به نام خدا

دانشگاه صنعتی امیرکبیر

Amirkabir University
of Technology

تکلیف دوم - تحلیل کلان داده ها

استادمربوطه: دکتر حقیر چهرقانی

نام: زهرا اخلاقی

شماره دانشجویی: ۴۰۱۱۳۱۰۶۴

بهار ۱۴۰۲

فهرست:

بخش اول Clustering

الگوریتم CURE، یک روش خوشه‌بندی است که برای پیدا کردن خوشه‌های با شکل‌های پیچیده و غیر کروی طراحی شده است. این الگوریتم یک روش خوشه‌بندی توزیعی و مقیاس‌پذیر است که می‌تواند با داده‌های غیرکرویی و با ساختار پیچیده به خوبی کنار بیاید.

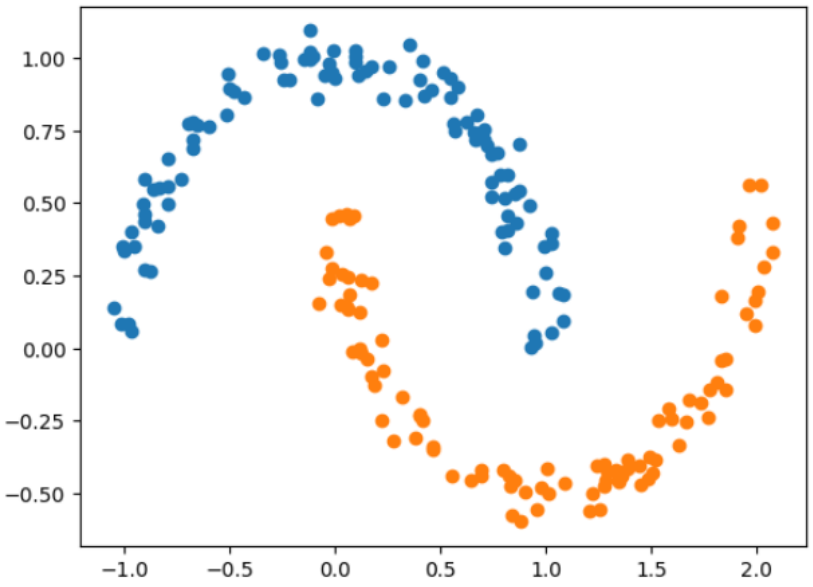
الگوریتم CURE به جای نماینده‌ی مرکزی برای هر خوشه، از چندین نماینده استفاده می‌کند که در سراسر خوشه پخش شده‌اند. این نمایندگان به یکدیگر نزدیک‌تر هستند تا از این طریق خوشه‌هایی با شکل‌های پیچیده و غیرکرویی را مدل کنند.

الگوریتم CURE شامل چندین مرحله است:

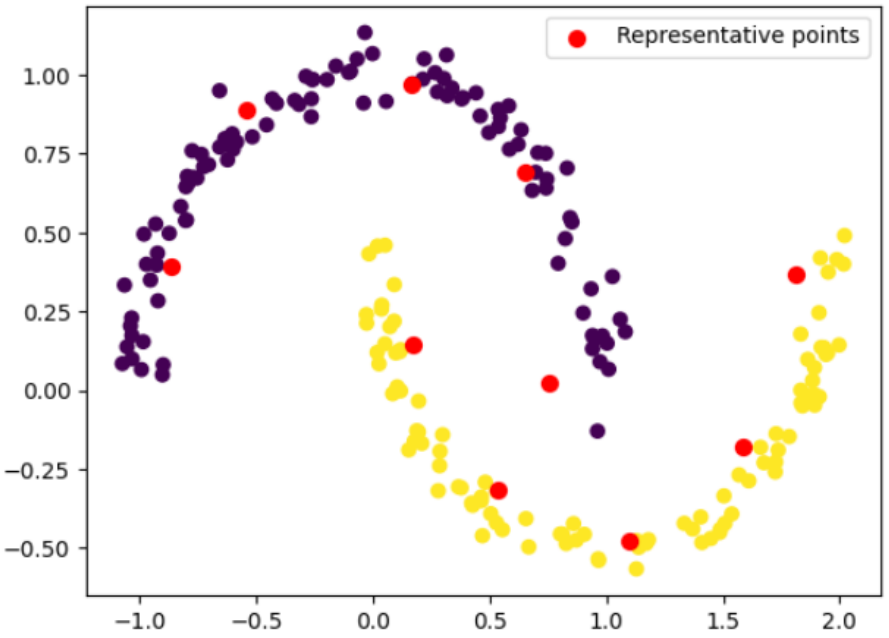
1. ابتدا، هر نقطه‌ی داده به عنوان یک خوشه در نظر گرفته می‌شود.
2. سپس، فاصله بین هر دو خوشه محاسبه می‌شود و دو خوشه‌ی نزدیک به هم الحاق می‌شوند.
3. این فرآیند تا زمانی که تعداد مشخصی خوشه باقی مانده باشد، ادامه می‌یابد.
4. در این مرحله، برای هر خوشه، نقاطی که بیشترین فاصله را از یکدیگر دارند به عنوان نمایندگان انتخاب می‌شوند. این نمایندگان سپس به سمت مرکز خوشه حرکت می‌کنند تا حساسیت به نقاط پرت کاهش یابد.
5. در نهایت، برای پیش‌بینی خوشه‌ی یک نقطه جدید، نقطه به خوشه‌ای اختصاص می‌یابد که نماینده‌ی نزدیک‌ترین به آن دارد.

