Nearest Neighbor Classifiers

CSE 4309 – Machine Learning

Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

The Nearest Neighbor Classifier

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

The Nearest Neighbor Classifier

- Let $\mathbb{X}$ be the space of all possible patterns for some classification problem.
- Let F be a **distance function** defined in $\mathbb{X}$.
  - F assigns a distance to every pair v_1, v_2 of objects in $\mathbb{X}$.
- 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

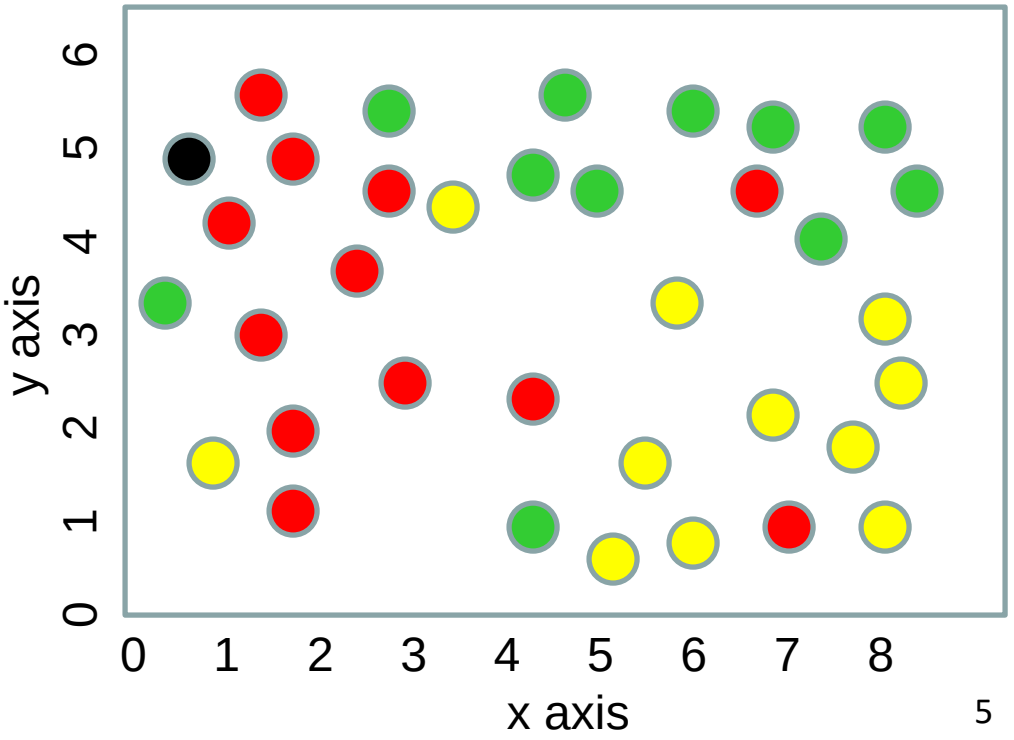
- Let F be a **distance function** defined in $\mathbb{X}$.
  - F assigns a distance to every pair v_1, v_2 of objects in $\mathbb{X}$.
- Let $x_1, x_2, \ldots, x_N$ be training examples.
- The nearest neighbor classifier classifies any pattern v as follows:
  - Find the training example $\mathrm{C}(v)$ that is the nearest neighbor of x (has the shortest distance to v among all training data).

$$\mathrm{C}(v) = \mathrm{argmin}_{x \in \{x_1, \ldots, x_N\}} \big(F(v, x)\big)$$

  - Return the class label of $\mathrm{C}(v)$.
- In short, each test pattern v is assigned the class of its nearest neighbor in the training data.

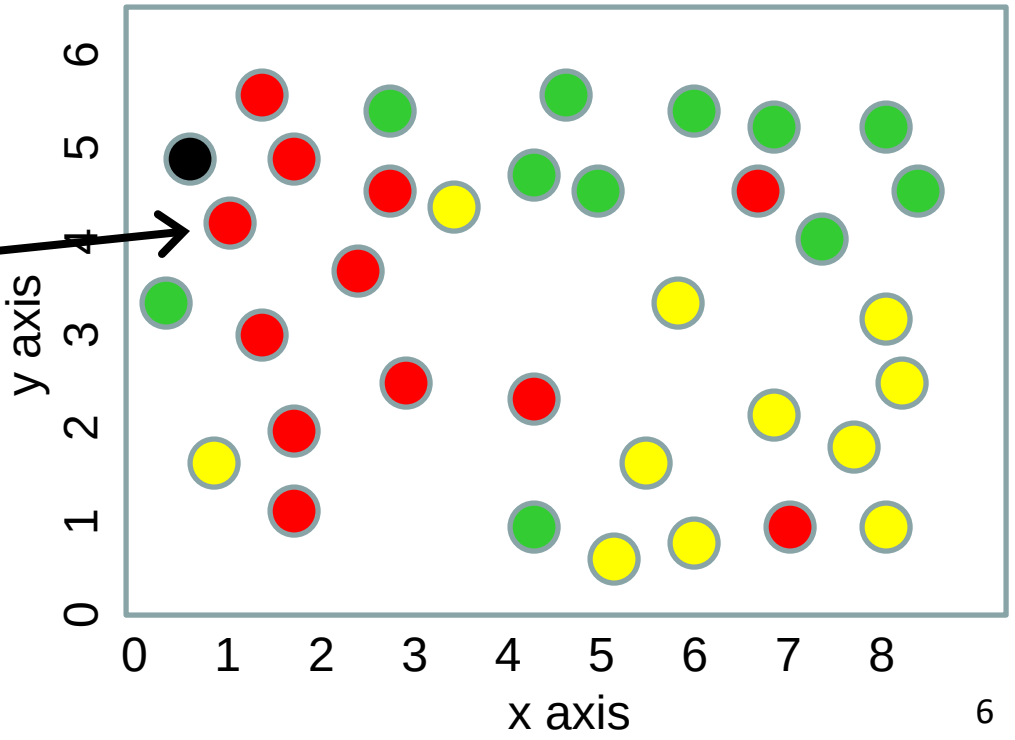
Example

- 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?



