۱- مقدمه

شرح متدولوژی انتخابی:

CRSIP-DM مخفف عبارت Cross-industry standard process for data mining و به معنی «فرآیند استاندارد صنعتی متقاطع داده کاوی» است. فرایند کریسپ، یک مدل فرآیندی استاندارد منبع باز است که رویکردهای عمومی متخصصان داده کاوی را تشریح می‌کند. این روش از پرکاربردترین مدل‌های تحلیلی است.

گروهی از شرکت‌های اروپایی در دهه ۱۹۹۰ برای اولین بار از فرایند کریسپ برای انجام پروژه‌های داده کاوی استفاده کردند. این فرایند دارای ۶ محله اصلی است. این مراحل متوالی، از درک نیازهای اصلی کسب و کار شروع شده و با ارائه راهکارهای مفید به پایان می‌رسد.

همان طور که اشاره کردیم، روشcrisp dm ، الگوی فرآیند محور داده کاوی است که راهکاری نظام‌مند، مفید و کاربردی برای ابعاد داده و همچنین خوشه بندی داده‌ها ارائه می‌کند. از آنجا که خوشه بندی از مباحث مهم و اساسی در داده کاوی محسوب می‌شود، خوب است به خوشه بندی نیز اشاره مختصری داشته باشیم. خوشه (Cluster) به مجموعه‌ای از داده‌های شبیه به هم گفته می‌شود و خوشه بندی فرآیندی است که به کمک آن می‌توانید مجموعه‌ای از اشیا را به گروه‌های مجزا تفکیک کنید.

برای انجام خوشه‌بندی، داده‌ها را به خوشه‌های مختلف تقسیم می‌کنند. بنابراین، شباهت میان داده‌های درون هر خوشه به حداکثر و شباهت میان داده‌های درون خوشه‌های متفاوت به حداقل می‌رسد. در طبقه‌بندی، به هر داده یک طبقه یا کلاس از پیش تعیین شده اختصاص می‌یابد. در حالی که در خوشه‌بندی هیچ اطلاعی از کلاس‌های موجود درون داده‌ها وجود ندارد و خوشه‌ها از داده‌ها استخراج می‌شوند. خوشه بندی را می‌توانیم به عنوان مهم‌ترین مساله در یادگیری بدون نظارت در نظر بگیریم.

مراحل متدولوژی CRISP :

۱) فهم کسب و کار(Business Understanding)

در این مرحله، یک متخصص علم داده بایستی کسب و کاری که می‌خواهد بر روی آن پروژه داده‌کاوی انجام دهد را به خوبی بشناسد. در مرحله‌ی فهم کسب و کار، بایستی زوایای مختلف آن کسب و کار، محدودیت‌ها، شرایط موجود و اهداف آن کسب و کار از پروژه یا پروژه‌های جاری را بررسی نمود. این مرحله ذهن متخصص

علم داده را برای کار بر روی پروژه آماده می‌کند و به او اجازه می‌دهد تا با شناخت بیشتر و بهتر به سراغ مراحل بعدی برود. در این مرحله، یک متخصص علم داده می‌تواند تا حدودی به کسب و کارِ موجود مسلط شده و فهم خود را از آن کسب و کار تا حد ممکن بالا ببرد.

۲) فهم داده‌ها(Data Understanding)

در مرحله‌ی فهم داده‌ها، متخصص علم داده، به سراغ داده‌های موجود کسب و کار رفته آن را برای شروع پروژه بررسی می‌کند. در این مرحله، عملیاتی مانند «آنالیز اکتشافی داده‌ها «EDA – و ساخت گزارش‌های اولیه از داده‌ها می‌تواند بسیار کمک کننده باشد. با فهم داده‌ها و درک ابعاد و ویژگی‌های مختلف آن، می‌توان ایده‌های مختلف را مطرح کرد و ساختار اصلی پروژه را تعیین نمود. در این مرحله می‌توان کیفیت داده‌ها را نیز ارزیابی کرد و در صورت نامناسب بودن داده‌ها، با مشورت و مشارکت قسمت‌های مختلف کسب و کار، این داده‌ها را بهبود بخشید.

۳) آماده‌سازی داده‌ها(Data Preparation)

بعد از فهم کسب و کار و فهم داده‌ها، حال می‌توان داده‌ها را آماده‌ی تحلیل و مدل‌سازی کرد. اگر دوره‌ی پیش پردازش داده‌ها را در چیستیو مطالعه کرده باشید، احتمالاً به سادگی می‌توانید این مرحله را درک کنید. در این مرحله، داده‌های کثیف، تمیز می‌شوند و داده‌ها به صورت ساختاری، برای مرحله‌ی بعدی آماده‌سازی می‌شوند. در این مرحله همچنین می‌توان مجموعه داده‌های مختلف را با یکدیگر ترکیب کرد تا به مجموعه داده‌ی بهتر و با کیفیت‌تری رسید.

۴) مدل‌سازی(Modeling)

بسته به اینکه مسئله‌ی شما چه نوع مسئله‌ایست در این مرحله بایستی از الگوریتم‌ها و روش‌های مخصوص به خود استفاده کنید. مثلاً اگر مسئله‌ی شما طبقه‌بندی داده‌هاست، بایستی از الگوریتم‌های طبقه‌بندی برای یادگیری استفاده کنید و یا اگر مسئله‌ی شما در دسته‌ی خوشه‌بندی قرار می‌گیرد، می‌توانید یکی از الگوریتم‌های خوشه‌بندی را برای پروژه‌ی خود مورد استفاده قرار دهید. البته در یک پروژه‌ی داده‌کاوی، ممکن است مسائل مختلف و ترکیبی وجود داشته باشد که نیاز به عملیات پیچیده‌تری جهت مدل سازی دارند.

۵) ارزیابی(Evaluation)

چیزی که قابل ارزیابی نباشد، بهبود پیدا نمی‌کند. اگر در مراحل قبلی داده‌ها را آماده کردید و مدلی ساختید، بایستی بتوانید مدلِ خود را ارزیابی کنید. این ارزیابی بستگی به مدلِ انتخابی دارد. برای مثال اگر مسئله‌ی

شما طبقه‌بندی بود، می‌توانید از روش‌های ارزیابی الگوریتم‌های طبقه‌بندی استفاده کنید. طبیعتاً اگر مدلِ شما به اندازه‌ی کافی کیفیت نداشت، بهتر است به مراحل قبلی بازگرید و مدل یا داده‌ها یا روش‌های آماده‌سازی داده‌هایتان را بهبود بخشیده و مجدداً ارزیابی را انجام دهید.

۶) پیاده‌سازی و انتشار(Deploy)

در نهایت، بایستی نرم افزاری توسعه دهید تا کاربران بتوانند از زحمات شما استفاده کنند. این مرحله، معمولاً با کمک مهندسین نرم افزار و برنامه نویسان انجام می‌شود.

۲- گام اول: درک خواسته های کسب و کار یا Business Understanding

یکی از مهمترین موضوعات در دنیای مدرن، توجه به امنیت در کلیه حوزه های زندگی است. مسئله امنیت اطلاعات در شرکت ها به ویژه بخش های مرتبط با اطلاعات حیاتی یک موضوع پررنگ و چالش برانگیز است. اطلاعات حیاتی ممکن است به وسیله عوامل داخلی و خارجی، به صورت خواسته یا ناخواسته به سرقت برود و خسارتهای جبران ناپذیری را به ارمغان بیاورد.

امنیت اطلاعات به منظور حفظ و نگهداری اطلاعات از فعالیت‌هایی هم‌چون جلوگیری از دسترسی غیرمجاز، استفاده، خواندن، تغییر، دستکاری و یا افشا کردن اطلاعات تا تعریف ساختار صحیح مدیریت دسترسی، استفاده صحیح و روالمند از داده‌های سازمان تشکیل می‌شود و موجب کنترل محرمانگی، یکپارچگی و در دسترس بودن داده‌ها در سازمان خواهد شد.

بدین ترتیب سازمان ها ملزم هستند به منظور جلوگیری از سرقت اطلاعات نسبت به اجرا استانداردهای امنیتی اقدام کنند. با پیاده سازی راهبردهای امنیتی ضمن ایجاد سیاستهای کنترلی، امنیتی و محافظت از داده ها سبب شکل گیری راهبرد و تکنیک های خاص در راستای افزایش قدرت تصمیم گیری، رقابت برتر، شکل گیری اعتماد نزد اعتماد و استفاده از سیستمهای مدیریت امنیت اطلاعات ISMS در سازمانها می گردد .

هدف از ایجاد امنیت اطلاعات حفظ و نگهداری از داده های حیاتی از فعالیت هایی نظیر افشاء، دسترسی غیرمجاز، خواندن، دستکاری، تغییر و سوء استفاده است. در واقع امنیت اطلاعات در یکپارچگی، کنترل محرمانگی و در دسترس بودن اطلاعات برای کارمندان حائز اهمیت است .

به طور مختصر وجود هرگونه مشکل امنیتی سبب تحمیل خسارتهای جبران ناپذیر به سازمان می شود. از سری خسارتهای مشکل امنیتی در سازمان می توان به موارد ذیل اشاره کرد :

- کاهش اعتماد مشتریان، سرمایه گذاران و کارمندان
- کاهش چشمگیر مشتریان سازمان
- از دست رفتن اعتبار سازمان در میان رقبا
- افزایش هزینه های سازمانی
- کاهش سطح بهره وری و درآمد سازمان

واژه امنیت اطلاعات تنها برای یک الی چند دستگاه کاربرد ندارد، بلکه برای کلیه سطوح سازمانی در نظر گرفته می شود. به عبارتی ضرورت دارد تمهیداتی در نظر گرفته شود که از بدو ورود اطلاعات به صورت سخت افزاری و نرم افزاری الی محل قرار گیری، ذخیره و بایگانی آن کلیه تدابیر امنیتی در راستای مقابله با تهدیدات مختلف شکل بگیرد.

راه‌کار بسیاری از سازمان‌ها استفاده از گران‌قیمت‌ترین فایروال‌ها و برنامه‌های ضد باج‌افزار و آنتی‌ویروس‌ها می‌باشد اما شایان ذکر هست که فقط داشتن این ابزارهای امنیتی کافی نیست و این ابزارها زمانی سازنده و مفید هستند که با شناخت و تحلیل‌های دقیق امنیتی، استفاده از روال‌های استاندارد در به کارگیری و کنترل سیستم‌های امنیتی و به‌روزرسانی مداوم این سیستم‌ها صورت بگیرد.

این پروژه جهت بررسی نقاط ضعف امینت سایبری یک سازمان که در حوزه کسب و کار اینترنتی فعالیت دارد، انجام می شود. پس بنابراین هدف ما در این پروژه بررسی وضعیت امنیت سازمان، شناسایی نقاط ضعف و اطلاع به سازمان جهت برطرف سازی نقاط ضعف امنیتی می باشد.

سازمان مورد نظر ما به طور چشم گیری مورد حملات امنیتی مختلف قرار گرفته است و جهت مقابله با حملات سایبری از یک سیستم تشخیص نفوذ (IDS) استفاده کرده است. ما از log های این IDS استفاده خواهیم و نقاط ضعف سازمان را پیدا می کنیم. این نقاط ضعف را به سازمان اعلام می کنیم تا سازمان در زمینه برطرف نمودن این نقاط ضعف تصمیمات مقتضی را بگیرد.

۳- گام دوم: درک داده ها یا Data Understanding

دیتاست NSL-KDD در سال ۲۰۰۹ بعنوان نسخه جدید تجدید نظر شده در مجموعه داده اصلی KDDCup99 ارائه شد. از یک طرف ، NSL-KDD ویژگی های سودمند و چالش برانگیز KDDCup را حفظ کرد.

از طرف دیگر، با حذف رکوردهای اضافی، عقلانی کردن تعداد نمونه ها و حفظ تنوع نمونه های انتخاب شده، به اشکالاتی که از مجموعه داده اصلی به ارث رسیده است پرداخته است. شایان ذکر است که مجموعه داده NSL-KDD برای به حداکثر رساندن سختی پیش بینی، که ویژگی های برجسته آن را تشکیل می دهد، گردآوری شده است.

در کل ۴۲ ویژگی برای دیتاست در نظر گرفته شده که ۴۱ ویژگی مربوط به داده های جمع آوری شده و ویژگی آخر نیز به عنوان برچسب هدف با عنوان نرمال یا حمله می باشد.

در این تحقیق از مجموعه های +KDDTrain ، +KDDTest وKDDTrain+_20Percent مجموعه داده های NSL-KDD استفاده شده است. مجموعه داده NSL-KDD شامل مجموعه های +KDDTrain، +KDDTest و KDDTrain+_20Percent می باشد. مجموعه داده های مجموعه +KDDTrain به عنوان مجموعه داده برای آموزش شامل ۱۲۵۹۷۳ نمونه است که شامل ۵۸۶۳۰ مورد ترافیک حمله و ۶۷۳۴۳ نمونه ترافیک عادی است. مجموعه +KDDTest شامل ۲۲۵۴۴ نمونه است و برای تست از آن استفاده می شود. KDDTrain+_20Percent شامل ۲۰ درصد از +KDDTrain (۱۱۸۵۰ نمونه) می باشد و برای ارزیابی از آن استفاده می شود.

همانطور که قبلاً هم گفته شده در این دیتاست ۴۲ ستون وجود دارد که ۴۱ ستون یا ویژگی مربوط به شبکه هست و ستون ۴۲ نیز برای برچسب گذاری نوع حمله و نرمال بودن است. (البته یک ستون ۴۳ نیز وجود دارد که از آن استفاده نمی شود). جدول زیر مشخصات ۴۱ ویژگی یا ستون مربوطه را نمایش می دهد.

	Feature name	Type	Description

1	Duration	continuous	length (number of seconds) of the connection
2	protocol_type	discrete	type of the protocol, e.g., tcp, udp, etc
3	Service	discrete	Network service on the destination, e.g., http, telnet, etc.
4	src_bytes	continuous	number of data bytes from source to destination
5	dst_bytes	continuous	number of data bytes from destination to source
6	Flag	discrete	normal or error status of the connection
7	Land	discrete	1 if connection is from/to the same host/port; 0 otherwise
8	wrong_fragment	continuous	number of ‘‘wrong’’ fragments
9	Urgent	continuous	number of urgent packets
10	Hot	continuous	number of ‘‘hot’’ indicators
11	num_failed_logins	continuous	number of failed login attempts
12	logged_in	discrete	1 if successfully logged in; 0 otherwise
13	num_compromised	continuous	number of ‘‘compromised’’ conditions
14	root_shell	discrete	1 if root shell is obtained; 0 otherwise
15	su_attempted	discrete	1 if ‘‘su root’’ command attempted; 0 otherwise
16	Num_root	continuous	number of ‘‘root’’ accesses
17	Num_file_creations	continuous	number of file creation operations
18	Num_shells	continuous	number of shell prompts
19	Num_access_files	continuous	number of operations on access control files
20	num_outbound_cmds	continuous	number of outbound commands in an ftp session
21	is_hot_login	discrete	1 if the login belongs to the ‘‘hot’’ list; 0 otherwise
22	is_guest_login	discrete	1 if the login is a ‘‘guest’’login; 0 otherwise

23	Count	continuous	number of connections to the same host as the current connection in the past two seconds
24	serror_rate	continuous	% of connections that have ‘‘SYN’’ errors
25	rerror_rate	continuous	% of connections that have ‘‘REJ’’ errors
26	same_srv_rate	continuous	% of connections to the same service
27	diff_srv_rate	continuous	% of connections to different services
28	srv_count	continuous	number of connections to the same service as the current connection in the past two seconds
29	srv_serror_rate	continuous	% of connections that have ‘‘SYN’’ errors
30	srv_rerror_rate	continuous	% of connections that have ‘‘REJ’’ errors
31	srv_diff_host_rate	continuous	% of connections to different hosts
32	dst_host_count	continuous	count for destination host
33	dst_host_srv_count	continuous	srv_count for destination host
34	dst_host_same_srv_rate	continuous	same_srv_rate for destination host
35	dst_host_diff_srv_rate	continuous	diff_srv_rate for destination host
36	dst_host_same_src_port_rate	continuous	same_src_port_rate for destination host
37	dst_host_diff_host_rate	continuous	diff_host_rate for destination host
38	dst_host_serror_rate	continuous	serror_rate for destination host
39	dst_host_srv_serror_rate	continuous	srv_serror_rate for destination host
40	dst_host_rerror_rate	continuous	rerror_rate for destination host
41	dst_host_srv_rerror_rate	continuous	srv_serror_rate for destination hos

۴- گام سوم: آماده سازی داده ها یا Data Preparation

چون مجموعه داده ها شامل نام ستون ها نمی شود، ابتدا باید نام ستون ها را اضافه کنیم.

