Hadoop – Mahout
1-Clustering

عادل قاضی خانی
هیئت علمی گروه کامپیوتر دانشگاه امام رضا (ع)

تعریف

- یکی از پروژه های Apache که از سال ۲۰۰۸ آغاز شد
- مجموعه ای از **تکنیکها و الگوریتمها** حول سه محور **سیستم های توصیه گر، خوشه بندی و طبقه بندی**
- قابلیت اعمال **الگوریتمهای یادگیری** روی **داده های حجیم**

چند مثال

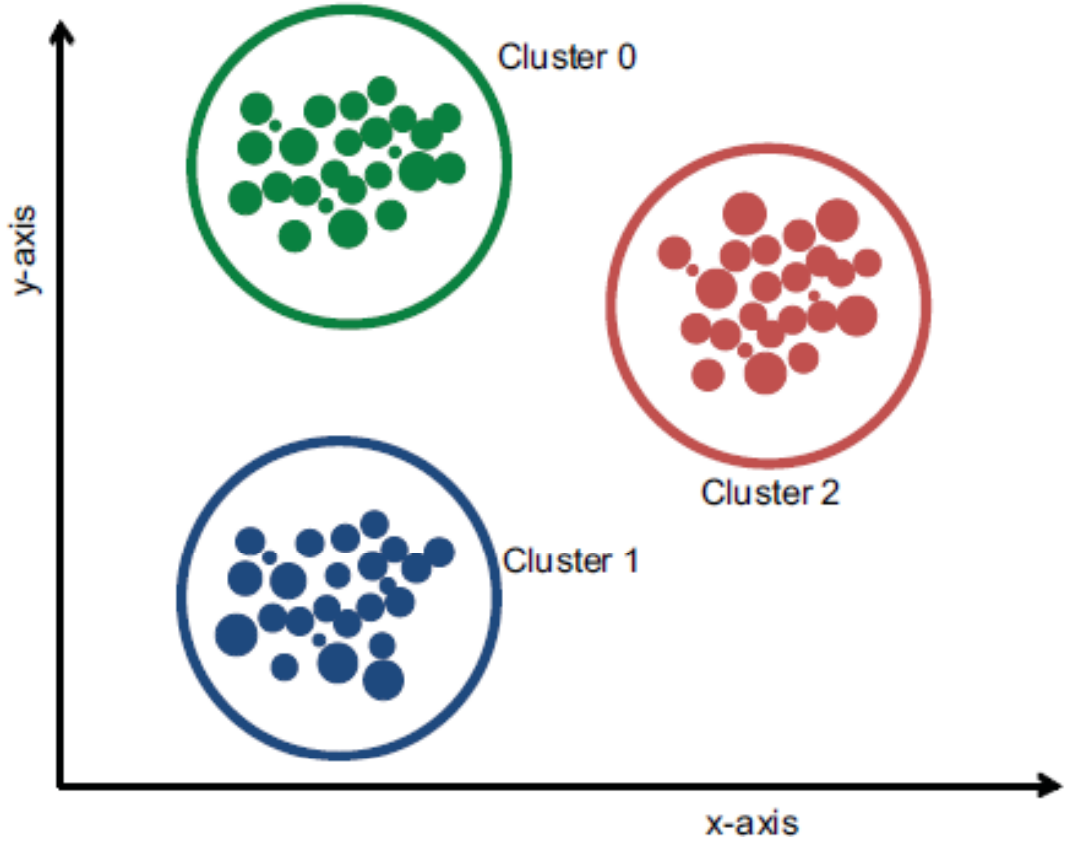
- Google News در روز 3.5 میلیون خبر جدید ایجاد می کند
- Netflix برای اهدای جایزه ۱۰۰ میلیون رتبه بندی را در اختیار داشت (سیستم توصیه گر)

خوشه بندی در Mahout

- عبارت است از یادگیری از داده های آموزشی که برچسب (y) ندارند. در این نوع یادگیری داده‌های آموزشی بر اساس **شباهت بین ورودی‌ها (ویژگی‌ها)** به چند خوشه تقسیم می‌شوند

