Nearest Neighbor Classifiers

CSE 4308/5360: Artificial Intelligence I

University of Texas at Arlington

The Nearest Neighbor Classifier

- Let X be the space of all possible patterns for some classification problem.
- Let F be a **distance function** defined in X.
  - F assigns a distance to every pair v_1, v_2 of objects in X.
- Examples of such a distance function F?

The Nearest Neighbor Classifier

- Let X be the space of all possible patterns for some classification problem.
- Let F be a **distance function** defined in X.
  - F assigns a distance to every pair v_1, v_2 of objects in X.
- Examples of such a distance function F?

- The L_2 distance (also known as Euclidean distance, straight-line distance) for vector spaces.

$$L_2(v_1, v_2) = \sqrt{\sum_{i=1}^{D} \left(v_{1,i} - v_{2,i}\right)^2}$$

- The L_1 distance (Manhattan distance) for vector spaces.

$$L_1(v_1, v_2) = \sum_{i=1}^{D} \left|v_{1,i} - v_{2,i}\right|$$

The Nearest Neighbor Classifier

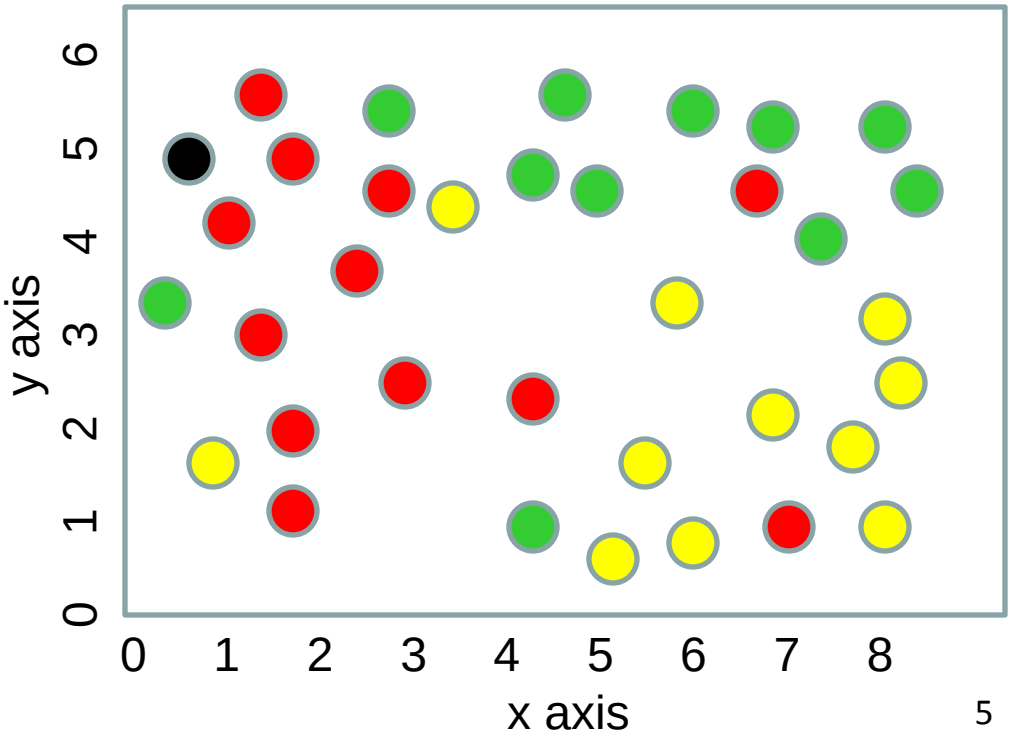
- Let F be a **distance function** defined in X.
  - F assigns a distance to every pair v_1, v_2 of objects in X.
- Let x_1, …, x_n be training examples.
- The nearest neighbor classifier classifies any pattern v as follows:
  - Find the training example NN(v) that is the nearest neighbor of v (has the shortest distance to v among all training data).

$$\mathrm{NN}(v) = \mathrm{argmax}_{x \in \{x_1, \ldots, x_n\}} \big(F(v, x)\big)$$

  - Return the class label of NN(v).
- In short, each test pattern v is assigned the class of its nearest neighbor in the training data.

