BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC PHENIKAA**

**KHOA CÔNG NGHỆ THÔNG TIN**

**------------oOo-----------**

**Ảnh có chứa Phông chữ, biểu tượng, Đồ họa, văn bản

Mô tả được tạo tự động**

**Đề tài:**

**“Clustering-Algorithm Spark using method KMeans”**

**Giảng viên: TS. Phạm Tiến Lâm, TS. Trịnh Thành**

**Nhóm sinh viên thực hiện:**

* **Nguyễn Thanh Quang – 21011417 – K15-CNTTVJ1**
* **Nguyễn Phát – 21011414 – K15-CNTTVJ1**

**Học phần: Tích hợp và phân tích dữ liệu lớn**

**Hà Nội, 05/2023**

|  |  |  |  |
| --- | --- | --- | --- |
| Họ và tên | MSSV | Lớp | Phân công |
| Nguyễn Phát | 21011414 | K15-CNTTVJ1 | 1) Giới thiệu tổng quan về báo cáo mục tiêu  2) Tiền xử lí và phân tích dữ liệu  3)Trực quan hóa dữ liệu  4) Tiến hành chạy và thu kết quả AHC |
| Nguyễn Thanh Quang | 21011417 | K15-CNTTVJ1 | 1) Apache Spark, K-means và lý do dùng Spark cho K-means  2) Triển khai thuật toán K-means áp dụng Spark  3) Xác định số tâm cụm tối ưu và tiến hành thực nghiệm  4) Kết quả thu được (K-means)  5) So sánh với kết quả của AHC  6) Slide |

Mục lục

**I. Giới thiệu tổng quan về báo cáo mục tiêu4**

**II. Tiền xử lí và phân tích dữ liệu4**

1. Dataset4

2. Tiền xử lí dữ liệu5

**III. Trực quan hóa dữ liệu7**

**IV. Apache Spark, K-means và lý do dùng Spark cho K-means9**

1. Apache Spark9

1.1. Giới thiệu Apache Spark9

1.2. Ưu điểm của Apache Spark9

1.2.1. Tốc độ9

1.2.2. Dễ sử dụng10

1.2.3. Tính tổng quát10

1.2.4. Tính linh hoạt10

1.3. Cách hoạt động của Spark11

2. K-means11

2.1. Giới thiệu K-means11

2.2. Phương pháp tính khoảng cách12

2.1. Cách hoạt động12

3. Lý do áp dụng Spark vào K-means14

**V. Triển khai thuật toán K-means áp dụng Spark14**

1. Ngôn ngữ lập trình14

2. Thư viện sử dụng14

2.1. PySpark14

2.2. Numpy15

2.3. Pandas15

2.4. Matplotlib15

3. Triển khai thuật toán15

3.1. Dữ liệu đầu vào16

3.1.1. Đọc dữ liệu16

3.1.2. Chuẩn hóa dữ liệu16

3.2. Khởi tạo môi trường Spark16

3.3. Hàm tính khoảng cách17

3.4. Chọn ngẫu nhiên tâm cụm18

3.5. Phân cụm các điểm dữ liệu19

3.6. Đặt lại tâm cụm và thực hiện phân cụm19

3.7. Hoàn thiện thuật toán K-means20

3.8. Phương pháp Elbow22

3.9. Class K-means23

**VI. Xác định tâm cụm tối ưu và tiến hành thực nghiệm28**

1. Tối ưu hóa số cụm28

2. Thực nghiệm29

**VII. Kết quả thu được29**

**VIII. So sánh với kết quả của AHC30**

**IX. Kết luận31**

**X. Tài liệu tham khảo31**

**XI. Link Dataset32**

**I. Giới thiệu tổng quan về báo cáo mục tiêu**

K-means là một thuật toán phân cụm không giám sát phổ biến, cho phép nhóm các điểm dữ liệu thành các cụm dựa trên sự tương đồng giữa chúng. Mục tiêu chính của báo cáo là trình bày quy trình thực hiện K-means Clustering và đánh giá hiệu suất và chất lượng của mô hình.

Phân cụm (clustering) là một phương pháp quan trọng trong khám phá dữ liệu, cho phép chúng ta khám phá các cấu trúc ẩn và nhóm các điểm dữ liệu có tính chất tương tự. K-means là một phương pháp phân cụm đơn giản và hiệu quả, phổ biến trong nhiều lĩnh vực như khoa học dữ liệu, thị trường học máy và khai phá dữ liệu. Báo cáo này sẽ giới thiệu về mô hình K-means Clustering và ứng dụng của nó trong xử lý dữ liệu lớn bằng Apache Spark.

Trong báo cáo này, chúng ta sẽ tìm hiểu cách áp dụng mô hình K-means Clustering vào dữ liệu phân cụm khách hàng sử dụng Apache Spark. Dữ liệu phân cụm này chứa thông tin về đặc điểc, vị trí của khách hàng, và chúng ta sẽ sử dụng K-means để phân cụm khách hàng thành các nhóm dựa trên các đặc điểm địa lí.

Bên cạnh đó, chúng ta cũng sẽ đánh giá hiệu suất và chất lượng của mô hình K-means thông qua các chỉ số đánh giá và sử dụng các công cụ phân tích kết quả. Điều này giúp chúng ta hiểu rõ hơn về khả năng của mô hình K-means trong việc phân cụm dữ liệu và ứng dụng của nó trong thực tế.

**II. Tiền xử lí và phân tích dữ liệu**

1. Dataset

Dữ liệu được sử dụng là Customer segmentation dataset:

Trong báo cáo này, chúng ta sẽ sử dụng tập dữ liệu về thông tin khách hàng để áp dụng mô hình K-means Clustering. Tập dữ liệu này chứa các thông tin liên quan đến khách hàng và sẽ giúp chúng ta tìm hiểu và phân cụm khách hàng dựa trên vị trí địa lí của họ và tìm ra vị trí đặt trung tâm hoặc chi nhánh công ty dịch vụ.

Các trường dữ liệu của khách hàng bao gồm các trường sau:

1. 'first\_name': Đây là trường chứa tên riêng (tên gọi) của một người hoặc khách hàng trong bộ dữ liệu. Ví dụ: "John", "Mary", "David".

2. 'last\_name': Đây là trường chứa họ (tên đệm) của một người hoặc khách hàng trong bộ dữ liệu. Ví dụ: "Smith", "Johnson", "Williams".

3. 'title': Đây là trường chứa chức danh hoặc vị trí công việc của một người. Ví dụ: "Manager", "Director", "Engineer".

4. 'gender': Đây là trường chứa giới tính của một người. Thông thường, giá trị có thể là "Nam", "Nữ", hoặc "Khác".

5. 'email': Đây là trường chứa địa chỉ email của một người hoặc khách hàng.

6. 'city': Đây là trường chứa tên thành phố hoặc địa điểm nơi một người hoặc khách hàng có thể sống hoặc làm việc.

7. 'country': Đây là trường chứa tên quốc gia hoặc địa điểm quốc gia mà một người hoặc khách hàng thuộc về.

8. 'latitude': Đây là trường chứa vĩ độ địa lý (tọa độ) của một địa điểm hoặc vị trí.

9. 'longitude': Đây là trường chứa kinh độ địa lý (tọa độ) của một địa điểm hoặc vị trí.

10. 'phone': Đây là trường chứa số điện thoại của một người hoặc khách hàng.

11. 'street\_number': Đây là trường chứa số nhà hoặc số kiểm soát của một địa chỉ đường.

12. 'street\_suffix': Đây là trường chứa phụ tố hoặc phần cuối của tên đường hoặc phố. Ví dụ: "Street", "Avenue", "Lane".

13. 'company\_name': Đây là trường chứa tên công ty hoặc tổ chức mà một người làm việc hoặc liên quan đến.

14. 'department': Đây là trường chứa tên phòng ban hoặc bộ phận trong một công ty hoặc tổ chức

15. 'job\_title': Tên công việc

16. 'ip\_address': địa chỉ ip

17. 'purchase\_amount': sức mua

Dữ liệu này sẽ cung cấp thông tin về các khách hàng, bao gồm địa chỉ, thông tin liên hệ, vị trí địa lý và các thuộc tính công việc. Sử dụng mô hình K-means Clustering trên tập dữ liệu này, chúng ta có thể tìm ra sự phân bổ của các nhóm khách hàng nhờ các đặc điểm dịa lí, giúp chúng ta hiểu rõ hơn về mật độ khách hàng trên toàn quốc.

2. Tiền xử lí dữ liệu

Dữ liệu được sử dụng là Customer Segmentation được tạo từ trang <https://www.mockaroo.com/>. Sau khi tải về thì bắt đầu gộp các trường thông tin để tạo 1 file csv hoàn chỉnh và tiến hành đọc file dữ liệu. Vì dữu liệu được tạo ra để thử nghiệm mô hình nên dữ liệu có thể sẽ khá sạch.

