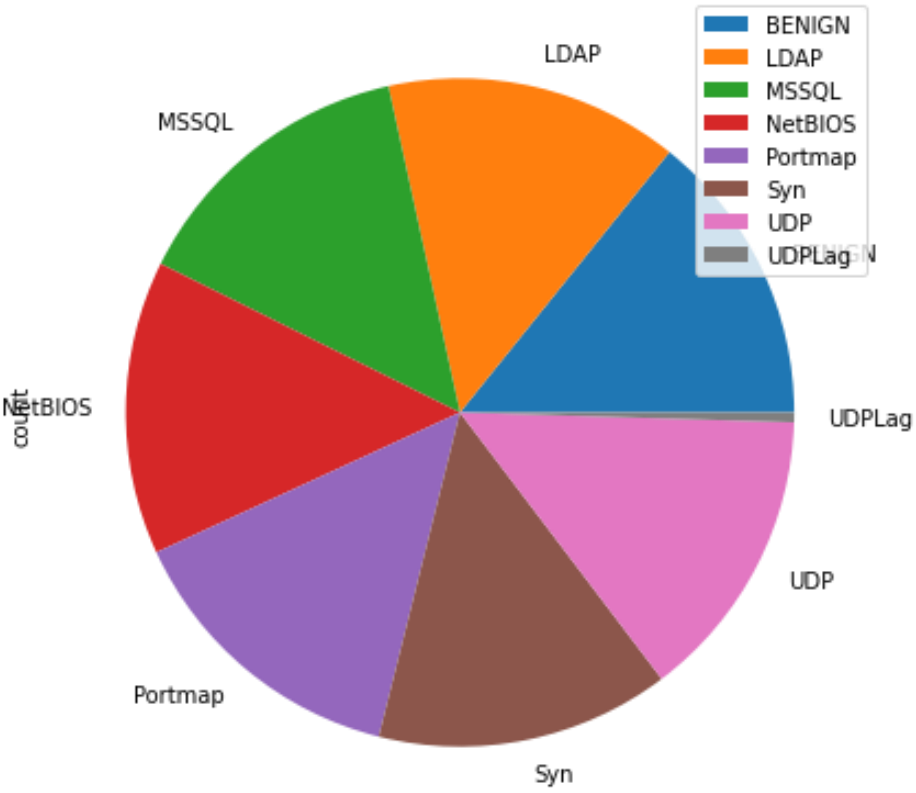
اعداد زیادی خوب هستند! که این بیشتر به علت کامل و مناسب بودن دیتاست است.

همینطور برای درخت تصمیم با ۳ ویژگی هم این کار را کردیم که نتایج بازهم قابل اتکا بودند و کمی ضعیف تر شده بود:

```
fold 0 done with accuracy: 99.79 %
fold 1 done with accuracy: 99.76 %
fold 2 done with accuracy: 99.83 %
fold 3 done with accuracy: 99.82 %
fold 4 done with accuracy: 99.85 %
fold 5 done with accuracy: 99.79 %
fold 6 done with accuracy: 99.85 %
fold 7 done with accuracy: 99.83 %
fold 8 done with accuracy: 99.78 %
fold 9 done with accuracy: 99.85 %
Ttest_1sampResult(statistic=9639.475617551636, pvalue=7.085817732643563e-33)
```