نتیجه اجرا روی مجموعه داده اول با انتخاب ۲۰۰ نمونه تصادفی:

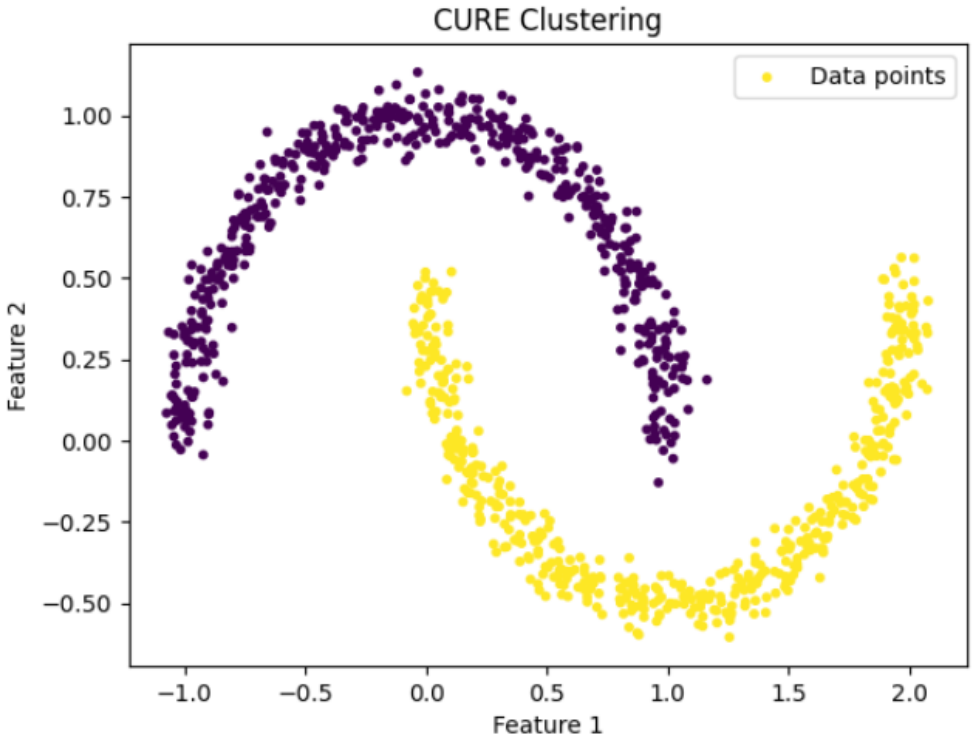
خوشه بندی ۲۰۰ نمونه تصادفی:



انتخاب ۵ Representation point برای هر خوشه:

