بسمه تعالی



دانشکده مهندسی کامپیوتر

**داده کاوی**

نام استاد: دکتر حسین رحمانی

پروژه دوم

آرمان حیدری

شماره دانشجویی: **97521252**

فروردین 1401

فهرست

[1. بارگذاری داده و نمونه برداری 3](#_Toc103733764)

[2. پیش پردازش داده 6](#_Toc103733765)

[حذف null 6](#_Toc103733766)

[حذف ستون‌ها 6](#_Toc103733767)

[جایگذاری برچسب با اعداد 7](#_Toc103733768)

[نرمال سازی 8](#_Toc103733769)

[جداسازی train و test 9](#_Toc103733770)

[3. مدل‌ها 10](#_Toc103733771)

[شبکه عصبی: 10](#_Toc103733772)

[درخت تصمیم 11](#_Toc103733773)

[SVM 12](#_Toc103733774)

[4. ارزیابی روش‌ها 13](#_Toc103733775)

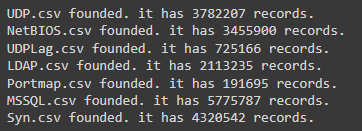
[5. یافتن مهم‌ترین ویژگی‌ها 16](#_Toc103733776)

[6. P-value و t-test: 19](#_Toc103733777)

[تشخیص نوع حمله 20](#_Toc103733778)

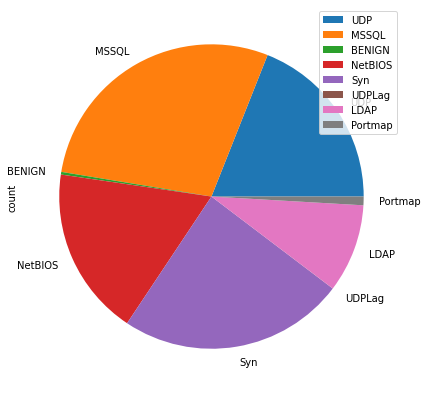
# بارگذاری داده و نمونه برداری

ابتدا داده ها را از url داده شده دانلود میکنیم و از حالت zip خارج میکنیم. سپس مسیر مربوط به 7 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:



در این پروژه به دلیل زیاد بودن حجم دیتا (حدودا 2 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:



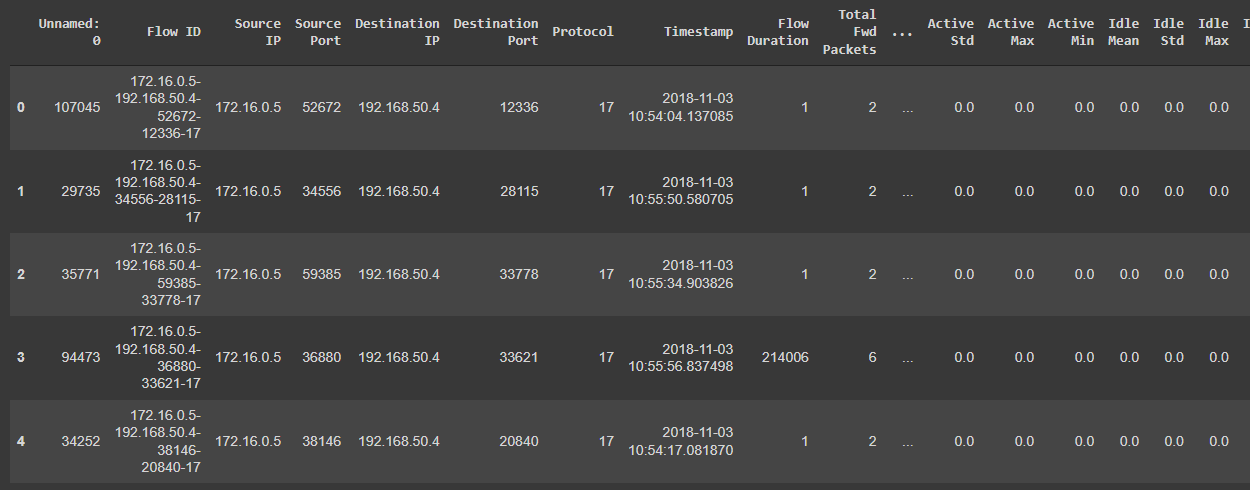
که میبینیم حالت BENIGN که در واقع کلاس 0 ماست داده های خیلی خیلی کمتری از حالت کلاس 1 ما دارد. به همین خاطر نوع خاصی از نمونه برداری را در تابع Sampling خود پیاده سازی کردم.