- دسته بندی کتاب بر اساس موضوع

خوشه بندی در Mahout

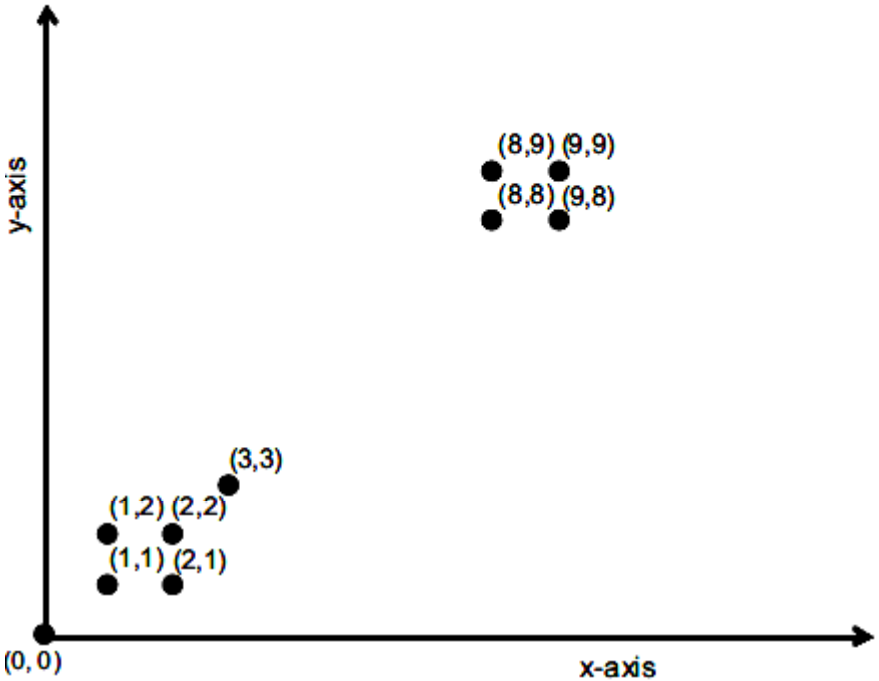


خوشه بندی در Mahout

- **خوشه بندی معمولاً سه قسمت دارد**
  - الگوریتم
  - معیار شباهت
  - شرط پایان

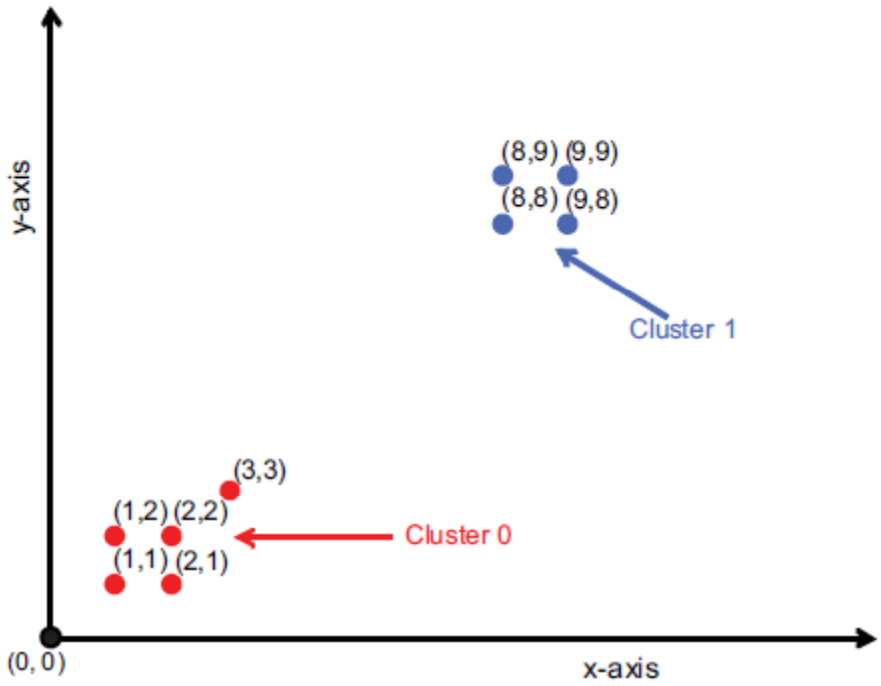
خوشه بندی در Mahout

- مثال
- ورودی



خوشه بندی در Mahout

- مثال
- خروجی الگوریتم



خوشه بندی در Mahout

- **معیارهای شباهت:** با استفاده از این معیار داده ها گروه بندی می شوند

خوشه بندی در Mahout

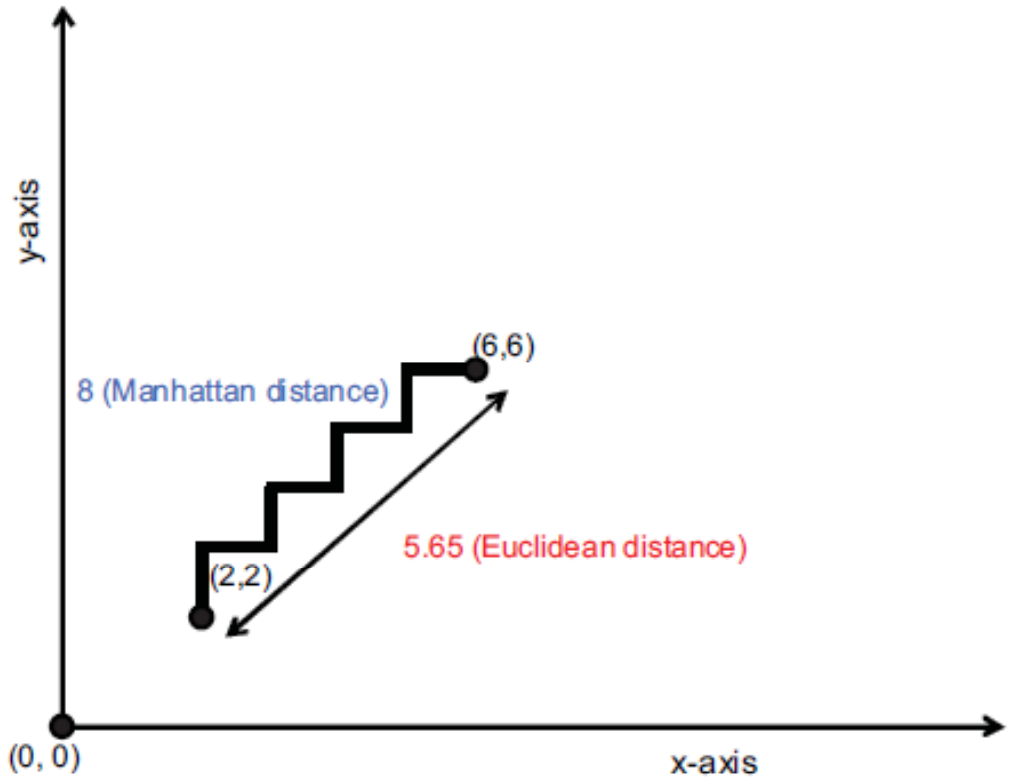
- **معیار فاصله اقلیدسی:**

$$d = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2}$$

- **معیار فاصله منهتن:**

$$d = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|$$

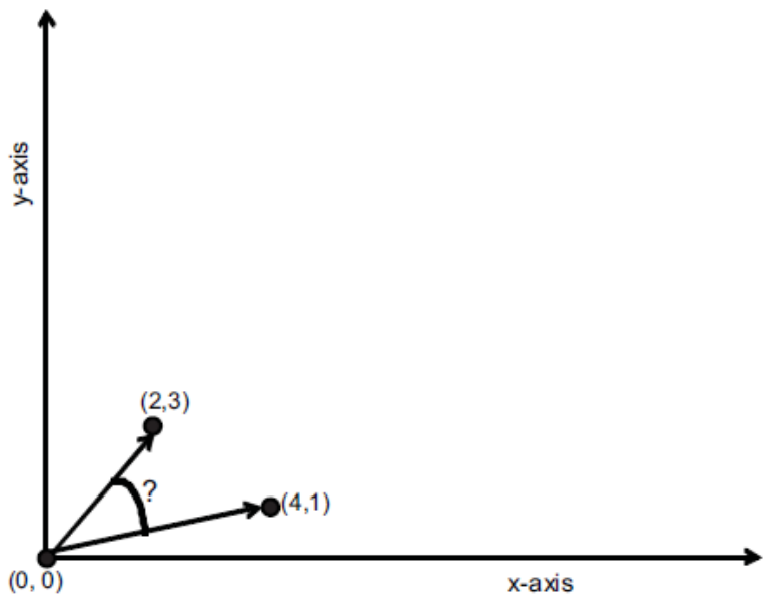
خوشه بندی در Mahout



خوشه بندی در Mahout

- **معیار فاصله کسینوسی:**

$$d = 1 - \frac{(a_1 b_1 + a_2 b_2 + \ldots + a_n b_n)}{\left(\sqrt{(a_1^2 + a_2^2 + \ldots + a_n^2)}\sqrt{(b_1^2 + b_2^2 + \ldots + b_n^2)}\right)}$$

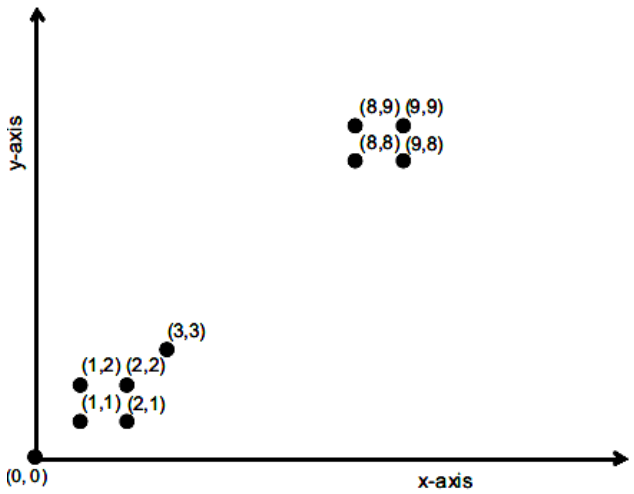


خوشه بندی در Mahout

- **معیار فاصله تانی موتو:**

$$d = 1 - \frac{(a_1b_1 + a_2b_2 + \ldots + a_nb_n)}{\sqrt{(a_1^2 + a_2^2 + \ldots + a_n^2)} + \sqrt{(b_1^2 + b_2^2 + \ldots + b_n^2)} - (a_1b_1 + a_2b_2 + \ldots + a_nb_n)}$$