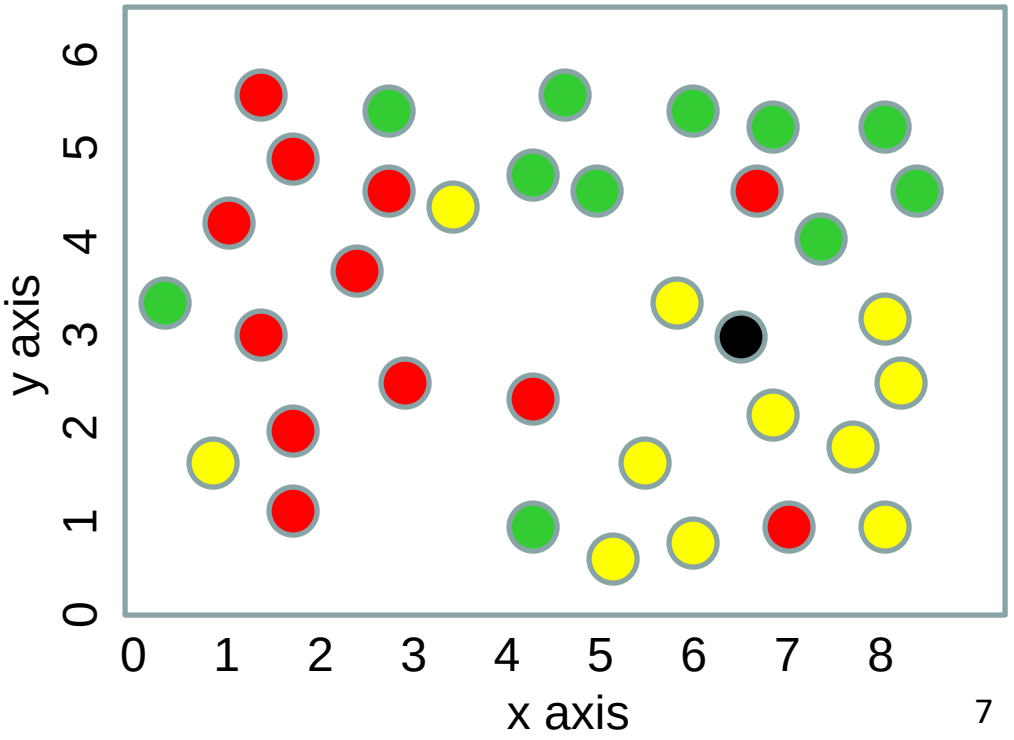
Example

- 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?
- $C(v)$:
- Class of $C(v)$: red.
- Therefore, v is classified as red.



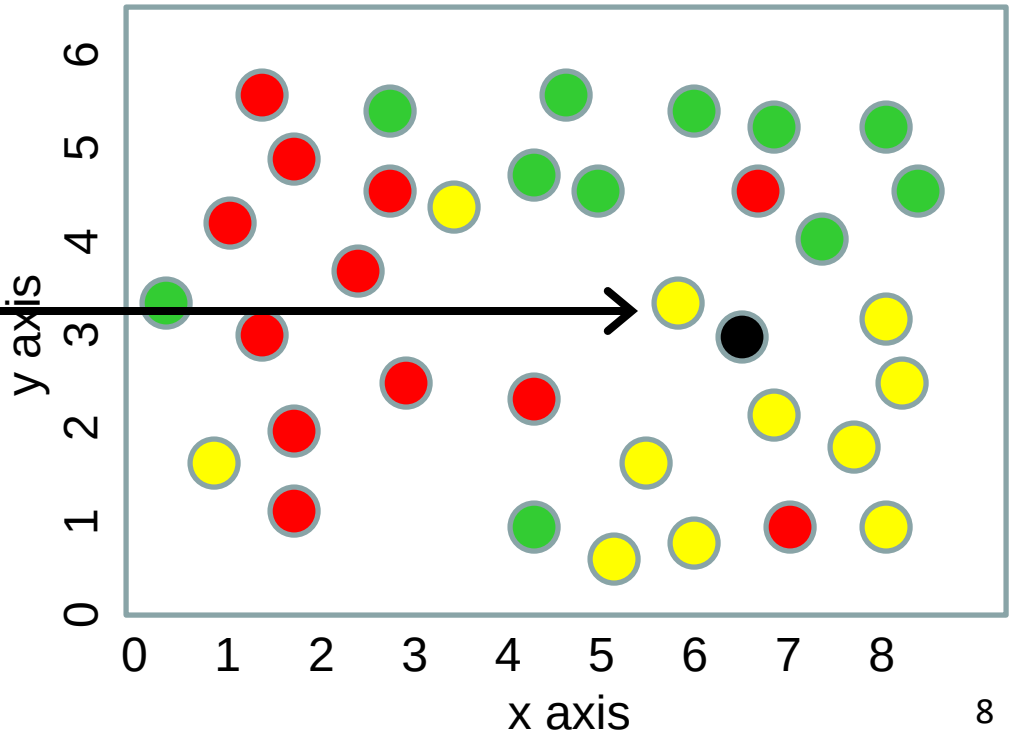
Example

- 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?



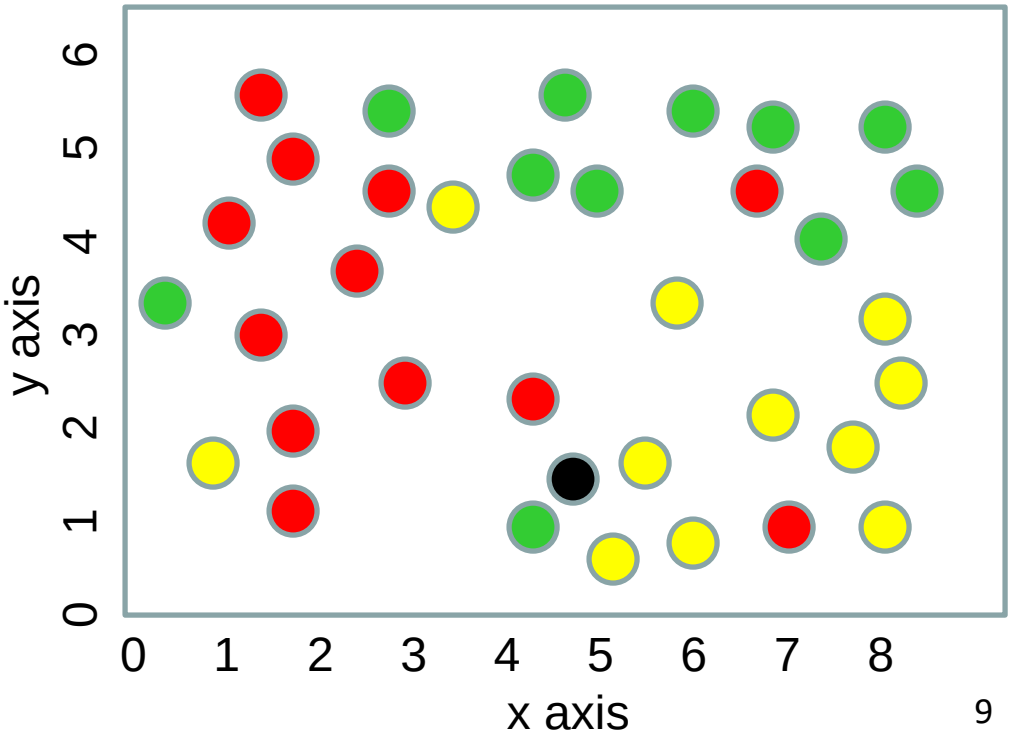
Example

- 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?
- $C(v)$:
- Class of $C(v)$: yellow.
- Therefore, v is classified as yellow.