Các trường dữ liệu của khách hàng bao gồm các trường sau:

1. 'first\_name': Đây là trường chứa tên riêng (tên gọi) của một người hoặc khách hàng trong bộ dữ liệu. Ví dụ: "John", "Mary", "David".

2. 'last\_name': Đây là trường chứa họ (tên đệm) của một người hoặc khách hàng trong bộ dữ liệu. Ví dụ: "Smith", "Johnson", "Williams".

3. 'title': Đây là trường chứa chức danh hoặc vị trí công việc của một người. Ví dụ: "Manager", "Director", "Engineer".

4. 'gender': Đây là trường chứa giới tính của một người. Thông thường, giá trị có thể là "Nam", "Nữ", hoặc "Khác".

5. 'email': Đây là trường chứa địa chỉ email của một người hoặc khách hàng.

6. 'city': Đây là trường chứa tên thành phố hoặc địa điểm nơi một người hoặc khách hàng có thể sống hoặc làm việc.

7. 'country': Đây là trường chứa tên quốc gia hoặc địa điểm quốc gia mà một người hoặc khách hàng thuộc về.

8. 'latitude': Đây là trường chứa vĩ độ địa lý (tọa độ) của một địa điểm hoặc vị trí.

9. 'longitude': Đây là trường chứa kinh độ địa lý (tọa độ) của một địa điểm hoặc vị trí.

10. 'phone': Đây là trường chứa số điện thoại của một người hoặc khách hàng.

11. 'street\_number': Đây là trường chứa số nhà hoặc số kiểm soát của một địa chỉ đường.

12. 'street\_suffix': Đây là trường chứa phụ tố hoặc phần cuối của tên đường hoặc phố. Ví dụ: "Street", "Avenue", "Lane".

13. 'company\_name': Đây là trường chứa tên công ty hoặc tổ chức mà một người làm việc hoặc liên quan đến.

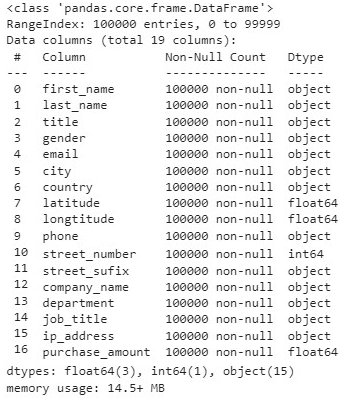
14. 'department': Đây là trường chứa tên phòng ban hoặc bộ phận trong một công ty hoặc tổ chức.

15. 'job\_title': Tên công việc

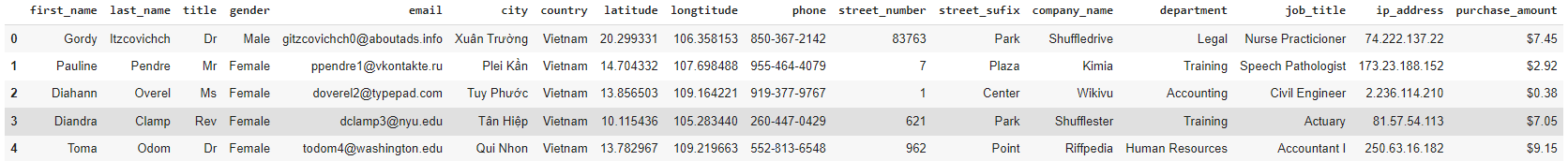
16. 'ip\_address': địa chỉ ip

17. 'purchase\_amount': sức mua

Ngoài ra, Dataset này có 100,000 mục nhập (entries) và có tổng cộng 14 cột (columns) được khởi tạo. và info của dataset như sau:



Năm dòng đầu:



Ở đây, chúng ta có thể thấy được là dữ liệu khá là sạch vì ở cột purchase\_amount có null nên chúng tôi sẽ xử lí trường thông tin này bằng cách đặt nó là 0 để tiến hành bước chuẩn hóa sau này.

Ngoài ra, chúng tôi phát hiện cột purchase\_amount đang ở định dạng là object nên đã chuyển về dạng float để có thể thể hiện sơ bộ về dữ liệu thu thập được.

Đầu tiên là thử chuyển cột dữ liệu đó về dạng ‘float’. Nhưng vì trong dữ liệu có chứa cả dạng số và dạng chữ nên chúng tôi đã tách các kí tự ra và đã chuyển tải được dữ liệu về dạng số.

**III. Trực quan hóa dữ liệu**

Hình ảnh trực quan về mật độ khách hàng.

A picture containing text, line

Description automatically generated

Nhìn vào biểu đồ chúng ta có thể thấy được sự phân bố chủ yếu ở các nơi như vùng đồng bằng sông Hồng và khu vực Nam Bộ. Chúng ta có thể thấy được việc đặt các trụ sở chính tập trung ở các vùng đậm màu sẽ cho chúng ta tiếp cận khách hàng một cách nhanh nhất và hiệu quả nhất.

Biểu đồ phân bố sức mua sắm cả nước.

A picture containing text, screenshot, diagram, plot

Description automatically generated

Ở biểu đồ này chúng ta có thể thấy được sức mua của thị trường mua chủ yếu tập trung ở 2 vùng trung tâm rõ rệt . Điều này cho chúng ta có thể thấy được là việc mở các trung tâm mua sắm cũng như mở các chi nhánh ở đây sẽ thu hút được nhiều du khách, tối ưu được lợi nhuận thu về.

**IV. Apache Spark, K-means và lý do dùng Spark cho K-means**

1. Apache Spark

1.1. Giới thiệu Apache Spark

Apache Spark là một data processing framework có thể xử lí những tác vụ trên một bộ dữ liệu khổng lồ một cách nhanh chóng và có thể phân phối các tác vụ xử lí dữ liệu trên nhiều máy tính, một mình hoặc song song với các công cụ điện toán phân tán khác. Nó mang hai yếu tố là chìa khóa cho thế giới của dữ liệu lớn và học máy, đó là sức mạnh tính toán khổng lồ để xử lí kho dữ liệu lớn và dễ sử dụng, tiện lợi của điện toán phân tán và xử lí dữ liệu lớn nhờ có một API.

Spark có thể phát triển theo nhiều cách khác nhau, cung cấp các liên kết cho các ngôn ngữ lập trình Java, Scala, Python và R, đồng thời hỗ trợ SQL, truyền dữ liệu, học máy và xử lí biểu đồ. Spark được sử dụng rộng rãi bởi các ngân hàng, công ty viễn thông, công ty trò chơi, chính phủ và tất cả các ông lớn như Apple, IBM, Meta và Microsoft.

1.2. Ưu điểm của Apache Spark

1.2.1. Tốc độ

Thời gian chạy của Spark nhanh hơn gấp 100 lần so với Hadoop MapReduce trên bộ nhớ hoặc 10 lần trên đĩa. Apache Spark có một công cụ thực thi DAG tiên tiến hỗ trợ luồng dữ liệu theo chu kỳ và tính toán trong bộ nhớ.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

1.2.2. Dễ sử dụng

Chúng ta có thể nhanh chóng viết một chương trình ứng dụng với Spark bằng Java, Scala, Python, R. Spark cung cấp hơn 80 toán tử cấp cao giúp dễ dàng xây dựng các ứng dụng song song. Và bạn có thể sử dụng nó một cách tương tác từ Scala, Python và R shell.

1.2.3. Tính tổng quát

Kết hợp SQL, streaming và phân tích phức tạp. Spark hỗ trợ một chồng thư viện bao gồm SQL và DataFrames, MLlib dành cho máy học, GraphX và Spark Streaming. Bạn có thể kết hợp các thư viện này liền mạch trong cùng một ứng dụng.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

1.2.4. Tính linh hoạt

Spark chạy trên Hadoop, Mesos, độc lập hoặc trên cloud. Nó có thể truy cập các nguồn dữ liệu đa dạng bao gồm HDFS, Cassandra, HBase và S3.

Ảnh có chứa văn bản, biểu tượng, Phông chữ, thiết kế đồ họa

Mô tả được tạo tự động

1.3. Cách hoạt động của Spark

Spark có một cơ sở mã nhỏ và hệ thống được chia thành nhiều lớp khác nhau. Mỗi lớp có một số trách nhiệm. Các lớp độc lập với nhau.

