

به نام خدا

درس تحلیل کلان داده‌ها

تمرین سری دوم

سنا موحدین ۹۹۳۱۰۸۰

سؤال اول

بخش ۱)

مراحل الگوریتم LSH:

۱. shingling شینگلینگ یک تکنیک است که برای تبدیل یک سند به نمایش مجموعه‌ای (بردار بولین) استفاده می‌شود. این شامل شکستن سند به زیرمجموعه‌های کوچکتر و همپوشانی دار از عناصری مانند کلمات یا حروف است. سپس هر زیرمجموعه به عنوان یک عنصر منحصر به فرد در مجموعه نمایش داده می‌شود، جایی که حضور یا عدم حضور زیرمجموعه نشان می‌دهد که آیا آن در سند وجود دارد یا خیر. با استفاده از شینگلینگ، ما می‌توانیم حضور ویژگی‌های مهم در سند را ضبط کنیم در حالی که ترتیب دقیق این ویژگی‌ها را نادیده می‌گیریم.

۲. min-hash: مین-هشینگ یک روش برای تبدیل مجموعه‌های بزرگ به امضاهای کوتاه است، در حالی که همچنان اندازه مشابهی را حفظ می‌کند. این با استفاده از هش کردن عناصر هر مجموعه به تعداد کمتری از بیشه‌ها انجام می‌شود، معمولاً با استفاده از یک تابع هش. ایده کلیدی این استفاده از چندین تابع هش است و کمترین مقدار هش را برای هر تابع نگه داشته، ایجاد کردن یک امضا برای هر مجموعه. این امضاهای به مراتب کوچکتر از مجموعه‌های اصلی هستند اما هنوز ویژگی را دارند که مجموعه‌های با محتوای مشابه احتمالاً امضاهای مشابهی داشته باشند. مین-هشینگ به طور خاص برای مقایسه مجموعه‌های بزرگ به طور موثر در حالت‌هایی مانند تحلیل تشابه سند و سیستم‌های توصیه بکار می‌رود.

۳. LSH: هش کردن حساس به مکان بر روی جفت‌های امضاهای مرکز می‌کند که احتمالاً از اسناد مشابهی باشند. این با هش کردن امضاهای اسناد به بیشه‌ها به گونه‌ای انجام می‌شود که امضاهای مشابه احتمالاً در همان یا نزدیک همان بیشه‌ها هش کرده شوند. این امکان را برای بازیابی موثر اسناد مشابه فراهم می‌کند با پرس‌وحجی تنها در یک زیرمجموعه از بیشه‌ها به جای کل مجموعه. LSH مرحله حیاتی در جستجوی نزدیکترین همسایه تقریبی است، زیرا این امکان را فراهم می‌کند که جفت‌های نامزد که احتمالاً مشابه هستند را شناسایی کند و هزینه محاسباتی را نسبت به روش‌های جستجوی کامل به طور قابل توجهی کاهش دهد.

بخش ۲)

یکی از اصلی‌ترین معایب الگوریتم هشینگ حساس به مکان (LSH) این است که می‌تواند به مقدار زیادی حافظه و فضای ذخیره‌سازی نیاز داشته باشد، به خصوص وقتی با داده‌های با بعد بالا کار می‌کنیم. این مشکل ناشی از نیاز به ایجاد چندین جدول هش برای دستیابی به تعادل مناسب بین بازیابی و دقت است. هر جدول هش شامل بسیاری از توابع هش و سطلهای هش حاصل می‌شود که می‌تواند منجر به استفاده زیاد از حافظه شود. علاوه بر این، نیاز به نگهداری چندین جدول هش می‌تواند پیچیدگی محاسباتی را نیز افزایش دهد، هم از نظر زمان و هم از نظر فضای زیرا هر پرس و جو ممکن است نیاز داشته باشد که در چندین جدول پردازش شود تا احتمال بالایی برای یافتن آیتم‌های مشابه داشته باشد. این معاوضه بین تعداد جداول هش و دقت نتایج شباهت یک نکته کلیدی در پیاده‌سازی LSH است.

علاوه بر این مسائل می‌توان به کاهش دقت برای برخی توزیع‌های داده هم به عنوان یکی از معایب این الگوریتم اشاره کرد. زمانی که نقاط داده به طور یکنواخت توزیع شده‌اند یا الگوی توزیع خاصی را دنبال می‌کنند که با فرضیات تابع هش مطابقت دارد، خوب عمل می‌کند. با این حال، اگر داده‌ها دارای توزیع نامنظم یا خوش‌های باشند، LSH ممکن است نتواند به طور مؤثر آیتم‌های مشابه را در همان سطلهای گروه‌بندی کند که منجر به کاهش دقت در یافتن آیتم‌های مشابه می‌شود. این به این دلیل است که LSH بر این فرض متکی است که آیتم‌های مشابه به همان یا به باکت‌های نزدیک هش می‌شوند، فرضی که ممکن است در همه توزیع‌های داده صدق نکند.

بخش ۳)

a. در این بخش، با استفاده از کد نوشته شده بررسی می‌کنیم که اگر کتگوری قبلاً به دیکشنری‌ای که تعریف کرده‌ایم اضافه نشده باشد، آن را اضافه می‌کنیم. سپس برای اضافه کردن آیتم‌های مربوط به هر کتگوری از `set` استفاده می‌کنیم، به این صورت، عدم وجود اسمی تکراری به خودی خود هندل خواهد شد. در نهایت برای این که مشخص شود اسمی تکراری نوشته نشده‌اند، می‌توانیم موارد مربوط به هر کتگوری را سورت کرده و سپس نمایش دهیم. (برای مثال، در همان اوایل دیتابست مشخص است که کاربر ۵۰۷۱ و ۲۷۴۷، هر دو `bike` خریده‌اند ولی در نمایش آن، `Part a` در نوبت بوك مشخص شده.)

(کد مربوط به این بخش با عنوان `Part a` در نوبت بوك مشخص شده)

Category: Books	Category: Clothing	Category: Electronics	Category: Home & Kitchen	Category: Sports & Outdoors
Name: Adventure	Name: Backpack	Name: Action Camera	Name: Air Fryer	Name: Baseball Bat
Name: Animals	Name: Belt	Name: Air Purifier	Name: Air Purifier	Name: Basketball
Name: Art Book	Name: Blazer	Name: Baby Monitor	Name: Barbecue Grill	Name: Bike
Name: Biography	Name: Blouse	Name: Bluetooth Speaker	Name: Blender	Name: Boxing Gloves
Name: Business	Name: Bodysuit	Name: Camera	Name: Bread Maker	Name: Camping Stove
Name: Cars	Name: Boots	Name: Car Dash Cam	Name: Computer Speaker	Name: Camping Tent
Name: Children's Book	Name: Cap	Name: Cordless Phone	Name: Coffee Maker	Name: Climbing Rope
Name: Classic Literature	Name: Cardigan	Name: Coat	Name: Countertop Oven	Name: Compass
Name: Cookbook	Name: Coat	Name: Dress	Name: Drome	Name: Cooler
Name: Crafts	Name: Dress Shirt	Name: Dress Shirt	Name: E-reader	Name: Cycling Helmet
Name: DIY	Name: Duffle Bag	Name: Duffle Bag	Name: Electric Blanket	Name: Dartboard
Name: Design	Name: Gloves	Name: Gloves	Name: Electric Scooter	Name: Dumbbells
Name: Drama	Name: Hat	Name: Hat	Name: External Hard Drive	Name: Exercise Bike
Name: Education	Name: Hoodie	Name: Hoodie	Name: Fitness Scale	Name: First Aid Kit
Name: Fantasy	Name: Jacket	Name: Jacket	Name: Fitness Tracker	Name: Fishing Rod
Name: Fashion	Name: Jeans	Name: Jeans	Name: GPS Navigator	Name: Football
Name: Finance	Name: Kimono	Name: Kimono	Name: Gaming Console	Name: Goalkeeper Gloves
Name: Fitness	Name: Leggings	Name: Leggings	Name: Headphones	Name: Gym Clubs
Name: Food	Name: Overalls	Name: Overalls	Name: Home Assistant	Name: Gym Bag
Name: Gardening	Name: Pajamas	Name: Pajamas	Name: Home Security Camera	Name: Headlamp
Name: Graphic Novel	Name: Pants	Name: Pants	Name: Key Finder	Name: Helmet
Name: Health	Name: Polo Shirt	Name: Polo Shirt	Name: LED TV	Name: Hiking Backpack
Name: History	Name: Poncho	Name: Poncho	Name: Laptop	Name: Hiking Boots
Name: Hobbies	Name: Raincoat	Name: Raincoat	Name: Microphone	Name: Hockey Stick
Name: Humor	Name: Robe	Name: Robe	Name: Mini Fridge	Name: Jump Rope
Name: Medicine	Name: Sandals	Name: Sandals	Name: Monitor	Name: Kayak
Name: Music	Name: Scarf	Name: Scarf	Name: Portable Charger	Name: Outdoor Grill
Name: Mystery	Name: Shorts	Name: Shorts	Name: Power Bank	Name: Outdoor Heater
Name: Nature	Name: Skirt	Name: Skirt	Name: Printers	Name: Paddle Board
Name: Parenting	Name: Slippers	Name: Slippers	Name: Projector	Name: Picnic Basket
Name: Photography	Name: Sneakers	Name: Sneakers	Name: Robot Arm	Name: Portable Grill
Name: Poetry	Name: Socks	Name: Socks	Name: Router	Name: Punching Bag
Name: Psychology	Name: Suit	Name: Suit	Name: Satellite Radio	Name: Resistance Bands
Name: Religion	Name: Sweater	Name: Sweater	Name: Smart Bulb	Name: Running Shoes
Name: Romance	Name: Sweatpants	Name: Sweatpants	Name: Smart Scale	Name: Skateboard
Name: Science Fiction	Name: Swimsuit	Name: Swimsuit	Name: Smart Watch	Name: Ski Poles
Name: Self-Help	Name: T-Shirt	Name: T-Shirt	Name: Smartphone	Name: Sleeping Bag
Name: Space	Name: Tank Top	Name: Tank Top	Name: Stand Mixer	Name: Snowboard
Name: Sports	Name: Tie	Name: Tie		Name: Snowshoes
Name: Sustainability	Name: Trench Coat	Name: Trench Coat		Name: Soccer Ball
Name: Technology				Name: Sunglasses

```

categories = {}
for index, row in data.iterrows():
    category = row[object_category_column]
    name = row[object_name_column]
    # Add the category to the dictionary if it's not already present
    if category not in categories:
        categories[category] = set()
    # Add the object name to the category's set (avoids duplicates automatically)
    categories[category].add(name)
# Sort categories and within each category, sort the object names
sorted_categories = sorted(categories.items())
# Print sorted categories and their corresponding sorted unique names
for category, names in sorted_categories:
    sorted_names = sorted(names)
    print(f'Category: {category}!')
    for name in sorted_names:
        print(f'  Name: {name}')

