به نام خدا

مهرانه مقتدائی فر ۹۷۲۲۲۰۸۶

تاریخ: ۱۴۰۰/۱۰/۱۷

گزارش تمرین سری ۴ - بخش دوم

درس: مبانی علوم داده

استاد درس: دکتر خردپیشه

در این بخش با استفاده از دیتاست داده شده به پیاده سازی مدل های classification پرداخته شده است. این دیتاست مربوط به ad_tracking بررسی میکند که هر شخص پس از مشاهده تبلیغ اپلیکیشن آیا به سراغ دانلود کردن آن میرود یا خیر. در این بخش باید طبقه بندی دو کلاسه انجام دهیم و دو حالت که آیا دانلود میکند(1) یا دانلود نمیکند(0) را بررسی کنیم.

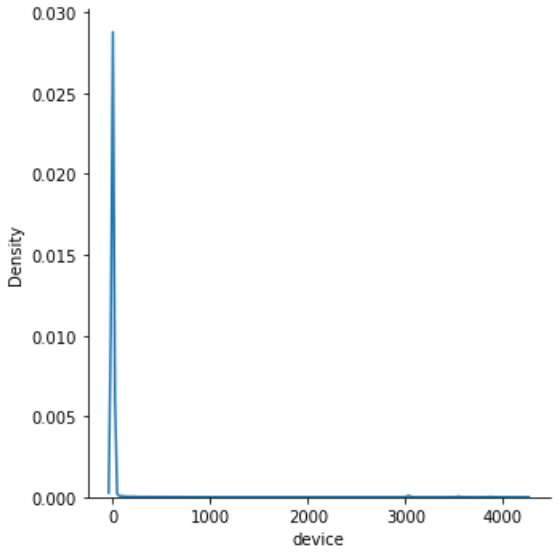
خواندن و پیش پردازش داده ها

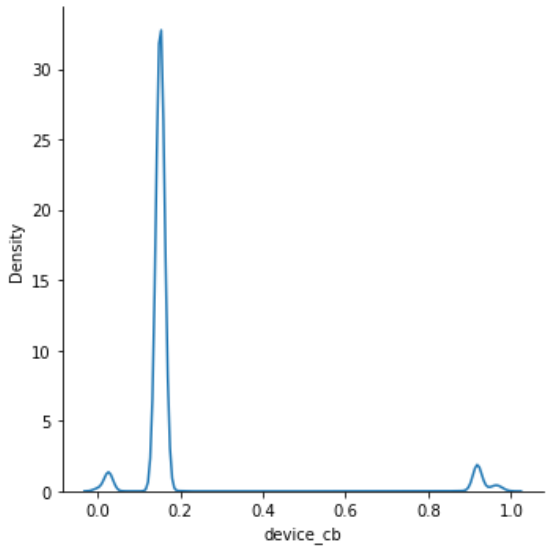
با توجه به دیتایی که در Kaggle وجود داشت، آن را لود کرده و میبینیم که این دیتاست شامل یک فایل scv و چندین فایل pqt است. فایل های pqt در واقع تعداد فیچر اضافی است که جداگانه از دیتاست ما قرار داده شده است. این ویژگی ها مانند downloads، time_deltas، past_6hr_events اطلاعات خوبی را در اختیار ما قرار میدهند. پس نیاز است تا این فیچر ها را به دیتاست اصلی اضافه کنیم.

همچنین ۴ ستون app,os,device,channel در دو فایل pqt به صورت encode شده قرار داده شده‌اند. یک فایل catboost encoding هست که این روش چیزی مانند target encoding می‌باشد اما تفاوت هایی دارد. در هرصورت نوعی انکودینگ با توجه به ستون هدف است.

فایل دیگری بر مبنای count encoding وجود دارد که همانطور که مشخص است، بر اساس "تعداد" encoding انجام شده است.

در اینجا این ۴ ستون با ستون های موجود در catboost encoding جایگزین شدند زیرا مقادیر این داده ها پس از استفاده از catboost توزیع نرمال تری داشتند برای مثال برای فیچر device قبل و بعد از encode شدن داریم:





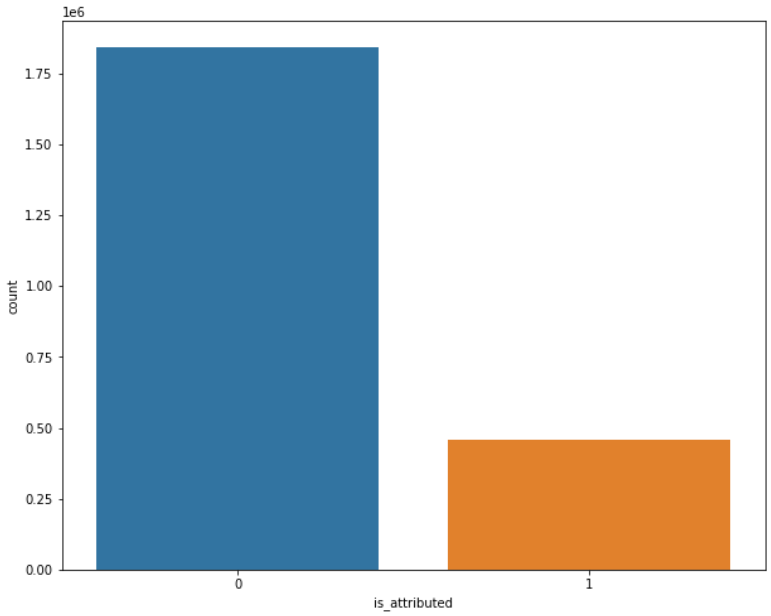
همانطور که مشخص است، توزیع داده ها بهتر شده است. همچنین مقدار نمونه ها نسبت به حالت count encoding کمتر است و داده های مناسب تری برای مدل ها هستند. پس به جای این ۴ستون مقدار catboost آنها را قرار دادیم.