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.

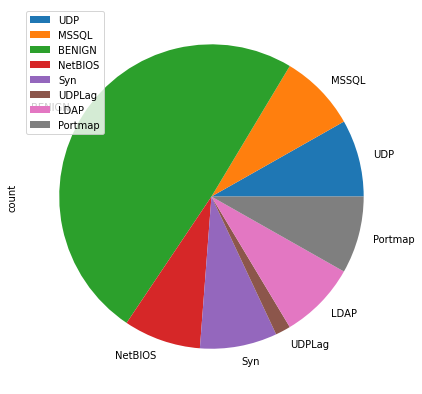
برای اجرای این تابع، مجددا تک تک فایل های csv را میخوانیم و در chunk هایی به طول 10\*\*6، dataframe میسازیم و به تایع sampling پاس میدهیم. البته یک دیکشنری هم داریم که نشان میدهد تاکنون از هر کلاس چقدر انتخاب کرده ایم و تابع کنترل میکند که مقدار انتخاب هایمان از threshold بیشتر نشود.

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



که 115802 سطر و 88 ستون دارد.

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



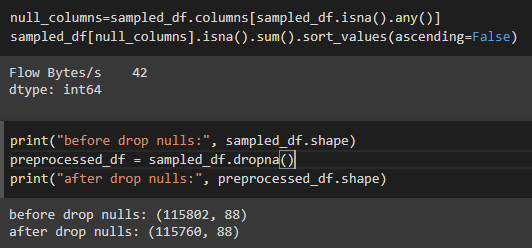
نکته: از نوع حمله UDPLag داده های بسیار کمی داریم. و اگر آن را ملاک میگذاشتیم باید داده کمی انتخاب میکردیم و به نتیجه خوبی نمیرسیدیم. پس تمام رکوردهای این کلاس را هم نمونه برداری میکنیم.

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

# 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 42 رکورد است. پس با حذف رکورد ها چیزی از دست نمیدهیم و آن را حذف میکنیم.

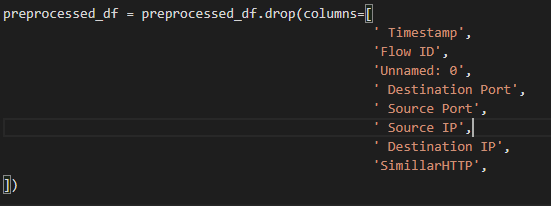


در صورت پروژه تشخیص outlier در این بخش خواسته شده اما چون پس از نرمال سازی راحت تر است، در گام پنجم این مار را انجام داده ام.

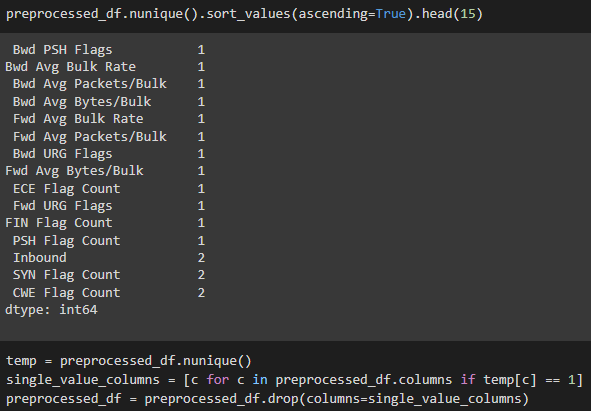
## حذف ستون‌ها

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:



همچنین ستون هایی داریم که فقط مقدار 0 را دارند، در واقع تک مقداره هستند. آن ها را با کمک تابع dataframe.nunique() پیدا کرده ایم و حدود 12 ستون را حذف میکنیم:

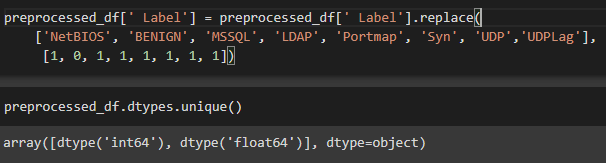


پس از حذف این ها به 67 ستون (ویژگی) میرسیم.