خوشه بندی در Mahout

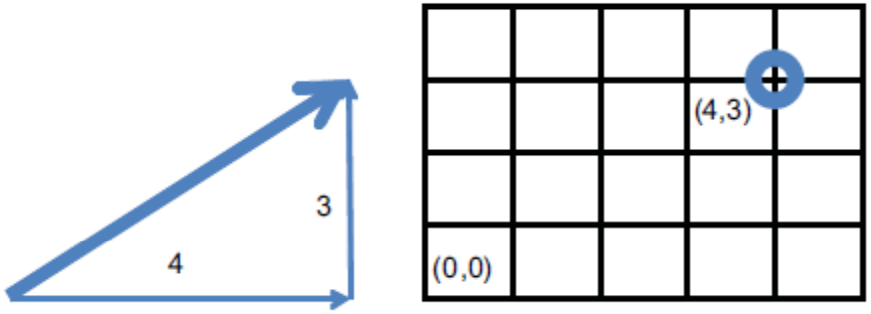


- نتایج خوشه بندی با معیارهای مختلف

Distance measure	Number of iterations	Vectors[a] in cluster 0	Vectors in cluster 1
EuclideanDistanceMeasure	3	0, 1, 2, 3, 4	5, 6, 7, 8
SquaredEuclideanDistanceMeasure	5	0, 1, 2, 3, 4	5, 6, 7, 8
ManhattanDistanceMeasure	3	0, 1, 2, 3, 4	5, 6, 7, 8
CosineDistanceMeasure	1	1	0, 2, 3, 4, 5, 6, 7, 8
TanimotoDistanceMeasure	3	0, 1, 2, 3, 4	5, 6, 7, 8

نمایش داده ها با بردار

- در Mahout



نمایش داده ها با بردار

- در Mahout سه کلاس برای تعریف بردار وجود دارد که با توجه به نوع داده ها انتخاب کنیم

- DenseVector
- RandomAccessSparseVector
- SequentialAccessSparseVector

نمایش داده ها با بردار

- در Mahout سه کلاس برای تعریف بردار وجود دارد که با توجه به نوع داده ها انتخاب کنیم
- DenseVector class
  - آرایه ای از اعداد اعشاری که اندازه آن برابر با تعداد ویژگی ها است. برای داده هایی که بیشتر ویژگی ها مقدار دارند

نمایش داده ها با بردار

- در Mahout سه کلاس برای تعریف بردار وجود دارد که با توجه به نوع داده ها انتخاب کنیم
- RandomAccessSparseVector class
- ویژگی های که مقدار دارند فقط ذخیره می شوند. دسترسی به داده ها بصورت تصادفی

نمایش داده ها با بردار

- در Mahout سه کلاس برای تعریف بردار وجود دارد که با توجه به نوع داده ها انتخاب کنیم
- SequentialAccessSparseVector class
- ویژگی های که مقدار دارند فقط ذخیره می شوند. داده ها تنک[1] هستند. دسترسی به داده ها بصورت ترتیبی

1-sparse

نمایش داده ها با بردار

- ویژگی های داده ها باید عددی باشد
- مثال

Apple	Weight (kg) (0)	Color (1)	Size (2)	Vector
Small, round, green	0.11	510	1	[0.11, 510, 1]
Large, oval, red	0.23	650	3	[0.23, 650, 3]
Small, elongated, red	0.09	630	1	[0.09, 630, 1]
Large, round, yellow	0.25	590	3	[0.25, 590, 3]
Medium, oval, green	0.18	520	2	[0.18, 520, 2]

نمایش داده ها با بردار

- **مثال**
- تفاوت بین بازه ویژگی ها ← وزندهی به ویژگی ها
- اگر شباهت ویژگی رنگ مهم نباشد، رنگهای مختلف را بصورت ویژگی بیان می کردیم (تنک)

نمایش داده های متنی با بردار

- ویژگی ها ← کلمات
- مقدار هر ویژگی تعداد تکرار کلمه
- داده ها تنک
- وزن ویژگی ها بر اساس تکرار نیست(کلمات مهم نادر هستند)

وزن دهی به ویژگی ها در متن

- روش TFIDF*
- در این روش علاوه بر تعداد تکرار کلمه، تعداد اسناد متنی که کلمه در آن وجود دارد مهم است.
- وزن ویژگی:

$$W_i = TF_i \cdot \log \frac{N}{DF_i}$$

- TF_i تعداد تکرار کلمه
- N تعداد اسناد متنی
- DF_i تعداد متنهایی که کلمه در آن است

Term frequency–inverse document frequency

وزن دهی به ویژگی ها در متن

- **روش TFIDF** از وابستگی بین ویژگی ها پشتیبانی نمی کند
- **مثلاً:** Coca cola

- **مفهوم** ***ngram*** ؟

وزن دهی به ویژگی ها در متن

- **یک گروه از کلمات** یک ***n-gram*** است
- یک کلمه یک *unigram* دو کلمه وابسته یک *bigram* و ....
- در ***Mahout*** می توان چند کلمه وابسته را بصورت ***n-gram*** تعریف کرد
- سپس از **TFIDF** استفاده کرد

نرمال سازی داده ها

- در **داده های عددی** برای حذف تأثیر **ویژگی های با بازه بزرگ**
- در **داده های متنی** حذف تأثیر متنهایی که **بدلیل بزرگ بودن** و **غیر صفر بودن خیلی ویژگی ها** شبیه خیلی از متنها هستند

نرمال سازی داده ها

- نرمال سازی برای داده هایی با **سه ویژگی**

$$\frac{x}{(|x|^p + |y|^p + |z|^p)^{1/p}}, \frac{y}{(|x|^p + |y|^p + |z|^p)^{1/p}}, \frac{z}{(|x|^p + |y|^p + |z|^p)^{1/p}}$$

$$\frac{x}{|x| + |y| + |z|}, \frac{y}{|x| + |y| + |z|}, \frac{z}{|x| + |y| + |z|}$$

$$\frac{x}{\max(|x|, |y|, |z|)}, \frac{y}{\max(|x|, |y|, |z|)}, \frac{z}{\max(|x|, |y|, |z|)}$$