فایل دیگری تحت عنوان interactions وجود دارد، این مقادیر نشان دهنده مقادیر مختلف ستون ها دو به دو با یکدیگر است. استفاده از این فایل و اضافه کردن این ستون ها به دیتا موجب میشود که دیتاست بزرگتری داشته باشیم و بعد دیتای ما افزایش یابد. با بررسی برخی مدل ها برروی این دیتا ها دیدیم که دقت خوبی ندارند. پس تنها مقادیر encode شده را به دیتا اضافه کردیم و با آن مدل های مختلف را بررسی نمودیم.

علاوه بر این فیچرها باید توجه داشت که دیتاست ما نمونه های بسیار زیادی دارد. این نمونه ها حدود ۲ میلیون و ۳۰۰ هزار تا هستند. اگر از تمامی این دیتا استفاده کنیم شاهد کرش کردن مدل ها میشویم زیرا حجم این داده ها بسیار زیاد است.

در نیتجه برای کاهش این حجم از متد های undersampling استفاده میکنیم.

همانطور که در نمودار زیر نیز مشخص است، مقادیر کلاس 0 و 1 ما بسیار unbalanced هستند. و بهتر است برای کاهش نمونه ها، مقادیر کلاس 0 را کم کنیم.



قبل از هرچیزی نیاز است تا مقادیر test را به عنوان داده های دیده نشده از دیتاست جدا کنیم تا هیچ عملیاتی برروی آنها صورت نگیرد. 80% داده ها را به عنوان داده های آموزشی و 20% را به عنوان داده تست در نظر گرفتیم.

حال برای آنکه داده های آموزشی را کاهش دهیم، از چندین روش برای Undersample کردن استفاده کردیم.

روش اول NearMiss:

در این روش و به طور کل روش هایی که با همسایه های داده ها سر و کار دارند، اینگونه عمل میشود که ابتدا، فاصله بین تمام نمونه های کلاس اکثریت و نمونه های کلاس اقلیت محاسبه میشود، سپس k نمونه از کلاس اکثریت که کمترین فاصله را با کلاس اقلیت

دارند انتخاب می شوند. اگر n نمونه در کلاس اقلیت وجود داشته باشد، منجر به انتخاب k*n نمونه از کلاس اکثریت می شود. روش NearMiss نیز نمونه هایی از کلاس اکثریت را انتخاب می کند که میانگین فاصله آنها تا نزدیکترین نمونه کلاس اقلیت کوچکترین باشد. انتخاب این روش برای sample کردن مناسب نبود، زیرا تعداد داده ها را به صورت مناسب کاهش نمیداد و داده هایی که نیازبودند را از بین میبرد و به همین علت دقت مدل ها پایین می‌آمد.

روش دوم Neighbourhood Cleaning Rule:

این روش هنگام نمونه برداری از مجموعه داده ها با نمونه های اکثریت و اقلیت به طور جداگانه سروکار دارد. این روش توسط یک الگوریتم و انتخاب سه نزدیک ترین همسایه های داده، تعدادی داده را از کلاس اکثریت حذف میکند تا نزدیک به کلاس اقلیت شود.

این روش نیز برای سمپل کردن مناسب نبود زیرا به اندازه کافی تعداد کلاس 0 را کم نمیکرد. پس از اعمال این روش هنوز تعداد نمونه ها در کلاس 0 خیلی بیشتر از کلاس 1 بود.

روش سوم RandomUnderSmapling:

در این روش به صورت خیلی تصادفی و بدون هیچ محاسبه‌ای نمونه هایی از کلاس 0 حذف میشدند تا تعداد دو کلاس به طور برابر شود و در اینجا تعداد نمونه های هر دو کلاس به 365264 رسید. این روش با اینکه به صورت تصادفی داده ها را از بین میبرد اما از روش های دیگر عملکرد بهتری داشت.

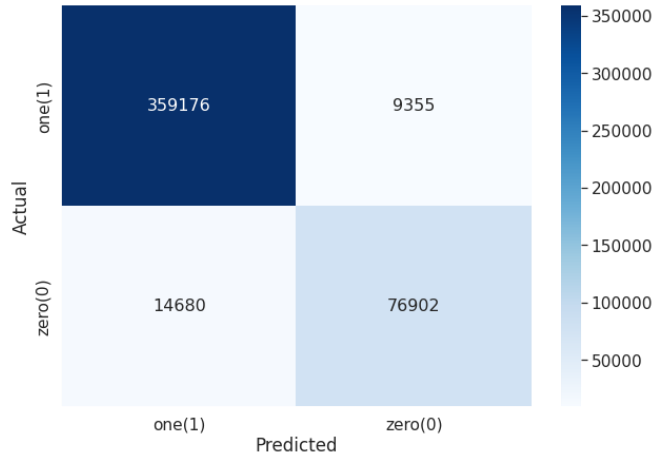
پس از سمپل گرفتن از داده ها و کم کردن تعداد آنها نیاز است تا با استفاده از StandardScaler آنها را scale کنیم و سپس به پیاده سازی مدل های مختلف بر روی این داده ها بپردازیم.

پیاده سازی مدل های Classifier

برای هر یک از مدل ها، ابتدا مدل را برروی نمونه ها سمپل شده پیاده سازی کردیم، برای هر کدام دقت ، ماتریس confusion و AUC curve را میبینیم. سپس همان مدل را برروی دو فیچری که بیشترین تاثیر را در classification ما داشتند پیاده کرده و نمودار مربوط به decision boundary آنها را مشاهده خواهیم کرد. در انتها نیز برای مدل هایی که روش class_Weight داشتند آن را نیز برروی داده های سمپل نشده امتحان کردیم تا ببینیم با استفاده از این متد عملکرد مدل ها در مقایسه با حالتی که داده ها را سمپل کردیم چگونه بوده است.