```

b. (در این بخش بعد از توضیح توابع پیاده‌سازی شده، بخشنی از نتیجه‌ی آن تابع هم قرار داده خواهد شد. کد مربوط به این بخش با عنوان Part b در نوبت‌بوق مشخص شده).

برای این بخش باید الگوریتم LSH را پیاده‌سازی کنیم. ابتدا برای هر کاربر (هر آیدی) تمام آیتم‌هایی که خریده را در کنار هم قرار می‌دهیم تا بتوانیم در ادامه shingling را انجام دهیم.

```
customer_objects = {}
for index, row in data.iterrows():
    customer_id = row[customer_id_column]
    object_name = row[object_name_column]
    if customer_id not in customer_objects:
        customer_objects[customer_id] = set()
    customer_objects[customer_id].add(object_name)
customer_objects_combined = {customer_id: ''.join(sorted(names)) for customer_id, names in customer_objects.items()}
print("Combined object names for each customer:")
for customer_id, combined_names in customer_objects_combined.items():
    print(f'Customer ID: {customer_id}, Combined Names: {combined_names}')
```

سپس باید shingling را انجام دهیم. در اینجا $k=2$ تعریف کردیم به همین دلیل توکن‌ها ۲ تا ۲ تا خواهند بود.

```
def shingle(text: str, k: int):
    shingle_set = []
    for i in range (len(text) - k + 1):
        shingle_set.append(text[i:i+k])
    return(set(shingle_set))

k = 2
customer_shingles = {customer_id: shingle(object_names, k) for customer_id, object_names in customer_objects_combined.items()}
for customer_id, shingles in customer_shingles.items():
    print(f'Customer ID: {customer_id}')
    print(f' Shingles: {shingles}')


Customer ID: 3310
Shingles: {'Du', 'la', 'tr', 'ut', 'ba', 're', '0', 'E', 't', 'Be', 'G', 'ec', 'ph', 'ag', 'B', 'le', 'ri', 'Sl', 'nd', 'M', 'hi', 'uf', 'td', 'ne', 'to', 'Bl
Customer ID: 7091
Shingles: {'de', 'Ka', 't', 'G', 'B', 'le', 'rf', 'nd', 'M', 'Yo', 'Sn', 'rp', 'tt', 'ea', 'ne', 'to', 'Bl', 'ya', 'ug', 'ui', 'am', 'ay', 'et', 'r', 'Ki', 'K
Customer ID: 6928
Shingles: {'fi', 'ip', 'Fr', 'Fo', 'ad', 'oo', 'E', 'Co', 'Ai', 'or', 'in', 'de', 'd', 're', 'id', 'Sk', 'ki', 'C', 'rt', 'ge', 'dl', 'E', 'r', 'P', 'ur', 'r-
Customer ID: 6387
Shingles: {'la', 'ol', 're', 't', 'Be', 'G', 'ec', 'B', 'ri', 'Mo', 'Pa', 'ee', 'M', 'ys', 'ck', 'rp', 'ea', 'ne', 'ui', 'am', 'si', 'r', 'K', 'ke', 'te', 'rb
Customer ID: 2346
Shingles: {'ol', 'Hi', 'nt', 't', 'G', 'B', 'le', 'Mo', 'Pa', 'H', 'M', 'Tr', 'An', 'ls', 'ck', 'ea', 'ne', 'am', 'si', 'sh', 'gn', 'D', 'r', 'Ga', 'ke', 'te
```

بعد یک مجموعه از تمام توکن‌های موجود درست می‌کنیم. پس از آن، برای هر آیدی، به ازای توکن‌هایی که دارد، خانه‌ی مربوطه را مساوی ۱ و به ازای آن‌هایی که نداشت، ۰ می‌گذاریم.

```
union_set = set.union(*customer_shingles.values())
binary_dict = {}
for key, value in customer_shingles.items():
    binary_representation = []
    for item in union_set:
        if item in value:
            binary_representation.append(1)
        else:
            binary_representation.append(0)
    binary_dict[key] = binary_representation
for key, value in binary_dict.items():
    print(f"\{key}: {value}\")
```

به بخش `hash` می‌رسیم. در ابتدا توابع `hash` را تولید می‌کنیم. در اینجا تعريف شده که از ۵۰ تابع هش مختلف استفاده شود. در ادامه این توابع را روی داده‌ها پیاده می‌کنیم.

```
def create_hash(vector: list):
    signature = []
    for func in minhash_func:
        for i in range(1, len(union_set)+1):
            indx = func.index(i)
            sig_val = vector[indx]
            if sig_val == 1:
                signature.append(indx)
                break
    return signature
hashed_signatures = {}

for key, value in binary_dict.items():
    hashed_signatures[key] = create_hash(value)

for key, value in hashed_signatures.items():
    print(f"Key: {key}, Hashed Signature: {value}")

minhash_func = build_minhash_func(len(union_set), 50)
print(minhash_func)

Key: 3310, Hashed Signature: [184, 204, 26, 164, 167, 186, 111, 111, 38, 38, 235, 141, 109, 36, 290, 109, 101, 86, 38, 31, 284, 255, 183, 378, 109, 225, 118, 305, 430
Key: 7091, Hashed Signature: [184, 287, 345, 401, 315, 46, 111, 265, 38, 38, 235, 240, 109, 376, 115, 109, 170, 88, 115, 339, 372, 255, 371, 378, 78, 339, 118, 104, 1
Key: 6920, Hashed Signature: [178, 89, 235, 164, 167, 46, 235, 265, 320, 268, 235, 320, 109, 376, 290, 109, 170, 88, 218, 89, 284, 378, 183, 378, 109, 225, 118, 305,
Key: 6387, Hashed Signature: [184, 294, 235, 414, 57, 186, 111, 111, 66, 254, 235, 131, 109, 227, 290, 109, 101, 86, 218, 31, 107, 358, 101, 378, 364, 225, 118, 104,
Key: 2346, Hashed Signature: [40, 247, 315, 164, 330, 256, 214, 107, 38, 338, 235, 195, 109, 331, 115, 109, 136, 86, 115, 245, 107, 255, 155, 378, 315, 225, 118, 22,
```

حالا باید باکت‌بندی کنیم. ما ۵۰ تابع هش داشتیم، در اینجا تعريف کرده‌ایم که ۱۰ باکت داشته باشیم، پس هر باکت ۵ تایی خواهد بود.

```
def split_vec(signature, b):
    assert len(signature) % b == 0
    r = int(len(signature) / b)
    subvec = []
    for i in range(0, len(signature), r):
        subvec.append(signature[i:i+r])
    return subvec
bands = {}
for key, value in hashed_signatures.items():
    bands[key] = split_vec(hashed_signatures[key], 10)
for key, value in bands.items():
    print(f"Key: {key}, bands: {value}")

Key: 3310, bands: [[209, 386, 259, 151, 201], [381, 285, 301, 331, 79], [125, 165, 414, 46, 316], [3, 416, 344, 10, 210], [290, 331, 171, 93, 144], [229, 306, 177, 22
Key: 7091, bands: [[254, 386, 407, 324, 201], [381, 190, 212, 20, 431], [177, 155, 414, 51, 386], [71, 89, 110, 430, 241], [71, 331, 171, 407, 144], [23, 386, 389, 57
Key: 6920, bands: [[40, 100, 439, 394, 201], [381, 79, 223, 433, 79], [177, 202, 238, 258, 313], [3, 162, 110, 10, 242], [330, 348, 40, 313, 330], [200, 306, 177, 225
Key: 6387, bands: [[209, 44, 292, 55, 201], [381, 285, 425, 20, 431], [14, 165, 349, 51, 386], [3, 262, 55, 10, 341], [207, 14, 397, 94, 123], [229, 386, 279, 57, 214
Key: 2346, bands: [[84, 198, 439, 55, 201], [381, 79, 212, 433, 431], [339, 73, 198, 431, 232], [130, 280, 55, 232, 341], [101, 348, 278, 130, 136], [104, 202, 279, 2
```

در نهایت باید کاربران شبیه به هم را پیدا کنیم. به طور کلی در الگوریتم، اگر دو کاربر حداقل یک باکت مثل هم داشته باشند، کاندیدای شباهت خواهند بود. در اینجا ما ۱۰ کاربر برتر به کاربران مدنظر ۱۰ کاربر را مشخص خواهیم کرد را می‌خواهیم شناسایی کنیم. به همین دلیل برای کاربرانی که کاندیدای شبیه بودن هستند، تعداد باکت‌های شبیه را می‌شماریم و سپس ۱۰ تای شبیه‌تر را بر می‌گردانیم.

