به نام خدا

محمد زیاری - (97222047)

تاریخ: 1400 /10/15

گزارش تمرین سری 4

درس: مبانی علوم داده

استاد درس: دکتر خردپیشه

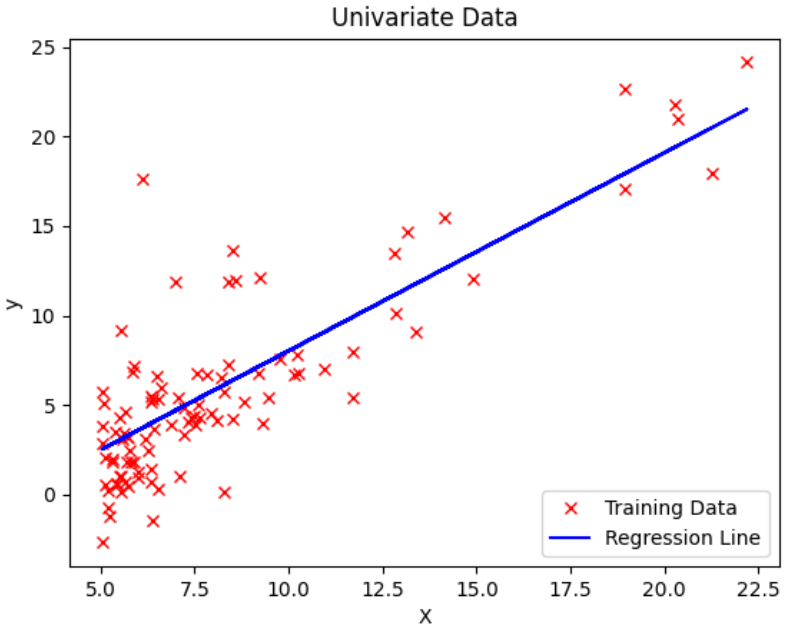
تمرین 1)

در تمرین اول با یک کد از رگرسیون خطی مواجه هستیم که باید طبق فرمول هایی که وجود دارد ، قسمت های مختلف کد را که انجام نشده است تکمیل کنیم.

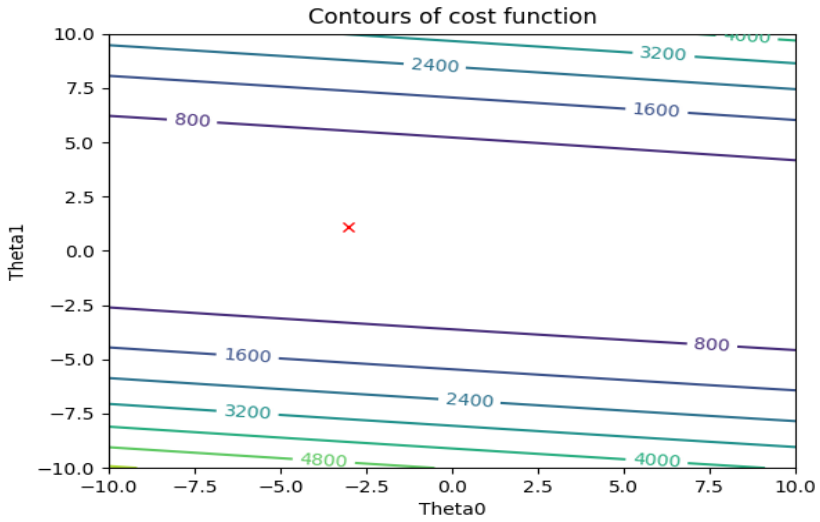
به نظر نیاز به توضیح کد نیست اما در بخش gradientDescent در هر مرحله طبق فرمول theta اپدیت می شود .

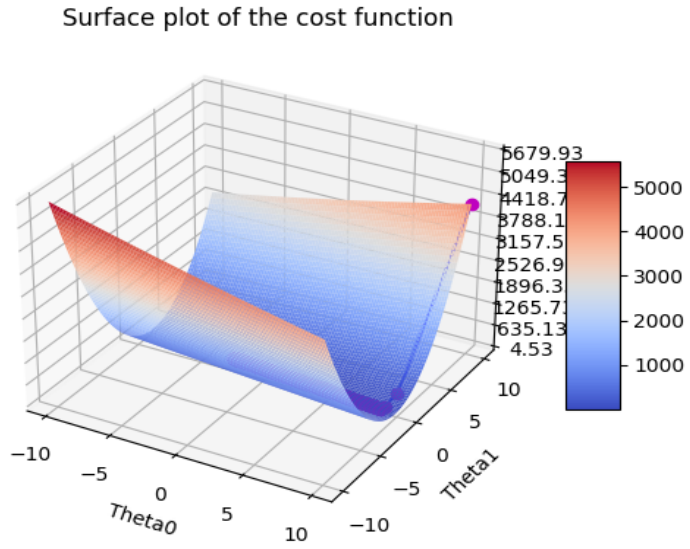
در تابع computeCost باید میزان خطا برحسب MSE ، طبق فرمولی که داریم محاسبه شود که این میزان معمولا در هر مرحله از یادگیری روند کاهشی را باید داشته باشد تا در واقع خطایمان کاهش پیدا کند. در مرحله predict نیز تنها لازم است تابع رگرسیون خطی مان که در پیشبینی به ما کمک میکند را برگردانیم.

پس از انجام کد رگرسیون خطی باید آن را روی data 3 ی داده شده تست کنیم و نتایج را گزارش کنیم. در ابتدا روی Univariate Data ان را تست می کنیم. نمودار scatter داده ها را در زیر با هم مشاهده می کنیم که با استفاده از مدل رگرسیون خطی پیشبینی شده اند.



نمودار های زیر نیز نمایانگر تابع هزینه می باشد.





اما میزان تابع cost در قدم 1500ام 4.544580590822311 بود که به نسبت پکیج sklearn مقدار مناسبی به نظر می آید.

بعد از آن نوبت به تست روی داده های multivariate می رسد که در این تابع میزان هزینه یا خطا پس از 2000 مرتبه تکرار 2043280083.2124722 بود که این مورد هم با توجه به اینکه داده های چند متغیره داریم به نسبت پکیج sklearn عملکرد بهتری داشت.

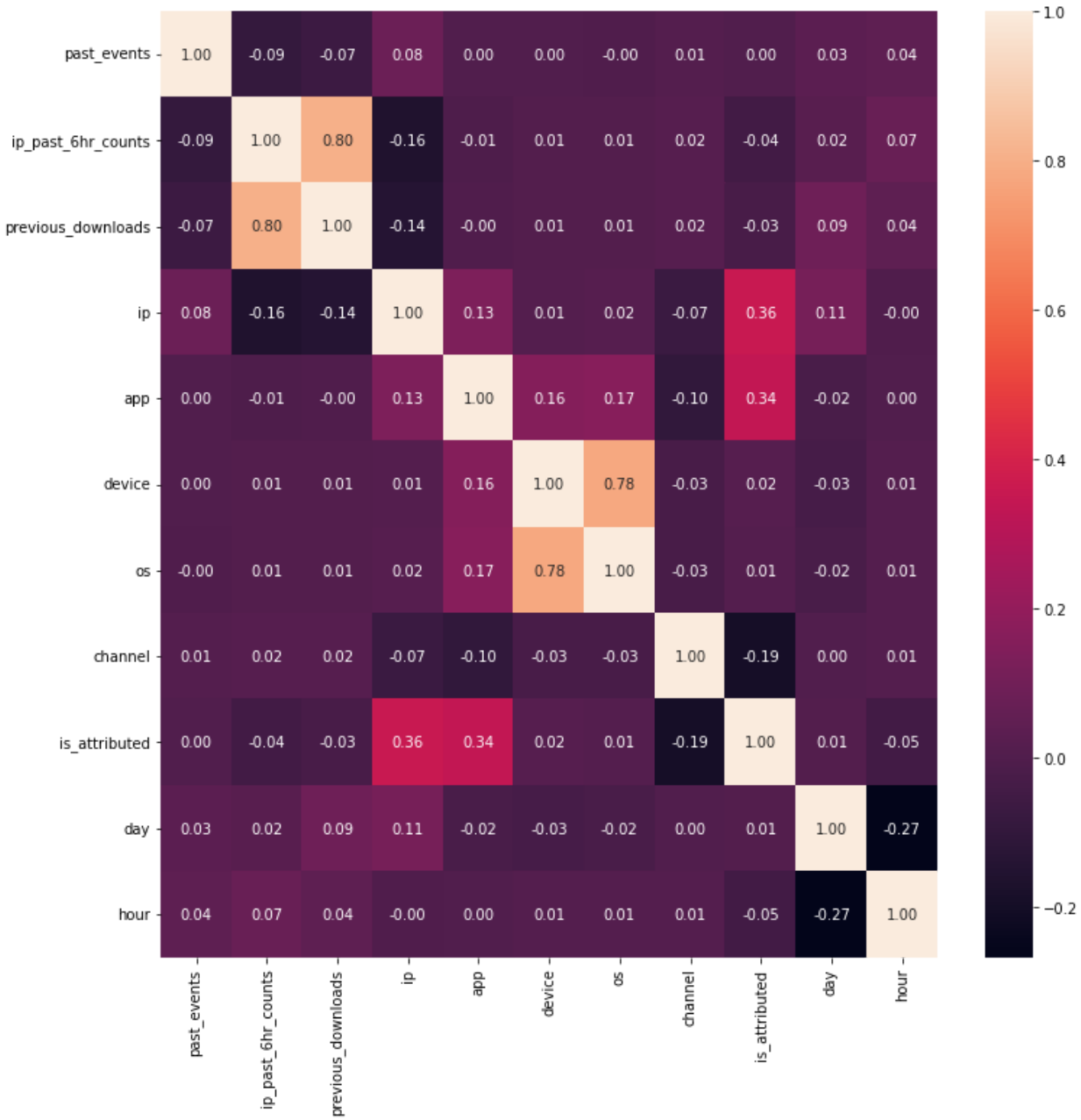
در آخرین بخش نیز روی holdout رگرسیون خطی مان را اعمال کردیم. برای این کار مانند داده ی قبلی که چند متغیره بود باید با آن رفتار شود که میزان MSE در آن پس از 2000 تکرار 1064493913.6931804 بود و RMSE نیز 32626.58293013089 بود.
لازم به ذکر است برای محاسبه RMSE طبق فرمول زیر تنها کافی است میزان جذر cost قبلی که مربوط به mse بود را در هر مرحله خروجی دهیم.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

کد مربوط به rmse را دیگر ارسال نکردم چون با تنها یک جذر گرفتن ساده بدست می امید و تنها فایل مربوط به mse ارسال شده است.

