Notes 03-2: Distance-Based Classification

Each item that is mapped to a class may be thought of as being more similar to the other items in that class than it is to the items in other classes. Distance measures can be used to quantify the similarity of different items.

Distance Measures

A distance measure is

The best known distance measures are Euclidean distance and Manhattan distance. According to the *Euclidean distance* (or *straight line distance*), for some arbitrary *m*-dimensional instance p_i described by (p_{i1}, p_{i2}, …, p_{im}), where p_{iu} denotes the value of the *u*-th attribute of p_i, then the difference between two instances p_i and p_j is defined as

$$d(p_i, p_j) = \sqrt{\sum_{u=1}^{m} (p_{iu} - p_{ju})^2}$$

An alternative distance measure is the *Manhattan distance*.

$$d(p_i, p_j) = \sum_{u=1}^{m} |p_{iu} - p_{ju}|$$

k-Nearest Neighbour Technique

The k-nearest neighbour technique assumes instances can be represented by points in a Euclidean space.

The nearest neighbours of an instance are defined in terms of the standard Euclidean distance.

Example – General idea behind k-nearest neighbour classification

EXAMPLE = Classification.D.1.a

A *k*-nearest neighbour classifier

```
Algorithm: KNN
Input: D = a set of points (i.e., instances) in Euclidean space
       of the form ⟨p1, p2, … pn⟩
       k = the number of neighbours to consider
       p′ = an unlabeled instance
Output: CKNN = the class label

Method:
1.   j = 0
2.   for each p ∈ D
3.       j ++
4.       if k == 1
5.           if d (p, p′) <= NN [1].distance
6.               NN [1].distance = d (p, p′)
7.               NN [1].class = class (p)
8.       else if j <= k
9.           i = j
10.          while i > 1 and d (p, p′) <= NN [i – 1].distance
11.              NN [i].distance = NN [i – 1].distance
12.              NN [i].class = NN [i – 1].class
13.              i --
14.          NN [i].distance = d (p, p′)
15.          NN [i].class = class (p)
16.      else
17.          i = k + 1
18.          while i > 1 and d (p, p′) <= NN [i – 1].distance
19.              NN [i].distance = NN [i – 1].distance
20.              NN [i].class = NN [i – 1].class
21.              i --
22.          NN [i].distance = d (p, p′)
23.          NN [i].class = class (p)
24.  c = 1
25.  class [c].class = NN [1].class
26.  class [c].count = 1
27.  for j = 2 to k
28.      newClass = true
29.      for i = 1 to c
30.          if class [i].class == NN [j].class
31.              class [i].count ++
32.              newClass = false
33.              break
34.      if newClass
35.          c ++
36.          class [c].class = NN [j].class
37.          class [c].count = 1
```

```
38.  CKNN = class [1].class
39.  count = class [1].count
40.  for i = 2 to c
41.      if class [c].count >= count
42.          CKNN = class [i].class
43.          count = class [i].count
```

Example – Predicting a class label using KNN

Tuple	Gender	Height	Class
t_1	*f*	1.6	*short*
t_2	*m*	2.0	*tall*
t_3	*f*	1.9	*medium*
t_4	*f*	1.8	*medium*
t_5	*f*	1.7	*short*
t_6	*m*	1.85	*medium*
t_7	*f*	1.6	*short*
t_8	*m*	1.7	*short*
t_9	*m*	2.2	*tall*
t_{10}	*m*	2.1	*tall*
t_{11}	*f*	1.8	*medium*
t_{12}	*m*	1.95	*medium*
t_{13}	*f*	1.9	*medium*
t_{14}	*f*	1.8	*medium*
t_{15}	*f*	1.75	*medium*

The class label attribute is Class and has three unique values: *short*, *medium*, and *tall*.

The unlabeled instance to be classified is $\langle t_{16}, f, 1.6, ? \rangle$.

Assume k = 5. The five nearest neighbors to $\langle t_{16}, f, 1.6, ? \rangle$ are $\langle t_1, f, 1.6, short \rangle$, $\langle t_5, f, 1.7, short \rangle$, $\langle t_7, f, 1.6, short \rangle$, $\langle t_8, m, 1.7, short \rangle$, and $\langle t_{15}, f, 1.6, short \rangle$. Since four out of five of the nearest neighbors are short, C_{KNN} = *short*, so we have $\langle t_{16}, f, 1.6, short \rangle$.

Normalizing Values

Since different attributes can be measured on different scales, the Euclidean distance formula may exaggerate the effect of some attributes having larger scales. Normalization can be used to force all values to lie between 0 and 1.

$$a_i = \frac{v_i - \min(v_i)}{\max(v_i) - \min(v_i)}$$

where v_i is the actual value of attribute i and $\min(v_i)$ and $\max(v_i)$ are the minimum and maximum values, respectively, taken over all instances.

Example – Normalizing attribute values

> Given the values 5, 3, 8, 12, 9, 15, 11, 6, and 1, the normalized values are (5–1)/(15–1) = 4/14, 2/14, 7/14, 11/14, 8/14, 14/14, 10/14, 5/14, and 0/14.