تغییر بعدی که می خواهیم انجام دهیم در مورد attack field است. ما ستونی را با عنوان attack_flag اضافه می کنیم که مقادیر "عادی" را به عنوان ۰ و مقدار حمله را به عنوان ۱ رمزگذاری می کند.

در مرحله بعد، ما یک فیلد جدید به نام attack_map اضافه می کنیم و هر یک از حملات را بر اساس نوع حمله (فیلد attack) و بر اساس جدول زیر به ۵ دسته (۴ دسته حمله و یک دسته ترافیک نرمال) طبقه‌بندی می‌کنیم.

attack_map		attack
0	Normal	normal
1	Denial of Service attacks	▪ apache2 ▪ back ▪ land ▪ neptune ▪ mailbomb ▪ pod ▪ processtable ▪ smurf ▪ teardrop ▪ udpstorm ▪ worm
2	Probe attacks	▪ ipsweep ▪ mscan ▪ nmap ▪ portsweep ▪ saint ▪ satan
3	Privilege escalation attacks	▪ buffer_overflow ▪ loadmdoule ▪ perl ▪ ps ▪ rootkit ▪ sqlattack ▪ xterm

4	Remote access attacks	▪ ftp_write ▪ guess_passwd ▪ http_tunnel ▪ imap ▪ multihop ▪ named ▪ phf ▪ sendmail ▪ snmpgetattack ▪ snmpguess ▪ spy ▪ warezclient ▪ warezmaster ▪ xclock ▪ xsnoop

برخی حدسیات اولیه از آنچه در مجموعه داده داریم می تواند مفید باشد. در اینجا ما یک جدول ساده از attack و پروتکلی که برای این حمله از آن استفاده شده است داریم. در تجزیه و تحلیل ترافیک شبکه ،پروتکل یک ابزار ساده برای ایجاد چند دسته اولیه برای دسته بندی داده هاست. در این بخش ما ترافیک عادی (normal) را به عنوان یک معیار در مجموعه درنظر گرفته ایم.

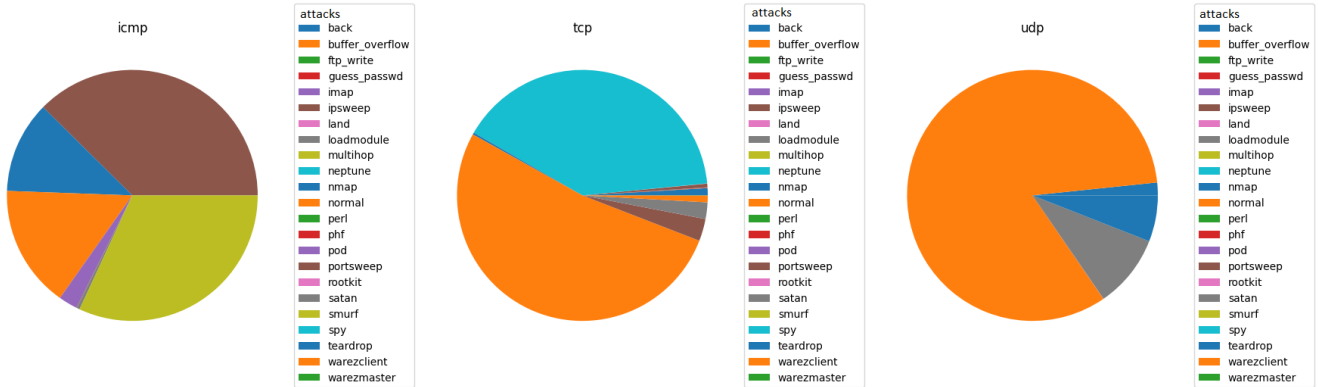
protocol_type attack	icmp	tcp	udp
back	0	956	0
buffer_overflow	0	30	0
ftp_write	0	8	0
guess_passwd	0	53	0
imap	0	11	0
ipsweep	3117	482	0
land	0	18	0
loadmodule	0	9	0
multihop	0	7	0
neptune	0	41214	0
nmap	981	265	247
normal	1309	53599	12434
perl	0	3	0
phf	0	4	0

pod	201	0	0
portsweep	5	2926	0
rootkit	0	7	3
satan	32	2184	1417
smurf	2646	0	0
spy	0	2	0
teardrop	0	0	892
warezclient	0	890	0
warezmaster	0	20	0

ما از جدول بالا می توانیم بفهمیم که بیشتر حملات یک پروتکل خاص را هدف قرار می دهند. اما چندین پروتکل (مثلا satan, nmap, ipsweep) وجود دارند که از هر سه پروتکل استفاده می کنند.

همچنین از اینکه داده های icmp در ترافیک معمولی کمتر یافت می شود می توان حدس زد که از این پروتکل جهت حمله استفاده شده است.

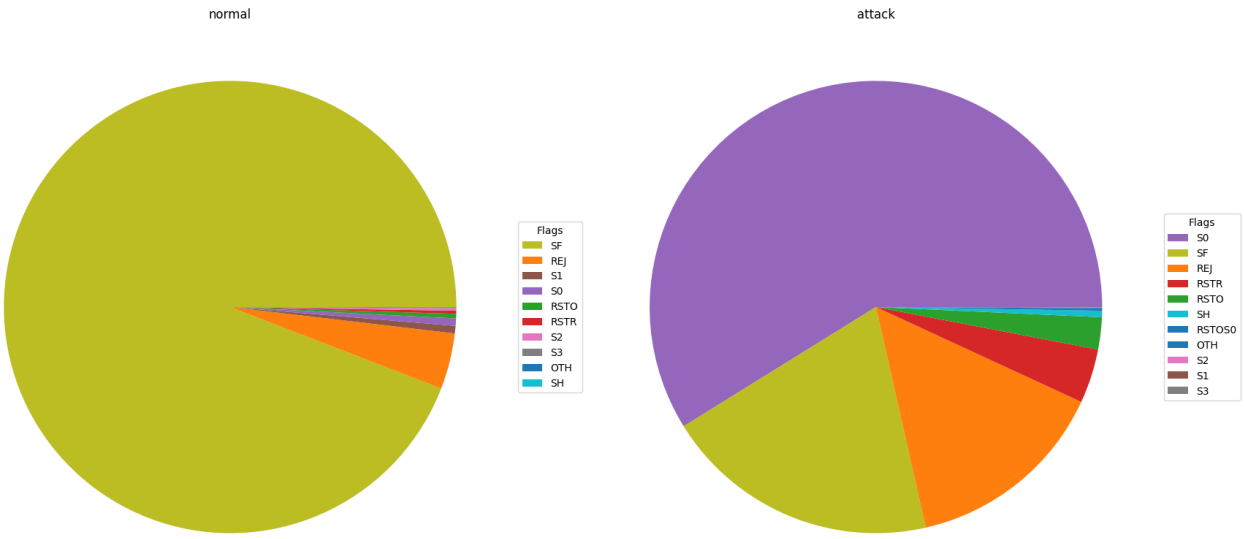
نمودار زیر با استفاده از داده های جدول بالا رسم شده است



نکته قابل توجه در اینجا تفاوت در هر نوع پروتکل است. می بینیم که هر نوع پروتکل بیشتر جهت یک یا چند نوع از حملات به خصوصی استفاده شده است.

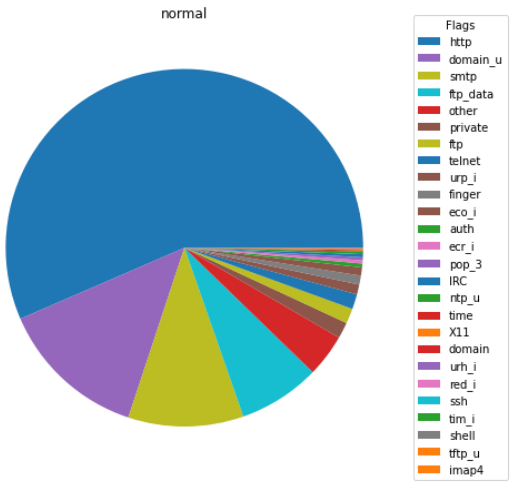
ما در اینجا می توانیم حدس بزنیم که پروتکل ممکن است برای شناسایی نوع ترافیکی که مشاهده می کنیم مفید باشد.

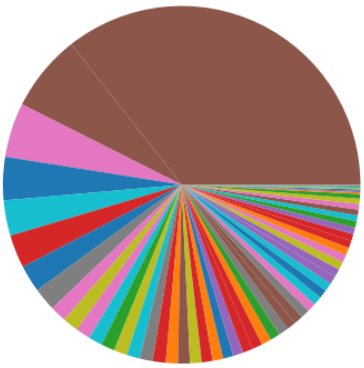
حالا می خواهیم ببینیم که آیا Flag هم می تواند جهت شناسایی نوع ترافیک به ما کمک کند. برای این منظور ما ترافیک های عادی و حمله شبکه را بر اساس فیلد Flag دسته بندی می کنیم و نمودار زیر را به دست می آوریم :



در اینجا مشاهده می کنیم که بیشتر ترافیک های عادی شبکه Flag آن ها برابر SF می باشد و بیشتر ترافیک های حمله هم Flag آن ها برابر S0 است. در نتیجه Flag هم می تواند جهت شناسایی نوع ترافیک مفید باشد.

در مرحله بعد همین کار روی Service نیز انجام می دهیم که نمودار زیر دسته بندی ترافیک های عادی و حمله شبکه را بر اساس فیلد Flag نمایش می دهد.





در اینجا نیز می بینیم که در ترافیک حمله بیشترین Flag برابر Private و در ترافیک عادی بیشترین ترافیک برابر http می باشد.(که قابل انتظار نیز می باشد چرا که بیشتر کاربران عادی جهت دسترسی به صفحات وب از اینترنت استفاده می کنند.)

و نکته مهم تر این است که در حالی که حجم عظیمی از ترافیک عادی http است، چند سرویس در مجموعه حمله وجود دارد یعنی ترافیک حمله ما در همه جا است. به این معنی است که حملات بسیاری مسیرهای مختلف را در سیستم ها جستجو می کنند.

اگر از چشم یک مدیر شبکه به این موضوع فکر کنیم، ترکیب پروتکل، پرچم و سرویس به نظر می رسد که باید چیزهای زیادی در مورد ماهیت ترافیک ما به ما بگوید. جفت کردن آنها با مدت زمان اتصال و مقدار داده در آن اتصال نقطه شروع خوبی برای ما به نظر می رسد.

به نظر می رسد که ویژگی هایی که در مرحله قبل به دست آوردیم یعنی protocol_type، سرویس و پرچم می تواند مفید باشد. تنوع کافی بین اینها وجود دارد که ما باید بتوانیم سطح پایه ای از شناسایی را بدست آوریم. ما همچنین قصد داریم برخی از داده های عددی اولیه را وارد کنیم: مدت زمان، src_bytes، dst_bytes. این ویژگی ها احتمالا باید چیزهای زیادی در مورد آنچه در شبکه ما اتفاق می افتد به ما بگویند.

قبل از اینکه وارد مرحله مدل سازی باید داده های خود کد گذاری کنیم. ما از روش کد گذاری one-hot encoding استفاده می کنیم. روشی است که به ما امکان می‌دهد یک کدگذاری سریع روی ستون‌های خود انجام دهیم. این روش هر مقداری را که در یک ستون پیدا می‌کند را می‌گیرد و برای هر مقدار یک ستون جداگانه با ۰ یا ۱ ایجاد می‌کند، که نشان می‌دهد آیا آن ستون "فعال" است یا خیر.

گام چهارم: مدل سازی یا Modeling

حال که اهداف طبقه بندی خود را تعیین کردیم می توانیم کار مدل سازی را شروع کنیم.

در این مرحله ما از ویژگی هایی که در مرحله قبل به دست آوردیم استفاده می کنیم و داده ها را طبقه بندی می کنیم.

ما روی مجموعه آموزشی خود دو مدل را آموزش می دهیم و تست می کنیم : طبقه بندی دودویی و چندگانه

ما برای طبقه بندی از الگوریتم طبقه بندی جنگل تصادفی (random forest) و k نزدیک ترین همسایه (K-nearest neighbors) استفاده می کنیم.

۱. الگوریتم جنگل تصادفی (Random Forest):

جنگل تصادفی یا Random Forest نوعی الگوریتم یادگیری گروهی (Ensemble Learning Algorithm) است که چندین درخت تصمیم را برای پیش‌بینی ترکیب می‌کند. هر درخت تصمیم در جنگل تصادفی بر روی یک زیر مجموعه تصادفی از داده‌های آموزشی و یک زیر مجموعه تصادفی از فیچرها آموزش داده می‌شود. سپس خروجی جنگل تصادفی با تجمیع پیش‌بینی‌های همه درخت‌های تصمیم تعیین می‌شود. این رویکرد به جنگل تصادفی اجازه می‌دهد تا بسیار دقیق و مقاوم در برابر مشکل بیش‌برازش، یک مشکل رایج در یادگیری ماشین که در آن مدل در داده‌های آموزشی خوب عمل می‌کند اما در داده‌های جدید ضعیف است، باشد.

مراحل الگوریتم جنگل تصادفی

۱. آماده سازی داده‌ها : اولین مرحله آماده سازی داده‌ها برای آموزش مدل است که شامل پاکسازی داده‌ها، حذف مقادیر از دست رفته و تبدیل متغیرهای طبقه‌بندی شده (Categorical) به متغیرهای عددی است. سپس داده ها به مجموعه های آموزشی و آزمایشی تقسیم می شوند.
۲. انتخاب تصادفی زیر مجموعه : مرحله بعدی انتخاب تصادفی زیرمجموعه ای از فیچرها برای هر درخت در جنگل است. این کار برای کاهش بیش‌برازش و افزایش تنوع درختان انجام می‌شود.
۳. ساخت درخت : مرحله بعدی ساخت درخت‌های تصمیم با استفاده از زیرمجموعه‌ای از فیچرهاست که به طور تصادفی انتخاب شده‌اند. درخت‌های تصمیم با استفاده از یک الگوریتم تقسیم باینری بازگشتی

ساخته می‌شوند، که در آن هر گره داخلی داده‌ها را بر اساس یک فیچر و مقدار آستانه انتخاب شده به دو زیر مجموعه تقسیم می‌کند.

۴. رای گیری : هنگامی که تمام درخت های تصمیم ساخته شدند، مرحله بعدی پیش بینی مجموعه تست است. این کار با جمع‌آوری پیش‌بینی‌ها از تمام درختان جنگل با استفاده از مکانیزم رای‌گیری انجام می‌شود. در تسک‌های طبقه‌بندی، کلاسس که توسط اکثریت درختان به عنوان پیش بینی نهایی، پیش‌بینی شده انتخاب می‌شود. در تسک‌های رگرسیون، میانگین پیش‌بینی درختان به عنوان پیش بینی نهایی انتخاب می‌شود.

۵. ارزیابی مدل : در نهایت، عملکرد مدل با استفاده از معیارهای مختلفی مانندAccuracy، precision ، recall و F1 score ارزیابی می‌شود. اگر عملکرد مدل رضایت بخش باشد، می‌توان از آن برای پیش بینی داده‌های جدید استفاده کرد. این مراحل به صورت مکرر، با زیرمجموعه‌های مختلف فیچرها و زیرمجموعه‌های مختلف داده، تکرار می‌شوند تا به سطح مورد نظر از دقت دست پیدا کنند .

