Spending habits of people of different ages using K-mean clustering

Rui Chang
School of Software Engineering
South China University of Technology
Guangzhou, Guangdong, P.R.China
e-mail: 1584539345@qq.com

Qian Tao
School of Software Engineering
South China University of Technology
Guangzhou, Guangdong, P.R.China
e-mail: taoqian@scut.edu.cn

***Abstract*—Having interested in people's spending habits, and there are not too many researches about it. I decided to cluster people's spending habits based on three features, which are ages, spending amount and income. Therefore, I spidered a dataset from Kaggle. And spark is the platform I use to deploy my distributed nodes to speed up the calculation. Using Scala to program, finally, I get three clusters which corresponding to three main ages.**

Keywords- k-mean algorithm, spark, spending habits

I. INTRODUCTION

With the rapid development of smart phones and Mobile payment, and people's life becomes better and better. The spending habits of people of different ages change a lot. Such as, it is obviously people who are young are more likely to spend more and on the contract their income is lower than the cost. Usually most older people are prefer saving money, but now because of mobile payment, some of them spend much more. In a word, it is interesting to extract some useful information from these dirty data, so I use k-mean algorithm to look deeper and more directly upon these subtle changes and spending habits. Later I may use linear regression to predict someone's cost and income based on his age.

II. METHODOLOGY

k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. k-means clustering minimizes within-cluster variances (squared Euclidean distances), but not regular Euclidean distances, which would be the more difficult Weber problem: the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances. For instance, better Euclidean solutions can be found using k-medians [1]

The problem is computationally difficult (NP-hard); however, efficient heuristic algorithms converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both k-means and Gaussian mixture modeling. They both use cluster centers to model the data; however, k-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

The algorithm has a loose relationship to the k-nearest neighbor classifier, a popular machine learning technique for classification that is often confused with k-means due to the name. Applying the 1-nearest neighbor classifier to the cluster centers obtained by k-means classifies new data into the existing clusters. This is known as nearest centroid classifier or Rocchio algorithm.

Defined data in $\mathbb{R}m$

Associate each cluster with a prototype $\boldsymbol{\mu}k \in \mathbb{R}m$, for $k \in \{1, \cdots, K\}$

Make an assignment $\mathbf{r}i$ of each sample to $\mathbf{x}i$ a cluster Objective: find prototypes and an assignment to minimize

$$\boldsymbol{L}(\{\boldsymbol{r}\},\{\boldsymbol{\mu}\}) = \sum_{i=1}^{n}\sum_{k=1}^{K} r_{ik}\|\mathbf{x}_i - \boldsymbol{\mu}_k\|$$

Where $rik \in 0, 1$, $rik = 1$ denotes $\mathbf{x}i$ belongs to the cluster k, $rik = 0$ otherwise, n is the number of samples

Initialize prototypes $\boldsymbol{\mu}1, \cdots, \boldsymbol{\mu}k$

Repeat until converged: Step 1: Assign each sample to the closest prototype

$$k^* = \underset{k \in \{1,\cdots,K\}}{\operatorname{argmin}} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|, r_{ik} = \begin{cases} 1, & k = k^* \\ 0, & otherwise \end{cases}$$

Step 2: For each k, set $\boldsymbol{\mu}k$ to the centroid of assigned samples

$$\boldsymbol{\mu}_k = \frac{1}{n_k}\sum_{i=1}^{n} r_{ik}\mathbf{x}_i$$

where $nk = \sigma i\ ri$k

Loss function:

$$\boldsymbol{L}(\{\mathbf{r}\},\{\boldsymbol{\mu}\}) = \sum_{i=1}^{n}\sum_{k=1}^{K} r_{ik}\|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

Solve this via coordinate descent:

Step 1: Fix $\mathbf{r}$, update $\boldsymbol{\mu}$: minimize loss by assigning each sample to the cluster that is closest

Step 2: Fix $\boldsymbol{\mu}$, update $\mathbf{r}$: work with squared distance

$$L = \sum_{i=1}^{n}\sum_{k=1}^{K} r_{ik}\,(\mathbf{x}_i - \boldsymbol{\mu}_k)^{\mathrm{T}}(\mathbf{x}_i - \boldsymbol{\mu}_k)$$

$$\frac{\partial L}{\partial \boldsymbol{\mu}_k} = -2\sum_{i=1}^{n} r_{ik}\,(\mathbf{x}_i - \boldsymbol{\mu}_k) = 0$$

$$\leftrightarrow \sum_{i=1}^{n} r_{ik}\,\mathbf{x}_i = \boldsymbol{\mu}_k \sum_{i=1}^{n} r_{ik} \leftrightarrow \boldsymbol{\mu}_k = \frac{\sum_{i=1}^{n} r_{ik}\mathbf{x}_i}{\sum_{i=1}^{n} r_{ik}}$$

III. Experiments

Step1: Get the dataset, I spidered a dataset from Kaggle about people's spending and income.