الگوریتمهایی خوشه‌بندی

- الگوریتم *k-means*
- الگوریتم *fuzzy c-means*

خوشه‌بندی *k-means*

- **الگوریتم**
- k داده را بصورت تصادفی بعنوان مراکز خوشه انتخاب کن (m_k)
- **تکرار کن**
- **برای هر داده**
- برای هر داده x_i فاصله آنرا با مراکز خوشه بدست بیاور

$$C(i) = \arg\min_{1 \le k \le K} \|x_i - m_k\|^2,\ i = 1, \ldots, N$$

- x_i را در خوشه‌ای قرار بده که به مرکز آن نزدیکتر است

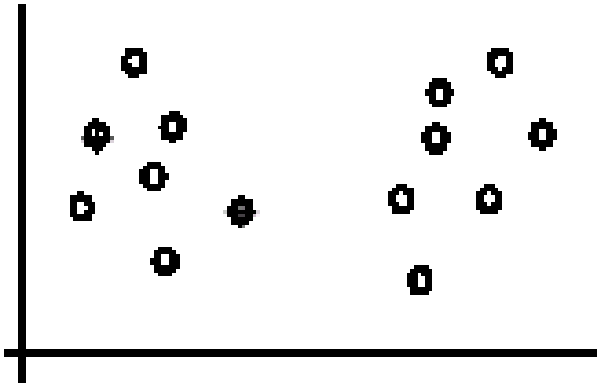
خوشه‌بندی *k-means*

- ادامه
- ...
- مراکز خوشه‌ها را مجدداً محاسبه کن (میانگین داده‌های موجود در خوشه)

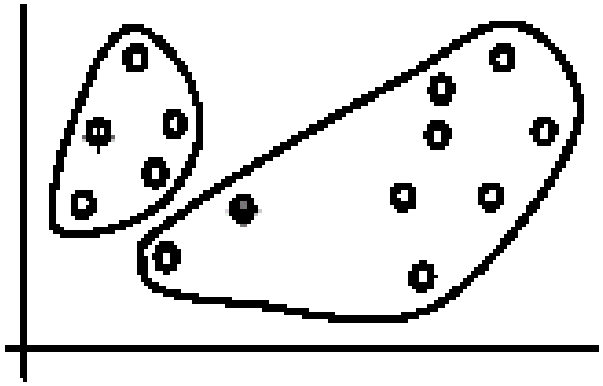
$$m_k = \frac{\sum_{i:C(i)=k} x_i}{N_k}, \; k = 1, \ldots, K.$$

