

تحلیل داده‌های حجمی

پروژه نهایی

نام و نام خانوادگی: ناصر کاظمی

شماره دانشجویی: ۹۹۱۰۲۰۵۹

• مقدمه

در این پروژه به بررسی و تحلیل داده‌های ترافیکی موجود در دیتابیس TrafficData می‌پردازیم. من نوتبوک مربوط به هر بخش از ایده‌های مطرح شده در داک پروژه را جدا قرار داده‌ام. جزئیات پیاده‌سازی هر کدام از روش‌ها به صورت کامنت و متن مارکداون در هر نوتبوک آمده است. در این گزارش به توضیح کلیات هر بخش و تحلیل نتایج و روش‌های به کار رفته می‌پردازم.

در انجام این پروژه برای ایده‌ها و روش‌های به کار رفته با آقای علی فلاحتی با شماره دانشجویی ۹۹۱۰۴۱۲۴ مشورت کرده‌ام.

• پیش‌پردازش داده‌ها

یکی از نخستین کارهایی که می‌توانیم انجام دهیم، دور انداختن داده‌های پرت است. برای مثال ماشین‌هایی که به طور غیرعادی از آن‌ها رکورد ثبت شده‌اند را دور می‌اندازیم. چون برای مثال می‌توانند ماشین‌هایی باشند که در یک محل پارک شده‌اند.

• دوربین‌های مجاور

برای این قسمت از پروژه باید تعداد خودروهایی که دوربین‌های مجاور ثبت میکنند را بدست بیاوریم. ابتدا باید دوربین‌های مجاور را پیدا کنیم. طبق تعریف داده شده برای دوربین‌های مجاور، دوربین‌هایی که عملای از نظر فیزیکی به هم نزدیک‌اند مجاور محسوب می‌شوند. و از آنجایی که ما می‌خواهیم مجاورت را از این منظر بررسی کنیم، بهتر است دوربین‌هایی از یک نوع را و به طور خاص دوربین‌هایی که فقط عبور خودروها را ثبت می‌کنند را در نظر بگیریم. یعنی دوربین‌های 281, 81 و 283.

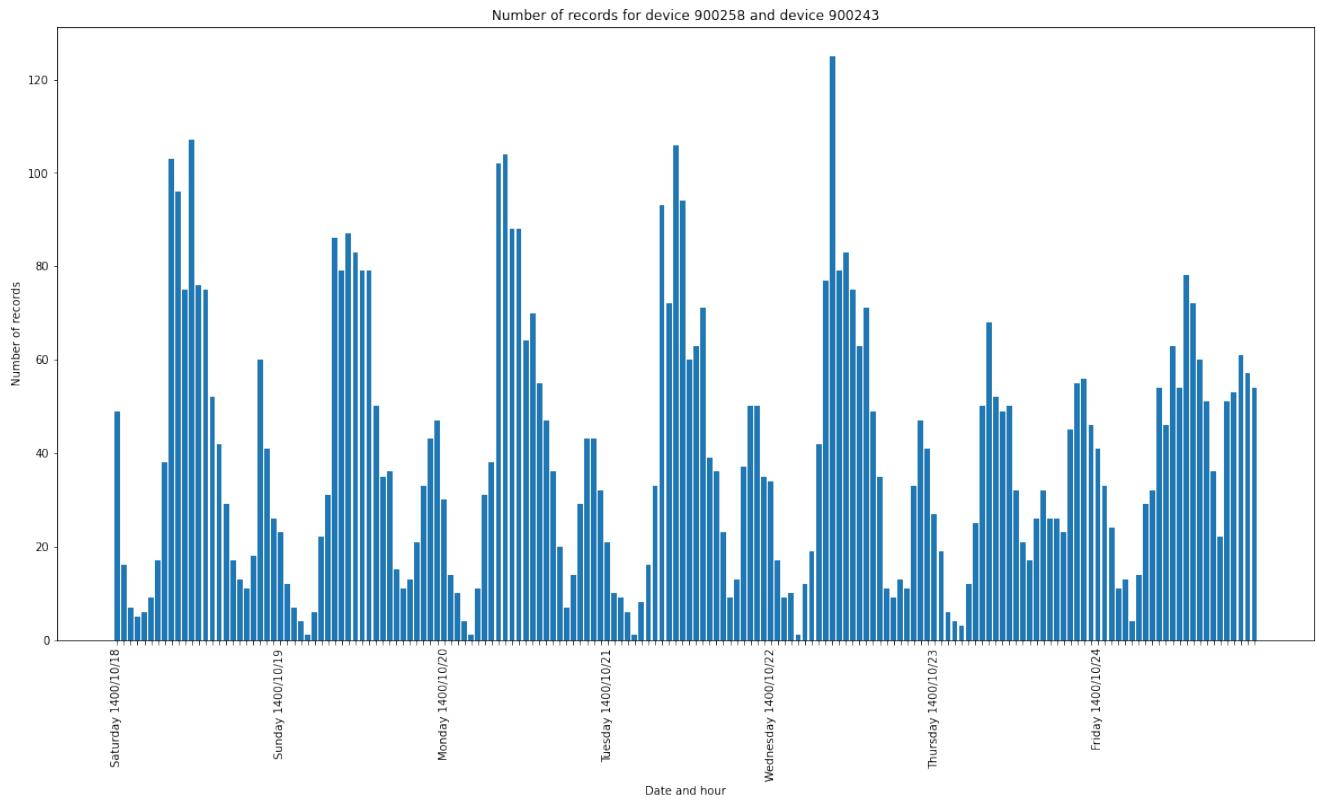
یک معیار خوب برای بررسی مجاورت دو دوربین، می‌تواند فاصله زمانی ثبت یک ماشین توسط آن‌ها باشد. به طور کلی می‌توان گفت هر چه این زمان کمتر باشد، احتمال مجاور بودنشان زیاد است. زیرا در این فاصله دوربین دیگری ان ماشین را ثبت نکرده است یا به عبارتی بین این دو دوربین، دوربین دیگری قرار نگرفته است.

پس ابتدا برای هر ماشین دوربین‌هایی که آن را ثبت کرده‌است را بدست می‌آوریم و بر اساس زمان ثبت ماشین مرتب‌شان می‌کنیم. حذف کردن داده‌های پرت و ماشین‌هایی که در یک مکان پارک بوده‌اند اینجا معنی‌دار می‌شود. حال یک بازه آستانه برای فاصله زمانی بین ثبت دو دوربین مجاور درنظر می‌گیریم. برای مثال ۱۵ دقیقه. برای هر ماشین هر دو زوج دوربین متوالی که فاصله زمانی ثبت کردن آن ماشین در یک روز از ۱۵ دقیقه کمتر است را نگه‌می‌داریم. حال ممکن است در این بین داده‌های نادرست نیز داشته باشیم. بنابراین از بین این جفت‌ها تنها آن‌هایی را نگه‌می‌داریم که حداقل ۱۰۰۰ ماشین را در ثبت کرده‌اند. تعدادی از این جفت‌های مجاور به همراه تعداد رکوردهای مشترک‌شان عبارت‌اند از:

```
1 adjacent_device_pairs.take(10)

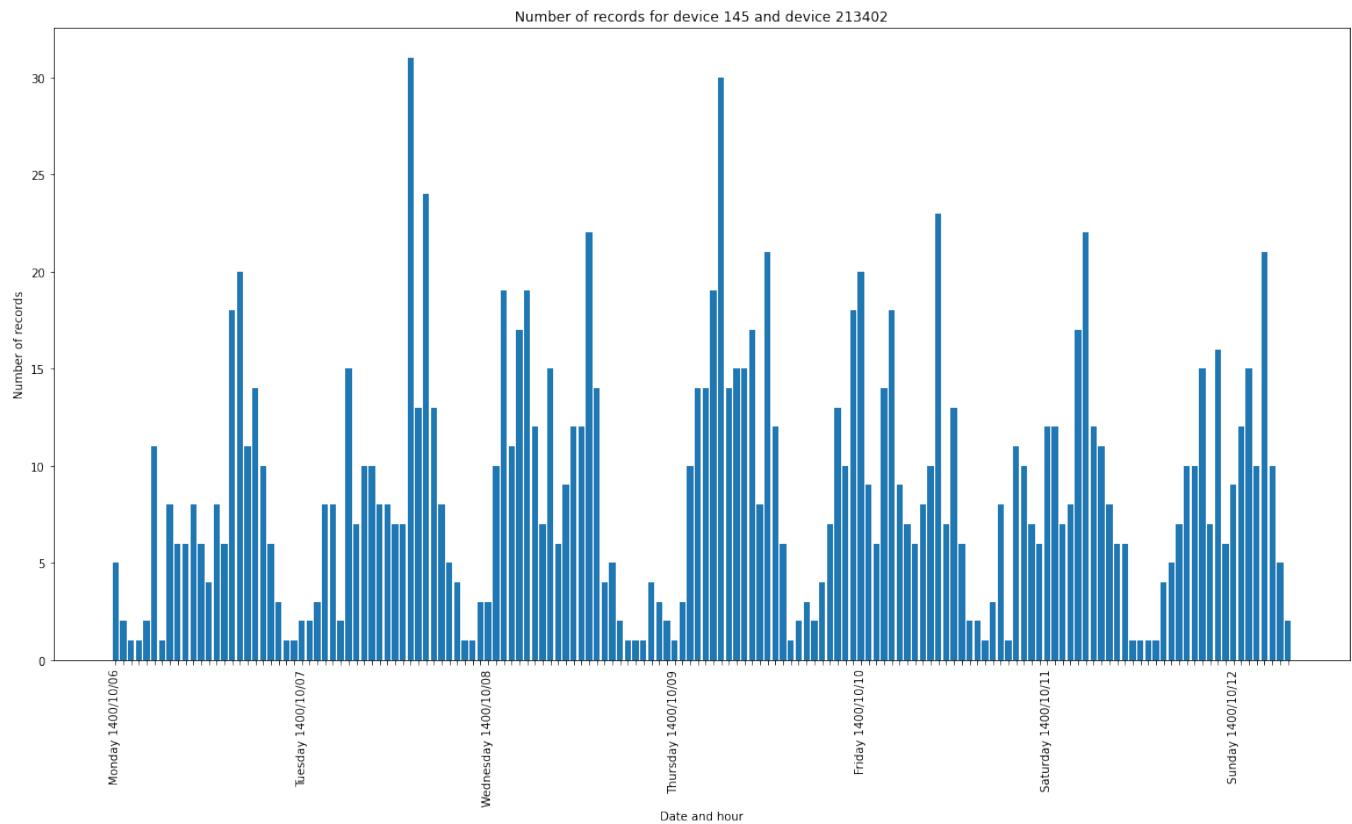
[((900221, 212501), 11970),
 ((900126, 900215), 1011),
 ((900247, 145), 1098),
 ((900164, 631369), 1389),
 ((900164, 631368), 24050),
 ((900236, 631357), 1364),
 ((900155, 631363), 1635),
 ((211301, 900191), 1248),
 ((213402, 230106), 2455),
 ((900212, 900269), 1906)]
```

بعد از اینکه جفت دوربین‌های مجاور را پیدا کردیم، یکی از آن‌ها را انتخاب می‌کنیم و به تحلیل داده‌های مسیر بین آن‌ها می‌پردازیم. هیستوگرام حاصل برای یک بازه یک هفته‌ای رو یه یک مجموعه داده را می‌بینید.



مشاهده می‌کنید که برای هر روز در ساعات اولیه روز تعداد رکوردها کم است. و رفته رفته زیاد می‌شود تا اینکه در میانی روز به حداقل می‌رسد که نشان‌دهنده شلوغ‌ترین ساعات آن مسیر است اگر این محاسبات را برای جفت‌های دیگر بررسی کنیم به نتایج مشابه می‌رسیم. همچنین مشاهده می‌شود در روزهای آخر هفته به طور کلی تعداد عبورهای ثبت شده در این مسیر کاهش می‌یابد. این موضوع می‌تواند به این معنی باشد که مسیر بین این دو دوربین مسیری است که اکثر افراد برای رسیدن به محل کار از آن عبور می‌کنند. ممکن است برای یک جفت به نتایج برعکسی بررسیم. این موضوع می‌تواند نشان‌دهنده مسیر مکان‌های تفریحی باشد.

