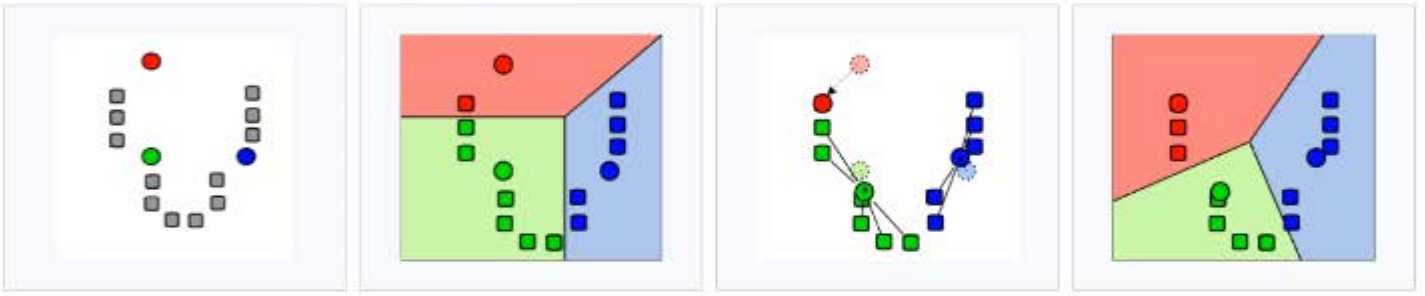
package poo.clustering

Premessa – Clustering K-Means

L'operazione di clustering, nell'ambito delle tecniche non supervisionate di machine learning, mira a raggruppare i dati di un dataset, in modo che i dati appartenenti ad uno stesso cluster (gruppo) siano più "affini" o "simili" tra loro, rispetto ai dati di altri cluster. L'algoritmo K-Means itera un certo numero di passi rivedendo ad ogni passo l'appartenenza dei dati ai cluster.

E' possibile un'interpretazione geometrica del modo di operare di K-Means. I dati sono visti come punti es. del piano (più in generale di uno spazio ad n dimensioni), ossia dispongono di due componenti corrispondenti alle due coordinate <X,Y>. L'obiettivo è raggruppare i punti secondo una certa metrica, ad es. basata sulla distanza Euclidea di ogni punto da un punto centrale del cluster detto centroide.



K-Means assume inizialmente di conoscere il numero K dei cluster in cui verosimilmente il data set è decomponibile. I K centroidi sono inizializzati in modo randomatico. L'algoritmo quindi ripete una successione di fasi in cui vengono iterati i seguenti passi fondamentali:

1. si determinano le distanze di ciascun punto da ciascun centroide;
2. ogni punto è quindi associato al cluster rispetto al quale ha minima distanza dal centroide;
3. se almeno un punto ha cambiato il suo cluster di appartenenza, si ritorna al passo 1, dopo aver ridefinito le posizioni dei centroidi come punti medi di ciascun cluster; altrimenti si è raggiunta la convergenza e l'algoritmo termina.

E' possibile che dopo ciascuna fase non si raggiunga la convergenza. In questi casi, occorre definire un limite al numero delle iterazioni e l'algoritmo termina quando tale numero è stato raggiunto.

Prima parte

Scrivere una classe Java di utilità **DataSet** che ammetta (almeno) un metodo *crea()* che riceve il nome esterno (es. mediante un oggetto File) di un file di tipo testo, e tre costanti intere N, MIN e MAX, e genera un data set costituito da N punti del piano, le cui coordinate (double) sono determinate randomaticamente come segue:

```
X=Math.random()*(MAX-MIN)+MIN;
Y=Math.random()*(MAX-MIN)+MIN;
```

I valori di X e Y di un punto, separati da uno o più spazi, sono posti su una stessa linea del file.

Scrivere una classe di supporto **Punto** (immutabile) destinata a memorizzare le coordinate <X,Y> di un punto del data set, e dotata dei metodi getter nonché i metodi canonici *equals*(), *hashCode*() e *toString*(). La classe **Punto** deve esportare anche i metodi:
double distanza(Punto p) che calcola e ritorna la distanza Euclidea tra this e p, e *Punto puntoMedio(Punto p)*, che determina e ritorna il punto medio tra this e p.

