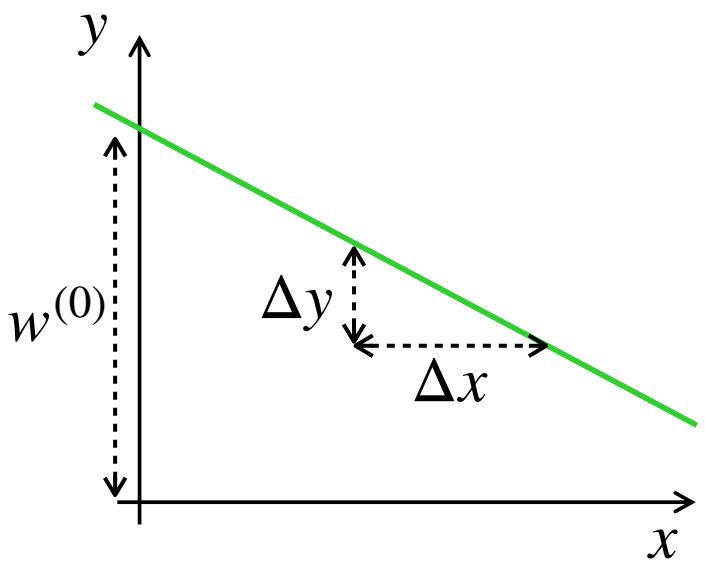
Lecture 6: k-Nearest Neighbors & Feature Expansion

Recap: A simple parametric model: The line



- Defined by 2 parameters
  - The y-intercept $w^{(0)}$
  - The slope $w^{(1)} = \dfrac{\Delta y}{\Delta x}$
- Mathematically, a line is expressed as

$$y = w^{(1)}x + w^{(0)}$$

Recap: Hyperplane

- This can be generalized to higher dimensions
- In dimension D, we can write

$$y = w^{(0)} + w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \ldots + w^{(D)}x^{(D)} = \mathbf{w}^T \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(D)} \\ 1 \end{bmatrix}$$

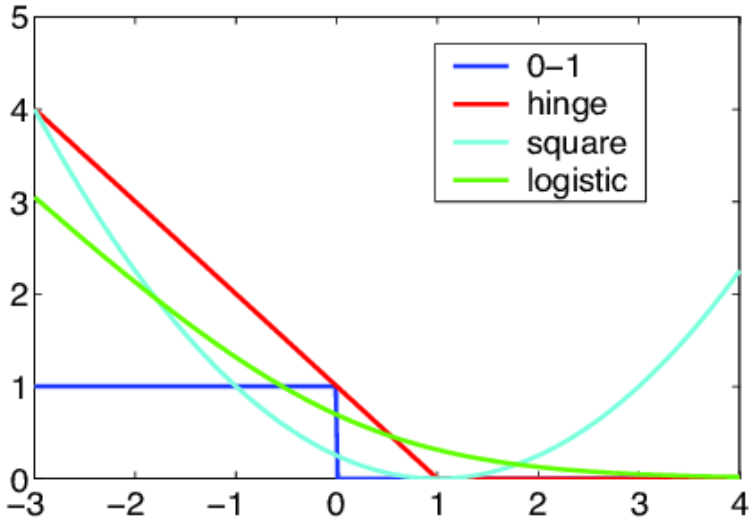
- Ultimately, whatever the dimension, we can write

$$y = \mathbf{w}^T\mathbf{x}$$

with $\mathbf{x} \in \mathbb{R}^{D+1}$, where the extra dimension contains a 1 to account for $w^{(0)}$

Recap: Linear models

- We have seen 3 algorithms that use the same linear model
- The real difference between these methods is the loss function

