Instance Based Learning (K Nearest Neighbors)

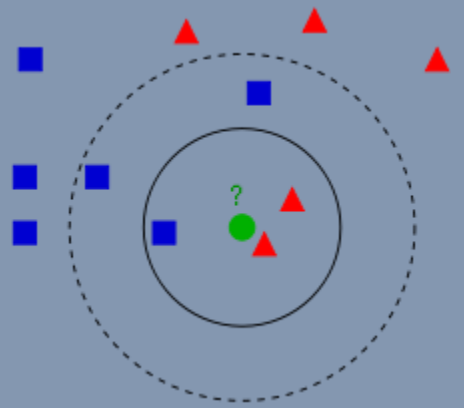
Payson Lippert

Introduction

- What is kNN?
- My implementation
- Improvements
- Examples in the real world

What is KNN?

- Finds the k nearest of instances and chooses the most common data class from the k entries.



History

- First used in the 1960’s
- Diagnosis in 1980’s

My Implementation

```
Enter the file name with extension : Test.txt

Enter the number of neighbors : 5
Classified as: 1.0, Actual: 2.0
Classified as: 2.0, Actual: 2.0
Classified as: 2.0, Actual: 1.0
```

My Implementation

```
public static class DataEntry{
    private ArrayList<Double> data;
    private Object result;

    public DataEntry(ArrayList<Double> x, Object y){
        data = x;
        result = y;
    }

        public ArrayList<Double> getX(){
            return data;
        }

        public Object getY(){
            return result;
        }
    }
}
```

My Implementation

```
    private static double convertDistance(double d){
        return 1.0/d;
    }

    /**
    * Computes Euclidean distance
    * @param a From
    * @param b To
    * @return Distance
    */
    public static double distance(DataEntry a, DataEntry b){
        double distance = 0.0;
        int length = a.getX().size();
        for(int i = 0; i < length; i++){
            double t = a.getX().get(i)-b.getX().get(i);
            distance = distance+t*t;
        }
        return Math.sqrt(distance);
    }
```

My Implementation

```
private DataEntry[] getNearestNeighbors(DataEntry x){
    //Set classes
    DataEntry[] nearestType = new DataEntry[this.numberOfNeighbors];
    double nearest = Double.MIN_VALUE;
    int index = 0;
    for(DataEntry tempEntry : this.dataSet){
        //Euclidean distance for every other element in the set
        double distance = distance(x,tempEntry);
        //initialize neighbors
        if(nearestType[nearestType.length-1] == null){
            int j = 0;
            while(j < nearestType.length){
                if(nearestType[j] == null){
                    nearestType[j] = tempEntry; break;
                }
                j++;
            }
            //Find the nearest data class
            if(distance > nearest){
                index = j;
                nearest = distance;
            }
        }
        else{
            //if there is a closer class
            if(distance < nearest){
                nearestType[index] = tempEntry;
                double f = 0.0;
                int ind = 0;
                for(int j = 0; j < nearestType.length; j++){
                    double tempDist = distance(nearestType[j],x);
                    if(tempDist > f){
                        f = tempDist;
                        ind = j;
                    }
                }
                nearest = f;
```

My Implementation

```java
/**
 * Actually performs the classification of the given data.
 * @param e Entry to be classified
 * @return The class of the most probable class
 */
public Object classify(DataEntry e){
    HashMap<Object,Double> classcount = new HashMap<Object,Double>();
    //Get the classes
    DataEntry[] entry = this.getNearestNeighbors(e);

    //Initialize hash map of the data values
    for(int i = 0; i < entry.length; i++){

        double distance = NearestNeighbour.convertDistance(NearestNeighbour.distance(entry[i], e));
        if(!classcount.containsKey(entry[i].getY())){
            classcount.put(entry[i].getY(), distance);
        }
        else{
            classcount.put(entry[i].getY(), classcount.get(entry[i].getY())+distance);
        }
    }
    //Find right choice
    Object o = null;
    double max = 0;
    for(Object ob : classcount.keySet()){
        if(classcount.get(ob) > max){
            max = classcount.get(ob);
            o = ob;
        }
    }

    return o;
}
```

My Implementation

```
Classified as: 1.0, Actual: 1.0
Classified as: 2.0, Actual: 1.0
Classified as: 2.0, Actual: 2.0
Classified as: 2.0, Actual: 2.0
Classified as: 1.0, Actual: 1.0
Classified as: 2.0, Actual: 1.0
Classified as: 1.0, Actual: 2.0
Classified as: 1.0, Actual: 1.0
Classified as: 1.0, Actual: 1.0
Classified as: 1.0, Actual: 1.0
67.93% accuracy
```

Results

- DeerHunter: 67%-70%

Improvements

- Distance calculation
- Weight dimensions
- Choose examples to add to data set
- Fuzzy logic

Examples

- 2011 bankruptcy prediction: 81%
- Image recognition
- Tumor identification

Summary

- Simple classification method
- Distance measurement most important aspect
- Can be customized to data sets
- Commonly used in a range of applications