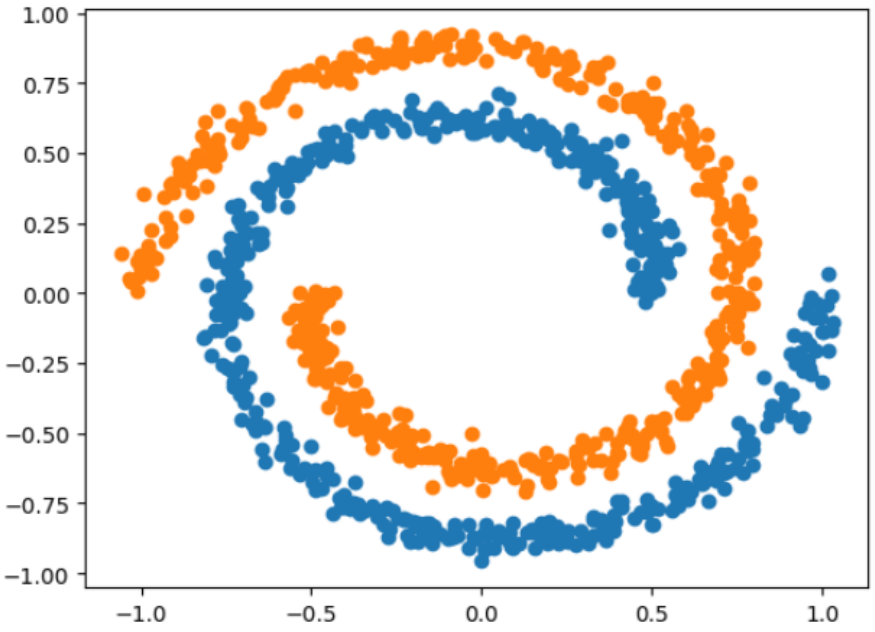


نتیجه نهایی خوشه بندی:

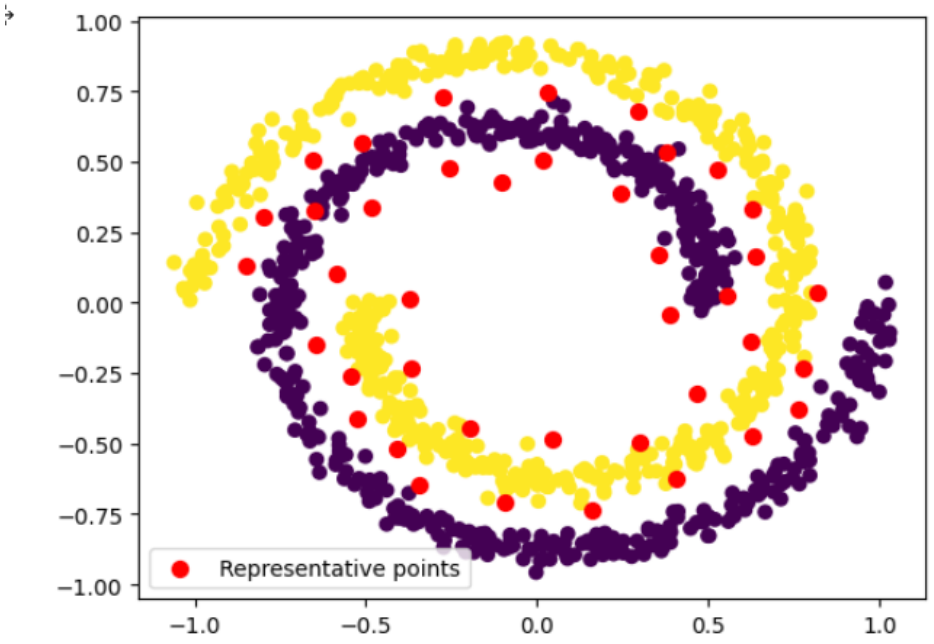


نتیجه اجرا روی مجموعه داده اول با انتخاب ۱۰۰۰ نمونه تصادفی:

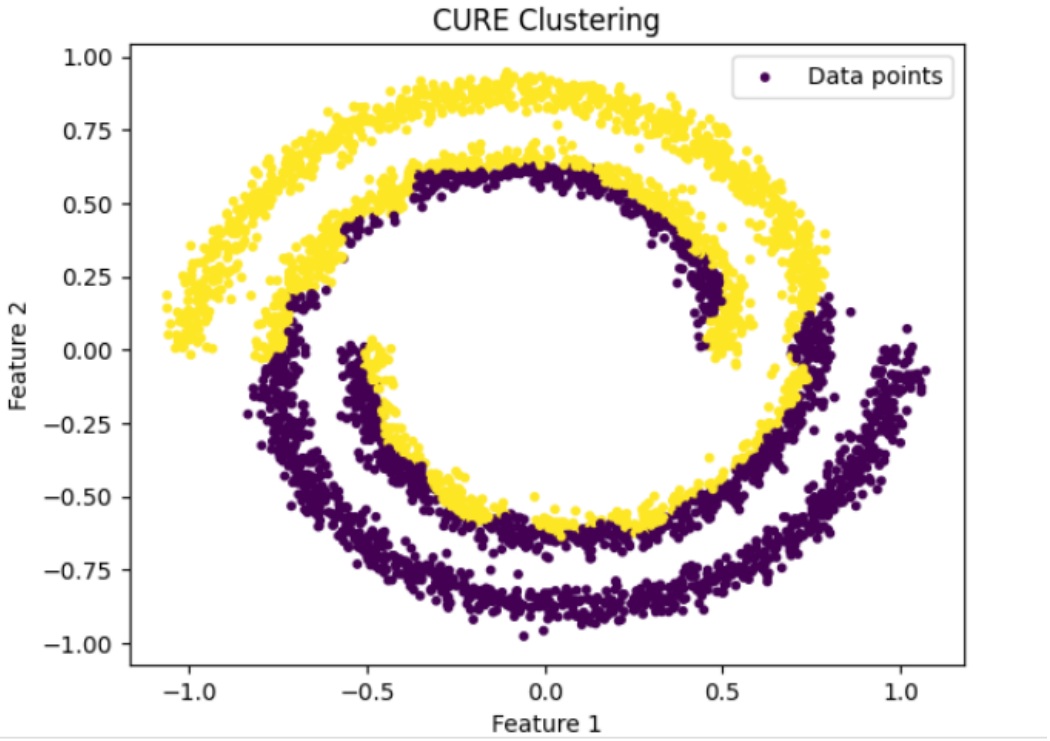
خوشه بندی ۱۰۰۰ نمونه تصادفی:



انتخاب ۲۰ Representation point برای هر خوشه:



نتیجه نهایی خوشه بندی:



همانطور که در تصویر بالا مشخص است، بعضی از داده ها در این خوشه بندی در دسته مناسبی قرار نگرفته اند، با توجه به اینکه در این مجموعه داده فاصله دو ماه نزدیک به یکدیگر است و مرکز خوشه بیرون از دیتاست ها قرار دارد، با حرکت دادن نقاط نماینده ۲۰ درصد به سمت مرکز خوشه باعث میشود نقاط نماینده ۲۰ درصد به مرکز نزدیک تر شده و در نتیجه باعث خوشه بندی اشتباه در فاصله میان دو ماه می‌شود.

نقاط نماینده با حرکت دادن به مرکز نزدیک تر شده و باعث میشود نزدیک به خوشه دیگری قرار بگیرند که در نتیجه باعث خوشه بندی اشتباه در این مجموعه داده می‌شود.

د)

الگوریتم BFR یک الگوریتم خوشه‌بندی است که از فرضیاتی در مورد داده‌ها استفاده می‌کند که آن‌ها را برای مجموعه‌های بزرگ داده که به صورت دیسک بر روی دیسک ذخیره می‌شوند، مناسب می‌کند.

در BFR، داده‌ها در سه دسته تقسیم می‌شوند:

1. خوشه‌های کامل: خوشه‌هایی که کاملا مدل شده‌اند و تمامی نقاط آن‌ها در حافظه اصلی موجود است.
2. خوشه‌های جزئی: خوشه‌هایی که فقط جزئی از آن‌ها مدل شده است و برخی از نقاط آن‌ها در حافظه اصلی موجود است.

3. نقاط غیر خوشه: نقاطی که هنوز در هیچ خوشه‌ای قرار نگرفته‌اند.

BFR بر اساس فرضیات زیر کار می‌کند:

- نقاط داده در هر خوشه از یک توزیع گوسی چند متغیره مستقل پیروی می‌کنند.
- همه ابعاد داده‌ها با یکدیگر مستقل هستند.
- توزیع داده‌ها در هر بُعد نرمال است.