Lớp đầu tiên là trình thông dịch, Spark sử dụng trình thông dịch Scala, với một số sửa đổi. Khi bạn nhập mã của mình vào bảng điều khiển Spark (tạo RDD và áp dụng các toán tử), Spark sẽ tạo một biểu đồ toán tử. Khi người dùng thực hiện một hành động (collect), Biểu đồ sẽ được gửi tới bộ lập lịch biểu DAG. Bộ lập lịch DAG chia biểu đồ toán tử thành các giai đoạn (map vàreduce). Một giai đoạn bao gồm các tác vụ dựa trên các phân vùng của dữ liệu đầu vào. Trình lập lịch biểu DAG phối hợp các toán tử với nhau để tối ưu hóa biểu đồ. Ví dụ: Nhiều toán tử bản đồ có thể được lên lịch trong một giai đoạn duy nhất. Sự tối ưu hóa này là chìa khóa cho hiệu suất của Sparks. Kết quả cuối cùng của bộ lập lịch DAG là một tập hợp các giai đoạn. Các giai đoạn được chuyển đến TaskScheduler. TaskScheduler khởi chạy các tác vụ thông qua trình quản lý cụm. (Spark Standalone/Yarn/Mesos). TaskScheduler không biết về sự phụ thuộc giữa các giai đoạn.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

2. K-means

2.1. Giới thiệu K-means

K-means là một thuật thuật toán dùng trong khai phá dữ liệu với mục đích phân cụm, phân nhóm hoặc nhóm N đối tượng dựa trên các đặc trưng hoặc thuộc tính của chúng vào K nhóm (cụm) khác nhau. Nếu chúng ta ứng dụng thuật toán K-means vào một bộ dữ liệu gồm N đối tượng thì kết quả của thuật toán sẽ là K nhóm riêng biệt. Trong đó, mỗi cụm chứa các đối tượng tượng tương đồng với nhau. Lưu ý, K phải là số nguyên dương.

2.2. Phương pháp tính khoảng cách

Để xác định hai điểm dữ liệu có tương đồng với nhau không, K-means sẽ tính khoảng cách giữa chúng trong không gian dữ liệu để xác định độ tương đồng. Chúng ta sẽ có một số phương pháp tính khoảng cách sau.

* Khoảng cách Euclid:
* Khoảng cách Manhattan:
* Khoảng cách Cosine:

2.3. Cách hoạt động

Ý tưởng của thuật toán là việc phân chia dữ liệu thành K nhóm con mà trong mỗi nhóm các điểm dữ liệu có sự tương đồng với nhau. Để có thể thực hiện việc phân cụm cho bộ dữ liệu không có nhãn, Thuật toán K-means sẽ áp dụng cách tính khoảng cách giữa các điểm với tâm của mỗi cụm.

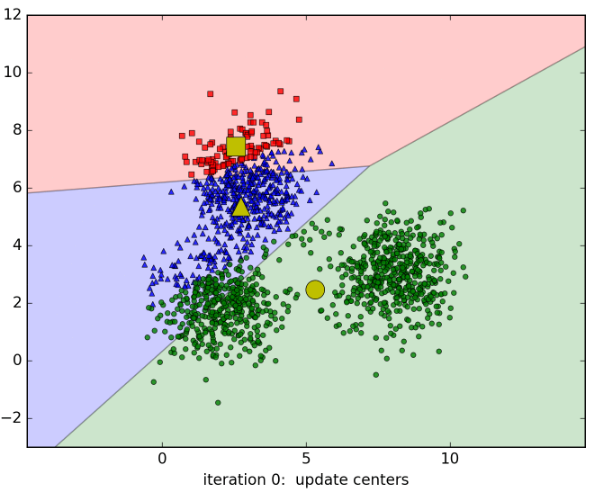
**Bước đầu tiên**, thuật toán K-means sẽ chọn ngẫu nhiên ra K điểm dữ liệu ban đầu làm tâm hay tâm cụm. Việc sử dụng thuật toán K-means để chia bao nhiêu tâm cụm phụ thuộc vào bài toán và mục đích sử dụng riêng biệt. Vậy nên, để sử dụng tối ưu thuật toán K-means thì việc xác định trước số cụm là cần thiết.

**Bước 2**, sau khi đã đẩy được dữ liệu và xác định số cụm cần phân chia, thuật toán sẽ tính toán khoảng cách của những tâm được chọn ngẫu nhiên trước đó với tất cả các điểm trong bộ dữ liệu. Theo đó, những điểm nào ở gần tâm cụm nào nhất sẽ được chia vào cụm chứa tâm đó.

Ảnh có chứa ảnh chụp màn hình, Nhiều màu sắc

Mô tả được tạo tự động

**Bước 3**, sau khi tính toán phân chia xong cụm từ ***Bước 2***, thuật toán K-means sẽ đi tính lại tâm cụm của các cụm tương ứng và đặt lại tâm cụm mới cho cụm tương ứng.



**Bước 4**, nếu như các cụm mới được sinh ra bởi không có thay đổi gì mới so với các cụm trước đó thì dừng thuật toán. Nếu không, thuật toán sẽ quay lại ***Bước 2***. Tóm lại, thuật toán sẽ lập đi lập lại ***Bước 2*** và ***Bước 3*** cho đến khi các cụm đã được phân chia rõ ràng và sẽ không có thay đổi gì mới với các cụm khi thực hiện lại hai bước trên.

Ảnh có chứa ảnh chụp màn hình, Nhiều màu sắc

Mô tả được tạo tự động

Thuật toán sẽ lặp lại cho đến khi các tâm cụm sẽ không còn thay đổi nào hoặc không còn trao đổi những điểm qua lại giữa các cụm qua mỗi lần lặp. Đến lúc này, chúng ta đã thu được K cụm mong muốn nhờ vào thuật toán K-means. Trên thực tế, việc cho lặp đi lặp lại cho đến khi tìm ra các cụm tối ưu là rất mất thời gian và có còn có thể là lặp vô hạn vì không có điểm dừng. Vì vậy, người ta thường sẽ chỉ lặp qua số vòng lặp đã chỉ định hoặc đã đặt được số lượng đối tượng đã đủ cho mỗi cụm.

3. Lý do áp dụng Spark vào K-means

Với đặc thù về ý tưởng thuật toán, K-means sẽ tính một lượng lớn khoảng cách của tất cả các điểm đến các tâm cụm kết hợp với tác vụ này lập lại nhiều lần. Điều này vô hình chung làm cho khối lượng tính toán của máy tính trở nên khổng lồ, nhất là khi dữ liệu thực sự quá lớn. Nó sẽ làm cho máy tính mất nhiều thời gian tính toán hơn, lâu và đem lại hiệu suất không cao.

Trong khi đó, Apache Spark lại có ưu thế lớn trong việc xử lí những kho dữ liệu khổng lồ nhờ có sự quản lí công việc thông minh. Như vậy, việc áp dụng Apache Spark vào thuật toán K-means sẽ giúp thuật toán xử lí các dữ liệu nhanh hơn đáng kể và có hiệu suất tốt. Việc chạy song song trên nhiều máy tính cũng sẽ khai thác được tối đa công suất cũng như hiệu suất tính toán để phục vụ cho thuật toán K-means.

**V. Triển khai thuật toán K-means áp dụng Spark**

Để có thể sử dụng Spark tiến hành chạy song song các máy trong mạng, chúng ta cần phải làm việc với RDD. Vì vậy, trong quá trình viết chương trình, chúng ta phải có những câu lệnh hỗ trợ RDD của PySpark.

1. Ngôn ngữ lập trình

Python là [ngôn ngữ lập trình máy tính bậc cao](https://glints.com/vn/blog/ngon-ngu-lap-trinh-bac-cao/) thường được sử dụng để xây dựng trang web và phần mềm, tự động hóa các tác vụ và tiến hành [phân tích dữ liệu](https://glints.com/vn/blog/ky-nang-phan-tich-du-lieu/). Python là ngôn ngữ có mục đích chung, nghĩa là nó có thể được sử dụng để tạo nhiều chương trình khác nhau và không chuyên biệt cho bất kỳ vấn đề cụ thể nào.

2. Thư viện sử dụng

2.1. PySpark

PySpark là sự kết hợp giữa Apache Spark và Python. Apache Spark là một framework tính toán cụm mã nguồn mở, được xây dựng xung quanh tốc độ, sự dễ sử dụng và phân tích luồng trong khi Python là một ngôn ngữ lập trình cấp cao, đa mục đích.

PySpark không chỉ cho phép bạn viết các ứng dụng Spark bằng cách sử dụng API Python mà còn cung cấp vỏ PySpark để phân tích dữ liệu của bạn một cách tương tác trong môi trường phân tán. PySpark hỗ trợ hầu hết các tính năng của Spark như Spark SOL, DataFrame, Streaming, Mllib và Spark Core.

2.2. Numpy

**Numpy (Numeric Python): là một thư viện toán học phổ biến và mạnh mẽ của Python. Cho phép làm việc hiệu quả với ma trận và mảng, đặc biệt là dữ liệu ma trận và mảng lớn với tốc độ xử lý nhanh hơn nhiều lần khi chỉ sử dụng “core Python” đơn thuần.**

**2.3. Pandas**

**Pandas là một thư viện Python cung cấp các cấu trúc dữ liệu nhanh, mạnh mẽ, linh hoạt và mang hàm ý. Tên thư viện được bắt nguồn từ panel data (bảng dữ liệu). Pandas được thiết kế để làm việc dễ dàng và trực quan với dữ liệu có cấu trúc (dạng bảng, đa chiều, có tiềm năng không đồng nhất) và dữ liệu chuỗi thời gian.**

**2.4. Matplotlib**

Để thực hiện các suy luận thống kê cần thiết, cần phải trực quan hóa dữ liệu của bạn và Matplotlib là một trong những giải pháp như vậy cho người dùng Python. Nó là một thư viện vẽ đồ thị rất mạnh mẽ hữu ích cho những người làm việc với Python và NumPy. Module được sử dụng nhiều nhất của Matplotib là Pyplot cung cấp giao diện như MATLAB nhưng thay vào đó, nó sử dụng Python và nó là nguồn mở.