تمرین 2)

در این تمرین با داده هایی روبرو هستیم و از ما خواسته شده است تا با استفاده از feature های مناسب بررسی کنیم که آیا تبلیغی که برای شخصی آمده ، او را متقاعد میکند آن را دانلود کند یا خیر.
برای این کار از مدل های classifier مختلف بهره میگیریم که به نوبت در ارتباط با آنها صحبت خواهیم کرد.
پس از خواندن داده ها از ککل ، میخواهیم پیش پردازش داده ها را انجام دهیم. برای این کار از داده train_sample فیچرهای attributed time و click_time را حذف میکنیم.
به جای اولی که لازم نیست مقداری قرار بگیرد زیرا اگر داده ای attributed time داشته باشد is_attributed آن 1 است و در غیر اینصورت مقدار null دارد . به همین منظور نیازی به این فیچر نیست. اما اگر بخواهیم از click_time استفاده کنیم ، اگر منطقی نگاه کنیم این دیتاست طی 4 روز تهیه شده است. نکته ای که میتوان در نظر گرفت آن است که در طی ساعات و روزهای مختلف آیا میزان دانلودها تفاوتی داشته است یا خیر. از این جهت بهترین کار آن است که ستونهای day و hour را از baseline_data دریافت کنیم ولی منطقی به نظر نمیاد از دقیقه و ثانیه استفاده شود.
در آخر هم دیتاست های [time_deltas , past_6hr_events , downloads] که به ترتیب نمایانگر تعداد دانلود های قبلی ، تعداد بازدید از ip در 6 ساعت اخیر و مدت زمانی که از آخرین رخداد از همان ip مورد نظر ، میگذرد ، است . قابل ذکر است این اولین تست ما روی داده ها خواهد بود و در مرحله های بعدی از encode و interaction های مختلف هم بهره میگیریم.
قبل از آنکه به سراغ بررسی کلاس بندی و مدلها برویم ، چند نمودار از ارتباط فیچرهای مختلف را باهم بررسی کنیم.
اول از همه heatmap کورولیشن فیچرهای مختلف جدول مان را در شکل زیر مشاهده می کنیم و پس از آن ، بررسی خواهیم کرد.

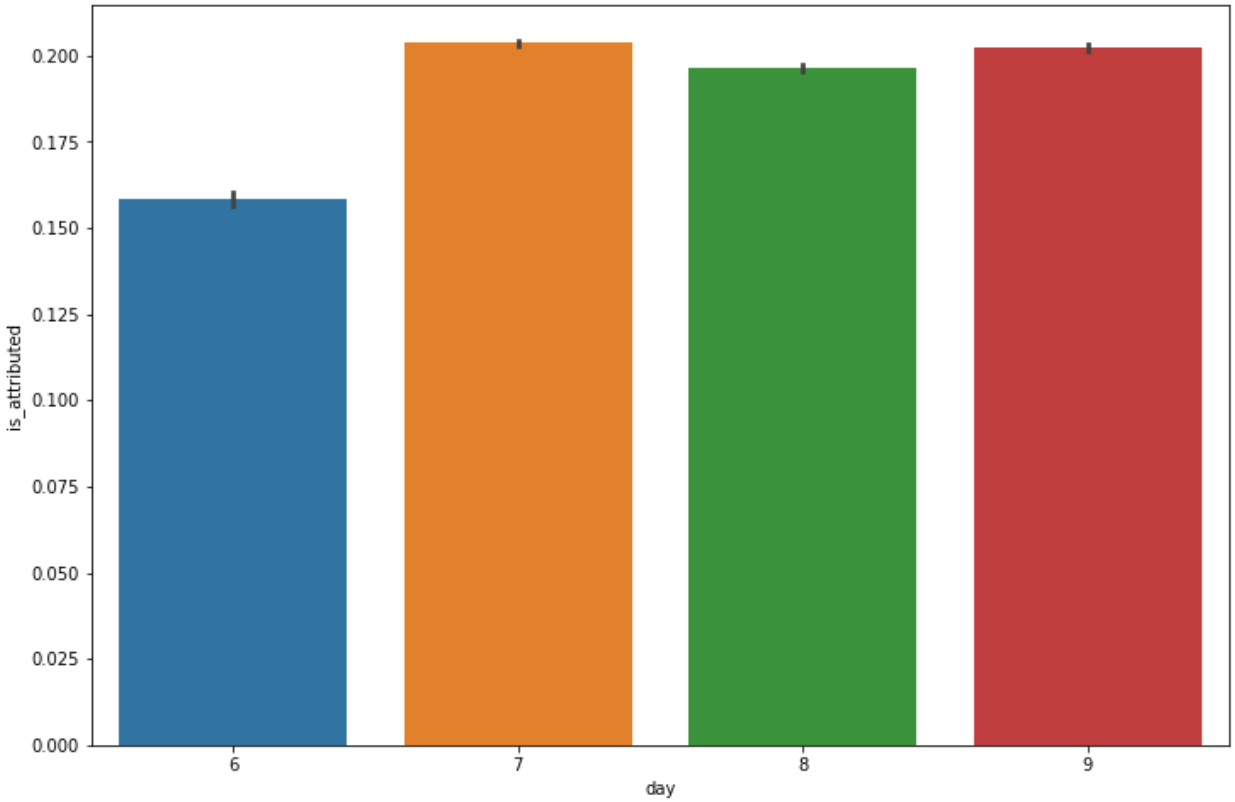


از بین روابط بالا ، رابطه ی ستون is_attributed با ستون های ip و app قابل توجه است و میزان بالایی دارد.

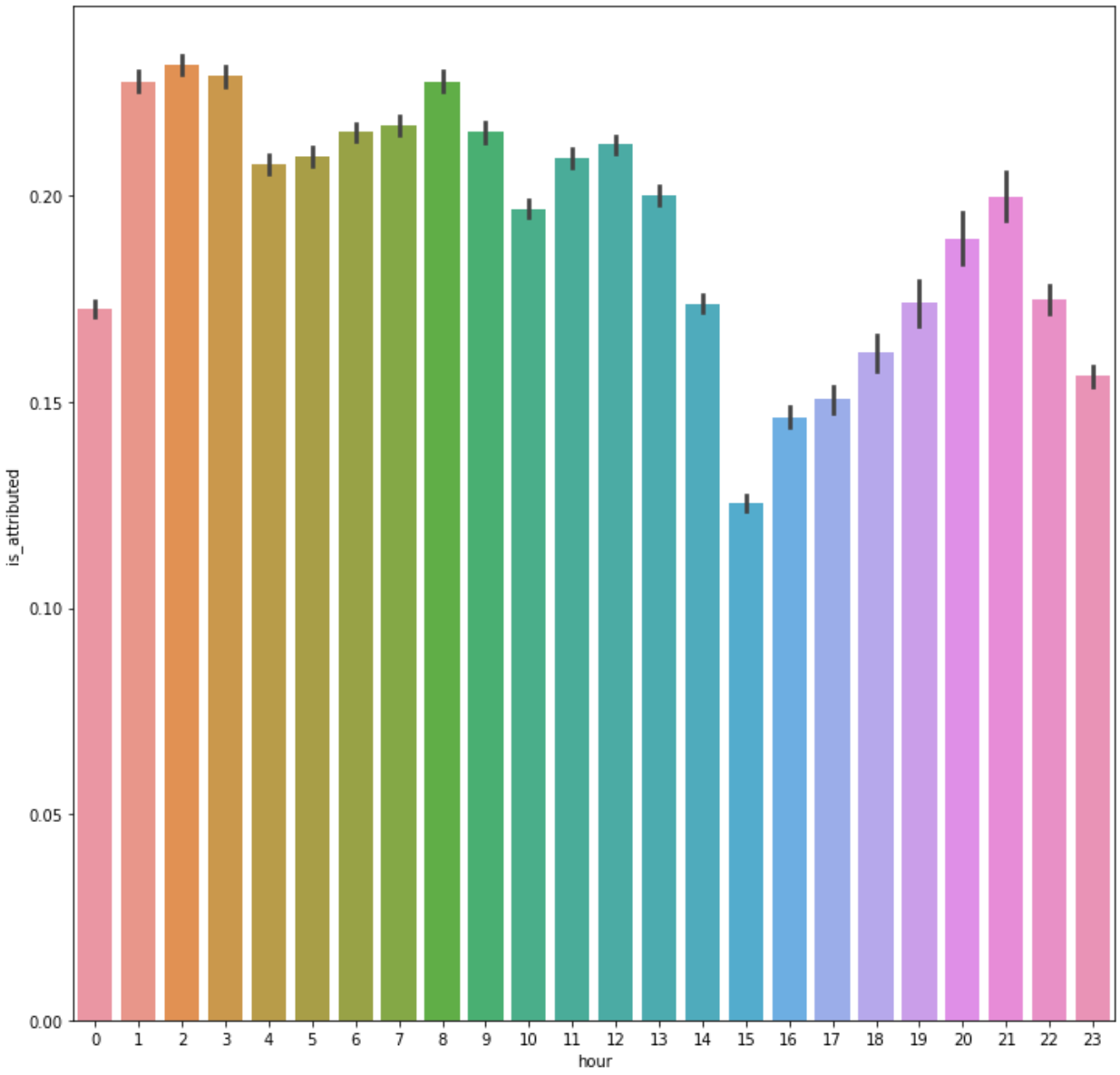
همچنین این میزان با فیچر channel رابطه عکس دارد و میزان آن نزدیک به 0.2- میباشد .

اما لازم است این مورد ذکر شود که کم بودن یا در واقع نزدیک به صفر بودن این مقدار دلیلی بر تاثیر نداشتن آن ستون ها در مدل و یادگیری ما نخواهد شد.برای مثال داده های hour و day که عدد های پایینی به آنها نسبت داده شده است را در نمودارهایی جداگانه با ستون is_attributed بررسی میکنیم.

نمودار زیر که نشان دهنده رابطه روزها و دانلود شدن app می باشد. در روز اول میزان دانلود کمتر است اما در روزهای بعدی میزان دانلود شدن به نسبت هر ورود، به نسبت افزایش داشته است.



نمودار زیر که نمایانگر همان ستون is_attributed این بار با hour است که بررسی آن نشان می دهد که در ساعات یک یا دو بامداد به نسبت افراد بیشتری که وارد سایت تبلیغ می شوند ، app را دانلود میکنند. این میزان در ساعات عصر ، مثلا در ساعت 15 در کمترین میزان خود قرار دارد.

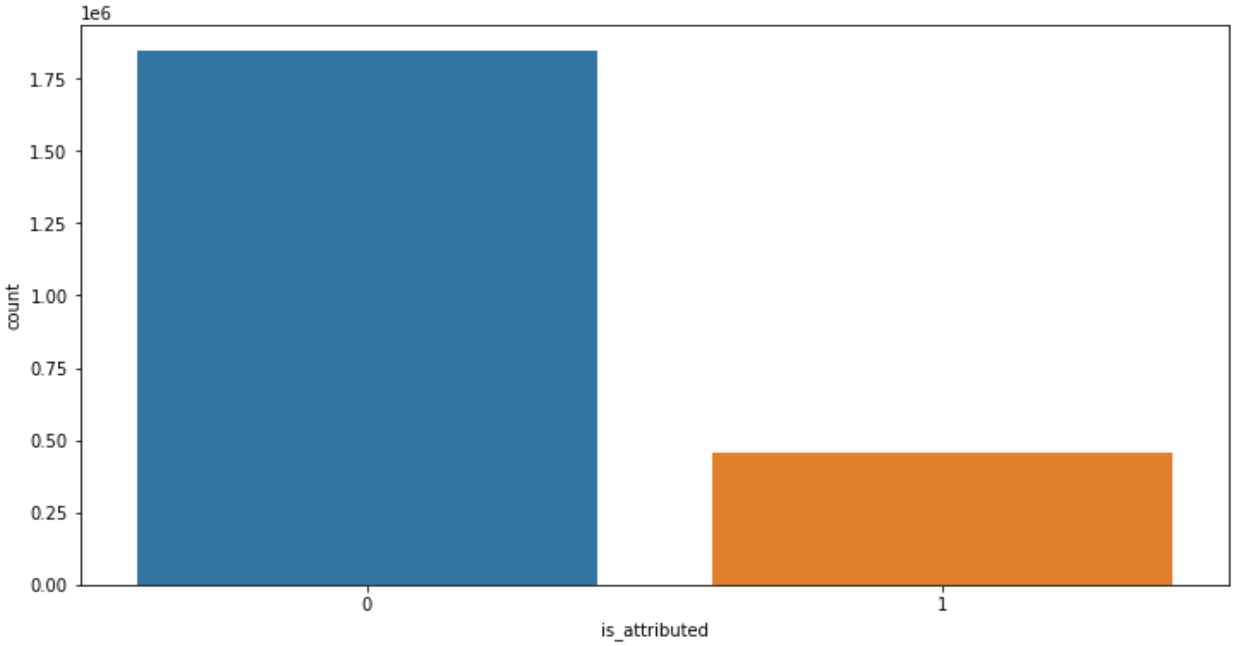


این نمودارها تا حدی نشان می دهد که برای مثال بین ساعات مختلف تفاوت وجود دارد ، اما به طوری نیست که رابطه مستقیم یا معکوس را نتیجه دهد.ولی ممکن است در مدل ما موثر باشد.

