Laskennallisesti Älykkäät Järjestelmät

Sumean kmeans ja kmeans –algoritmien vertailu

Annemari Auvinen (annauvi@st.jyu.fi)
Anu Niemi (anniemi@st.jyu.fi)
28.5.2002

1 Tehtävän kuvaus

Tehtävänämme oli verrata kmeans- ja sumea kmeans –algoritmeja toisiinsa. Kieleksi valitsimme Javan, joka tosin ei kovin hyvin sopinut näin jälkikäteen ajatellen tähän tehtävään.

Testasimme algoritmit seuraavasti:
1. muodostetaan satunnaisia normaalijakautuneita ryppäitä erilaisilla keskiarvoilla, varianssina 2.5
2. ajetaan k-means ja saadaan keskiarvot
3. ajetaan sumea k-means ja saadaan keskiarvot
4. lasketaan menetelmien tuottamat virheet (kohdassa 1 olevan keskipisteen ja saadun keskipisteen etäisyys) ja talletetaan ne
5. kohtaan 1

Silmukkaa kierretään m kertaa. Lopuksi tulostetaan menetelmien antamat virheet kierroksittain kaikille keskipisteille.

2 Koodit

2.1 Kmeans

```
import java.util.*;
import java.lang.*;
import java.lang.Math;
import java.lang.Integer;


/**
 * Title:        Kmeans.java
 * Description:  Class for  kmeans-algorithm.
 * Copyright:    Annemari Auvinen, Anu Niemi (c) 2002
 * Company:
 * @author       Annemari Auvinen, Anu Niemi
 * @version      1.0
 *
 * @param   data        Vector for datapoints
 * @param   newW        Vector for midpoints.
 * @param   w           Vector for the values of midpoints.
 * @param   k           Int number of clusters
 * @param   epsilon     Double value of epsilon.
 * @param   classedData Vector for classified data.
 */

public class Kmeans {

  private Vector data = new Vector();
  private Vector w = new Vector();
  private Vector newW = new Vector();
  private Vector classedData = new Vector();
  private int k;
  private double epsilon = 0.1;

/**
  * Returns vector, which includes the midpoints sorted by descending order
based on
  * value of x.
  * @return       Vector of midpoints
  */
  public Vector getW()
  {
    ArrangeVector arranged = new ArrangeVector();
    Vector classes = arranged.arrange(w);
    return classes;
  }

/**
  * Sets the epsilon.
  * @param    epsilon    Int value of epsilon
  */
  public void setEpsilon(double value)
  {
    epsilon = value;
  }
/**
  * Sets the number of clusters.
  * @param    value    Int value of clusters
  */
  public void setK(int value)
  {
    k = value;
  }

/**
  * Sets the w vector.
  */
  public void setWVector(){
    int points = data.size();
    int ii;
    Vector used = new Vector();
    Random random = new Random();
    Integer index = new Integer(0);

    for (int i = 0; i<k; i++)
    {
      do{
        ii = random.nextInt(points);
        index = new Integer(ii);
      }while(used.contains(index));

      used.add(index);
      w.add(data.elementAt(ii));
    }
   }

/**
  * Creates the datavectors where data will be classed.
  */
  public void createDataVectors()
  {
    for (int i = 0; i<k; i++)
    {
      classedData.add(new Vector());
    }
  }

/**
  * Constructor for Kmeans.
```

```
   * @param data        Vector of datapoints
   * @param k           Int number of clusters
   */
   public Kmeans(Vector data, int k ) {
     this.data = data;
     this.k = k;
     setWVector();
     createDataVectors();
     classData();
     do {
       avarages();
       boolean end =testEnd();
       if (end)
       {
          break;
       }
       else {
         w.removeAllElements();
         for (int i=0; i<k; i++)
         {
           Observation h = (Observation)newW.elementAt(i);
           w.add(h);
         }
         newW.removeAllElements();
         classedData.removeAllElements();
         createDataVectors();
         classData();
       }
     } while (true);

   }

  /**
   * Classifies the data
   */
   public void classData()
   {
     double x;
     double y;

     double wx;
     double wy;

     double dist;  //distance
     int index = 0;

     for (int i=0; i<data.size(); i++)
     {
       dist = -1;
       Observation dataPoint = (Observation)data.elementAt(i);
   x = dataPoint.getX();
   y = dataPoint.getY();

   for (int j=0; j<k; j++)
   {
    Observation ww = (Observation)w.elementAt(j);
    wx = ww.getX();
    wy = ww.getY();
    double testDist;
    testDist = Math.sqrt( (Math.pow( (x-wx), 2.0) + Math.pow( (y-wy), 2.0) ) );

    if (dist < 0)
     {
      dist = testDist;
      index = j;
     }

    if (dist > testDist)
    {
     dist =testDist;
     index = j;
    }
   }

   Vector vector = (Vector)classedData.elementAt(index);
   vector.add(dataPoint);
  }

 }

/**
 * Calculates the avarages. Avarages are stored in the vector newW.
 */
 public void avarages()
 {
   double x = 0.0;
   double y = 0.0;

   for(int i=0; i<k; i++)
   {
    x = 0.0;
    y = 0.0;
    Vector classedPoints = (Vector)classedData.elementAt(i);
    int dataPoints = classedPoints.size();

    //X-value
    for (int j=0; j<dataPoints; j++)
    {
     Observation point = (Observation)classedPoints.elementAt(j);
     x = x + point.getX();
```

```
       }
       x = x / dataPoints;

       //y-value
       for (int j=0; j<dataPoints; j++)
       {
         Observation point = (Observation)classedPoints.elementAt(j);
         y = y + point.getY();
       }
       y = y / dataPoints;
      Observation o = new Observation(x,y);
      newW.add(o);
     }
  }


/**
  * Tests whether there is enoough iterations.
  *@return   true if the distance between the elements of w and newW vectors
is
  *         less than epsilon, otherwise false
  */
  public boolean testEnd()
  {
    for(int i = 0; i<k; i++ )
    {
       Observation newWvalue = (Observation)newW.elementAt(i);
       Observation oldWvalue = (Observation)w.elementAt(i);
       double testDist;
       testDist = Math.sqrt( (Math.pow( (oldWvalue.getX()-newWvalue.getX()),
2.0) + Math.pow( (oldWvalue.getY()- newWvalue.getY()), 2.0) ) );
       if (testDist > epsilon )
       {
         return false;
       }
    }
    return true;
  }

}
```