Random Forest یک الگوریتم یادگیری ماشین قدرتمند است که می‌تواند وظایف پیچیده را انجام دهد و پیش‌بینی‌های بسیار دقیقی را با حداقل بیش‌برازش ایجاد کند

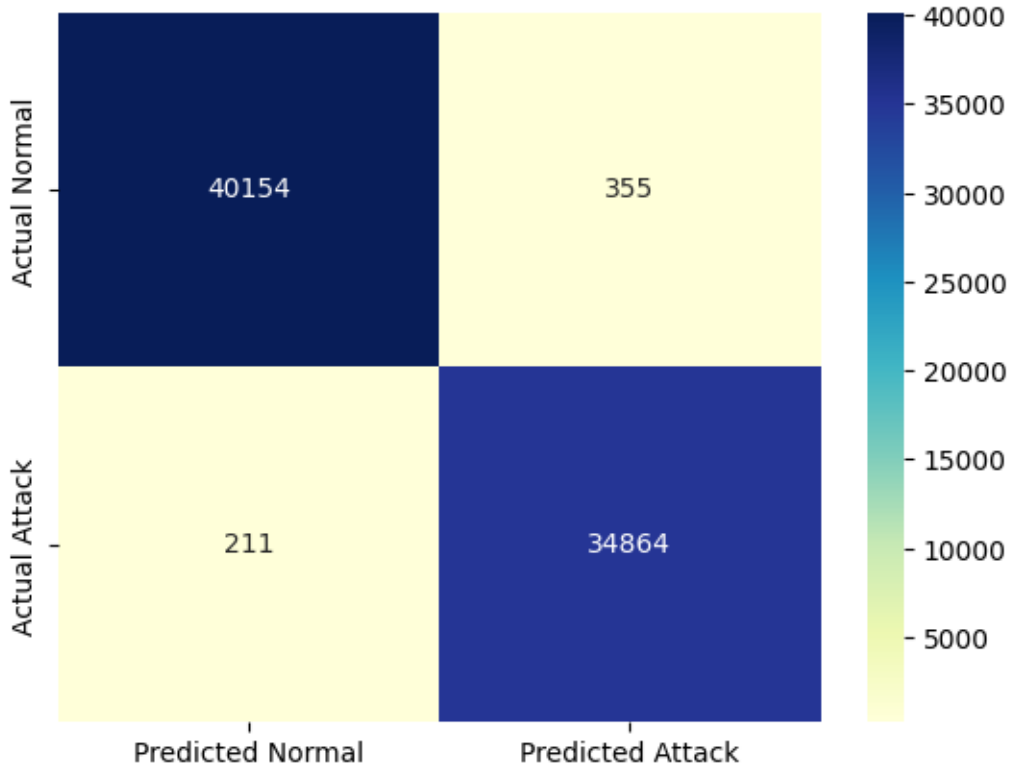
ابتدا ما از الگوریتم random forest استفاده کنیم و با این الگوریتم مدل سازی می کنیم و سپس با استفاده از داده های آموزشی مدل خود را آموزش می دهیم و با استفاده از داده های اعتبار سنجی تست می کنیم و به دقت 0.9925116426756986 دست پیدا می کنیم.

```
[189] # model for the binary classification
      binary_model = RandomForestClassifier()
      binary_model.fit(binary_train_X, binary_train_y)
      binary_predictions = binary_model.predict(binary_val_X)

      # calculate and display our base accuracty
      base_rf_score = accuracy_score(binary_predictions,binary_val_y)
      base_rf_score

      0.9925116426756986
```

همانطور که مشاهده می کنیم الگوریتم random forest به دقت قابل قبولی دست پیدا کرده است.

اکنون می‌توانیم نتایج خود را دقیق‌تر بررسی کنیم تا ببینیم آیا می توانیم به دقت بهتری نیز دست پیدا کنیم یا خیر؟

اولین کاری که می‌توانیم انجام دهیم این است که یک ماتریس سردرگمی را رسم کنیم، که در این مورد طبقه‌بندی پیش‌بینی‌شده را در کنار طبقه‌بندی واقعی ترسیم می کند تا آن ها را با هم مقایسه کنیم:



که به نظر می رسد برخی از حملات را به درستی پیش بینی نکرده است و همچنین برخی از مقادیری را که به عنوان حمله پیش بینی کرده است در واقع حمله نبوده اند.

به عبارتی دیگر ما در اینجا خطای مثبت کاذب زیادی (ترافیک معمولی که به عنوان یک حمله شناسایی شده است) و منفی های کاذب (ترافیک حمله که به صورت عادی شناسایی شده است) را می بینیم.

بنابراین کمی خطاهای پیش بینی را بررسی کنیم تا ببینیم آیا اطلاعات بیشتری برای استخراج وجود دارد یا خیر؟ تا با کمک آن ها دقت مدل سازی خود را تقویت کنیم.

در این مرحله ما برای این که بفهمیم کدام یک از ویژگی ها در مدل سازی اهمیت داشته است و ما آن ویژگی ها را در نظر نگرفته ایم می توانیم از نرخ مثبت کاذب (false positive) و منفی کاذب (false negative) ویژگی ها استفاده کنیم که در شکل های زیر به ترتیب مقدار انحراف معیار (std) مثبت کاذب و منفی کاذب هریک از معیار ها را مشاهده می کنیم:

```
# see the standard deviation of the false positives
binary_prediction_data['false_positives'].std()
```

```
<ipython-input-194-809ec2f8632b>:2: FutureWarning: The de
  binary_prediction_data['false_positives'].std()
duration                          595.441628
src_bytes                        2923.291051
dst_bytes                      605539.572944
land                                0.129084
urgent                              0.000000
hot                                 0.139911
num_failed_logins                   0.000000
logged_in                           0.148626
num_compromised                     0.480461
root_shell                          0.053074
su_attempted                        0.000000
num_root                            0.477670
num_file_creations                  0.053074
num_shells                          0.000000
num_access_files                    0.000000
is_guest_login                      0.000000
count                               4.452383
srv_count                           4.172750
serror_rate                         0.326623
srv_serror_rate                     0.316181
rerror_rate                         0.437145
srv_rerror_rate                     0.433947
same_srv_rate                       0.117579
diff_srv_rate                       0.102166
srv_diff_host_rate                  0.272373
dst_host_count                     90.213694
dst_host_srv_count                106.237251
dst_host_same_srv_rate              0.377705
dst_host_diff_srv_rate              0.101081
dst_host_same_src_port_rate         0.320210
dst_host_srv_diff_host_rate         0.132350
dst_host_serror_rate                0.283925
dst_host_srv_serror_rate            0.182586
dst_host_rerror_rate                0.301523
dst_host_srv_rerror_rate            0.373862
level                               3.909007
attack_flag                         0.000000
attack_map                          0.000000
predicted                           0.000000
actual                              0.000000
dtype: float64
```

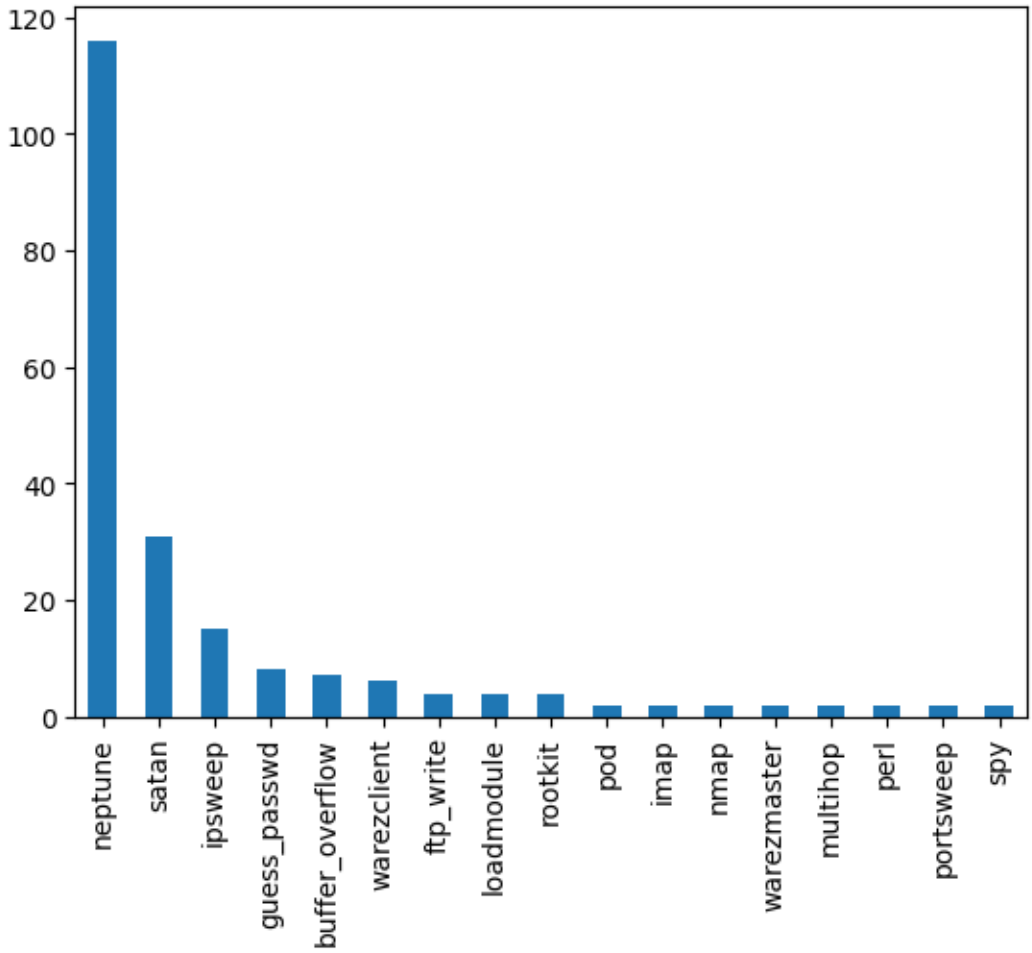
```
# see the standard deviation of the false negatives
binary_prediction_data['false_negatives'].std()
```

```
<ipython-input-195-1109220fab1a>:2: FutureWarning: The de
  binary_prediction_data['false_negatives'].std()
duration                          1048.301599
src_bytes                       609361.248413
dst_bytes                        44966.749563
land                                 0.000000
urgent                               0.068843
hot                                  1.790746
num_failed_logins                    0.386175
logged_in                            0.389086
num_compromised                      3.239350
root_shell                           0.152464
su_attempted                         0.068843
num_root                             4.727105
num_file_creations                   1.577631
num_shells                           0.193087
num_access_files                     0.153644
is_guest_login                       0.136699
count                              122.631186
srv_count                            6.142133
serror_rate                          0.155657
srv_serror_rate                      0.169730
rerror_rate                          0.458935
srv_rerror_rate                      0.477820
same_srv_rate                        0.448537
diff_srv_rate                        0.274056
srv_diff_host_rate                   0.118670
dst_host_count                     103.273703
dst_host_srv_count                  25.013822
dst_host_same_srv_rate               0.316666
dst_host_diff_srv_rate               0.281491
dst_host_same_src_port_rate          0.297791
dst_host_srv_diff_host_rate          0.234940
dst_host_serror_rate                 0.111448
dst_host_srv_serror_rate             0.144366
dst_host_rerror_rate                 0.451557
dst_host_srv_rerror_rate             0.474081
level                                7.567850
attack_flag                          0.000000
attack_map                           1.054164
predicted                            0.000000
actual                               0.000000
dtype: float64
```

همانطور که مشاهده می کنیم اکثر معیار ها در نرخ مثبت کاذب انحراف معیار کمی دارند اما در منفی های کاذب، همه سطرها درجاتی از واریانس دارند. این به ما نشان می دهد که ممکن است اطلاعات خوبی در آن سطر ها وجود داشته باشد زیرا بین مشاهدات یک طبقه بندی در مقابل طبقه دیگر تفاوت وجود دارد.

پس به موارد منفی کاذب توجه می کنیم تا ببینیم چه نوع حملاتی را از دست داده ایم.

شکل زیر نوع حملات با بیشترین مقدار منفی کاذب را نشان می دهد.



Neptune و Satan بیشترین نوع حملات از دست رفته اند. (حملاتی که شناسایی نشده اند.)

در این مرحله ما تلاش می کنیم که بتوانیم این نوع حملات را نیز به درستی طبقه بندی کنیم یا به عبارتی نرخ منفی کاذب آن ها را پایین بیاوریم.

به این منظور ما ویژگی هایی که بیشترین نرخ منفی کاذب را دارند را دست می آوریم که در شکل زیر به ترتیب آورده شده اند:

```
# we'll need to pull these from the data set
outcomes = ['attack_flag','attack_map', 'actual']

# get the new features we're interested in and drop the outcomes
new_features = (binary_prediction_data['false_positives']==0).all(axis=0)
feature_cols = binary_prediction_data['false_positives'].loc[:,new_features]
feature_cols = feature_cols.drop(outcomes,axis=1)

# Let's get these in a list and take a look
new_feature_columns = list(feature_cols.columns)
new_feature_columns
```

```
['urgent',
 'num_failed_logins',
 'su_attempted',
 'num_shells',
 'num_access_files',
 'is_guest_login']
```

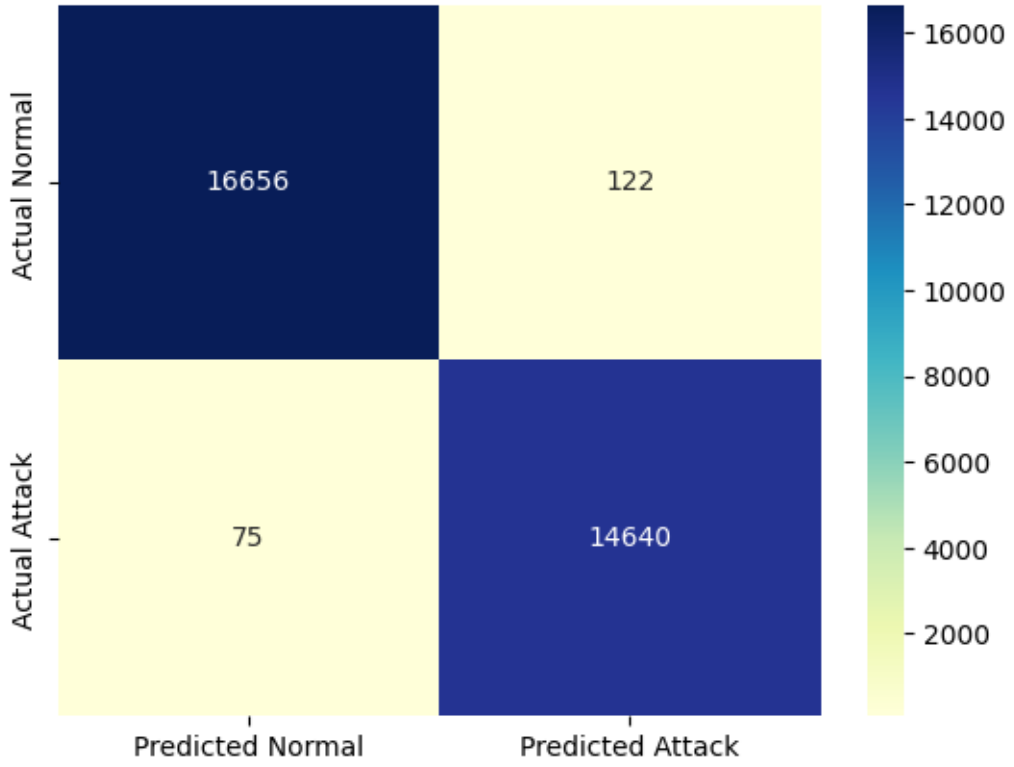
ما ویژگی های فوق را نیز به مجمعه ویژگی های خود جهت طبقه بندی اضافه می کنیم و مجددا مدل را آوزش می دهیم. این بار روی داده های اعتبارسنجی ما به دقتی برابر 0.9937446416664021 دست پیدا می کنیم که می بینیم دقت ما نسبت به مرحله قبل بهبود پیدا کرده است.

```
[198] # add the new features
      to_fit_new_features = to_fit.join(df[new_feature_columns])

      # build the training sets
      new_feature_train_X, new_feature_val_X, new_feature_train_y, new_feature_val_y = train_test_split(to_fit_new_features, binary_y)
```

```
# model for the binary classification
new_feature_model = RandomForestClassifier()
new_feature_model.fit(new_feature_train_X, new_feature_train_y)
new_feature_predictions = new_feature_model.predict(new_feature_val_X)

# get the score for the model
new_feature_score = accuracy_score(new_feature_predictions,new_feature_val_y)

new_feature_score
```

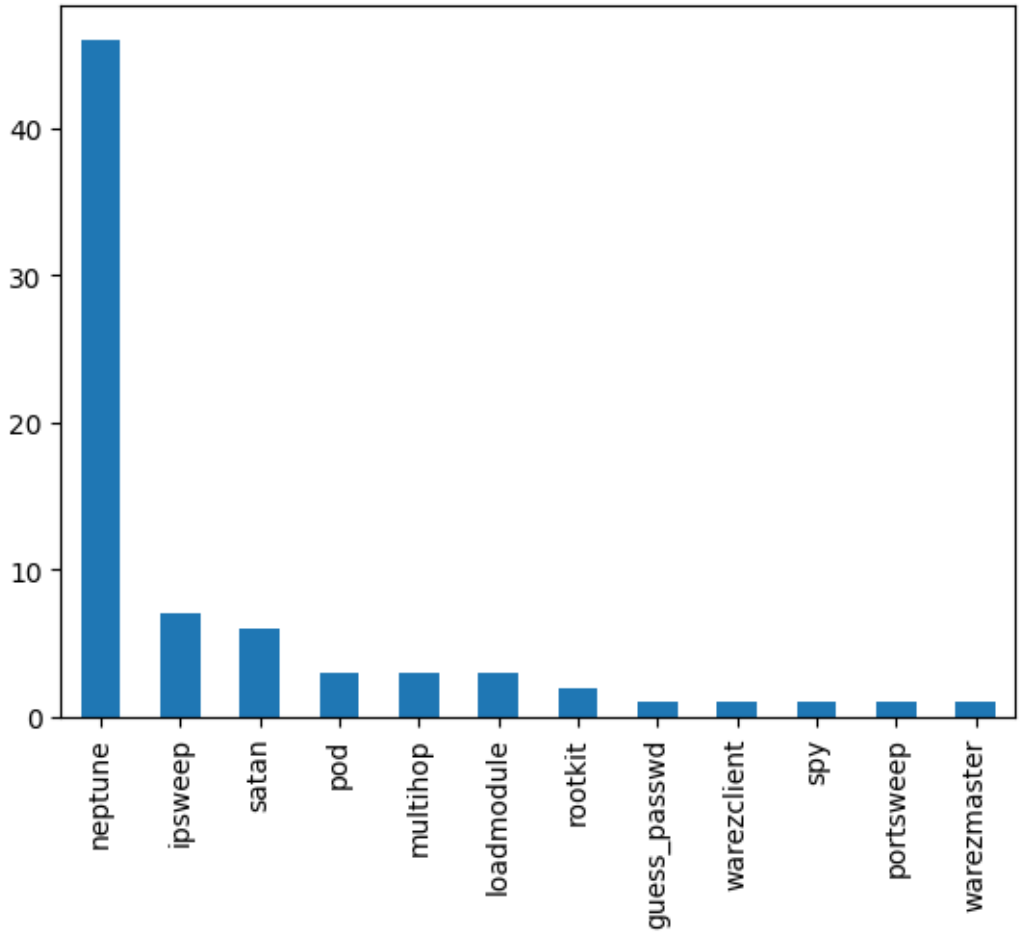
```
0.9937446416664021
```

مجددا جهت مشاهده پیش بینی های از دست رفته ماتریس سردرگمی (confustion matrix) را رسم می کنیم.