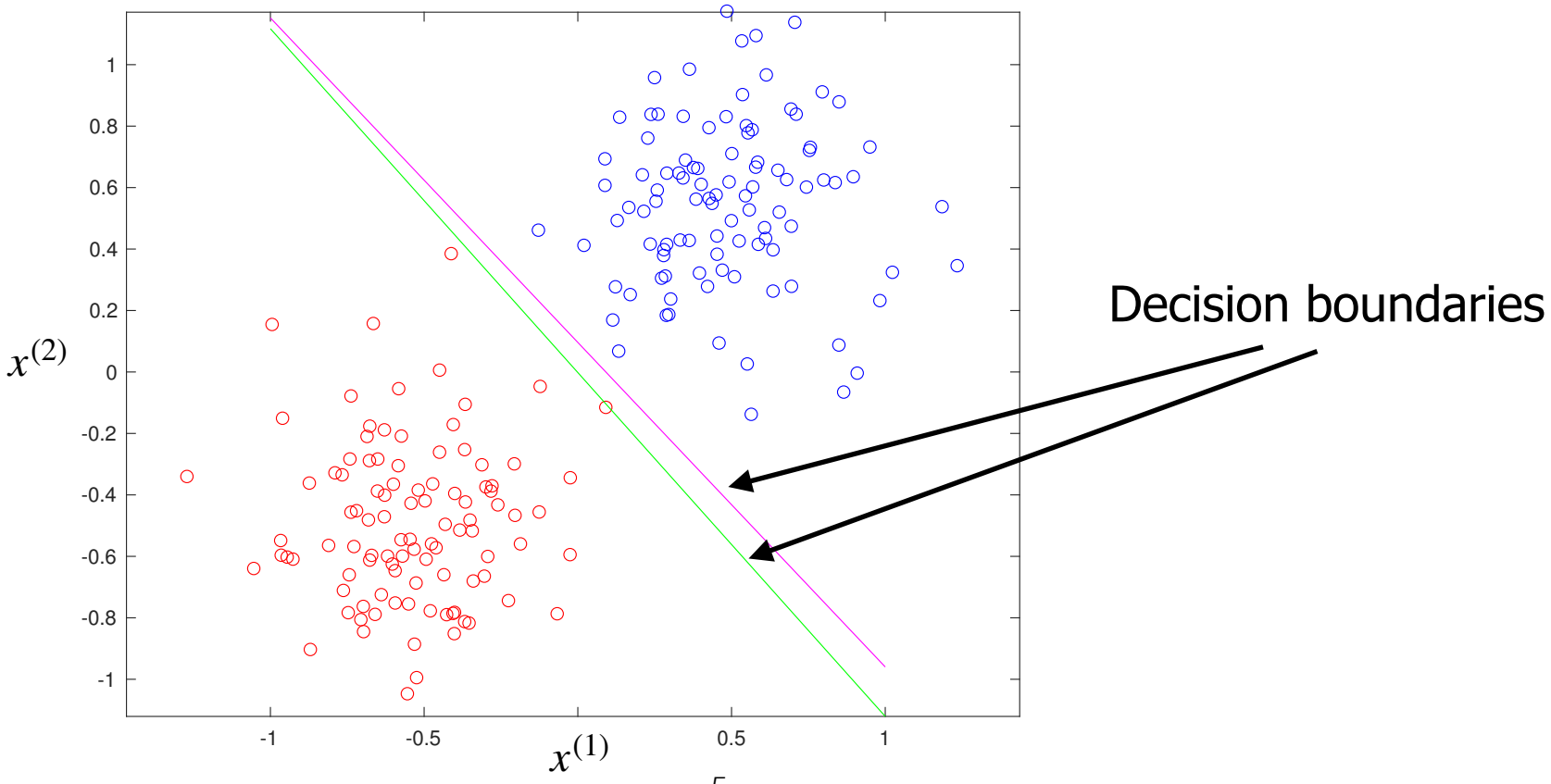


- Square: Least-square classification
- Logistic: Logistic regression
- Hinge: SVM
- 0-1: Ideal

- Choosing an appropriate loss function has an impact on the resulting algorithm

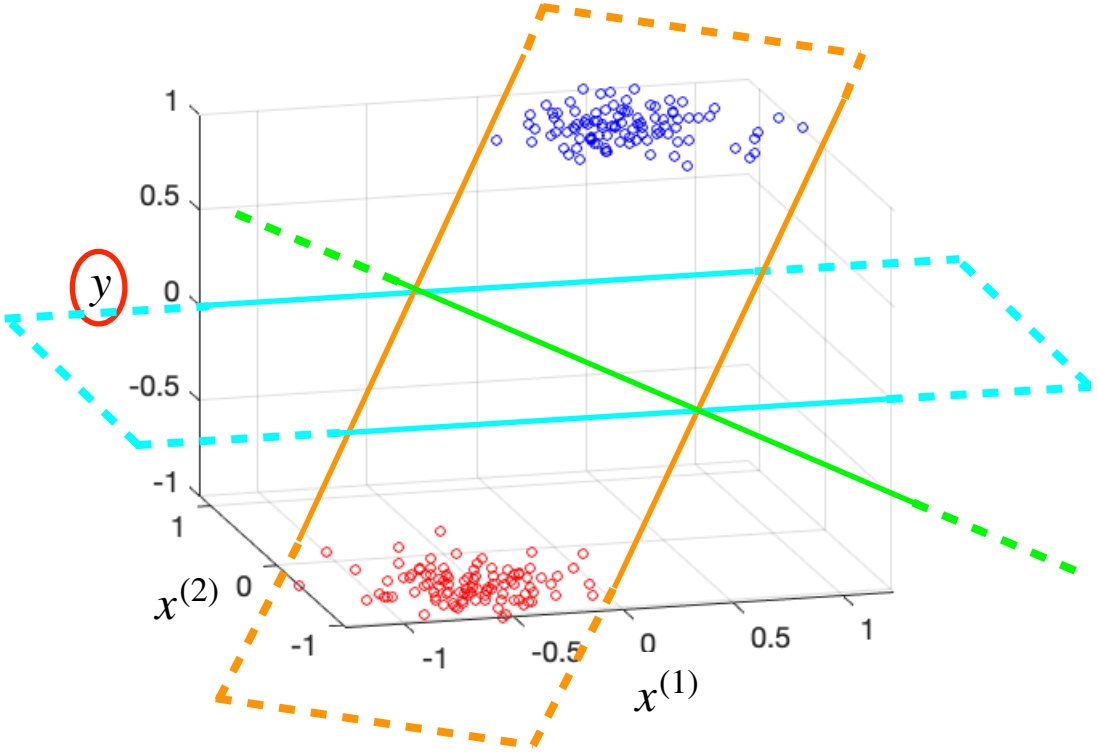
Linear model vs decision boundary

- Example: 2D inputs, 2 classes
  - Least-square classification in green, logistic regression in magenta