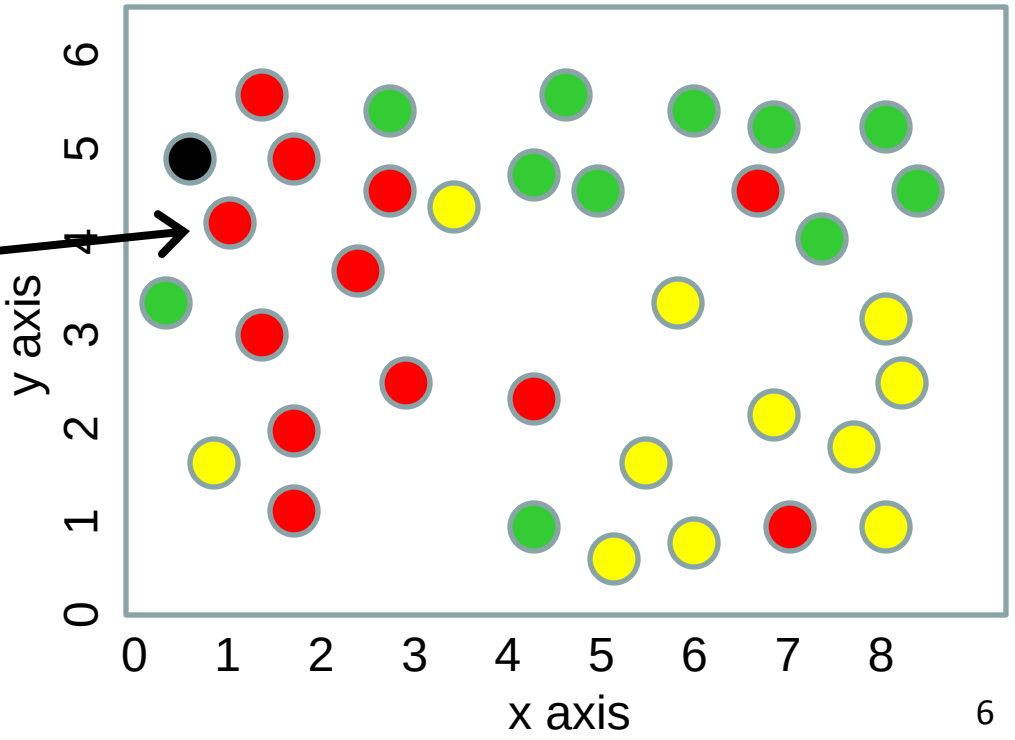
Example

- Suppose we have a problem where:
  - We have three classes (red, green, yellow).
  - Each pattern is a two-dimensional vector.
- Suppose that the training data is given below:
- Suppose we have a test pattern v, shown in black.
- How is v classified by the nearest neighbor classifier?