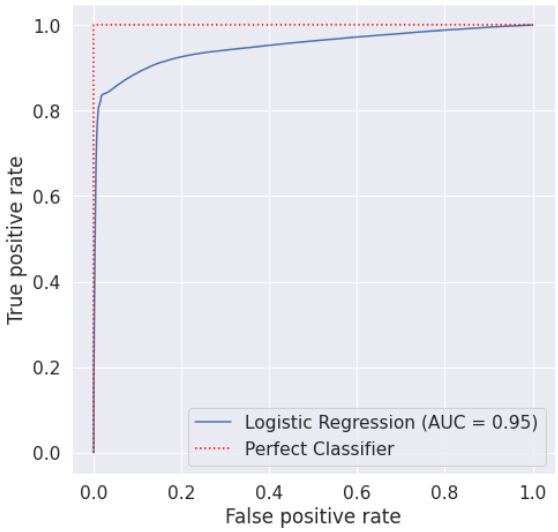
مدل Logistic Regression:

دقت این مدل برروی داده های سمپل شده حدود 94.7 است، در زیر اطلاعات دیگر مانند precision،recall و f1-score و همچنین ماتریس confussion را میبینیم:



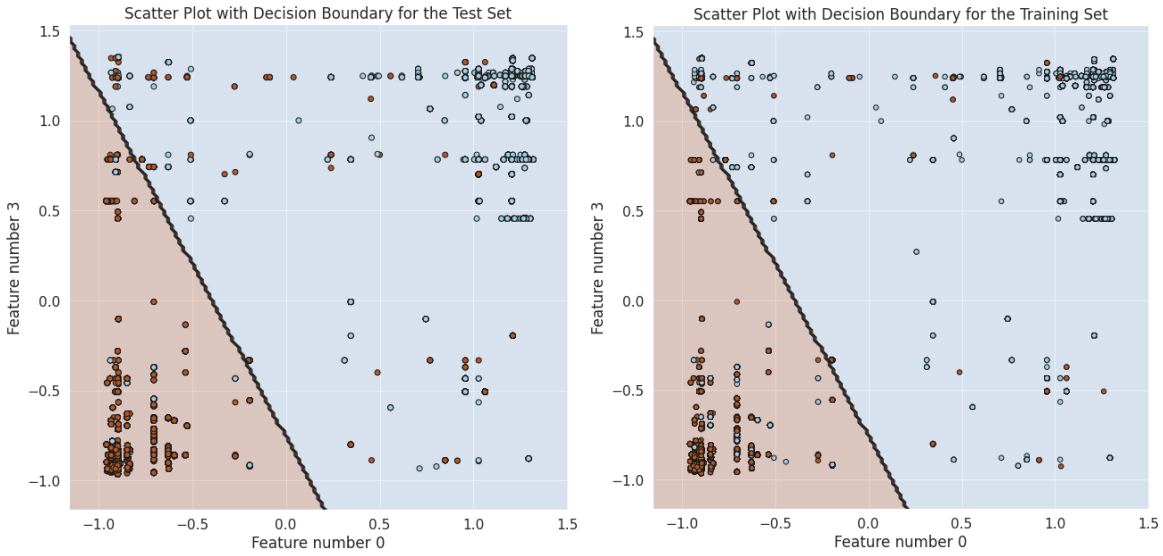
	precision	recall	f1-score	support
0	0.96	0.97	0.97	368531
1	0.89	0.84	0.86	91582
accuracy			0.95	460113
macro avg	0.93	0.91	0.92	460113
weighted avg	0.95	0.95	0.95	460113

AUC curve :

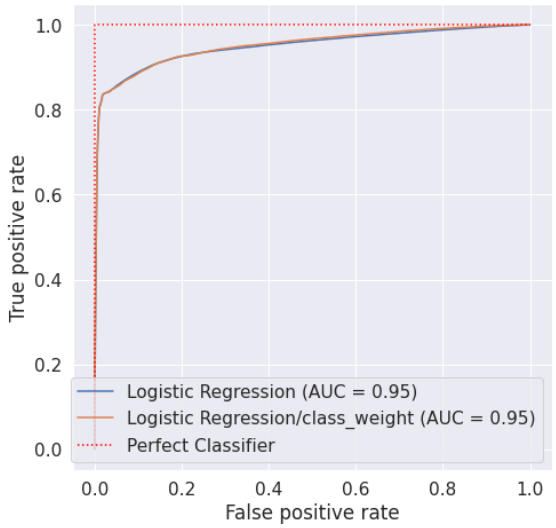


دو فیچری که بیشترین تاثیر را در این classification داشتند، فیچر شماره 0 و3 بوده اند. این دو فیچر app و channel هستند. حال با استفاده از این دو فیچر نمودار decision boundary را مشاهده میکنیم

*این نمودار ها تنها 100 هزار نمونه را نشان میدهند



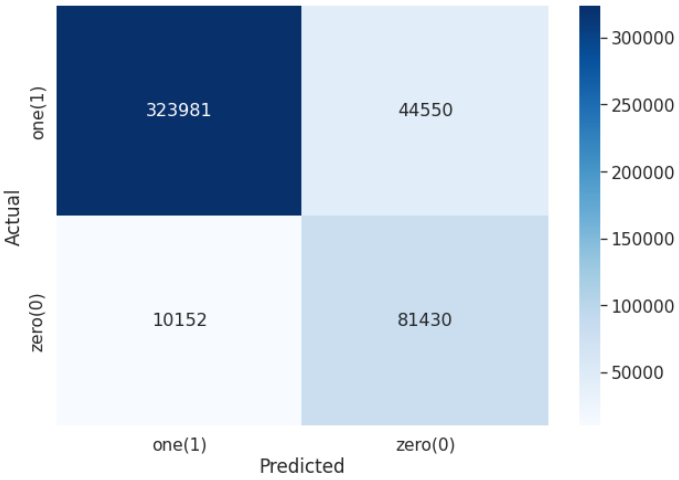
در حالتی که مدل را با استفاده از class_weight='balanced' برروی داده های سمپل نشده پیاده کردیم، دقت مدل کمی بیشتر و در حدود 95.2 شد. نمودار AUC را برای هر دوحالت همزمان باهم مشاهده میکنیم:



مدل Decision Tree:

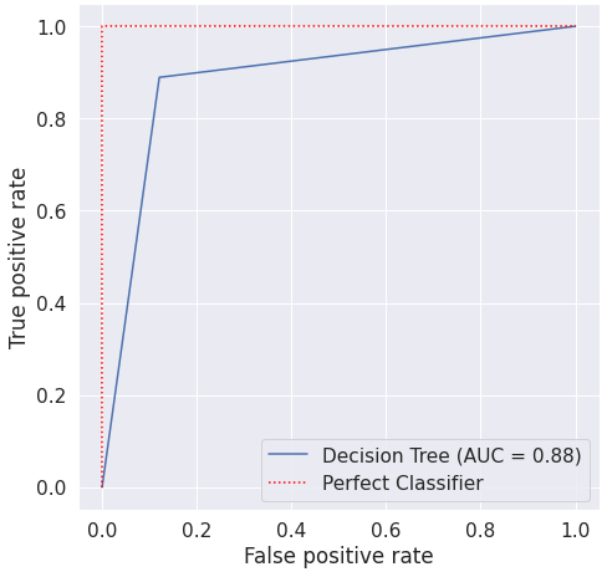
نتایج مدل برروی داده های سمپل شده:

دقت حدود 88.11 درصد است، ماتریس confusion و مقدار معیار های دیگر را باهم میبینیم:

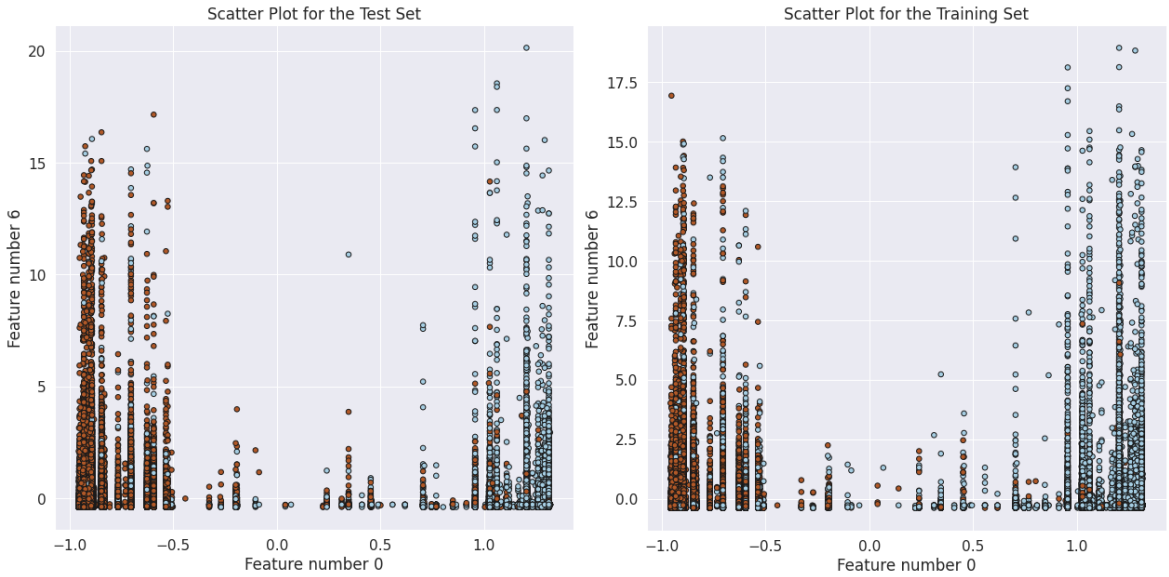


	precision	recall	f1-score	support
0	0.97	0.88	0.92	368531
1	0.65	0.89	0.75	91582
accuracy			0.88	460113
macro avg	0.81	0.88	0.84	460113
weighted avg	0.91	0.88	0.89	460113

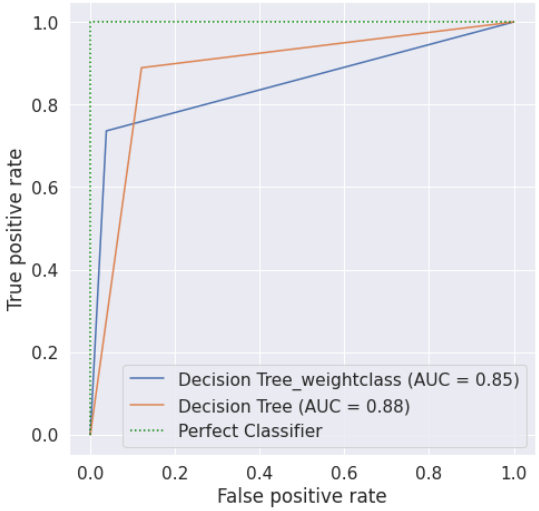
AUC curve:



برای این مدل دو فیچر 0 و 6 بیشترین تاثیر را داشتند، نمودار های زیر را برای این دو مقدار میبنییم:

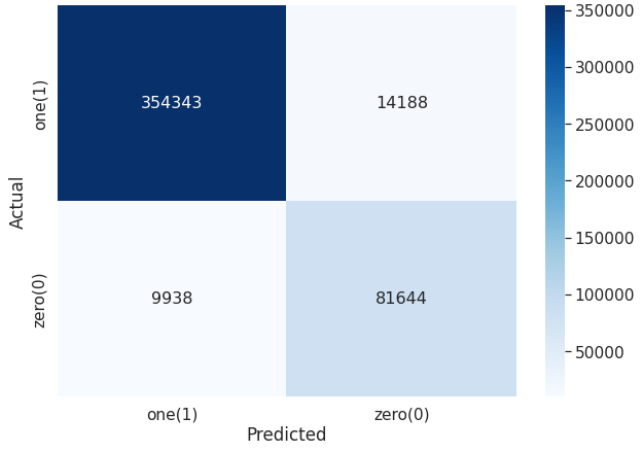


با استفاده از روش class_weight و بدون استفاده از داده های سمپل شده دقت مدل به حدود 91 درصد رسید، نمودار AUC برای هردوحالت:



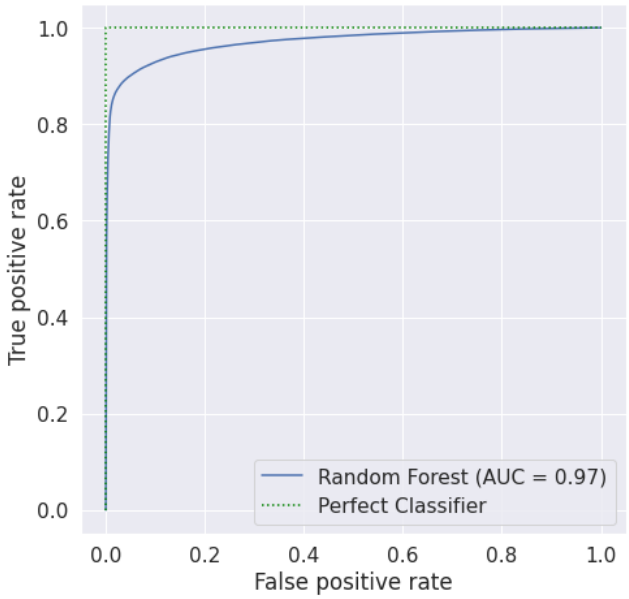
مدل Random Forest:

دقت این مدل برروی داده های سمپل شده تقریبا 95% است. ماتریس confusion و معیار های درستی دیگر را در زیر میبینیم:

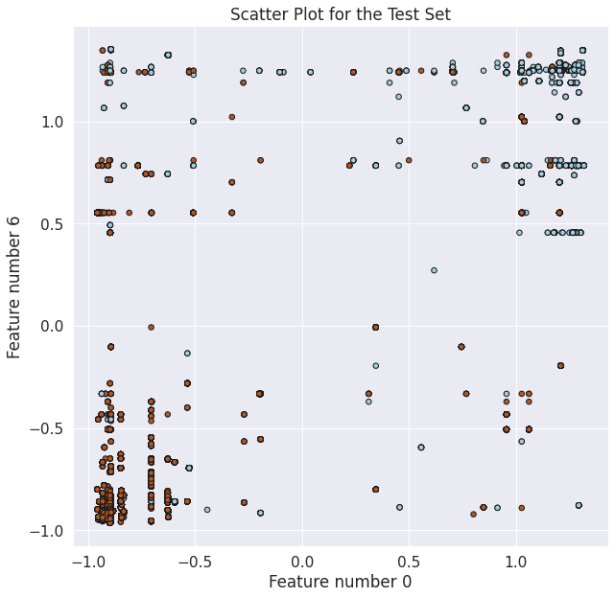


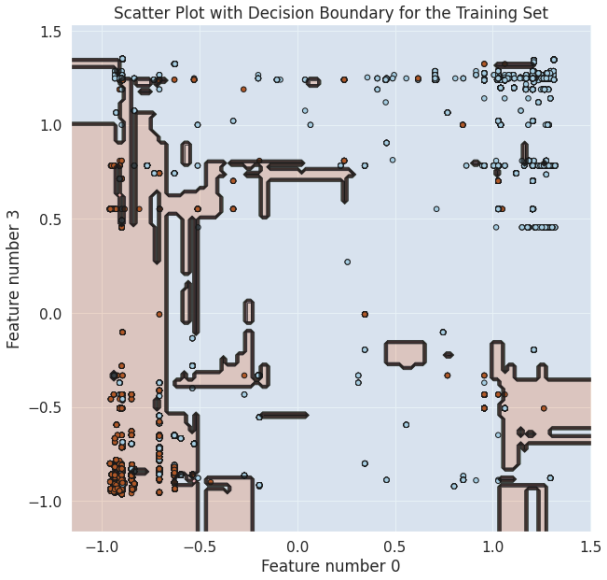
	precision	recall	f1-score	support
0	0.97	0.96	0.97	368531
1	0.85	0.89	0.87	91582
accuracy			0.95	460113
macro avg	0.91	0.93	0.92	460113
weighted avg	0.95	0.95	0.95	460113

AUC curve:

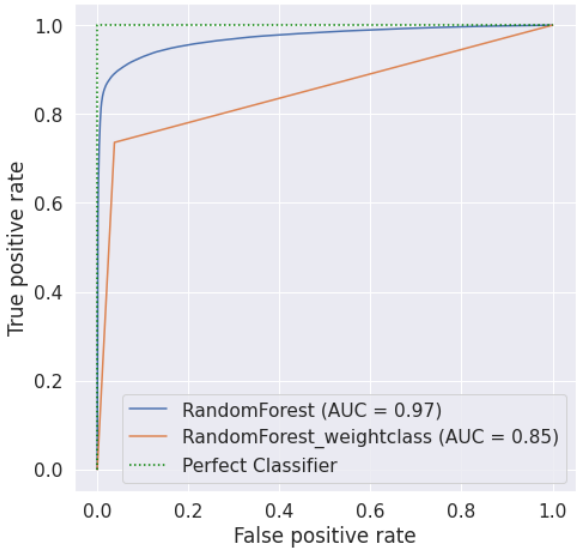


برای این مدل دو فیچر 0 و 3 بیشترین تاثیر را داشتند:





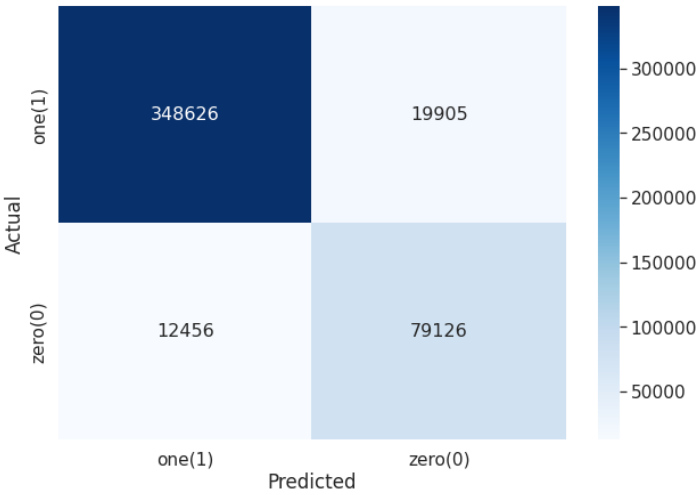
استفاده از روش class_weight برای این مدل خوب نبود و دقت را کمی پایینتر آورد و به حدود 91% رسید.



مدل K nearest Neighbour:

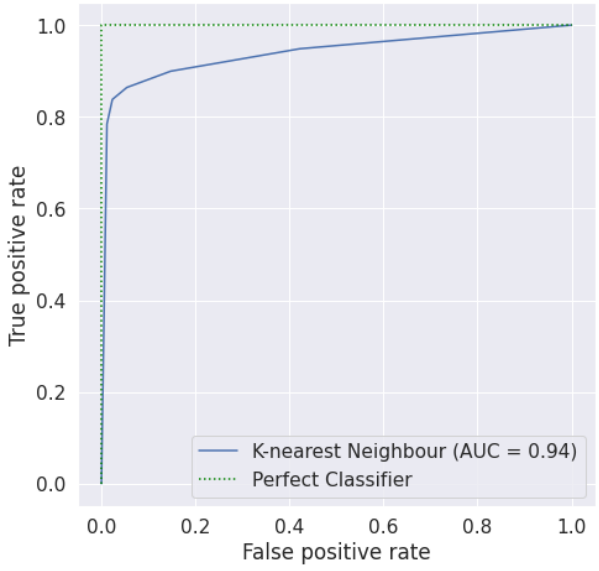
در این مدل چون مقدار n_neighbors داده نشده است، خود مدل 5 همسایه را برای انجام الگوریتم انتخاب میکند.

دقت این مدل برروی داده های سمپل شده حدود 93% است.

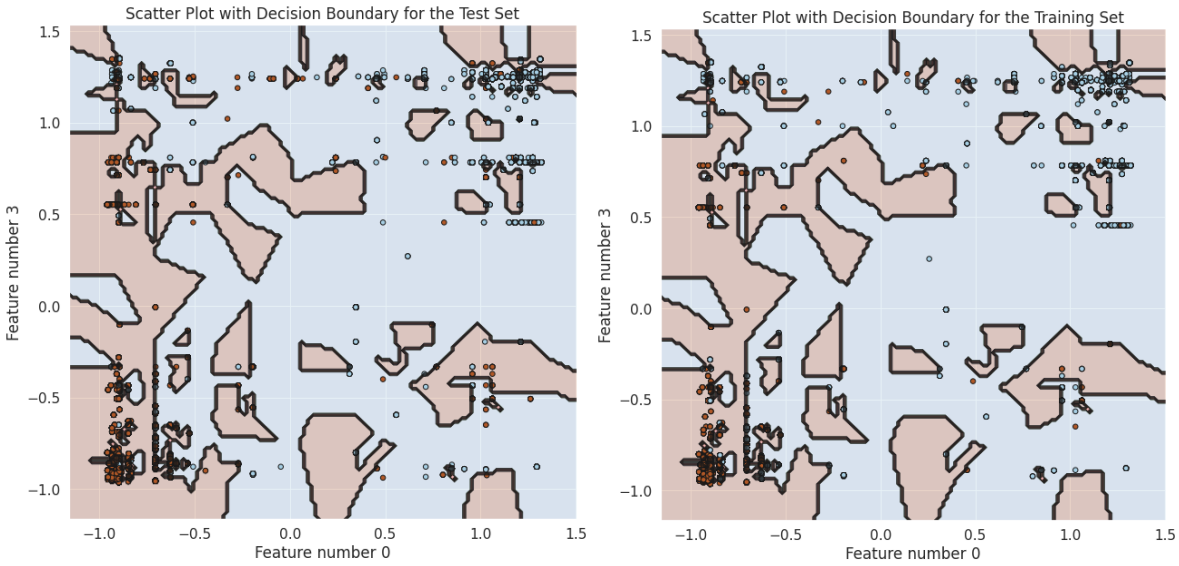


	precision	recall	f1-score	support
0	0.97	0.95	0.96	368531
1	0.80	0.86	0.83	91582
accuracy			0.93	460113
macro avg	0.88	0.90	0.89	460113
weighted avg	0.93	0.93	0.93	460113

AUC curve:



و باز هم دو فیچر 0 و 3 بیشترین تاثیر را برروی مدل داشتند. مرز تصمیم این مدل کمی متفاوت تر از بقیه مدل ها می باشد.