3. Triển khai thuật toán

Việc đầu tiên cần làm trong mỗi một chương trình Python, chúng ta cần có các thư viện cần thiết để phân tích và phục vụ bài toán.

import pandas as pd

from pyspark.conf import SparkConf

from pyspark.context import SparkContext

import numpy as np

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

Các biến được định nghĩa trước trong bài gồm: số vòng lặp ‘iter’, ngưỡng điều kiện ‘threshold’, phương pháp tính khoảng cách ‘method’ và số tâm cụm ‘k’. Để tiện cho việc diễn giải, tôi sẽ định nghĩa các biến này này.

iter = 5

threshold = 1e-7

method = 'euclid'

k = 3

3.1. Dữ liệu đầu vào

3.1.1. Đọc dữ liệu

Để có thê phân tích, phân cụm được dữ liệu, chúng tôi cần phải lấy được dữ liệu vào chương trình để có thể phân tích. Để làm được như vậy, chúng tôi sẽ sử dụng thư viện Pandas của Python để đọc dữ liệu. Vì dữ liệu mà chúng tôi có được lưu dưới dạng một file csv.

df = pd.read\_csv("/content/drive/MyDrive/Customer\_Segmentation\_VN.csv")

Với câu lệnh này, chúng tôi sẽ sử dụng phương thức ‘read\_csv’ của pandas để có thể đọc file dạng csv. Trong đó, ‘df’ là tên biến sẽ lưu DataFrame khi dữ liệu dược đọc vào chương trình và tham số truyền vào ‘read\_csv’ là đường dẫn của file dữ liệu.

3.1.2. Chuẩn hóa dữ liệu

Như đã biết, K-means là một thuật toán dựa trên cách tính khoảng cách giữa hai điểm. Điều này có nghĩa K-means sẽ dựa vào các giá trị của các trường để tính khoảng cách giữa hai điểm bất kì. Tuy nhiên, các trường dữ liệu có thể sẽ có đơn vị khác nhau hay có giá trị rất lớn hoặc giá trị rất nhỏ, điều này ảnh hưởng rất lớn đến việc tính khoảng cách giữa hai điểm và ảnh hưởng đến hiệu quả của thuật toán K-means. Như vậy, bước chuẩn hóa dữ liệu rất quan trọng trong K-means. Nó giúp không có trường dữ liệu nào quá nổi trội so với phần còn lại, giúp thuật toán K-mean trở nên hiệu quả hơn.

scaler = MinMaxScaler()

data = scaler.fit\_transform(data)

Với phương thức ‘fit\_transform’ trong module ‘MinMaxScaler’ của sklearn, chúng ta sẽ chuẩn hóa các giá trị trong các trường dữ liệu về trong khoảng [0, 1] tương ứng với [giá trị nhỏ nhất, giá trị lớn nhất] với mỗi trường.

3.2. Khởi tạo môi trường Spark

Để làm việc và sử dụng Spark, chúng tôi cần khởi tạo moi trường Spark.

# Tạo môi trường Spark

conf = SparkConf().setAppName("K-means").setMaster("local[\*]")

sc = SparkContext.getOrCreate(conf)

rdd = sc.parallelize(data)

Đây là hai dòng code PySpark để tạo một đối tượng ‘SparkConf’ và một đối tượng ‘SparkContext’. Đối tượng ‘SparkConf’ được sử dụng để cấu hình các tham số cho chương trình Spark. Trong trường hợp này, tên ứng dụng được đặt là “K-means” và đặt URL master là “local[\*]” để chạy trên máy cục bộ với tất cả các lõi có sẵn.

Đối tượng ‘SparkContext’ là điểm vào chính cho các chức năng Spark. Nó được sử dụng để tạo RDD, tính toán phân tán và liên kết với một cụm Spark. Trong trường hợp này, phương thức ‘getOrCreate’ để lấy một đối tượng SparkContext hiện có hoặc tạo một đối tượng mới nếu chưa có.

Tạo một RDD từ một bộ dữ liệu có sẵn bằng cách sử dụng phương thức ‘parallelize’ của đối tượng ‘sc’. Phương thức ‘parallelize’ chuyển đổi một bộ dữ liệu có sẵn thành một RDD phân tán trên cụm Spark.

Trong trường hợp này, bộ dữ liệu data được chuyển đổi thành một RDD và gán nó cho biến rdd. Bây giờ, chúng tôi có thể thực hiện các phép biến đổi và hành động trên RDD này để xử lý dữ liệu.

Lưu ý, sau khi hoàn thành các tác vụ rồi, chúng ta nên lưu kết quả và giải phóng tài nguyên cho lần sử dụng tiếp theo

sc.stop()

Đây là một dòng code PySpark để dừng đối tượng ‘sc’ bằng cách sử dụng phương thức ‘stop’. Khi chúng tôi gọi phương thức ‘stop’, nó sẽ dừng ‘sc’ và giải phóng tài nguyên được sử dụng bởi nó.

Sau khi gọi phương thức ‘stop’,bạn sẽ không thể sử dụng đối tượng ‘sc’ để tạo RDD mới hoặc thực hiện các phép toán trên RDD hiện có. Nếu bạn muốn tiếp tục sử dụng Spark trong chương trình của mình, bạn sẽ cần tạo một đối tượng SparkContext mới.

3.3. Hàm tính khoảng cách

Như đã đề cập, K-means là thuật toán dựa trên cách tính khoảng cách giữa các điểm tới các tâm cụm. Như vậy chúng ta cần một hàm để thực thi tính khoảng cách.

# Hàm tính khoảng cách

def distance(a, b, method):

    if(method == 'euclid'):

      return np.sqrt(np.sum((a - b)\*\*2, axis=1))

    elif(method == 'manhattan'):

      return np.sum(np.abs(a - b), axis=1)

    elif(method == 'cosine'):

      return np.sum(np.dot(a, b)/(np.linalg.norm(a)\*np.linalg.norm(b)), axis=1)

Với hàm tính khoảng cách ‘distance’, chúng tôi muốn đầu vào sẽ là hai điểm bất kì (áp dụng cho thuật toán K-means sẽ là điểm dữ liệu và tâm cụm) và tên phương pháp tính khoảng cách (cụ thể có 3 phương pháp là ‘euclid’, ‘manhattan’ và ‘cosine’). Hàm này sẽ trả về khoảng cách giữa hai điểm ‘a’ và ‘b’ dựa theo phương pháp tính khoảng cách ‘method’.

3.4. Chọn ngẫu nhiên tâm cụm

Với thuật toán K-means, bước chọn tâm cụm ban đầu rất quan trọng và có thể ảnh hưởng lớn đến kết quả thuật toán.

# Chọn random k điểm dữ liệu từ tập dữ liệu

centroids = np.array(rdd.takeSample(False, k))

Chúng tả sẽ dử dụng phương thức ‘takeSample’ của ‘rdd’ để có thể chọn ngẫu nhiên k tâm cụm từ tập dữ liệu. Trong đoạn code trên, chúng tôi sẽ lấy ra k điểm dữ liệu bất kì làm tâm cụm ban đầu và sẽ đặt nó thành một mảng nhờ hàm ‘np.array’ để tiện hơn cho việc xử lí dữ liệu sau này.

Ngoài ra, việc chọn ngẫu nhiên có thể ảnh hưởng nhiều đến kết quả, ở bài này, chúng tôi cũng sẽ triển khai một chiến thuật chọn ngẫu nhiên tâm ban đầu K-mean++.

# Hàm áp dụng cho lambda RDD trả về khoảng cách nhỏ nhất từ một điểm đến các tâm cụm

def centroids\_to\_all(x, centroids):

  dt\_list = []

  for i in centroids:

    print(i)

    dt\_list.append(distance(np.array([x]), np.array([i]), 'euclid'))

  return np.min(dt\_list)

# Hàm hiết lập tâm cụm ban đầu nhờ K-means++

def initialize(rdd, k):

  centroids = np.array(rdd.takeSample(False, 1))

  for i in range(k - 1):

  idx = np.argmax(rdd.map(

lambda x: centroids\_to\_all(x, centroids)

).collect()

)

    centroids = np.vstack((centroids, rdd.collect()[idx]))

  return centroids

3.5. Phân cụm các điểm dữ liệu

Bước tiếp theo của thuật toán K-means là phân cụm các dữ liệu về các cụm có tâm tương ứng. Chúng tôi sẽ thực hiện việc này với phương thức ‘map’ của ‘rdd’ mà chúng tôi đã định nghĩa trước đó.