حال به سراغ آماده کردن داده ها برای دادن به مدل می رویم. قبل از انجام هر کاری موردی قابل ذکر است. برای این کار داده ها را با تقسیم 0.8 به 0.2 بین train و test تقسیم بندی کردیم و پس از آن logistic regression را روی آن پیاده سازی کردیم که نتیجه آن در جدول مقابل نشان داده است.

	precision	recall	f1-score	support
0	0.86	0.97	0.91	368531
1	0.74	0.39	0.51	91582
accuracy			0.85	460113
macro avg	0.80	0.68	0.71	460113
weighted avg	0.84	0.85	0.83	460113

قبل از آنکه جدول را تحلیل کنیم باید این نکته را ذکر کنم که همین داده ها را وقتی به svm دادم بعد از یکساعت و نیم هنوز ران نشد ، پس باید از داده ها سمپل گرفت. اما نکته آن است که سمپل باید به چه شکلی انجام بگیرد. از نتیجه ای که بالا رقم خورد و چیزی که از قبل هم می دانستیم تعداد داده هایی که is_attributed انها 1 بوده است ، یا در واقع فردی که با رفتن به آدرس مورد نظر آن را دانلود میکند بسیار کمتر از میزان 0 یا دانلود نکردن است که نمودار آن را هم را در زیر مشاهده می کنیم.



اما نکته ای که از نتایج logistic regression ، ما را به این نکته واقف کرد ، آن بود که میزان recall در حالت 1 بسیار پایین بود . با توجه فرمول recall این مورد نشان دهنده آن است که داده هایی که باید دانلود شده تلقی می شدند به اشتباه دانلود نشدنشان پیش بینی شده است. به عبارتی دیگر مدل ما دارد به سمتی پیش می رود که به جای generalized کردن ، memorize می کند و به بیشتر داده ها is attributed را 0 نسبت می دهد.

برای درست کردن این مشکل باید سمپل گیری را به نحوی انجام دهیم که داده هایمان حالت imbalanced نداشته باشند. برای این کار از آنجایی که میخواهیم تعداد داده ها را کاهش دهیم از downsampling استفاده میکنیم . روشهایی که مطرح میشود را به ترتیب بررسی میکنیم.

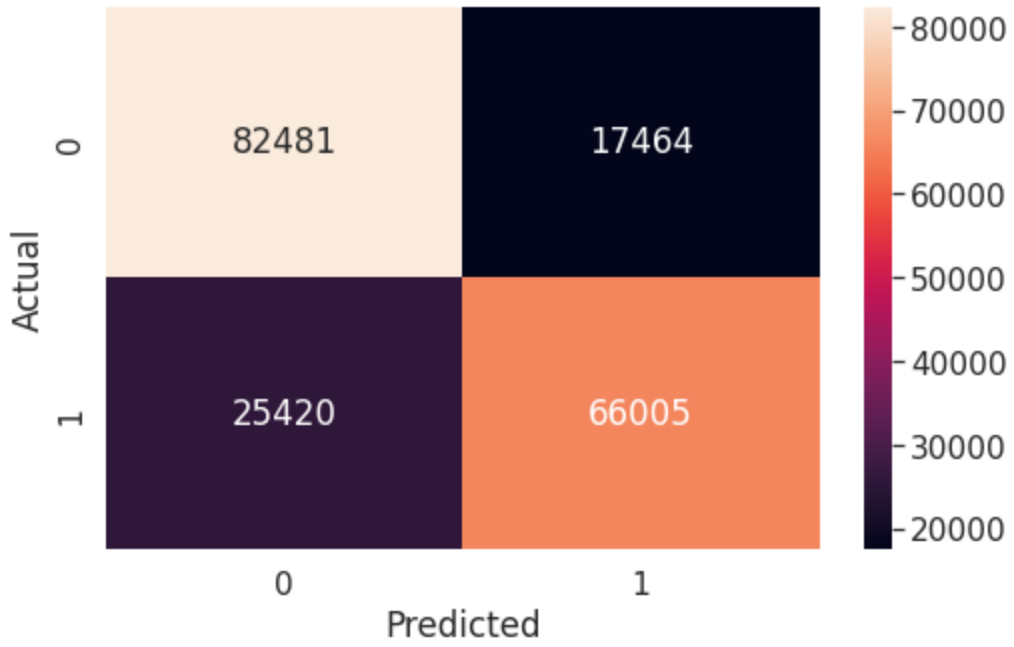
روش اول NeighbourhoodCleaningRule است که پس از فیت کردن داده ها بر روی آن مقادیر is_attributed به شکل زیر تغییر یافت :

1327843	0
456846	1

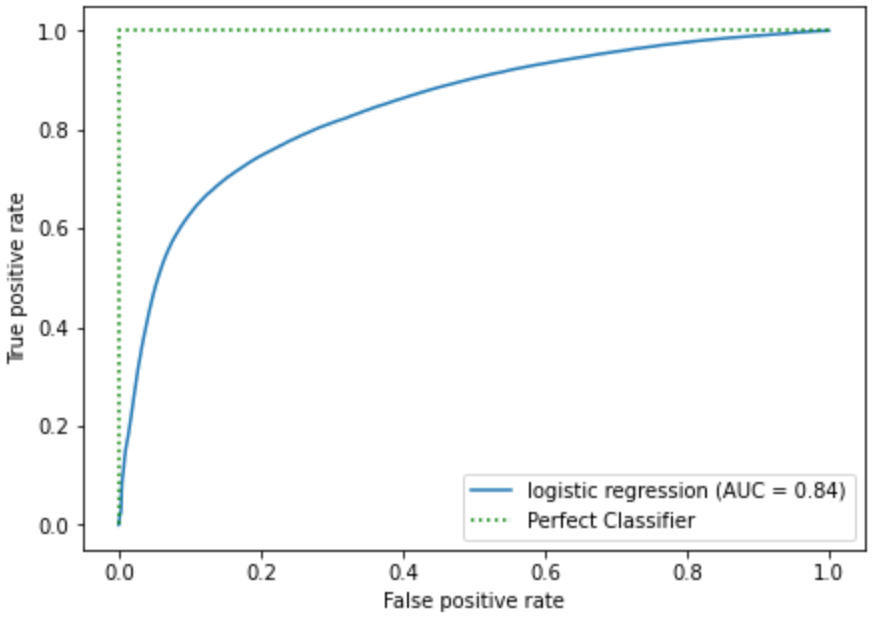
مشخصا همچنان میزان داده ها مربوط به کلاس 0 بیشتر از 1 است پس بهتر است از این روش استفاده نکنیم. برای سمپل گرفتن به صورت downsampling ما 500000 تا از داده هایی که برچسب 0 دارند را به صورت رندم انتخاب می کنیم و با داده هایی که برچسب 1 دارند در یک دیتا فریم قرار می دهیم.

قبل از این که شروع به گزارش مدلها بکنم ذکر این نکته ضروری است برای افزایش بیش از حد حجم گزارش همه ی نمودار ها برای همه مدلها در گزارش آورده نشده است. البته سعی کردم برای هر مدل نمودار مربوط و قابل بررسی و تحلیل را قرار دهم.

حال پس از انجام این کار و جدا کردن داده های اموزشی و تست و استفاده از استاندارد اسکیلر ، بار دیگر logistic regression را روی آن پیاده سازی کردیم که نتیجه آن در جدول زیر نشان داده است.



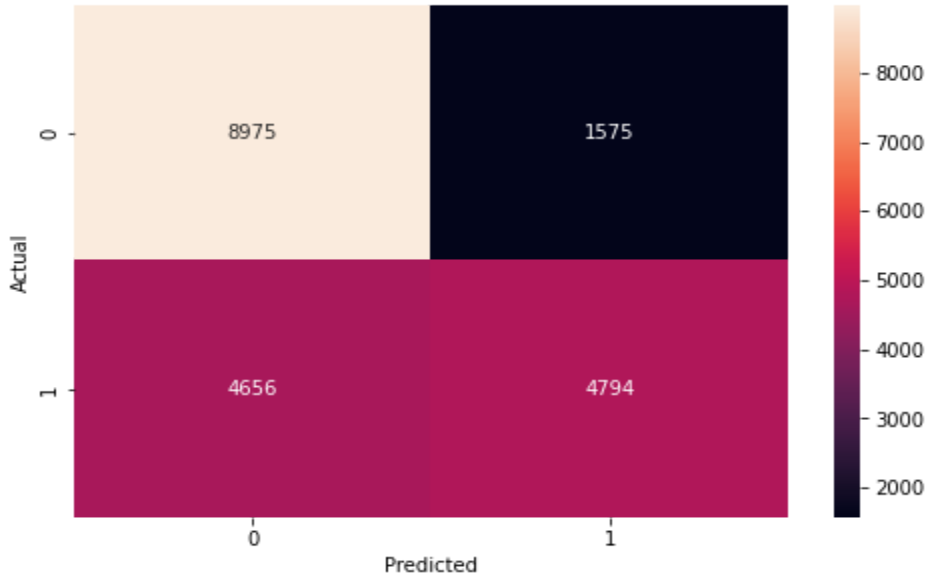
دقت ما 77 درصد است که شاید میزان بالایی نباشد اما نکته ای که قابل توجه است ، آن است که اشتباه ها در یک جا متمرکز نیستند و در واقع یادگیری در حال انجام است و این تاثیر بالانس کردن داده هاست. اما در حالتی که روی داده های بالانس نشده متد 'class_weight='balanced را پیاده سازی کردیم دقت ما بالاتر رفت و به نزدیک 80 درصد رسید. نمودار زیر هم نمایش دهنده منحنی auc برای آن است.



حالا به سراغ روش های کلاسبندی دیگر می رویم و پس از آن فیچر هایمان را عوض میکنیم تا ببینیم تاثیری در دقت دارد یا خیر.