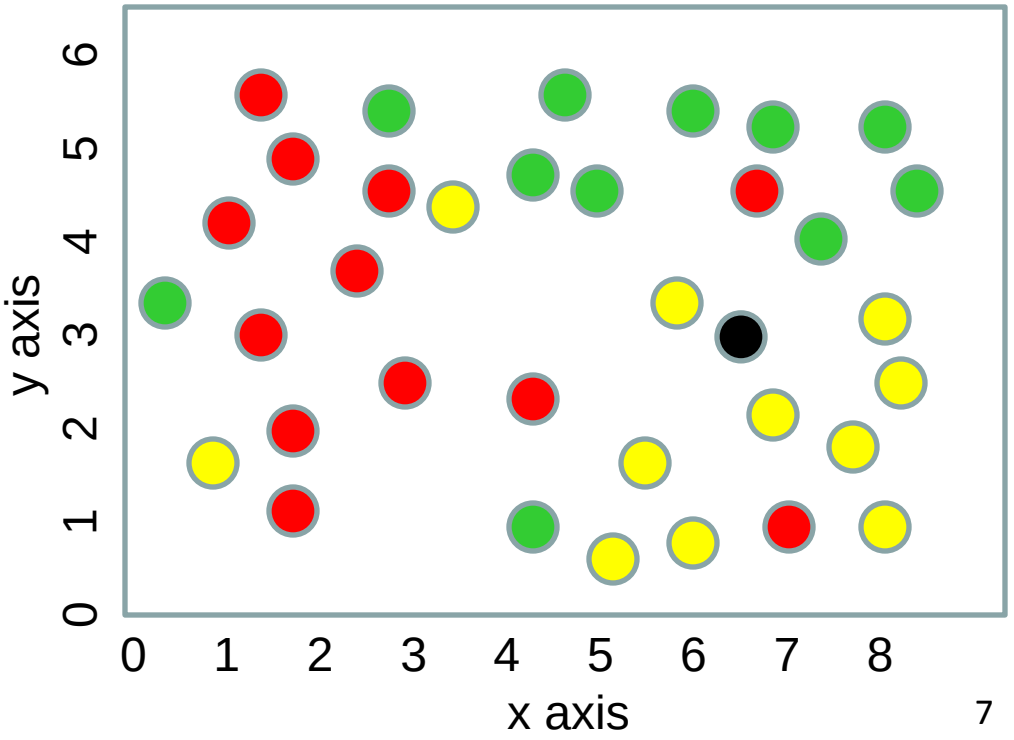
Example

- Suppose we have a problem where:
  - We have three classes (red, green, yellow).
  - Each pattern is a two-dimensional vector.
- Suppose that the training data is given below:
- Suppose we have a test pattern v, shown in black.
- How is v classified by the nearest neighbor classifier?
- NN(v):
- Class of NN(v): red.
- Therefore, v is classified as red.



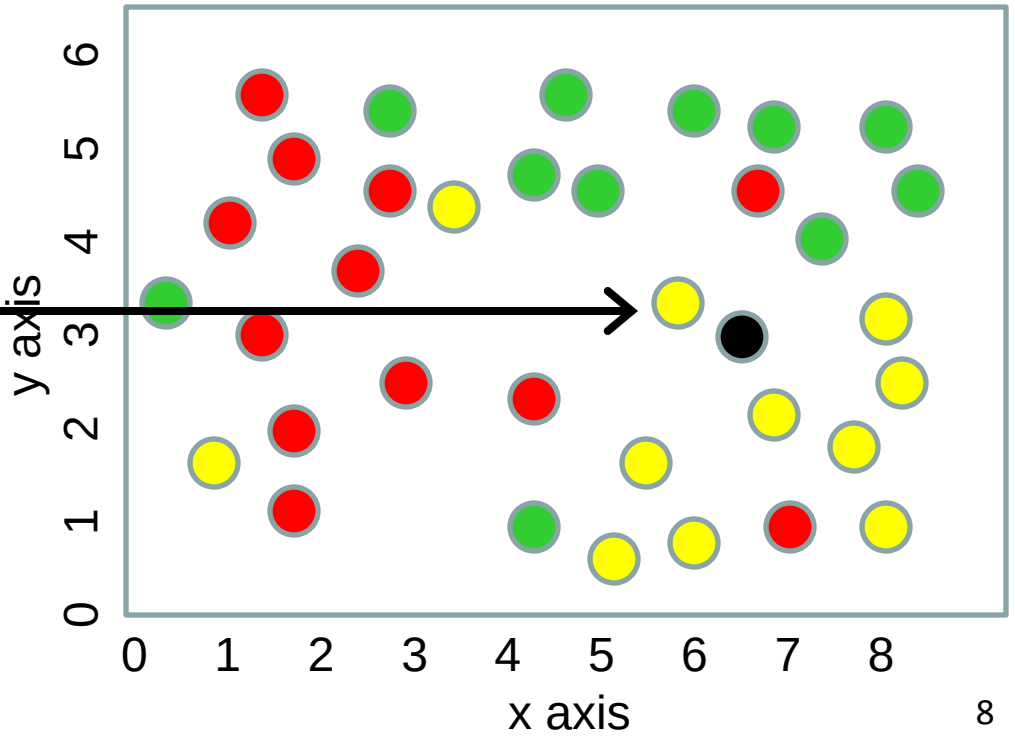
Example

- Suppose we have a problem where:
  - We have three classes (red, green, yellow).
  - Each pattern is a two-dimensional vector.
- Suppose that the training data is given below:
- Suppose we have another test pattern v, shown in black.
- How is v classified by the nearest neighbor classifier?