- **تا زمانیکه شرط پایان (عدم تغییر مراکز خوشه، ...)**

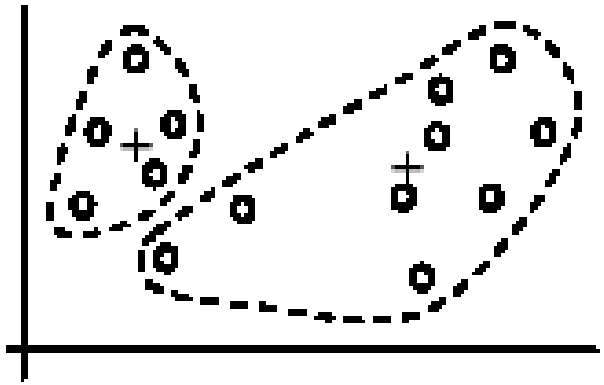
خوشه‌بندی *k-means* – مثال



(A). Random selection of *k* centers

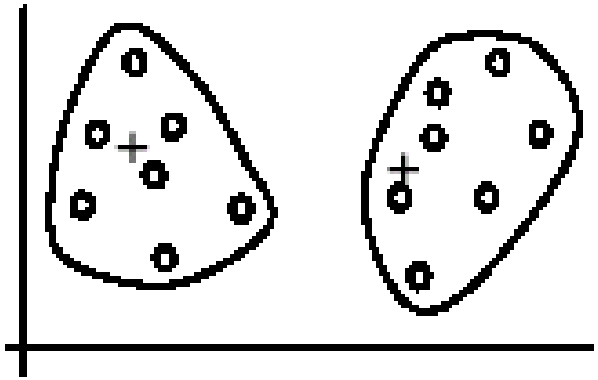


Iteration 1: (B). Cluster assignment

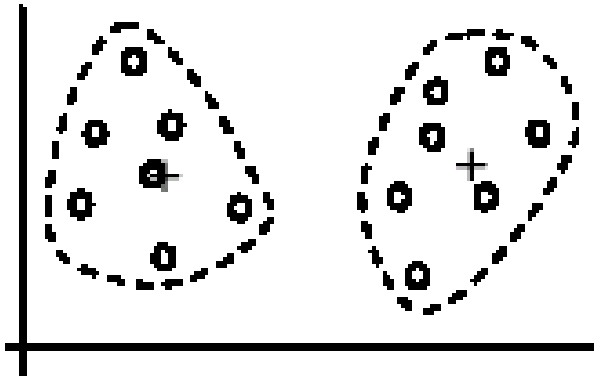


(C). Re-compute centroids

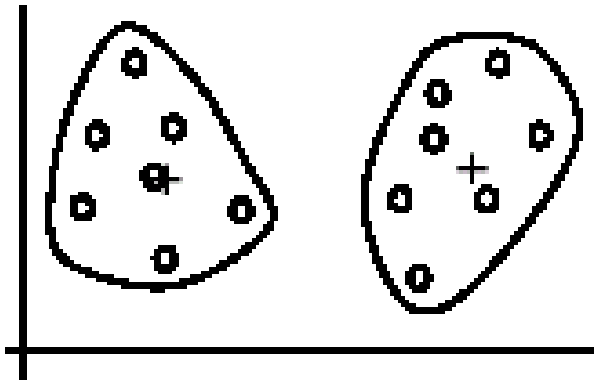
خوشه‌بندی k-means – مثال



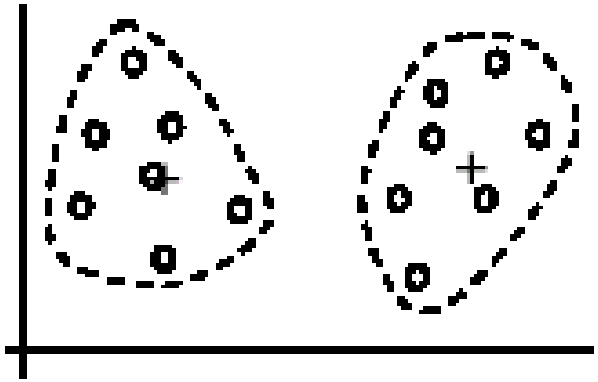
Iteration 2: (D). Cluster assignment



(E). Re-compute centroids

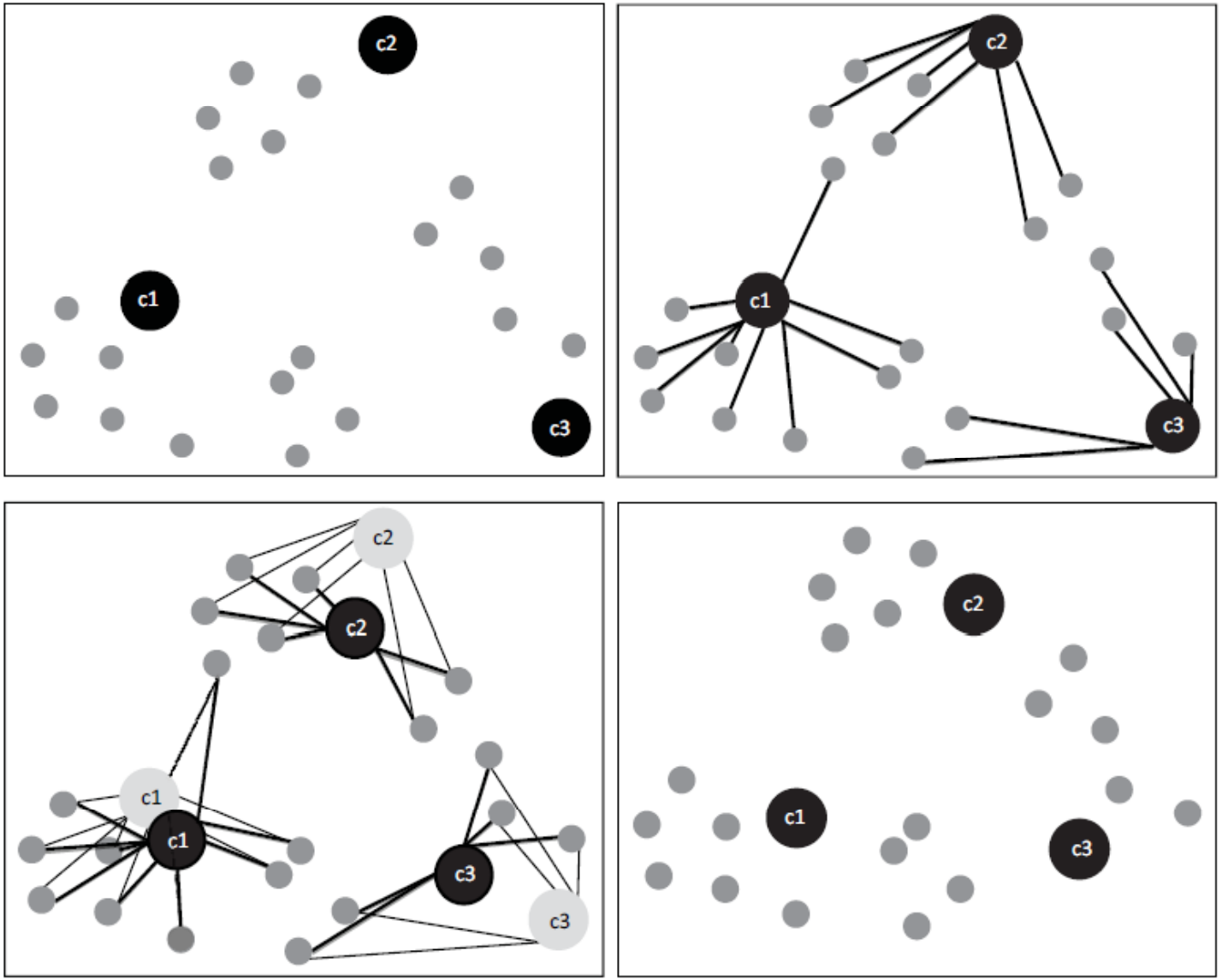


Iteration 3: (F). Cluster assignment



(G). Re-compute centroids

خوشه‌بندی k-means – مثال



خوشه‌بندی k-means

- مثال در Mahout
- تولید داده

```
private static void generateSamples(List<Vector> vectors, int num,
  double mx, double my, double sd) {
     for (int i = 0; i < num; i++) {
       vectors.add(new DenseVector(
         new double[] {
           UncommonDistributions.rNorm(mx, sd),
           UncommonDistributions.rNorm(my, sd)
         }
       ));
     }
   }
```

خوشه‌بندی *k-means* در Mahout

```
public static void main(String[] args) {
  List<Vector> sampleData = new ArrayList<Vector>();

  generateSamples(sampleData, 400, 1, 1, 3);
  generateSamples(sampleData, 300, 1, 0, 0.5);
  generateSamples(sampleData, 300, 0, 2, 0.1);

  int k = 3;

  List<Vector> randomPoints = RandomPointsUtil.chooseRandomPoints(
    sampleData, k);
    List<Cluster> clusters = new ArrayList<Cluster>();

    int clusterId = 0;
    for (Vector v : randomPoints) {
      clusters.add(new Cluster(v, clusterId++,
          new EuclideanDistanceMeasure()));
    }

    List<List<Cluster>> finalClusters
        = KMeansClusterer.clusterPoints(sampleData, clusters,
            new EuclideanDistanceMeasure(), 3, 0.01);
    for(Cluster cluster : finalClusters.get(
        finalClusters.size() - 1)) {
      System.out.println("Cluster id: " + cluster.getId()
          + " center: " +
          cluster.getCenter().asFormatString());
    }
  }
}
```