حال کل داده‌ها را که برای یک ماه است را بررسی می‌کنیم. ابتدا رکوردها را به ۴ هفته تقسیم می‌کنیم و محاسبات قبلی را برای هر بخش انجام می‌دهیم. هیستوگرام مربوط به یکی دیگر از این هفته‌ها را در زیر مشاهده می‌کنید.



همانطور که می‌بینید الگوی قبلی تا حد خوبی در اینجا مشاهده می‌شود. از این موضوع می‌توان نتیجه گرفت رفتار افراد در حمل و نقل و عبور از مسیرها به طور کلی به صورت هفتگی تکرار می‌شوند (البته این تکرار بیشتر به صورت روزانه است). همانطور که در نمودارها می‌بینید هیستوگرام روزهای کاری هفته بسیار به هم نزدیک هستند).

Frequent Itemsets •

در این بخش برای پیدا کردن مسیرهای پرتردد از الگوریتم Apriori استفاده می‌کنیم. مراحل این الگوریتم به این صورت است:

مرحله ۱: مجموعه آیتم‌های frequent (لیست دوربین‌ها یا معادلاً یک مسیر) کاندید را ایجاد می‌کنیم.
در این مرحله مجموعه آیتم‌های کاندید را تولید می‌کنیم. ما با اولین مجموعه اقلام کاندید شروع می‌کنیم، که مجموعه‌ای از تمام item‌های منحصر به فرد در transaction است.

برای تولید مجموعه آیتم‌های کاندید بعدی، روی مجموعه آیتم‌های کاندید قبلی iterate می‌کنیم و با پیوستن به مجموعه آیتم‌ها، مجموعه آیتم‌های نامزد بعدی را ایجاد می‌کنیم. سپس روی تراکنش‌ها iterate می‌کنیم و بررسی می‌کنیم که آیا مجموعه آیتم‌های کاندید زیر مجموعه‌ای از تراکنش (منظور مسیری است که یک خودرو در یک روز طی کرده) است یا خیر. اگر چنین باشد، تعداد مجموعه آیتم‌های نامزد را افزایش می‌دهیم. سپس روی مجموعه آیتم‌های کاندید این کار را تکرار می‌کنیم و اگر تعداد آن‌ها از support threshold بیشتر یا مساوی باشد، مجموعه آیتم‌ها را به مجموعه آیتم‌های frequent اضافه می‌کنیم.

مرحله ۲: مجموعه آیتم‌های frequent را ایجاد می‌کنیم.
در این مرحله مجموعه آیتم‌های مکرر را تولید می‌کنیم. ما روی مجموعه آیتم‌های نامزد iterate می‌کنیم و اگر تعداد آن‌ها از support threshold بیشتر یا مساوی باشد، مجموعه آیتم‌ها را به مجموعه آیتم‌های frequent اضافه می‌کنیم. سپس مجموعه آیتم‌های کاندید بعدی را با join کردن مجموعه آیتم‌های frequent ایجاد می‌کنیم. این روند را تا زمانی تکرار می‌کنیم که مجموعه آیتم‌های کاندید دیگری نداشته باشیم.

A-Priori الگوریتم

برای تولید همه مجموعه‌های frequent، الگوریتم A-Priori را روی تراکنش‌ها اجرا می‌کنیم. در هر iteration، مجموعه آیتم‌های کاندید و مجموعه آیتم‌های مکرر را تولید می‌کنیم. سپس مجموعه آیتم‌های frequent را به لیست همه مجموعه‌های اقلام اضافه می‌کنیم. سپس مجموعه آیتم‌های کاندید بعدی را با پیوستن به مجموعه آیتم‌های frequent ایجاد می‌کنیم. این روند را تا زمانی تکرار می‌کنیم که دیگر مجموعه آیتم‌های کاندیدی نداشته باشیم یا به حداقل تعداد آیتم‌ها در مجموعه آیتم‌ها برسیم.

ابتدا تنها خودروهایی را نگه‌دارم که در یک روز از بیشتر از ۲۰ دوربین رد نشده باشد. زیرا اگر لیست دوربین‌های یک ماشین زیاد باشد، به یک ساپورت قوی برای تعداد زیادی مسیر تبدیل می‌شود که ممکن است در اصل مسیر پرتردد نباشد.

یکی از مشکلاتی که در ابتدا به آن برخوردم این بود که برای بعضی نمونه‌های دیتابیس الگوریتم لیست خالی بر می‌گرداند. همچنین برای داده کامل یک بازه یک هفته‌ای، تعداد مسیرهای پرتردد کمتر از آنچه که منطقی به نظر می‌رسید بود. با کمی آزمون و خطا متوجه شدم که این اتفاق به خاطر این می‌افتد که

مجموعه‌هایی که از دوربین‌های مناظر با یک ماشین و یک روز می‌سازیم، وقتی برای حذف موارد تکراری به set تبدیل می‌شوند، ترتیب دوربین‌هایشان به هم می‌خورد یا از همان اول ترتیب یکسانی ندارند. در نتیجه ترکیب‌هایی که از این‌ها بدست‌می‌آید حتی با وجود داشتن عناصر یکسان، خود لیست یا تاپلشان برابر نیست. برای رفع این مشکل هر کدام از این set‌ها را به یک لیست تبدیل و سپس مرتب کردم. برای پیدا کردن پرترددترین مسیرها، support threshold را برابر با ۱۰۰۰ قرار دادم و مسیرهای به طول ۲ تا ۶ دوربین را ابتدا برای یک بازه یک‌هفته‌ای محاسبه کردم. نتیجه این محاسبات به صورت زیر شد:

```

FrequentItems.ipynb ×
FrequentItems.ipynb > ⚡ !pip install pyspark
+ Code + Markdown | Run All Clear All Outputs Restart Variables Outline ...
myEnv (Python 3.8.13)

1 # print number of frequent paths
2 print(f'Number of frequent paths of length 2: {len(FI2)}')
3 print(f'Number of frequent paths of length 3: {len(FI3)}')
4 print(f'Number of frequent paths of length 4: {len(FI4)}')
5 print(f'Number of frequent paths of length 5: {len(FI5)}')
6 print(f'Number of frequent paths of length 6: {len(FI6)}')

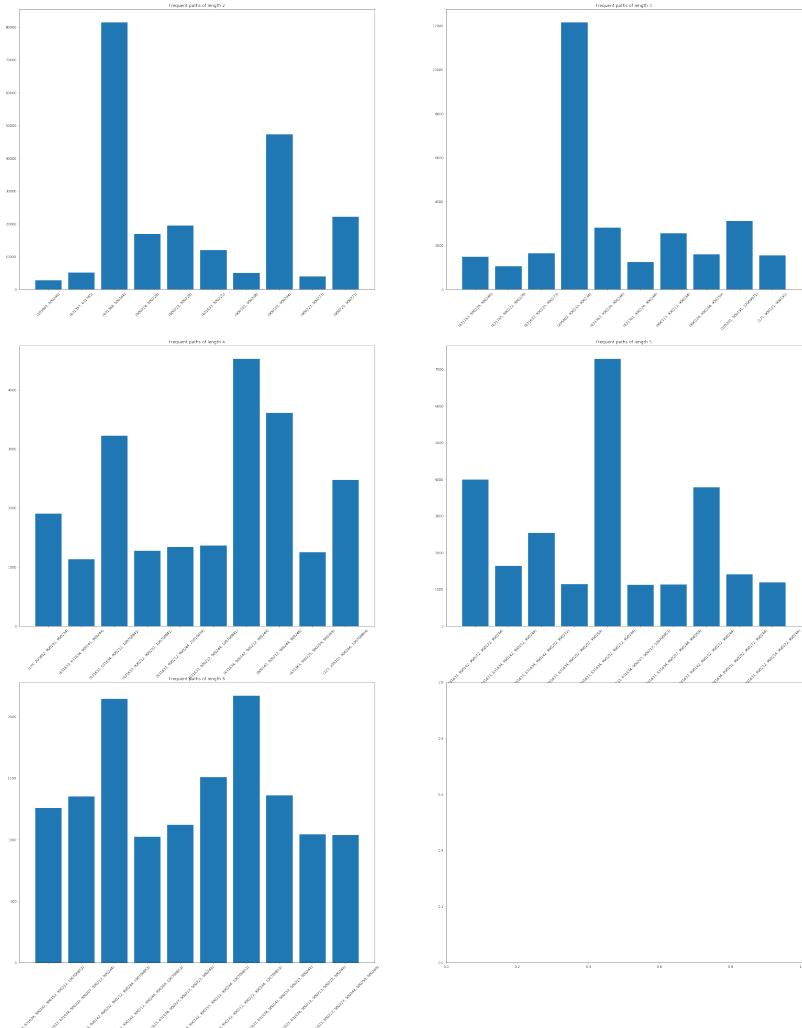
[20] Python
...
Number of frequent paths of length 2: 8786
Number of frequent paths of length 3: 9562
Number of frequent paths of length 4: 2882
Number of frequent paths of length 5: 451
Number of frequent paths of length 6: 38

1 # print the frequent paths
2 print("Frequent paths of length 2: ", [x for x in list(FI2)[10]])
3 print("Frequent paths of length 3: ", [x for x in list(FI3)[10]])
4 print("Frequent paths of length 4: ", [x for x in list(FI4)[10]])
5 print("Frequent paths of length 5: ", [x for x in list(FI5)[10]])
6 print("Frequent paths of length 6: ", [x for x in list(FI6)[10]])

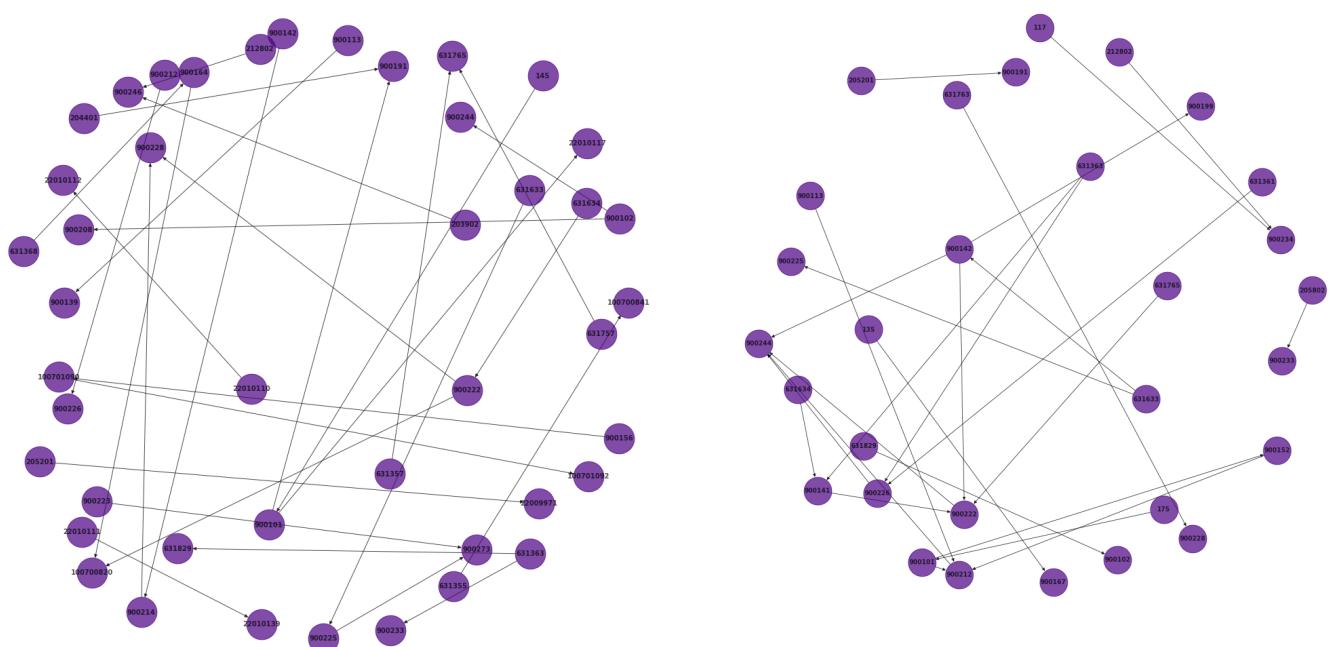
[30] Python
...
Frequent paths of length 2: [(203902, 900246), (631357, 631765), (631368, 900164), (900214, 900228), (900222, 900228), (631633, 900225), (900102, 900208), (900102, 900244), (900223, 900273)]
Frequent paths of length 3: [(631763, 900228, 900246), (631765, 900222, 900228), (631633, 900225, 900273), (205802, 900233, 900234), (631363, 900226, 900246), (631361, 900226, 900244), (900113, 900212, 900244), (900226, 900244, 900259), (205201, 900191, 22009971), (175, 900101, 900191)]
Frequent paths of length 4: [(175, 205802, 900191, 900234), (631633, 631634, 900141, 900244), (631634, 900212, 100700841), (631633, 900212, 900222, 100700841), (631633, 900212, 900244, 100700841), (631633, 900222, 22010039), (631633, 900222, 900244, 100700841), (631634, 900142, 900222, 900244), (900142, 900212, 900244, 900246), (631363, 900225, 900259, 900269), (123, 209103, 900266, 100700804)]
Frequent paths of length 5: [(631633, 900142, 900212, 900222, 900244), (631633, 631634, 900142, 900152, 900244), (631633, 631634, 900142, 900202, 900212), (631633, 631634, 900202, 900207, 900269), (631633, 631634, 900207, 900212, 100700853), (631633, 631634, 900207, 900207, 900244), (631633, 900142, 900152, 900212, 900244), (631634, 900101, 900202, 900212, 900244), (631633, 900212, 900214, 900222, 900244)]
Frequent paths of length 6: [(631633, 631634, 900142, 900152, 900212, 100700853), (631633, 631634, 900202, 900207, 900212, 900244), (631633, 631634, 900207, 900212, 900222, 900244), (631633, 631634, 900207, 900207, 900222, 900244), (631633, 631634, 900142, 900155, 900212, 900244, 100700853), (631633, 900142, 900152, 900212, 900222, 900244, 100700853), (631633, 631634, 900142, 900152, 900212, 900222, 900244), (631633, 631634, 900212, 900222, 900244), (631633, 900212, 900222, 900244, 900259, 900269)]

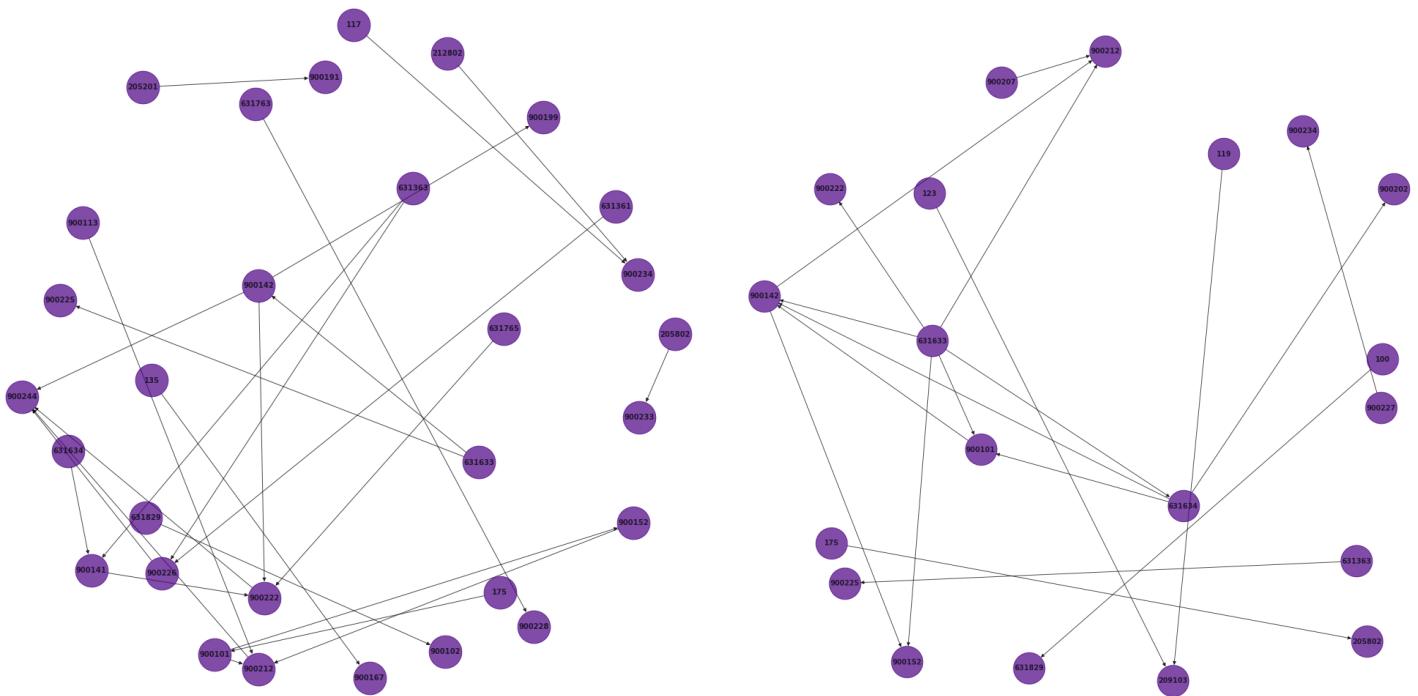
```