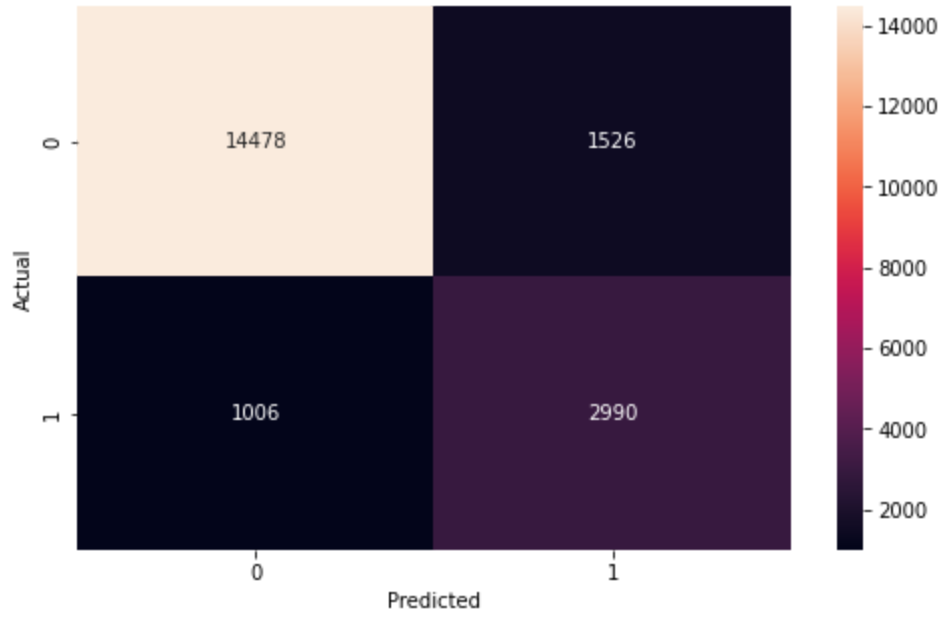
نوبت به svm می رسد اما حتی با این میزان از داده نیز این روش به سرعت ران نمی شود و روند بسیار کندی دارد. از این جهت پس از چندین بار ران کردن به این نتیجه رسیدم تا مقدار داده ای که برای آموزش و تست به آن بدهم 100000 تا باشد تا در زمان معقولی برای ما قابل انجام باشد.

همین مقدار نیز نزدیک به 10 دقیقه زمان اجرایش طول می کشد. پس از این کار درصد دقت به دست آمده 69 درصد بود که شاید اگر داده های بیشتری به آن می دادیم نتایج بهتری دریافت میکردیم. همچنین نمودار زیر نمایانگر confusion matrix این مدل می باشد.

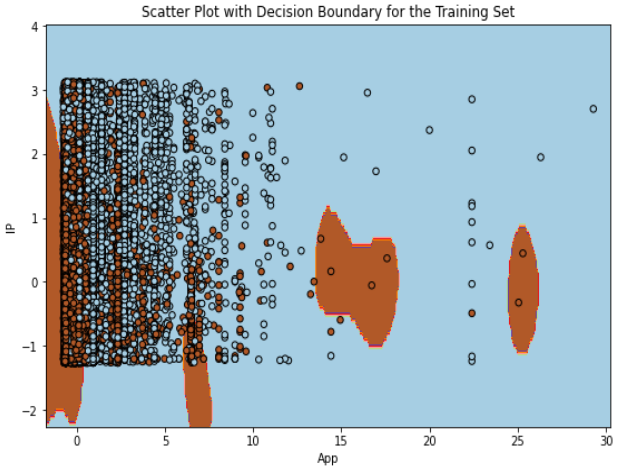


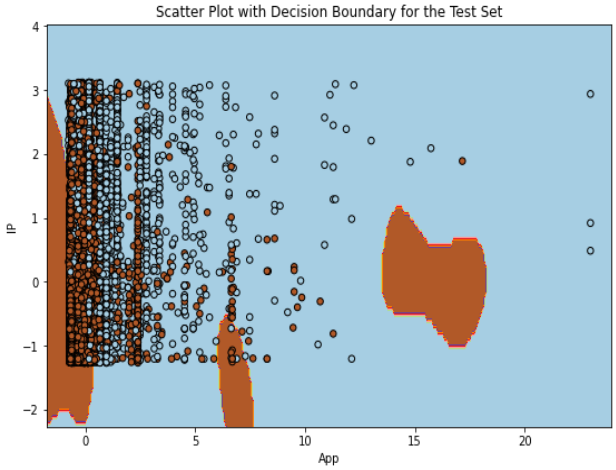
داده های مربوط به کلاس 0 تقریبا خوب پیشبینی شده اند اما داده های مربوط به کلاس 1 تقریبا نصف به نصف درست پیشبینی شده اند که این نتیجه خوبی نیست و مطمئنا باید تغییری در داده ها ایجاد کنیم تا دقت بهتری داشته باشیم. موردی که ما روی این مدل لحاظ نکردیم استفاده از اسکیلرها بود. همانطور که میدانیم استفاده از اسکیل ها تاثیر زیادی دارد و جلوی تاثیر بیش از حد برخی فیچرهایی که تفاوت زیادی در مقادیر آنها به چشم میخورد را میگیرد. برای این کار از استاندارد اسکیلر استفاده کردیم و پس از آن میزان درصد درستی ما به 83 درصد رسید که پیشرفت خوبی محسوب می شود.

زمانی که به جای سمپل کردن رندم از داده های بالانس شده از متد 'class_weight='balanced استفاده کردیم میزان دقت ما در این حالت بیشتر شد و به 87 درصد رسید که پیشرفت خوبی به نظر می آید. نمودار confusion matrix این مدل را نیز در زیر با هم بررسی می کنیم.

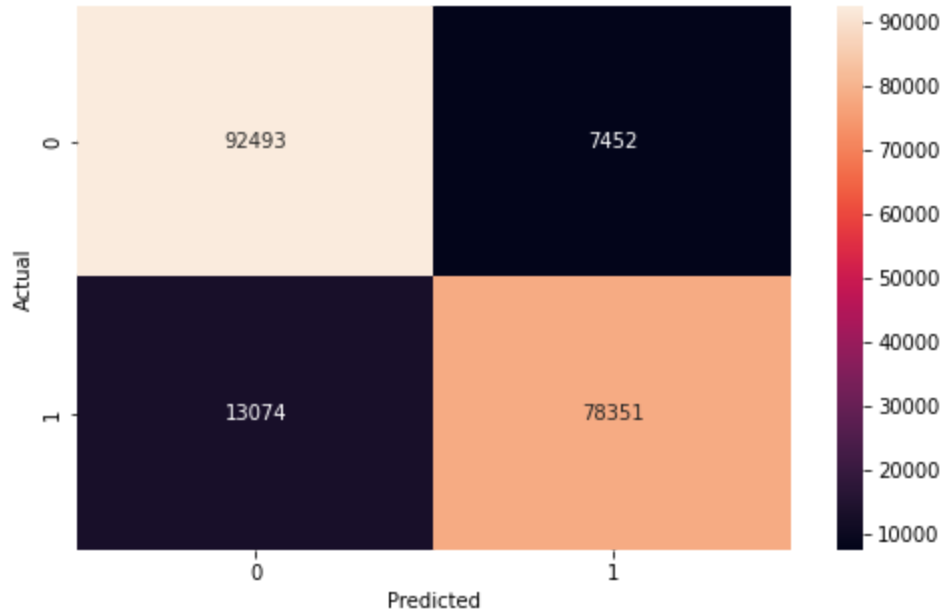


اما در ارتباط با مرز تصمیم برای این الگوریتم بیشترین تاثیر مربوط به فیچرهای app و ip بودند که پس از رسم شکل مرز تصمیم آنها به نمودار زیر رسیدیم.

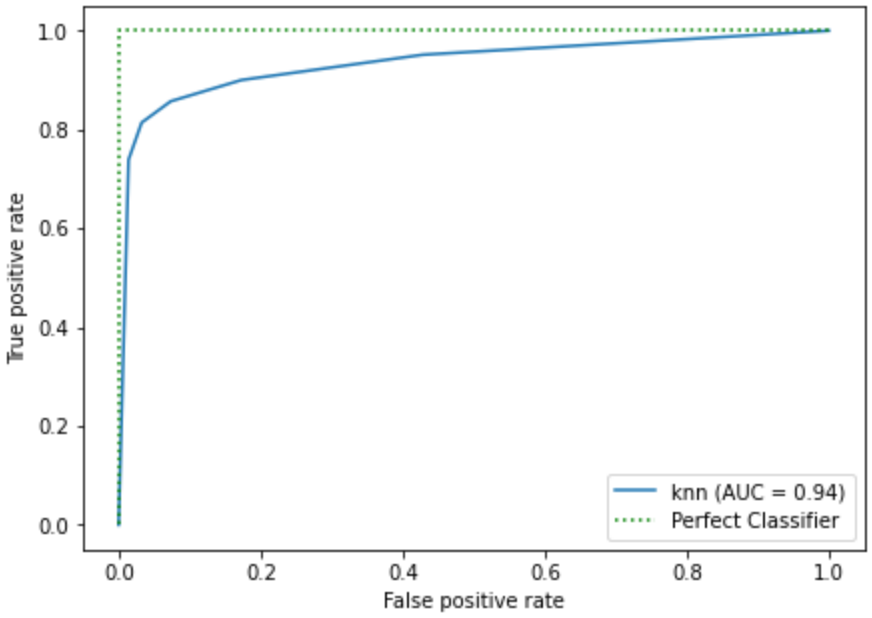




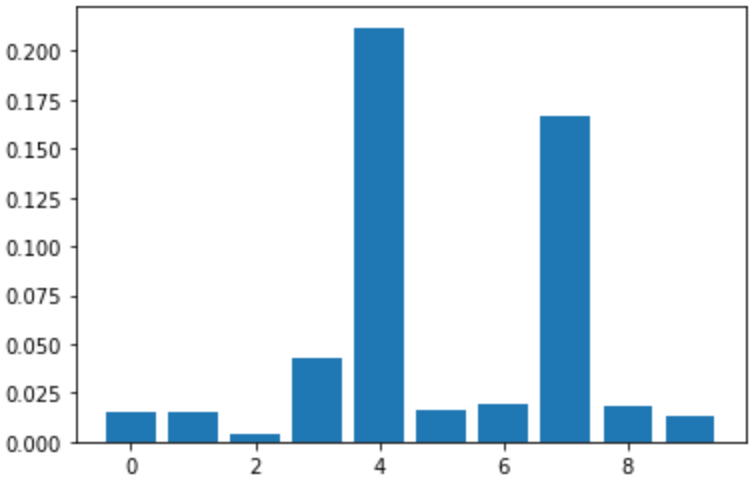
روش بعدی که از آن استفاده میکنیم knn می باشد .پس از انجام یادگیری دقت ما 89 درصد شد که به نسبت مدل های قبلی بهتر بود. نمودار confusion matrix این مدل را نیز در زیر با هم بررسی می کنیم. میتوان گفت باز هم داده های مربوط به کلاس 0 بهتر تشخیص داده شده اند اما این بار به طور چشمگیری درصد درستی تشخیص داده های مربوط به کلاس 1 هم افزایش داشته که نشان می دهد مدل ما تقریبا مدل خوبی محسوب می شود.



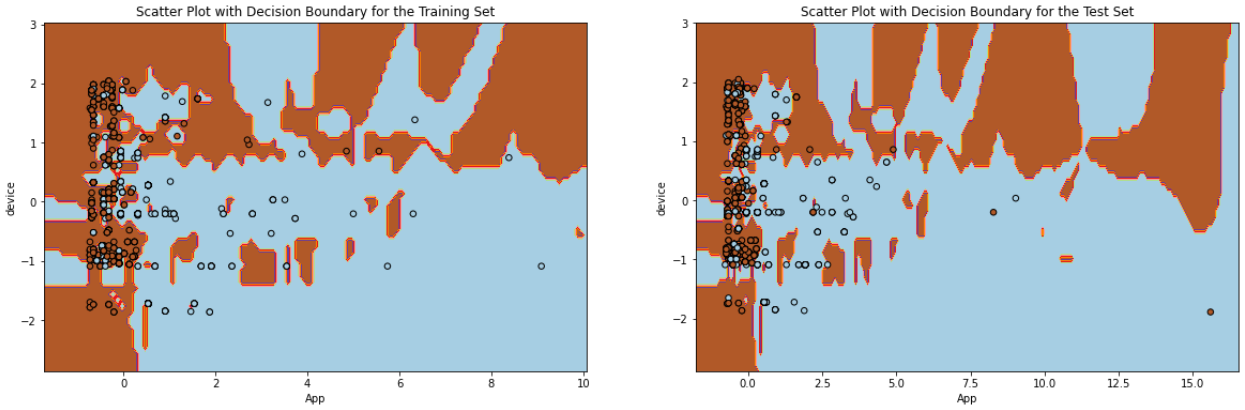
نمودار زیر نیز نمودار auc است که با توجه به میزان سطح زیر نمودار که مقدار بالایی دارد نشان دهنده آن است که مدل ما یک مدل خوب محسوب می شود.