الگوریتم BFR و CURE هر دو تکنیک‌هایی برای خوشه‌بندی در داده‌های بزرگ هستند که در مواقعی که حجم داده‌ها بیشتر از حجم حافظهٔ فیزیکی است، مفید هستند. این دو الگوریتم در نحوه تقسیم بندی داده‌ها و تعیین مراکز خوشه تفاوت دارند.

الگوریتم BFR از فرضیاتی درباره داده‌ها استفاده می‌کند تا پردازش را سریع‌تر و کم‌هزینه‌تر کند. به طور خاص، BFR فرض می‌کند که داده‌ها توزیع نرمال می‌گیرند و در نتیجه از این فرضیات برای بهینه‌سازی فرآیند خوشه‌بندی استفاده می‌کند. بنابراین، BFR برای داده‌هایی که این فرضیات را برآورده می‌کنند، عملکرد خوبی دارد.

به طور مقابل، الگوریتم CURE یک الگوریتم خوشه‌بندی هرمی است که به جای استفاده از یک نماینده واحد برای هر خوشه، از چند نقطه نمایندگی استفاده می‌کند. این تکنیک به CURE اجازه می‌دهد تا به خوشه‌هایی با شکل غیر کروی نیز بپردازد. بنابراین، CURE می‌تواند در داده‌هایی با توزیع‌های پیچیده‌تر، از جمله توزیع‌های غیرنرمال، عملکرد بهتری داشته باشد.

با این حال، CURE معمولاً نیاز به بیشترین منابع پردازشی دارد و ممکن است سرعت کمتری نسبت به BFR داشته باشد. انتخاب بین BFR و CURE بستگی به خصوصیات داده‌هایی که قرار است پردازش شوند و منابع موجود دارد.

ه)

این داده‌ها به طور معمول شکل یا ساختار خاصی ندارند که بتوانند با فرض توزیع نرمال مدل شوند و الگوریتم BFR یک روش خوشه‌بندی است که برای داده‌هایی با توزیع‌های نرمال طراحی شده است. این الگوریتم فرض می‌کند که داده‌ها در هر خوشه از توزیع گوسی (نرمال) پیروی می‌کنند. از این جهت، استفاده از الگوریتم BFR احتمالاً مناسب نخواهد بود.

با توجه به اینکه در الگوریتم BFR فرض برای توزیع داده ها وجود دارد و توزیع داده ها در این مجموع داده انتخابی به صورت نرمال نیست و الگوریتم BFR قادر به پردازش و تشخیص خوشه هایی به صورت غیر کروی مناسب نیست، نمی‌تواند انتخاب مناسبی باشد.

ی)

روش Elbow ی در الگوریتم K-means یکی از روش‌های رایج برای انتخاب تعداد خوشه‌ها (k) است. در این روش، الگوریتم K-means برای تعدادی مختلف از خوشه‌ها اجرا می‌شود و سپس خطای (SSE, Sum of Squared Errors) برای هر تعداد خوشه رسم می‌شود. در نمودار حاصل، تعداد خوشه‌های مناسب زمانی مشخص می‌شود که نمودار شکل زانویی پیدا کند، یعنی جایی که افزایش تعداد خوشه‌ها منجر به کاهش قابل توجه خطا نمی‌شود.

با این حال، استفاده از روش Elbow برای تعیین k در الگوریتم K-NN معمولاً مناسب نیست. K-NN یک الگوریتم دسته‌بندی است و k در اینجا تعداد همسایگان نزدیک‌ترین برای در نظر گرفتن در پیش‌بینی برچسب یک نمونه است. در این حالت، ما به دنبال تعدادی از همسایگان هستیم که بتواند تعادل مناسبی بین بیاس و واریانس ایجاد کند. بنابراین، روش Elbow که بر اساس SSE کار می‌کند در اینجا مناسب نیست و به جای استفاده از روش Elbow، می‌توان از اعتبارسنجی متقابل (Cross-Validation) استفاده کرد. این روش با اجرای K-NN برای مقادیر مختلف k و مقایسه دقت دسته‌بندی، مقدار مناسب k را تعیین می‌کند.