همانطور که می بینیم نرخ مثبت و منفی کاذب کمتری را شاهد هستیم و این به معنای بهبود عملکرد الگوریتم طبقه بندی ما می باشد.

مجددا نوع حملات با بیشترین مقدار منفی کاذب را رسم می کنیم.



به نظر می رسد که عملکرد الگوریتم مدل سازی ما بهتر شده و ما از هرکدام از انواع حملات فقط چند مورد خاص را به درستی شناسایی نکرده ایم. البته حمله Neptune همچنان نرخ منفی کاذب زیادی دارد، که به نظر می رسد شناسایی این نوع حمله برای الگوریتم سخت می باشد.

در مرحله بعد، توجه خود را به سناریوی چند طبقه بندی معطوف می کنیم. در اینجا می خواهیم ببینیم که آیا می توانیم نوع حمله را از روی داده ها شناسایی کنیم. به یاد داشته باشید، ما چهار نوع حمله داریم:

DOS

Probe

Privilege escalation

Remote access

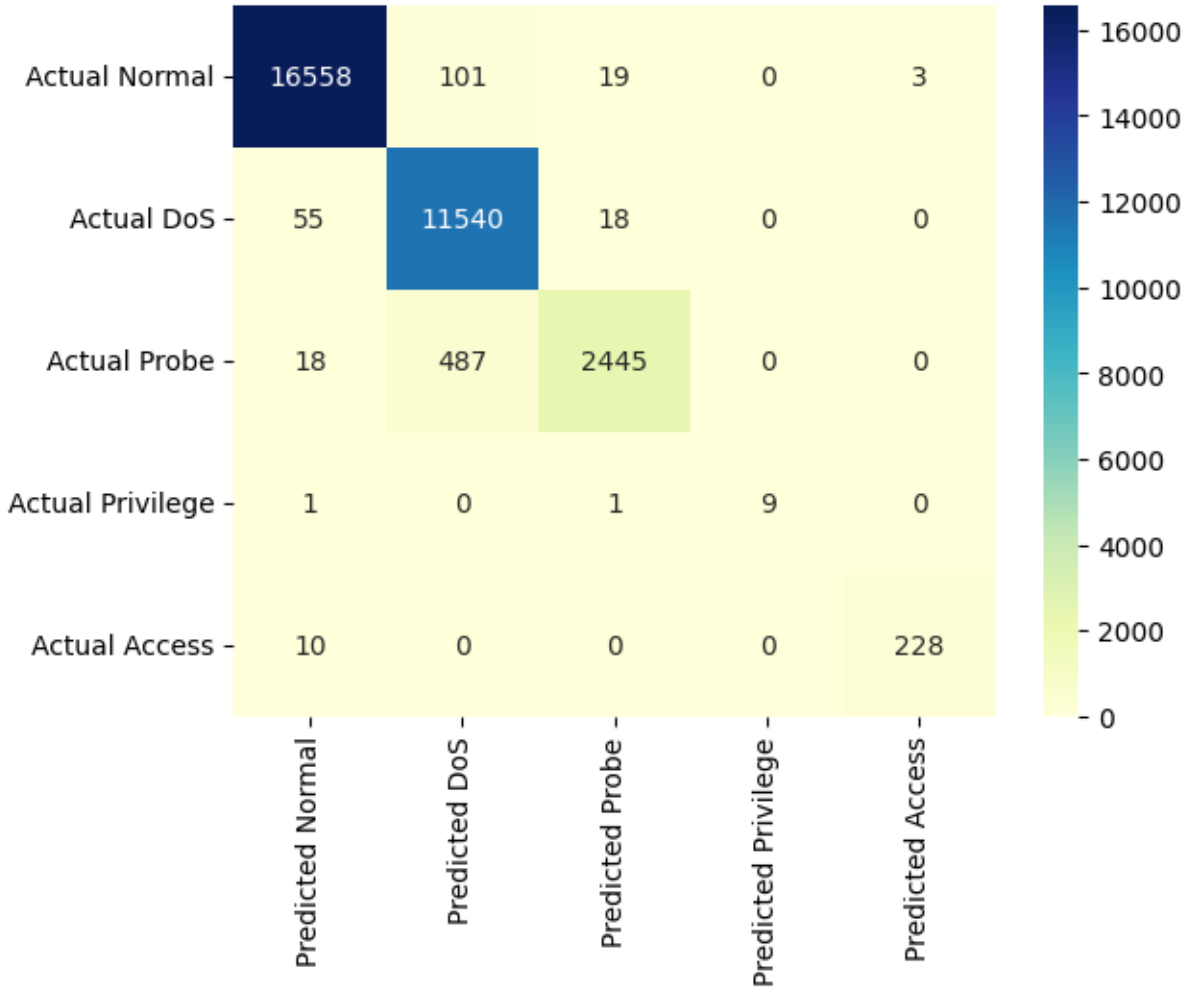
ما مدل پایه خود را بر اساس الگوریتم جنگل تصادفی می سازیم و با استفاده از داده های آموزشی مدل خود را آموزش می دهیم و با استفاده از داده های اعتبار سنجی تست می کنیم و به دقت 0.976489733276884 دست پیدا می کنیم.

```
[202] # model for the mulit classification
      multi_model = RandomForestClassifier()
      multi_model.fit(multi_train_X, multi_train_y)
      multi_predictions = multi_model.predict(multi_val_X)

      # get the score
      accuracy_score(multi_predictions,multi_val_y)

      0.976489733276884
```

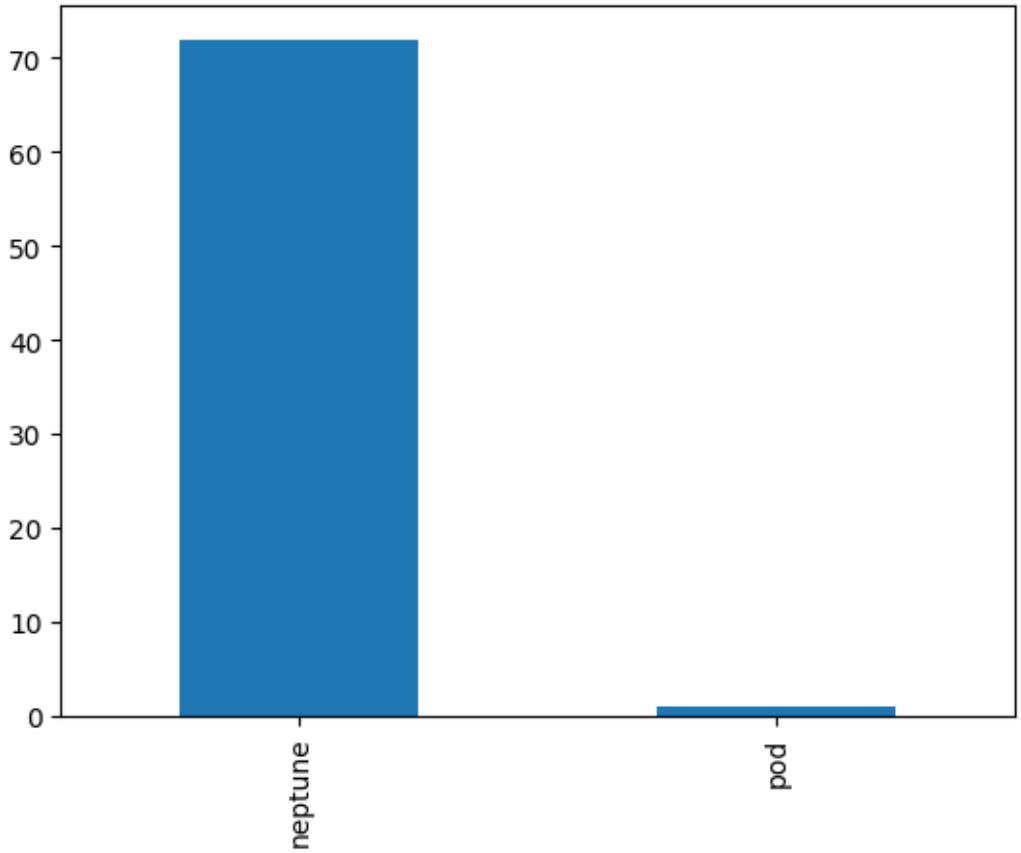
سپس ویژگی های جدیدی که در مرحله قبل به دست آوردیم را به مدل اضافه می کنیم و به دقت 0.9773600482646937 دست پیدا می کنیم و همانطور که مشاهده می کنیم دقت ما بهبود پیدا کرده است.

```
[203] # build the training sets
      multi_feature_train_X, multi_feature_val_X, multi_feature_train_y, multi_feature_val_y = train_test_split(to_fit_new_features, multi_y)

      # model for the mulit classification
      multi_model = RandomForestClassifier()
      multi_model.fit(multi_feature_train_X, multi_feature_train_y)
      multi_predictions = multi_model.predict(multi_feature_val_X)

      # get the score
      accuracy_score(multi_predictions,multi_feature_val_y)

      0.9773600482646937
```

جهت بررسی بیشتر مدل خود، ما ماتریس سردرگمی را برای طبقه بندی چند کلاسه (طبقه بندی انواع حمله) رسم خواهیم کرد. این ماتریس نوع حمله پیش بینی شده را بر اساس نوع حمله واقعی ترسیم می کند.



شکل زیر بیشترین نوع حملاتی را که به درستی طبقه بندی نشده اند را برای دسته بندی چند کلاسه نشان می دهد.



۲- الگوریتم K-nearest neighbors :

الگوریتم K نزدیکترین همسایه (K-Nearest Neighbors) یکی از این الگوریتم های توانمند است. الگوریتم K نزدیکترین همسایه (KNN) یک الگوریتم یادگیری ماشین نظارت شده است و از ویژگی های آن سادگی و آسانی این الگوریتم برای پیاده سازی است که می تواند برای حل مسائل طبقه بندی و رگرسیون مورد استفاده قرار گیرد. الگوریتم نزدیک ترین همسایگی کاربرد فراوانی در داده کاوی دارد و یک الگوریتم بسیار محبوب در این زمینه است.

الگوریتم K نزدیک ترین همسایه یا KNN یکی از ساده‌ترین الگوریتم‌های یادگیری ماشین باناظر است که برای حل مسائل طبقه‌بندی و رگرسیون استفاده می‌شود.

این الگوریتم برای مسائل طبقه‌بندی k نزدیک ترین همسایه را پیدا و با اکثریت آرا نزدیک‌ترین همسایگان کلاس را پیش‌بینی می‌کند.

برای مسائل رگرسیون k نزدیک‌ترین همسایه را پیدا و با محاسبه‌ی میانگین مقدار نزدیک‌ترین همسایه‌ها، مقدار مدنظر را پیش‌بینی می‌کند.

مراحل الگوریتم K نزدیک‌ترین همسایه

۱. داده‌ها را بارگذاری می‌کنیم.

۲. مقدار K را تعیین می‌کنیم که همان تعداد نزدیک‌ترین همسایه‌ها هستند.

۳. برای هر نمونه داده:

- فاصله‌ی میان نمونه داده‌ی جدید را با نمونه داده‌های موجود محاسبه می‌کنیم.
- فاصله و شاخص هر نمونه را به یک فهرست وارد می‌کنیم.

۴. کل لیست را براساس فاصله‌ی نمونه داده‌ها، از کمترین به بیشترین فاصله، مرتب می‌کنیم.

۵. K تا از اولین نمونه‌های فهرست مرتب‌شده را به‌عنوان K نزدیک‌ترین همسایه انتخاب می‌کنیم.

۶. برچسب این K نمونه را بررسی می‌کنیم.

۷. اگر مسئله رگرسیون باشد، میانگین برچسب‌های این K نمونه داده برچسب نمونه داده جدیدمان خواهد بود.

۸. درصورتی‌که مسئله طبقه‌بندی باشد، نمونه‌ی جدید هم همان برچسب K همسایه را خواهد داشت.

در این مرحله ما از الگوریتم KNN استفاده کنیم و با این الگوریتم مدل سازی می کنیم و سپس با استفاده از داده های آموزشی مدل خود را آموزش می دهیم و با استفاده از داده های اعتبار سنجی تست می کنیم.

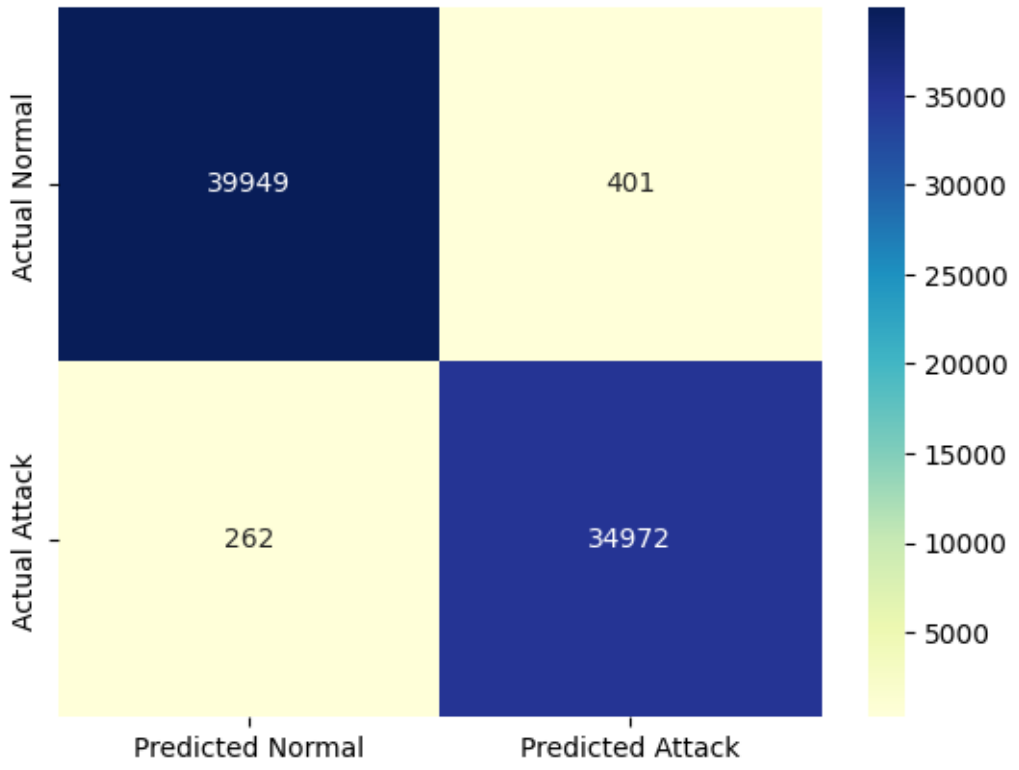
در مرحله اول ما الگوریتم KNN را با پارامترهای پیش فرض خود الگوریتم اجرا می کنیم و به دقتی معادل 0.9912283022861982 دست پیدا می کنیم.

```
# model for the binary classification
binary_model = KNeighborsClassifier()
binary_model.fit(binary_train_X, binary_train_y)
binary_predictions = binary_model.predict(binary_val_X)

# calculate and display our base accuracty
base_rf_score = accuracy_score(binary_predictions,binary_val_y)
base_rf_score
```

```
0.9912283022861982
```

ما سپس تلاش می کنیم که با تغییر پارامترها و یا ویژگی های دقت الگوریتم را بهبود ببخشیم.