تشخیص نوع حمله

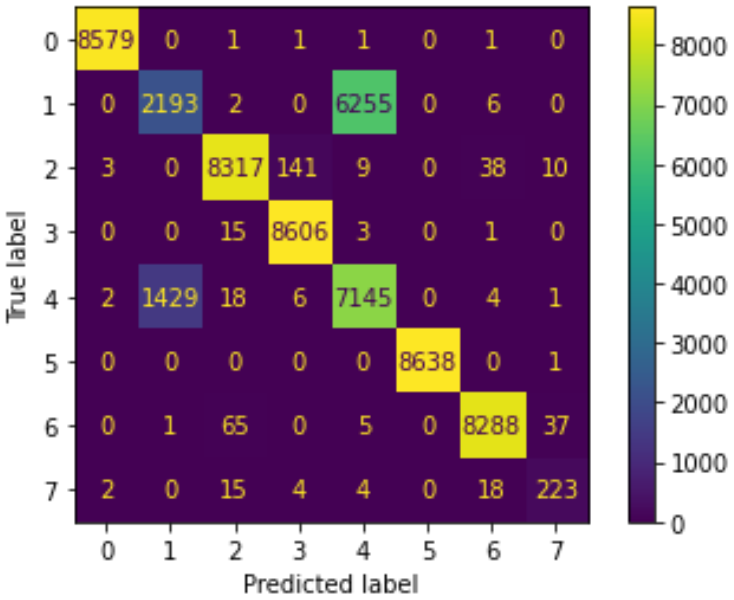
برای بخش امتیازی این پروژه، میخواستیم نوع حمله را هم تشخیص دهیم. برای این کار به دیتاست balance دیگری نیاز داریم، پس از اول sampling را انجام دادیم به نحوی که ۷ کلاس برابر (به جز UDPFlag که کم بود) داشته باشیم:



و حالا همان پیش پردازش های دفعه قبل را عینان روی دیتا انجام دادیم. با یک فرق که label ها را از ۰ تا ۷ شماره زدیم تا نوع حمله هم تشخیص داده شود.

با همان توابع قبلی این بار هم درخت تصمیم را پیاده سازی کردیم اما نتایج به خوبی قبل نبودند و دقت ۸۰ درصد داریم:

ویژگی هایی که کاملا متمایز کنند و انتروپی فاحشی داشته باشند هم دیگر وجود ندار چون مسئله پیچیده تر است.



به خصوص بین کلاس ۱ و ۴ یعنی NETBIOS و Portmap به شدت دچار مشکل شده ایم. این بار هم از شبکه عصبی استفاده میکنیم، ولی چون دچار overfit شدم به آن regularizer و لایه های dropout هم اضافه کردم. تعداد نورون لایه ورودی و مخفی را تغییری ندادم ولی لایه خروجی را با ۸ خروجی و تابع فعالسازی softmax قرار میدهیم که مسئله ۸ کلاسه را حل کند.

نکته: y_train و y_test را در این قسمت به صورت categorical در می آوریم. تا با خروجی شبکه یکنواخت باشد و همچنین ۷ بودن یک کلاس به معنی بیشتر بودن آن از کلاس ۵ نباشد و صفا نوع حمله متفاوت باشد.