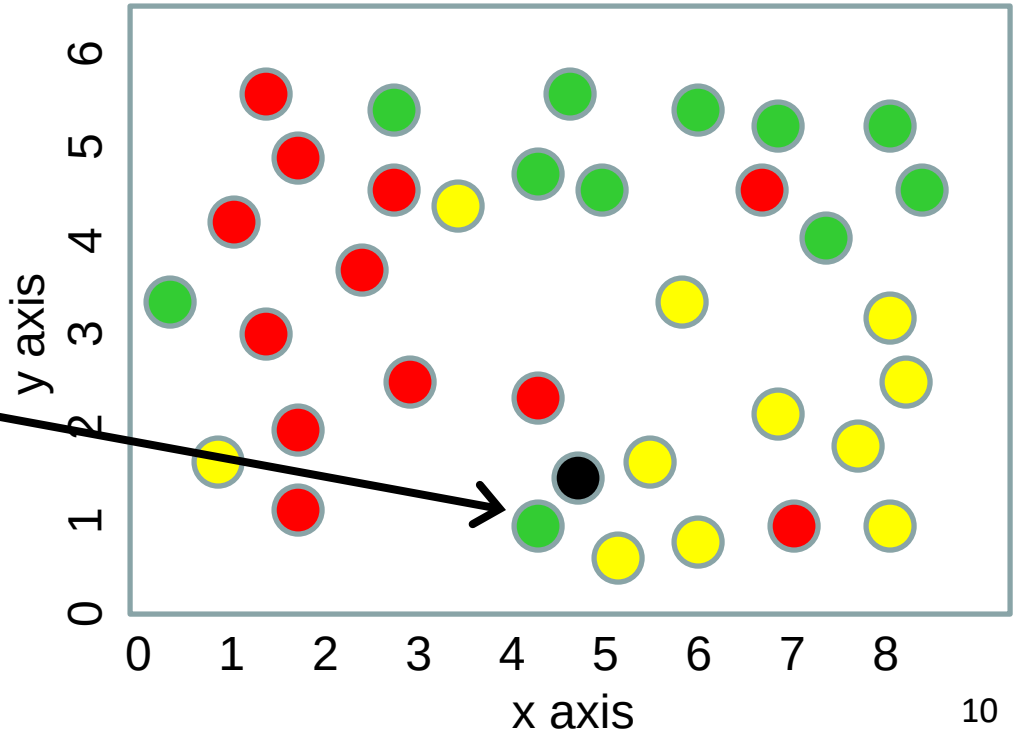
Example

- 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?



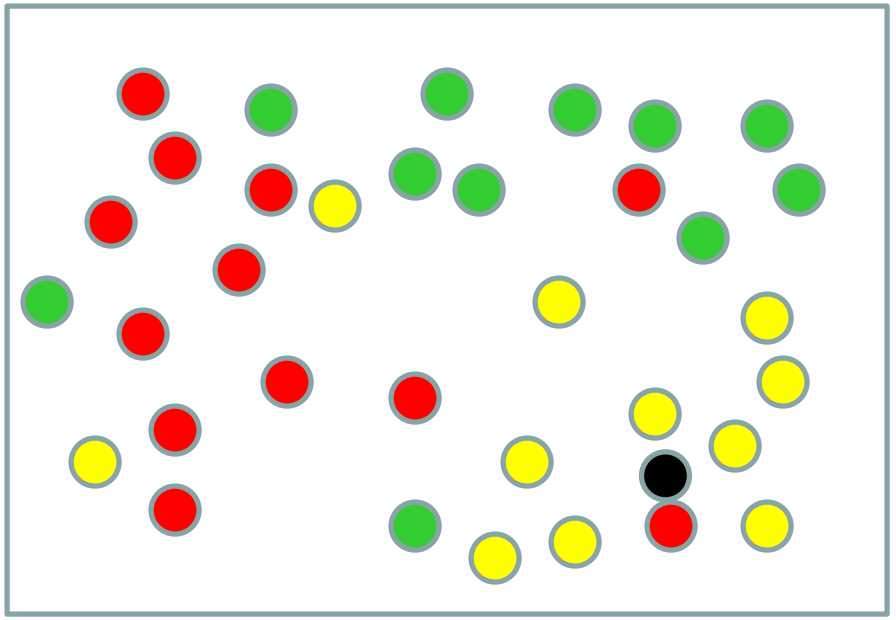
Example

- 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?
- $C(v)$:
- Class of $C(v)$: yellow.
- Therefore, v is classified as yellow.



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 mass, measured in pounds.
- The surface temperature can vary from 6,000 degrees to 100,000 degrees.
- The mass 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 mass, measured in pounds.
- The surface temperature can vary from 6,000 degrees to 100,000 degrees.
- The mass 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			Normalized Data: Min = 0, Max = 1		Normalized Data: Mean = 0, std = 1	
Object ID	Temp. (F)	Mass (lb.)	Temp.	Mass	Temp.	Mass
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.

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 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.

Case Study: Hand Pose Estimation

- **Hand pose** is defined by:
  - **Handshape**: the position, orientation, and shape of each finger.
  - **3D orientation**: the direction that the hand points to.

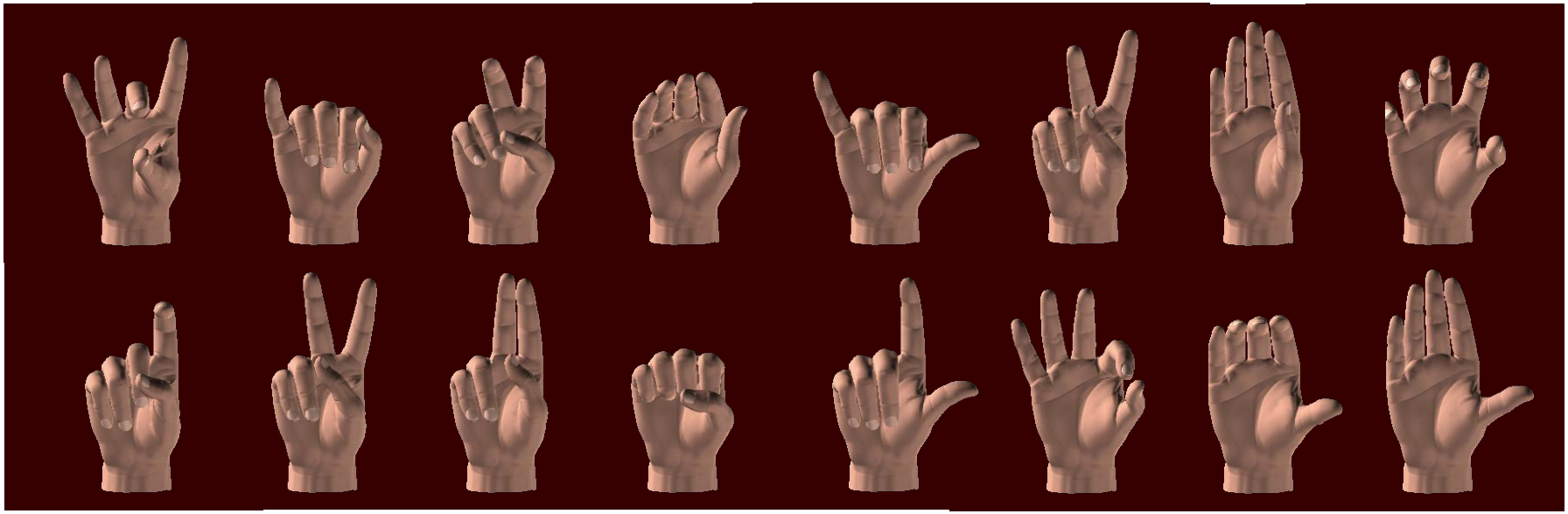
input image

Hand pose:

- Hand shape
- 3D hand orientation.