بخش دوم Filter Bloom

(Bloom Filter) یک ساختار داده احتمالاتی است که برای بررسی عضویت یک عنصر در یک مجموعه استفاده می‌شود. این ساختار داده از توابع هش متعدد استفاده می‌کند تا عضویت یک عنصر را بررسی کند و به صورت زیر کار می‌کند:

1. ابتدا یک بیت آرایه به طول مشخص ایجاد می‌کنیم که همه بیت‌های آن صفر هستند.
2. برای اضافه کردن یک عنصر، عنصر را به تمام توابع هش ورودی می‌دهیم. خروجی هر تابع هش یک اندیس برای بیت آرایه است. ما هر بیت مربوطه را به یک تغییر می‌دهیم.
3. برای بررسی عضویت یک عنصر، عنصر را به همان توابع هش ورودی می‌دهیم. اگر همه بیت‌های مربوطه یک باشند، ما فرض می‌کنیم که عنصر در مجموعه است. در غیر این صورت (یعنی حداقل یک بیت صفر است)، عنصر قطعاً در مجموعه نیست.

نکته اصلی این است که فیلتر بلوم ممکن است خطای (false positives) داشته باشد، یعنی ممکن است یک عنصر را که در واقع در مجموعه نیست، در مجموعه باشد. اما فیلتر بلوم هیچ‌گاه خطای (false negatives) ندارد، یعنی اگر یک عنصر در واقع در مجموعه باشد، همیشه درست تشخیص داده می‌شود.

الف)

با توجه به اینکه حجم داده ها بسیار زیاد است و اضافه شده هر نام کاربری نیازمند این است که مقدار آن با داده های قبلی بررسی شود، اگر از الگوریتم bloom-filter برای انجام این کار استفاده نشود، ورودی جدید باید با هر یک از داده های قبلی برای تطابق بررسی شود و مسئله دارای مرتبه زمانی o(n) میباشد.
در صورت استفاده از bloom-filter داده جدید وارد شده به تابع هش داده میشود، این تابع هش داده ورودی را به عددی بین 0تاm مپ میکند و داده های قبلی نیز به این تابع هش داده شده اند و به هر خانه ای که توسط تابع هش تعیین می شود مقدار آن از صفر به یک تغییر پیدا میکند، حال اگر داده ورودی جدید حاصل از تابع هش مقدار آن خانه برابر با ۱ باشد به این معناست که داده از قبل وجود داشته است و مرتبه زمانی مسئله به o(1) تغییر پیدا میکند.
با توجه به اینکه حجم داده بسیار زیاد است و هر نام ورودی باید تطابق آن با نام های قبلی بررسی شود، استفاده از bloom-filter میتواند انتخاب مناسبی باشد. یکی از مشکلات bloom-filter خطای false positive میباشد که برای این مسئله قابل قبول است (نام کاربری که در سیستم وجود ندارد به اشتباه تشخیص داده شود که وجود دارد).

ب)

با توجه به اینکه در این قسمت از ۴ تابع hash استفاده میشود، احتمال FP به صورت زیر تعریف میشود :

$$\text{false positive probability} = (1 - e^{-km/n})^k$$

با توجه به درصد داده شده و اینکه $k=4$, $m= 1000000$ میباشد و درصد احتمال داده شده نسبت m/n باید ⅕ باشد، $n=5000000$ میباشد.

پس ۵ برابر انتخاب میشود.

استفاده از تابع هش اول روی مجموعه داده:

```
hash_functions_I = [lambda s, p=p: hashI(p, string_to_num(s), M) for p in p_values]
bloom_filter_I = BloomFilter(M, hash_functions_I)

for user_name in user_dataset.UserNames:
     bloom_filter_I.add(user_name)
```

ج)

استفاده از تابع هش دوم روی مجموعه داده:

```
[21] hash_functions_II = [lambda s, p=p: hashII(p, string_to_num(s), M) for p in p_values]
     bloom_filter_II = BloomFilter(M, hash_functions_II)
```

```
[22] for user_name in user_dataset.UserNames:
         bloom_filter_II.add(user_name)
```

د)

```
[19] calculate_false_positive(user_requests, bloom_filter_I)

     TP:2128, TN:41719, FP:6154, FN:0
     false positive rate of hash type:0.12854845111022914
```

ه)

```
user_requests = csv.reader(open(path2));

calculate_false_positive(user_requests, bloom_filter_II)
```

```
TP:2128, TN:43501, FP:4372, FN:0
false positive rate of hash type:0.09132496396716312
```