## جایگذاری برچسب با اعداد

برای مسئله اصلی، تمام برچسب ها به جز BENIGN را با 1 و برچسب های BENIGN که فاقد حمله است را با 0 نمایش میدهیم.

همچنین در انتها برای بخش امتیازی، هر کدام از حملات را با عددی مشخص کرده ایم و از 1 تا 7 آن ها را نامگذاری کرده ایم.

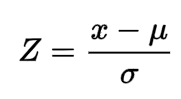


پس از این مرحله میبینیم که به کل داده های عددی داریم و همگی float با int هستند.

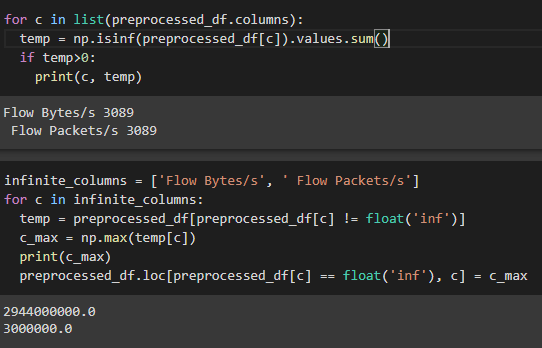
## نرمال سازی

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 0 و انحراف استاندارد 1 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.

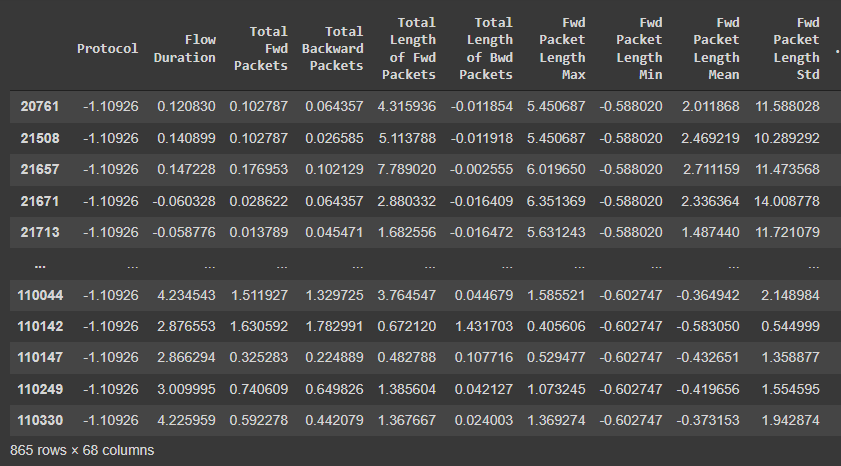


نکته: وقتی این نرمال سازی را پیاده کردم، دچار مشکلی به دلیل inf بودن برخی میانگین ها و nan بودن انحراف معیارها شدم. که با بررسی متوجه شدم به دلیل وجود چندین مقادیر بی نهایت در جدول است. به جای این مقادیر، حداکثر مقدار ممکن در ستون خودشان را قرار میدهیم و در نتیجه مشکل حل می شود. البته این مرحله را باید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



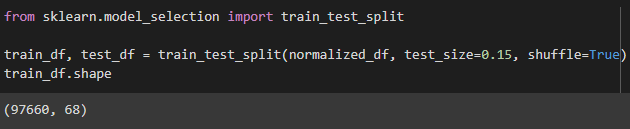
پس از حل این مشکل و نرمال کردن داده ها با تابع، میتوانیم داده های outlier را هم به راحتی پیدا کنیم. طبق نرمال سازی z\_score، معمولا حدود 97 درصد مقادیر موجود در جدول، بین -3 تا +3 قرار میگیرند. من اعداد بالای 10 یا کمتر از -10 را به عنوان outlier در نظر گرفتم چون کاملا از حدود ستون خود خارج هستند.