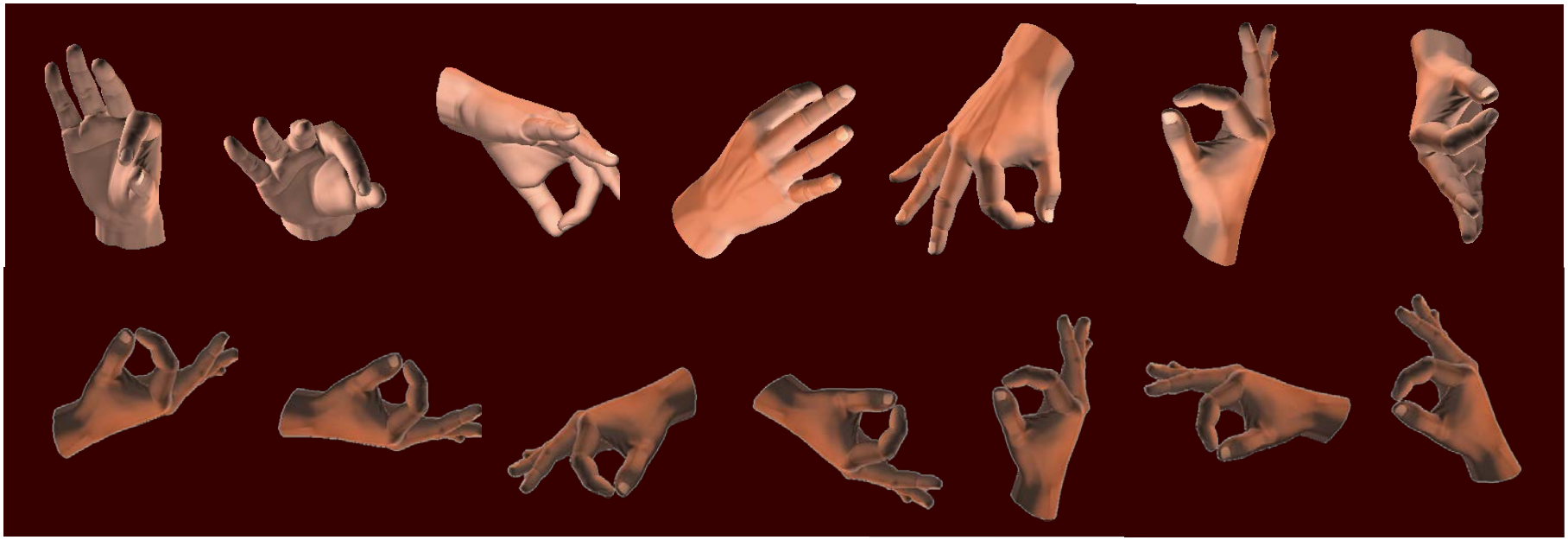
Hand Shapes

- Handshapes are specified by the joint angles of the fingers.
- Hands are very flexible.
  - Hundreds or thousands of distinct hand shapes.



3D Hand Orientation

- Images of the same handshape can look VERY different.
- Appearance depends on the 3D orientation of the hand with respect to the camera.
- Here are some examples of the same shape seen under different orientations.



Hand Pose Estimation: Applications

- There are several applications of hand pose estimation (if the estimates are sufficiently accurate, which is a big if):
  - Sign language recognition.
  - Human-computer interfaces (controlling applications and games via gestures).
  - Clinical applications (studying the motion skills of children, patients…).

input image

Hand pose:

- Hand shape
- 3D hand orientation.

Hand Pose Estimation

- There are many different computer vision methods for estimating hand pose.
  - The performance of these methods (so far) is much lower to that of the human visual system.
  - However, performance can be good enough for specific applications.

Hand pose:

- Hand shape
- 3D hand orientation.

Hand Pose Estimation

- We will take a look at a simple method, based on nearest neighbor classification.
- This method is described in more detail in this paper:

Vassilis Athitsos and Stan Sclaroff, "An Appearance-Based Framework for 3D Hand Shape Classification and Camera Viewpoint Estimation". *IEEE Conference on Automatic Face and Gesture Recognition*, 2002. http://vlm1.uta.edu/~athitsos/publications/bucs-2001-022.pdf

Hand pose:

- Hand shape
- 3D hand orientation.

Preprocessing: Hand Detection

- This specific nearest-neighbor method assumes that, somehow, the input image can be preprocessed, so as to:
  - Detect the location of the hand.
  - Segment the hand (i.e., separate the hand pixels from the background pixels).

segmented hand

Preprocessing: Hand Detection

- In the general case, hand detection and segmentation are in themselves very challenging tasks.
- However, in images like the example input image shown here, standard computer vision methods work pretty well.
  - You will learn to implement such methods if you take the computer vision class.

input image

segmented hand

Problem Definition

- Assuming that the hand has been segmented, our task is to learn a function, that maps the segmented hand image to the correct hand pose.