در این کد ما برای بررسی میزان شباهت از معیار جکارد استفاده می‌کنیم. دلایل متعددی برای استفاده از آن و خوب بودن این معیار در چنین داده‌هایی وجود دارد. برای مثال در بسیاری از برنامه‌ها، داده‌ها به طور طبیعی به عنوان مجموعه‌ها نمایش داده می‌شوند، مانند اسناد که با مجموعه‌های کلمات یا مواردی که با مجموعه‌های ویژگی‌ها نمایش داده می‌شوند. در این مساله هم ما از داده‌ها به همین صورت استفاده می‌کنیم. به علاوه شباهت جاکارد در برابر انحراف داده مقاوم است، به این معنی که حتی زمانی که اندازه مجموعه‌هایی که مقایسه می‌شوند به طور قابل ملاحظه‌ای متفاوت است، عملکرد خوبی دارد. این مهم است در LSH، جایی که اندازه مجموعه‌ها می‌تواند به طور گسترده‌ای متغیر باشد.

```

def find_similar(desired_costumer_id, bands):
    similarities = []
    for key in bands:
        for desired_band, other_band in zip(bands[desired_costumer_id], bands[key]):
            if desired_costumer_id != key:
                if desired_band == other_band:
                    similarity = jaccard(set(hashed_signatures[desired_costumer_id]), set(hashed_signatures[key]))
                    similarities.append((key, similarity))
                    break
    similarities.sort(key=lambda x: x[1], reverse=True)
    return similarities

def print_part_b(similarities, desired_customer_id, num):
    print(f"\n{num}) top 10 similar candidates for customer {desired_customer_id}:")
    for i, (customer_id, similarity) in enumerate(similarities[:10]):
        print(f" {i+1}) customer ID: {customer_id}, Similarity: {similarity:.2f}")

def jaccard(a, b):
    return len(a.intersection(b)) / len(a.union(b))

```

خروجی نهایی:

1) top 10 similar candidates for customer 3310:	5) top 10 similar candidates for customer 9362:	8) top 10 similar candidates for customer 2812:
1) customer ID: 1470, Similarity: 0.49	1) customer ID: 1538, Similarity: 0.34	1) customer ID: 4678, Similarity: 0.49
2) customer ID: 7218, Similarity: 0.39	2) customer ID: 4262, Similarity: 0.31	2) customer ID: 2750, Similarity: 0.48
3) customer ID: 1812, Similarity: 0.38	3) customer ID: 4214, Similarity: 0.28	3) customer ID: 5211, Similarity: 0.46
4) customer ID: 2856, Similarity: 0.38	4) customer ID: 8530, Similarity: 0.27	4) customer ID: 3972, Similarity: 0.46
5) customer ID: 4996, Similarity: 0.37	5) customer ID: 7988, Similarity: 0.27	5) customer ID: 8637, Similarity: 0.45
6) customer ID: 5332, Similarity: 0.36	6) customer ID: 4661, Similarity: 0.26	6) customer ID: 2656, Similarity: 0.44
7) customer ID: 8667, Similarity: 0.36	7) customer ID: 5027, Similarity: 0.26	7) customer ID: 4589, Similarity: 0.43
8) customer ID: 7948, Similarity: 0.36	8) customer ID: 9621, Similarity: 0.25	8) customer ID: 5296, Similarity: 0.42
9) customer ID: 2776, Similarity: 0.36	9) customer ID: 2455, Similarity: 0.25	9) customer ID: 1041, Similarity: 0.42
10) customer ID: 2834, Similarity: 0.35	10) customer ID: 3996, Similarity: 0.25	10) customer ID: 1941, Similarity: 0.42
2) top 10 similar candidates for customer 3441:	6) top 10 similar candidates for customer 6785:	9) top 10 similar candidates for customer 1277:
1) customer ID: 6372, Similarity: 0.46	1) customer ID: 6712, Similarity: 0.35	1) customer ID: 6445, Similarity: 0.42
2) customer ID: 2747, Similarity: 0.45	2) customer ID: 3644, Similarity: 0.32	2) customer ID: 3938, Similarity: 0.36
3) customer ID: 8476, Similarity: 0.44	3) customer ID: 3246, Similarity: 0.31	3) customer ID: 1174, Similarity: 0.35
4) customer ID: 3940, Similarity: 0.44	4) customer ID: 2980, Similarity: 0.31	4) customer ID: 5608, Similarity: 0.34
5) customer ID: 9425, Similarity: 0.43	5) customer ID: 4830, Similarity: 0.31	5) customer ID: 6372, Similarity: 0.32
6) customer ID: 2246, Similarity: 0.39	6) customer ID: 4683, Similarity: 0.30	6) customer ID: 5211, Similarity: 0.32
7) customer ID: 1607, Similarity: 0.39	7) customer ID: 7591, Similarity: 0.30	7) customer ID: 1853, Similarity: 0.31
8) customer ID: 2348, Similarity: 0.38	8) customer ID: 1756, Similarity: 0.29	8) customer ID: 7286, Similarity: 0.31
9) customer ID: 7369, Similarity: 0.38	9) customer ID: 2156, Similarity: 0.29	9) customer ID: 2173, Similarity: 0.30
10) customer ID: 5278, Similarity: 0.38	10) customer ID: 3411, Similarity: 0.28	10) customer ID: 1428, Similarity: 0.30
3) top 10 similar candidates for customer 9317:	7) top 10 similar candidates for customer 1590:	10) top 10 similar candidates for customer 5489:
1) customer ID: 6711, Similarity: 0.36	1) customer ID: 1634, Similarity: 0.42	1) customer ID: 2779, Similarity: 0.50
2) customer ID: 2374, Similarity: 0.30	2) customer ID: 5507, Similarity: 0.41	2) customer ID: 5211, Similarity: 0.43
3) customer ID: 4512, Similarity: 0.29	3) customer ID: 6617, Similarity: 0.35	3) customer ID: 9293, Similarity: 0.40
4) customer ID: 7802, Similarity: 0.28	4) customer ID: 9805, Similarity: 0.33	4) customer ID: 4560, Similarity: 0.40
5) customer ID: 5169, Similarity: 0.28	5) customer ID: 5139, Similarity: 0.31	5) customer ID: 7943, Similarity: 0.38
6) customer ID: 6781, Similarity: 0.28	6) customer ID: 3277, Similarity: 0.30	6) customer ID: 2497, Similarity: 0.38
7) customer ID: 9333, Similarity: 0.25	7) customer ID: 9032, Similarity: 0.30	7) customer ID: 1491, Similarity: 0.37
8) customer ID: 9447, Similarity: 0.25	8) customer ID: 1112, Similarity: 0.30	8) customer ID: 2244, Similarity: 0.37
4) top 10 similar candidates for customer 4376:	9) customer ID: 6919, Similarity: 0.29	9) customer ID: 9852, Similarity: 0.37
1) customer ID: 2851, Similarity: 0.39	10) customer ID: 2870, Similarity: 0.29	10) customer ID: 2713, Similarity: 0.37
2) customer ID: 5760, Similarity: 0.38		
3) customer ID: 9403, Similarity: 0.38		
4) customer ID: 2475, Similarity: 0.37		
5) customer ID: 6911, Similarity: 0.36		
6) customer ID: 1605, Similarity: 0.35		
7) customer ID: 5300, Similarity: 0.35		
8) customer ID: 8565, Similarity: 0.34		
9) customer ID: 9356, Similarity: 0.33		
10) customer ID: 9074, Similarity: 0.32		

نکته:

طبق خروجی دریافتی، می‌توانیم تعداد توابع هش یا تعداد باکت‌ها را تغییر دهیم. برای مثال، بالا بردن اندازه‌ی باکت‌ها دقیق را ممکن است بالا ببرد ولی ممکن است که نتواند به تعداد دلخواه ما، کاربر مشابه با کاربر مدنظر پیشنهاد دهد. پس باید این موارد را در نظر داشته باشیم.

برای این بخش از نتایج بخش قبلی استفاده می‌کنیم. می‌توانیم با توجه به آیتم‌های خریداری شده توسط کاربران شبیه به هر کاربر مدنظر تصمیم بگیریم. پس تعداد تکرار آیتم کاربران شبیه را محاسبه کرده و بیشترین آن‌ها را بر می‌گردانیم. در این کد، ۳ آیتم پیشنهادی برای هر کاربر مدنظر خواهیم داشت.

```
def suggest_products(desired_customer_ids, bands, products_data, num_top_products=3):
    suggested_products = {}
    for desired_customer_id in desired_customer_ids:
        similarities = find_similars(desired_customer_id, bands)
        similar_customers = [customer_id for customer_id, _ in similarities[:10]]
        similar_products = []
        product_source = {}
        # Gather products purchased by similar customers
        for customer_id in similar_customers:
            for product in products_data.get(customer_id, []):
                similar_products.append(product)
                if product not in product_source:
                    product_source[product] = []
                product_source[product].append(customer_id)
        # Count occurrences of each product
        product_counts = Counter(similar_products)
        # Sort products by frequency
        sorted_products = sorted(product_counts.items(), key=lambda x: x[1], reverse=True)
        # Extract top products
        top_products = [product for product, _ in sorted_products[:num_top_products]]
        # Store suggested products with sources
        suggested_products[desired_customer_id] = {
            "products": top_products,
            "sources": {product: product_source[product] for product in top_products}
        }
    return suggested_products

purchase_data = pd.read_csv(data_file_path)
products_data = {}
# Group products purchased by each customer
for _, row in purchase_data.iterrows():
    customer_id = row["customer_id"]
    product_name = row["object_name"]
    if customer_id not in products_data:
        products_data[customer_id] = []
    products_data[customer_id].append(product_name)
# Assuming hashed_signatures and bands are defined
desired_customer_ids = [3310, 3441, 9317, 4376, 9362, 6785, 1590, 2812, 1277, 5489]
suggested_products = suggest_products(desired_customer_ids, bands, products_data, num_top_products=3)
# Print suggested products with sources
for num, desired_customer_id in enumerate(desired_customer_ids, start=1):
    similarities = find_similars(desired_customer_id, bands)
    data = suggested_products[desired_customer_id]
    products = data["products"]
    sources = data["sources"]
    print(f"\nRecommended products for customer {desired_customer_id}:")
    for product in products:
        source_customers = sources[product]
        print(f"Product: {product}")
        print("Customers who purchased this product:")
        for customer in source_customers:
            print(f"Customer ID: {customer}")
```