نمودار برخی از مسیرهای به طول های مختلف نیز به صورت زیر است:

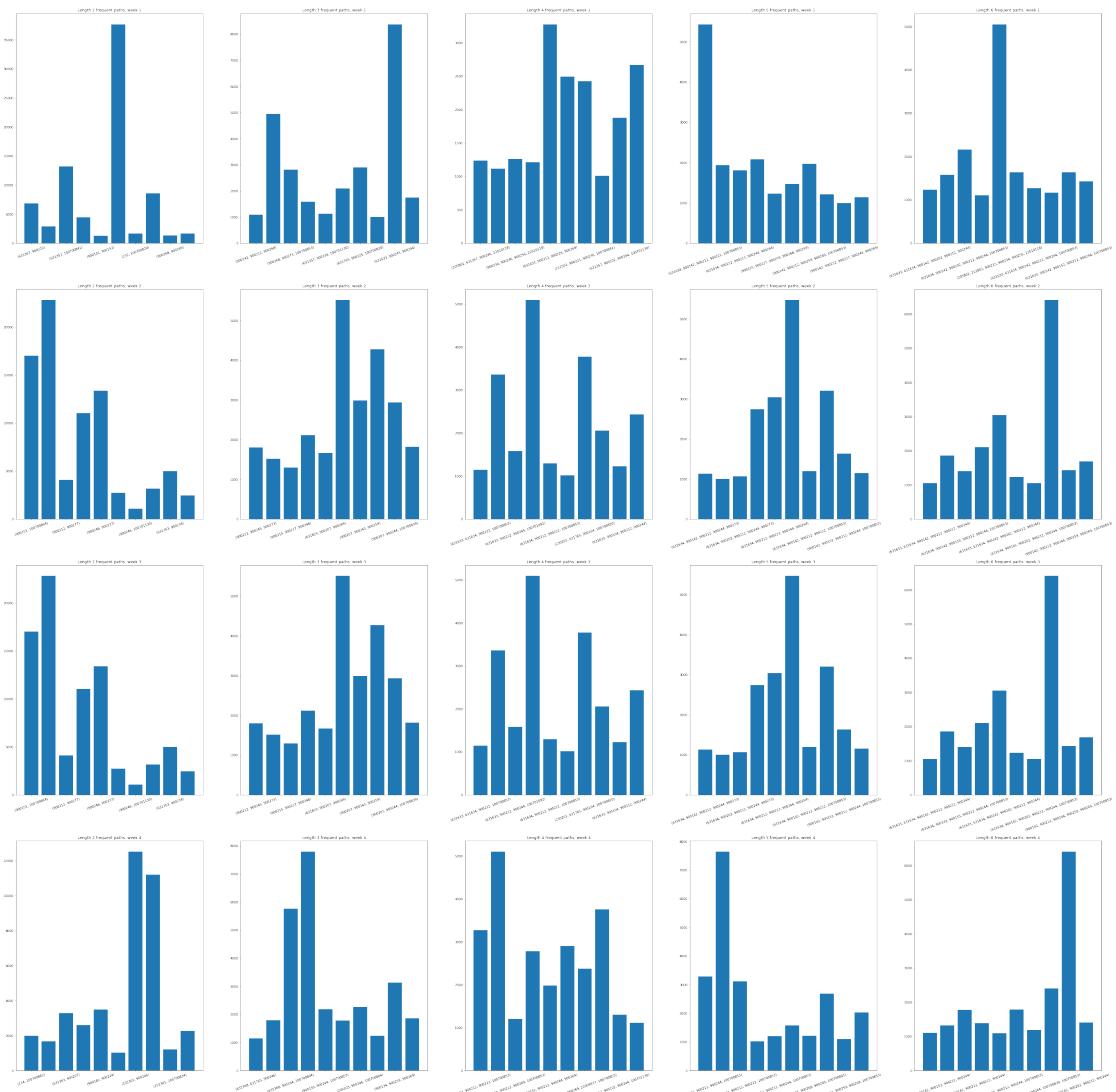


همچنین سعی کردم با یک گراف برخی از مسیرهای پرتردد را نمایش دهم:





سپس داده اصلی را به ۴ قسمت تقسیم کردیم. به طوری که داده‌های هر هفته در یک RDD باشد. سپس محاسبات را برای هر کدام تکرار کردیم. در نمودار زیر بخری از مسیرهای بدست آمده با طول‌های مختلف در هر یک از هفته‌ها را نشان داده‌ام.



می‌توان مشاهده کرد که مسیرهای پرتردد در بازه‌های یک هفته‌ای تقریباً یکسان‌اند. این موضوع منطقی است. زیرا همانطور که قبلاً هم ذکر شد، رفتارهای افراد برای تردد معمولاً به صورت هفتگی ثابت‌اند.

در این تحلیل یکی از مواردی که می‌توان به آن اشاره کرد، این است که دوربین‌هایی که از هم دور هستند در مسیرهای پرتردد با هم نمی‌آیند. حذف شدن دوربین‌های دور از هم به دلیل اصل monotonicity است. اگر دو دوربین C_1 و C_2 نزدیک هم نباشند، آنگاه اگر $\{C_1, C_2\}$ بخواهد frequent باشد، خود C_1 و C_2 نیز باید frequent باشند. که احتمال چنین چیزی بسیار کم است. مخصوصاً اگر اندازه ترکیب subset frequent که در این صورت احتمال بودن frequent بودن آن‌ها که مکانشان کنار هم نیست ناچیز می‌شود و در نتیجه حذف می‌شود. در واقع اگر نتایج این بخش را با نتایج تحلیل Pixie مقایسه کنیم مشابه شbahet آن‌ها می‌شویم. دوربین‌هایی که در این بخش به عنوان مسیرهای پرتردد کنار هم قرار می‌گیرند، با احتمال بیشتری هم در نتایج بخش Pixie مشاهده می‌شوند.

رفتارهای تکرارشونده دیگری را نیز می‌توان بررسی کرد. برای مثال اگر بخواهیم زمان‌های پرتردد را پیدا کنیم کافی است جفت‌هایمان را به صورت <FINAL_CAR_KEY, PASS_DAY_TIME> بگیریم. چنین رفتار تکرارشونده‌ای معنی‌دار نیز هست. اگر بازه‌های مشخص از زمان را در نظر بگیریم، آنها بیشترین تعداد ماشین در آنها تردد داشته‌اند مطلوب ما هستند.
برای خودروهای پرتردد نیز کافی است جفت‌های <(FINAL_CAR_KEY, PASS_DAY_TIME)> را در نظر بگیریم. سپس groupByKey کنیم و سپس براساس DEVICE_CODE این کار را تکرار کنیم و طول لیست زمان‌ها را مقایسه کنیم.