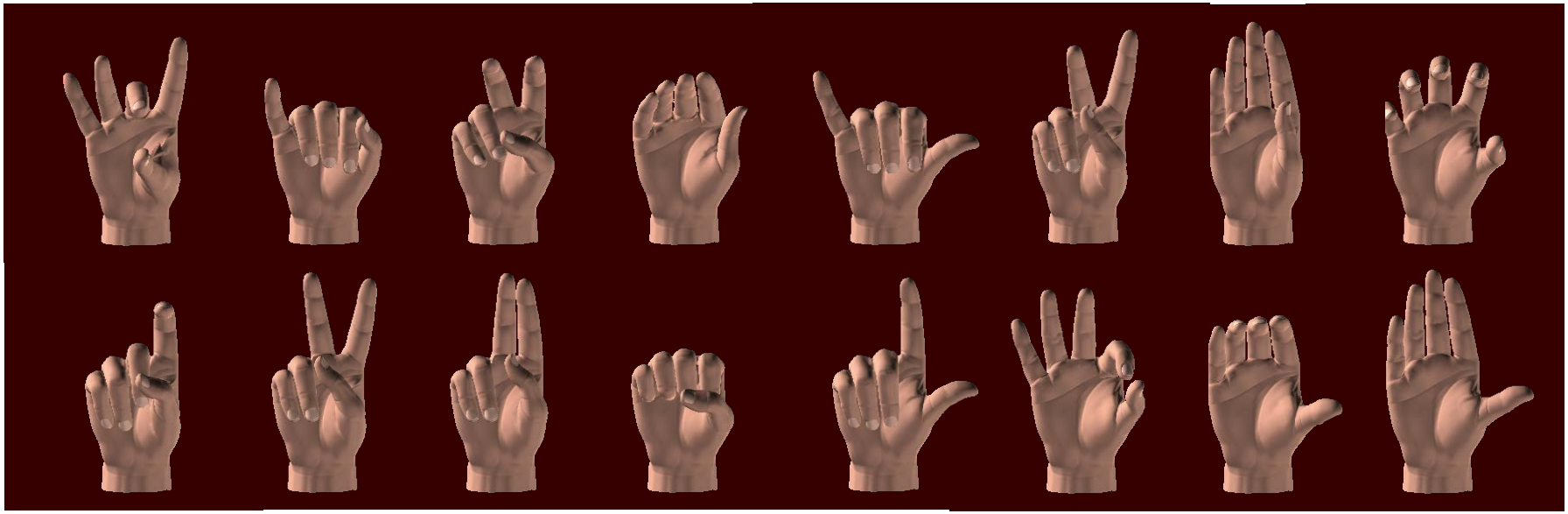
segmented hand

Hand pose:

- Hand shape
- 3D hand orientation.

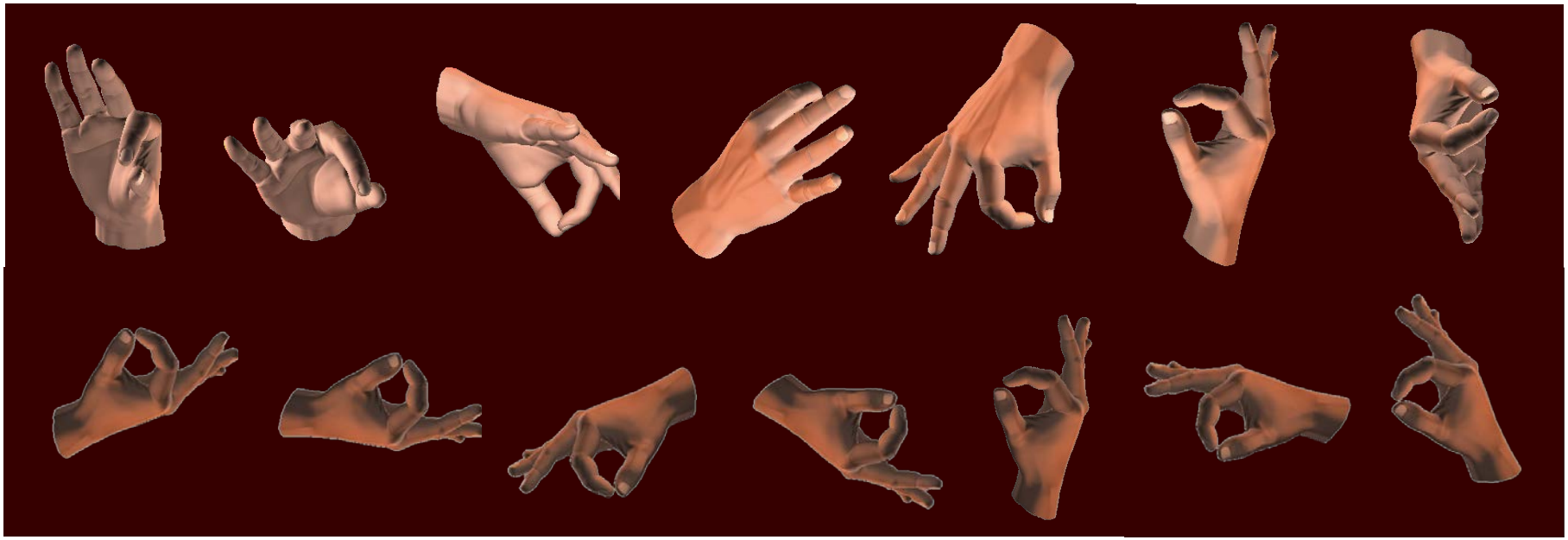
Problem Definition: Hand Shapes

- We will train a system that only recognizes some selected handshapes (for example, 20-30 specific handshapes).
  - This means that the system will not recognize arbitrary handshapes.
  - A few tens of shapes, however, is sufficient for some applications, like sign language recognition or using hands to control games and applications.