در ایننجا حدود 800 داده به این شکل داریم که حداقل در یک سطر خود outlier هستند و آن ها را حذف میکنیم.



## جداسازی train و test

با استفاده از تابع آماده کتابخانه scikit-learn، 15 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.



سپس آرایه هایی عددی میسازیم که label ها در y\_train و y\_test قرار میگیرد و سایر 67 ستون در x\_train و x\_test قرار میگیرند. اکنون داده ما آماده تحلیل شده است.

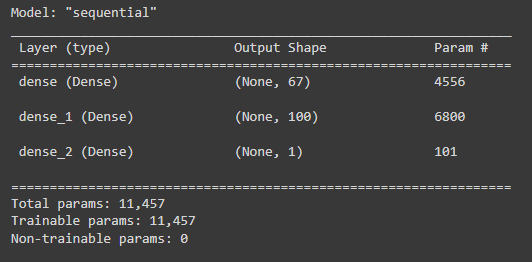
# مدل‌ها

## 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 3 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.

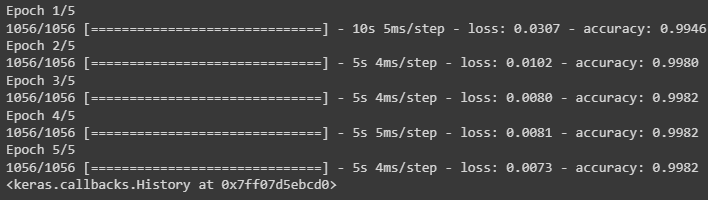
به ترتیب 67 (به تعداد ویژگی های ورودی، چون best practice است) و 100 واحد در دو لایه اول میگذاریم. و چون مسئله 2 کلاسه است لایه آخر را تک نورونه با تابع فعالسازی sigmoid می گذاریم. تابع فعالسازی دو لایه اول را relu میگذاریم، به دلیل سرعت بیشتر محاسباتی و غیر خطی بودن، این هم best practice است. ساختار مدل با توجه به ورودی 67 تایی چنین چیزی می شود:



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.

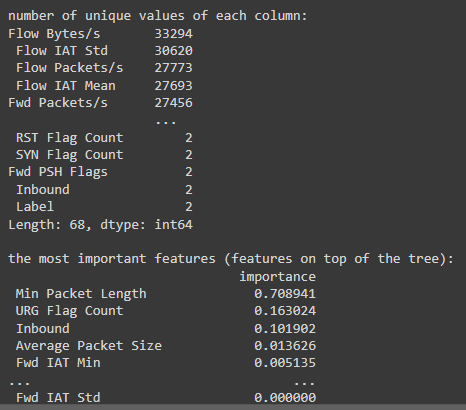
مدل را آموزش میدهیم و روند کاهشی loss و افزایشی دقت مشخص است:



## 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 برایر 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.

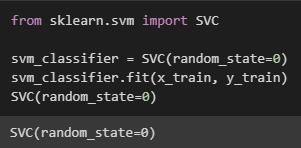
این تابع را با داده آموزشی صدا میکنیم و درخت را روی آن آموزش میدهیم. خروجی بدین صورت است:



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.

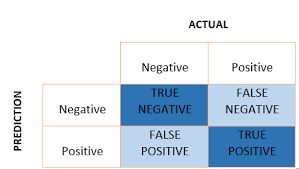
## SVM

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.

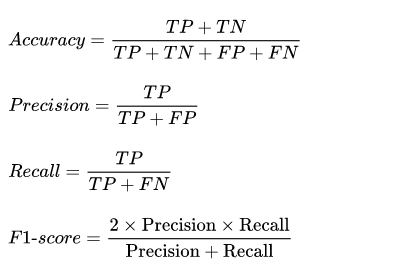


# ارزیابی روش‌ها

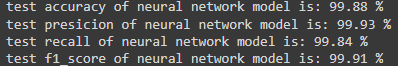
ابتدا مقادیر tp، tn، fp،fn را طبق برچسب های حقیقی و برچسب های پیشبینی شده، در تابع calculate\_metrics حساب میکنیم.