سایر هایپرپارامترها را نیز تغییری نمیدهیم.

```
Epoch 1/15
4789/4789 [==============================] - 62s 3ms/step - loss: 0.4216 - accuracy: 0.8069 - precision_5:
0.8319 - recall_5: 0.7753 - val_loss: 0.3355 - val_accuracy: 0.8407 - val_precision_5: 0.8419 - val_recall_5: 0.8389
Epoch 2/15
4789/4789 [==============================] - 15s 3ms/step - loss: 0.3312 - accuracy: 0.8301 - precision_5:
0.8370 - recall_5: 0.8209 - val_loss: 0.3158 - val_accuracy: 0.8418 - val_precision_5: 0.8433 - val_recall_5: 0.8403
Epoch 3/15
4789/4789 [==============================] - 13s 3ms/step - loss: 0.3217 - accuracy: 0.8320 - precision_5:
0.8377 - recall_5: 0.8245 - val_loss: 0.3122 - val_accuracy: 0.8261 - val_precision_5: 0.8272 - val_recall_5: 0.8243
Epoch 4/15
4789/4789 [==============================] - 13s 3ms/step - loss: 0.3152 - accuracy: 0.8327 - precision_5:
0.8376 - recall_5: 0.8261 - val_loss: 0.3091 - val_accuracy: 0.8275 - val_precision_5: 0.8286 - val_recall_5: 0.8251
Epoch 5/15
4789/4789 [==============================] - 13s 3ms/step - loss: 0.3126 - accuracy: 0.8327 - precision_5:
0.8374 - recall_5: 0.8266 - val_loss: 0.3114 - val_accuracy: 0.8410 - val_precision_5: 0.8417 - val_recall_5: 0.8401
Epoch 6/15
4789/4789 [==============================] - 15s 3ms/step - loss: 0.3115 - accuracy: 0.8331 - precision_5:
0.8376 - recall_5: 0.8271 - val_loss: 0.3097 - val_accuracy: 0.8337 - val_precision_5: 0.8346 - val_recall_5: 0.8321
Epoch 7/15
4789/4789 [==============================] - 17s 3ms/step - loss: 0.3077 - accuracy: 0.8332 - precision_5:
0.8377 - recall_5: 0.8275 - val_loss: 0.3078 - val_accuracy: 0.8290 - val_precision_5: 0.8299 - val_recall_5: 0.8267
Epoch 8/15
```

```
4789/4789 [==============================] - 15s 3ms/step - loss: 0.3078 - accuracy: 0.8335 - precision_5:
0.8377 - recall_5: 0.8282 - val_loss: 0.3025 - val_accuracy: 0.8330 - val_precision_5: 0.8341 - val_recall_5: 0.8308
Epoch 9/15
4789/4789 [==============================] - 15s 3ms/step - loss: 0.3072 - accuracy: 0.8329 - precision_5:
0.8370 - recall_5: 0.8274 - val_loss: 0.3044 - val_accuracy: 0.8386 - val_precision_5: 0.8396 - val_recall_5: 0.8378
Epoch 10/15
4789/4789 [==============================] - 16s 3ms/step - loss: 0.3058 - accuracy: 0.8333 - precision_5:
0.8373 - recall_5: 0.8280 - val_loss: 0.3054 - val_accuracy: 0.8306 - val_precision_5: 0.8311 - val_recall_5: 0.8302
Epoch 11/15
4789/4789 [==============================] - 15s 3ms/step - loss: 0.3061 - accuracy: 0.8343 - precision_5:
0.8383 - recall_5: 0.8288 - val_loss: 0.3010 - val_accuracy: 0.8505 - val_precision_5: 0.8521 - val_recall_5: 0.8493
Epoch 12/15
4789/4789 [==============================] - 15s 3ms/step - loss: 0.3058 - accuracy: 0.8339 - precision_5:
0.8379 - recall_5: 0.8287 - val_loss: 0.3065 - val_accuracy: 0.8297 - val_precision_5: 0.8306 - val_recall_5: 0.8280
Epoch 13/15
4789/4789 [==============================] - 14s 3ms/step - loss: 0.3049 - accuracy: 0.8345 - precision_5:
0.8383 - recall_5: 0.8298 - val_loss: 0.3014 - val_accuracy: 0.8473 - val_precision_5: 0.8488 - val_recall_5: 0.8448
Epoch 14/15
4789/4789 [==============================] - 16s 3ms/step - loss: 0.3051 - accuracy: 0.8344 - precision_5:
0.8385 - recall_5: 0.8292 - val_loss: 0.3074 - val_accuracy: 0.8301 - val_precision_5: 0.8314 - val_recall_5: 0.8291
Epoch 15/15
4789/4789 [==============================] - 15s 3ms/step - loss: 0.3046 - accuracy: 0.8342 - precision_5:
0.8382 - recall_5: 0.8291 - val_loss: 0.3071 - val_accuracy: 0.8349 - val_precision_5: 0.8362 - val_recall_5: 0.8336
```

در نهایت حدودا به دقت نه چندان بد و نه چندان خوب ۸۴ درصد میرسیم. نزدیکی presicion و recall هم یعنی به سمت کلاس خاصی متمایل نشده است و این خوب است.