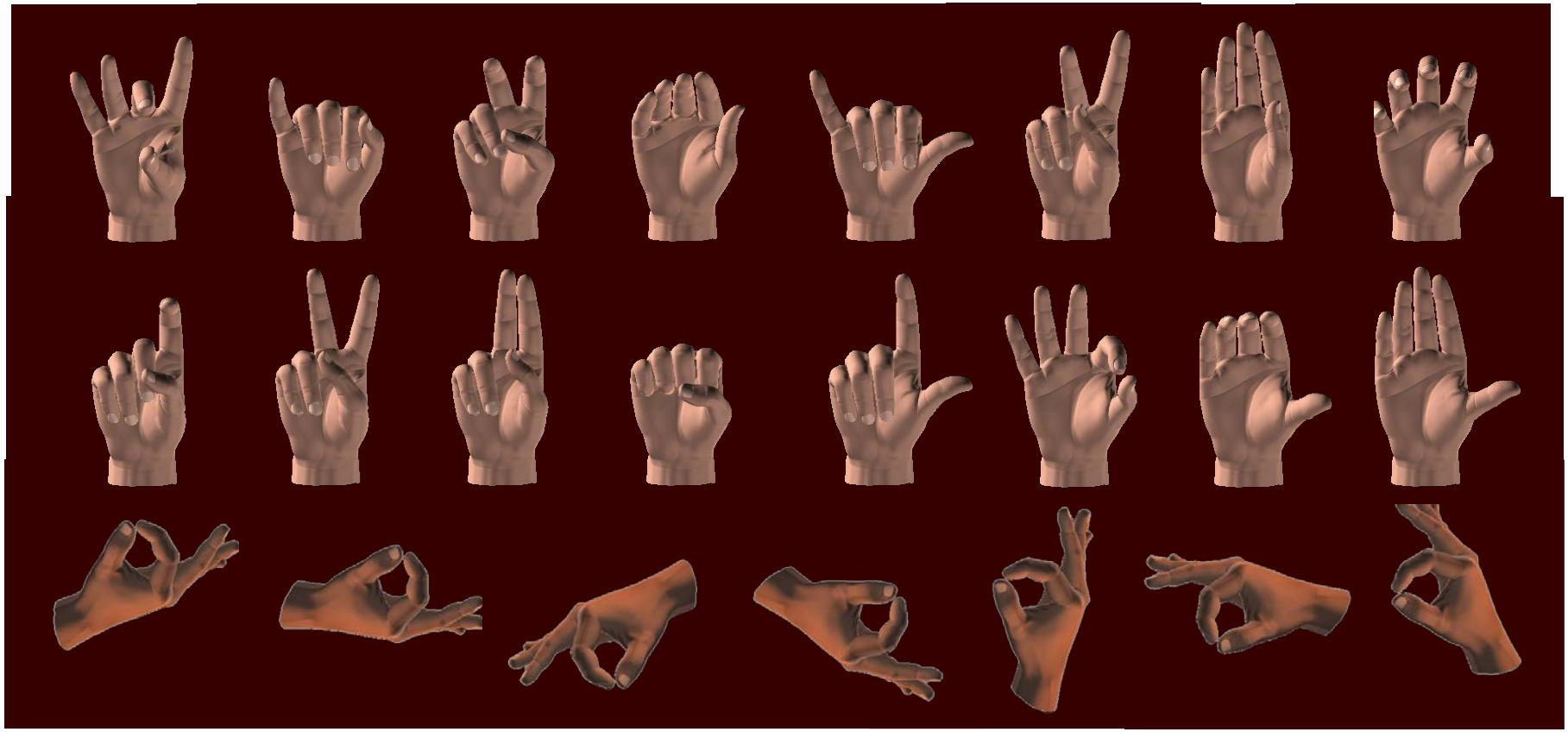
Problem Definition: Orientations

- We will allow the hand to have any 3D orientation whatsoever.
  - Our goal is to be able to recognize the handshape under any orientation.
  - The system will estimate both the handshape and the 3D orientation.



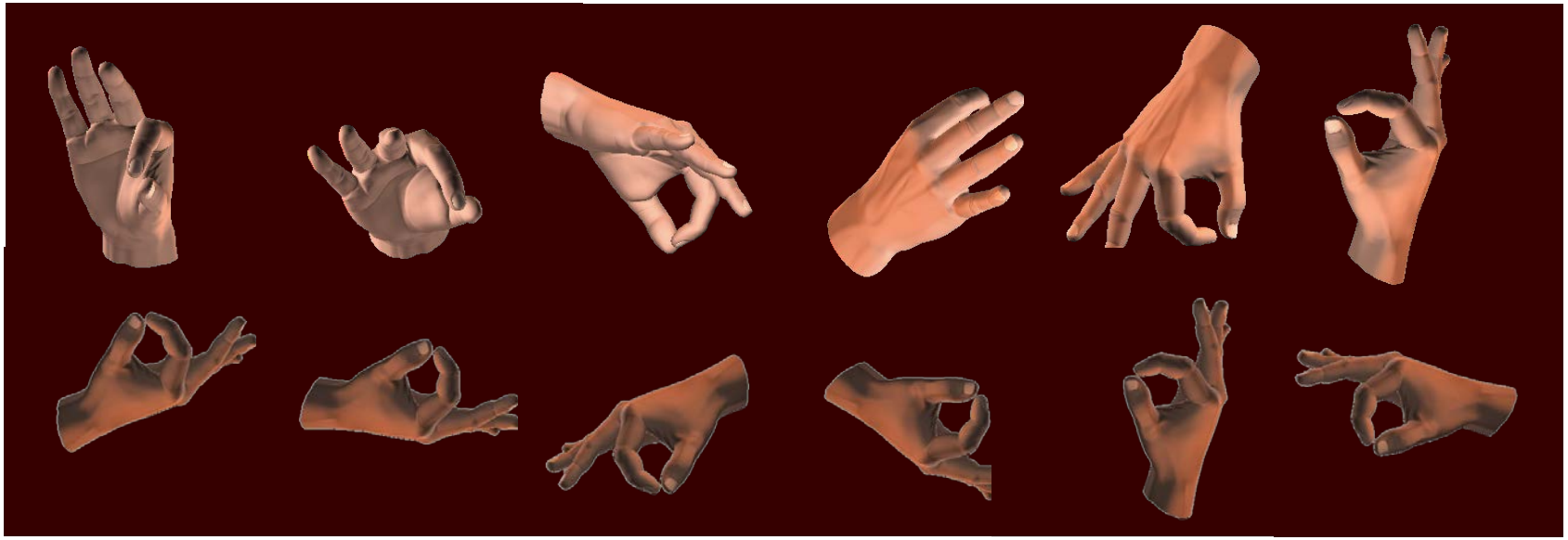
Training Set

- The training set is generated using computer graphics software.
  - 26 hand shapes.
  - 4000 orientations for each shape.
  - Over 100,000 images in total.



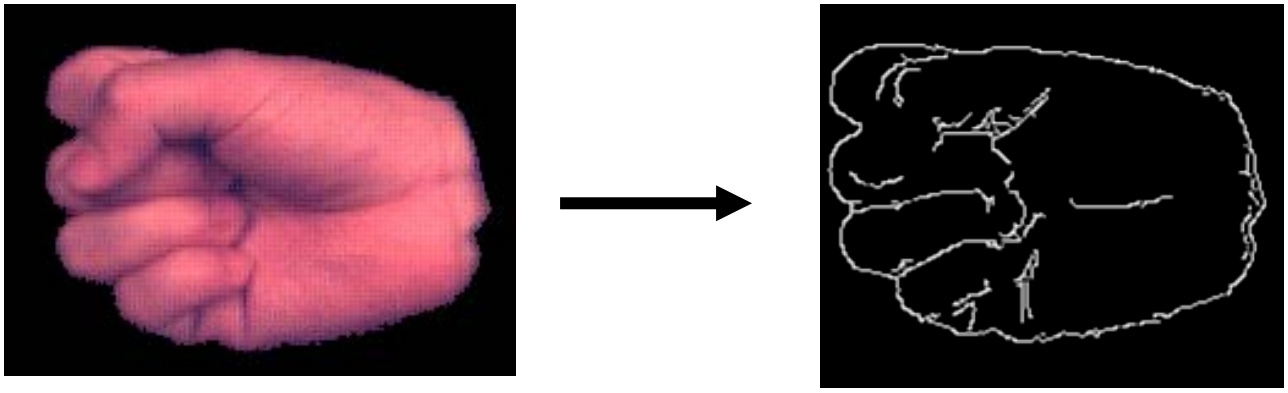
Euclidean Distance Works Poorly

- Suppose we want to use nearest neighbors classification, and we choose the Euclidean Distance as the distance measure.
- The Euclidean distance between two images compares the color values of the two images in each pixel location.
- Color is not a reliable feature. It depends on many things, like skin color, lighting conditions, shadows…