183	46	98	15		
184	29	98	88		
185	41	99	39		
186	30	99	97		
187	54	101	24		
188	28	101	68		
189	41	103	17		
190	36	103	85		
191	34	103	23		
192	32	103	69		
193	33	113	8		
194	38	113	91		
195	47	120	16		
196	35	120	79		
197	45	126	28		
198	32	126	74		
199	32	137	18		
200	30	137	83		
201					
202					
203					
204					
205					
206					
207					
208					
209					

CUSTOMER

Figure 1 dataset from Kaggle

From left to right, the column are age, spend and cost separately. This dataset is not that big, but for k-mean cluster it is enough.

Step2: choose a development environment, Spark is my option. And I run it on IJ Idea, a powerful platform.

As we know, Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing.

```
0/12/09 14:46:22 INFO DAGScheduler: Missing parents: List()
0/12/09 14:46:22 INFO DAGScheduler: Submitting ResultStage 3 (MapPartitionsRDD[12] at mapPartitionsWithIndex at KMeans.scala:392),
0/12/09 14:46:22 INFO MemoryStore: Block broadcast_5 stored as values in memory (estimated size 7.9 KiB, free 897.3 MiB)
0/12/09 14:46:22 INFO MemoryStore: Block broadcast_5_piece0 stored as bytes in memory (estimated size 3.8 KiB, free 897.3 MiB)
0/12/09 14:46:22 INFO BlockManagerInfo: Added broadcast_5_piece0 in memory on LAPTOP-D4V6SDAA:5282 (size: 3.8 KiB, free: 897.5 MiB)
0/12/09 14:46:22 INFO SparkContext: Created broadcast 5 from broadcast at DAGScheduler.scala:1223
0/12/09 14:46:22 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 3 (MapPartitionsRDD[12] at mapPartitionsWithIndex a
0/12/09 14:46:22 INFO TaskSchedulerImpl: Adding task set 3.0 with 1 tasks
0/12/09 14:46:22 INFO TaskSetManager: Starting task 0.0 in stage 3.0 (TID 3, LAPTOP-D4V6SDAA, executor driver, partition 0, PROCESS
0/12/09 14:46:22 INFO Executor: Running task 0.0 in stage 3.0 (TID 3)
0/12/09 14:46:22 INFO BlockManager: Found block rdd_6_0 locally
0/12/09 14:46:22 INFO BlockManager: Found block rdd_10_0 locally
0/12/09 14:46:22 INFO Executor: Finished task 0.0 in stage 3.0 (TID 3). 1429 bytes result sent to driver
0/12/09 14:46:22 INFO TaskSetManager: Finished task 0.0 in stage 3.0 (TID 3) in 16 ms on LAPTOP-D4V6SDAA (executor driver) (1/1)
0/12/09 14:46:22 INFO TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
0/12/09 14:46:22 INFO DAGScheduler: ResultStage 3 (collect at KMeans.scala:392) finished in 0.024 s
0/12/09 14:46:22 INFO DAGScheduler: Job 3 is finished. Cancelling potential speculative or zombie tasks for this job
0/12/09 14:46:22 INFO TaskSchedulerImpl: Killing all running tasks in stage 3: Stage finished
0/12/09 14:46:22 INFO DAGScheduler: Job 3 finished: collect at KMeans.scala:392, took 0.030187 s
0/12/09 14:46:22 INFO MemoryStore: Block broadcast_6 stored as values in memory (estimated size 528.0 B, free 897.3 MiB)
0/12/09 14:46:22 INFO MemoryStore: Block broadcast_6_piece0 stored as bytes in memory (estimated size 507.0 B, free 897.3 MiB)
0/12/09 14:46:22 INFO BlockManagerInfo: Added broadcast_6_piece0 in memory on LAPTOP-D4V6SDAA:5282 (size: 507.0 B, free: 897.5 MiB)
0/12/09 14:46:22 INFO SparkContext: Created broadcast 6 from broadcast at KMeans.scala:381
```

Figure 2 Spark environment

So what we need to do is to set master and worker, a traditional distributional approach. To compute faster, I set 3 local master nodes to help me calculate.