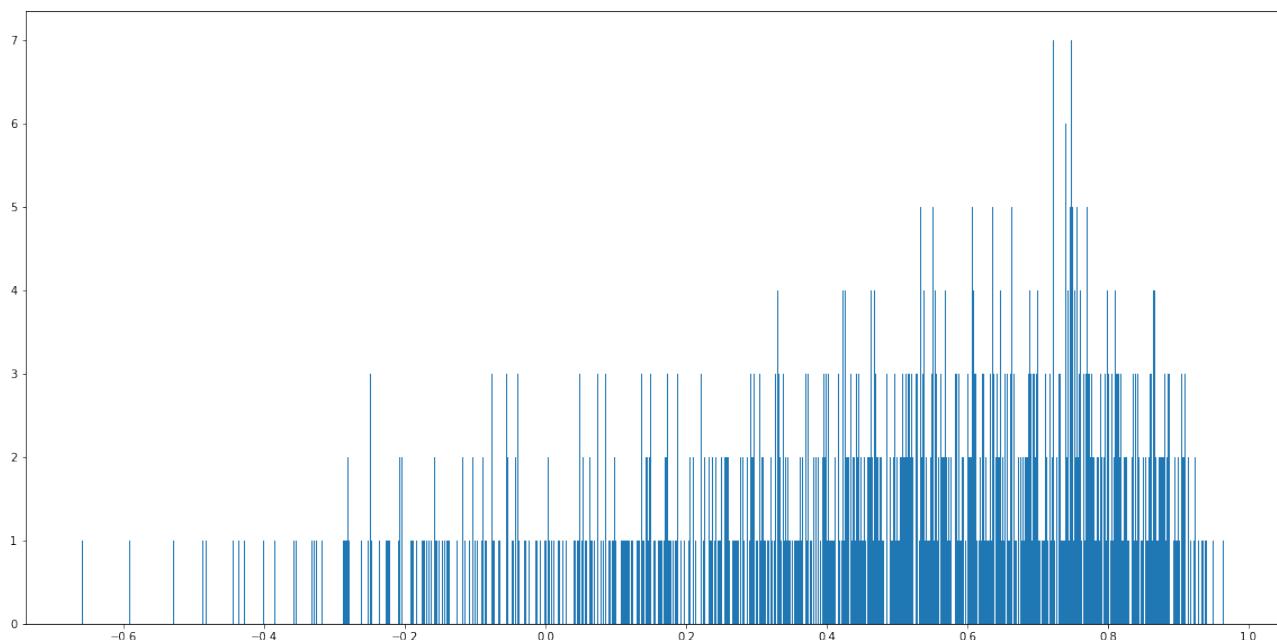
CF •

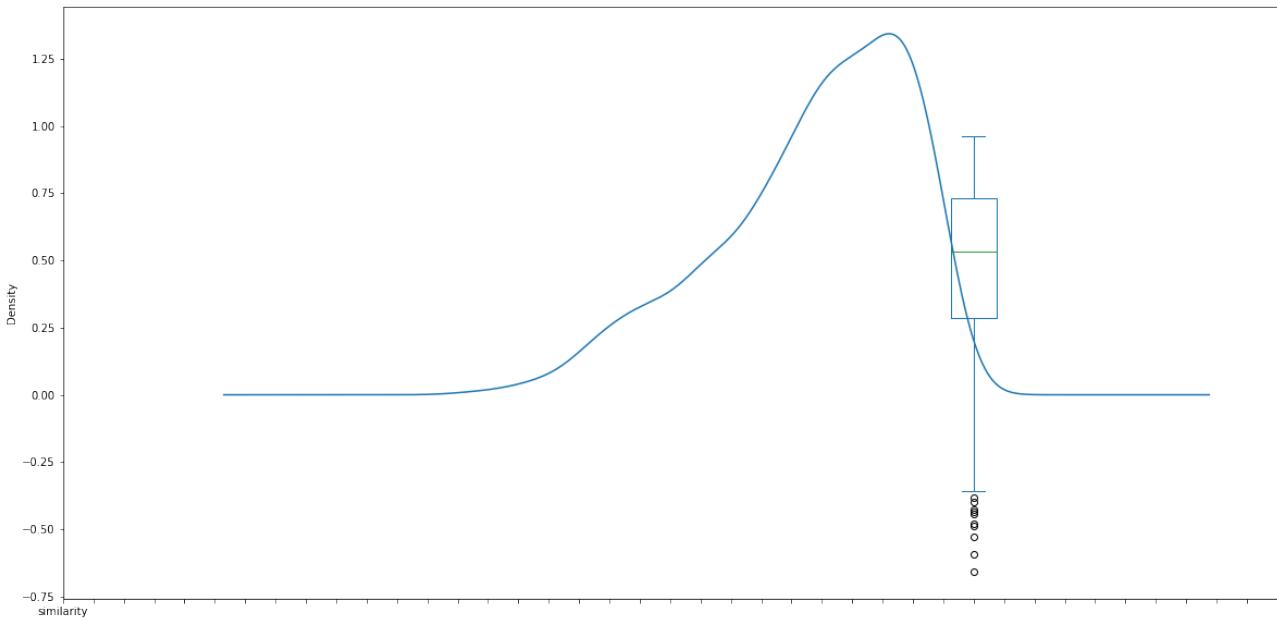
در این بخش یک سیستم توصیه‌گر با استفاده از روش Collaborative Filtering پیاده می‌کنیم. یک روش مبتنی بر تعامل کاربران با آیتم‌های سیستم است. به این صورت که برای یک کاربر براساس میزان تعامل سایر کاربران با یک آیتم، و همچنین ترجیحات خود کاربر آن آیتم را پیشنهاد می‌دهد.

برای اعمال CF از الگوریتم Alternating Least Square یا ALS استفاده می‌کنیم. این روش درواقع یک matrix factorization است. این روش مخصوصاً برای ماتریس‌های sparse بزرگ خوب جواب می‌دهد. در این روش ماتریس اصلی را به صورت ضرب دو ماتریس با ابعاد کمتر درمی‌آوریم که یکی ماتریس متناظر با دوربین‌هاست که سطرهای آن دوربین‌هاست و ستون‌های آن فاکتورهای پنهان‌هاست. ماتریس دیگر متناظر با ساعت‌های هفته است و ستون‌هایش ساعت‌های هفته و سطرهایش فاکتورهای پنهان هستند.

ابتدا یک مدل ALS تعریف می‌کنیم و آن را روی بخشی از دیتا آموزش می‌دهیم. با این کار مدل یاد می‌گیرد که در فضای فاکتورهای پنهان برای داده‌های ویژگی‌های معنادار پیدا کند و از این طریق شباهت دوربین‌ها را بدست آورد.

هدف این بخش پیدا کردن شباهت دوربین‌هاست. جزئیات پیاده‌سازی در نوتبوک این بخش ذکر شده‌است. بعد از اعمال ALS، ماتریس مربوط به دوربین‌ها را در نظر بگیرید. هر ردیف متناظر با یک دوربین است. برای یک ردیف خاص فاصله کسینوسی را با بقیه ردیف‌ها محاسبه می‌کنیم و نزدیکترین‌ها را گزارش می‌دهیم. دوربین 22010043 را نظر بگیرید. دوربین‌های مشابه این دوربین را با استفاده از روش گفته شده بدست می‌آوریم. نتیجه محاسبات به صورت زیر است:





تصاویر فوق نمودارهای توزیع مقادیر similarity سایر دوربین‌ها با دوربین 22010043 است.
همچمنی ۱۰ دوربین با بیشترین میزان تشابه عبارت‌اند از:

device_code	similarity
22010044	0.9670791726505334
900227	0.8183656037733871
22009973	0.8092162497154531
900256	0.7896699275976415
22009909	0.7886760237867068
635619	0.7747928235649074
1001004	0.7736895242287394
100701131	0.7671274807129105
22010112	0.7598339387431378
1001017	0.7490617187759447

روش‌های دیگر CF می‌تواند استفاده از الگوریتم KNN باشد. مسئله کاربران و فیلم‌های سینمایی را در نظر بگیرید. این الگوریتم به این صورت عمل می‌کند که با استفاده از متريک‌هایی مثل فاصله کسینوسی، نزدیکی دوربین‌ها را با هم می‌سنجد و با اعمال KNN برای پيشنهاد آيتم‌ها به کاربران از k نزدیک‌ترین همسایه‌ها استفاده می‌کند. یعنی هر بار از k نزدیک‌ترین همسایه‌ها را پیدا می‌کند و براساس مد انتخاب آن‌ها، پيشбинی را انجام می‌دهد. اين روش برای ماترييس‌های sparse خوب عمل نمی‌کند. زيرا ممکن است در k همسایه يك کاربر، اطلاعات مربوط به تعداد زیادي از فیلم‌ها موجود نباشد. اما در مسئله ما اين مشكل را نداريم و برای هر دوربين اطلاعات کافي موجود است. و از آنجايی که ما صرفا به دنبال

شباخت دوربین‌ها هستیم، می‌توانیم صرفا برای یک دوربین فواصل کسینوسی را با بقیه دوربین‌ها محاسبه و آتای اول را گزارش کنیم. جزئیات پیاده‌سازی این روش در نوتبوك این بخش آمده است. در تصویر زیر نیز می‌توانید نتیجه روش KNN را ببینید.

```
[((114, 210142), (1.0, 1)),  
 ((114, 22010114), (0.9906682527215792, 6)),  
 ((114, 900266), (0.9896882979632247, 168)),  
 ((114, 900235), (0.9894178605064041, 168)),  
 ((114, 100700871), (0.9879138827337552, 168)),  
 ((114, 100700812), (0.9870927560704167, 168)),  
 ((114, 100701096), (0.9868160968319144, 153)),  
 ((114, 100), (0.985013986845675, 168)),  
 ((114, 206602), (0.984678138499174, 168)),  
 ((114, 100700894), (0.984510516241513, 168))]
```

این دو روش تقریباً یک کار را انجام می‌دهند. اما در روش اول از فاکتورها در فضای latent استفاده می‌کنیم. اما در روش دوم نزدیکی را مستقیماً محاسبه می‌کنیم. به همین دلیل استفاده از روشی مثل ALS سریع‌تر و بهینه‌تر است.

حال به بررسی میزان دقت و عملکرد روش ALS در دو حالت گفته شده برای اندازه پنجره های زمانی می پردازیم. برای بررسی دقت از معیار Root-mean-square error استفاده می کنیم. در اینجا از کلاس RegressionEvaluator از مژول ml کتابخانه Spark استفاده می کنیم. برای حالتی که برای هر بازه یک ساعت رکوردها را بررسی کنیم، میزان نزدیکی پیش بینی ها به مقادیر واقعی و دقت این روش به صورت زیر است:

The screenshot shows a Jupyter Notebook interface with several code cells and their outputs.

Evaluation

We use the Root Mean Squared Error (RMSE) to evaluate the model. We can use the `pyspark.ml.evaluation.RegressionEvaluator` class to evaluate the model.

```
1 from pyspark.ml.evaluation import RegressionEvaluator  
2  
3 # evaluate the model by computing the RMSE on the test data  
4 predictions = model.transform(test)  
5 evaluator = RegressionEvaluator(metricName="rmse", labelCol="count",  
6 | | | | predictionCol="prediction")  
7  
8 rmse = evaluator.evaluate(predictions)
```

[86]

```
1 predictions.show(5)
```

[87]

```
+---+---+---+---+  
|DEVICE_CODE|HOUR_OF_WEEK|count|prediction|  
+---+---+---+---+  
| 100| 18| 1026| 960.2455|  
| 100| 24| 217| 206.23524|  
| 100| 28| 28| 33.774128|  
| 100| 59| 622| 532.8242|  
| 100| 80| 569| 560.1146|  
+---+---+---+---+  
only showing top 5 rows
```

```
1 print("Root-mean-square error = " + str(rmse))
```

[91]

```
... Root-mean-square error = 92.20055391998214
```

Python 3.7.12

حال برای بازه‌های زمانی ۶ ساعته این روش را تکرار می‌کنیم. در این صورت دقت و میزان خطای این روش برابر است با:

```
1 predictions.show(5)

+-----+-----+-----+
|DEVICE_CODE|HOUR_OF_WEEK|count|prediction|
+-----+-----+-----+
|     100|          4|  529| 557.89124|
|     100|         19| 4224| 4869.5107|
|     101|          3|  239| 245.19199|
|     101|         21|  115| 105.95039|
|     102|          9|  414| 378.54663|
+-----+-----+-----+
only showing top 5 rows

1 print("Root-mean-square error = " + str(rmse))

Root-mean-square error = 886.6702653079421
```

مشاهده می‌کنیم که در این حالت خطا افزایش می‌یابد که قابل پیش‌بینی بود. زیرا داده‌های ما در فضای با ابعاد کمتری هستند، پس دقت کم می‌شود.