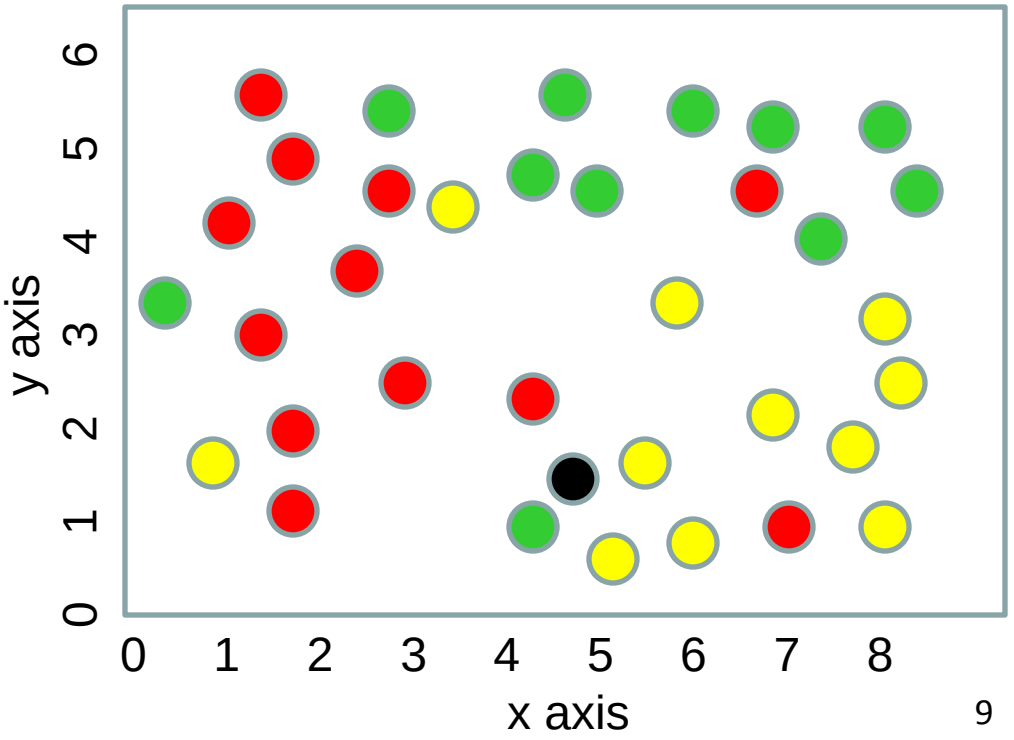
Example

- Suppose we have a problem where:
  - We have three classes (red, green, yellow).
  - Each pattern is a two-dimensional vector.
- Suppose that the training data is given below:
- Suppose we have another test pattern v, shown in black.
- How is v classified by the nearest neighbor classifier?
- NN(v):
- Class of NN(v): yellow.
- Therefore, v is classified as yellow.



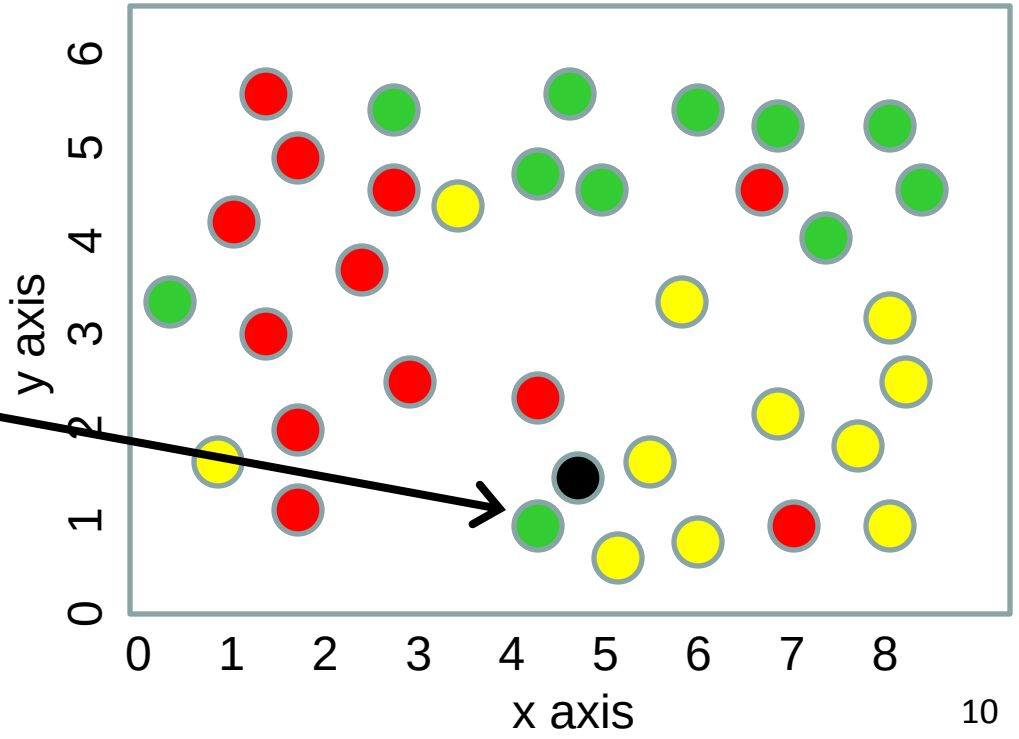
Example

- Suppose we have a problem where:
  - We have three classes (red, green, yellow).
  - Each pattern is a two-dimensional vector.
- Suppose that the training data is given below:
- Suppose we have another test pattern v, shown in black.
- How is v classified by the nearest neighbor classifier?