مشابه کاری که در الگوریتم random forest انجام دادیم ابتدا ماتریس سردرگمی را رسم کنیم، که مورد طبقه‌بندی پیش‌بینی‌شده را در کنار طبقه بندی واقعی ترسیم می کند تا آن ها را با هم مقایسه کنیم:



که به نظر می رسد برخی از حملات را به درستی پیش بینی نکرده است و همچنین برخی از مقادیری را که به عنوان حمله پیش بینی کرده است در واقع حمله نبوده اند.

به عبارتی دیگر ما در آنجا نکات مثبت کاذب زیادی (ترافیک معمولی که به عنوان یک حمله شناسایی شده است) و منفی های کاذب (ترافیک حمله که به صورت عادی شناسایی شده است) را می بینیم.

بنابراین کمی خطاهای پیش بینی را بررسی کنیم تا ببینیم آیا اطلاعات بیشتری برای استخراج وجود دارد یا خیر؟ تا با کمک آن ها دقت مدل سازی خود را تقویت کنیم.

در این مرحله ما برای این که بفهمیم کدام یک از ویژگی ها در مدل سازی اهمیت داشته است و ما آن ویژگی ها را در نظر نگرفته ایم می توانیم از نرخ مثبت کاذب (false positive) و منفی کاذب (false negative) ویژگی ها استفاده کنیم که در شکل های زیر به ترتیب مقدار انحراف معیار (std) مثبت کاذب و منفی کاذب هریک از معیار ها را مشاهده می کنیم:

```
[31] # see the standard deviation of the false positives
     binary_prediction_data['false_positives'].std()
```

```
<ipython-input-31-809ec2f8632b>:2: FutureWarning: The d
  binary_prediction_data['false_positives'].std()
duration                          5414.593938
src_bytes                       428785.281845
dst_bytes                       575239.970812
land                                 0.111104
wrong_fragment                       0.000000
urgent                               0.000000
hot                                  0.122067
num_failed_logins                    0.157798
logged_in                            0.282600
num_compromised                      0.900139
root_shell                           0.049938
su_attempted                         0.111552
num_root                             0.449439
num_file_creations                   0.049938
num_shells                           0.000000
num_access_files                     0.000000
is_guest_login                       0.000000
count                                3.807600
srv_count                            4.288823
serror_rate                          0.316645
srv_serror_rate                      0.299130
rerror_rate                          0.410966
srv_rerror_rate                      0.409993
same_srv_rate                        0.111114
diff_srv_rate                        0.129156
srv_diff_host_rate                   0.278659
dst_host_count                      96.392736
dst_host_srv_count                 113.242236
dst_host_same_srv_rate               0.403053
dst_host_diff_srv_rate               0.194594
dst_host_same_src_port_rate          0.359803
dst_host_srv_diff_host_rate          0.125888
dst_host_serror_rate                 0.268309
dst_host_srv_serror_rate             0.134544
dst_host_rerror_rate                 0.306342
dst_host_srv_rerror_rate             0.380007
level                                3.620395
attack_flag                          0.000000
attack_map                           0.000000
predicted                            0.000000
actual                               0.000000
dtype: float64
```

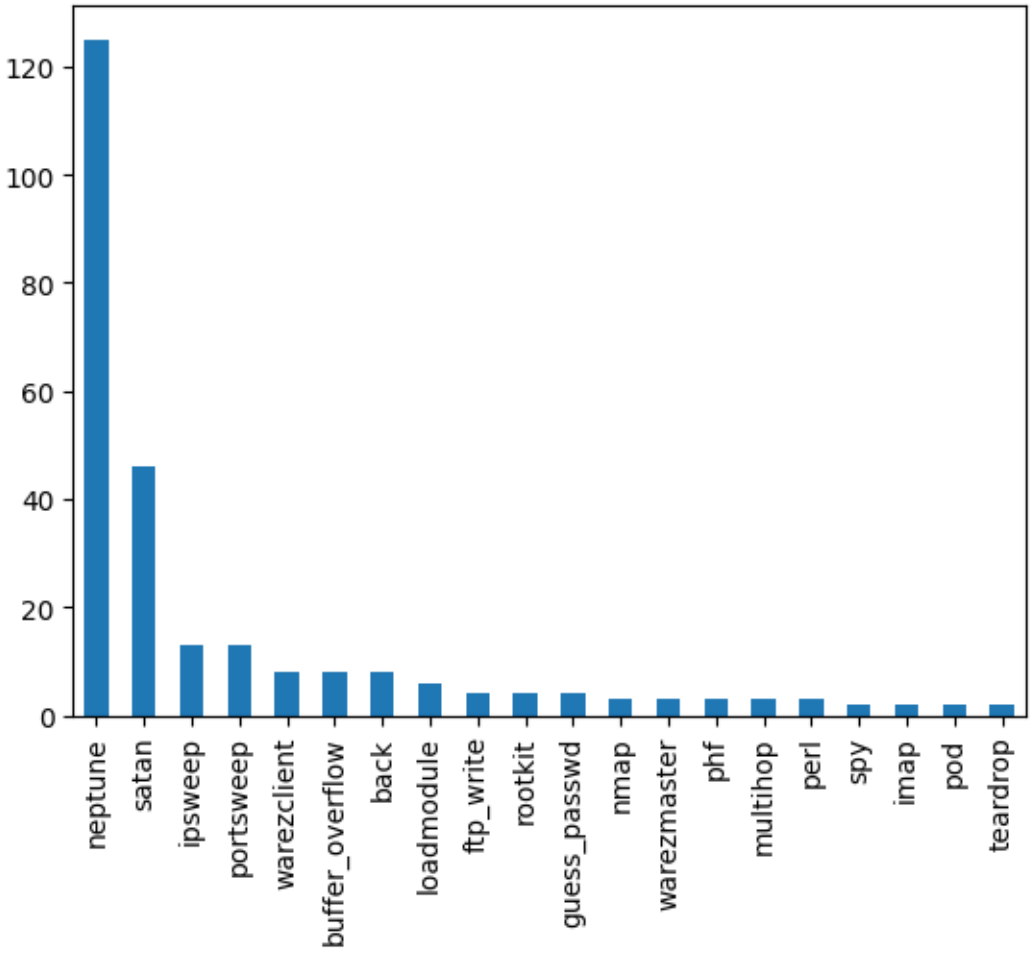
```
# see the standard deviation of the false negatives
binary_prediction_data['false_negatives'].std()
```

```
<ipython-input-32-1109220fab1a>:2: FutureWarning: The def
  binary_prediction_data['false_negatives'].std()
duration                        1.843325e+03
src_bytes                       1.480659e+06
dst_bytes                       1.019525e+05
land                            0.000000e+00
wrong_fragment                  2.616086e-01
urgent                          1.508460e-01
hot                             1.582904e+00
num_failed_logins               3.205179e-01
logged_in                       4.327319e-01
num_compromised                 2.393681e+00
root_shell                      2.253300e-01
su_attempted                    6.178021e-02
num_root                        3.467292e+00
num_file_creations              1.422693e+00
num_shells                      2.205083e-01
num_access_files                2.023884e-01
is_guest_login                  1.498735e-01
count                           1.272109e+02
srv_count                       9.904900e+00
serror_rate                     1.115487e-01
srv_serror_rate                 1.527367e-01
rerror_rate                     4.690116e-01
srv_rerror_rate                 4.894234e-01
same_srv_rate                   4.622197e-01
diff_srv_rate                   2.818243e-01
srv_diff_host_rate              1.187984e-01
dst_host_count                  1.002477e+02
dst_host_srv_count              4.882478e+01
dst_host_same_srv_rate          3.597455e-01
dst_host_diff_srv_rate          2.881144e-01
dst_host_same_src_port_rate     3.584824e-01
dst_host_srv_diff_host_rate     1.992104e-01
dst_host_serror_rate            1.033519e-01
dst_host_srv_serror_rate        1.196107e-01
dst_host_rerror_rate            4.533019e-01
dst_host_srv_rerror_rate        4.917370e-01
level                           7.398164e+00
attack_flag                     0.000000e+00
attack_map                      1.017009e+00
predicted                       0.000000e+00
actual                          0.000000e+00
dtype: float64
```

همانطور که مشاهده می کنیم اکثر معیار ها در نرخ مثبت کاذب انحراف معیار کمی دارند اما در منفی های کاذب، همه ستون ها درجاتی از واریانس دارند. این به ما نشان می دهد که ممکن است اطلاعات خوبی در آن ستون ها وجود داشته باشد زیرا بین مشاهدات یک طبقه بندی در مقابل طبقه دیگر تفاوت وجود دارد.

پس به موارد منفی کاذب توجه می کنیم تا ببینیم چه نوع حملاتی را از دست داده ایم.

شکل زیر نوع حملات با بیشترین مقدار منفی کاذب را نشان می دهد.



Neptune و Satan بیشترین نوع حملات از دست رفته اند. (حملاتی که شناسایی نشده اند.)

در این مرحله ما تلاش می کنیم که بتوانیم این نوع حملات را نیز به درستی طبقه بندی کنیم یا به عبارتی نرخ منفی کاذب آن ها را پایین بیاوریم.

به این منظور ما ویژگی هایی که بیشترین نرخ منفی کاذب را دست می آوریم که در جدول زیر به ترتیب آورده شده اند:

```
# we'll need to pull these from the data set
outcomes = ['attack_flag','attack_map', 'actual']

# get the new features we're interested in and drop the outcomes
new_features = (binary_prediction_data['false_positives']==0).all(axis=0)
feature_cols = binary_prediction_data['false_positives'].loc[:,new_features]
feature_cols = feature_cols.drop(outcomes,axis=1)

# Let's get these in a list and take a look
new_feature_columns = list(feature_cols.columns)
new_feature_columns
```

```
['wrong_fragment',
 'urgent',
 'num_shells',
 'num_access_files',
 'is_guest_login']
```

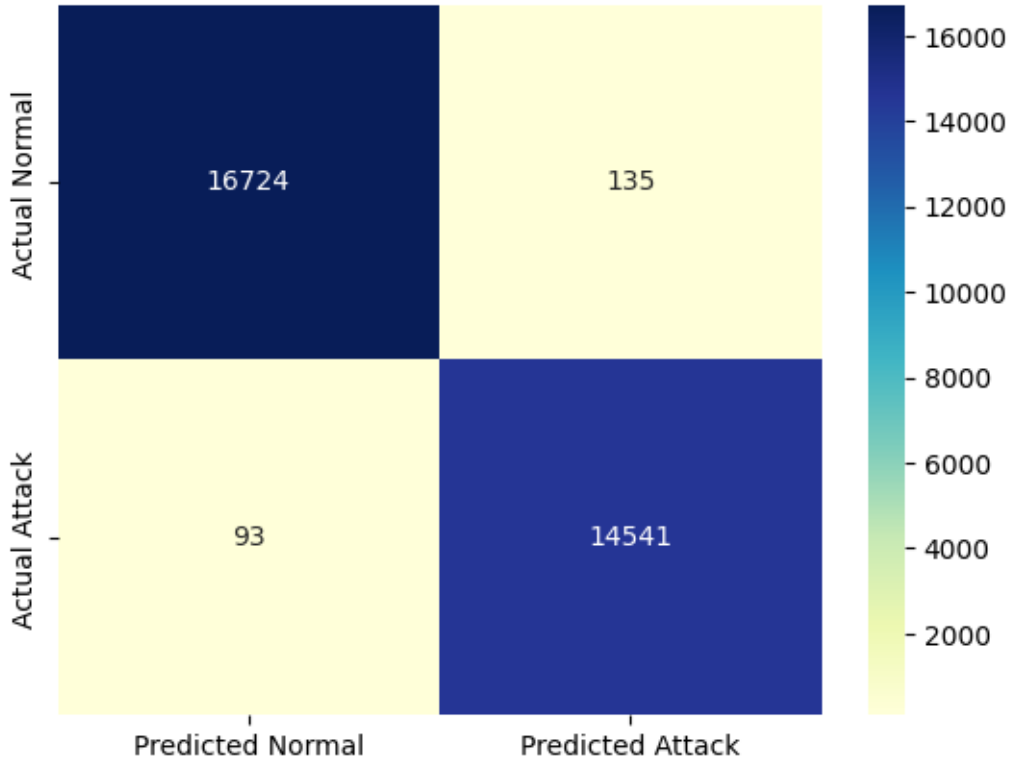
ما ویژگی های فوق را نیز به مجمعه ویژگی های خود جهت طبقه بندی اضافه می کنیم و مجددا مدل را آموزش می دهیم. این بار روی داده های اعتبارسنجی ما به دقتی برابر 0.99276029593878 دست پیدا می کنیم که می بینیم دقت ما نسبت به مرحله قبل بهبود پیدا کرده است.

```
[35] # add the new freatures
     to_fit_new_features = to_fit.join(df[new_feature_columns])

     # build the training sets
     new_feature_train_X, new_feature_val_X, new_feature_train_y, new_feature_val_y = train_test_split(to_fit_new_features, binary_y)
```

```
[36] # model for the binary classification
     new_feature_model = KNeighborsClassifier()
     new_feature_model.fit(new_feature_train_X, new_feature_train_y)
     new_feature_predictions = new_feature_model.predict(new_feature_val_X)

     # get the score for the model
     new_feature_score = accuracy_score(new_feature_predictions,new_feature_val_y)

     new_feature_score
```

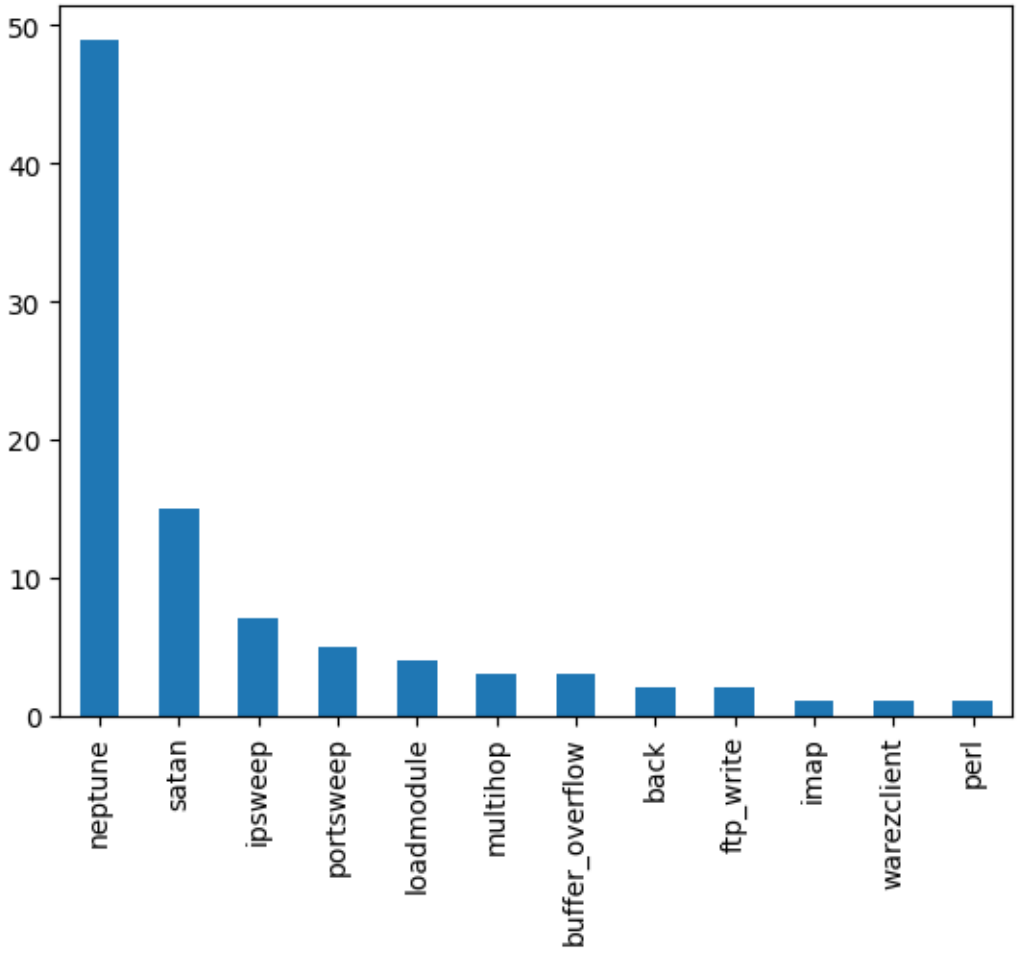
```
0.99276029593878
```

مجددا جهت مشاهده پیش بینی های از دست رفته ماتریس سردرگمی (confustion matrix) را رسم می کنیم.