در نهایت یکی از کارهایی که انجام دادم، بررسی نوع دوربین‌های گزارش شده به عنوان دوربین‌های مشابه بود. در ۱۰۰ مورد با بیشترین شباهت نوع این دوربین‌ها تقریباً همه به یک دسته تعلق داشت، که با نتایج بخش تحلیل Pixie همخوانی دارد.

```
1 top_100

   device_code  similarity  DEVICE_CODE  SYSTEM_ID
0    100701096  0.964479  100701096      283
1     631781  0.949487    631781       81
2   100700965  0.939276  100700965      283
3     900266  0.938007    900266      283
4    900199  0.934916    900199      283
...
95    710401  0.834061    710401       83
96   1001063  0.833629  1001063       83
97      102  0.829208      102       81
98      106  0.827144      106       81
99    900195  0.826369    900195      283

100 rows × 4 columns

1 top_100['SYSTEM_ID'].unique()

array([283,  81, 182,  83, 102], dtype=int32)
```

طبق خواسته سوال ابتدا یک گراف دو بخشی از ماشین‌ها و دوربین‌ها تشکیل می‌دهیم. به این شکل که در یک RDD کلیدهای (FINAL_CAR_KEY, DEVICE_CODE) و در یک RDD دیگر کلیدهای (DEVICE_CODE, FINAL_CAR_KEY) نگه‌میداریم. در این گراف بین هر ماشین و دوربینی که آن را ثبت کرده است به تعداد دفعات ثبت شدن یال ایجاد می‌شود. البته می‌توان براساس این کلیدها group by کرد و تنها یک یال در نظر گرفت. همچنین می‌توان به این شکل عمل کرد که براساس گره‌های هر بخش دسته‌بندی کرد و گراف را به صورت یک لیست مجاورت نگه داشت. به طوری که تکرار یال‌ها لحاظ شود.

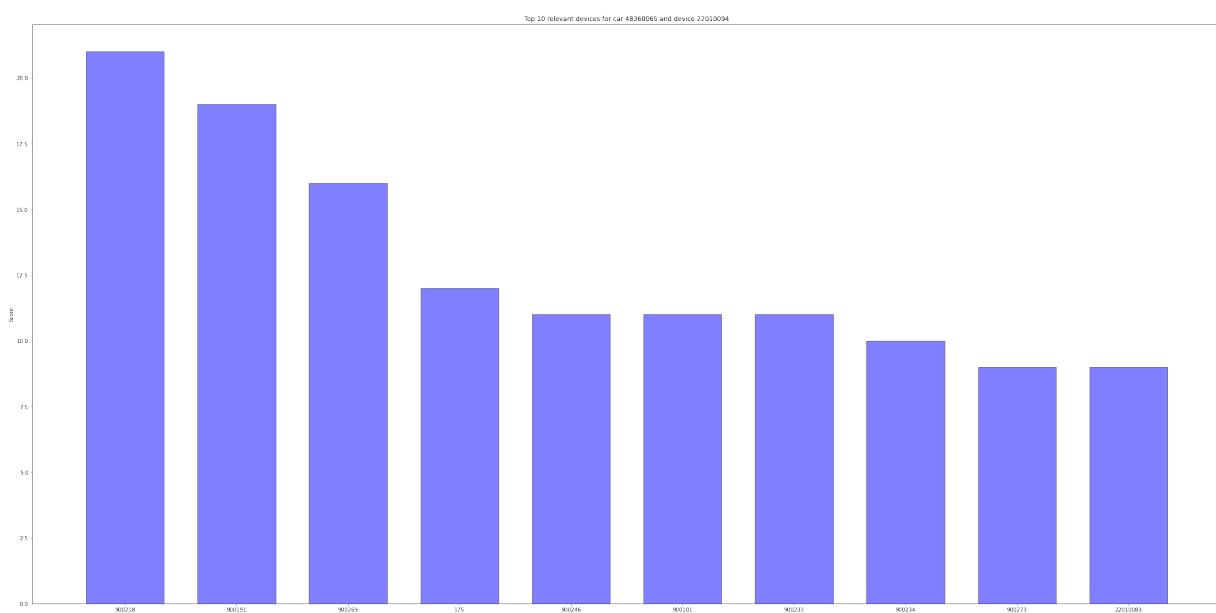
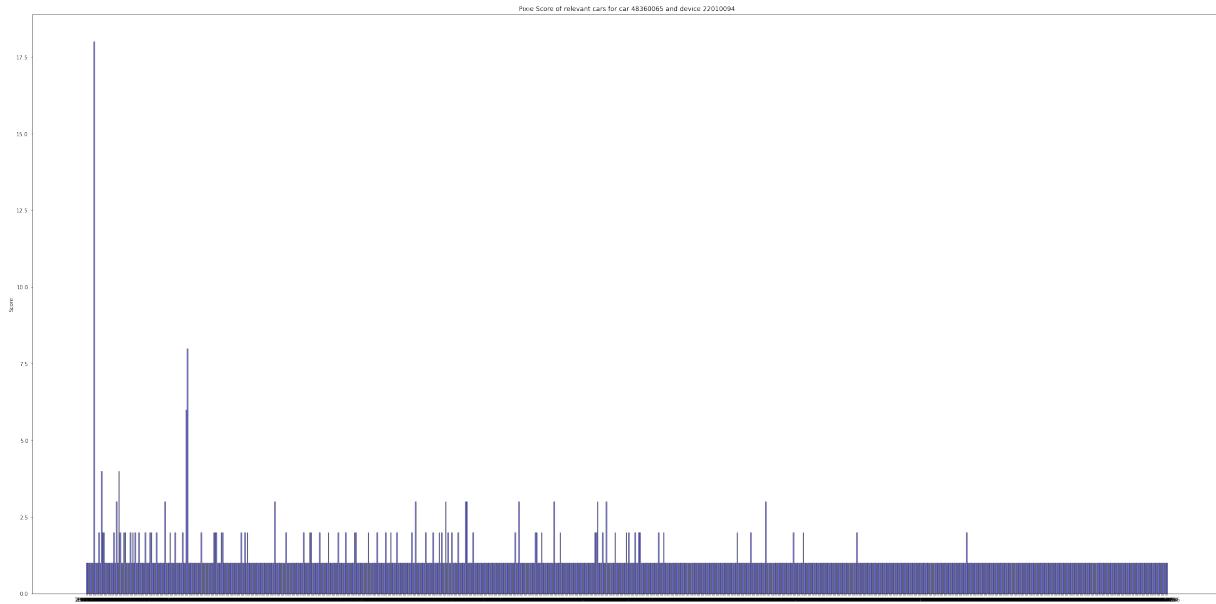
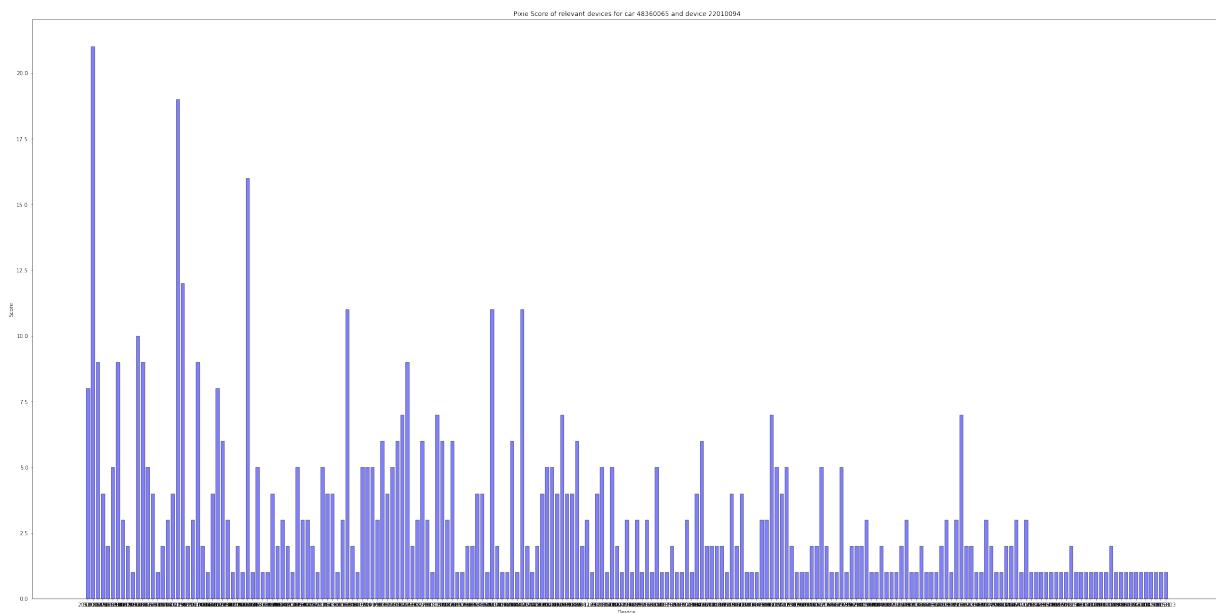
در توضیح این بخش گفته شده‌است که شباهت بین ماشین‌ها و دوربین‌ها در ساعت‌های مختلف روزهای مختلف بررسی کنیم. این بخش برای من گنگ بود. برداشت‌های متفاوتی از آن داشتم که توضیح میدهم.

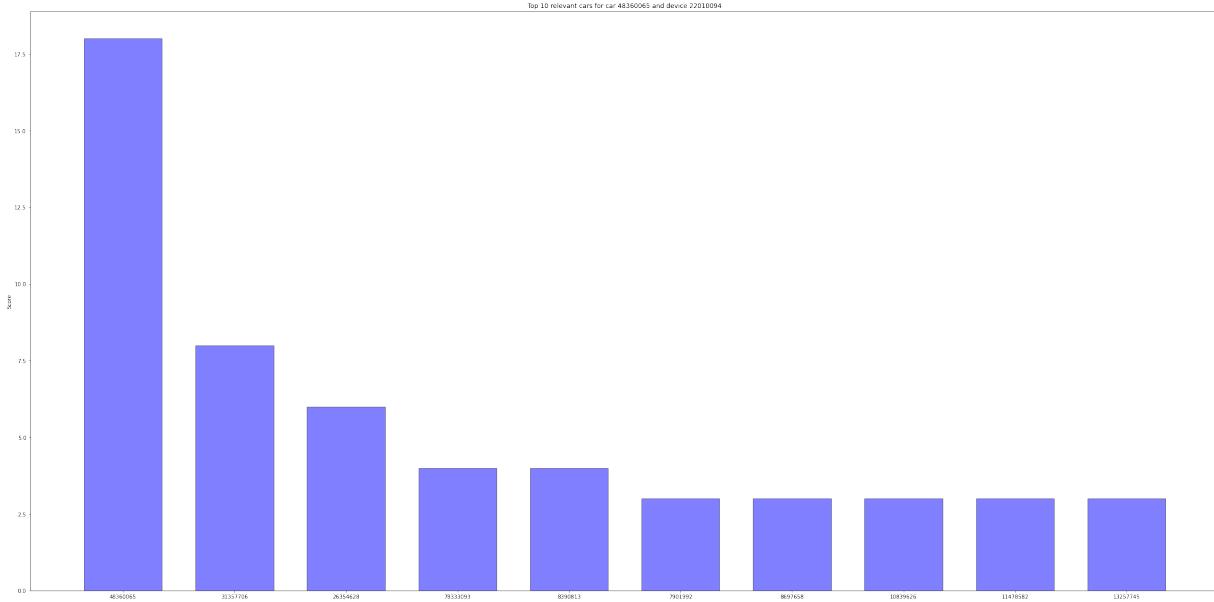
در الگوریتم Pixie برای Pinterest، یک گراف دوبخشی از Pin و Board دارم و یک User که یک Pin به یک Board وصل می‌کند. حال این User یک سری ویژگی دارد که براساس آن‌ها عمل "Biasing the Random Walk" انجام می‌شود. به این معنی که احتمال رفتن به یال‌هایی که به آن یوزر هستند را افزایش می‌دهد.