Converting to Edge Images

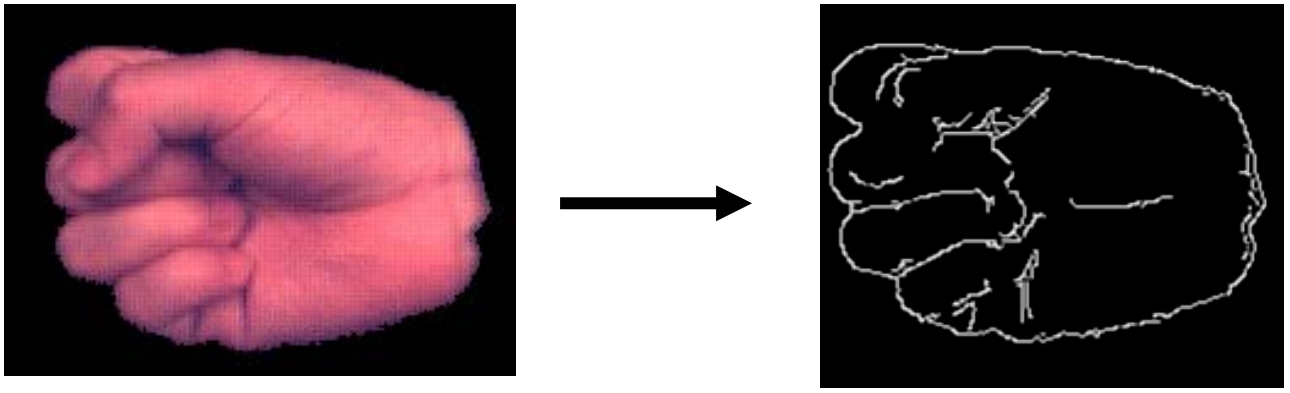
- Edge images: images where edge pixels are white, and everything else is black.
  - Roughly speaking, edge pixels are boundary pixels, having significantly different color than some of their neighbors.
- One of the first topics in any computer vision course is how to compute edge images.



original image edge image

Converting to Edge Images

- Edge images are more stable than color images.
  - They still depend somewhat on things like skin color, illumination, shadows, but not as much as color images.

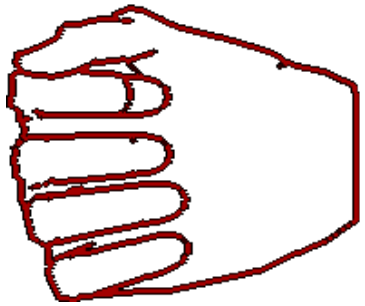


original image edge image

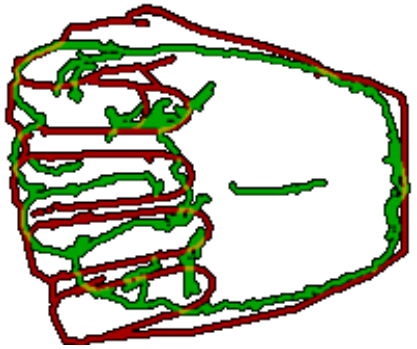
Euclidean Distance on Edge Images

test image

training image



test image superimposed on training image

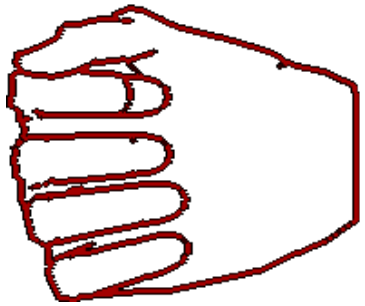


- The Euclidean distance also works poorly on edge images.
  - The distance between two edge images depends entirely on the number of pixel locations where one image has an edge pixel and the other image does not have an edge pixel.
  - Even two nearly identical edge images can have a large distance, if they are slightly misaligned.

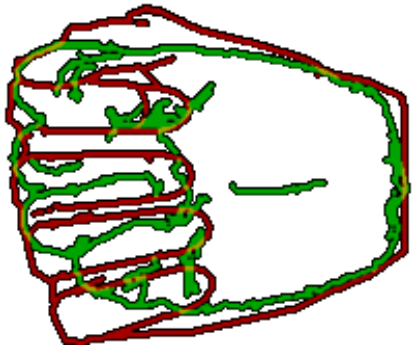
Euclidean Distance on Edge Images

test image

training image



test image superimposed on training image

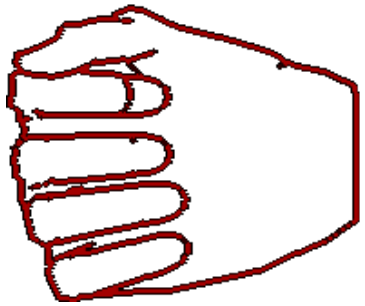


- For example, consider the test image superimposed on the training image:
  - Even though the two images look similar, the edge pixel locations do not match.
  - The Euclidean distance is high, and does not capture the similarity between those two images.

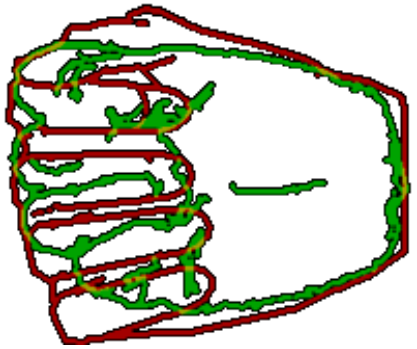
Chamfer Distance

test image

training image



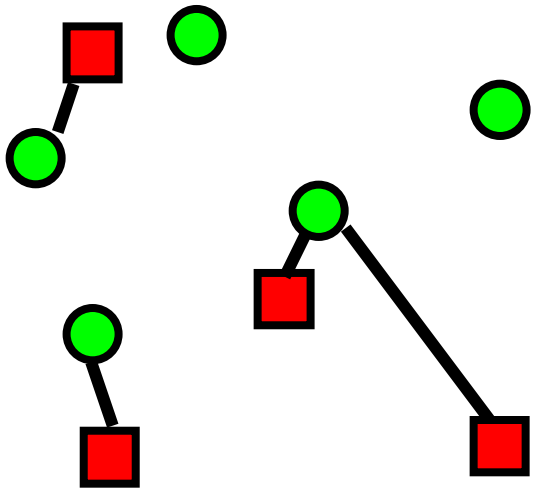
test image superimposed on training image



- In the chamfer distance, each edge image is simply represented as a set of points (pixel locations).
  - This set contains the pixel location of every edge pixel in that image.
  - The pixel location is just two numbers: row and column.
- How can we define a distance between two sets of points?

Directed Chamfer Distance

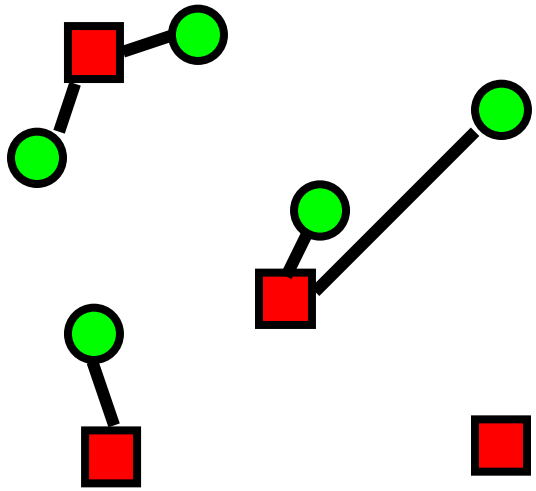
- Input: two sets of points.
  - R is the set of red points.
  - G is the set of green points.
- For each red point P_r:
  - Identify the green point P_g that is the closest to P_r.