Scrivere quindi (la prima parte di) una classe **KMeans** che implementi l'algoritmo K-Means, **dettagliando inizialmente il suo costruttore**. Il costruttore deve ricevere:

- un file di tipo testo contenente il data set
- un intero K che esprime il numero di centroidi da utilizzare
- un secondo file di tipo testo contenente i K punti centroidi iniziali.

I punti centroidi vanno caricati su un array **centroidi** di K punti. I punti del data set, invece, vanno memorizzati su una mappa **dataSet** <Punto,Integer> in cui la chiave è un punto del data set, il valore (inizialmente un negativo es. -1) è un indice sull'array **centroidi** che specifica il cluster di appartenenza del punto.

package poo.clustering

Seconda parte

Aggiungere alla classe **KMeans** il metodo fondamentale:

void run(int maxIte)

che riceve il numero massimo delle iterazioni previste, e avvia l'algoritmo K—Means.

Ad ogni iterazione, di ogni punto del **dataSet** si determina la distanza dai centroidi correnti. L'indice del punto centroide avente minima distanza dal punto del data set, va posto come valore corrispondente sulla mappa. Una variabile boolean di **KMeans**, **terminazione**, posta a true (ottimismo) all'inizio di ogni iterazione, va cambiata a false non appena si scopre che il valore di indice di centroide di un almeno un punto cambia rispetto al valore precedente.

Se dopo un'iterazione occorre effettuarne un'altra, prima si deve aggiornare l'array dei **centroidi** definendo i nuovi centroidi come punti medi dei gruppi di punti del data set che sono emersi come cluster nell'ultima fase.

Dotare la classe **KMeans** del metodo *toString()* che deve ritornare i punti **centroidi** finali identificati, ed il contenuto della mappa **dataSet** che mostra, per ogni punto del data set, l'indice del centroide cui è associato.

Aggiungere, infine, alla classe **KMeans** un metodo main() di test, che costruisca inizialmente un data set sul file es. "c:\\poo-file\\dataset.txt", utilizzando le costanti N=1000, MIN=500 e MAX=1500. Nello stesso main, creare un secondo file di tipo testo "c:\\poo-file\\centroidi.txt" contenente N=4 punti (il valore di K) e utilizzando le stesse costanti MIN e MAX usate per generare il data set.

DataSet

```java
package poo.clustering;

import java.io.BufferedReader;

public final class DataSet {
    private DataSet() {}
    public static void crea( File f, int N, int MIN, int MAX ) throws IOException{
        if( N<=0 ) throw new IllegalArgumentException();
        PrintWriter pw=new PrintWriter(f);
        for( int i=0; i<N; ++i ) {
            double X=Math.random()*(MAX-MIN)+MIN;
            double Y=Math.random()*(MAX-MIN)+MIN;
            pw.println(String.format("%1.3f",X)+" "+String.format("%1.3f",Y));
        }
        pw.close();
    }//crea
}//DataSet
```

Punto

```java
package poo.clustering;

public class Punto {
    private final double X, Y;
    public Punto( double X, double Y ) { this.X=X; this.Y=Y; }
    public double getX() { return X; }
    public double getY() { return Y; }
    public double distanza( Punto p ) {
        return Math.sqrt((this.X-p.X)*(this.X-p.X)+(this.Y-p.Y)*(this.Y-p.Y));
    }//distanza
    public Punto puntoMedio( Punto p ) {
        double x=(this.X+p.X)/2;
        double y=(this.Y+p.Y)/2;
        return new Punto(x,y);
    }
    public boolean equals( Object o ) {
        if( !(o instanceof Punto) ) return false;
        if( o==this ) return true;
        Punto p=(Punto)o;
        return this.X==p.X && this.Y==p.Y;
    }
    public int hashCode() {
        return Double.valueOf(X).hashCode()*83+Double.valueOf(Y).hashCode();
    }
    public String toString() {
        return "Punto<"+X+","+Y+">";
    }
    public static void main( String[] args ) {
        Punto p=new Punto(5,8);
        System.out.println(p);
    }
}//Punto
```