یک برداشت من این بود که در این مسئله ساعانت مختلف روز درواقع همان User‌ها هستند و وقتی یک کوئری شامل یک ساعت از هفته، یک ماشین و یک دوربین داریم، فقط یال‌هایی بررسی شوند که مربوط به آن ساعت از روز هستند. در این حالت کوئری می‌تواند یک ساعت از هفته باشد که در آن یک دوربین یک ماشین را ثبت کرده است. حال می‌توانیم در آن ساعت مسیرهای پرترد را پیدا کنیم. برداشت دیگر من این بود که صرفاً خواسته سوال این است که گراف خواسته‌شده را برای یک ساعت مشخص بسازیم و الگوریتم Pixie را اجرا کنیم. در نوبتیک این بخش هر دو پیاده سازی را آورده‌ام.

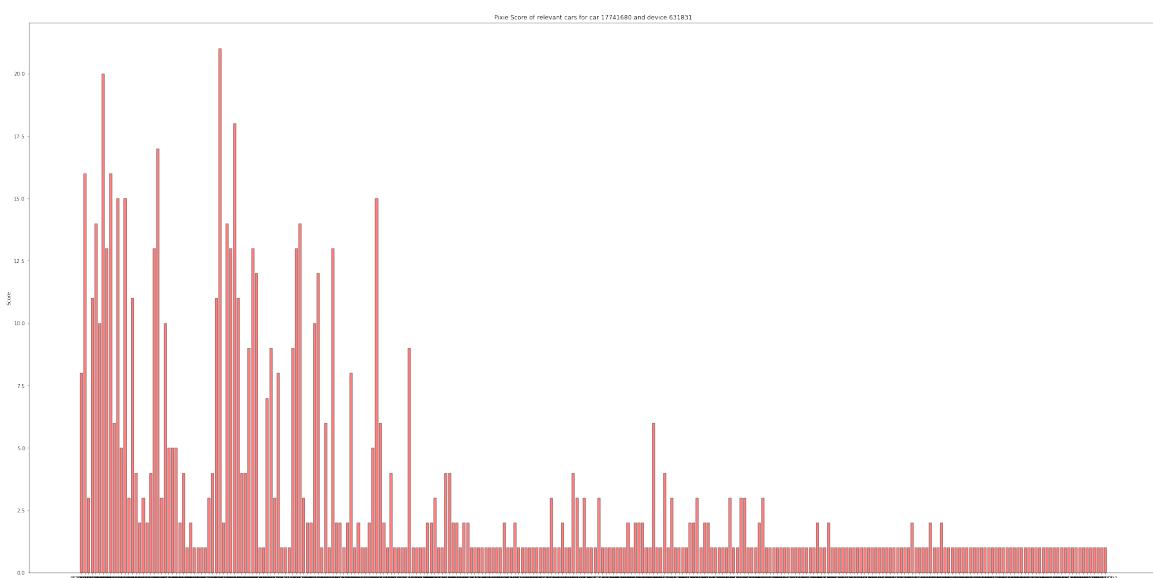
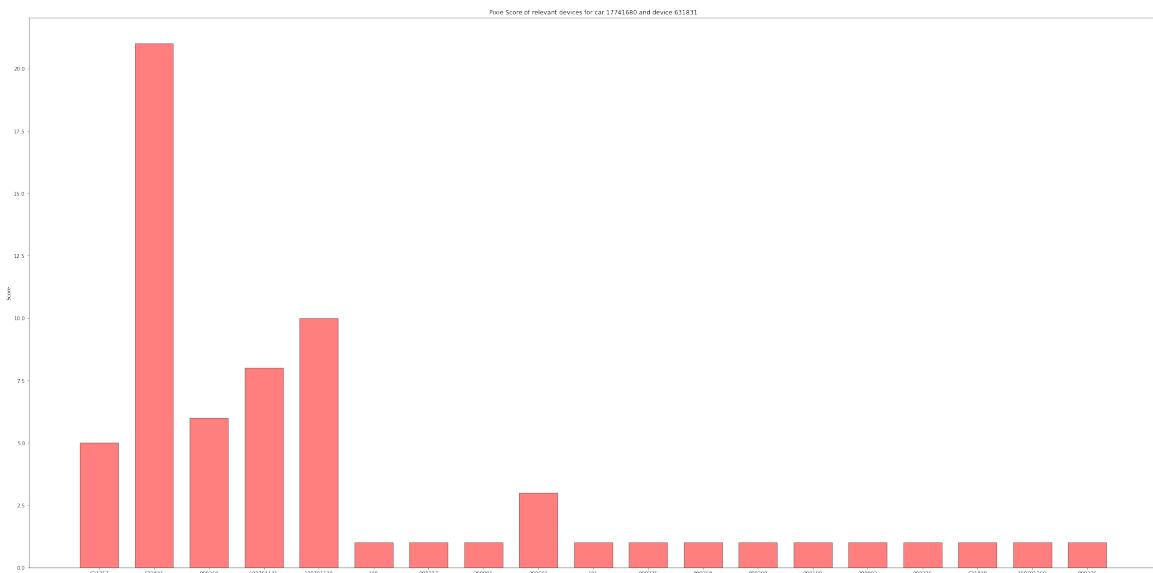
همچین Pin و Board را در این مسئله می‌توان هر کدام از ماشین یا دوربین گرفت. در کلیت مسئله تفاوتی ایجاد نمی‌کند. اما هر کدام می‌توانند بیانگر نتایج مختلفی باشند. ابتدا فرض می‌کنیم دوربین‌های Pin باشند. در این صورت می‌توانیم با استفاده از Pixie دوربین‌های نزدیک به هم را پیدا کنیم. در این حالت کوئری در واقع یک دوربین و یک ماشین ثبت شده است. حال اینکه مفهوم نزدیکی دوربین‌ها چیست قابل بحث است.

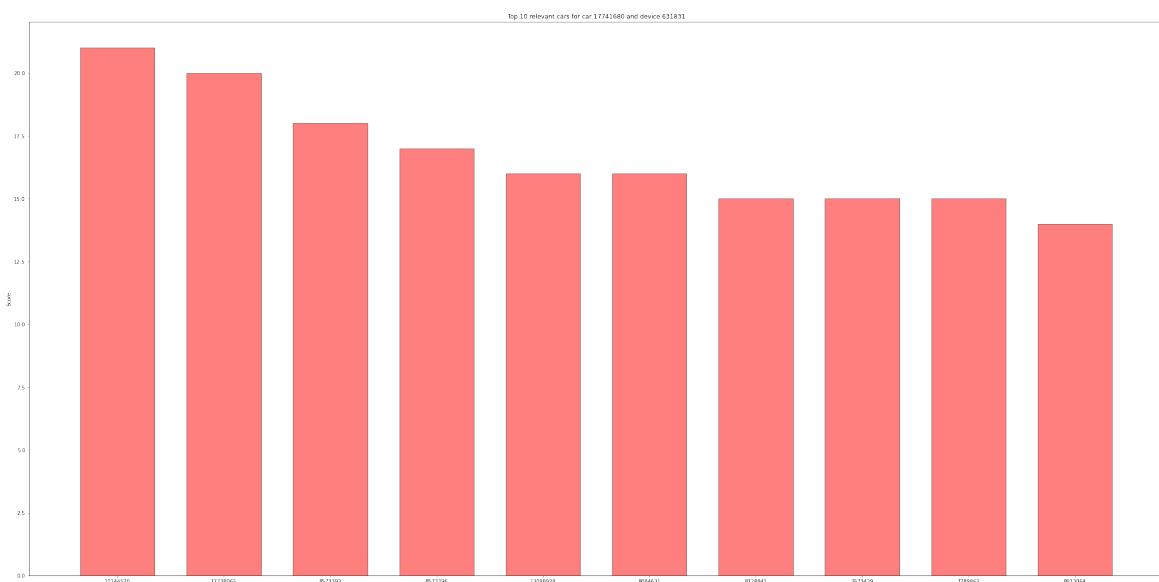
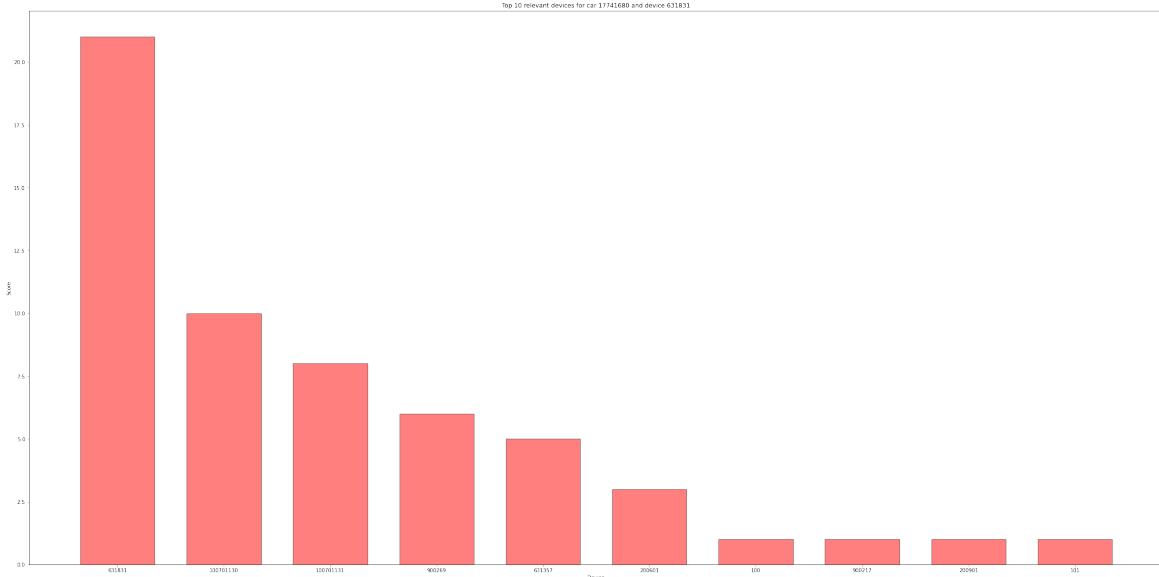
کوئری (73257028, 100700853) که 73257028 کلید ماشین و 100700853 شماره دوربین است را در نظر بگیرید. نتایج بدست آمده با اجرای pixie برای آن به صورت زیر است:





در نمودارهای فوق زمان را در نظر نگرفته‌ایم. حال زمان را در کانکست Pixie به عنوان User درنظر می‌گیریم و الگوریتم را اجرا می‌کنیم. نتایج این به بخش برای خودروهای مشابه با خودروی مورد نظر به صورت زیر خواهد بود:





نرديکي می تواند به معنای نرديکي فيزيکي باشد. شهود پشت اين نرديکي اين است که يك دوربین يك ماشين را ثبت می کند و در اجرای پيكسي رو گراف به دوربین ديگري می رويم که اين ماشين را ثبت کرده است و اين کار را تکرار می کنیم. بنابراین اگر دوربین ها را Pin درنظر بگيريم آن هايی به عنوان خروجي الگوريتم برای يك کوئري برگردانده می شوند که از نظر فيزيکي نرديکاند و مسیرهای مشابهی را ثبت کرده اند. يك کاربرد اين نتایج اين است که صرفا با داشتن نرديکي های دوربین ها و موقعیت هر کدام، می توان نقشه شهر را با تقریب خوبی بازسازی کرد.