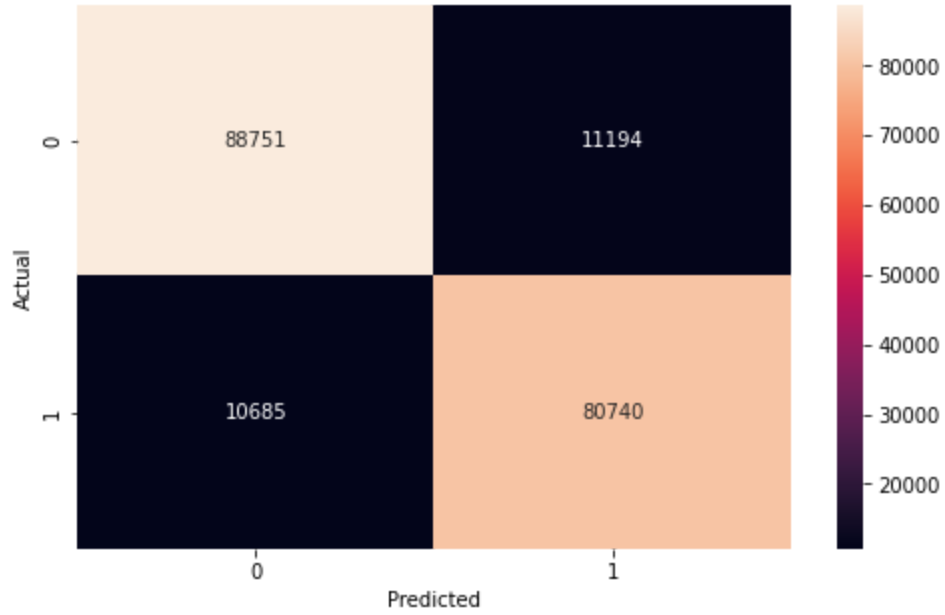
برای مرز تصمیم کدی که برای تشخیص تاثیر فیچرها بر مدل وجود داشت بسیار مدت زمان زیادی برای ران کردن داشت. در نتیجه من چندین بار داده های 1000 تایی به ان دادم و پس از چندین بار آزمون و خطا به نظر داده های ستون 4 و 7 یعنی app و channel بیشترین تاثیر را داشتند.



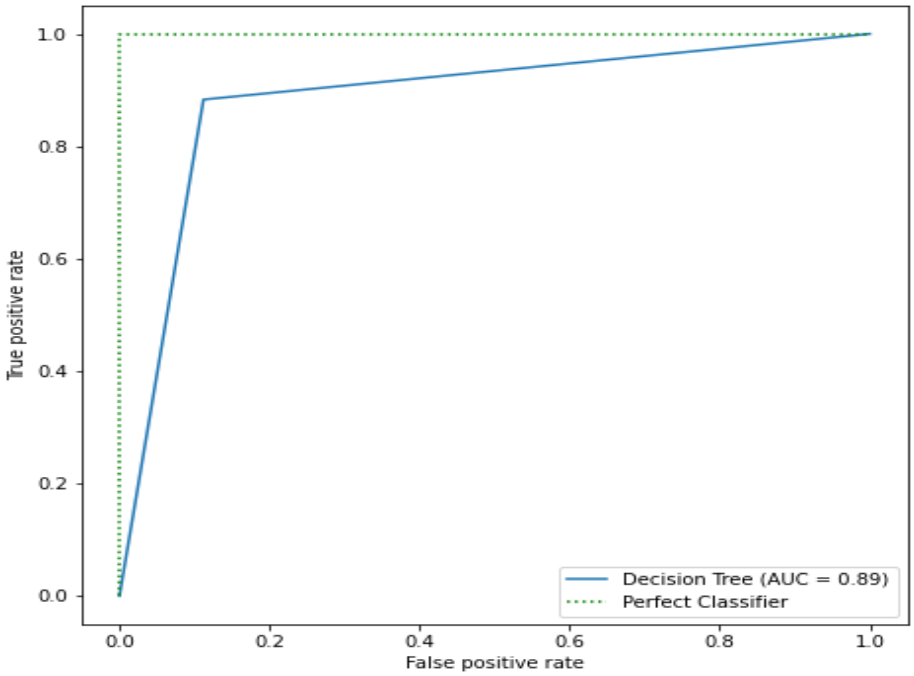
پس از آن با فیت کردن مدل روی این 2 فیچر بررسی میکنیم که در حالت 2 بعدی چطور مرز تصمیم برای آنها ترسیم می شود. نمودار زیر نشان دهنده همین مرز تصمیم است که مشخص است بعضی از داده مربوط به کلاس دیگر که نزدیک به داده های کلاس مخالف هستند به طبع آن در ان کلاس مرزبندی شده اند.



روش بعدی که بررسی شده است درخت تصمیم یا decision tree است. در این روش دقت ما بیش از 88 درصد بود. اما نکته قابل توجه آن است که با توجه به ماتریس زیر مدل ما بسیار با توازن داده ها را درست و اشتباه تشخیص داده است و به نظرم تا به حال بهترین مدل ما مربوط به همین مدل می باشد.

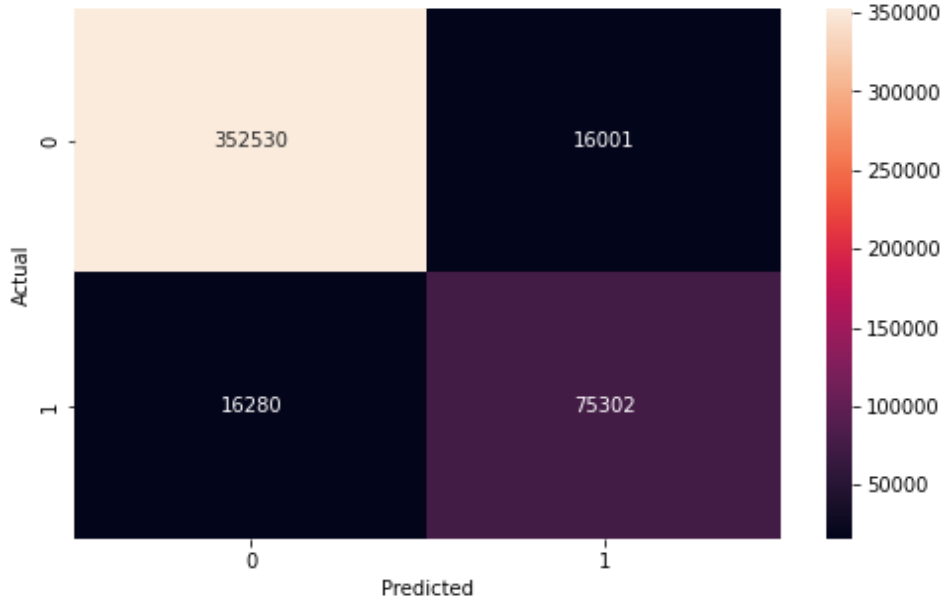


نمودار auc آن را هم در زیر مشاهده می کنیم که نزدیک به میزان تخمین درست می باشد.

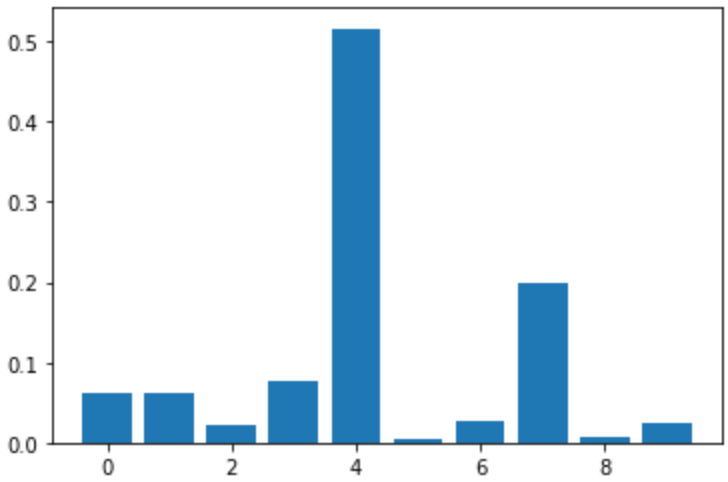


اما در حالتی که داده ها را بالانس نکرده ایم و از downsampling استفاده نکرده ایم و تنها با همان داده ی جدا شده از train و test متد 'class_weight='balanced را روی مدل درخت تصمیمان پیاده سازی کردیم دقت مدلمان تقریبا 93 درصد شد.

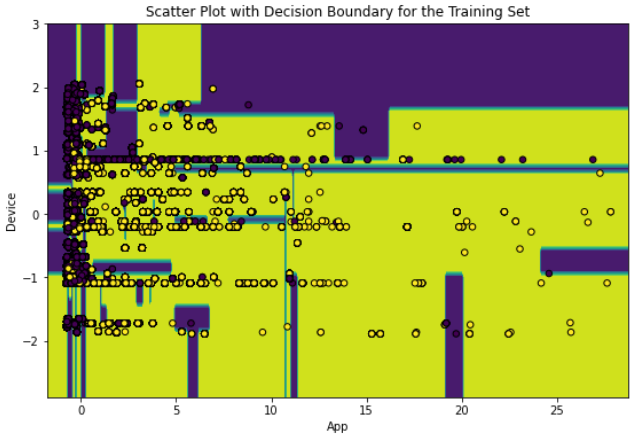
هرچند بیشتر داده هایی که مربوط به کلاس 1 بودند هم درست تشخیص داده شدند اما مشخص است طبق confusion matrix زیرکه مدلمان بیشتر داده ها را 0 تشخیص می دهد که تا البته تا حدی درست است. f1-score برای کلاس 0 حدود 96 درصد و برای کلاس 1 تقریبا 83 درصد بود.

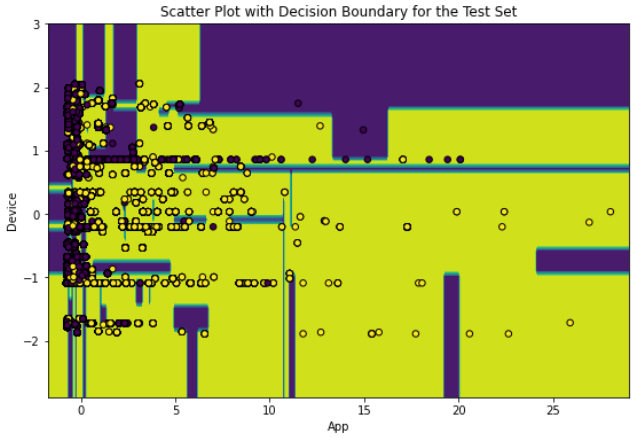


حال که یک مدل به نسبت قابل قبول داریم می خواهیم مرز تصمیم یا decision boundaries را برای 2 فیچری که بیشترین تاثیر را در آن داشتند مشاهده کنیم. برای اینکار ابتدا importance همه فیچرها را مقایسه میکنیم که در نمودار زیر مشخص است.