- The directed chamfer distance $c(R, G)$ is the average distance from each red point to its nearest green point.
- For the two sets shown on the figure:
  - Each red point is connected by a link to its nearest green point.
  - $c(R, G)$ is the average length of those links.

Directed Chamfer Distance

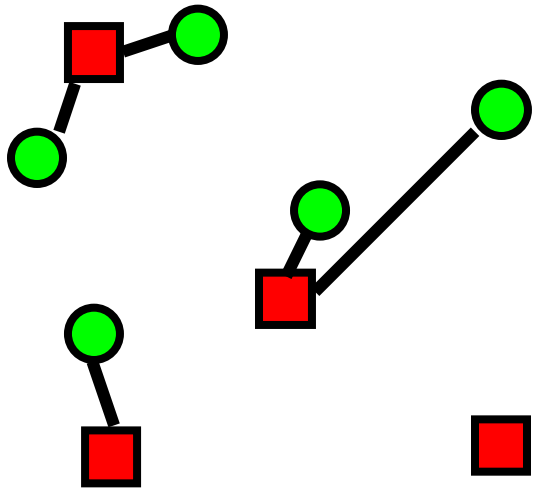
- Input: two sets of points.
  - R is the set of red points.
  - G is the set of green points.
- For each green point P_g:
  - Identify the red point P_r that is the closest to P_g.



- The directed chamfer distance $c(G, R)$ is the average distance from each green point to its nearest red point.
- For the two sets shown on the figure:
  - Each green point is connected by a link to its nearest red point.
  - $c(G, R)$ is the average length of those links.

Chamfer Distance

- Input: two sets of points.
  - R is the set of red points.
  - G is the set of green points.
- $c(R, G)$:
  - Average distance from each red point to nearest green point.



- $c(G, R)$:
  - Average distance from each red point to nearest green point.
- The **chamfer distance** $C(R, G)$ is simply the sum of the two directed chamfer distances $c(R, G)$ and $c(G, R)$:

$$\boldsymbol{C(R, G) = c(R, G) + c(G, R)}$$

Chamfer Distance Example

- $R = \{(12, 5), (8, 6), (5,13)\}$
- $G = \{(5,7), (6,11)\}$
- First, we compute the directed distance $c(R, G)$:
- For $(12, 5)$, the closest point in G is $(5, 7)$.
  - Distance: $\sqrt{(12-5)^2+(5-7)^2} = 7.28$.
- For $(8, 6)$, the closest point in G is $(5, 7)$.
  - Distance: $\sqrt{(8-5)^2+(6-7)^2} = 3.16$.
- For $(5, 13)$, the closest point in G is $(6, 11)$.
  - Distance: $\sqrt{(5-6)^2+(13-11)^2} = 2.24$.
- $c(R, G) = \frac{7.28+3.16+2.24}{3} = 4.23$.

Chamfer Distance Example

- $R = \{(12, 5), (8, 6), (5,13)\}$
- $G = \{(5,7), (6,11)\}$
- Second, we compute the directed distance $c(G, R)$:
- For $(5, 7)$, the closest point in R is $(8, 6)$.
  - Distance: $\sqrt{(5-8)^2+(7-6)^2} = 3.16$.
- For $(6, 11)$, the closest point in R is $(5, 13)$.
  - Distance: $\sqrt{(6-5)^2+(11-13)^2} = 2.24$.
- $c(R, G) = \frac{3.16+2.24}{2} = 2.70$.

Chamfer Distance Example

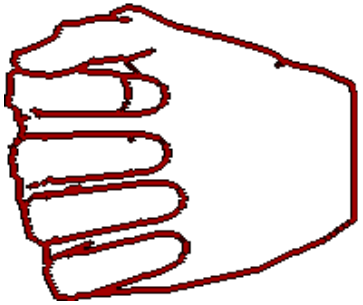
- $R = \{(12, 5), (8, 6), (5,13)\}$
- $G = \{(5,7), (6,11)\}$
- Third, we compute the undirected chamfer distance $C(R, G)$:
  - The undirected chamfer distance is the sum of the two directed distances.

$$C(R, G) = c(R, G) + c(G, R) = 4.23 + 2.70 = 6.93$$

Efficiency of the Chamfer Distance

test image

training image



- For every edge pixel in one image we must find the distance to the closest edge pixel in the other image.
- Suppose that our two images have approximately D edge pixels.
  - It takes $O(d^2)$ to measure the distance between every edge pixel in one image and every edge pixel in the other image.
  - There are more efficient algorithms, taking $O(d \log d)$ time.

Other Non-Euclidean Distance Measures

- Bipartite matching.
  - Like the chamfer distance, bipartite matching can be used as a distance measure for sets of points.
  - It finds the optimal 1-1 correspondence between points of one set and points of the other set.
  - Time complexity: cubic to the number of points.
- Kullback-Leibler distance.
  - It is a distance measure for probability distributions.

Distance/Similarity Measures for Strings

- Edit distance.
  - The number of the fewest insert/delete/substitute operations that convert one string to another string.
- Smith-Waterman.
  - A similarity measure for strings, high values indicate higher similarity.
  - Here, the “nearest neighbor” would be the training string with the highest Smith-Waterman score with respect to the test string.
- Longest Common Subsequence (LCSS).

Distance Measures for Time Series

- Dynamic Time Warping (DTW).
  - We will talk about that in more detail.
- Edit Distance with Real Penalty (ERP).
- Time Warping Edit Distance (TWED).
- Move-Split-Merge (MSM).

Nearest Neighbors: Learning

- What is the "training stage" for a nearest neighbor classifier?

Nearest Neighbors: Learning

- What is the "training stage" for a nearest neighbor classifier?
- In the simplest case, none.
  - Once we have a training set, and we have chosen a distance measure, no other training is needed.

Nearest Neighbors: Learning

- What is the "training stage" for a nearest neighbor classifier?
- In the simplest case, none.
  - Once we have a training set, and we have chosen a distance measure, no other training is needed.
- There are things that can be learned, if desired:
  - Choosing elements for the training set. **Condensing** is a method that removes training objects that actually hurt classification accuracy.
  - Parameters of the distance measure. For example, if we use the Euclidean distance, the weight of each dimension (or a projection function, mapping data to a new space) can be learned.

Nearest Neighbor Classification: Recap

- Nearest neighbor classifiers can be pretty simple to implement.
  - Just pick a training set and a distance measure.
- They become increasingly accurate as we get more training examples.
- Nearest neighbor classifiers can be defined even if we have only one training example per class.
  - Most methods require significantly more data.

Nearest Neighbor Classification: Recap

- In Euclidean spaces, normalizing the values in each dimension may be necessary, before measuring distances.
  - Or, alternatively, learning optimal weights for each dimension.
- Non-Euclidean distance measures are defined for non-Euclidean spaces (edge images, strings, time series, probability distributions),
- Finding nearest neighbors in high-dimensional spaces, or non-Euclidean spaces, can be very, very slow, when we have lots of training data.