يک معیار نرديکي ديگر می تواند نرديکي از نظر نوع دوربین ها و اطلاعاتی که ثبت می کنند باشد. در معیار قبلی تحلیل ما زمانی دقیق است که فرض کنیم دوربین ها همه از يك نوع اند یا فقط دوربین های يك نوع را درنظر بگيريم. اما با مشاهده نتایج بدست آمده می توان دید که برای مثلا برای دوربین 100700853 که از نوع دوربین ثبت عبور است، دوربین های مشابه آن همگی تقریبا از همان نوع هستند.

```

1 top_devices_info = df.rdd.filter(lambda x: x[0] in top_k_devices).collect()
2 # get type of device
3 top_devices_info = {x[1] for x in top_devices_info}
4 top_devices_info

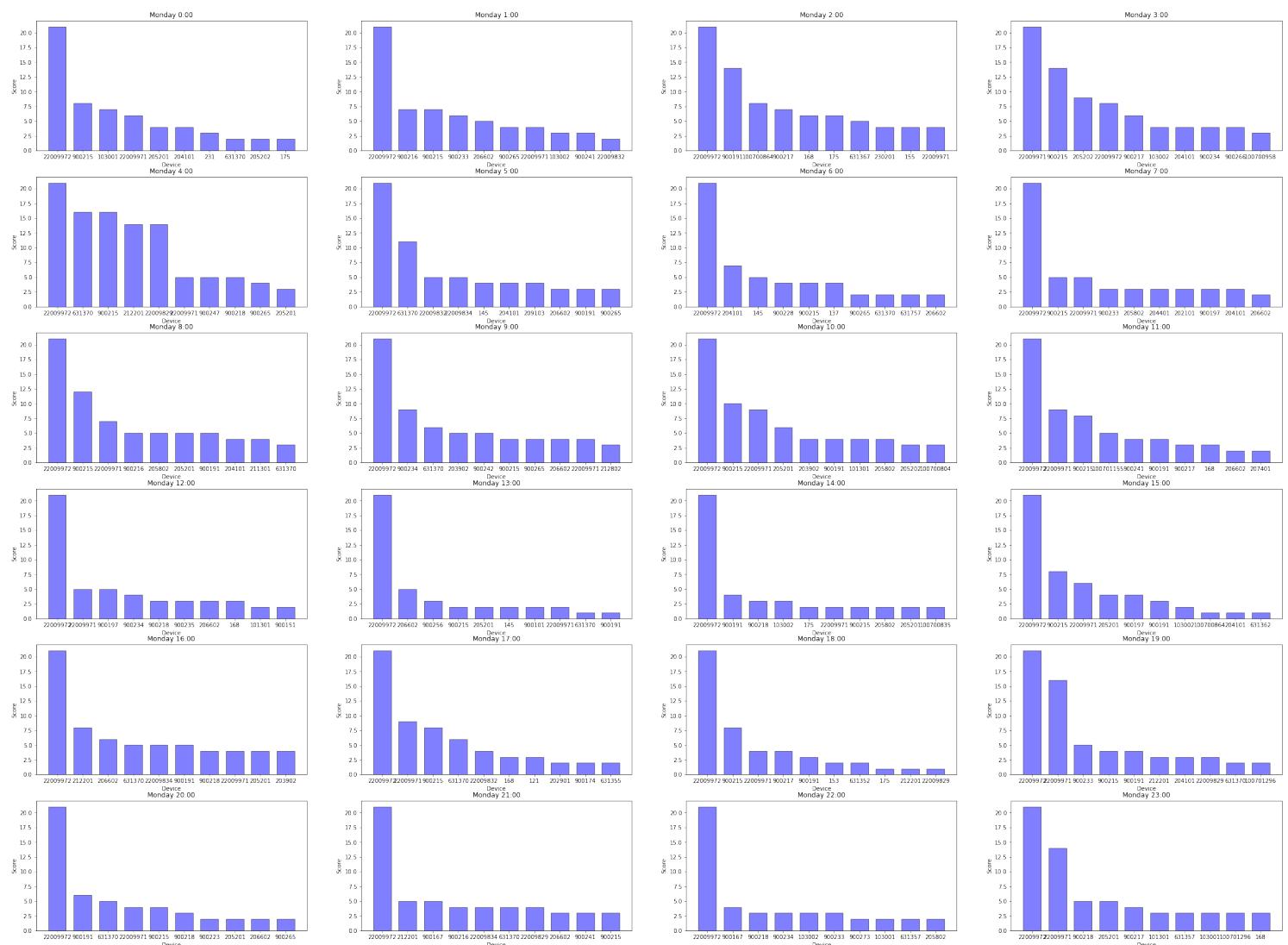
```

Python

{81, 282, 283}

در پیاده‌سازی این بخش برای traverse کردن گراف دوچشمی، از لیست مجاور استفاده کردیم. برای اینکه به صورت مسقتیم به هر لیست در RDD مربوط درسترسی داشته باشیم از تابع lookup استفاده کردیم. اما هر بار لیست خالی برمی‌گرداند. این مشکل را با استفاده از sortByKey روی RDD و سپس lookup حل کردیم. بعد از چند بار تکرار باز هم همین مشکل را داشتم و به همین شکل حل شد. این ممکن است صرفا یک باغ باشد زیرا دلیلی برایش پیدا نکردم. البته ممکن است برای دسترسی اسپارک از الگوریتمی مثل binary search استفاده کند که در این صورت منطقی است.

در آخر برای روزهای مختلف دیتابیس این نتایج را جداگانه محاسبه می‌کنیم. در نمودا زیر شبیه‌ترین دوربین‌ها به دوربین 22009972 در ساعات مختلف روز دوشنبه را می‌بینید.



مشاهده می‌کنید که میزان امتیاز Pixie برای دوربین‌ها و ۱۰ دوربین با بالاترین میزان شباهت در ساعات مختلف تقریباً به یک نسبت است. از نمودار فوق موارد مختلفی را می‌توان بیان کرد.

اولاً اینکه به طور منطقی دوربین 22009972 خودش بیشترین امتیاز را کسب کرده است و اختلافش با سایر دوربین‌ها معنی‌دار است.

همچنین دوربین‌های 22009971، 900215 در اکثر ساعات روز بیشترین شباهت را با دوربین 22009972 را داشته‌اند. یکی از دلایل این موضوع می‌تواند مجاورت این دوربین‌ها باشد. دلیل دیگر می‌تواند یک نوع بودن دوربین‌ها باشد. مخصوصاً اینکه شماره‌های 22009971 و 22009972 متوالی‌لند و می‌تواند این حدس را تقویت کرد.

HITS •

در این بخش با استفاده از الگوریتم HITS مهمترین authorityها را پیدا می‌کنیم. برای مثال می‌توانیم دوربین‌ها را Hub و ساعت‌های هفته را Authority در نظر بگیریم. در این صورت هر دوربین به یک ساعت از هفته لینک دارد. نتیجه‌ای که به دنبالش هستیم در واقع پر ترددترین ساعت هفته و مهمترین دوربین‌ها از نظر تعداد رکورد در ساعت مختلف هستیم.

برای اجرا HITS از روش تجزیه SVD استفاده می‌کنیم. ابتدا تعداد رکوردهای هر دوربین در هر ساعت را به صورت یک ماتریس محاسبه می‌کنیم. سپس با تجزیه SVD را انجام می‌دهیم. مهمترین hubها و entryها متناظر با بزرگترین مقادیر ماتریس‌های U و V هستند. جزئیات پیاده‌سازی و توضیحات هر بخش در notebook مربوط آمده است.

برای اینکه بررسی کنیم چرا تجزیه SVD جواب صحیح را به ما می‌دهد، روش Power iteration را نظر بگیرید. معادله‌ای که امتیازهای Hub و Authority در حالت همگرایی a ها تشکیل می‌دهند به صورت زیر است:

$$a = \alpha A^T h$$

$$h = \beta A a$$

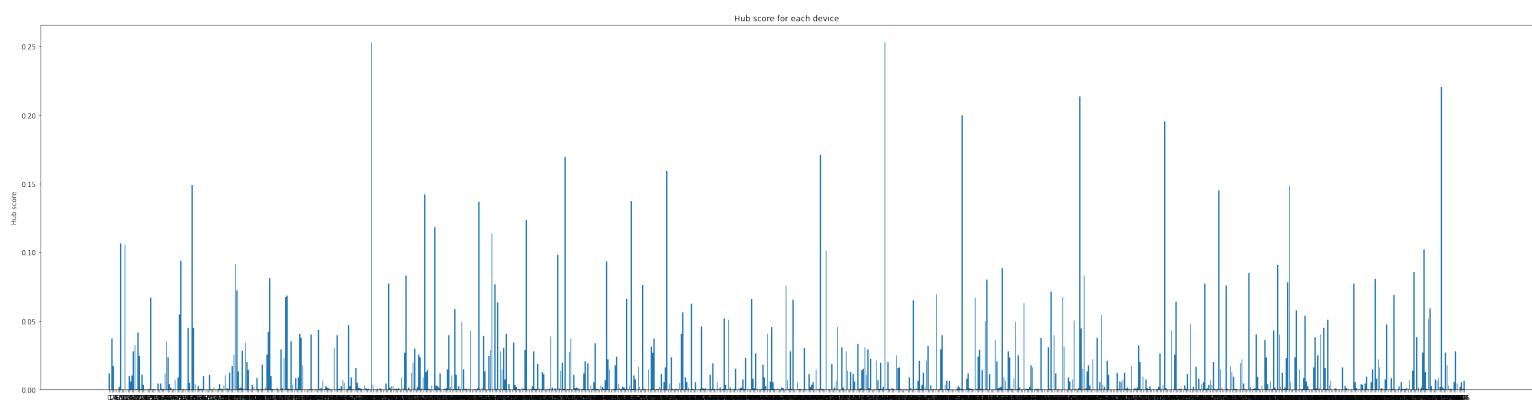
که A ماتریس مجاورت گراف مسئله، a بردار امتیازهای Hub و h بردار امتیازهای Authority است. α و β نیز مقادیر ثابتی هستند که به عنوان ضریب در هر مرحله iteration در امتیازها ضرب می‌شوند. در این صورت خواهیم داشت:

$$a = \alpha \beta A^T A a$$

$$h = \alpha \beta A A^T h$$

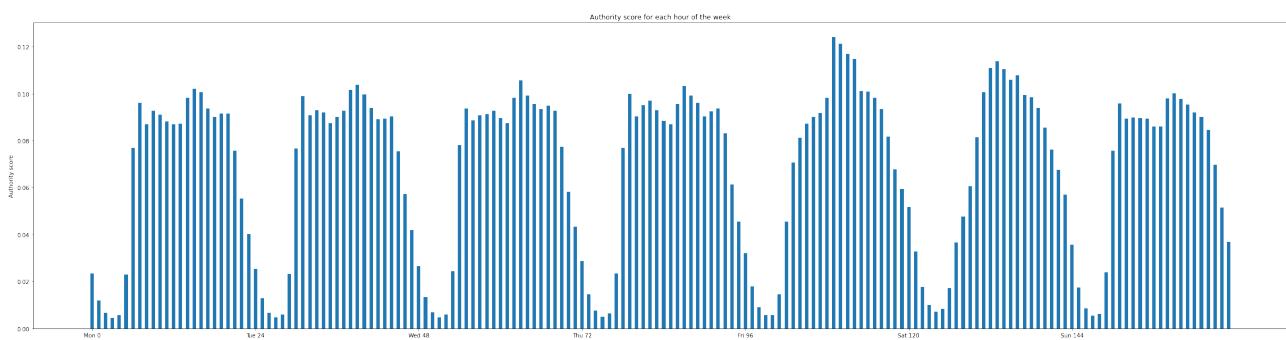
بنابراین a و h به ترتیب بردارهای ویژه ماتریس‌های $A^T A$ و $A A^T$ هستند که می‌توان آن‌ها را با تجزیه SVD ماتریس A محاسبه کرد.

می‌توانید نمودارهای مربوط به Hubها مشاهده کنید:



مهمترین hub‌ها آن‌هایی هستند که بیشترین امتیاز hub را دارند که در نمودار بالا می‌توان مشاهده کرد. Hub‌ها که دوربین‌ها در نظر گرفته شده‌اند، پرترددترین مسیرها را نشان می‌دهند.