2.2 Sumea kmeans

```
/**
 * Title:       FuzzyKmeans.java
 * Description:  Class for fuzzy kmeans-algorithm.
 * Copyright:    Annemari Auvinen, Anu Niemi (c) 2002
 * Company:
 * @author      Annemari Auvinen, Anu Niemi
 * @version      1.0
 *
 * @param   datapoints        Vector for datapoints
 * @param   midpoints         Vector for midpoints.
 * @param   oldMidPoints      Vector for the values of old midpoints.
 * @param   membFunctions     Hashtable for values of member functions.
 * @param   oldMembFuctions   Hashtable for old values of member
functions.
 * @param   clusters          Int number of clusters
 * @param   b                 Int width parameter.
 * @param   epsilon           Double value of epsilon.
 * @param   change            Double value of change between iterations.
 */

import java.util.*;
import java.lang.*;

public class FuzzyKmeans {
    private Vector datapoints= new Vector();
    private Vector midpoints = new Vector();
    private Vector oldMidpoints = new Vector();
    private Hashtable membFunctions = new Hashtable();
    private Hashtable oldMembFunctions = new Hashtable();
    private int clusters;
    private int b;
    private double epsilon;
    private double change;

/**
  * Constructor for FuzzyKmeans.
  * @param datapoints  Vector of datapoints
  * @param k           Int number of clusters
  * @param b           Int width parameter
  * @param epsilon     Double value of epsilon
  */
  public FuzzyKmeans(Vector datapoints, int k, int b, double epsilon) {
    this.datapoints = datapoints;
    this.clusters = k;
    this.b = b;
    this.epsilon = epsilon;
  }

/**
  * Returns vector, which includes the midpoints sorted by descending order
based on
  * value of x.
  * @return   vector of midpoints
  */
  public Vector getMidpoints() {
    ArrangeVector arranged = new ArrangeVector();
    Vector classes = arranged.arrange(midpoints);
   return classes;
  }

/**
  * Initializes the midpoints and member function values.
  */
  public void initialize()
  {
     Random randomValue = new Random();
     //includes the indexes of selected points
     Vector takenPoints = new Vector();
     Observation mp;
     Integer index;
     int value, nmb,i,j;
     nmb = datapoints.size();
     //initializes the random selected datapoints to midpoints, same points is
selected only once
     for ( i=0; i<clusters; i++)  {
       do {
         value  =  randomValue.nextInt(nmb);
         index = new Integer(value);
       } while(takenPoints.contains(index));
       mp = (Observation)datapoints.elementAt(value);
       takenPoints.add(index);
       midpoints.add(mp);
       for ( j=0; j<nmb; j++) {
         Membership ms = new Membership(0.0);
         String place = i+"_"+j;
         membFunctions.put(place,ms);
       }
    }
    change = epsilon + 1; //condition for ending
  }

/** Calculates the midpoints.
  */
  public void calculateMidpoints()
  {
    double numeratorX;
    double numeratorY;
    double denominator;
    int nmb = datapoints.size();
    oldMidpoints.removeAllElements();
    //puts the old midpoints to vector
    for (int l=0; l<clusters;l++) {
      Observation o = (Observation)midpoints.elementAt(l);
      oldMidpoints.add(o);
    }
    midpoints.removeAllElements();
    for (int i=0; i<clusters;i++) {
      numeratorX = 0;
      numeratorY = 0;
      denominator = 0;
      for (int j=0; j<nmb;j++) {
        String place = i+"_"+j;
        Membership ms = (Membership)membFunctions.get(place);