# Hàm áp dụng cho lambda RDD trả về [tâm cụm, khoảng cách] của một điểm đến tâm cụm gần nhất

def function(x, centroids, method):

    dt = distance(x, centroids, method)

    idx = np.argmin(dt)

    return [idx, dt[idx]]

# Hàm phân cụm không gian giữ liệu

def assignment\_point(rdd, centroids, method):

    return rdd.map(lambda x: [function(x, centroids, method), x])

# Phân cụm tập dữ liệu

clustered = assignment\_point(rdd, centroids, method)

# Tổng các khoảng cách của mỗi đến đến cụm gần nhất tương ứng

total\_err = np.sum(np.array(clustered.map(lambda x: x[0][1]).collect()))

Như vậy, chúng tôi đã tạo ra hai hàm để phục vụ cho việc phân cụm các điểm. Với hàm ‘function’ hàm áp dụng lên một RDD sử dụng để trả về tâm cụm (được chuẩn hóa bằng category là các số nguyên) và khoảng cách của điểm dữ liệu tới tâm cụm gần nhất. Hàm ‘assigment\_point’ sẽ có đầu vào là RDD chứa dữ liệu các điểm và trả về một RDD mới chứa các dữ liệu về tâm cụm, khoảng cách và giá trị các trưởng của các điểm dữ liệu.

Biến ‘clustered’ sẽ dùng lưu RDD có chứa các thông tin là tâm cụm, khoảng cách ứng mỗi diểm và giá trị các trường của mỗi điểm dữ liệu. Trong khi đó, ‘total\_err’ sẽ lưu lại mảng chứa tổng khoảng cách của tất cả các điểm dữ liệu.

3.6. Đặt lại tâm cụm và thực hiện phân cụm

Sau khi phân cụm các điểm dữ liệu trong lần đầu tiên, chúng tôi cần đặt lại tâm cụm cho mỗi cụm để tiến hành tối ưu hóa các cụm trong tập dữ liệu.

# Tính lại tâm cụm

centroids=np.array(clustered.map(

lambda x: (x[0][0], x[1])).groupByKey().mapValues(list).map(

lambda x: (np.mean(x[1], axis=0))

).collect()

)

Trong dòng này, chúng tôi sẽ nhóm các điểm dữ liệu cùng một cụm chứa trong RDD thành một list nhờ có ‘.groupByKey ().mapValues (list)’. Sau đó, chúng tôi sẽ áp dụng lambda ‘np.mean (x[1], axis=0)’ để tính lại tâm cụm bằng cách lấy trung bình các trường dữ liệu trong mỗi cụm và dùng ‘.collect ()’ để lấy ra các tâm cụm mới rồi lưu dưới dạng mảng.

Như vậy, chúng tôi đã có cách để tính lại tâm cụm, việc của chúng tôi bây giờ là tiếp tục phân cụm cho đến khi thỏa mãn điều kiện nào đó hoặc đã đủ số vòng lặp.

pre\_total\_err = total\_err

for i in range(iter):

     print(f"`iter: {i}`")

     # Tính lại tâm cụm

     centroids=np.array(clustered.map(

lambda x:(x[0][0],x[1])).groupByKey().mapValues(list).map(

lambda x: (np.mean(x[1],axis=0))

).collect()

)

     # Phân cụm lại tập giữ liệu

     clustered = assignment\_point(rdd, centroids, method)

     total\_err = np.sum(clustered.map(lambda x: x[0][1]).collect())

     # Kiểm tra điều kiện dừng

     if(pre\_total\_err - total\_err < threshold or i == iter - 1):

       clustered = clustered.map(lambda x: (x[0][0], x[1])).collect()

       sc.stop()

       print(f"[err: {total\_err}]")

       break

     pre\_total\_err = total\_err

     print(f"[err: {total\_err}]")

Trong đoạn code này, chúng tôi sẽ dùng vòng lặp để lặp đi lặp lại các bước ‘chọn lại tâm cụm’ => ‘phân cụm’. Với các biến ‘iter’ và ‘threshold’ đã tạo thành các điều kiện dừng của thuật toán K-means để tiết kiệm thời gian. Trong đó, ‘iter’ sẽ quy định số lần lặp của thuật toán và ‘threshold’ là ngưỡng điều kiện cho biết thuật toán khi nào dừng (tổng khoảng cách ở vòng lặp trước – tổng khoảng cách của vòng lặp này < ‘threshold’ thì dừng lại). Bên cạnh đó khi kết thúc vòng lặp, chúng tôi sẽ lưu kết quả cuối cùng vào biến ‘clustered’ bằng phương thức ‘collect’ của RDD ‘clustered’ và đóng ‘sc’ trả lại tài nguyên cho máy tính.

3.7. Hoàn thiện thuật toán K-means

Như vậy, chúng tôi đã trình bày tất cả những bước cần làm để hoàn thiện triển khai K-means. Bây giờ, chúng tôi cần lắp ghép chúng lại thành một hàm để thuận tiện cho lần sử dụng sau.

def kmeans(data, k = 3, method = 'euclid', iter = 100, threshold = 1e-7, mode = 'k-means++'):

  # Tạo môi trường Spark

  conf = SparkConf().setAppName("K-means").setMaster("local[\*]")

  sc = SparkContext.getOrCreate(conf)

  rdd = sc.parallelize(data)

  # Chiến thuật chọn tâm cụm ban đầu

  if(mode == 'k-means++'):

    centroids = initialize(rdd, k)

  else:

    centroids = np.array(rdd.takeSample(False, k))

  # Phân cụm tập dữ liệu

  clustered = assignment\_point(rdd, centroids, method)

  # Tổng các khoảng cách của mỗi đến đến cụm gần nhất tương ứng

  total\_err = np.sum(np.array(clustered.map(

lambda x: x[0][1]

).collect())

)

  pre\_total\_err = total\_err

  for i in range(iter):

      print(f"`iter: {i}`")

      # Tính lại tâm cụm

      centroids=np.array(clustered.map(

lambda x: (x[0][0], x[1])).groupByKey().mapValues(list).map(

lambda x: (np.mean(x[1], axis=0))

).collect()

)

      # Phân cụm lại tập dữ liệu

      clustered = assignment\_point(rdd, centroids, method)

      total\_err = np.sum(clustered.map(lambda x: x[0][1]).collect())

      # Kiểm tra điều kiện dừng

      if(pre\_total\_err - total\_err < threshold or i == iter - 1):

        clustered = clustered.map(lambda x: (x[0][0], x[1])).collect()

        sc.stop()

        print(f"[err: {total\_err}]")

        break

      pre\_total\_err = total\_err

      print(f"[err: {total\_err}]")

  return clustered, centroids, total\_err

Với hàm kmeans, chúng tôi đã tổng hợp lại tất cả các bước để cấu thành thuật toán K-means. Hàm với đầu vào là bộ dữ liệu, số tâm cụm, một số lựa chọn khác và đầu ra sẽ là mảng chứa các điểm dữ liệu và các cụm tương ứng với mỗi điểm, danh sách các tâm cụm và tổng khoảng cách của các điểm dữ liệu tới các tâm cụm của chúng.

3.8. Phương pháp Elbow

Thuật toán K-means là thuật toán cần biết trước số tâm cụm để phân chia không gian dữ liệu vào các cụm tương ứng. Tuy nhiên, việc xác định trước số tâm cụm trong nhiều bài toán cũng không phải là điều dễ dàng. Vì vậy, để tìm ra số cụm tối ưu cho bài toán, tôi cần có một phương pháp để đánh giá xem bao nhiêu tâm cụm là tối ưu với bài toán.