حال نمودار امتیاز Authority‌ها که ساعات هفته در نظر گرفته شده‌اند را بینید.



نمودار فوق به بیانی می‌تواند بیانگر میزان شلوغی ساعات هفته باشد و زمان‌هایی را نشان دهد که بیشترین رکوردهای ماشین رد شده را ثبت کرده‌اند. اگر بررسی کنیم متوجه می‌شویم که نتایج این بخش تا حد خوبی با نتایج بخش اول که میزان تردد بین دو دوربین مجاور در ساعات مختلف هفته را بررسی کردیم مطابقت دارند.

اگر Hub‌ها را دوربین‌ها و Authority‌ها ماشین‌ها در نظر بگیریم، نیز مفاهیم جالب‌تری را نیز می‌توان استخراج کرد. در این فرمول‌بندی در واقع با پیدا کردن مهمترین Hub‌ها و Authority‌ها، مهمترین مسیرهایی که به سایر مسیرها لینک دارند را می‌توانیم پیدا کنیم. این اطلاعات به مفهوم Hub و Webpage در کانتکست Authority بسیار نزدیک است.

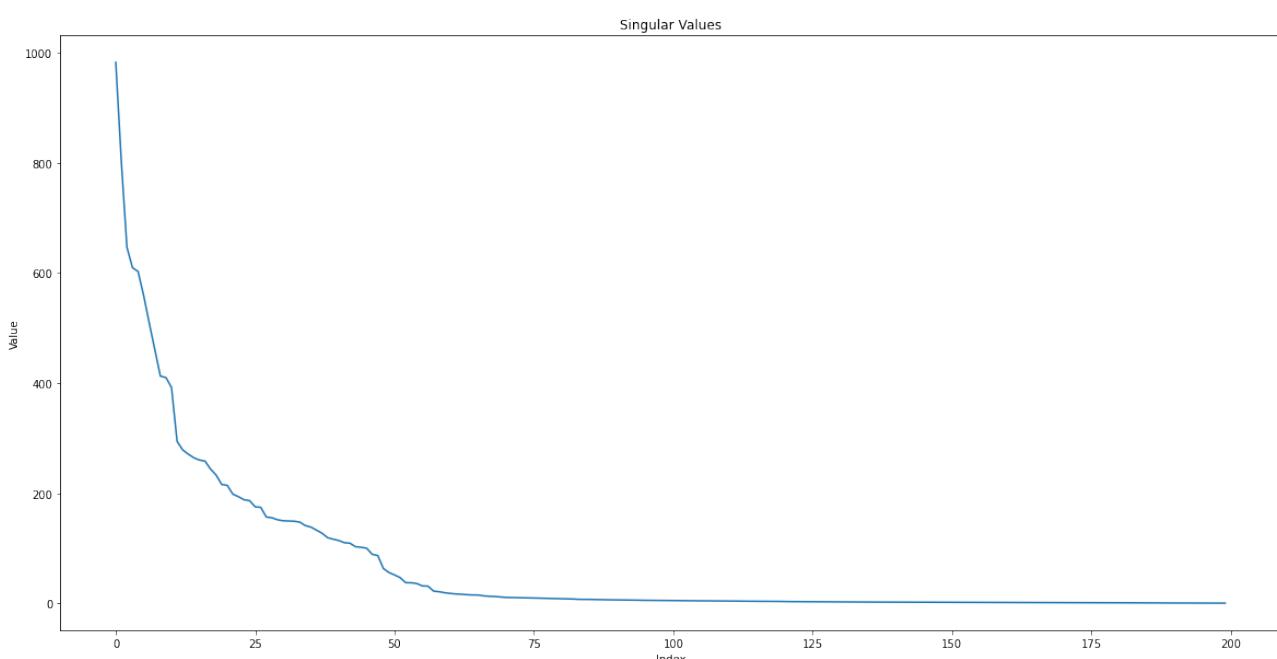
SVD •

در این بخش به پیاده‌سازی تجزیه مقادیر تکین یا SVD روی ماتریس utitliy مذکور می‌پردازیم. ابتدا ماتریس خواسته شده را می‌سازیم. ماتریس utility خودروها و دوربین‌ها ماتریسی است که سطرهای آن خودروها و ستون‌های آن دوربین‌ها هستند و هر درایه از ماتریس تعداد رکوردهای ثبت شده از خودروی متناظر با آن درایه توسط دوربین متناظر با آن درایه است. نحوه ساخت این ماتریس و پیاده‌سازی آن در نوبوک مربوط به این بخش آمد است.

برای ساخت ماتریس، ابتدا مقادیر DEVICE_CODE و FINAL_CAR_KEY را از دیتاباس گرفته و در یک RDD قرار می‌دهیم. سپس از این دو به عنوان کلید استفاده می‌کنیم و به هر جفت مقدار ۱ می‌دهیم و براساس کلید عناصر، مقادیر آن‌ها را جمع می‌زنیم تا در نهایت برای هر ماشین تعداد رکوردها به ازای هر ماشین را داشته باشیم.

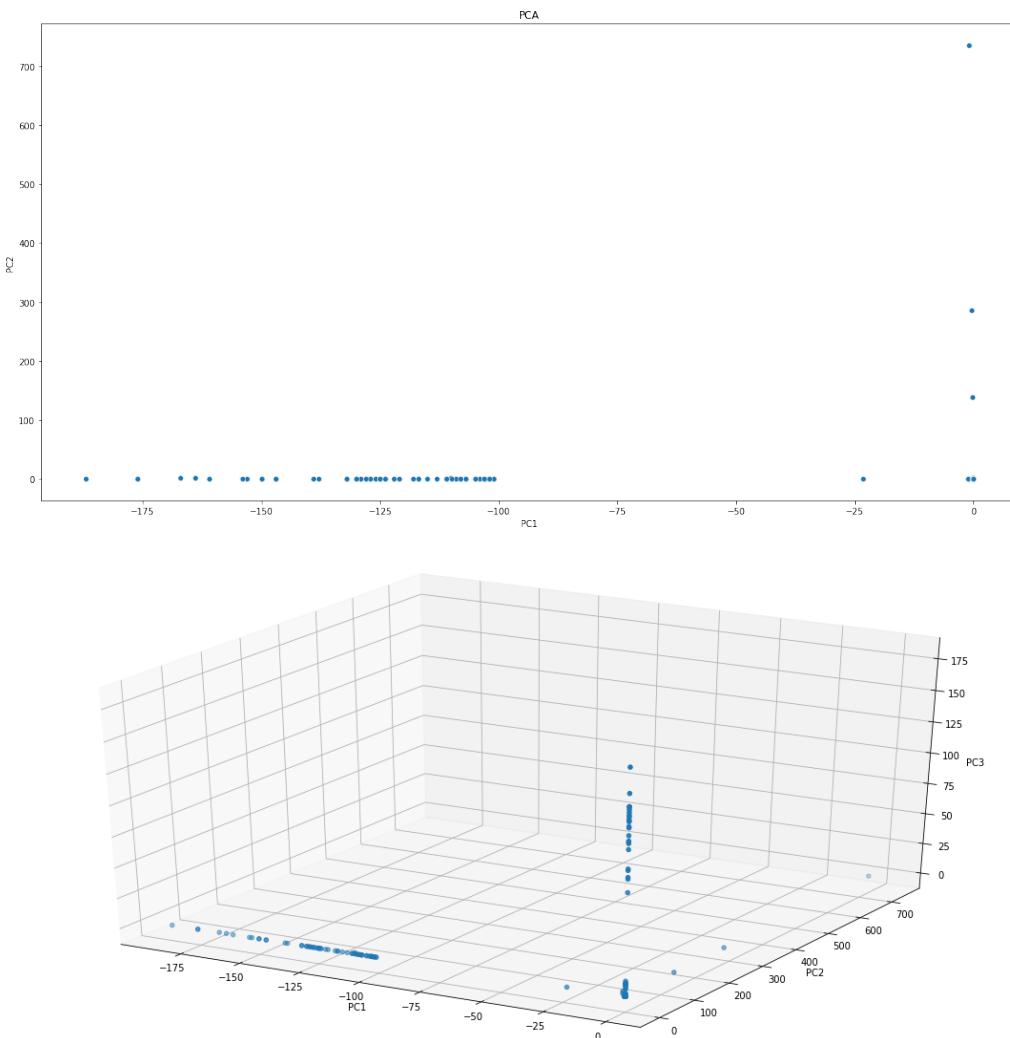
برای پیاده‌سازی SVD من از دو روش استفاده کردم. اولین روش تبدیل داده‌ها به Dataframe کتابخانه pandas و اعمال SVD کتابخانه numpy بود. روش دیگر استفاده از خود فریمورک Spark بود. در هر دو روش به نتیجه کاملاً یکسانی رسیدم. البته در پیاده‌سازی با استفاده از ماژول ml خود Spark مشکل خوردم. برای ساخت ماتریس و انجام SVD از کلاس CoordinateMatrix استفاده کردم، و از آنجایی که شماره دوربین‌ها بسیار بزرگ بود و این نوع ماتریس از کلیدهای RDD به عنوان ایندکس استفاده می‌کند، ماتریس بسیار بزرگ sparse به وجود می‌آمد که حجمش برای محاسبات SVD توسط Spark خیلی زیاد بود. برای همین هر کد دوربین را به یک عدد از ۰ تا ۳۹۵ نگاشت کردم که تعداد کدهای یکتای دوربین‌ها است.

من SVD را با ۲۰۰ مقدار تکین روی ماتریس utility اعمال کردم. نمودار مقادیر تکین حاصل به شکل زیر است:



با توجه به نمودار بالا مشاهده می‌کنید که تقریباً ۲۵ مقدار تکین اول ۹۰ درصد جمع کل مقادیر تکین را تشکیل می‌دهند. البته من این مقدار را دقیق حساب نکرده‌ام ولی به همین اعداد است. یعنی با استفاده از ۴۰ ستون اول ماتریس لاثان ۹۰ درصد داده‌های مربوط به تردد خودروها را بیان کرد. و با projection کل داده‌ها در یک فضای ۴۰ بعدی با استفاده از PCA حدود ۹۰ درصد اطلاعات حفظ می‌شود.

در ادامه با انجام PCA سعی کردم داده‌ها را به یک فضای کوچکتر ببرم تا فاکتورهای latent آن‌ها را بررسی کنم. نمودارهای PCA در ۲ و ۳ بعد را مشاهده می‌کنید:



همانطور که مشاهده می‌کنید مقادیر فاکتورهای پنهان در راستای برخی محورها توزیع شده‌اند. برداشتی که از این موضوع دادم این است که این مقادیر می‌توانند نشان‌دهنده این باشد که برخی ماشین‌ها صرفاً از برخی از دوربین‌ها و به صورت مکرر عبور می‌کنند که در واقع بیانگر همان مسیرهای معمول و همیشگی هر خودرو است. اما به طور کلی نمی‌توان براساس latent factor‌ها تحلیل دقیقی ارائه داد.

• سایر ایده‌ها

- یکی از ایده‌هایی که می‌توان پیاده کرد، دسته‌بندی دوربین‌هاست. برای این کار از همان ماتریس قسمت HITS می‌توان استفاده کرد. یعنی برای هر دوربین تعداد رکوردها در هر ساعت از هفته را در نظر بگیریم. سپس با اعمال الگوریتم LDA یا ترکیب ALS و KNN دوربین‌ها را خوشه‌بندی کنیم. انتظاری که از نتیجه این کار دارم، این است که دوربین‌های هم نوع در یک دسته قرار بگیرند، دوربین‌های نزدیک به هم نیز در دسته‌های نزدیک به هم یا در یک خوشه باشند و اینکه بتوان محله‌های خلوت و شلوغ را پیدا کرد.
- یکی از کارهای قشنگی که می‌توان انجام داد، استفاده از داده‌های قسمت CF و نزدیکی و شباهت دوربین‌ها و بازسازی تقریبی نقشه شهر با استفاده از آن‌هاست. اما فرصت نکردم آن را پیاده کنم.