خروجی نهایی:

```
Recommended products for customer 3310:
Product: Car Dash Cam
Customers who purchased this product:
Customer ID: 1487
Customer ID: 5866
Customer ID: 5296
Customer ID: 5296
Customer ID: 3352
Product: Science Fiction
Customers who purchased this product:
Customer ID: 2416
Customer ID: 6238
Customer ID: 9852
Customer ID: 9293
Customer ID: 3352
Product: Indoor Grill
Customers who purchased this product:
Customer ID: 1487
Customer ID: 5296
Customer ID: 6238
Customer ID: 9273
```

Recommended products for customer 3441:
Product: Tennis Shoes
Customers who purchased this product:
Customer ID: 6757
Customer ID: 6757
Customer ID: 6372
Customer ID: 2812
Product: Classic Literature
Customers who purchased this product:
Customer ID: 6757
Customer ID: 6681
Customer ID: 7900
Product: Exercise Bike
Customers who purchased this product:
Customer ID: 6757
Customer ID: 2812
Customer ID: 8309

Recommended products for customer 9317:
Product: Graphic Novel
Customers who purchased this product:
Customer ID: 5195
Customer ID: 7315
Product: LED TV
Customers who purchased this product:
Customer ID: 7315
Customer ID: 7315
Product: Turtleneck
Customers who purchased this product:
Customer ID: 5195

Recommended products for customer 4376:
Product: Action Camera
Customers who purchased this product:
Customer ID: 1672
Customer ID: 3936
Customer ID: 4456
Customer ID: 5376
Customer ID: 5407
Product: Headlamp
Customers who purchased this product:
Customer ID: 4260
Customer ID: 9356
Customer ID: 6889
Customer ID: 3936
Product: Science Fiction
Customers who purchased this product:
Customer ID: 4260
Customer ID: 6889
Customer ID: 5407

Recommended products for customer 9362:
Product: Photography
Customers who purchased this product:
Customer ID: 2026
Customer ID: 5766
Customer ID: 1214
Customer ID: 1214
Product: Blouse
Customers who purchased this product:
Customer ID: 2026
Customer ID: 1657
Customer ID: 4560
Customer ID: 4976
Product: Projector
Customers who purchased this product:
Customer ID: 1473
Customer ID: 4560
Customer ID: 1214

Recommended products for customer 6785:
Product: Wireless Keyboard
Customers who purchased this product:
Customer ID: 1670
Customer ID: 3644
Customer ID: 7349
Customer ID: 4578
Customer ID: 4578
Customer ID: 5172
Product: Smart Scale
Customers who purchased this product:
Customer ID: 1670
Customer ID: 4260
Customer ID: 7349
Customer ID: 7349
Customer ID: 7349
Customer ID: 4578
Product: Health
Customers who purchased this product:
Customer ID: 9403
Customer ID: 1670
Customer ID: 7349
Customer ID: 7349
Customer ID: 5172

Recommended products for customer 1590:
Product: Laptop
Customers who purchased this product:
Customer ID: 2607
Customer ID: 1634
Customer ID: 6650
Product: Sustainability
Customers who purchased this product:
Customer ID: 2607
Customer ID: 2407
Customer ID: 2407
Product: Smart Bulb
Customers who purchased this product:
Customer ID: 2607
Customer ID: 1580
Customer ID: 2455

Recommended products for customer 2812:
Product: Electric Kettle
Customers who purchased this product:
Customer ID: 8309
Customer ID: 8491
Customer ID: 9492
Customer ID: 6508
Customer ID: 8452
Customer ID: 9439
Product: Compass
Customers who purchased this product:
Customer ID: 8491
Customer ID: 1264
Customer ID: 1264
Customer ID: 1264
Customer ID: 9439
Product: Exercise Bike
Customers who purchased this product:
Customer ID: 8309
Customer ID: 8491
Customer ID: 6508
Customer ID: 3441

Recommended products for customer 5489:
Product: Knife Sharpener
Customers who purchased this product:
Customer ID: 2614
Customer ID: 7020
Customer ID: 2016
Customer ID: 2473
Customer ID: 2995
Customer ID: 2995
Customer ID: 2995
Customer ID: 9356
Customer ID: 3250
Product: Food Processor
Customers who purchased this product:
Customer ID: 7020
Customer ID: 2995
Customer ID: 9356
Customer ID: 1264
Customer ID: 1699
Product: Jeans
Customers who purchased this product:
Customer ID: 2995
Customer ID: 9356
Customer ID: 3250
Customer ID: 1699

Recommended products for customer 1277:
Product: Trash Can
Customers who purchased this product:
Customer ID: 9649
Customer ID: 8107
Customer ID: 9314
Customer ID: 3408
Product: Underwear
Customers who purchased this product:
Customer ID: 1607
Customer ID: 6669
Customer ID: 6669
Product: Fashion
Customers who purchased this product:
Customer ID: 9649
Customer ID: 6468
Customer ID: 2698

سوال دوم

بخش ۱)

۱) a)

1 0 0 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0

* Bit of value ۱ comes:

0 0 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1

merge:

0 0 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1

* Bit of value ۱ comes:

0 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 1 1

* Bit of value ۰ comes and then Bit of value ۱ comes:

1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1

merge:

1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1

merge: value ۱ comes

1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1

merge: value ۰ comes then Bit of value ۱ comes

1 1 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1

* Bit of value ۱ comes:

1 0 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1

* Bit of value ۰ comes, then Bit of value ۱ comes:

1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1

merge:

1 1 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1

in each time we will get

$$b) \underbrace{1+2+2+4}_{\text{in each time we will get}} + \frac{1}{2} \times 8 = 13$$

2) 1, 3, 2, 4, 2, 4, 1, 3, 1, 1, 3, 4, 2, 2, 1

a) 0th moment = number of distinct elements = 4

b) 1st moment = count of the numbers of elements =

length of the stream = 15

c) 2nd moment = surprise number 1 → count = 5 2nd moment =

$$\text{Kü-moment} = \sum_{i \in A} (m_i)^k$$

1 → count = 5 2 → count = 4 $\Rightarrow 5^2 + 4^2 + 3^2 + 3^2$
 3 → count = 3 4 → count = 3 = 59

d) $E(n \times (2 \times x.\text{value} - 1))$

$x_1.\text{el} = 3 \rightarrow x_1.\text{val} = 3$, $x_2.\text{el} = 1 \rightarrow x_2.\text{val} = 4$, $x_3.\text{el} = 1 \rightarrow x_3.\text{val} = 2$

$$\frac{15 \times (2 \times 3 - 1) + 15 \times (2 \times 4 - 1) + 15 \times (2 \times 2 - 1)}{3} = \underline{\underline{75}}$$

3) 5, 1, 2, 3, 3, 4, 2, 5, 1

a) $h(x) = (4x+1) \bmod 16$ $r(a) = \text{tail}$

$$x=5 \rightarrow 21 \bmod 16 = 5 \rightarrow 0101 \rightarrow \text{tail}=0$$

$$x=1 \rightarrow 5 \bmod 16 = 5 \rightarrow 0101 \rightarrow \text{tail}=0$$

$$x=2 \rightarrow 9 \bmod 16 = 9 \rightarrow 1001 \rightarrow \text{tail}=0$$

$$x=3 \rightarrow 13 \bmod 16 = 13 \rightarrow 1101 \rightarrow \text{tail}=0$$

$$x=4 \rightarrow 17 \bmod 16 = 1 \rightarrow 0001 \rightarrow \text{tail}=0$$

$$\Rightarrow 2^R = 2^0 = 1 \quad | \text{ estimated number of distinct elements using } h_1$$

b) $h_2(x) = (3x+6) \bmod 16$

$$x=1 \rightarrow 9 \bmod 16 = 9 \rightarrow 1001 \rightarrow \text{tail}=0$$

$$x=2 \rightarrow 12 \bmod 16 = 12 \rightarrow 1100 \rightarrow \text{tail}=2$$

$$x=3 \rightarrow 15 \bmod 16 = 15 \rightarrow 1111 \rightarrow \text{tail}=0$$

$$x=4 \rightarrow 18 \bmod 16 = 2 \rightarrow 0010 \rightarrow \text{tail}=1$$

$$x=5 \rightarrow 21 \bmod 16 = 5 \rightarrow 0101 \rightarrow \text{tail}=0$$

$$\Rightarrow 2^R = 2^2 = 4 \quad | \text{ estimated number of distinct elements using } h_2$$