Ở trong bài này, tôi sẽ sử dụng phương pháp đánh giá elbow để đánh giá và tìm ra số tâm cụm tối ưu. Elbow đơn giản là chúng ta sẽ lần lượt thử với nhiều số tâm cụm khác nhau và tìm ra điểm tối ưu nhất nhờ vào việc so sánh tổng khoảng cách của tất cả các điểm giữ liệu với tâm cụm tương ứng.

def elbow(data, cluster\_range, iter = 20, method = 'euclid', mode='k-means++'):

  # Tạo môi trường Spark

  conf = SparkConf().setAppName("K-means").setMaster("local[\*]")

  sc = SparkContext.getOrCreate(conf)

  rdd = sc.parallelize(data)

  # Mảng lưu tổng khoảng cách các lần thử

  err\_summary = []

  # Chiến thuật chọn tâm cụm ban đầu

  for i in range(cluster\_range):

    print(f"`{i+1} cluster`")

    if(mode == 'k-means++'):

      centroids = initialize(rdd, i+1)

    else:

      centroids = np.array(rdd.takeSample(False, i+1))

    # Phân cụm tập dữ liệu

    clustered = assignment\_point(rdd, centroids, method)

    # Tổng các khoảng cách của mỗi đến đến cụm gần nhất tương ứng

    total\_err = np.sum(np.array(clustered.map(

lambda x: x[0][1]).collect())

)

    pre\_total\_err = total\_err

    for k in range(iter):

      # Tính lại tâm cụm

      centroids = np.array(clustered.map(

lambda x: (x[0][0], x[1])).groupByKey().mapValues(list).map(

lambda x: (np.mean(x[1], axis=0))

).collect()

)

      # Phân cụm lại tập dữ liệu

      clustered = assignment\_point(rdd, centroids, method)

      total\_err = np.sum(clustered.map(lambda x: x[0][1]).collect())

    err\_summary.append(total\_err)

    print(f"[err: {total\_err}]")

  del(centroids)

  sc.stop()

  # Vẽ biểu đồ đánh giá

  plt.scatter(np.arange(1, cluster\_range + 1, 1), err\_summary)

  plt.plot(np.arange(1, cluster\_range + 1, 1), err\_summary)

  plt.grid()

  plt.show()

3.9. Class K-means

Để tiện lợi và linh hoạt khi sử dụng, chúng tôi chúng tôi sẽ triển khi code K-means bằng một class để có thể tiện hơn trong việc sử dụng và điều chỉnh các tham số của thuật toán và những tính năng khác.

from pyspark.conf import SparkConf

from pyspark.context import SparkContext

import numpy as np

import matplotlib.pyplot as plt

class kmeans():

  # Hàm khởi tạo

  def \_\_init\_\_(self, method = "euclid", iter = 100, threshold = 1e-7, mode = 'k-means++'):

    if method in ['euclid',  'manhattan', 'cosine']:

      self.method = method

    else:

      raise ValueError("Method should be 'euclid', 'manhattan', 'cosine'.")

    if isinstance(iter, int) and iter > 0:

      self.iter = iter

    else:

      raise ValueError("Iter should be an interger great than 0.")

    if isinstance(threshold, str):

      raise ValueError("Threshold should be a float.")

    else:

      self.threshold = threshold

    if mode in ['k-means++','k-means']:

      self.mode = mode

    else:

      raise ValueError("Two mode 'k-means' and 'k-means++'.")

    pass

  # Hàm thiết lập phương pháp tính khoảng cách

  def set\_method(self, method):

    if method in ['euclid',  'manhattan', 'cosine']:

      self.method = method

    else:

      raise ValueError("Method should be 'euclid', 'manhattan', 'cosine'.")

  # Hàm thiết lập số lần lặp tối đa

  def max\_iter(self, iter):

    if isinstance(iter, int) and iter > 0:

      self.iter = iter

    else:

      raise ValueError("Iter should be an interger great than 0.")

  # Hàm thiết lập ngưỡng điều kiện dừng

  def set\_threshold(self, threshold):

    if isinstance(threshold, str):

      raise ValueError("Threshold should be a float.")

    else:

      self.threshold = threshold

  # Hàm thiết lập chiến thuật chọn tâm cụm ban đầu

  def set\_mode(self, mode):

    if mode in ['k-means++','k-means']:

      self.mode = mode

    else:

      raise ValueError("Two mode 'k-means' and 'k-means++'.")

  # Hàm xác định phương pháp chuẩn hóa

  def scalered(self, method):

    if method == 'minmax':

      return MinMaxScaler()

    else:

      return StandardScaler()

  # Hàm khôi phục dữ liệu đã chuẩn hóa

  def inverse\_transform(self):

    conf = SparkConf().setAppName("K-means").setMaster("local[\*]")

    sc = SparkContext.getOrCreate(conf)

    self.clustered = sc.parallelize(self.clustered).map(

lambda x: [x[0], self.scaler.inverse\_transform(np.array([x[1]]))]

).collect()

    self.centroids = self.scaler.inverse\_transform(self.centroids)

    sc.stop()

  # Hàm tính khoảng cách

  def distance(self, a, b):

    if(self.method == 'euclid'):

      print(np.argmin(np.sqrt(np.sum((a - b)\*\*2, axis=1))))

      return np.sqrt(np.sum((a - b)\*\*2, axis=1))

    elif(self.method == 'manhattan'):

      return np.sum(np.abs(a - b), axis=1)

    else:

      return np.sum(np.dot(a, b)/(np.linalg.norm(a)\*np.linalg.norm(b)), axis=1)

  # Hàm áp dụng cho lambda RDD trả về [tâm cụm, khoảng cách] của một điểm đến tâm cụm gần nhất

  def function(self, x):

    dt = self.distance(x, self.centroids)

    return [np.argmin(dt), np.min(dt)]

  # Hàm phân cụm không gian dữ liệu

  def assignment\_point(self, rdd):

    return rdd.map(lambda x: [self.function(x), x])

  # Hàm áp dụng cho lambda RDD trả về khoảng cách nhỏ nhất từ một diểm đến các tâm cụm

  def centroids\_to\_all(self, x):

    dt\_list = []

    for i in self.centroids:

      dt\_list.append(self.distance(np.array([x]), np.array([i])))

    return np.min(dt\_list)

  # Hàm hiết lập tâm cụm ban đầu nhờ K-means++

  def initialize(self, rdd):

    self.centroids = np.array(rdd.takeSample(False, 1))

    for i in range(self.cluster\_num - 1):

      idx = np.argmax(rdd.map(

lambda x: self.centroids\_to\_all(x)

).collect()

)

      self.centroids = np.vstack((self.centroids, rdd.collect()[idx]))

  # Hàm tối ưu hóa mô hình

  def fit(self, data, k, scale = False):

    # Chọn phương pháp chuẩn hóa dữ liệu

    if scale:

      method = input('Scale method: ')

      if method in ['minmax', 'std']:

        self.scaler = self.scalered(method)

        data = self.scaler.fit\_transform(data)

      else:

        raise ValueError("Two mode 'minmax' and 'std'.")

    self.cluster\_num = k

    self.clusters = np.arange(k)

    # Tạo môi trường Spark

    conf = SparkConf().setAppName("K-means").setMaster("local[\*]")

    sc = SparkContext.getOrCreate(conf)

    rdd = sc.parallelize(data)

    # Chiến thuật chọn tâm cụm ban đầu

    if(self.mode == 'k-means++'):

      self.initialize(rdd)

    else:

      self.centroids = np.array(rdd.takeSample(False,self.cluster\_num))

    # Phân cụm tập dữ liệu

    clustered = self.assignment\_point(rdd)

    total\_err = np.sum(np.array(clustered.map(

lambda x: x[0][1]

).collect()

)

)

    self.total\_err = total\_err

    for i in range(self.iter):

      print(f"`iter: {i}`")

      # Tính lại tâm cụm

      self.centroids = np.array(clustered.map(

lambda x: (x[0][0], x[1])).groupByKey().mapValues(list).map(

lambda x: (np.mean(x[1], axis=0)

)

).collect())

      # Phân cụm lại tập dữ liệu

      clustered = self.assignment\_point(rdd)

      total\_err = np.sum(clustered.map(lambda x: x[0][1]).collect())

      # Kiểm tra điều kiện dừng

      if(self.total\_err-total\_err<self.threshold or i==self.iter-1):

        self.total\_err = total\_err

        self.clustered = clustered.map(

lambda x: (x[0][0], x[1])

).collect()

        sc.stop()

        print(f"[err: {total\_err}]")

        break

      self.total\_err = total\_err

      print(f"[err: {total\_err}]")

  # Hàm dự đoán điểm dữ liệu mới

    def predict(self, data, scale = False):

    if scale:

      data = self.scaler.fit\_transform(data)

    conf = SparkConf().setAppName("K-means").setMaster("local[\*]")

    sc = SparkContext.getOrCreate(conf)

    result = sc.parallelize(data).map(

lambda x: self.function(x)).map(

lambda x: [self.clusters[x[0]], x[1]]

).collect()

    sc.stop()

    return result

  # Hàm thay đổi nhãn dữ liệu

  def transform(self, label):

    if(len(label) != self.cluster\_num):

      raise ValueError(f"Model has {self.cluster\_num} clusters!")

    conf = SparkConf().setAppName("K-means").setMaster("local[\*]")

    sc = SparkContext.getOrCreate(conf)

    self.clusters = label

    self.clustered = sc.parallelize(self.clustered).map(

lambda x: [self.clusters[x[0]], x[1]]

).collect()

    sc.stop()

  # Hàm đánh giá elbow

  def elbow(self, data, cluster\_range, iter = 20 , scale = False):

# Chọn phương pháp chuẩn hóa dữ liệu

    if scale:

      method = input('Scale method: ')

    if method in ['minmax', 'std']:

      self.scaler = self.scalered(method)

      data = self.scaler.fit\_transform(data)

    else:

      raise ValueError("Two mode 'minmax' and 'std'.")