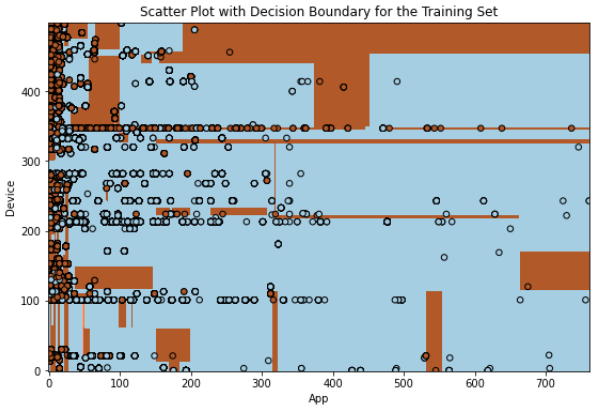
بیشترین تاثیر مربوط به ستونهای 4 و 7 است که همان app و channel هستند. حالا پس از فیت کردن مدل روی این 2 فیچر بررسی میکنیم که در حالت 2 بعدی چطور مرز تصمیم برای آنها ترسیم می شود.

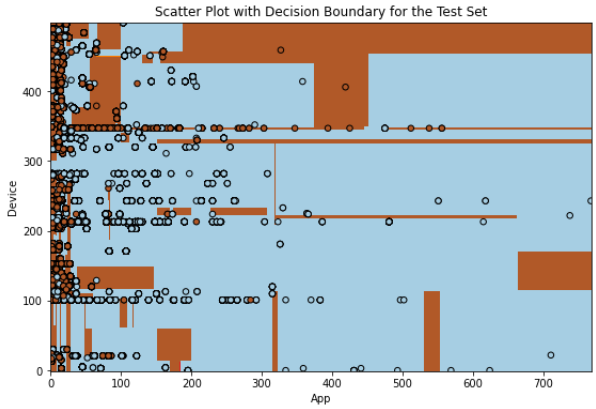




مشخص است شکل شباهت زیادی به یک درخت دارد و تقسیم بندی که در فضا صورت گرفته کاملا به شکل درختی است. با بررسی چشمی به نظر جدا کردن کلاس ها برای این داده ها تقریبا خوب صورت گرفته و این مورد از دقت مدل هم قابل تصور بود.

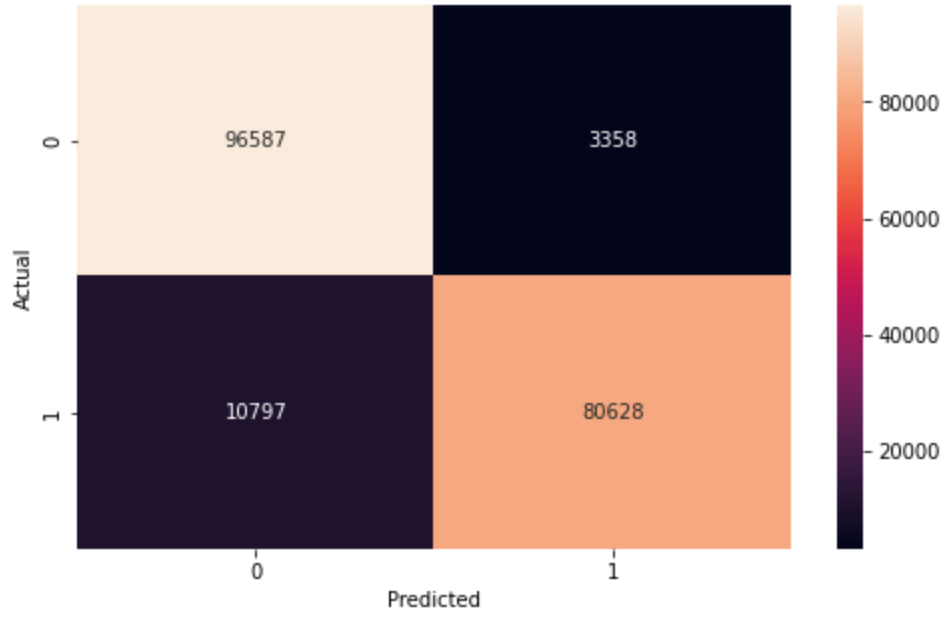
البته موردی که در بالا رعایت نشده ان است که در ستونهای app و channel اعداد به صورت استاندارد اسکیل شده اند و مقادیر اصلی قرار ندارند. از این رو بار دیگر اسکیل را ریورس میکنیم تا اعداد اصلی به دست آیند تا با انها نمودار را رسم کنیم. نمودار به صورت زیر قابل مشاهده است.

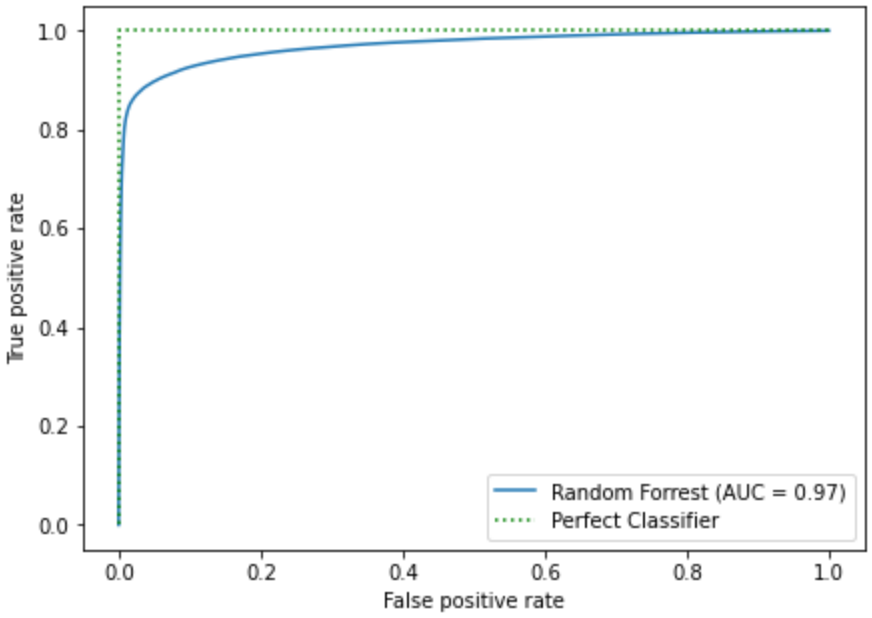




مشخص است یک مقدار تجمع در app کمتر از 20 زیاد است که اصلا دلیل استاندارد کردن داده ها همین موارد است. اما دسته بندی ها شکل به نسبت مناسبی دارند.

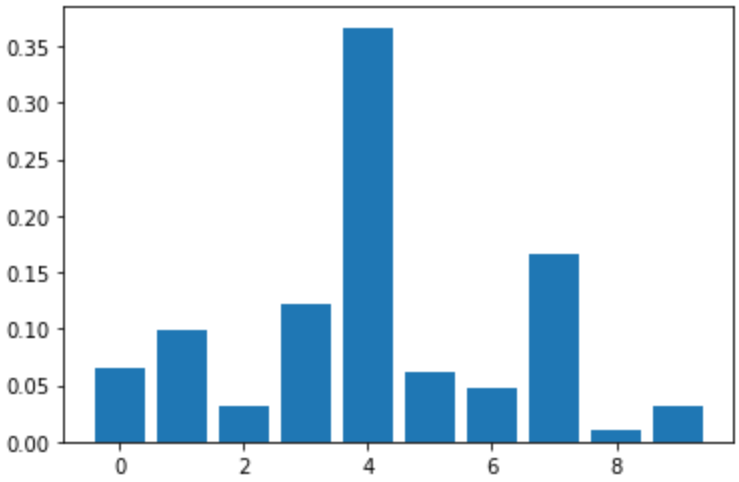
مدل بعدی که بررسی می کنیم random forrest است. در این مدل دقت ما 92 درصد بود. به نسبت درخت تصمیم در این مدل تشخیص کلاس 0 بهتر بود و نزدیک به 97 درصد داده های این کلاس به درستی لیبل زده شدند. در نمودار های زیر confusion matrix و auc curve را با هم میبینیم که به نظرم نیاز به توضیح بیشتری ندارد.



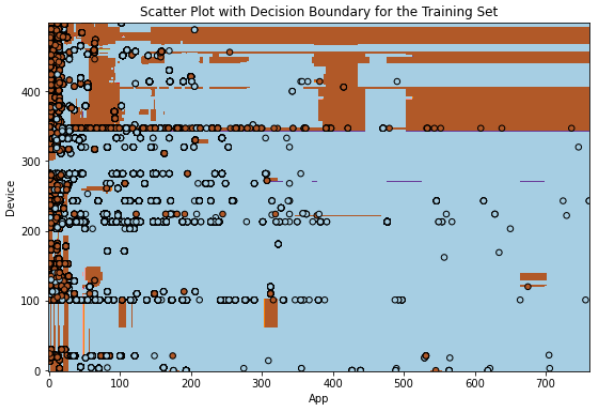


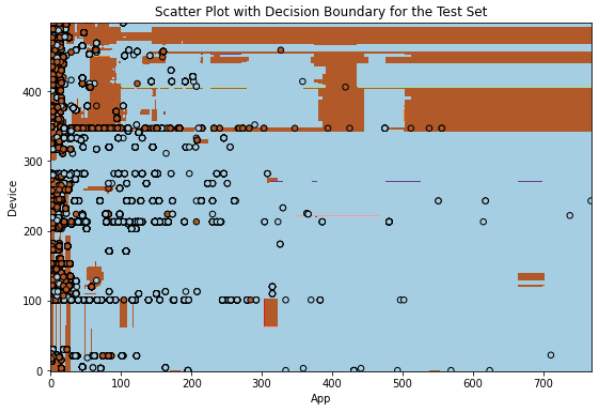
تنها نکته ای که باید ذکر کنم این است که مشخص است که میزان منحنی چه مقدار به حالت خوب نزدیک است و مساحت زیر سطح منحنی مقدار بالایی دارد.

برای نمودار مرزهای تصمیم random forrest نیز باید بر اساس 2 فیچر app و channel است که طبق نمودار زیر بیشترین تاثیرها را داشتند نمودار ها را رسم کنیم.



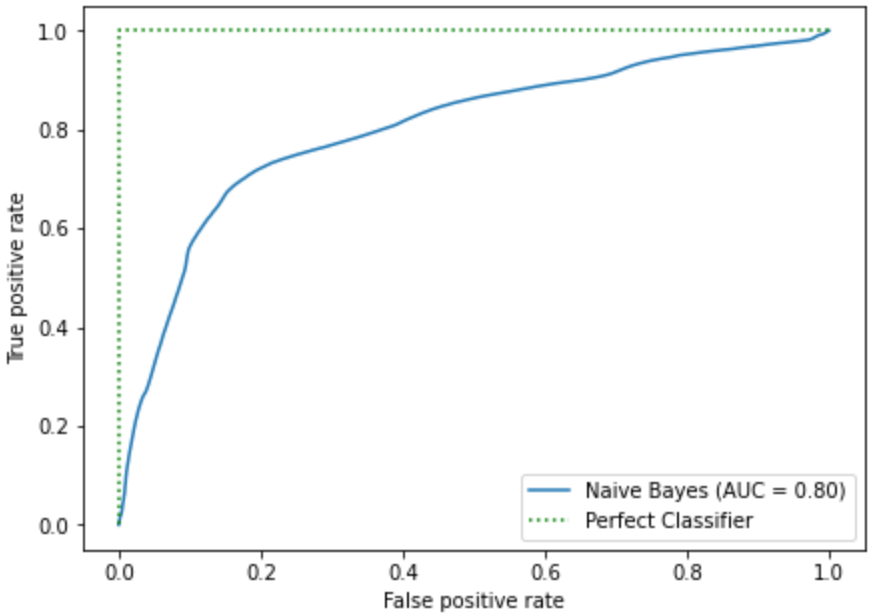
پس از ان نمودار را بر اساس همان ریورس استاندارد اسکیل ها که مقادیر واقعی ستونها را می دهد رسم کرده ایم.