بخش ۴)

a. توضیح روند کد مربوط به :dgim

کد ارائه شده الگوریتم DGIM (Datar-Gionis-Indyk-Motwani) را پیاده سازی می کند، که برای تخمین تعداد ۱ ها در یک پنجره کشویی از جریان های داده باینری استفاده می شود. اجزای اصلی کلاس های «Bucket» و «DGIM» هستند. کلاس «Bucket» دو ویژگی دارد «end_time» که نشان دهنده مهر زمانی آخرین بیت در باکت است و «size» که تعداد ۱ ثانیه را در آن را نشان می دهد. از این آبجکت در ادامه استفاده خواهد شد. کلاس 'DGIM' پنجره کشویی را مدیریت می کند و لیستی از این باکت ها را برای تخمین ۱ ها در اندازه پنجره مشخص نگهداری می کند. کلاس «DGIM» چند تابع مهم دارد. تابع update برای الگوریتم مرکزی است. زمان فعلی را به روز می کند، باکت های منقضی شده ای را که از پنجره بیرون می افتد حذف می کند و بیت ورودی را پردازش می کند. اگر بیت ۱ باشد، یک باکت جدید ایجاد می شود. این روش همچنین باکت های با همان اندازه را ادغام می کند تا از حفظ ویژگی های DGIM اطمینان حاصل کند و در نتیجه از تعداد بیش از حد باکت ها جلوگیری می کند. پس از بهروزرسانی باکت ها، تعداد ۱ ها را با استفاده از روش تخمین جمع آوری تخمین می زند که اندازه های همه باکت ها در پنجره به جز آخرین باکت را جمع می کند و برای تخمین دقیق تر نیمی از اندازه آخرین باکت را اضافه می کند. تابع estimate_sum تقریبی از تعداد ۱ ها در پنجره جاری را با جمع کردن اندازه های باکت ها، همراه با تصحیح برای آخرین باکت نیمه پر شده، ارائه می کند. این کلاس همچنین شامل تابعی است که وضعیت فعلی پنجره کشویی را چاپ می کند. در بخش مربوطه در نوت بوک، پنجره ها به همراه سایز باکت قدیمی در هر مرحله ای که آپدیتی صورت می گیرد، نشان داده شده. همچنین تخمین تعداد ۱ ها در پنجره نهایی ۲۰۰۰۰ تایی به این صورت است:

```
Estimated number of 1s in the last 2000 bits: 1146
```

b. در این بخش تابعی برای شمردن تعداد دقیق یکها در اخیرین پنجره نوشته شده. برای بهینه بودن، می توانیم از اول کل دیتا را نکنیم و صرفا ۲۰۰۰ بیت آخر را بگیریم و تعداد یکها را بشماریم.

```
def count_ones_last_2000_bits(data):
    last_2000_bits = data[-2000:]
    count = last_2000_bits.count('1')
    return count
ones_count = count_ones_last_2000_bits(data)
print("Number of 1s in the last 2000 bits:", ones_count)

Number of 1s in the last 2000 bits: 789
```

c. برای محاسبه accuracy از روش های متفاوتی می توانیم استفاده کنیم. برای مثال بدست آوردن نسبت تعداد یکهای تخمین زده شده، به تعداد واقعی یکها. ولی از آنجایی که شاید در مرحله ای عدد تخمین زده شده از عدد واقعی بیشتر باشد و در این صورت کسر گفته شده از ۱ بیشتر خواهد بود، ممکن است مقایسه را برایمان سخت کند. برای همین می توانیم از اختلاف این دو برای

استفاده کنیم. اگر از قدر مطلق استفاده کنیم، صرفاً تفاوت این دو را می‌فهمیم ولی اگر اختلاف معمولی را در نظر بگیریم، می‌توانیم این موضوع را هم متوجه شویم که تخمینمان از مقدار واقعی کمتر بوده یا بیشتر.

```
min accuracy(not using abs):-256  
min accuracy(using abs): 0  
max accuracy(not using abs):511
```

d. این تابع یک جریان داده با اینتری تصادفی با تعداد مشخصی از ۰ و ۱ تولید می‌کند. دو آرگومان نیاز دارد: «num_zeros» و «num_ones» که مقادیر دلخواه ۰ و ۱ را در جریان نشان می‌دهد. تابع یک آرایه numpy حاوی اعداد مشخص شده ۰ و ۱ ایجاد می‌کند، آنها را به هم متصل می‌کند و سپس آرایه را به هم می‌زند تا ترتیب بیت‌ها را تصادفی کند. در نهایت، عناصر آرایه numpy را به رشته تبدیل می‌کند، آنها را به یک رشته متصل می‌کند و این رشته را به عنوان جریان داده با اینتری برمی‌گرداند. این امر توزیع تصادفی تعداد مشخص شده ۰ و ۱ را در خروجی تضمین می‌کند.
تابع مربوط به تولید استریم‌های خواسته شده طبق شرایط بیان شده:

```
def generate_binary_stream(num_zeros, num_ones):  
    stream = np.array([0] * num_zeros + [1] * num_ones)  
    np.random.shuffle(stream)  
    return ''.join(map(str, stream))
```

هنگام استفاده از آن، تعداد ۰ و ۱ هایی که در هر حالت می‌خواهیم را به عنوان ورودی می‌دهیم. سپس برای اطمینان از این که طول آن درست است می‌توانیم len را پرینت کنیم. بخشی از دیتا را هم می‌توانیم به عنوان نمونه پرینت کنیم ولی از آنجایی که توزیع (با در نظر گرفتن نسبت بیان شده)، در کل طول استریم لزوماً یکی نیست، مشاهده‌ی آن لزوماً کمکی نخواهد کرد.

```
# Stream with twice as many 1s as 0s  
stream_twice_ones = generate_binary_stream(33334, 66666)  
# Stream with twice as many 0s as 1s  
stream_twice_zeros = generate_binary_stream(66666, 33334)  
# Stream with an equal number of 0s and 1s  
stream_equal = generate_binary_stream(50000, 50000)  
# Print the lengths to verify (Should be 100000)  
print(len(stream_twice_ones))  
print(len(stream_twice_zeros))  
print(len(stream_equal))  
  
✓ 0.0s  
100000  
100000  
100000
```

e. نتایج برای هر کدام به این شکل است:

برای استریم با تعداد یکهای دو برابر:

2678 بیت آخر برای 10000 accuracy

```
# output for stream with twice ones
dgim_e1 = DGIM(window_size=10000)
for bit_e1 in stream_twice_ones:
    value_e1 = int(bit_e1)
    dgim_e1.update(value_e1)
# dgim.print_window()
print("Results for stream with twice ones")
estimated_sume1 = dgim_e1.estimate_sum()
print(f"Estimated number of 1s in the last 10000 bits: {estimated_sume1}")
ones_counte1 = count_ones_last_n_bits(stream_twice_ones, 10000)
print("Number of 1s in the last 10000 bits:", ones_counte1)
min_accuracy_e1 = min(accuracy_list)
max_accuracy_e1 = max(accuracy_list)
print(f"min accuracy(not using abs):{min_accuracy_e1}")
if 0 in accuracy_list:
    print(f"min accuracy(using abs): 0")
else:
    print(f"min accuracy(using abs): 1")
print(f"max accuracy(not using abs):{max_accuracy_e1}")

```

✓ 19.5s

```
Results for stream with twice ones
Estimated number of 1s in the last 10000 bits: 9320
Number of 1s in the last 10000 bits: 6642
min accuracy(not using abs):-2048
min accuracy(using abs): 0
max accuracy(not using abs):4095
```

برای استریم با تعداد صفرهای دو برابر:

1429 بیت آخر برای 10000 accuracy

```
# output for stream with twice zeros
dgim_e2 = DGIM(window_size=10000)
for bit_e2 in stream_twice_zeros:
    value_e2 = int(bit_e2)
    dgim_e2.update(value_e2)
# dgim.print_window()
print("Results for stream with twice zeros")
estimated_sume2 = dgim_e2.estimate_sum()
print(f"Estimated number of 1s in the last 10000 bits: {estimated_sume2}")
ones_counte2 = count_ones_last_n_bits(stream_twice_zeros, 10000)
print("Number of 1s in the last 10000 bits:", ones_counte2)
min_accuracy_e2 = min(accuracy_list)
max_accuracy_e2 = max(accuracy_list)
print(f"min accuracy(not using abs):{min_accuracy_e2}")
if 0 in accuracy_list:
    print(f"min accuracy(using abs): 0")
else:
    print(f"min accuracy(using abs): 1")
print(f"max accuracy(not using abs):{max_accuracy_e2}")

```

✓ 9.7s

```
Results for stream with twice zeros
Estimated number of 1s in the last 10000 bits: 4660
Number of 1s in the last 10000 bits: 3231
min accuracy(not using abs):-2048
min accuracy(using abs): 0
max accuracy(not using abs):4095
```

برای استریم با تعداد یک و صفرهای برابر:

4035 بیت آخر برای 10000 accuracy

```
# output for stream with equal zeros and ones
dgim_e3 = DGIM(window_size=10000)
for bit_e3 in stream_equal:
    value_e3 = int(bit_e3)
    dgim_e3.update(value_e3)
# dgim.print_window()
print("Results for stream with twice ones")
estimated_sume3 = dgim_e3.estimate_sum()
print(f"Estimated number of 1s in the last 10000 bits: {estimated_sume3}")
ones_counte3 = count_ones_last_n_bits(stream_equal, 10000)
print("Number of 1s in the last 10000 bits:", ones_counte3)
min_accuracy_e3 = min(accuracy_list)
max_accuracy_e3 = max(accuracy_list)
print(f"min accuracy(not using abs):{min_accuracy_e3}")
if 0 in accuracy_list:
    print(f"min accuracy(using abs): 0")
else:
    print(f"min accuracy(using abs): 1")
print(f"max accuracy(not using abs):{max_accuracy_e3}")

```

✓ 14.6s

```
Results for stream with equal ones and zeros
Estimated number of 1s in the last 10000 bits: 9038
Number of 1s in the last 10000 bits: 5003
min accuracy(not using abs):-2048
min accuracy(using abs): 0
max accuracy(not using abs):4095
```

همان‌طور که مشخص است نتایج برای حالتی که تعداد صفرها دوباره بیکهاست بهتر از دو حالت دیگر است و بدترین حالت هم برای وقتی است که تعداد بیکها دو برابر تعداد صفرهاست. (هم از نظر زمان و هم از نظر accuracy) کمتر بودن تعداد بیکها علاوه بر این که باعث می‌شود باکت‌بندی و ادغام کمتری نیاز داشته باشیم، مسلماً در کمتر شدن خطای هم موثر است چون تخمین کمتری خواهیم داشت. این نتایج در اسکرین‌شات‌ها نمایش داده شده‌اند.