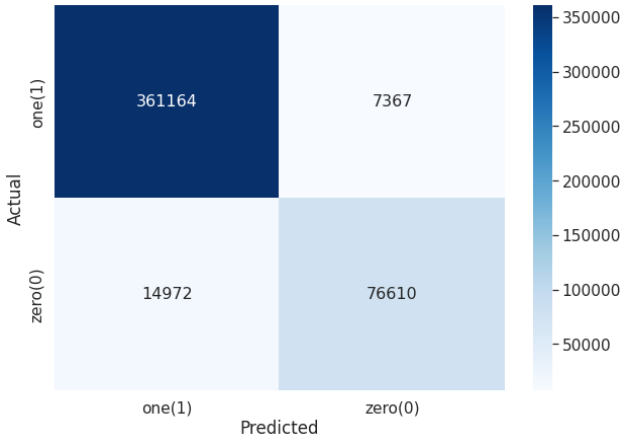


برای این مدل متد class_weight قابل استفاده نبود زیرا برای آن تعریف نشده است.

مدل SVM:

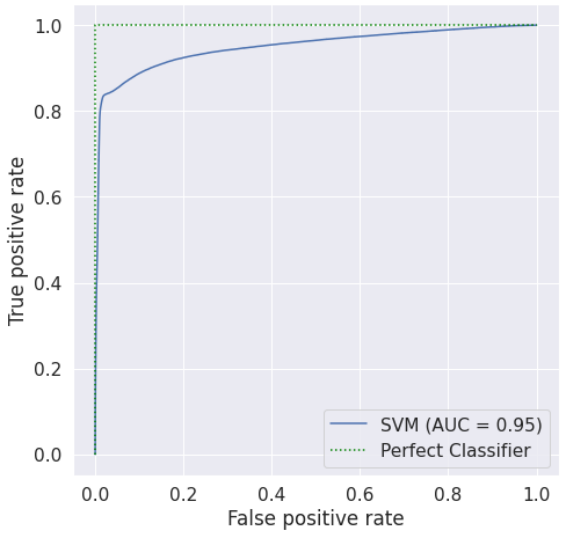
این مدل به علت محاسبات زیاد، پیاده کردن آن برروی تعداد 360 هزار داده بسیار زیاد طول کشید، به همین علت تنها برروی 100 هزار داده به صورت رندم انجام شده است. این 100هزار داده ابتدا balance شده هستند و ۵۰۰۰ تا مربوط به کلاس 0 و ۵۰۰۰ تا نیز مربوط به کلاس 1 است.

دقت این مدل برروی این 100هزار داده حدود 95% است.

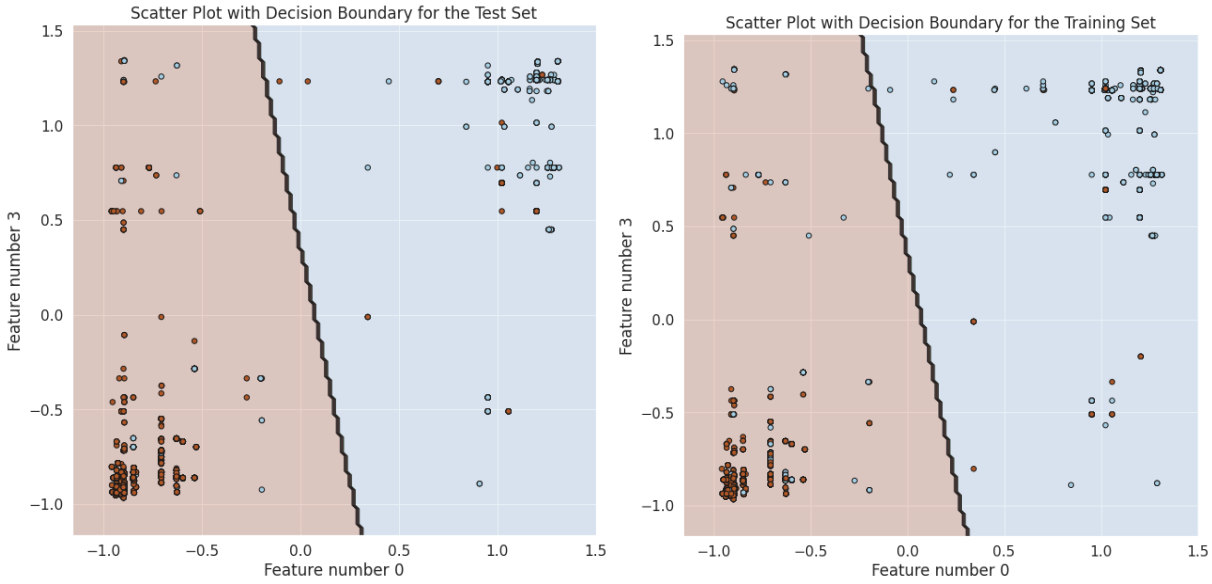


	precision	recall	f1-score	support
0	0.96	0.98	0.97	368531
1	0.91	0.84	0.87	91582
accuracy			0.95	460113
macro avg	0.94	0.91	0.92	460113
weighted avg	0.95	0.95	0.95	460113