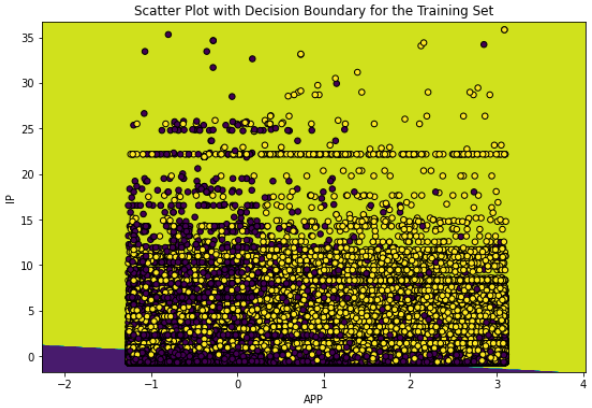


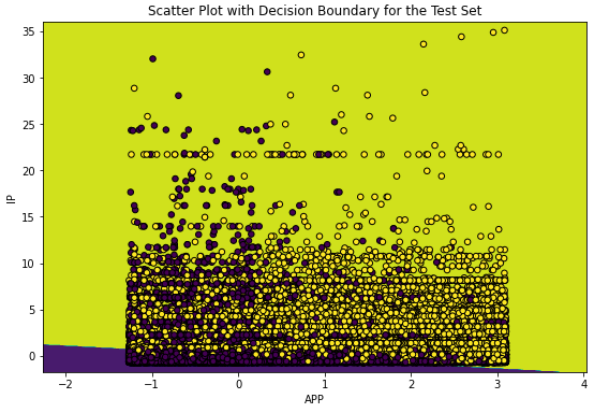
نمودار بسیار مشابه با درخت تصمیمان شد که علت آن تقریبا مشخص است . چون 2 تا از بهترین فیچرهای جفتشان یکسان بودند و دقتی که گرفتیم نیز برای هر دو مدل بالا بود که نشان می دهد این دو مدل پیشبینی های خوبی را دارند.

در Naive Bayes که با متد GaussianNB پیاده سازی شده است با توجه به اینکه به نظر توزیع داده ها بررسی می شود ، ابتدا داده ها را بدون اینکه بالانس کنیم به مدل دادیم که دقت ان 82 درصد بود ولی مشکل آن بود که f1-score برای کلاس 0 حدود 90 درصد و برای کلاس 1 تقریبا 48 درصد بود که میزان بسیار پایین و غیر قابل قبولی دارد و مدل ما کاملا دارد به اصطلاح کفه ترازو را به سمت حدس زدن کلاس 0 بالا می برد. نمودار زیر نیز AUC ان را نمایش می دهد.



یکی از دلایلی که میتوان برای این دقت پایین مطرح کرد فیچرهایی هستند که بیشترین تاثیر را داشتند. برای این مدل فیچر ip از همه مهمتر تلقی شده بود در حالی که در 2 مدل پیشین که دقت مناسبی داشت این فیچر جزو فیچرهایی با تاثیر بالا محسوب نمی شد. البته لازم به ذکر است که این دلیل قطعی نیست زیرا ممکن است با همین فیچر نیز بتوان مدل خوبی طراحی نمود. نمودار زیر مرز تصمیم برای این الگوریتم است که مشخصا اصلا مناسب و درست نیست همچنین به علت شباهت فیچر هایش با لجستیک رگرسیون نمودارهایش نیز به هم شبیه هستند. البته نمودار لجستیک در کد آورده نشده است.



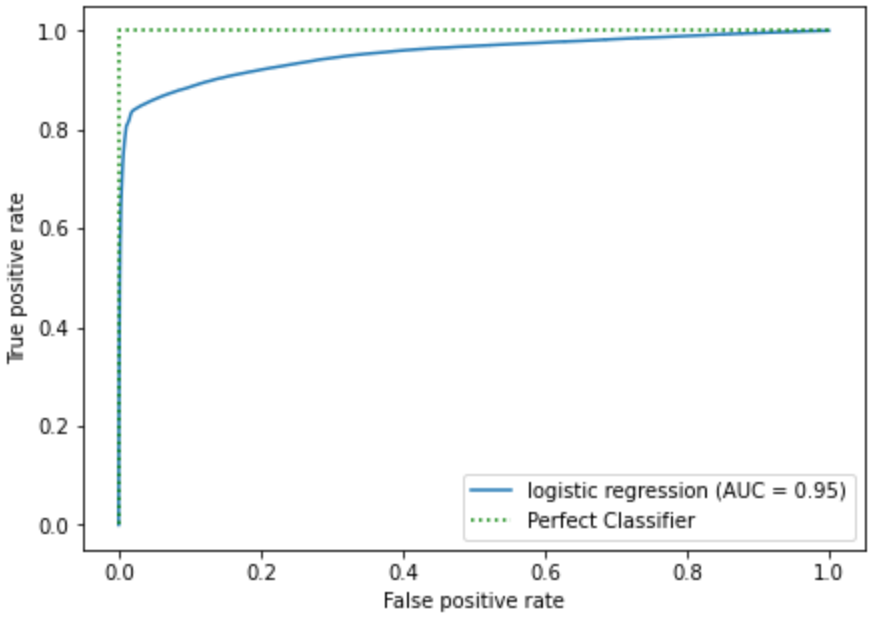


پس از بالانس کردن به روش های مختلف نیز نتایج برای این مدل زیاد خوشایند نبود . از این رو حالا که همه مدل ها به شکل های مختلف بررسی شد نوبت به آن میرسد تا فیچرهای دیگر را روی این مدل ها به انواع مختلف امتحان کنیم. در هر کدام از مدلها که تاثیر خوب و دقت بالاتری نسبت به چیزی که در بالا وجود داشت ببینیم آن را گزارش و نمودارهای مربوط به ان مدل را در زیر مشاهده و بررسی می کنیم.

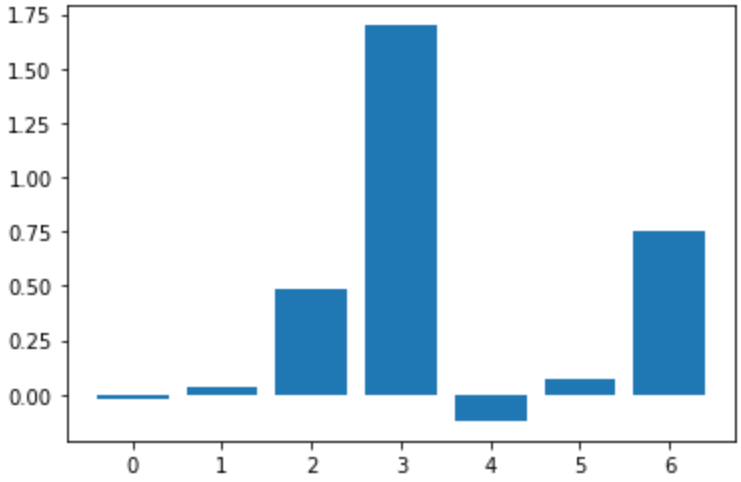
اولین تغییراتی که در مدل ایجاد کردم ان بود که ستون هایی که طبق تشخیص کلاس بندی هایی که دقت خوبی داشتند به نظر تاثیر کمتری داشتند را از دیتاست پاک کردیم. که ستون های ', 'os','device' , 'previous_downloads' 'hour' , 'day'' از این دسته بودند.

ابتدا از دیتاست interactons استفاده کردیم که تاثیر خوبی روی دقت الگوریتم ها برای پیش بینی درست نداشت. پس از آن به سراغ دیتاست دیگری رفتیم و چون میخواستیم فیچر های دیتاست catboost_encodings را اضافه کنیم پس نیازی به app و channel هم نبود و آنها را نیز پاک کردیم.

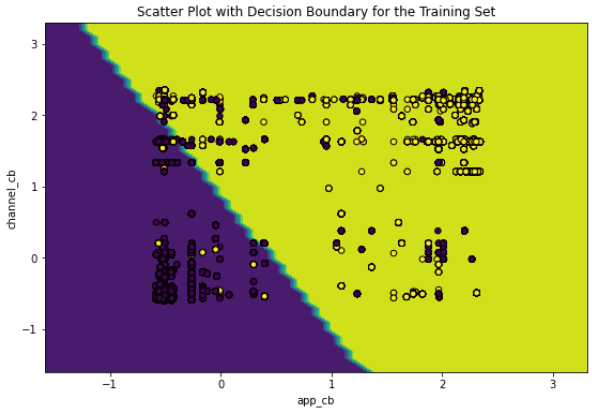
حالا زمانی که این داده ها را بدون سمپل گیری با متد 'class_weight='balanced به logistic regression مان دادیم میزان دقت ما به 95 درصد رسید که بسیار خوب تلقی می شود و به نسبت دفعه قبل که 80 درصد بود پیشرفت چشمگیری داشت. نمودار منحنی AUC آن را نیز در زیر مشاهده می کنیم.



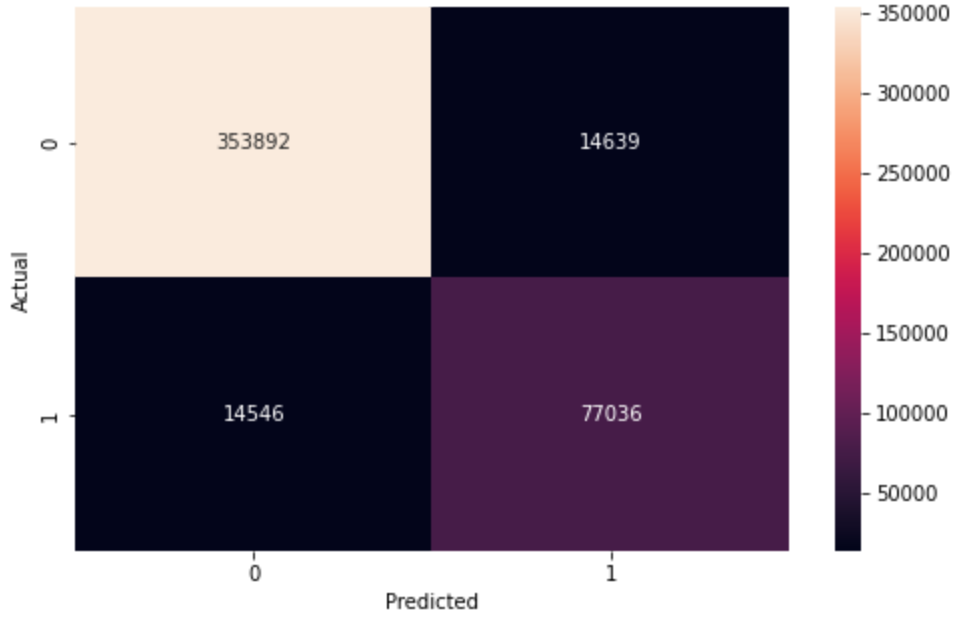
همچنین با توجه به نمودار زیر مهمترین فیچرها در این کلاس بندی ستونهای 3 و 6 هستند که ستونهای app-cb و channel-cb هستند.