Generate three sets of points

Run KMeansClusterer

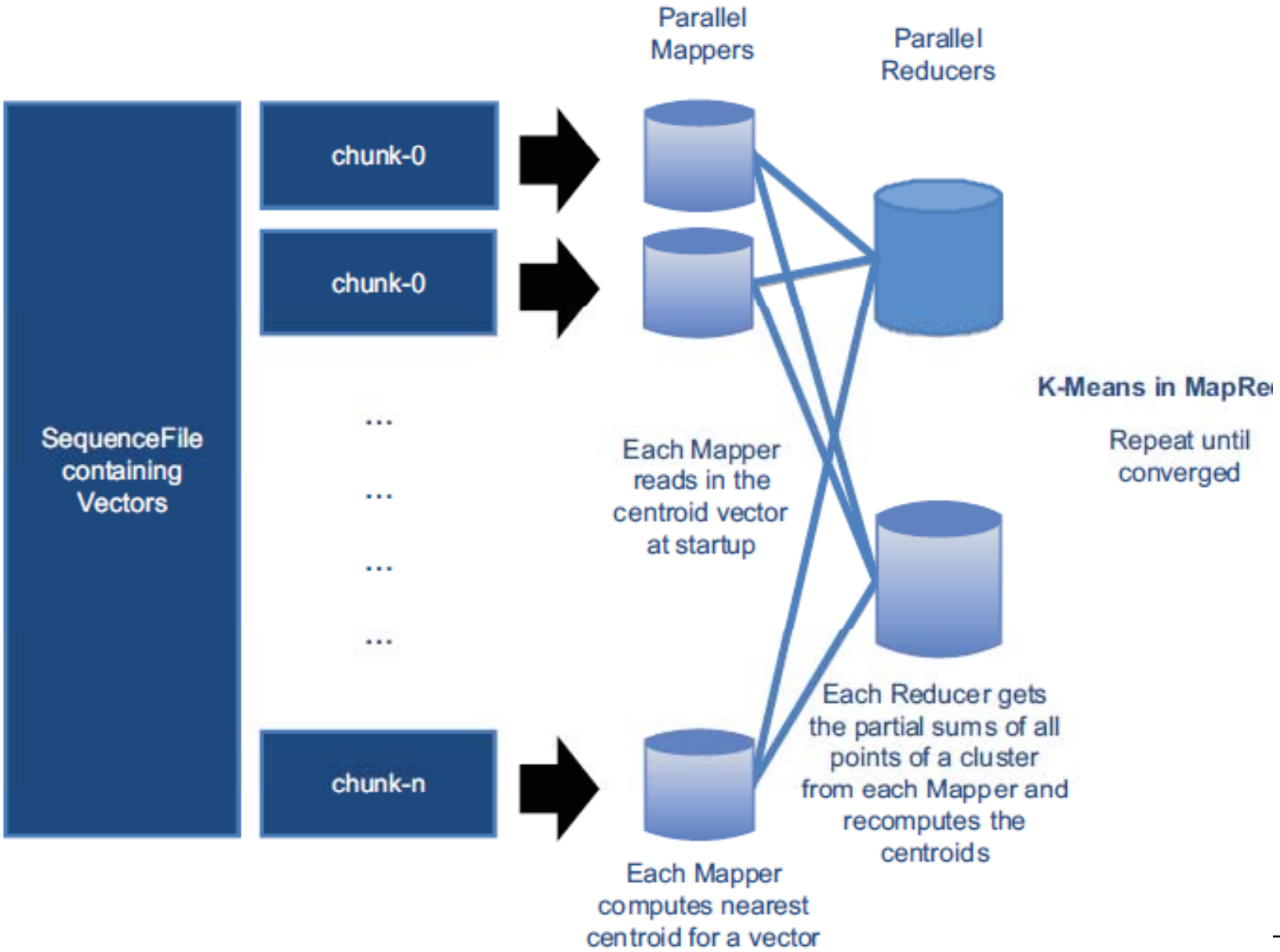
Read center of cluster and print it

خوشه‌بندی k-means

- مثال در Mahout
- فراخوانی k-means

```
KmeansDriver.runJob(hadoopConf,
      inputVectorFilesDirPath, clusterCenterFilesDirPath,
      outputDir, new EuclideanDistanceMeasure(),
      convergenceThreshold, numIterations, true, false);
```

خوشه‌بندی k-means ساختار MapReduce



خوشه‌بندی *k-means*

- مراکز خوشه اولیه ؟

- الگوریتمهای تکاملی
- خوشه بندی canopy

خوشه‌بندی canopy

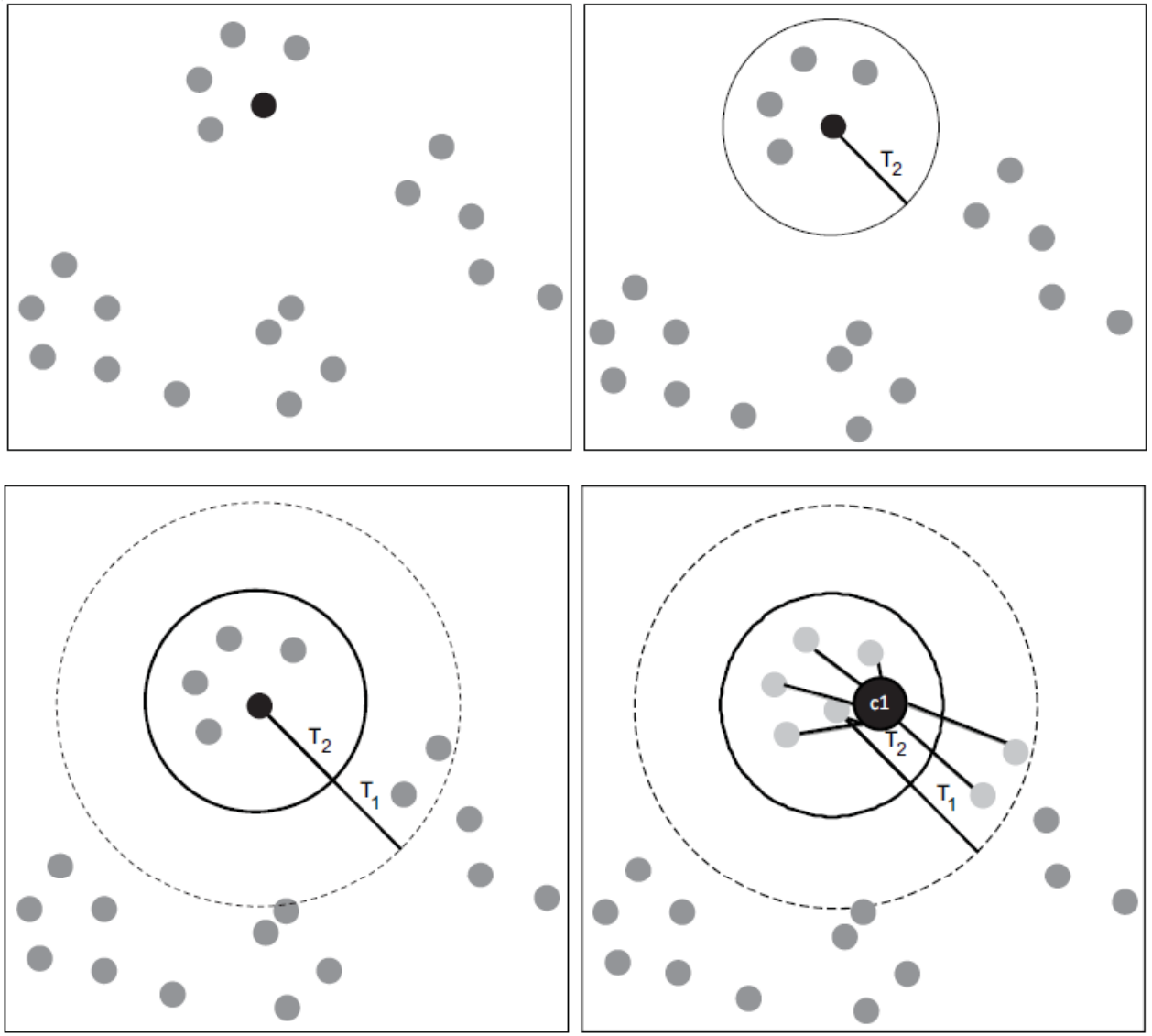
- **الگوریتم**
- دو آستانه *T1* و *T2* *(T1>T2)*

- یک داده را بطور تصادفی بعنوان مرکز اولین سایبان انتخاب کن و از لیست حذف کن
- **تا زمانیکه داده ای در لیست وجود دارد**
- برای داده *x* در لیست فاصله آنرا با مراکز سایبان بدست بیاور

$$d_i = \|x - c_i\|^2$$

- اگر $T_2 \le d < T_1$ ، x را در سایبان i قرار بده و حذف کن
- اگر $d < T_2$ ، x را حذف کن
- اگر x در هیچ سایبانی قرار نگرفت سایبان جدید ایجاد کن

خوشه‌بندی canopy



خوشه‌بندی canopy برای k-means در Mahout

```
public static void CanopyExample() {
  List<Vector> sampleData = new ArrayList<Vector>();

  generateSamples(sampleData, 400, 1, 1, 2);
  generateSamples(sampleData, 300, 1, 0, 0.5);
  generateSamples(sampleData, 300, 0, 2, 0.1);

  List<Canopy> canopies = CanopyClusterer.createCanopies(
    sampleData, new EuclideanDistanceMeasure(), 3.0, 1.5);

  for(Canopy canopy : canopies) {
    System.out.println("Canopy id: " + canopy.getId()


                + " center: " +
                canopy.getCenter().asFormatString());
   }
  }
```

