K-Nearest Neighbours (KNN) for Simple Data

Zbigniew Dendzik
Institute of Physics, University of Silesia

2025

Introduction

The K-Nearest Neighbours (KNN) algorithm is one of the simplest supervised learning algorithms. The task: given a new data point (for example a point in 2D space), predict its class (label) by looking at the classes of its nearest neighbours in the training set.

In the simplest variant:

- we represent each example as a point in feature space (e.g. coordinates (x, y)),
- we define a distance metric (e.g. Euclidean distance),
- for a new point we find its K nearest neighbours in the training set,
- we predict the class based on majority voting among those neighbours.

KNN does not build a complex "model" – the training data themselves are the model, and computation happens at classification time (lazy learning). In this set we will implement:

- classes representing labelled points,
- a function computing Euclidean distance in 2D,
- a simple KNN classifier,
- a test program comparing results for different K values.

Example I – 2D point representation

We start with a `Point2D` class representing a point in 2D space with a label.

```
class Point2D
{
   private double x;
   private double y;
   private String label; //e.g. "A", "B"

   public Point2D(double x, double y, String label)
   {
      this.x=x;
```

```
10          this.y=y;
11          this.label=label;
12      }
13
14      public double getX()
15      {
16          return x;
17      }
18
19      public double getY()
20      {
21          return y;
22      }
23
24      public String getLabel()
25      {
26          return label;
27      }
28
29      public String toString()
30      {
31          return "Point2D("+x+", "+y+", "+label+")";
32      }
33  }
```

Listing 1: Point2D.java

We add a small utility class for Euclidean distance:

```
1   class Distance
2   {
3       public static double euclidean(Point2D a, Point2D b)
4       {
5           double dx=a.getX()-b.getX();
6           double dy=a.getY()-b.getY();
7           return Math.sqrt(dx*dx+dy*dy);
8       }
9   }
```

Listing 2: Distance.java

Ex. 3.1

Write a program `TestPoint2D` that:

- creates several `Point2D` objects belonging to two different classes (e.g. “A” and “B”),
- prints the coordinates and labels of each point,
- computes and prints the Euclidean distance between some pairs of points.

```
1   public class TestPoint2D
2   {
```

```
    public static void main(String[] args)
    {
        Point2D p1=new Point2D(1.0,2.0,"A");
        Point2D p2=new Point2D(2.0,3.0,"A");
        Point2D p3=new Point2D(4.0,5.0,"B");

        System.out.println(p1);
        System.out.println(p2);
        System.out.println(p3);

        double d12=Distance.euclidean(p1,p2);
        double d13=Distance.euclidean(p1,p3);

        System.out.println("d(p1,p2) = "+d12);
        System.out.println("d(p1,p3) = "+d13);
    }
}
```

Listing 3: TestPoint2D.java – skeleton

Example II – dataset and nearest neighbours

We define a `PointSet` class that stores training points and provides a method to find nearest neighbours of a query point.

```
import java.util.*;

class PointSet
{
    private List<Point2D> points;

    public PointSet()
    {
        points=new ArrayList<Point2D>();
    }

    public void add(Point2D p)
    {
        points.add(p);
    }

    public List<Point2D> getPoints()
    {
        return points;
    }

    static class PointDistance
    {
        Point2D point;
        double distance;

```

```
27          PointDistance(Point2D point, double distance)
28          {
29              this.point=point;
30              this.distance=distance;
31          }
32      }
33
34      public List<Point2D> nearestNeighbours(Point2D q)
35      {
36          List<PointDistance> list=
37              new ArrayList<PointDistance>();
38
39          for(Point2D p : points)
40          {
41              double d=Distance.euclidean(p,q);
42              list.add(new PointDistance(p,d));
43          }
44
45          Collections.sort(list,
46              new Comparator<PointDistance>()
47              {
48                  public int compare(PointDistance a,
49                                     PointDistance b)
50                  {
51                      return Double.compare(a.distance,
52                                            b.distance);
53                  }
54              });
55
56          List<Point2D> result=new ArrayList<Point2D>();
57          for(PointDistance pd : list)
58              result.add(pd.point);
59
60          return result;
61      }
62  }
```

Listing 4: PointSet.java – skeleton

Ex. 3.2

Write a program `TestPointSet` that:

- creates a `PointSet` and adds a dozen points from two classes (e.g. “A” and “B”),
- creates a query point `q` (without label),
- prints the list of points sorted by distance from `q`,
- prints the first K points (for a chosen K, e.g. K=3).

```java
import java.util.*;

public class TestPointSet
{
   public static void main(String[] args)
   {
      PointSet set=new PointSet();
      set.add(new Point2D(1,2,"A"));
      set.add(new Point2D(2,1,"A"));
      set.add(new Point2D(4,5,"B"));
      set.add(new Point2D(5,4,"B"));
      set.add(new Point2D(0,0,"A"));

      Point2D q=new Point2D(3,3,"?");

      List<Point2D> neigh=
         set.nearestNeighbours(q);

      System.out.println("Nearest neighbours of "+
         q.getX()+","+q.getY()+":");

      for(Point2D p : neigh)
      {
         double d=Distance.euclidean(p,q);
         System.out.println(p+" d="+d);
      }

      int K=3;
      System.out.println("First "+K+" neighbours:");
      for(int i=0;i<K && i<neigh.size();i++)
      {
         System.out.println("K="+(i+1)+": "+neigh.get(i));
      }
   }
}
```

Listing 5: TestPointSet.java – skeleton

Example III – KNN classifier

We define a `KNN` class that:

- stores a training `PointSet`,
- for a query point finds K nearest neighbours,
- performs majority voting on their labels.

```java
import java.util.*;

```

```
3  class KNN
4  {
5     private PointSet set;
6     private int K;
7
8     public KNN(PointSet set, int K)
9     {
10       this.set=set;
11       this.K=K;
12    }
13
14    public int getK()
15    {
16       return K;
17    }
18
19    public void setK(int K)
20    {
21       this.K=K;
22    }
23
24    public String classify(Point2D q)
25    {
26       List<Point2D> neigh=
27          set.nearestNeighbours(q);
28
29       Map<String,Integer> counts=
30          new HashMap<String,Integer>();
31
32       for(int i=0; i<K && i<neigh.size(); i++)
33       {
34          Point2D p=neigh.get(i);
35          String label=p.getLabel();
36          Integer c=counts.get(label);
37          if(c==null) c=0;
38          counts.put(label,c+1);
39       }
40
41       String bestLabel=null;
42       int bestVotes=-1;
43
44       for(String label : counts.keySet())
45       {
46          int votes=counts.get(label);
47          if(votes>bestVotes)
48          {
49             bestVotes=votes;
50             bestLabel=label;
51          }
52       }
53
```

```
54          return bestLabel;
55      }
56  }
```

Listing 6: KNN.java – skeleton

Ex. 3.3

Write a program `TestKNN` that:

- creates a training `PointSet` with a dozen points of classes “A” and “B”,
- creates a `KNN` object for chosen K values (e.g. K=1,3,5),
- for several test points (without labels) prints the predicted class,
- compares the results for different K values.

```
1   import java.util.*;
2
3   public class TestKNN
4   {
5       public static void main(String[] args)
6       {
7           PointSet set=new PointSet();
8           set.add(new Point2D(1,2,"A"));
9           set.add(new Point2D(1,1,"A"));
10          set.add(new Point2D(2,1,"A"));
11          set.add(new Point2D(4,5,"B"));
12          set.add(new Point2D(5,4,"B"));
13          set.add(new Point2D(6,5,"B"));
14
15          Point2D[] test=new Point2D[]
16          {
17              new Point2D(2,2,"?"),
18              new Point2D(4,4,"?"),
19              new Point2D(0,0,"?")
20          };
21
22          int[] Ks={1,3,5};
23
24          for(int K : Ks)
25          {
26              KNN knn=new KNN(set,K);
27              System.out.println("=== K="+K+" ===");
28              for(Point2D q : test)
29              {
30                  String label=knn.classify(q);
31                  System.out.println("Point "+q.getX()+","+
32                      q.getY()+" -> label: "+label);
33              }
34          }
```

```
    }
}
```

Listing 7: TestKNN.java – skeleton

Example IV – random data and accuracy evaluation

To better understand KNN behaviour we generate a simple random dataset: two classes arranged around two different centres. Then:

- we split data into training and test sets,
- we compute the accuracy of KNN for different K values.

```
import java.util.*;

class DataGenerator
{
   private Random rand=new Random();

   public Point2D randomPoint(double cx, double cy,
                              double spread,
                              String label)
   {
      double x=cx + rand.nextGaussian()*spread;
      double y=cy + rand.nextGaussian()*spread;
      return new Point2D(x,y,label);
   }

   public List<Point2D> generateCluster(int n,
                                        double cx,
                                        double cy,
                                        double spread,
                                        String label)
   {
      List<Point2D> list=new ArrayList<Point2D>();
      for(int i=0;i<n;i++)
      {
         list.add(randomPoint(cx,cy,spread,label));
      }
      return list;
   }
}
```

Listing 8: DataGenerator.java – skeleton

Ex. 3.4

Write a program `EvaluateKNN` that:

- generates two clusters of points:

  - class “A” around $(0, 0)$,
  - class “B” around $(5, 5)$,
- splits them into training and test sets (e.g. 70% train, 30% test),
- for several K values (e.g. 1,3,5,7) computes classification accuracy on the test set.

```java
import java.util.*;

public class EvaluateKNN
{
   public static void main(String[] args)
   {
      DataGenerator gen=new DataGenerator();

      List<Point2D> classA=
         gen.generateCluster(50,0.0,0.0,1.0,"A");
      List<Point2D> classB=
         gen.generateCluster(50,5.0,5.0,1.0,"B");

      List<Point2D> all=new ArrayList<Point2D>();
      all.addAll(classA);
      all.addAll(classB);

      Collections.shuffle(all);

      int n=all.size();
      int nTrain=(int)(0.7*n);

      PointSet train=new PointSet();
      List<Point2D> test=new ArrayList<Point2D>();

      for(int i=0;i<n;i++)
      {
         Point2D p=all.get(i);
         if(i<nTrain)
            train.add(p);
         else
            test.add(p);
      }

      int[] Ks={1,3,5,7};

      for(int K : Ks)
      {
         KNN knn=new KNN(train,K);
         int correct=0;

         for(Point2D q : test)
         {
            String pred=knn.classify(q);
```

```
45              if(pred.equals(q.getLabel()))
46                  correct++;
47          }
48
49          double accuracy=
50              (double)correct / (double)test.size();
51          System.out.println("K="+K+
52              " accuracy="+accuracy);
53      }
54  }
55  }
```

Listing 9: EvaluateKNN.java – skeleton

Ex. 3.5

Think about:

- what happens when K is too small (e.g. K=1),
- what happens when K is too large (comparable to the whole dataset size),
- how the distribution of points in space affects KNN behaviour.

Compare this with more advanced models (e.g. large language models) which instead of simple numeric coordinates use embedding vectors and more complex decision mechanisms.