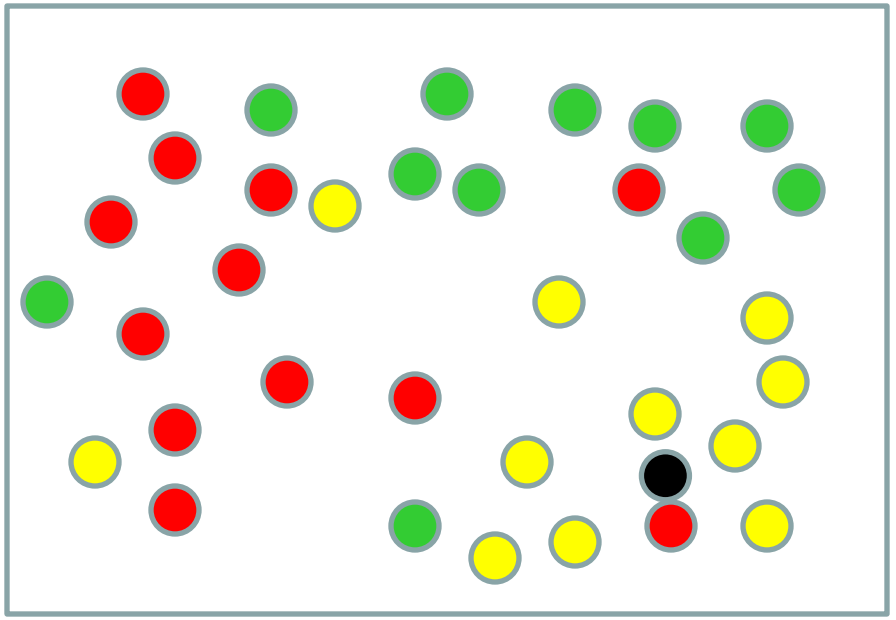
Example

- Suppose we have a problem where:
  - We have three classes (red, green, yellow).
  - Each pattern is a two-dimensional vector.
- Suppose that the training data is given below:
- Suppose we have another test pattern v, shown in black.
- How is v classified by the nearest neighbor classifier?
- NN(v):
- Class of NN(v): green.
- Therefore, v is classified as green.



The K-Nearest Neighbor Classifier

- Instead of classifying the test pattern based on its nearest neighbor, we can take more neighbors into account.
- This is called k-nearest neighbor classification.
- Using more neighbors can help avoid mistakes due to noisy data.
- In the example shown on the figure, the test pattern is in a mostly “yellow” area, but its nearest neighbor is red.
- If we use the 3 nearest neighbors, 2 of them are yellow.



Normalizing Dimensions

- Suppose that your test patterns are 2-dimensional vectors, representing stars.
  - The first dimension is surface temperature, measured in Fahrenheit.
  - Your second dimension is weight, measured in pounds.
- The surface temperature can vary from 6,000 degrees to 100,000 degrees.
- The weight can vary from 10^{29} to 10^{32}.
- Does it make sense to use the Euclidean distance or the Manhattan distance here?

Normalizing Dimensions

- Suppose that your test patterns are 2-dimensional vectors, representing stars.
  - The first dimension is surface temperature, measured in Fahrenheit.
  - Your second dimension is weight, measured in pounds.
- The surface temperature can vary from 6,000 degrees to 100,000 degrees.
- The weight can vary from 10^{29} to 10^{32}.
- Does it make sense to use the Euclidean distance or the Manhattan distance here?
- No. These distances treat both dimensions equally, and assume that they are both measured in the same units.
- Applied to these data, the distances would be dominated by differences in mass, and would mostly ignore information from surface temperatures.

Normalizing Dimensions

- It would make sense to use the Euclidean or Manhattan distance, if we first normalized dimensions, so that they contribute equally to the distance.
- How can we do such normalizations?
- There are various approaches. Two common approaches are:
  - Translate and scale each dimension so that its minimum value is 0 and its maximum value is 1.
  - Translate and scale each dimension so that its mean value is 0 and its standard deviation is 1.