همانطور که می بینیم نرخ مثبت و منفی کاذب کمتری را شاهد هستیم و این به معنای بهبود عملکرد الگوریتم طبقه بندی ما می باشد.

مجددا نوع حملات با بیشترین مقدار منفی کاذب را رسم می کنیم.



در مرحله بعد ما جهت بهبود عملکرد الگوریتم طبقه بندی خود، سعی می کنیم برخی پارامترهای الگوریتم KNN را تغییر دهیم.

همانطور که می دانیم مقادیر پیش فرض الگوریم KNN به شرح زیر است :

Weights = ’uniform’

Algorithm = ’auto’

Metric = ’minkowski’

در مرحله اول ویژگی metric='manhattan' قرار می دهیم و به دقتی 0.9928555552027435 برابر دست پیدا می کنیم که بهتر می باشد.

```
[39] # model for the binary classification
     new_feature_model = KNeighborsClassifier(metric='manhattan')
     new_feature_model.fit(new_feature_train_X, new_feature_train_y)
     new_feature_predictions = new_feature_model.predict(new_feature_val_X)

     # get the score for the model
     new_feature_score = accuracy_score(new_feature_predictions,new_feature_val_y)

     new_feature_score

     0.9928555552027435
```

در مرحله دوم ویژگی weights='distance' قرار می دهیم و به دقتی 0.9931413329946337 برابر دست پیدا می کنیم که بهتر می باشد.

```
# model for the binary classification
new_feature_model = KNeighborsClassifier(weights='distance')
new_feature_model.fit(new_feature_train_X, new_feature_train_y)
new_feature_predictions = new_feature_model.predict(new_feature_val_X)

# get the score for the model
new_feature_score = accuracy_score(new_feature_predictions,new_feature_val_y)

new_feature_score

0.9931413329946337
```

در مرحله سوم ویژگی algorithm='brute' قرار می دهیم که دقت مدل تفاوت چندانی نمی کند.

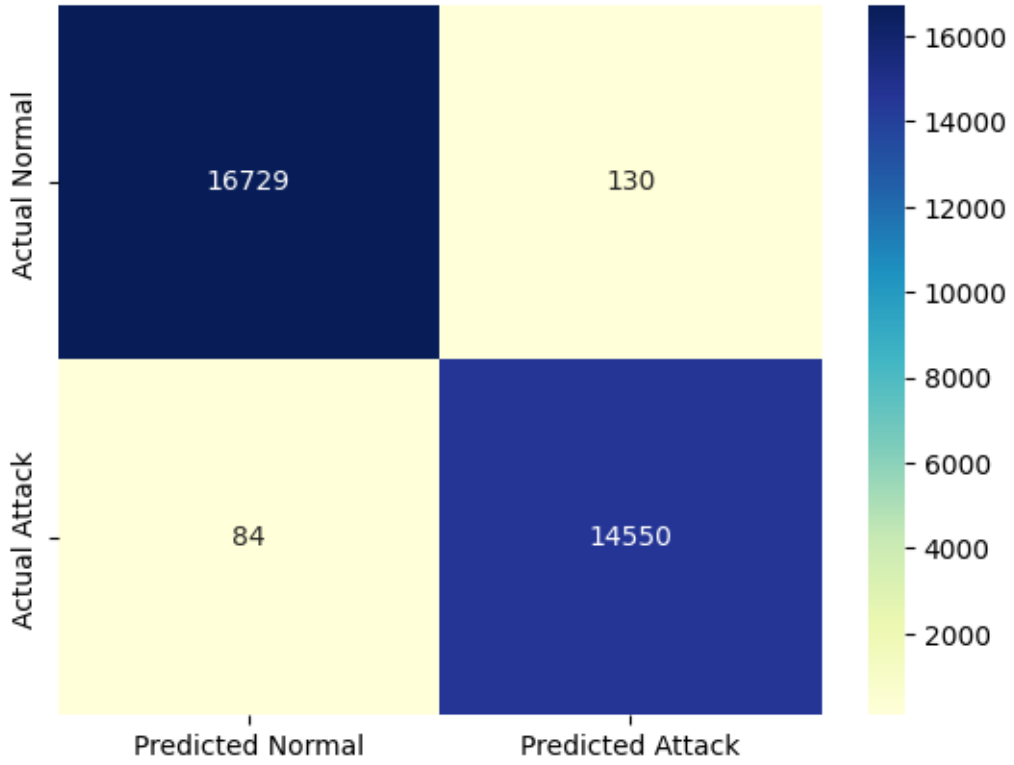
```
[41] # model for the binary classification
     new_feature_model = KNeighborsClassifier(algorithm='brute')
     new_feature_model.fit(new_feature_train_X, new_feature_train_y)
     new_feature_predictions = new_feature_model.predict(new_feature_val_X)

     # get the score for the model
     new_feature_score = accuracy_score(new_feature_predictions,new_feature_val_y)

     new_feature_score

     0.99276029593878
```

در مرحله آخر، ویژگی های metric و weights را تنظیم می کنیم و با آن ها مدل را آموزش می دهیم و ارزیابی می کنیم و به دقت 0.9932048391706093 که دقت قابل قبولی می باشد.

```
# model for the binary classification
new_feature_model = KNeighborsClassifier(metric='manhattan', weights='distance')
new_feature_model.fit(new_feature_train_X, new_feature_train_y)
new_feature_predictions = new_feature_model.predict(new_feature_val_X)

# get the score for the model
new_feature_score = accuracy_score(new_feature_predictions,new_feature_val_y)

new_feature_score
```

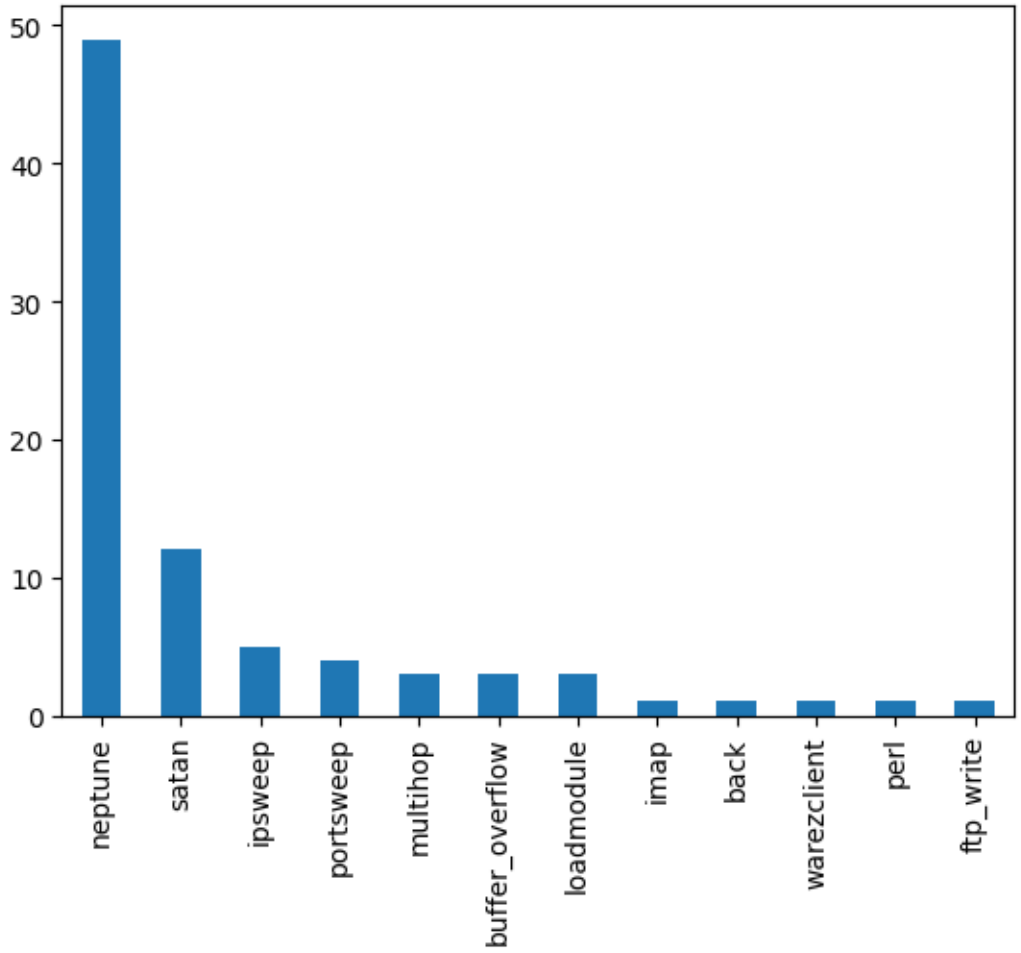
```
0.9932048391706093
```

شکل زیر ماتریس سردرگمی نهایی را برای دسته بندی دودویی نمایش می دهد:



شکل زیر نیز نوع حملات با بیشترین مقدار منفی کاذب را نشان می دهد

همان طور که می بینیم کاهش داشته است.



به نظر می رسد که عملکرد الگوریتم مدل سازی ما بهتر شده و ما از هرکدام از انواع حملات فقط چند مورد خاص را به درستی شناسایی نکرده ایم. البته حمله Neptune همچنان نرخ منفی کاذب زیادی دارد، که به نظر می رسد شناسایی این نوع حمله برای الگوریتم سخت می باشد.

در مرحله بعد، توجه خود را به سناریوی چند طبقه بندی معطوف می کنیم. در اینجا می خواهیم ببینیم که آیا می توانیم نوع حمله را از روی داده ها شناسایی کنیم. به یاد داشته باشید، ما چهار نوع حمله داریم:

DOS

Probe

Privilege escalation

Remote access

ما مدل پایه خود را بر اساس الگوریتم KNN می سازیم و با استفاده از داده های آموزشی مدل خود را آموزش می دهیم و با استفاده از داده های اعتبار سنجی تست می کنیم و به دقت 0.9756429932260796 دست پیدا می کنیم.

```
[56]
      # model for the mulit classification
      multi_model = KNeighborsClassifier(metric='manhattan', weights='distance')
      multi_model.fit(multi_train_X, multi_train_y)
      multi_predictions = multi_model.predict(multi_val_X)

      # get the score
      accuracy_score(multi_predictions,multi_val_y)

      0.9756429932260796
```

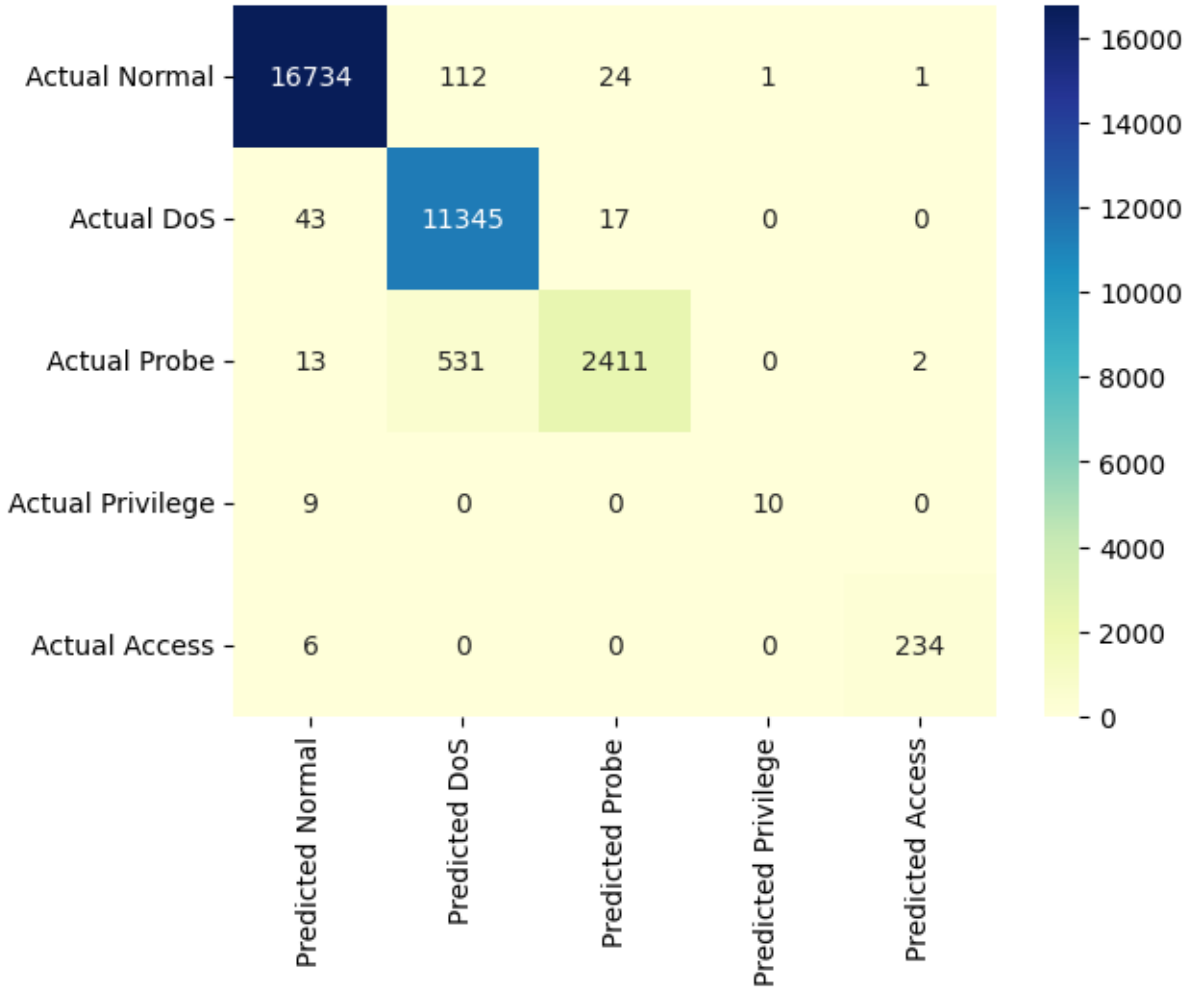
سپس ویژگی های جدیدی که در مرحله قبل به دست آوردیم را به مدل اضافه می کنیم و به دقت 0.9758994062172547 دست پیدا می کنیم و همانطور که مشاهده می کنیم دقت ما بهبود پیدا کرده است.

```
[60] # model for the mulit classification
      multi_model = KNeighborsClassifier(metric='manhattan', weights='distance')
      multi_model.fit(multi_feature_train_X, multi_feature_train_y)
      multi_predictions = multi_model.predict(multi_feature_val_X)

      # get the score
      accuracy_score(multi_predictions,multi_feature_val_y)

      0.9758994062172547
```

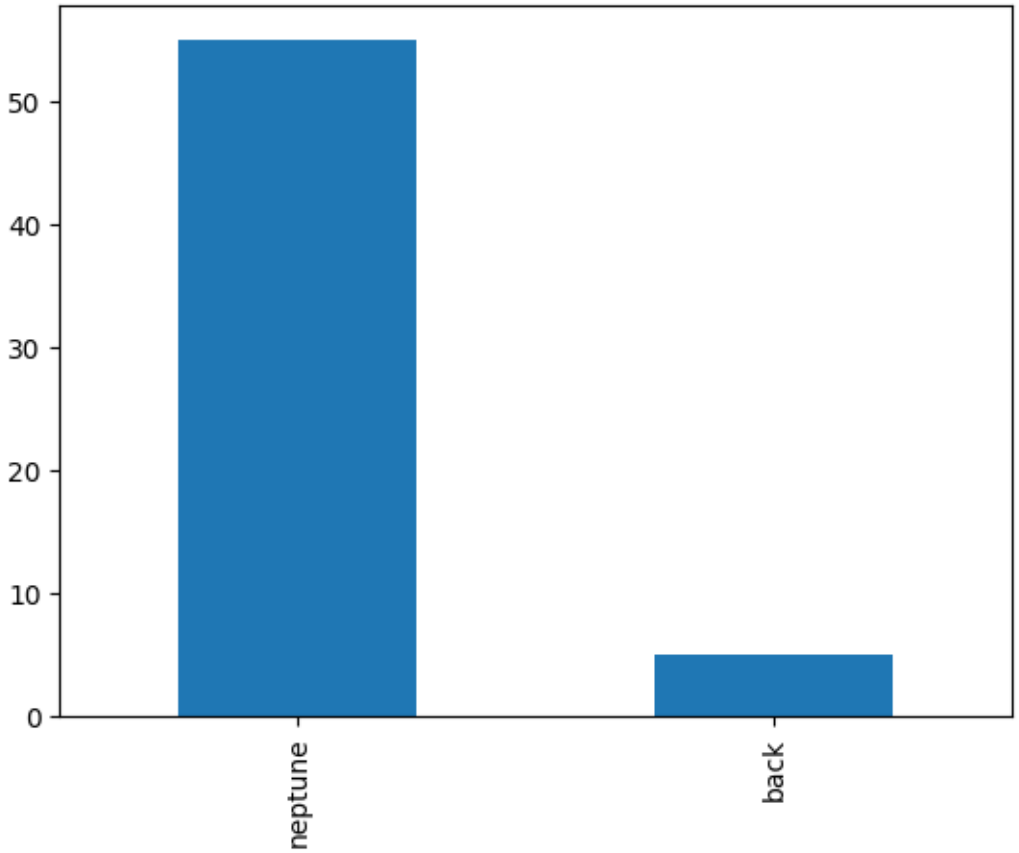
در این مرحله الگوریتم هم از تعداد پارامترهای مناسبی برخوردار است و هم پارامترهای الگوریتم به دقت تنظیم شده است.