سؤال سوم

(توابع هر بخش به همراه خروجی‌شان، در نوت‌بوک مربوطه با همان عنوان مربوطه قرار دارند.)

Data Exploration and Normalization

در این سوال، در ابتدا باید دیتاست را تحلیل کرده و ببینیم چه فیچرهایی را در درنظر بگیریم و کدام‌ها را نه. ابتدا با استفاده از تابع `calculate_similarity_percentage`، میزان شباهت دیتای هر ستون را بررسی می‌کنیم. در ادامه می‌خواهیم ستون‌هایی که درصد بالایی از موارد عین هم دارند را از بین فیچرهای حذف کرده و درنظر نگیریم. وجود چنین فیچرهایی که در بین همه یا درصد زیادی از دیتاپوینت‌ها عیناً یکسان است، به کلاسترینگ کمکی نمی‌کند. علاوه بر آن، در ستون‌هایی مانند `uri`, `track_href`, `analysis_url` یک بخش مشخصی تکرار شده. با حذف آن می‌بینیم چیزی که باقی می‌ماند همان `id` است. پس این ستون‌ها هم باید در فیچرهای شرکت داده شوند. این موضوع در تابع `remove_url_prefix` هندل شده. از بین فیچرهای باقی‌مانده، `genre` هم باید حذف کنیم چون طبق خواسته‌ی سوال، این ستون، ستون هدف جهت بررسی نتایج است. من `duration_ms` را هم حذف کردم چون بنظر می‌رسد اهمیت آن در دسته‌بندی‌ای که هدفش ژانر است، خیلی زیاد نیست و چون دیتاست بسیار بزرگ است، حذف یک فیچر هم می‌تواند در سرعت الگوریتم‌ها موثر باشد. علاوه بر این موارد، جهت نرمال‌سازی می‌توانیم فیچرهای را بصورت عددی مپ کنیم که این مورد را با استفاده از تابع `enocde_genre` می‌توان هندل کرد. نکته‌ی مهم دیگری که در بخش پیش‌پردازش اهمیت دارد، این است که با خانه‌های `null` و بدون دیتا چه کنیم. همان طور که گفته شد، دو سنتونی که تماماً خالی بودند در همان ابتدا حذف شدند. ولی این مورد را برای ستون‌های مورد استفاده در الگوریتم باید مدنظر قرار دهیم. به این منظور از تابع `count_null_values` استفاده می‌کنیم و می‌بینیم در فیچرهای باقی‌مانده هیچ خانه‌ای `null` نبوده و در نتیجه نیازی به اقدام اضافه نیست.

:CURE کلی الگوریتم

در الگوریتم CURE از فاصله اقلیدسی استفاده می‌کنیم. این الگوریتم این اجازه را می‌دهد که کلاسترها هر شکلی داشته باشند. مبنای الگوریتم استفاده از یک سری `representative` برای هر کلاستر است. مراحل انجام این الگوریتم را به طور کلی می‌توان به دو مرحله‌ی `starting` و `ending` تقسیم کرد که البته هر کدام از این‌ها خودشان از مراحلی تشکیل شده‌اند. در ابتدا به صورت رندوم یک سری سمپل استفاده می‌کنیم. تعداد این سمپل بستگی به میزان حافظه‌ی در دسترس دارد. سپس باید این نقاط به صورت سلسله مراتبی با نقاط یا کلاسترها مشابهشان گروه شوند. سپس باید برای هر کلاستر تعدادی `representative` انتخاب کنیم. این نقاط انتخاب شده بهتر است بیشترین فاصله را از هم داشته باشند. علاوه بر آن در این بخش یک مرحله‌ی دیگر هم داریم که هر `representative` را به اندازه‌ی آلفا (مثلاً ۲۰ درصد) به سمت مرکز کلاستر حرکت می‌دهیم. در بخش بعدی باید دیتاست را دوباره اسکن کرده و هر نقطه‌ی `p` را در دیتاست مشاهده کنیم، نزدیکترین `representative` به نقطه‌ی `p` را پیدا کرده و آن را به کلاستر مربوط به آن نماینده اضافه می‌کنیم. شرط پایان الگوریتم می‌تواند این باشد که به تعداد خاصی از کلاستر رسیده باشیم. البته موارد دیگری هم می‌تواند دخیل باشد. مثلاً نوع کلاسترینگ سلسله‌مراتبی پیاده‌سازی شده، نمونه‌های برداشته شده و...

توضیح کد پیاده‌سازی شده‌ی مربوط به CURE:

در کد مربوطه ابتدا یکتابع hierarchical clustering تعریف می‌کنیم که بخش کلاسترینگ ما را انجام دهد. در ادامه کد را همان‌طور که پیش‌تر هم بیان شد، به دو pass تقسیم می‌کنیم. در اول کلاسترینگ اولیه و انتخاب رپرزنتیوها صورت می‌گیرد و در pass دوم بخش اضافه شدن هر نقطه به کلاستر نزدیکیش تعریف می‌شود. کارایی الگوریتم CURE به عوامل مختلفی بستگی دارد. در کل در این الگوریتم های پرپارامترهایی مانند تعداد کلاسترها، اندازه‌ی نمونه‌ی اولیه، تعداد نماینده‌های هر کلاستر و آلفا دارد که باید آن‌ها را تعیین کنیم و مقداری که برای آن مشخص می‌کنیم نقش تعیین کننده‌ای در خوبی عملکرد الگوریتم دارد. در این تمرین تعداد کلاسترها را برای الگوریتم‌های مختلف ۱۵ در نظر گرفته‌ایم. اندازه‌ی نمونه‌ی اولیه به حافظه‌ی در دسترس بستگی دارد. با توجه به این که این الگوریتم یک حالت سلسله‌مراتبی دارد، افزایش این اعداد در کاهش سرعت الگوریتم خیلی موثر است ولی کارایی را بالا می‌برد. آلفا مقداری مانند ۰.۲ درصد تعیین می‌شود. با افزایش و کاهش تعداد سمپل اولیه، به خوبی می‌توانیم تغییر میزان accuracy را ببینیم.

برای جلوگیری از محاسبات تکراری و غیر لازم می‌توان از توابع بعضی کتابخانه‌ها استفاده کرد ولی با توجه به این که در صورت سوال گفته شده از توابع آماده استفاده نشود، از این موارد استفاده نشده. (فاصله بصورت اقلیدسی محاسبه می‌شود. cdist و تابع euclidian distance معادلند).

در این کد تعداد نمونه یا همان representative‌های هر کلاستر یک عدد ثابت در نظر گرفته نشده. برای محاسبه‌ی آن برای هر کلاستر از عبارت num of points in cluster * alpfa استفاده شده. به این صورت تعداد نمونه‌های هر کلاستر با توجه به کل نقاط درون آن تعیین می‌شود و این داینامیک بودن می‌تواند به بهبود عملکرد کل کمک کند. تابع select dispersed move representatives toward centroid همین مساله را پیاده‌سازی می‌کند. همچنین تابع representativedriven نماینده‌ها به سمت مرکز را هندل می‌کند.

توضیح کلی الگوریتم BFR:

الگوریتم BFR یک الگوریتم خوشبندی است که برای مجموعه‌داده‌های بزرگ طراحی شده است. مراحل آن به شرح زیر است: Initialization: با انتخاب تصادفی یک زیرمجموعه کوچک از مجموعه‌داده به عنوان نقاط نماینده اولیه (مرکز خوش) آغاز می‌شود. سپس هر نقطه داده را به نزدیکترین مرکز خوش خود تخصیص می‌دهد. این نقاط نماینده برای خلاصه کردن مجموعه‌داده و راهنمایی فرآیند خوشبندی استفاده می‌شوند. (در این الگوریتم فرض بر این است که توزیع داده‌ها یکسان بوده است).

Batch processing and Refinement: در مرحله پردازش دسته‌ای، BFR به طور تکراری اجزای داده را پردازش می‌کند، نقاط نماینده و تخصیص‌های خوش را به روزرسانی می‌کند. این الگوریتم از یک رویکرد افزایشی برای مدیریت مجموعه‌داده‌های بزرگ بهره می‌برد. الگوریتم به طور پویا مرکز خوش‌ها و خوش‌ها را بر اساس داده‌های جدید تنظیم می‌کند و آن‌ها را بهبود می‌بخشد تا بهترین نمایش از ساختار مجموعه‌داده را ارائه دهد. این فرآیند تا همگرایی یا تا زمانی که یک معیار توقف پیش‌تعیین شده محقق شود ادامه می‌یابد. در نهایت، BFR مراکز نهایی و تخصیص‌های خوش را ارائه می‌دهد و نمایش فشرده‌ای از ساختار خوشبندی مجموعه‌داده را ارائه می‌دهد.