KMeans #1

```
package poo.clustering;

import java.io.BufferedReader;

public class KMeans {

    private Punto[] centroidi;
    private Map<Punto,Integer> dataSet=new HashMap<>();
    private int k;
    private boolean terminazione;

    public KMeans( File ds, int k, File c ) throws IOException{
        if( k<=0 ) throw new IllegalArgumentException();
        this.k=k;
        centroidi=new Punto[k];
        BufferedReader br=new BufferedReader( new FileReader(c) );
        int i=0;
        try {
            for(;;) {
                String linea=br.readLine();
                if( linea==null ) break;
                String[] coord=linea.split("\\s+");
                if( coord.length!=2 ) throw new IllegalArgumentException("Bad centroid file.");
                if( i>=k ) throw new IllegalArgumentException("Bad centroid file.");
                centroidi[i]=new Punto( Double.valueOf(coord[0]), Double.valueOf(coord[1]));
                i++;
            }
        }finally {
            if( i!=k ) throw new IllegalArgumentException("Bad centroid file.");
            br.close();
        }
        br=new BufferedReader( new FileReader(ds) );
        try {
            for(;;) {
                String linea=br.readLine();
                if( linea==null ) break;
                String[] coord=linea.split("\\s+");
                if( coord.length!=2 ) throw new IllegalArgumentException("Bad dataset file.");
                dataSet.put( new Punto( Double.valueOf(coord[0]), Double.valueOf(coord[1])));
            }
        }finally {
            br.close();
        }
    }
```

KMeans #2

```
public void run( int numIte ) throws IOException{
    if( numIte<=0 ) throw new IllegalArgumentException();
    int ni=numIte;
    do {
        terminazione=true;
        for( Punto p: dataSet.keySet() ) {
            int iC=-1; double dMin=Double.MAX_VALUE;
            for( int i=0; i<k; ++i ) {
                double d=p.distanza(centroidi[i]);
                if( d<dMin ) {
                    iC=i; dMin=d;
                }
            }
            if( dataSet.get(p)!=iC ) {
                terminazione=false;
                dataSet.put(p,iC);
            }
        }
        if( !terminazione ) {
            //aggiorna centroidi
            for( int i=0; i<k; ++i ) {
                Punto pm=null;
                for( Punto p: dataSet.keySet() ) {
                    if( dataSet.get(p)==i ) {
                        if( pm==null ) pm=p;
                        else pm=pm.puntoMedio(p);
                    }
                }
                centroidi[i]=pm;
            }
        }
        ni--;
    }while( !terminazione  && ni>0 );
    if( terminazione )
        System.out.println("Terminazione per convergenza. Iterazioni eseguite: "+ni);
    else
        System.out.println("Terminazione per raggiunto numero max di iterazioni.");
    File f=new File("c:\\poo-file\\centroidi-final.txt");
    PrintWriter pw=new PrintWriter( new FileWriter(f) );
    for( int i=0; i<k; ++i )
        pw.println(centroidi[i].getX()+" "+centroidi[i].getY());
    pw.close();
}//run
```

KMeans #3

```
    public String toString() {
        StringBuilder sb=new StringBuilder(1000);

        int[] dist=new int[k];
        for( Punto p: dataSet.keySet() )
            dist[dataSet.get(p)]++;

        for( int i=0; i<k; ++i ) {
            sb.append(i+": "+centroidi[i]+" cluster size: "+dist[i]);
            if( i<k-1 ) sb.append("\n");
        }
        sb.append("\n");
        for( Punto p:dataSet.keySet() ) {
            sb.append(p+" centroide: "+dataSet.get(p)+"\n");
        }
        return sb.toString();
    }

    public static void main( String[] args ) throws IOException{
        File ds=new File("c:\\poo-file\\dataset.txt");
        File c =new File("c:\\poo-file\\centroidi.txt");
        int k=5, N=1000;
        DataSet.crea(ds,N,500,1500);
        DataSet.crea(c,k,500,1500);
        KMeans km=new KMeans(ds,k,c);
        km.run(100);
        System.out.println("Situazione finale:");
        System.out.println(km);
    }

}//KMeans
```