Generate 3 sets of points

Run CanopyClusterer

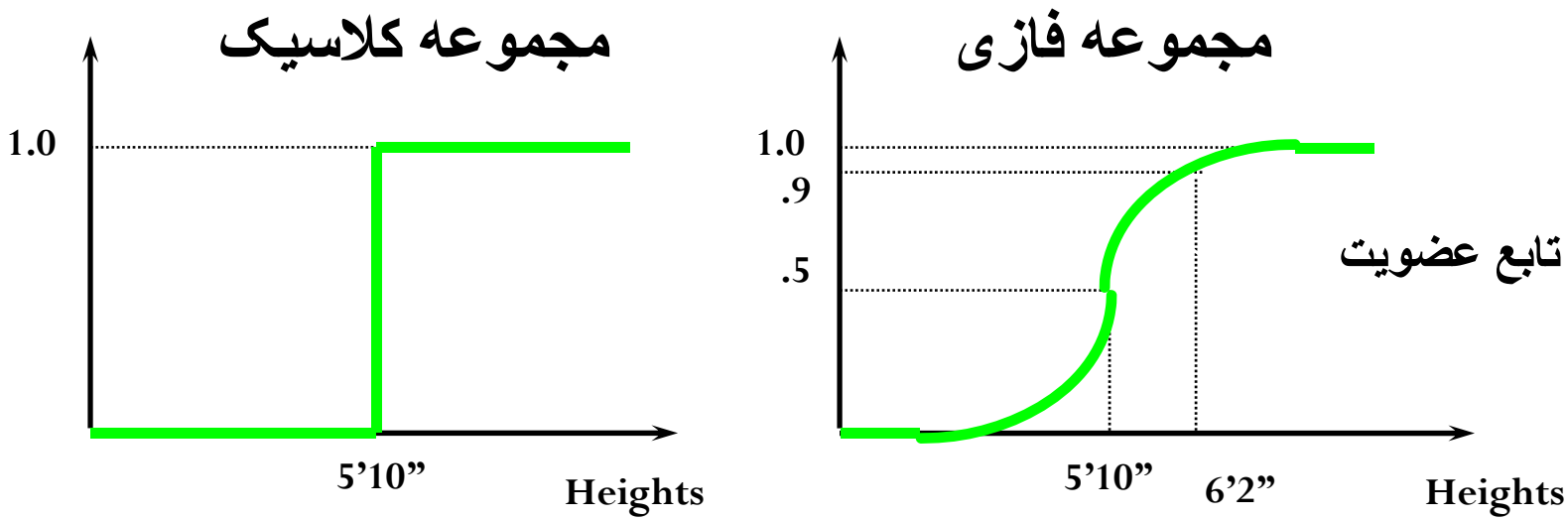
Read center of canopy and print

خوشه بندی *Fuzzy c-means*

- داده‌ها را به c خوشه نا همپوشان تقسیم می‌کند
- c توسط کاربر تعیین می‌شود
- استفاده از **مفهوم فازی برای تعلق داده‌ها به خوشه‌ها**

خوشه بندی Fuzzy c-means

- **یادآوری – مجموعه های فازی**
- در یک مجموعه فازی بر خلاف مجموعه های کلاسیک تعلق یک عضو بصورت عددی بین ۰ و ۱ مشخص می شود
- مثلاً : مجموعه افراد قد بلند



خوشه بندی *Fuzzy c-means*

- در این الگوریتم خوشه بندی، در خلال اجرای الگوریتم **تعلق هر نمونه داده به خوشه ها بصورت فازی** (بین [0,1]) تعریف می شود

خوشه بندی Fuzzy c-means

- **الگوریتم**
- **ورودی** : تعدادی داده (X) و تعداد خوشه ها (c)
- **خروجی** : تعلق هر داده به هر خوشه (U) و مراکز خوشه ها (V)
- مقداردهی اولیه بصورت تصادفی به V و تعیین m و c

خوشه بندی Fuzzy c-means

- در این الگوریتم برای **تعیین مقادیر بهینه** ***U و V*** ، تابع هدف زیر را باید بهینه کنیم

$$\min_{(\mathbf{U},\mathbf{V})}\left\{ J_m(\mathbf{U},\mathbf{V}) = \sum_{i=1}^{c}\sum_{k=1}^{n} u_{ik}^{m} D_{ik}^{2} \right\}$$

خوشه بندی Fuzzy c-means

- **تا زمانیکه** خطای تغییر خوشه ها (V) از حدی بیشتر است

$$\|\mathbf{V}_t - \mathbf{V}_{t-1}\| \le \varepsilon$$

- محاسبه U جدید

برای تمام داده ها

(یعنی تعلق هر داده به هر خوشه)

$$u_{ik} = \left[\sum_{j=1}^{c} \left(\frac{D_{ik}}{D_{jk}} \right)^{\frac{2}{m-1}} \right]^{-1}, \forall i, k$$

- محاسبه V جدید

$$\mathbf{v}_i = \left(\sum_{k=1}^{n} u_{ik}^m \mathbf{x}_k \Big/ \sum_{k=1}^{n} u_{ik}^m \right), \forall i$$