        Observation o= (Observation)datapoints.elementAt(j);
        double  factor = Math.pow(ms.getValue(),b);
        numeratorX +=  factor*o.getX();
        numeratorY += factor* o.getY();
        denominator += factor;
      }
      Observation obs;
      if(denominator != 0) {
         obs = new Observation(numeratorX/denominator,
numeratorY/denominator);

      } else {
        obs = (Observation)oldMidpoints.elementAt(i);
      }
       midpoints.add(obs);
    }
  }

/**
  *  Calculates the values of member functions
  */
  public void calculateMembFunctions()
  {
    int nmb,i;
    String place;
    Membership ms;
    double midX, midY,X,Y, distance, numerator, denominator,
power,midX2,midY2;
    nmb = datapoints.size();
    //puts the old values of member functions to table
    for( i=0; i<clusters;i++)
    {
      for (int l=0;l<nmb;l++) {
```

```
      place = i+"_"+l;
      Membership member = (Membership)membFunctions.get(place);
      oldMembFunctions.put(place,member);
     }
    }
    for( i=0; i<clusters;i++)
    {
      Observation mp = (Observation)midpoints.elementAt(i);
      midX = mp.getX();
      midY = mp.getY();
      //calculates numerator
      for( int j=0;j<nmb;j++)
      {
        Observation point = (Observation)datapoints.elementAt(j);
        X = point.getX();
        Y = point.getY();
        distance = Math.pow((X-midX),2.0)+Math.pow((Y-midY),2.0);
        if(distance == 0)
          distance = 0.000001;
        power = 1.0/(b-1);
        numerator = Math.pow((1.0/distance), power);
        denominator = 0.0;
        //calculates denominator
        for (int k=0; k<clusters; k++)
        {
          Observation o = (Observation)midpoints.elementAt(k);
          midX2=o.getX();
          midY2=o.getY();
          distance = Math.pow((X-midX2),2.0)+Math.pow((Y-midY2),2.0);
          if(distance == 0)
            distance = 0.000001;
          denominator += Math.pow((1.0/distance),power);
        }
        if(denominator != 0) {
           ms = new Membership(numerator/denominator);
        } else {
           ms = new Membership(0.0);
        }
          place = i+"_"+j;
          membFunctions.put(place,ms);
      }
    }
  }

/**
  * Caluculates the midpoints and values of member functions until the
changes of midpoints or
  * values of member functions are bigger than epsilon
  */
  public void iterate() {
    int nmb;
    String place;
    double membValue,oldValue,biggest,help,distance;
    biggest = 0.0;
    help = 0.0;
    nmb = datapoints.size();
    initialize();
    while (change > epsilon)
    {
      biggest = 0.0;
      calculateMidpoints();
      calculateMembFunctions();
      for(int i=0; i<clusters;i++)
      {
         Observation newMp = (Observation)midpoints.elementAt(i);
         Observation oldMp = (Observation)oldMidpoints.elementAt(i);
         distance = Math.sqrt(Math.pow((newMp.getX()-
oldMp.getX()),2.0)+Math.pow((newMp.getY()-oldMp.getY()),2.0));
        for (int j=0; j<nmb; j++)
        {
          place = i+"_"+j;
          Membership ms = (Membership)membFunctions.get(place);
          membValue = ms.getValue();
          Membership ms2 = (Membership)oldMembFunctions.get(place);
          oldValue = ms2.getValue();
          help = Math.abs(membValue-oldValue);

          if(help > biggest) {
            biggest = help;
          }
        }
       if(distance > biggest){
         biggest = distance;
       }
      }

      change = biggest;

    }

  }

}
```