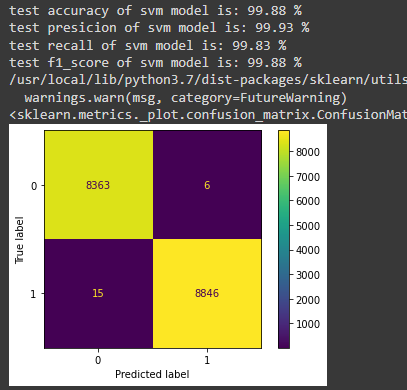
سپس طبق فرمول های presicion و recall و f1\_score در هر کدام از توابع توسعه داده شده ان ها را به دست میآوریم. تابع f1\_score نیاز به ضریب بتا هم دارد که میزان اهمین presicion و recall را نشان میدهد، که در اینجا به دلیل اهمیت برابر، من برابر 0.5 قرار دادم.



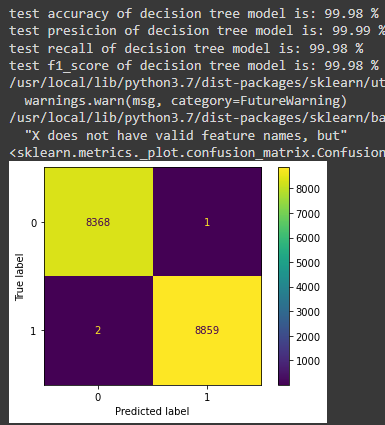
در هر کدام از سلول ها برای یکی از روشها تابع predict را روی داده آزمایشی صدا میزنیم، و پیشبینی های مدل را به این توابع ارزیابی میدهیم تا انواع دقت را بیابیم. به چنین نتایج عالی ای میرسیم:



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



میبینیم که svm هم خیلی خوب بوده و فقط 6 تا از داده هایی که حمله نبودند را حمله تشخیص داده و 15 تا را برعکس. و چون این روی داده تست است ککه مدل ندیده، نتیجه ای عالی حساب می شود.



اوضاع در درخت تصمیم فوق العاده است و مدل تقریبا اشتباهی ندارد. 3 نمونه در 17000 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.

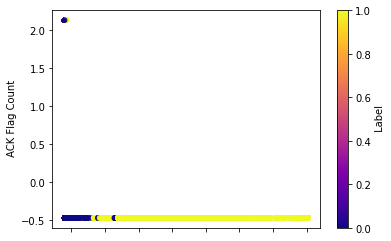
# یافتن مهم‌ترین ویژگی‌ها

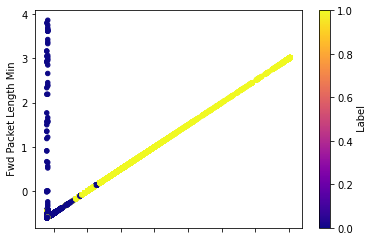
چون هم متغیرها عددی هستند، میتوانیمم از correlation استفاده کنیم. البته عده زیادی از آن ها nnominal هستند که correlation اشتباه میکند. باید دقت کنیم که اعداد منفی و مثبت بازگردانده شده کاملا معنی دار هستند و درواقع چون به دنبال میزان وابستکی هستیم، قدر مطلق این اعداد هرچقدر بیشتر باشد به معنی وابستگی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

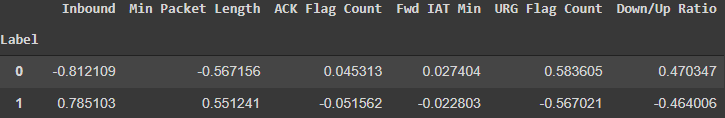
با حذف nominal ها از خروجی ها، و کنار هم قرار دادن خروجی بهترین متغیرهای درخت تصمیم، به تعدادی ویژگی مهم میرسیم:

[' 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 0 و کلاس 1 نوشتیم. میبینیم که همگی کاملا در دو کلاس متمایز ظاهر شده اند.