    # Tạo môi trường Spark

    conf = SparkConf().setAppName("K-means").setMaster("local[\*]")

    sc = SparkContext.getOrCreate(conf)

    rdd = sc.parallelize(data)

    # Mảng lưu tổng khoảng cách các lần thử

    err\_summary = []

    for i in range(cluster\_range):

      print(f"`{i+1} cluster`")

      # Chiến thuật chọn tâm cụm ban đầu

      self.cluster\_num = i + 1

      if(self.mode == 'k-means++'):

        self.initialize(rdd)

      else:

        self.centroids = np.array(rdd.takeSample(False, self.cluster\_num))

      # Phân cụm tập dữ liệu

      clustered = self.assignment\_point(rdd)

      total\_err = np.sum(np.array(clustered.map(

lambda x: x[0][1]

).collect()

)

)

      pre\_total\_err = total\_err

      for k in range(iter):

        # Tính lại tâm cụm

        self.centroids = np.array(clustered.map(

lambda x: (x[0][0], x[1])).groupByKey().mapValues(list).map(

lambda x: (np.mean(x[1], axis=0))

).collect()

)

        # Phân cụm lại dữ liệu

        clustered = self.assignment\_point(rdd)

        total\_err = np.sum(clustered.map(lambda x: x[0][1]).collect())

if total\_err - pre\_total\_err < self.threshold:

          break

        pre\_total\_err = total\_err

      err\_summary.append(total\_err)

      print(f"[err: {total\_err}]")

    del(self.centroids)

    del(self.cluster\_num)

    sc.stop()

    # Vẽ biểu đồ đánh giá

    plt.scatter(np.arange(1, cluster\_range + 1, 1), err\_summary)

    plt.plot(np.arange(1, cluster\_range + 1, 1), err\_summary)

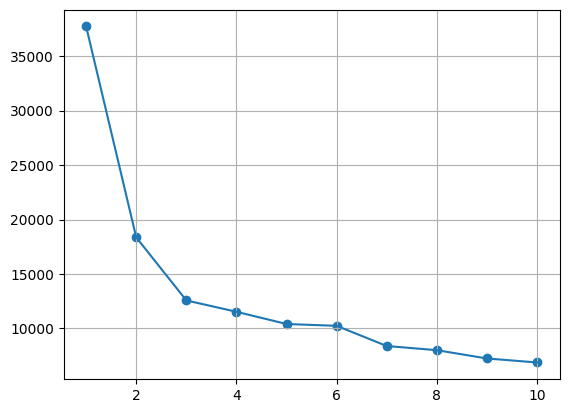
    plt.grid()

    plt.show()

**VI. Xác định tâm cụm tối ưu và tiến hành thực nghiệm**

1. Tối ưu hóa số cụm

Như đã đề cập, bài này chúng tôi sẽ sử dụng phương pháp đánh giá elbow để tìm ra số tâm cụm tốt nhất. Với phương pháp elbow, những điểm “khuỷu” sẽ là điểm mà số tâm cụm ở đó có thể tối ưu so với việc lấy số tâm cụm còn lại. Sau khi đã thực thực đánh giá trên tập dữ liệu của chúng tôi, chúng tôi thu được kết quả sau.



Nhìn vào biểu đồ đánh giá, ta thấy những điểm “khuỷu” đó là 3, 5 và 7. Như vậy, rất có thể và tùy vào quy mô, số tâm cụm trong tập dữ liệu này sẽ tối ưu khi chia làm 3, 5 và 7 cum.

2. Thực nghiệm

Sau khi đã biết được số tâm cụm mục tiêu, chúng tôi sẽ bắt đầu tối ưu hóa mô hình K-means trên tập dữ liệu.

model = kmeans()

model.fit(data, 3, scale = True)

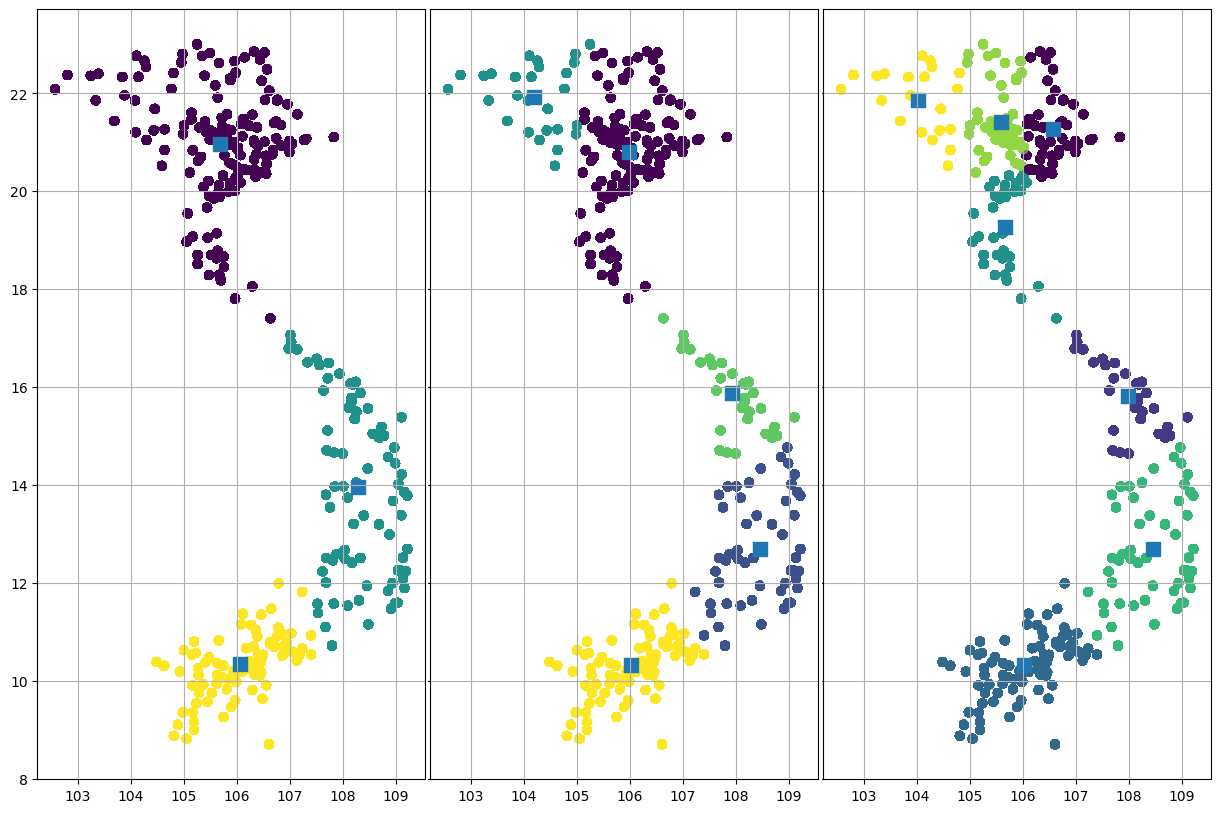
model1 = kmeans()

model1.fit(data, 5, True)

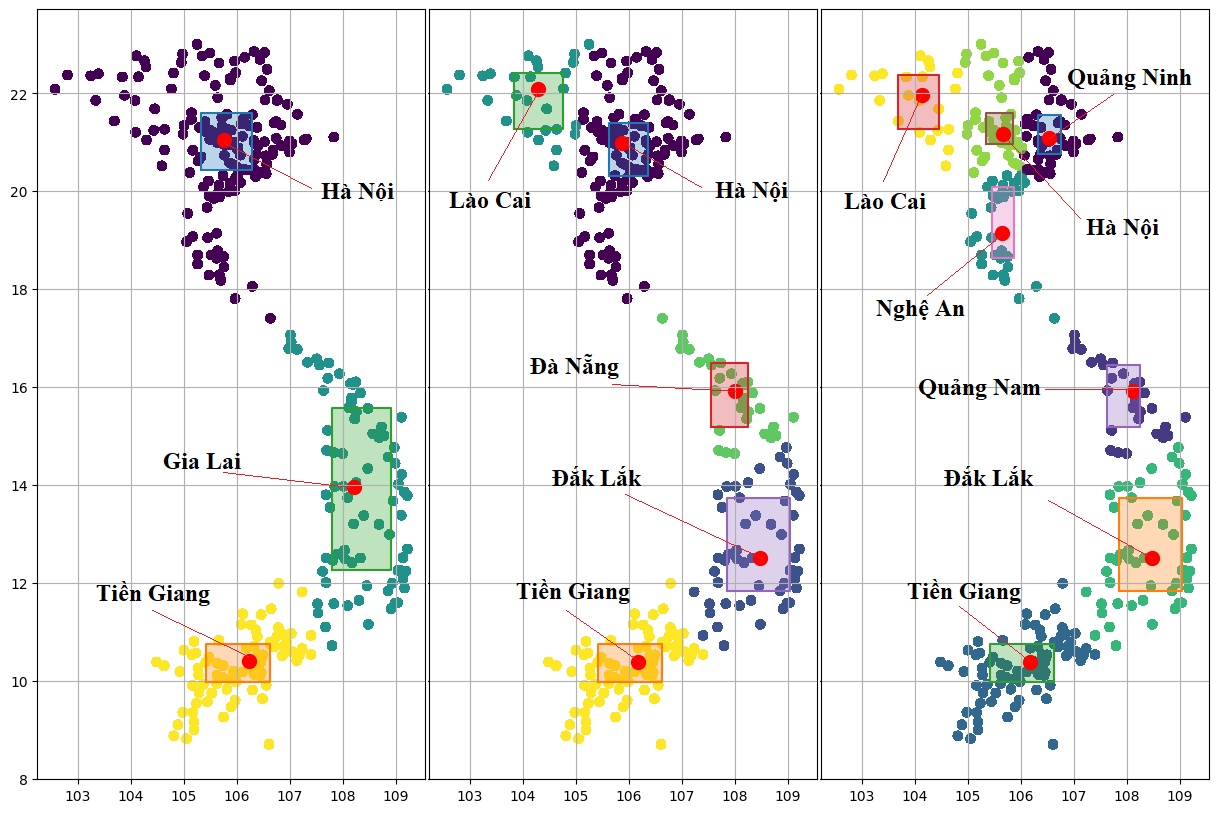
model2 = kmeans()