خوشه‌بندی *fuzzy c-means* در Mahout

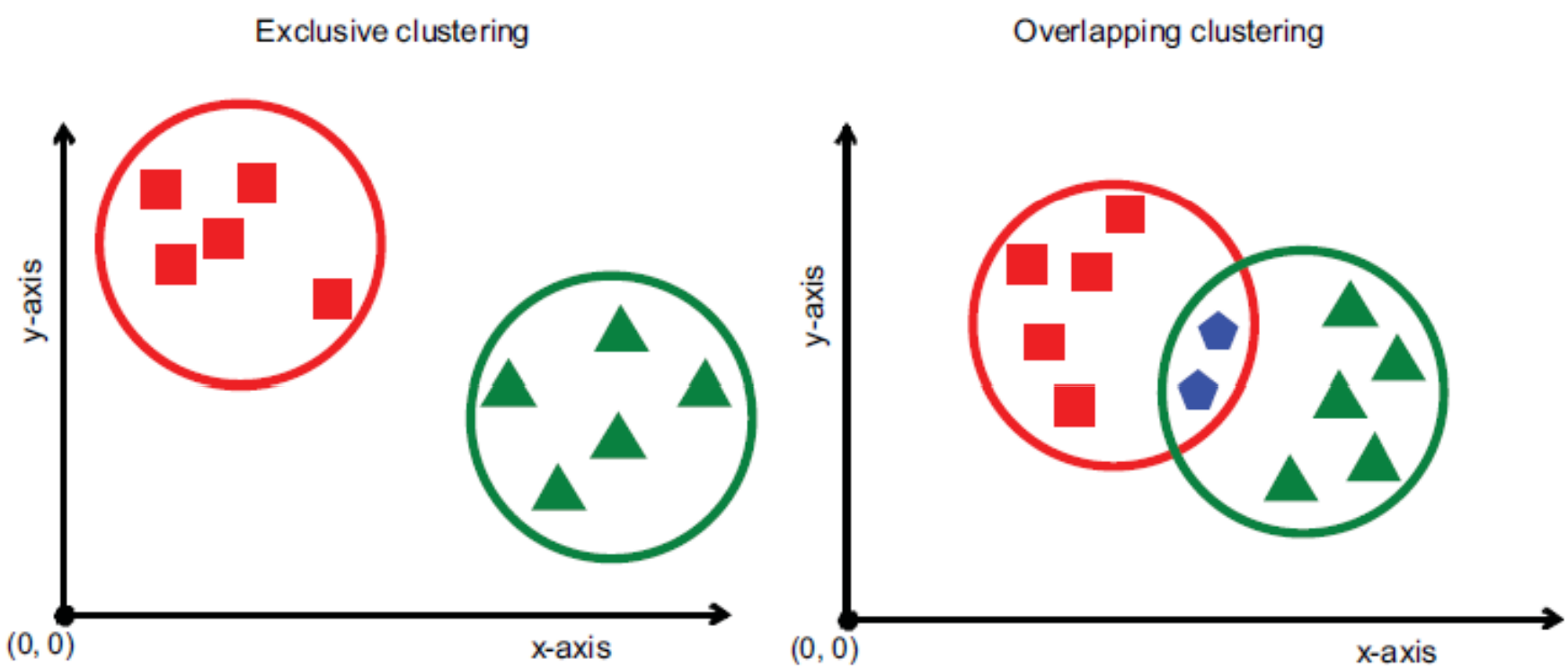
```
public static void FuzzyKMeansExample() {
  List<Vector> sampleData = new ArrayList<Vector>();

  generateSamples(sampleData, 400, 1, 1, 3);
  generateSamples(sampleData, 300, 1, 0, 0.5);
  generateSamples(sampleData, 300, 0, 2, 0.1);

   int k = 3;
  List<Vector> randomPoints
     = RandomPointsUtil.chooseRandomPoints(sampleData, k);
     List<SoftCluster> clusters = new ArrayList<SoftCluster>();

     int clusterId = 0;

  for (Vector v : randomPoints) {
    clusters.add(new SoftCluster(v, clusterId++,
        new EuclideanDistanceMeasure()));
  }

  List<List<SoftCluster>> finalClusters
      = FuzzyKMeansClusterer.clusterPoints(sampleData,
          clusters, new EuclideanDistanceMeasure(),
          0.01, 3, 10);
  for(SoftCluster cluster : finalClusters.get(
      finalClusters.size() - 1)) {
 System.out.println("Fuzzy Cluster id: "
     + cluster.getId()
     + " center: " + cluster.getCenter().asFormatString());
  }

}
```

Generate three sets of points

Run FuzzyKMeansClusterer

Read centers of fuzzy clusters

تفاوت میان *k-means* و *fuzzy c-means*



مثال واقعی

- یافتن کاربران مشابه در Twitter

- ورودی؟
- خوشه بندی؟

مثال واقعی - ورودی

- **یافتن کاربران مشابه در Twitter**

- ورودی ها باید بصورت کلید مقدار باشند
- **کلید:** نام کاربری ، **مقدار:** متن نوشته شده توسط کاربر

مثال واقعی - ورودی

- **یافتن کاربران مشابه در Twitter**

- **فاز نگاشت خواندن ورودی**
- هر خط از داده ها خوانده می شود، که در آن نام کاربری، برچسب زمانی و متن نوشته شده وجود دارد.
- در خروجی فاز نگاشت **کلید:** نام کاربری ، **مقدار:** متن نوشته شده توسط کاربر

مثال واقعی - ورودی

- **یافتن کاربران مشابه در Twitter**

- **فاز کاهش خواندن ورودی**
- تمام متن های نوشته شده توسط هر کاربر ادغام می شود
- در خروجی فاز کاهش **کلید:** نام کاربری ، **مقدار:** تمام متن نوشته شده توسط کاربر

مثال واقعی – twitter فاز نگاشت ورودی

```
public class ByKeyMapper extends Mapper<LongWritable,Text,Text,Text> {
  private Pattern splitter = Pattern.compile("\t");
  private int selectedField = 1; // tweet
  private int groupByField = 0; // username

  @Override
  protected void map(LongWritable key, Text value,
 Context context) throws IOException,
InterruptedException {
    String[] fields = splitter.split(value.toString());
    if (fields.length - 1 < selectedField ||
        fields.length - 1 < groupByField) {
      context.getCounter(
          "Map", "LinesWithErrors").increment(1);
      return;
    }

    String oKey = fields[groupByField];
    String oValue = fields[selectedField];
    context.write(new Text(oKey), new Text(oValue));

  }

}
```

Split line using Regex

Count line errors

Output username, tweet

مثال واقعی – twitter فاز کاهش ورودی

```
public class ByKeyReducer extends Reducer<Text,Text,Text,Text> {
  @Override
  protected void reduce(Text key,
                        Iterable<Text> values,
                        Context context) {
    StringBuilder output = new StringBuilder();
    for (Text value : values) {
      output.append(value.toString()).append(" ");
    }
    context.write(key, new Text(
        output.toString().trim()));
  }
}
```

Read user's tweet

Generate concatenated string

Output concatenated string

مثال واقعی - ورودی

- **یافتن کاربران مشابه در Twitter**

- **الگوریتم TFIDF** برای وزن دادن به ویژگی ها (کلمات)

- قبل از آن بهتر است ویژگی هایی (کلماتی) که خیلی در متنهای نوشته شده توسط کاربران تکرار می شوند **(stop-word)** **حذف** شوند

مثال واقعی - ورودی

- **یافتن کاربران مشابه در Twitter**

- **مشکل:** در متنهای twitter ممکن است **کلمات نا مفهوم** تایپ شده باشد

- می توان از **فیلترهای فونتیک** (آوایی) استفاده کرد که **یک کلمه را به نحوه بیان** آن تبدیل می کند

مثال واقعی - ورودی

- **یافتن کاربران مشابه در Twitter**

- **مثلاً:** کلمه Loke یا Loca به LK تبدیل می شود

- در اینجا این مسئله با **DoubleMetaphone** پیاده سازی شده است

مثال واقعی – twitter تبدیل کلمات

```
public class TwitterAnalyzer extends Analyzer {
  private DoubleMetaphone filter = new DoubleMetaphone();
  @Override
  public TokenStream tokenStream(String fieldName, Reader reader) {
    final TokenStream result = new PorterStemFilter(new StopFilter(
        true, new StandardTokenizer(Version.LUCENE_CURRENT, reader),
        StandardAnalyzer.STOP_WORDS_SET));

    TermAttribute termAtt = (TermAttribute) result
        .addAttribute(TermAttribute.class);
    StringBuilder buf = new StringBuilder();
    try {
      while (result.incrementToken()) {
        String word = new String(termAtt.termBuffer(), 0, termAtt
            .termLength());
        buf.append(filter.encode(word)).append(" ");
      }
    } catch (IOException e) {
      e.printStackTrace();
    }
    return new WhitespaceTokenizer(new StringReader(buf.toString()));
  }
}
```

Initialize encoder

Create phonetic root

مثال واقعی - ورودی

- **یافتن کاربران مشابه در Twitter**

- استفاده از *k-means* برای خوشه بندی کاربران