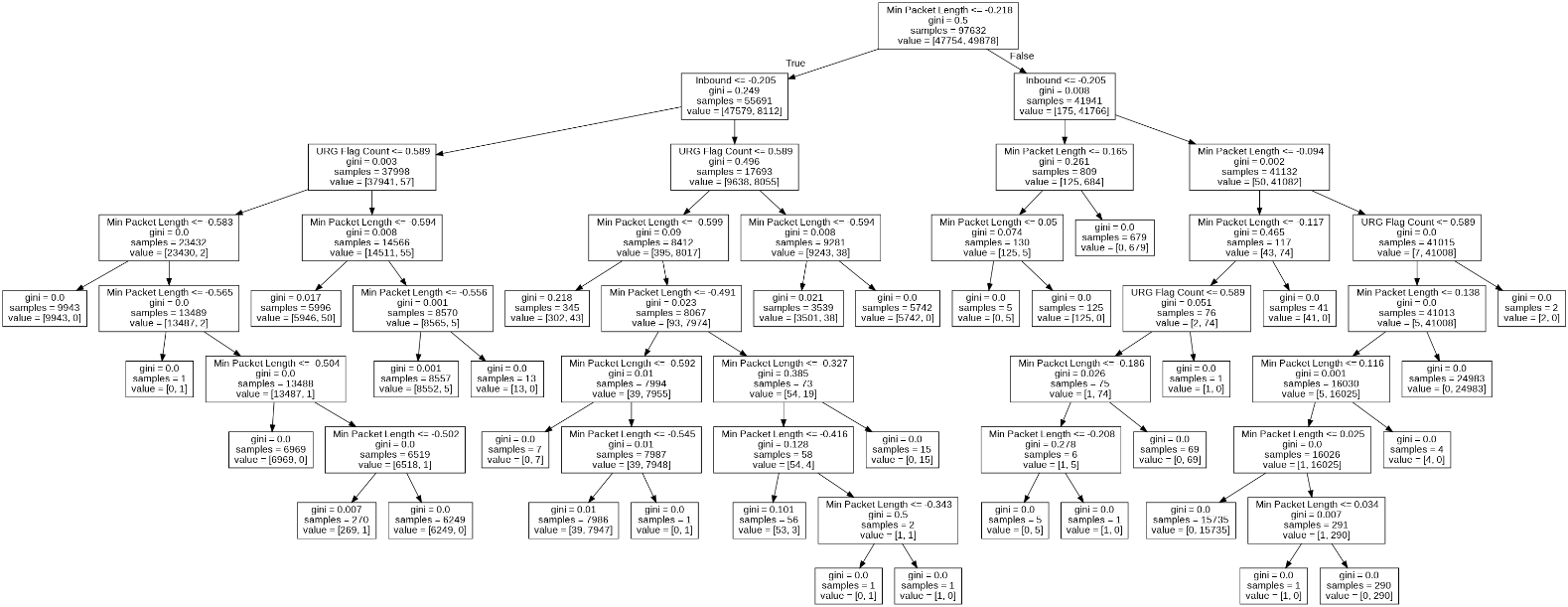




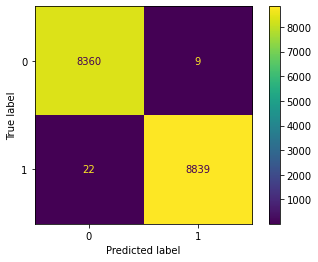


در ادامه فقط سه متغیر را بررسی کرده ایم: Min Packet Length ، URG Flag Count، Inbound

چون به نظر تفاوت بیشتری دارند، هم با میانگین هم انتروپی درخت تصمیم و هم شکل نمودار گویاست. در نهایت فقط همین 3 متغیر و برچسب را نگه داشتیم، و درخت تصمیمی را روی داده آموزشی fit کردیم وبه دقت 99.8 رسیدیم! یعنی فقط کمی کمتر از قبل، و این یعنی با همین 3 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 3 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:



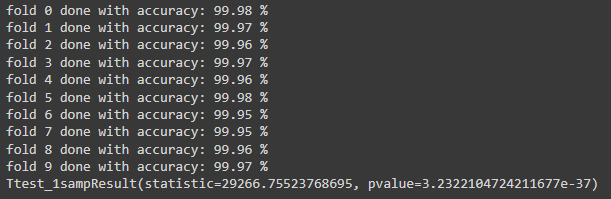
کمی خطا زیاد شده اما مدل دقت بالای 99 را میدهد، هم روی تست و هم ترین.

# 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.

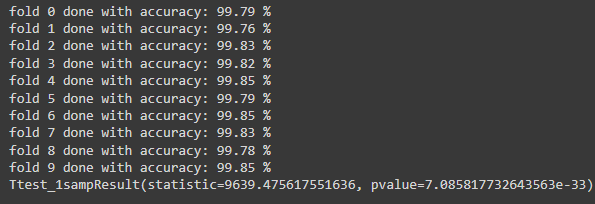
اگر دقت مدل را با روش k-fold بارها اندازه بگیریم، که این کار را در تابع k\_fold با k=10 برای یک درخت تصمیم دلخواه انجام داده ایم. که داده آموزشی را ورودی میگیرد. حالا میتوانیم با روش هایی مثل p-value با t-test، ببینیم که چقدر مدل برای مجموعه ههای مختلف train و test مشابه عمل کرده است و درواقع چقدر مدل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

اگر مقدار p-value کمتر از 0.05 باشد به معنی قابل اتکایی بیشتر مدل میباشد. در اینجا درخت تصمیم اول (67 ویژگی) را با تابع k\_fold ابتدا برای 10 تقسیم بندی مختلف سنجیدیم و سپس با استفاده از scipy.stats.ttest\_1samp آن ها را میسنجیم. به چنین مقادیری میرسد:



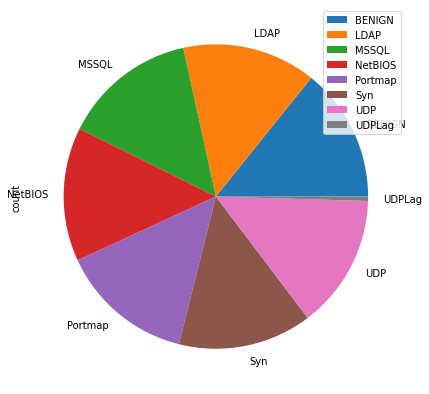
اعداد زیادی خوب هستند! که این بیشتر به علت کامل و مناسب بودن دیتاست است.

همینطور برای درخت تصمیم با 3 ویژگی هم این کار را کردیم که نتایج بازهم قابل اتکا بودند و کمی ضعیف تر شده بود:



# تشخیص نوع حمله

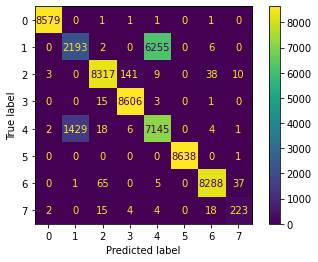
برای بخش امتیازی این پروژه، میخواستیم نوع حمله را هم تشخیص دهیم. برای این کار به دیتاست balance دیگری نیاز داریم، پس از اول sampling را انجام دادیم به نحوی که 7 کلاس برابر (به جز UDPFlag که کم بود) داشته باشیم:



و حالا همان پیش پردازش های دفعه قبل را عینان روی دیتا انجام دادیم. با یک فرق که label ها را از 0 تا 7 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 80 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 1 و 4 یعنی NETBIOS و Portmap به شدت دچاار 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 8 خروجی و تابع فعالسازی softmax قرار میدهیم که مسئله 8 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 7 بودن یک کلاس به معنی بیشتر بودن آن از کلاس 5 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 84 درصد میرسیم. نزدیکی presicion و recall هم یعنی به سمت کلاس خاصی متمایل نشده است و این خوب است.