توضیح کد پیاده‌سازی شده‌ی الگوریتم BFR:

Initialize_centroids: این تابع به طور تصادفی k مرکز اولیه (مراکز خوش) از داده‌ها را انتخاب می‌کند و آن‌ها را به عنوان مراکز اولیه مختص به خوش‌ها بر می‌گرداند.

assign_to_centroids: این تابع فاصله هر نقطه داده از مراکز خوش را محاسبه کرده و هر نقطه را به نزدیک‌ترین مرکز خوش تخصیص می‌دهد. برچسب‌های نقاط را بر می‌گرداند که نشان‌دهنده مرکز خوش متناظر با آن نقطه است.

update_centroids: این تابع مراکز خوش را با محاسبه میانگین داده‌های تخصیص‌داده شده به هر خوش به روزرسانی می‌کند.

bfr_clustering: این تابع الگوریتم خوش‌بندی BFR را اجرا می‌کند. ابتدا مراکز اولیه ایجاد می‌شوند، سپس تا همگرایی یا ترسیدن به تعداد حداقل تکرار مشخص شده، مراحل الگوریتم (تخصیص نقاط به مراکز و به روزرسانی مراکز) اجرا می‌شود. اگر مراکز جدید و قدیمی همگرا شوند، عملیات متوقف می‌شود و مراکز نهایی به همراه برچسب‌های خوش‌ها برگردانده می‌شوند.

مقایسه‌ی BFR و CURE

الگوریتم CURE پیچیدگی بالاتری نسبت به BFR دارد. برای هر دو الگوریتم تعداد کلاستر ۱۵ در نظر گرفته شده. در ادامه و بعد از اجرای دو الگوریتم، ابتدا ژانر assign شده به هر کلاستر را پرینت کرده و آن را می‌بینیم. سپس با استفاده از تابع calculate accuracy به این شکل عمل می‌کنیم که پی از اعمال خوش‌بندی و قرار دادن لیبل هر خوش برای هر دیتاپوینت بررسی می‌کنیم که آیا با لیبل اصلی اش یکی است یا خیر. (طبق خواسته‌ی سوال هدف روی ژانر است). همان طور که گفته شد با تغییر تعداد نمونه‌های اولیه، زمان الگوریتم CURE تغییر می‌کند. برای این که بتوانیم مقایسه‌ای از عملکرد آن‌ها داشته باشیم، سعی می‌کنیم زمان اجرا را یکسان در نظر بگیریم. مشاهده می‌شود که اکیوریسی BFR بهتر است. از طرف دیگر، برای گرفتن یک اکیوریسی حدوداً یکسان، می‌بینیم که الگوریتم CURE خیلی بیشتر طول می‌کشد.

با این حال با افزایش تعداد سمپل‌های اولیه می‌توانیم نتیجه‌ی بهتری از CURE بگیریم، ولی زمان هم افزایش می‌یابد. البته این نکته هم قابل توجه است که لزوماً با افزایش تعداد نقاط سمپل اولیه، اکیوریسی بهتر نخواهد شد و به پارامترهای دیگر هم وابسته است.

جهت نمایش بهتر نتایج در دو بعد، از یک متاداده در پایتون برای کاهش بعد بهره می‌بریم. سپس هر کلاستر با یک رنگ و ژانرهای هم با اشکال مختلف نمایش داده می‌شوند.

نمایش centroid‌ها و accuracy در حالتی که زمان اجرا تقریباً یکسان باشد:

```
Accuracy for BFR Clustering: 40.76350313201749
Accuracy for CURE Clustering: 37.246188393806875
```

```

data = df[['danceability', 'energy', 'loudness', 'speechiness',
           'acousticness', 'liveness', 'valence', 'tempo']].values

k = 15
sample_size = 50
alpha = 0.2

cure_centroids, cure_labels = cure_clustering(data, k, sample_size, alpha)
print("Final centroids:")
print(cure_centroids)
print("Labels:")
print(cure_labels)

```

✓ 0.5s

Final centroids:

```

[array([ 9.18000e-01,  5.91000e-01, -4.78000e+00,  5.14000e-02,
        4.78000e-02,  0.10000e+00,  1.04021e+02]), array([[ 0.75000e-01,  5.19000e-01, -3.05000e+00,  6.63000e-02,
        5.41000e-01,  1.09000e-01,  5.68000e-02,  1.10000e+00], [ 5.00000e-01,  6.33000e-01, -8.08000e+00,  1.72000e-01,
        7.26000e-01,  1.00000e-01,  4.00000e-02,  1.00000e+00]], array([[ 5.53000e-01,  9.20000e-01, -3.47000e+00,  2.85000e-01,
        1.18000e-01,  2.24000e-01,  4.37000e-01,  1.84563e+02]], array([[ 4.88000e-01,  3.25000e-01,  1.59430e+01,  2.33000e-01,
        1.51000e-03,  2.24000e-01,  4.37000e-01,  1.84563e+02]], array([[ 4.88000e-01,  3.25000e-01,  1.59430e+01,  2.33000e-01,
        7.70000e-01,  1.09000e-01,  1.19941e+02], [ 8.20000e-01, -3.47000e+00,  2.85000e-01,
        1.18000e-01,  2.24000e-01,  4.37000e-01,  1.84563e+02]], array([[ 5.63000e-01,  9.53000e-01,  6.09400e-01,  9.53000e-01,
        7.73000e-02,  5.55000e-01,  9.04000e-01,  2.01874e+02]], array([[ 5.69400e-01,  9.53000e-01,  6.09400e-01,  9.53000e-01,
        1.41040e-02,  1.90980e-01,  3.81400e-01,  1.73982e+02]], array([[ 7.65355556e-01,  7.5568889e-01, -7.19771111e+00,
        9.22266667e-02,  2.53486222e-02], [ 3.05197778e-01,
        4.17471111e-01,  1.25358622e+02]], array([[ 6.225600e-01,  9.401200e-01, -2.606280e+00,  1.616920e-01,
        1.98800e-01,  2.919840e-01,  2.961600e-01,  1.499762e+02]], array([[ 6.27800000e-01,  7.00266667e-01, -7.31913333e+00,
        3.55000000e-01,  2.48520000e-01,  2.99333333e-02,
        4.28866667e-01,  1.09636400e-01,  5.413000e-01,  7.721000e-01, -6.732700e+00,  6.364000e-02,
        1.840499e-02,  2.121000e-01,  1.450500e-01,  2.046023e+02]], array([[ 5.568590e-01,  8.056000e-01,  7.216000e-01, -7.061800e+00,  1.436600e-01,
        2.344200e-02,  2.638200e-01,  4.614000e-01,  1.6065900e-02]], array([[ 8.22600000e-01,  5.13866667e-01, -1.06799333e+01,
        1.99000000e-01,  5.77000000e-02,  9.74533333e-02,
        4.36440000e-01,  2.87555057e-02], [ 6.16857143e-01,  8.46428571e-01, -7.84371429e+00,
        5.72314200e-02,  2.49328571e-02,  1.32085714e-01,
        1.90245714e-01,  1.32312545e+02]]))

```

Labels:

```

[12  1 13 ...  9  9 12]

```

```

data = df_preprocessed_done[['danceability', 'energy', 'loudness', 'speechiness',
                           'acousticness', 'liveness', 'valence', 'tempo']].values

k = 15
bfr_centroids, bfr_labels = bfr_clustering(data, k)
print("Final centroids:")
print(bfr_centroids)
print("Labels:")
print(bfr_labels)

```

✓ 0.4s

Final centroids:

```

[array([ 6.42561905e-01, -7.48929505e+00,  1.14243919e-01,
        1.86703067e-01,  1.84373549e-01,  3.1596920e-01,  2.13362591e+02],
[ 6.92474029e-01,  6.72539474e-01, -6.71765586e+00,  2.00137149e-01,
        1.55624586e-01,  2.03419322e-01,  5.02737483e-01,  1.98033331e-01,
        6.56859574e-01,  7.05585576e-01, -6.77989741e+00,  1.06899871e-01,
        8.69899126e-02,  2.06108712e-01,  3.04892888e-01,  1.32185557e+02],
[ 7.17934238e-01,  6.50891480e-01, -7.18512556e+00,  1.54318989e-01,
        1.83544322e-01,  1.95952939e-01,  4.14165471e-01,  1.88257196e+02],
[ 6.30444726e-01,  8.02597462e-01, -6.13930598e+00,  1.37207561e-01,
        7.57383779e-02,  2.61376227e-01,  3.44745967e-01,  1.45615424e+02],
[ 7.00000000e-01,  1.00000000e+00,  6.00000000e+00,  8.33333333e-02,
        1.25507814e-02,  1.09932041e-01,  4.94364752e-01,  2.23366246e+02],
[ 7.800580216e-01,  6.71241189e-01, -1.09501056e-01,  9.23749177e-02,
        1.88580216e-01,  1.58016750e-01,  2.42509518e-01,  1.38552536e+02],
[ 6.36197142e-01,  6.65938597e-01, -7.07215370e+00,  2.88905460e-01,
        1.65249932e-01,  1.99679933e-01,  4.2857832e-01,  1.68942671e+02],
[ 6.88718137e-01,  6.25394680e-01, -6.87612754e+00,  2.31169853e-01,
        2.01615897e-01,  1.92741912e-01,  4.40124202e-01,  8.46617402e+01],
[ 6.39185378e-01,  6.73527571e-01, -7.01486629e+00,  2.20283550e-01,
        1.76365428e-01,  2.00934263e-01,  4.67296635e-01,  1.80975151e+02],
[ 5.508080308e-01,  8.58632318e-01, -4.29488854e+00,  1.48305657e-01,
        5.96526568e-02,  2.59740903e-01,  3.23988680e-01,  1.58057238e+02],
[ 6.809161008e-01,  8.08925931e-01, -6.71449447e+00,  1.17540179e-01,
        6.51427965e-02,  2.43886763e-01,  2.6386517e-01,  1.39107896e+02],
[ 7.831616325e-01,  6.29631955e-01, -7.72675708e+00,  1.34291314e-01,
        1.74934831e-01,  1.85923688e-01,  3.54777613e-01,  1.18728667e+02]])