ی)

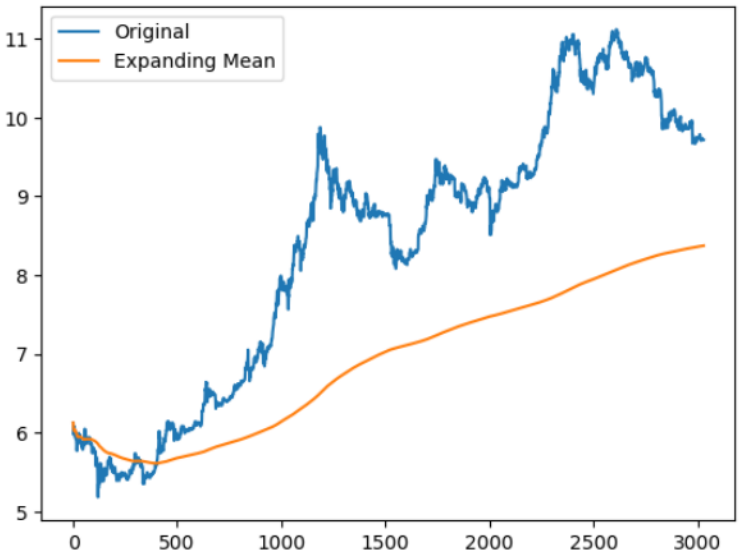
برای تابع دوم مقدار مورد انتظار برای False Positive با نتیجه ارائه شده مطابقت دارد ولی برای تابع اول مقدار False Positive بزرگتر از حد انتظار میباشد.

false positive در فیلتر بلوم به چندین عامل بستگی دارد، از جمله تعداد توابع هش، اندازه فیلتر بلوم (یعنی تعداد بیت‌هایی که در آرایه بیت استفاده می‌شوند)، و تعداد عناصری که به فیلتر اضافه شده‌اند.

با توجه به اینکه داده دیتاست آزمایش و تست برای هر دو تابع یکسان است و تعداد ورودی و توابع هش برای تابع هش نوع اول و دوم یکسان است به نظر می رسد مشکل از تابع هش اول است که قدرت کافی برای اینکه داده ها را به صورت نرمال توزیع کند ندارد و تابع هش باید به طور یکنواخت داده‌ها را بر روی فضای هش پخش کند. اگر تابع هش داده‌ها را به طور ناهمگون پخش کند، احتمال برخورد (و در نتیجه، false positive) افزایش می‌یابد.

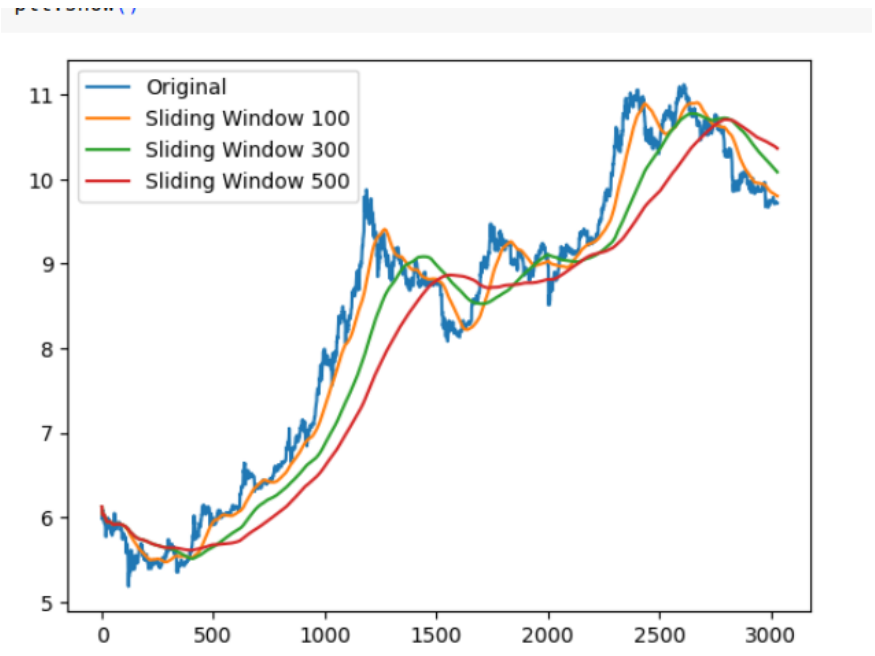
بخش سوم Data Stream

الف)