3 Testituloksia

3.1 Luokkia 2, pisteitä 2500/rypäs, leveysparametri 2, kierroksia 5

Kierros 1 keskipisteessä 1 tavallisen virhe 0.8328990275329275 ja sumean virhe 0.5728476707706995
Kierros 1 keskipisteessä 2 tavallisen virhe 0.7029763405333377 ja sumean virhe 0.6538848105069173

Kierros 2 keskipisteessä 1 tavallisen virhe 0.16716220673973625 ja sumean virhe 0.21192846059053422
Kierros 2 keskipisteessä 2 tavallisen virhe 0.3728240169397198 ja sumean virhe 0.29401598375787324

Kierros 3 keskipisteessä 1 tavallisen virhe 0.3789161207747754 ja sumean virhe 0.4015364135450447
Kierros 3 keskipisteessä 2 tavallisen virhe 0.47247520787011693 ja sumean virhe 0.283551375925339

Kierros 4 keskipisteessä 1 tavallisen virhe 0.83624956638595 ja sumean virhe 0.6604083683778255
Kierros 4 keskipisteessä 2 tavallisen virhe 1.0720859076965745 ja sumean virhe 0.7656651049863258

Kierros 5 keskipisteessä 1 tavallisen virhe 1.9635608873331691 ja sumean virhe 1.076275477637058
Kierros 5 keskipisteessä 2 tavallisen virhe 1.6697438042399304 ja sumean virhe 1.134964125925128

Toinen testi:

Kierros 1 keskipisteessä 1 tavallisen virhe 0.11725511653463881 ja sumean virhe 0.030343451936421104
Kierros 1 keskipisteessä 2 tavallisen virhe 0.0352301214434908 ja sumean virhe 0.05871656103150916

Kierros 2 keskipisteessä 1 tavallisen virhe 0.38424174088299984 ja sumean virhe 0.3995734436282096
Kierros 2 keskipisteessä 2 tavallisen virhe 0.643572402552616 ja sumean virhe 0.4215299588347693

Kierros 3 keskipisteessä 1 tavallisen virhe 0.6197476073367011 ja sumean virhe 0.5920175443326129
Kierros 3 keskipisteessä 2 tavallisen virhe 0.8921807868982677 ja sumean virhe 0.6186333715897095

Kierros 4 keskipisteessä 1 tavallisen virhe 1.3661560082386255 ja sumean virhe 0.9262651693899763

Kierros 4 keskipisteessä 2 tavallisen virhe 1.18329980227418 ja sumean virhe 1.042081640712577

Kierros 5 keskipisteessä 1 tavallisen virhe 2.0190765293601256 ja sumean virhe 2.3477621659350425

Kierros 5 keskipisteessä 2 tavallisen virhe 2.607813582292546 ja sumean virhe 2.403322185134944

3.2 Luokkia 4, pisteitä 2500/rypäs, leveysparametri 2, kierroksia 5

Kierros 1 keskipisteessä 1 tavallisen virhe 2.7568506883984703 ja sumean virhe 2.663553834196809

Kierros 1 keskipisteessä 2 tavallisen virhe 3.3528814844441492 ja sumean virhe 3.0827368270487385

Kierros 1 keskipisteessä 3 tavallisen virhe 5.533338118183551 ja sumean virhe 5.0562785042656415

Kierros 1 keskipisteessä 4 tavallisen virhe 3.1659013692975666 ja sumean virhe 3.9135720486300203

Kierros 2 keskipisteessä 1 tavallisen virhe 3.410411198945593 ja sumean virhe 0.8234522559629306

Kierros 2 keskipisteessä 2 tavallisen virhe 4.784796678835296 ja sumean virhe 0.8620077921347632

Kierros 2 keskipisteessä 3 tavallisen virhe 2.0268642715473058 ja sumean virhe 0.6931095916801053

Kierros 2 keskipisteessä 4 tavallisen virhe 0.36821172465115426 ja sumean virhe 0.6497784912113534

Kierros 3 keskipisteessä 1 tavallisen virhe 5.250867152613417 ja sumean virhe 0.5695661280396731

Kierros 3 keskipisteessä 2 tavallisen virhe 6.026159109270971 ja sumean virhe 0.36153761824100566

Kierros 3 keskipisteessä 3 tavallisen virhe 3.417438775285456 ja sumean virhe 0.5176160608290125

Kierros 3 keskipisteessä 4 tavallisen virhe 0.9970051660104776 ja sumean virhe 0.6105892375322048

Kierros 4 keskipisteessä 1 tavallisen virhe 1.8700753486145032 ja sumean virhe 0.7595388449335759

Kierros 4 keskipisteessä 2 tavallisen virhe 7.138074237880796 ja sumean virhe 1.7655985818341426

Kierros 4 keskipisteessä 3 tavallisen virhe 0.8194750386568503 ja sumean virhe 6.054028259799661

Kierros 4 keskipisteessä 4 tavallisen virhe 1.5383548758552597 ja sumean virhe 6.325420409870378

Kierros 5 keskipisteessä 1 tavallisen virhe 1.8099635423800982 ja sumean virhe 2.500793804681536

```
Kierros 5 keskipisteessä 2 tavallisen virhe 5.515305578215164 ja sumean virhe 4.485554640283862
Kierros 5 keskipisteessä 3 tavallisen virhe 5.199031128062355 ja sumean virhe 4.365444467775055
Kierros 5 keskipisteessä 4 tavallisen virhe 2.1904047123771897 ja sumean virhe 2.531930826632833
```