بنابراین در این مرحله امکان بهبود جدیدی در الگوریتم وجود ندارد.

شکل زیر ماتریس سردرگمی نهایی را برای دسته بندی چند کلاسه نمایش می دهد:



شکل زیر بیشترین نوع حملاتی را که به درستی طبقه بندی نشده اند را برای دسته بندی چند کلاسه نشان می دهد.


50
40
30
20
10
0
neptune
back

۵- گام پنجم: ارزیابی یا Evaluating

حالا که ما با استفاده از دو الگوریتم طبقه بندی جنگل تصادفی (random forest) و k نزدیک ترین همسایه (KNN) مدل خود را ساختیم، نوبت به ارزیابی مدل می رسد. مدل خود را که با استفاده از داده های آموزشی آموزش داده ایم، در برابر برخی از داده های نادیده (داده های تست) ارزیابی می کنیم. ما می توانیم این را به عنوان ترافیک شبکه جدید در نظر بگیریم.

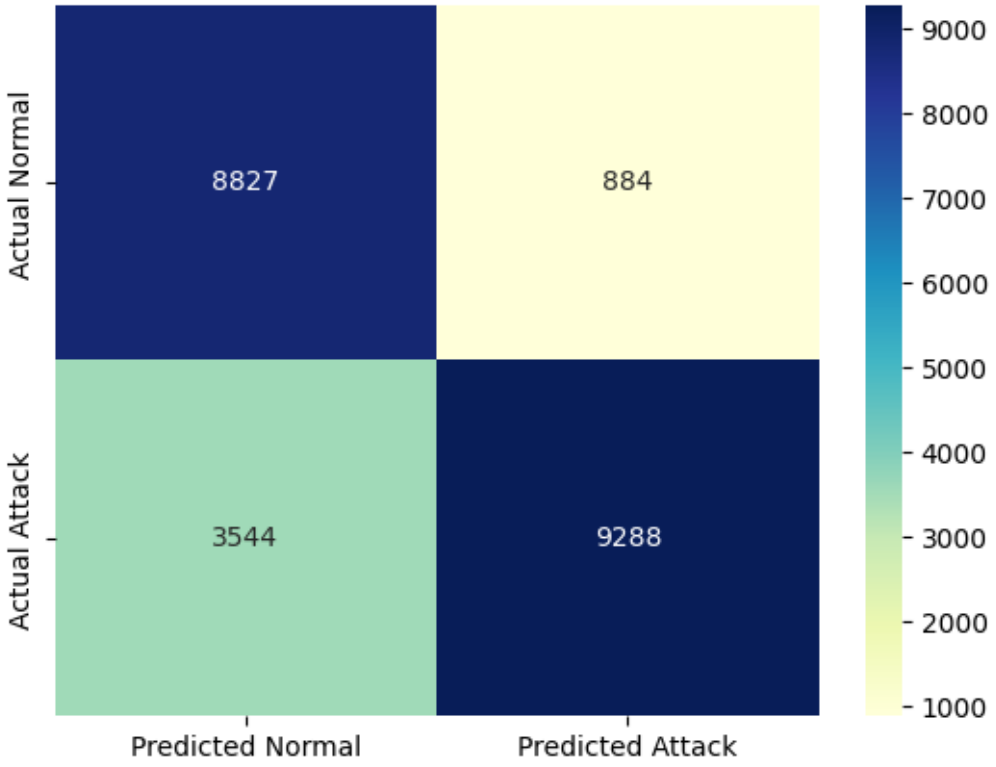
ارزیابی الگوریتم جنگل تصادفی :

ابتدا مدل طبقه بندی دودویی را تست می کنیم :

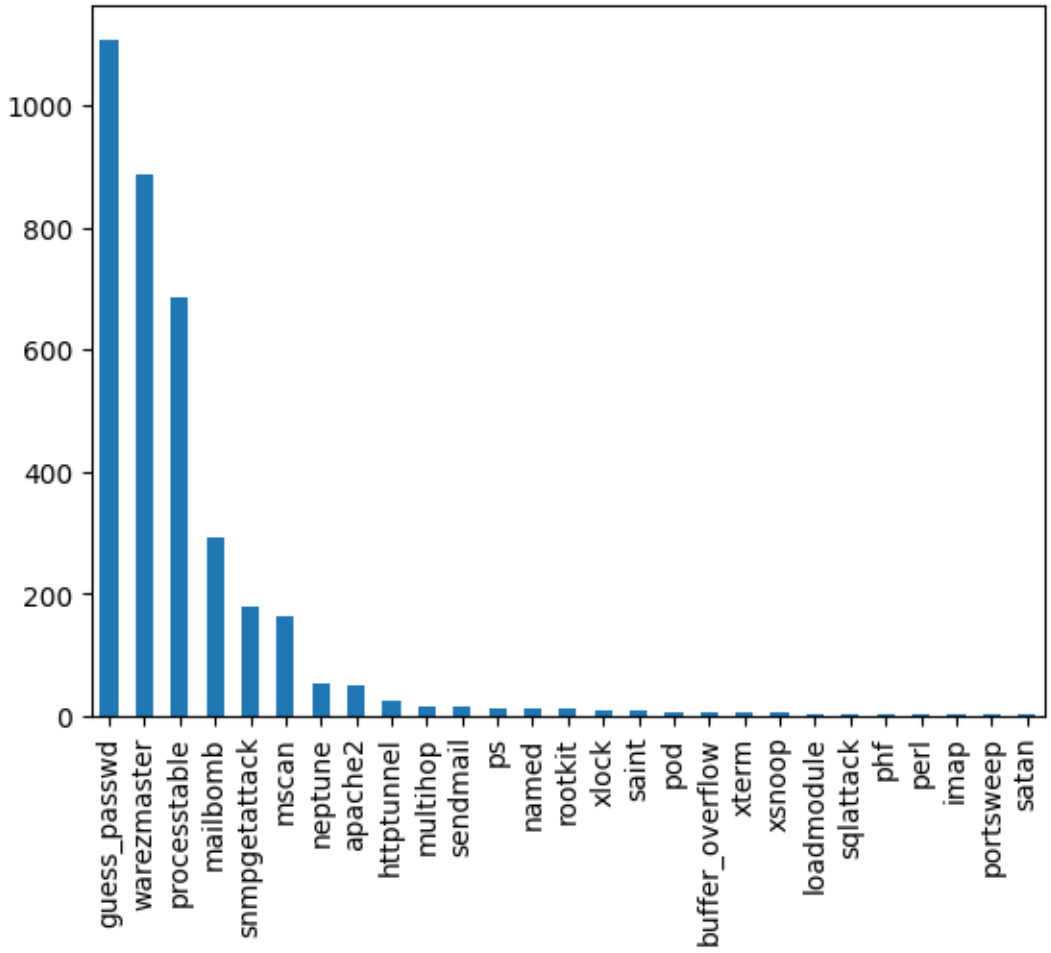
هنگام تست الگوریتم با داده های تا الان ندیده است (داده های تست) ما دقتی برابر با 0.8035753892560884 می گیریم. همانطور که ملاحظه می کنید، دقت مدل ما بر روی داده های تست افت داشته است.

```
[207] # model for the binary classification
      full_model = RandomForestClassifier(random_state=1)
      full_model.fit(to_fit, binary_y)
      full_predictions = full_model.predict(test_set)

      # get the score
      full_score = accuracy_score(full_predictions,test_binary_y)
      full_score

      0.8035753892560884
```

در شکل ماتریس سردرگمی را برای طبقه بندی دودویی می بینیم :



شکل زیر بیشترین نوع حملاتی را که به درستی طبقه بندی نشده اند را نشان می دهد.



سپس مدل طبقه بندی چند کلاسه را تست می کنیم :

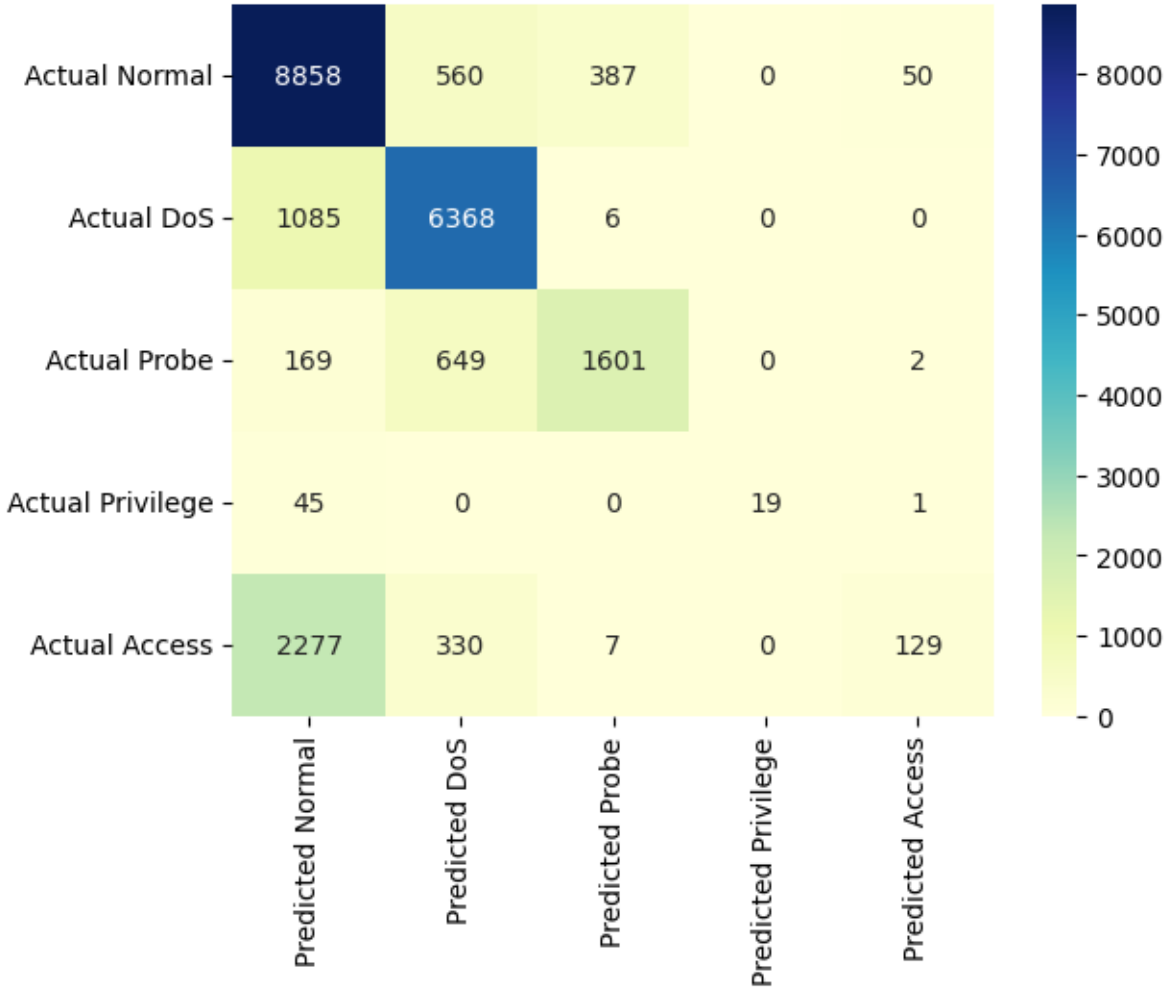
هنگام تست الگوریتم با داده های تست ما دقتی برابر با 0.7530053675198509 می گیریم.

```
# run the model on the smaller column set
multi_model.fit(to_fit, multi_y)
full_multi_predictions = multi_model.predict(test_set)

# get the score
accuracy_score(full_multi_predictions,test_multi_y)
```

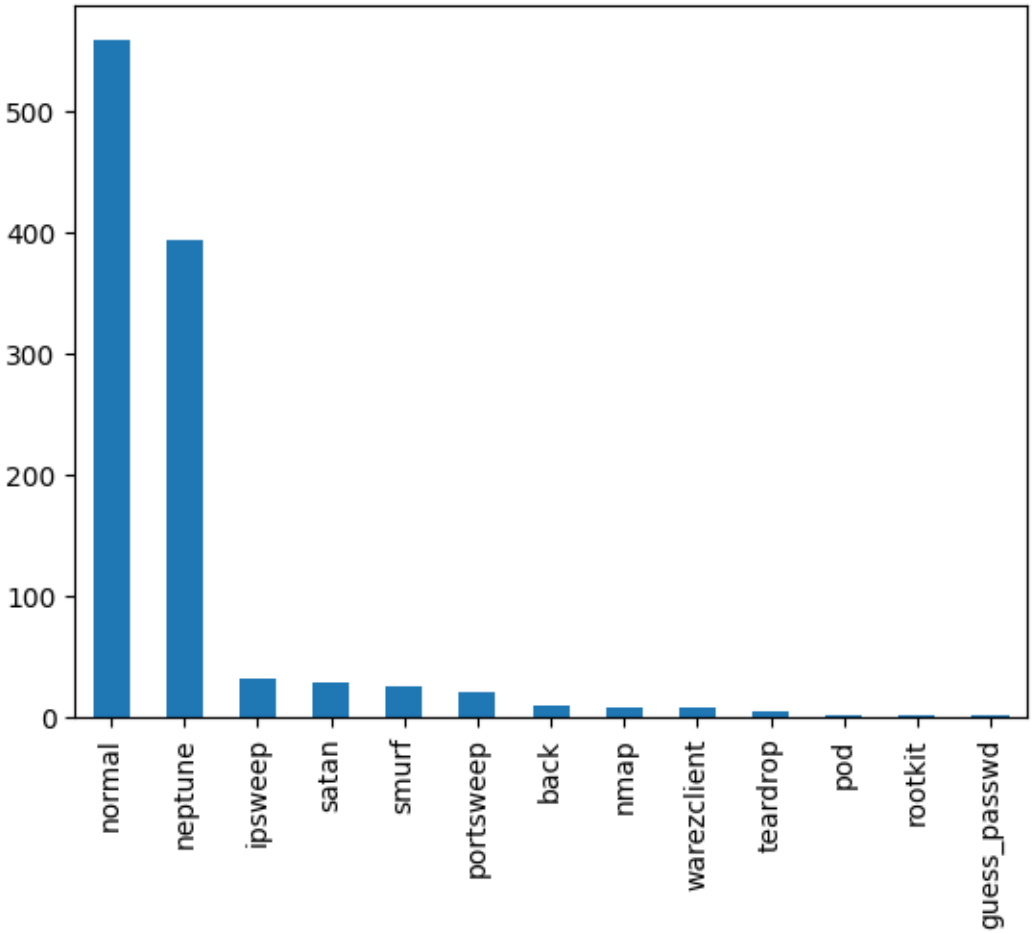
```
0.7530053675198509
```

همانطور که می بینیم دقت طبقه بندی چند کلاسه به دلیل اینکه در طبقه بندی چند کلاسه احتمال خطا بیشتر وجود نسبت به دسته بندی دودویی کمتر می باشد.

در شکل ماتریس سردرگمی را برای طبقه بندی چند کلاسه می بینیم :



شکل زیر بیشترین نوع حملاتی را که به درستی طبقه بندی نشده اند را برای دسته بندی چند کلاسه نشان می دهد.