Normalizing Dimensions – A Toy Example

Original Data			Min = 0, Max = 1		Mean = 0, std = 1	
Object ID	Temp. (F)	Weight (lb.)	Temp.	Weight	Temp.	Weight
1	4700	$1.5*10^{30}$	0.0000	0.0108	-0.9802	-0.6029
2	11000	$3.5*10^{30}$	0.1525	0.0377	-0.5375	-0.5322
3	46000	$7.5*10^{31}$	1.0000	1.0000	1.9218	1.9931
4	12000	$5.0*10^{31}$	0.1768	0.6635	-0.4673	1.1101
5	20000	$7.0*10^{29}$	0.3705	0.0000	0.0949	-0.6311
6	13000	$2.0*10^{30}$	0.2010	0.0175	-0.3970	-0.5852
7	8500	$8.5*10^{29}$	0.0920	0.0020	-0.7132	-0.6258
8	34000	$1.5*10^{31}$	0.7094	0.1925	1.0786	-0.1260

Scaling to Lots of Data

- Nearest neighbor classifiers become more accurate as we get more and more training data.
- Theoretically, one can prove that k-nearest neighbor classifiers become Bayes classifiers (and thus optimal) as training data approaches infinity.
- One big problem, as training data becomes very large, is time complexity.
  - What takes time?
  - Computing the distances between the test pattern and all training examples.

Nearest Neighbor Search

- The problem of finding the nearest neighbors of a pattern is called “nearest neighbor search”.
- Suppose that we have N training examples.
- Suppose that each example is a D-dimensional vector.
- What is the time complexity of finding the nearest neighbors of a test pattern?

Nearest Neighbor Search

- The problem of finding the nearest neighbors of a pattern is called "nearest neighbor search".
- Suppose that we have N training examples.
- Suppose that each example is a D-dimensional vector.
- What is the time complexity of finding the nearest neighbors of a test pattern?
- O(ND).
  - We need to consider each dimension of each training example.
- This complexity is linear, and we are used to thinking that linear complexity is not that bad.
- If we have millions, or billions, or trillions of data, the actual time it takes to find nearest neighbors can be a big problem for real-world applications.

Indexing Methods

- As we just mentioned, measuring the distance between the test pattern and each training example takes O(ND) time.
- This method of finding nearest neighbors is called “brute-force search”, because we go through all the training data.
- There are methods for finding nearest neighbors that are sublinear to N (even logarithmic, at times), but exponential to D.
- Can you think of an example?

Indexing Methods

- As we just mentioned, measuring the distance between the test pattern and each training example takes O(ND) time.
- This method of finding nearest neighbors is called “brute-force search”, because we go through all the training data.
- There are methods for finding nearest neighbors that are sublinear to N (even logarithmic, at times), but exponential to D.
- Can you think of an example?
- Binary search (applicable when D=1) takes O(log N) time.

Indexing Methods

- In some cases, faster algorithms exist that, however, are approximate.
  - They do not guarantee finding the true nearest neighbor all the time.
  - They guarantee that they find the true nearest neighbor with a certain probability.
- This was the topic of my Ph.D. thesis, so I can talk for days about it (but I won't).

The Curse of Dimensionality

- The “curse of dimensionality” is a commonly used term in artificial intelligence.
  - Common enough that it has a dedicated Wikipedia article.
- It is actually not a single curse, it shows up in many different ways.
- The curse consists of the fact that lots of AI methods are very good at handling low-dimensional data, but horrible at handling high-dimensional data.
- The nearest neighbor problem is an example of this curse.
  - Finding nearest neighbors in low dimensions (like 1, 2, 3 dimensions) can be done in close to logarithmic time.
  - However, these approaches take time exponential to D.
  - By the time you get to 50, 100, 1000 dimensions, they get hopeless.
  - Data in AI oftentimes has thousands or millions of dimensions.

More Exotic Distance Measures

- The Euclidean and Manhattan distance are the most commonly used distance measures.
- However, in some cases they do not make much sense, or they cannot even be applied.
- In such cases, more exotic distance measures can be used.
- A few examples (this is not required material, it is just for your reference):
  - The edit distance for strings (hopefully you've seen it in your algorithms class).
  - Dynamic time warping for time series.
  - Bipartite matching for sets.

Nearest Neighbor Classification: Recap

- Nearest neighbor classifiers can be pretty simple to implement.
- They become increasingly accurate as we get more training examples.
- Normalizing the values in each dimension may be necessary, before measuring distances.
- Finding nearest neighbors in high-dimensional spaces can be very, very slow, when we have lots of training data.