Then it is very simple to accomplish the mission, we just need to call its own MLlib. And in there, we have a K-mean model that can be use directly.

Step3: Choosing the number of clusters. Because of age is young, middle-age, old time, I set the number to be 3.

```
// 将数据集聚类,2个类,20次迭代,形成数据模型
val numClusters = 3
val numIterations = 20
val model = KMeans.train(parsedData, numClusters, numIterations)
```

Figure 3 number of cluster

Step4:Call the MLlib k-mean model to cluster

The command is in the Figure 3

Val model = KMeans.*train*(parsedData, numClusters, numIterations)

From the perspective of physical deployment, Spark is mainly divided into two types of nodes, Master node and Worker node. Master node mainly runs the centralized part of cluster manager, and its role is to allocate Application to Worker node and maintain the state of Worker node, Driver node and Application. The Worker node is responsible for the specific business operation.

From the perspective of Spark program running, Spark is mainly divided into driver node and actu[2]

Compared to Hadoop's MapReduce, Spark's memory-based operations are more than 100 times faster, and its hardware-based operations are more than 10 times faster. Spark implements an efficient DAG execution engine that can efficiently process data flows through memory. The intermediate result of the calculation exists in memory.

Thanks to fast speed of spark, I get the answer quickly.

IV. RESULTS AND DISCUSSION

After about 20 times iterations, I got the final result.

```
    // 返回数据集和结果
    val result = data.map {
      line =>
        val linevectore = Vectors.dense(line.split( regex = ",").map(_.toDouble))
        val prediction = model.predict(linevectore)
        line + " " + prediction
    }.collect.foreach(println)

    sc.stop
  }
}
```

Here are the three cluster centers:

From the result we can see that age actually do not make very big influence on the spending and cost habits, the three main types are saving money, overspending, and balance. It is very conformable with the reality.

Figure 4 Get the result

```
20/12/09 14:46:26 INFO BlockManagerInfo: Removed broadcast_36
20/12/09 14:46:26 INFO KMeans: KMeans converged in 9 iteratio
20/12/09 14:46:26 INFO KMeans: The cost is 187556.05851689828
20/12/09 14:46:26 INFO MapPartitionsRDD: Removing RDD 6 from
20/12/09 14:46:26 INFO BlockManager: Removing RDD 6
Cluster centres:
  [35.02272727272727,83.81818181818183,49.26136363636364]
  [45.21739130434783,26.304347826086957,20.913043478260867]
  [40.98876404494382,46.41573033707865,58.69662921348314]
Within Set Sum of Squared Errors = 187556.05851689828
Vectors 7.3 1.5 10.9 is belong to cluster:1
Vectors 4.2 11.2 2.7 is belong to cluster:1
Vectors 18.0 4.5 3.8 is belong to cluster:2
20/12/09 14:46:26 INFO MemoryStore: Block broadcast_39 stored
20/12/09 14:46:26 INFO MemoryStore: Block broadcast_39_piece0
20/12/09 14:46:26 INFO BlockManagerInfo: Added broadcast_39_p
20/12/09 14:46:26 INFO SparkContext: Created broadcast 39 fro
20/12/09 14:46:26 INFO SparkContext: Starting job: sum at KMe
```

```
20/12/09 14:46:26 INFO TaskSche
20/12/09 14:46:26 INFO DAGSchec
20/12/09 14:46:26 INFO DAGSchec
20/12/09 14:46:26 INFO TaskSche
20/12/09 14:46:26 INFO DAGSchec
19,15,39 1
21,15,81 2
20,16,6 1
23,16,77 2
31,17,40 1
22,17,76 2
35,18,6 1
23,18,94 2
64,19,3 1
30,19,72 2
67,19,14 1
35,19,99 2
58,20,15 1
24,20,77 2
37,20,13 1
22,20,79 2
35,21,35 1
20,21,66 2
52,23,29 1
35,23,98 2
```

Figure 5 allocation of data

V. CONCLUSION

Using Spark is a very convenient way to cluster or do machine learning. And after train my data, I get a very Fact-based result, which are saving money, overspending money and balance. And after observe my 200 data, I found that some old people also spend a lot ,while some young people spend little. Therefore we can say that although age and technology differ from each other , their cost behavior do not changes , or in another word, spending behavior do not have much relationship with age or technology .

REFERENCES

[1] https://en.wikipedia.org/wiki/K-means_clustering

[2] https://spark.apache.org/docs/latest/