ارزیابی الگوریتم KNN :

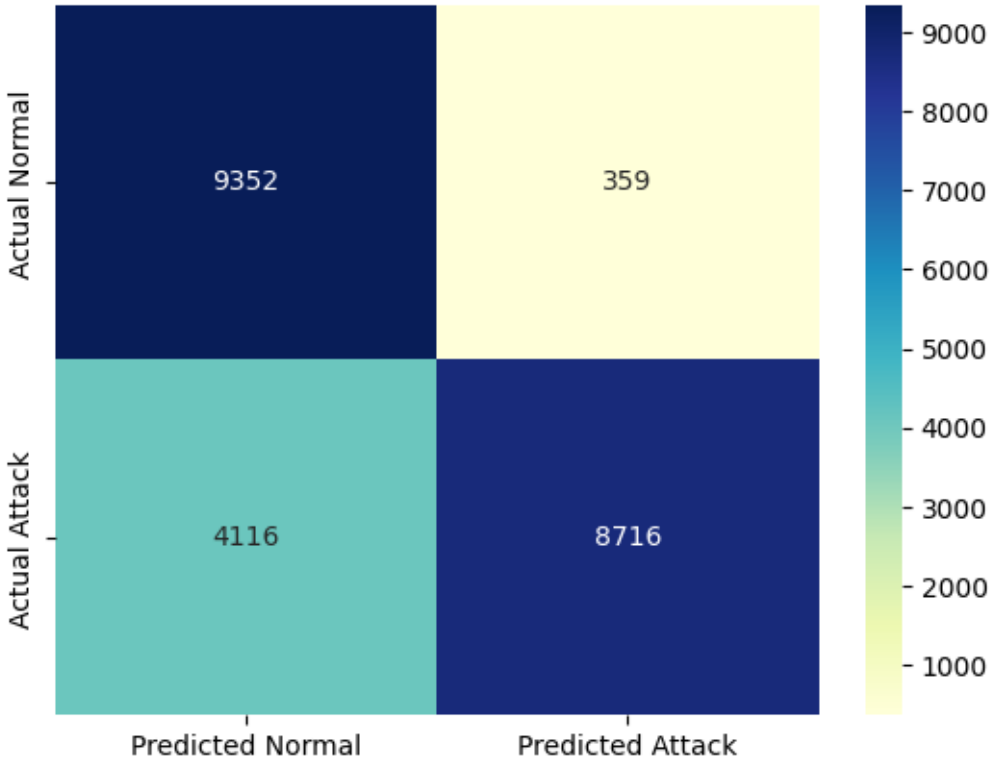
ابتدا مدل طبقه بندی دودویی را تست می کنیم :

هنگام تست الگوریتم با داده های تا الان ندیده است (داده های تست) ما دقتی برابر با 0.8014904848511734 می گیریم. همانطور که ملاحظه می کنید، دقت مدل ما بر روی داده های تست افت قابل ملاحظه ای داشته است.

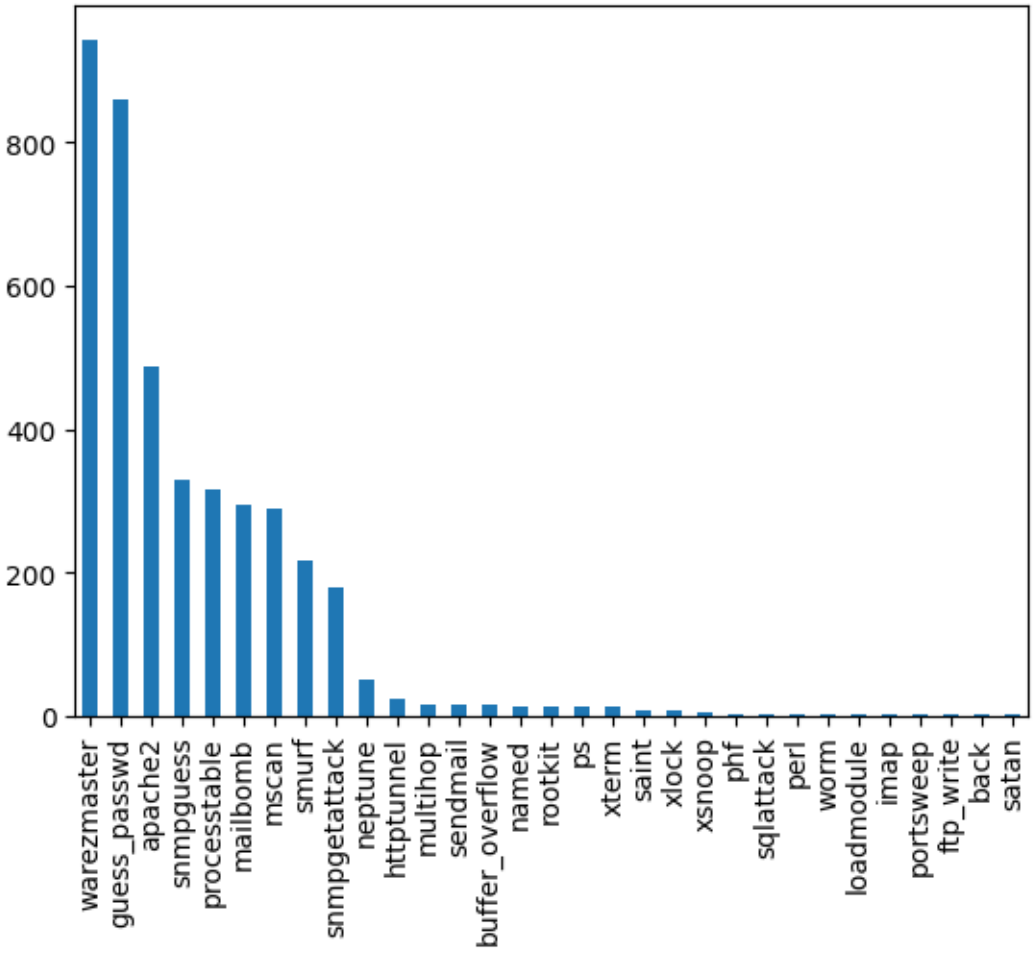
```
# model for the binary classification
full_model = KNeighborsClassifier(metric='manhattan', weights='distance')
full_model.fit(to_fit, binary_y)
full_predictions = full_model.predict(test_set)

# get the score
full_score = accuracy_score(full_predictions,test_binary_y)
full_score
```

```
0.8014904848511734
```

ماتریس سردرگمی را برای الگوریتم KNN مشاهده می کنیم.



شکل زیر بیشترین نوع حملاتی را که به درستی طبقه بندی نشده اند را نشان می دهد.



سپس مدل طبقه بندی چند کلاسه را تست می کنیم :

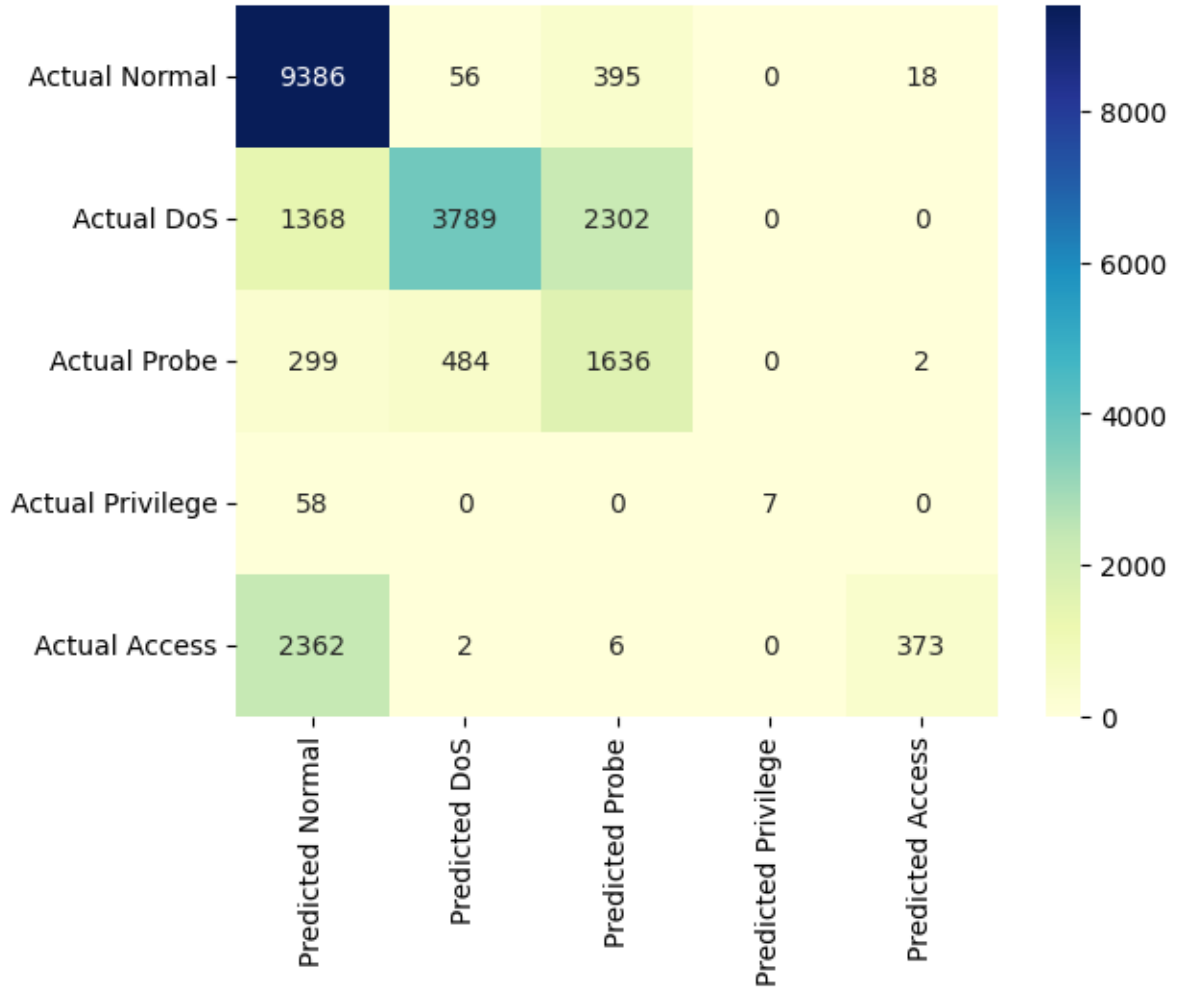
هنگام تست الگوریتم با داده های تست ما دقتی برابر با 0.6738677194694583 می گیریم.

```
[52] # run the model on the smaller column set
     multi_model.fit(to_fit, multi_y)
     full_multi_predictions = multi_model.predict(test_set)

     # get the score
     accuracy_score(full_multi_predictions,test_multi_y)

     0.6738677194694583
```

همانطور که می بینیم دقت طبقه بندی چند کلاسه به دلیل اینکه در طبقه بندی چند کلاسه احتمال خطا بیشتر وجود نسبت به دسته بندی دودویی کمتر می باشد. این احتمال وجود دارد که مدل ما دچار بیش

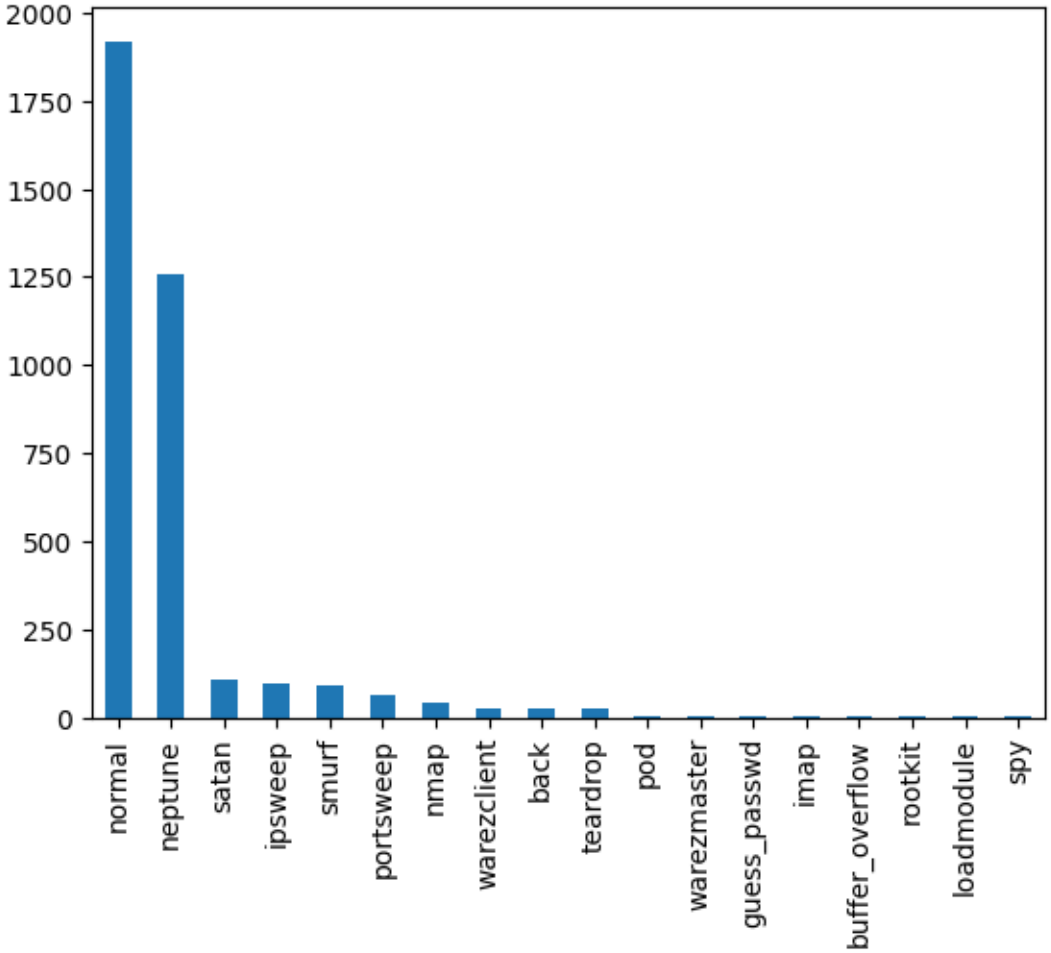
برازش(over fitting) شده باشد.یعنی اینکه ما ویژگی های زیادی به مدل داده ایم و مدل آن ها حفظ کرده است.

ماتریس سردرگمی را برای الگوریتم K-nearest neighbors چند کلاسه مشاهده می کنیم.



ما شاهد بسیاری از موارد منفی کاذب (حملات از دست رفته) هستیم.

شکل زیر بیشترین نوع حملاتی را که به درستی طبقه بندی نشده اند را برای دسته بندی چند کلاسه نشان می دهد.


2000
1750
1500
1250
1000
750
500
250
0
normal
neptune
satan
ipsweep
smurf
portsweep
nmap
warezclient
back
teardrop
pod
warezmaster
guess_passwd
imap
buffer_overflow
rootkit
loadmodule
spy

۶- گام ششم: بکارگیری یا Deployment

پروتکل icmp به طور قابل ملاحظه ای جهت نفوذ در شبکه استفاده شده است، بنابراین سیستم IDS جهت برقراری امنیت باید توجه قابل قبولی به ترافیک هایی که از این پروتکل استفاده می کنند نمایید.

حملات satan, nmap, ipsweep از هر سه پروتکل icmp و tcp و udp جهت نفوذ استفاده می کنند. بنابراین شناسایی آن ها سخت تر می باشد و سیستم IDS باید توجه بیشتری به این نوع حملات کند.

بیشترین نوع حملاتی که شناسایی نشده اند Neptune و Satan هستند که این یکی از نقاط ضعف سیستم امنیتی سازمان می باشد که باید در آینده تمهیداتی برای آن اندیشیده شود. حمله Neptune حتی بعد از بهبود هر دو الگوریتم ما همچنان نرخ منفی کاذب زیادی دارد، که به نظر می رسد شناسایی این نوع حمله برای هر دو الگوریتم ما نیز سخت می باشد.

یکی دیگر از مشکلاتی که در دسته بندی چند کلاسه مشاهده کردیم نرخ بالای ترافیک عادی بود که اشتباه به عنوان حمله شناسایی شده بود. که این نیز می تواند این نتیجه را ترافیک های حمله به ترافیک عادی شبکه شبیه هستند که این نیز یکی از مشکلاتی است که اغلب سیستم های تشخیص نفوذ از آن رنج می برند، که باعث می شود اعتماد به سیستم از بین برود. در نتیجه سیستم تشخیص نفوذ سازمان باید در جهت برطرف شدن این مشکل تلاش نماید.

اما یکی از مهم ترین نتایج مدل سازی هنگام تست مدل های طبقه بندی دودویی مشاهده شد. برخی از حملات از جمله guess_passwd و warezmaster و proccesstable و snmpguess و apache2 که در هنگام ارزیابی با داده های validation، مدل برای شناشایی آن ها مشکلی نداشت و به راحتی شناسایی می شدند، اما هنگام تست با داده های آزمون بیشترین تعداد خطا را در هر دو الگوریتم به خود اختصاص دادند. که این امر نشان می دهد از جمله حملات ناشناخته هستند و با دیگر انواع حملات تفاوت های چشم گیری دارند و یا به ترافیک عادی شبکه شباهت بسیاری دارند. بنابراین جزو حملاتی هستند که می توانند برای امینت سازمان خطر آفرین باشند، در نتیجه سیستم IDS احتمالا در شناسایی آن ها با مشکل موجه خواهد شد و سیستم IDS جهت مقابله با آن ها باید تقویت شود.