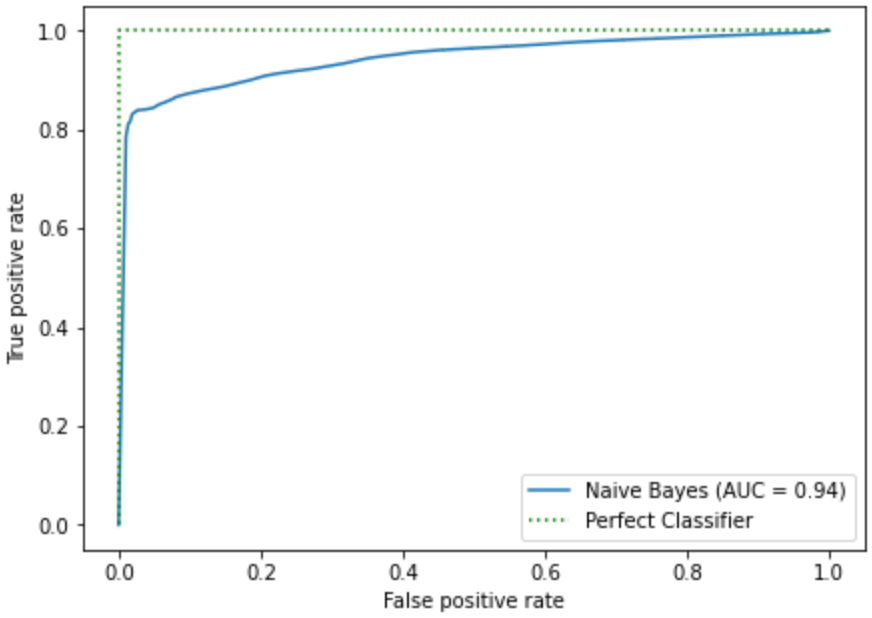


نمودار زیر نیز مرز تصمیم برای 2 فیچر گفته شده است که تقسیم به نظر خوبی می آید.

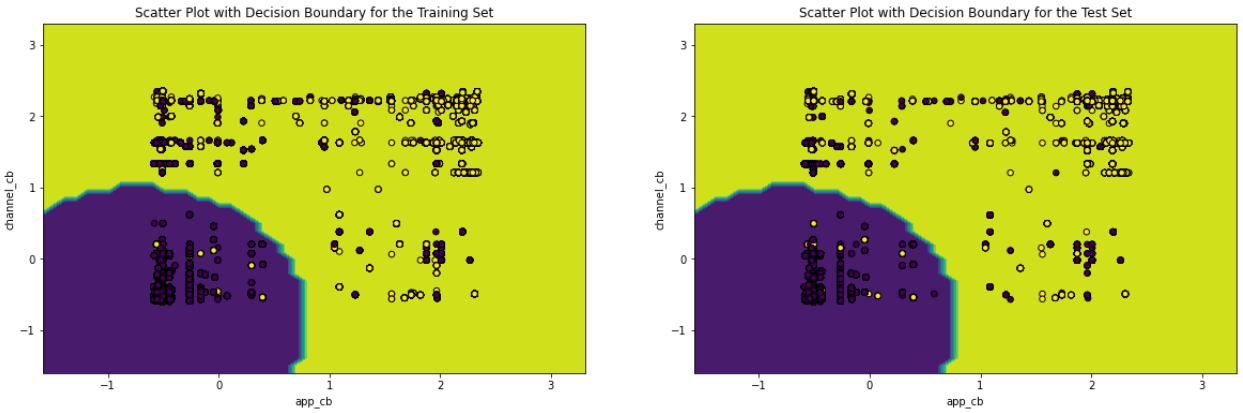


زمانی که همین دیتاست را بدون آنکه بالانس کنیم به Naive Bayes میدهیم میزان دقت این الگوریتم نزدیک به 94 درصد شد که به نسبت قبل 12 درصد افزایش داشت که پیشرفت خوبی است. نمودار های زیر confusion matrix و auc curve هستند که شاید اندکی کلاس های با لیبل 0 بهتر تشخیص داده شدند اما به طور کل مدل خوبی به نظر می آید.

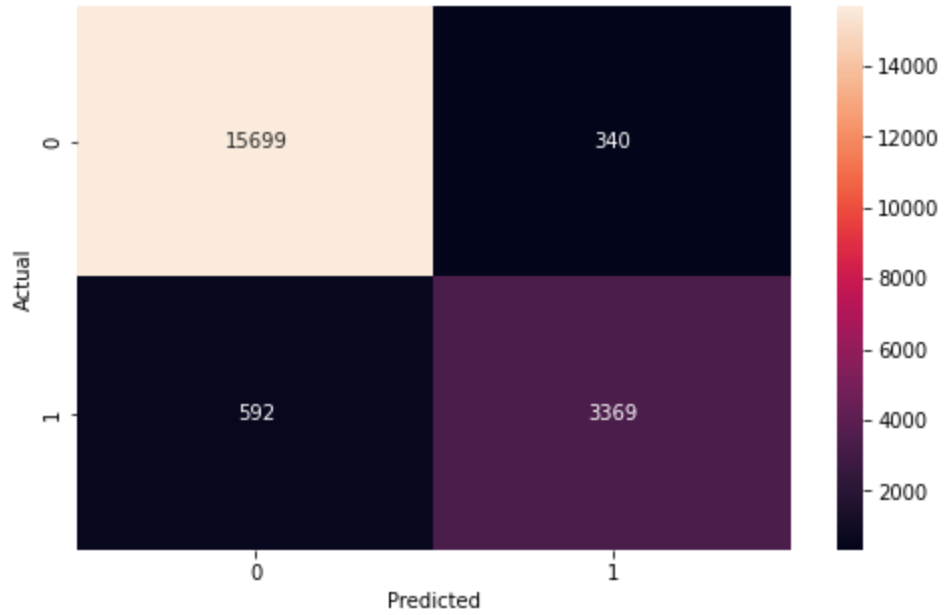




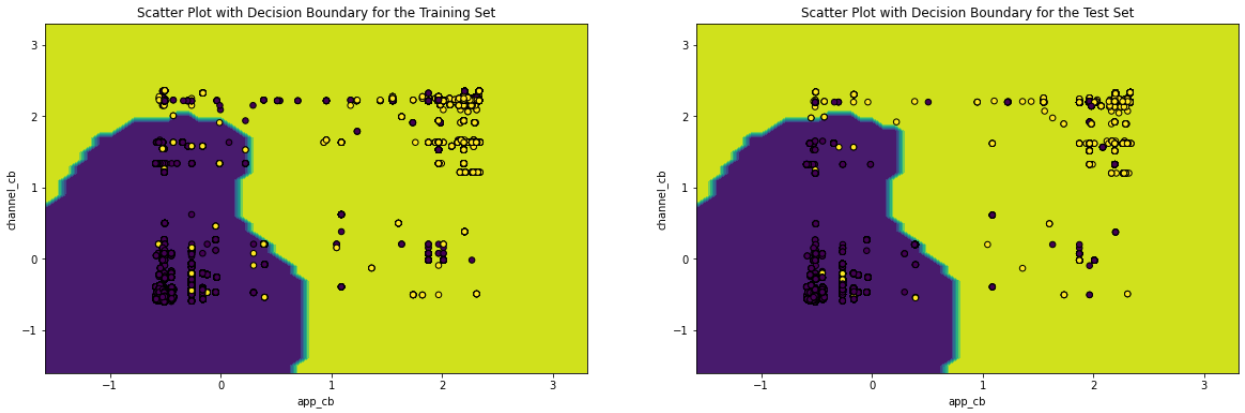
در این مدل نیز تاثیر ستونهای app-cb و channel-cb بیشتر از بقیه هستند از این جهت مرزهای تصمیم را روی این دو فیچر اعمال میکنیم که به صورت زیر شده است.



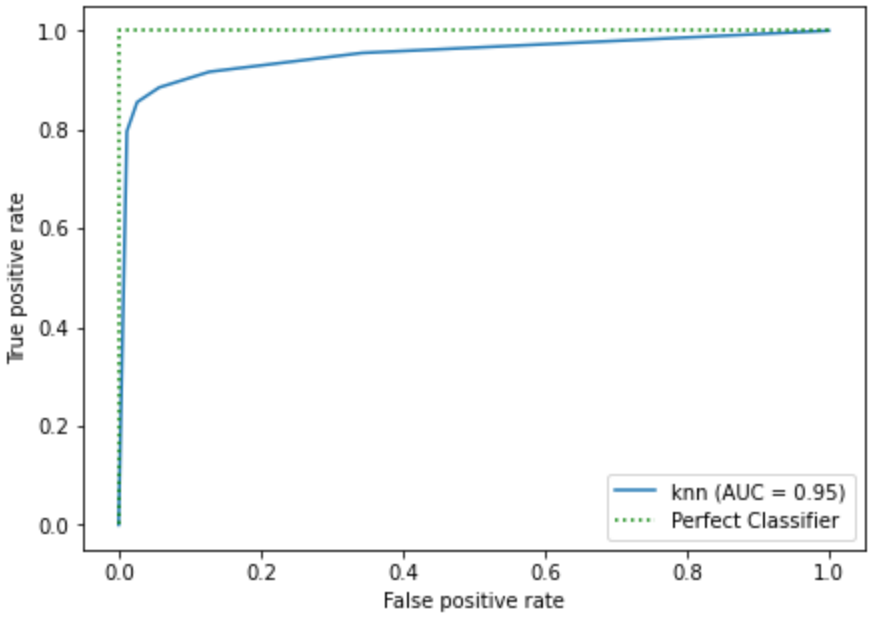
بعد از این کار دیتاست جدیدمان را بدون بالانس کردن و گرفتن سمپل 100000تایی با متد class_weight=balanced روی SVM فیت میکنیم که در این حالت دقت ما به بیش از 95 درصد رسید که میزان بالاتری نسبت به حالت قبلی دارد. نمودار زیر confusion matrix این متغیر را نمایش می دهد.



نمودار زیر نیز مرز تصمیم برای 2 فیچر مهمتر در کلاس بندی یعنی app-cb و channel-cb است که بی شباهت با Naive bayes نیست.



الگوریتم knn را نیز با دیتاست جدید امتحان کردیم. البته قبل از شروع کار downsampling برای بالانس شدن داده ها روی آن صورت گرفته است. میزان دقت این الگوریتم نزدیک به 92 درصد بود که بازهم پیشرفت داشته است. برای آنکه گزارش بیش از حد شلوغ نشود چون الگوریتم های ما مرز تصمیم های نزدیک به هم دارند تنها به نمودار AUC Curve بسنده می کنیم که نشان دهنده دقت بالای مدل می باشد.



حالا می توان گفت برای همه الگوریتم ها مدلی با دقت بالا پیدا کرده ایم. کمترین دقت در الگوریتم های ما 92 درصد بود که میزان قابل توجه ای به نظر می آید.

همچنین نکته ای که قابل توجه بود آن است که به جز الگوریتم های Decision Tree و Random Forrest بقیه الگوریتم ها روی داده های با catboost_encodings ها دقت بالاتری داشتند ولی دو الگوریتم ذکر شده با همان داده های خام دقت بالایی داشتند که به نظرم نشان دهنده توانایی تقسیم بندی صفحه یا همان مدل در این الگوریتم های درختی است.