model2.fit(data, 7, True)

**VII. Kết quả thu được**



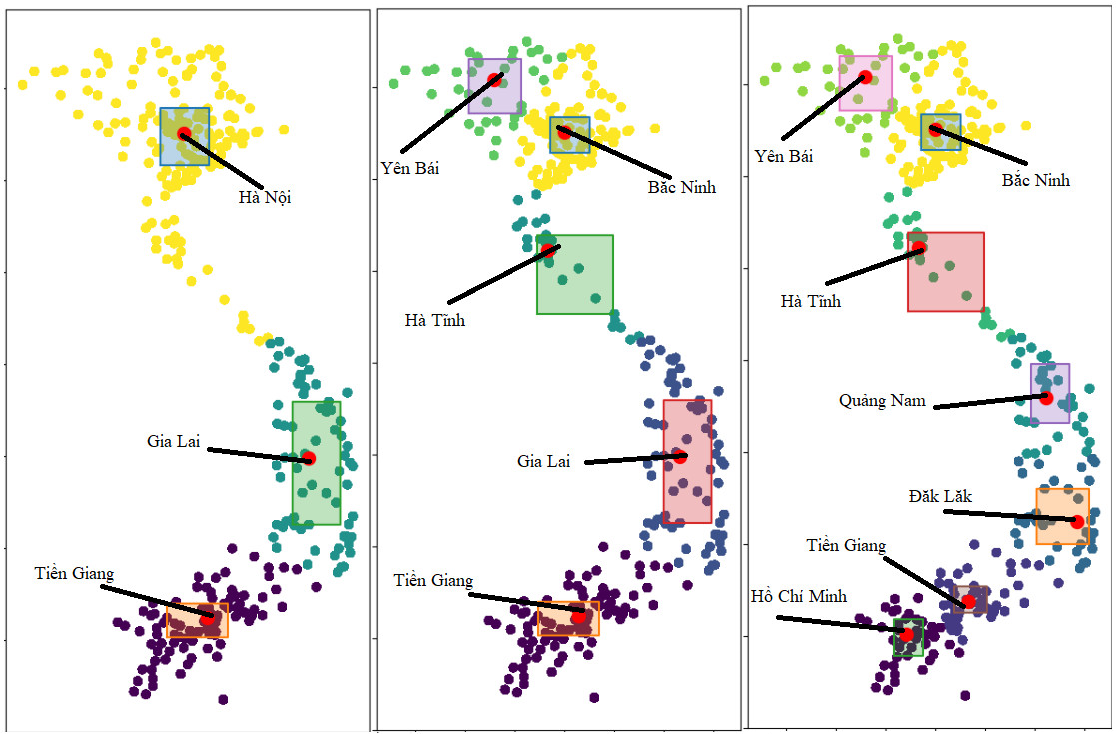
Nhìn vào ba kết quả, ta thấy thuật toán K-means phân chia thị trường Việt Nam khá tốt. Nó đã phân chia thị trường thành các cụm, trong đó mỗi cụm khá đều nhau. Như vậy, chúng tôi sẽ sử dụng tứ phân vị dựa trên vĩ độ và kinh độ để xác định vùng trọng yếu có thể đáp ứng việc đặt trung tâm, chi nhánh ở những cụm sao cho tốt nhất. Chúng tôi có kết quả như sau.



Sau khi biểu thị kết quả chi tiết hơn ta có thể thấy, Hà Nội và Tiền Giang là hai nơi trọng điểm và quan trọng nếu muốn đặt trung tâm hoặc chi nhánh. Các vùng còn lại, ta có thể thấy có những vùng có phân bố khách hàng thưa thớt và khó cho việc tất cả khách hàng trong cụm có thể tiếp cận. Những tỉnh được đánh dấu hầu hết là ở nơi mật độ khách hàng cao nhất trong cụm có mật độ dày đặc. Tuy nhiên, những cụm khách hàng thưa thớt thì phương pháp của chúng thôi chỉ đơn thuần là ở trung tâm của cụm.

**VIII. So sánh với kết quả của AHC**

Với tập dữ liệu hiện có, chúng tôi sẽ phân tích cụm trên một thuật toán phân cụm khác là AHC. AHC là một thuật toán phân cụm theo đồ thị, có nghĩa là nếu coi các điểm dữ liệu là những điểm của đồ thi, thì tập dữ liệu sẽ bao gồm nhiều đồ thị con hợp thành. Chúng tôi được kết quả như sau



So với kết quả của K-means. AHC lại làm tốt hơn ở điểm bắt được nhiều khu vực có mật độ khách hàng cao với phương pháp tứ phân vị của chúng tôi. Tuy nhiên, với phương pháp tứ phân vị còn nhiều điểm yếu khi mà những cụm thưa vẫn khó khăn trong việc chọn điểm đặt trung tâm/chi nhánh tốt nhất mà chỉ đơn thuần là tìm được trung tâm của cụm chứa nó.

**IX. Kết luận**

Như vậy, với việc có thể xử lí nhanh với bộ dữ khổng lồ, Spark là một công cụ hữu dụng và tiện lợi, đem lại hiệu suất cao và đẩy mạnh hiệu suất xử lí của máy tính. Spark có thể coi là bước đệm vững chắc cho nền tảng cho công nghệ thông tin sau này, nhất là đối với BigData và Machine learning cũng như trí tuệ nhân tạo.

Thuật toán K-means là thuật toán phân cụm với khối lượng tính toán khổng lồ, nhất là đối với những tập dữ liệu rất lớn. Vì thế, K-means sẽ có thể tốn rất nhiều thời gian để tính toán và đưa ra kết quả tùy vào kích thước của bộ dữ liệu. Việc áp dụng Spark vào K-means đã làm giảm đáng kể thời gian chạy và tính toán của thuật toán K-means nhờ vào những kĩ thuật và điểm mạnh của nó. Từ đó mà thuật toán cải thiện đáng kể hiệu suất xử lí, chúng ta cũng tiết kiệm được rất nhiều thời gian từ việc này.

**X. Tài liệu tham khảo**

# Feng, Wenqiang. "Learning Apache Spark with Python." 2019.

# Junjie Wu. *Advances in K-means Clustering: A Data Mining Thinking.* Springer Science & Business Media, 2012.

Mahmoud Parsian. *Data Algorithms: Recipes for Scaling Up with Hadoop and Spark.* O'Reilly Media, Inc., 2015.

*Naftali Harris: Visualizing K-Means Clustering.* January 19, 2014. https://www.naftaliharris.com/blog/visualizing-k-means clustering/.

*DOMINO: Getting started with k-means clustering in Python.* April 1, 2022. https://www.dominodatalab.com/blog/getting-started-with-k-means-clustering-in-python

**XI. Link Dataset**

link: https://l.facebook.com/l.php?u=https%3A%2F%2Fdrive.google.com%2Ffile%2Fd%2F1\_X-Qr1\_NIPR\_0wSRfBdQcMPQEQs4kQlR%2Fview%3Fusp%3Dsharing%26fbclid%3DIwAR3uqsqiEOY0UElng4klSRJMJfxyya\_psEXIwG4viAK17\_\_k0bFdGM5Zl9s&h=AT01Ej4Ka8TdJA0dAeKbQNYYM0\_W2rEFhb1G4DGDof7d8z2VjctnA71LUS6L4t4xuDaD98zNOoWS4XtJUVvduNQVcGOD-p0Tw1NWiZZ\_ls930\_SOZ5gQIPLWYhewkmtE2VBw0A