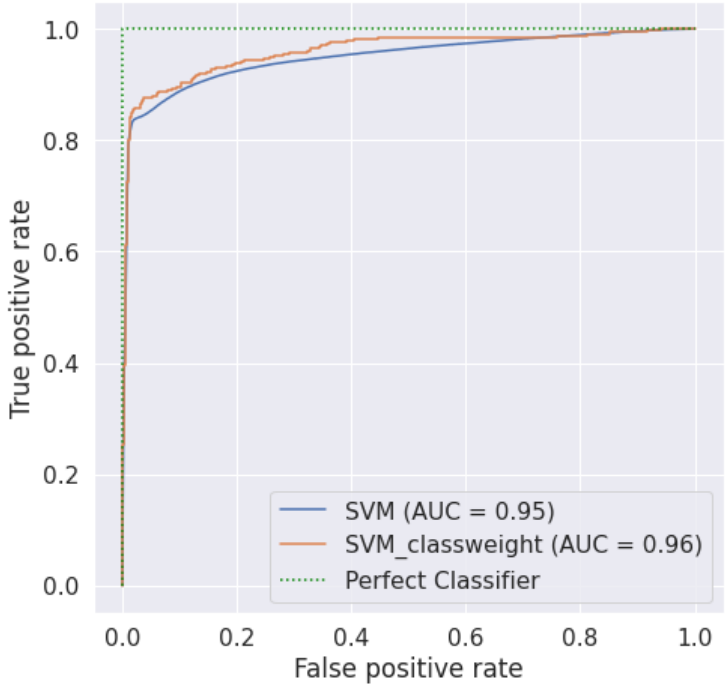
AUC curve:



برای یافتن فیچر های موثر نیاز بود که کرنل svm از حالت دیفالت به حالت linear تبدیل گردد. پس از اعمال کرنل linear مدل را برروی داده ها دوباره فیت کردیم و باز هم مشاهده مینماییم که دو فیچر 0 و 3 بیشترین تاثیر را بر مدل داشتند.

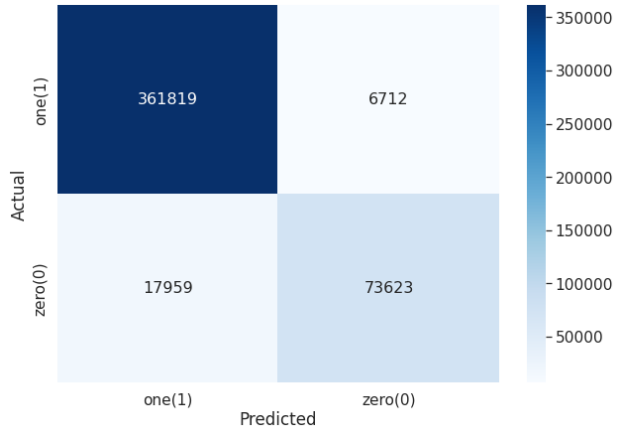


در انتها نیز مدل svm را با استفاده از class_weight پیاده کردیم و دقت مدل به 95.6 افزایش یافت



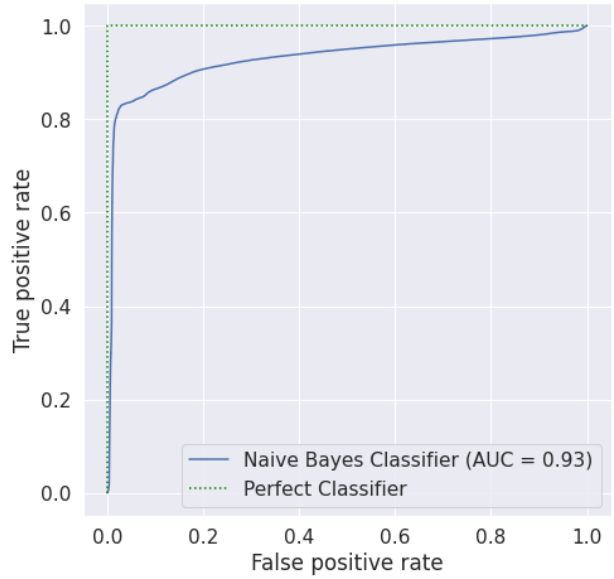
مدل Naive Bayes :

با توجه به generative بودن این مدل و اینکه این مدل توزیع داده ها را یاد میگیرد، برای آن همیتی ندارد که کلاس ما balance باشد یا خیر. پس این مدل را برروی داده های اصلی بدون سمپل گرفتن پیاده کردیم. نتیجه آنکه دقت آن حدود 94.6% شد



0	0.95	0.98	0.97	368531
1	0.92	0.80	0.86	91582
accuracy			0.95	460113
macro avg	0.93	0.89	0.91	460113
weighted avg	0.95	0.95	0.95	460113

AUC curve:



نتیجه گیری نهایی

- بدون در نظر گرفتن مدل SVM دقت تمامی مدل ها برروی داده های تست به نسبت خوب است. بیشترین دقت را مدل Random Forest و Logistic Regression دارند. کمترین دقت مربوط به مدل Decision Tree است.

- مدل SVM برروی داده ها با حجم زیاد بسیار طول میکشد و نمیتوان نتیجه آن را دید اما برای بخشی از داده ها دقت خوبی حدود 95% داشت.

- استفاده از class_Weight دقت برخی مدل ها را افزایش میدهد. برای مدل های SVM، Logistic Regression و Decision Tree باعث افزایش دقت شد اما برای مدل Random Forest نتیجه عکس داشت.

- همانطور که دیدم اکثرا دو فیچر App و Channel بیشترین تاثیر را در classifier های ما داشتند.

- مدل Naive Bayes برروی داده ها نتیجه خوبی داشت، با اینکه سمپل گرفته نشد. این مدل یک مدل generative است و دقت به نسبت خوبی نیز دارد.

- مدل KNN در predict کردن داده ها بسیار کند عمل میکند و حتی نتایج خیلی خوبی نیز ندارد و از آنجایی که به پارامتر های مدل وابسته است، باید با تعویض آنها بهترین نتیجه را برای آن پیدا کرد.

- سمپل گرفتن از داده ها برای کاهش آنها و balance کردن کلاس ها بسیار مناسب است و باعث میشود تا علاوه بر آنکه حجم داده ها کم میشود، دقت مدل نیز با گرفتن سمپل درست افزایش پیدا کند.