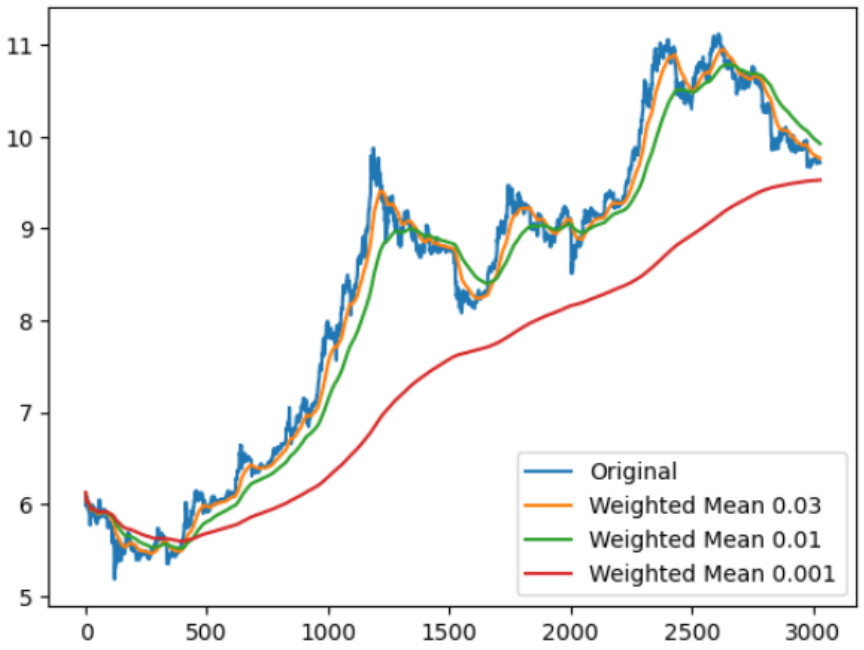
ب)

۱- پیاده سازی روش sliding window برای محاسبه میانگین:



همانطور که در شکل بالا نشان داده شده است، طول پنجره در محاسبه میانگین تاثیر گذار است و همه c روز گذشته از اهمیت یکسانی برخوردار هستند بنابراین روش مناسبی نیست.

۲- استفاده از میانگین وزن دار:



هر چه step size بزرگتر باشد، نمودار به حالت اصلی نزدیک تر است، یعنی تاثیر داده فعلی در محاسبه میانگین بیشتر است.

استفاده از میانگین وزن دار به علت اینکه درصد اهمیت نقاط در نتیجه میانگین گیری با گذشت زمان به صورت نمایی کاهش میابد و همه داده های گذشته از وزن یکسانی برخوردار نیستند بهتر است.

ج)

برای پیاده سازی این قسمت از هر دو روش میانگین وزن دار و پنجره ای به طول ثابت استفاده شده است.

با علت اینکه داده جدید ممکن است پرت باشد بهتر است واریانس در طول یک با بازه زمانی با میانگین واریانس مقایسه شود، برای رسیدن به این هدف از پنجره ای با طول ثابت ۸ استفاده شده است، اگر از پنجره به طول ثابت استفاده نشود ممکن است داده های نویزی به اشتباه به عنوان تغییر توزیع داده ها تشخیص داده شوند.

برای محاسبه میانگین واریانس از روش میانگین گیری وزن دار استفاده شده است، به طوری که داده های جدید اهمیت بیشتری دارند و این مقدار به صورت نمایی کاهش میابد.

در نهایت با استفاده از threshold اگر نسبت واریانس به میانگین آن از مقدار threshold بزرگتر بود داده به عنوان هشدار اضافه میشود و در plot نمایش داده میشود.

پیچیدگی زمانی این الگوریتم از o(1) میباشد. زمانی که داده جدید وارد میشد در آن بازه مد نظر واریانس محاسبه میشود و سپس میانگین واریانس بر اساس آن بروز می شود و در صورت اینکه نسبت آنها از threshold بزرگتر بود به عنوان هشدار اضافه می‌شود.