```

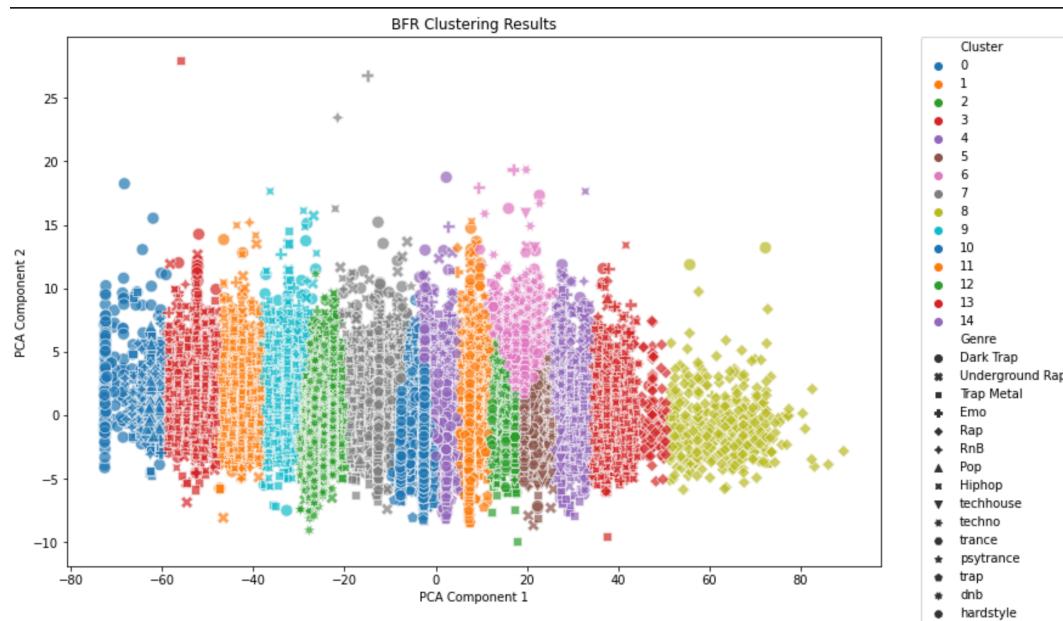
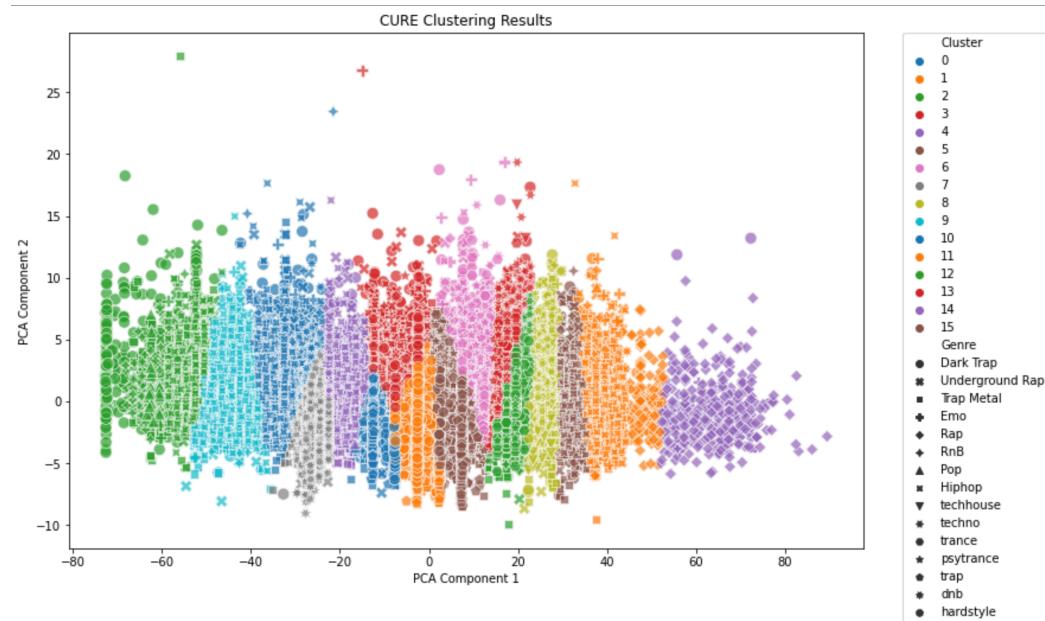
برای مثال با افزایش تعداد سمپل اولیه برای CURE accuracy به ۲۵۰، در حدود ۱ دقیقه زمان می‌برد.

```

Accuracy for BFR Clustering: 40.76350313201749
Accuracy for CURE Clustering: 42.59071031792932

```

نمونه‌ای از نمایش کلاسترها با کاهش بعد توسط PCA:
 (مربوط به حالتی است که هر دو الگوریتم از نظر زمان اجرا یکسان باشند.)



توضیح MapReduce-based K-means clustering

کد مربوط به این بخش پیاده‌سازی شده، ولی مشکلاتی دارد و به علت کمبود وقت تا زمان دಡلاین، موفق به رفع همهی مشکلات آن نشدم. در اینجا منطق کد و توابع مدنظر را توضیح می‌دهم.

این کد یک الگوریتم خوشبندی K-means را با استفاده از PySpark API پایتون برای Apache Spark توزیع شده را با استفاده از K-means پیاده‌سازی می‌کند. الگوریتم با راهاندازی یک `SparkContext` شروع می‌شود تا از قابلیت‌های محاسبات توزیع شدهی Spark استفاده کند. سپس یکتابع برای محاسبه فاصله اقلیدسی بین یک نقطه داده و یک مرکز ثقل تعریف می‌شود که برای اندازه‌گیری شباهت بین نقاط در فرآیند خوشبندی استفاده می‌شود. دوتابع اصلی، `map` و `reducer`، برای مدیریت مراحل `map` و `reduce` در پارادایم MapReduce پیاده‌سازی می‌شوند. تابع `map` هر نقطه داده را به نزدیک‌ترین مرکز ثقل اختصاص می‌دهد، در حالی که تابع `reducer` مرکز ثقل جدید را با میانگین‌گیری از نقاط اختصاص داده شده به هر خوش‌محاسبه می‌کند. در بخش اصلی کد، `centroid`ها اولیه به صورت تصادفی تولید می‌شوند. مجموعه داده‌ی مدنظر به یک مجموعه داده مقاوم توزیع شده (RDD) بارگذاری می‌شود که در آن هر خط به اجزای تشکیل‌دهنده خود تقسیم شده و ویژگی‌ها به آرایه‌های NumPy تبدیل می‌شوند. سپس الگوریتم به تعداد دفعات از پیش تعریف شده (`max_iterations`) تکرار می‌شود و در هر تکرار مراحل `map` و `reduce` انجام می‌شود. در مرحله `map`، نقاط داده به `centroid`ها نزدیک خود نگاشت می‌شوند و در مرحله `reduce`، نتایج جزئی برای محاسبه `centroid`ها جمع‌آوری می‌شوند. این فرآیند تکراری تا زمانی که `centroid`ها بین تکرارها به طور قابل توجهی تغییر نکنند یا حداقل تعداد تکرارها برسد ادامه می‌یابد.

در نهایت، `SparkContext` برای آزادسازی منابع متوقف می‌شود. رویکرد کلی تضمین می‌کند که خوشبندی K-means می‌تواند به صورت توزیع شده اجرا شود و از قابلیت‌های پردازش موازی Spark برای مدیریت مجموعه‌های داده بزرگ به طور کارآمد استفاده کند. بررسی همگرایی با استفاده از `np.allclose` تضمین می‌کند که الگوریتم در صورت ثبت `centroid`ها قبل از رسیدن به حداقل تعداد تکرارها زودتر متوقف می‌شود و کارایی محاسباتی را افزایش می‌دهد.

با توجه به عدم کارکرد این بخش از کد، من موفق به مقایسه نتایج با دو الگوریتم قبلی به صورت شهودی نشدم. با این حال با کمی سرج می‌توان درمورد مقایسه آن‌ها این طور گفت:

Accuracy

k-means با استفاده از MapReduce باید دقیق قابل مقایسه‌ای با CURE و BFR از نظر کیفیت خوشبندی ارائه دهد. تفاوت عمدتاً در نحوه مدیریت الگوریتم‌ها با مجموعه داده‌های بزرگ و تعاریف خوشبندی نهفته است.

BFR طوری طراحی شده است که با داده‌های با ابعاد بالا به خوبی کار می‌کند و مفروضاتی را ایجاد می‌کند که به خوبی با مجموعه داده‌های بزرگ مطابقت دارند، در حالی که CURE به دلیل رویکرد خوشبندی سلسله مراتبی خود، بهتر می‌تواند موارد پرت را مدیریت کند.

:Complexity

K-means: پیاده سازی MapReduce دارای پیچیدگی زمانی $O(tkn)$ است، که در آن t تعداد تکرارها، k تعداد خوشه ها و n تعداد نقاط داده است.

BFR: پیچیده تر به دلیل رسیدگی به آمار خلاصه و مراحل ادغام، مناسب برای مجموعه داده های بسیار بزرگ.
CURE: به دلیل رویکرد سلسله مراتبی آن، معمولاً از مرتبه $O(n^2 * \log n)$ است.