Linear model vs decision boundary

- The same data as before can also be seen in 3D
  - The label acts as the third dimension



- The linear model encodes the plane that passes through the samples (orange plane)
- The decision boundary (green line) is the intersection of this plane with the plane $y = 0$ (or $y = 0.5$ if the negative label is 0) (cyan plane)

Recap: From binary to multiclass classification

- Initially, the three methods we have studied could only handle two classes: positive vs negative
- In general, one would typically like to discriminate between more than two categories. E.g., for image recognition


Label 1


Label 2


Label 3

- We then extended our three linear methods can handle this scenario

Recap: Dealing with multiple classes

- The most common approach to modeling a multi-class problem consists of encoding the class label as a vector with a single 1 at the index corresponding to the category and 0s elsewhere
  - In a 5-class problem, a sample in class 2 is represented as

$$\mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

- This is referred to as *one-hot encoding* (or 1-of-C encoding)

Recap: Multi-output linear model

- Predicting such a one-hot encoding is then similar to making the model output multiple values, as we saw for multi-output regression

- We therefore again use a matrix $\mathbf{W} \in \mathbb{R}^{(D+1)\times C}$, such that

$$\hat{\mathbf{y}}_i = \mathbf{W}^T \mathbf{x}_i = \begin{bmatrix} \mathbf{w}_{(1)}^T \\ \mathbf{w}_{(2)}^T \\ \vdots \\ \mathbf{w}_{(C)}^T \end{bmatrix} \mathbf{x}_i$$

where each $\mathbf{w}_{(j)}$ is a $(D+1)$-dimensional vector, used to predict one output dimension, i.e., the score for one class

Recap: Multi-class least-square classification

- Multi-class least-square classification is then similar to a multi-output regression problem

- We can then write training as

$$\min_{\mathbf{W}} \sum_{i=1}^{N} \|\mathbf{W}^T \mathbf{x}_i - \mathbf{y}_i\|^2$$

where each $\mathbf{y}_i$ is now a C-dimensional vector with a single one at the index corresponding to the class label, and zeros everywhere else

Recap: Multi-class logistic regression

- As in the least-square classification case, for logistic regression, we can use one-hot encodings to represent multi-class labels

- Then, we again need to use a matrix $\mathbf{W} \in \mathbb{R}^{(D+1)\times C}$ to represent the model parameters

- In this case, the probability for a class k is given by the *softmax* function

$$\hat{y}^{(k)}(\mathbf{x}) = \frac{\exp(\mathbf{w}_{(k)}^T \mathbf{x})}{\sum_{j=1}^{C} \exp(\mathbf{w}_{(j)}^T \mathbf{x})}$$

Recap: Multi-class logistic regression

- The empirical risk can then be derived from the multi-class version of the cross entropy, which yields

$$R(\mathbf{W}) = -\sum_{i=1}^{N}\sum_{k=1}^{C} y_i^{(k)} \ln \hat{y}_i^{(k)}$$

- As in the binary case, a single $y_i^{(k)}$ is 1 for every sample i, so we really have one term per training sample

- As in the binary case, training is done by minimizing this loss using gradient descent

Recap: Dealing with multiple classes

- So far, we have seen how to extend binary least-square classification and binary logistic regression to the multi-class scenario. How about SVM?

- Unfortunately, there is no ideal solution to handle multiple classes in SVM

- Instead, one typically relies on using several binary classifiers
  - This can be done in a one-vs-rest or one-vs-one manner
  - Note that these two solutions are quite general, not specific to SVM

Exercises: Solutions

- You are training a multi-class least-square classifier to recognize cat, dog and car images. For the two images, $\mathbf{x}_1$, $\mathbf{x}_2$, below, what do the ground-truth vectors, $\mathbf{y}_1$, $\mathbf{y}_2$, look like numerically?
  - $\mathbf{y}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$, $\mathbf{y}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$
- Assuming the images are color images of size 32×32, what is the size of the parameter matrix $\mathbf{W}$?
  - $\mathbf{W}$ is of size $(D+1) \times C$. $D = 32 \cdot 32 \cdot 3 = 3072$ (3 for the number of color channels). $C = 3$ (number of categories).

$\mathbf{x}_1$

$\mathbf{x}_2$

Exercise: Solution

- You have trained a multi-class logistic regression classifier to recognize cat, dog and car images. For the two test images, $\mathbf{x}_1$, $\mathbf{x}_2$, below, assuming that the model predicts the correct class, what would you expect the outputs of the model, $\hat{\mathbf{y}}_1$, $\hat{\mathbf{y}}_2$, to look like numerically? Explain why.

$\mathbf{x}_1$

$\mathbf{x}_2$

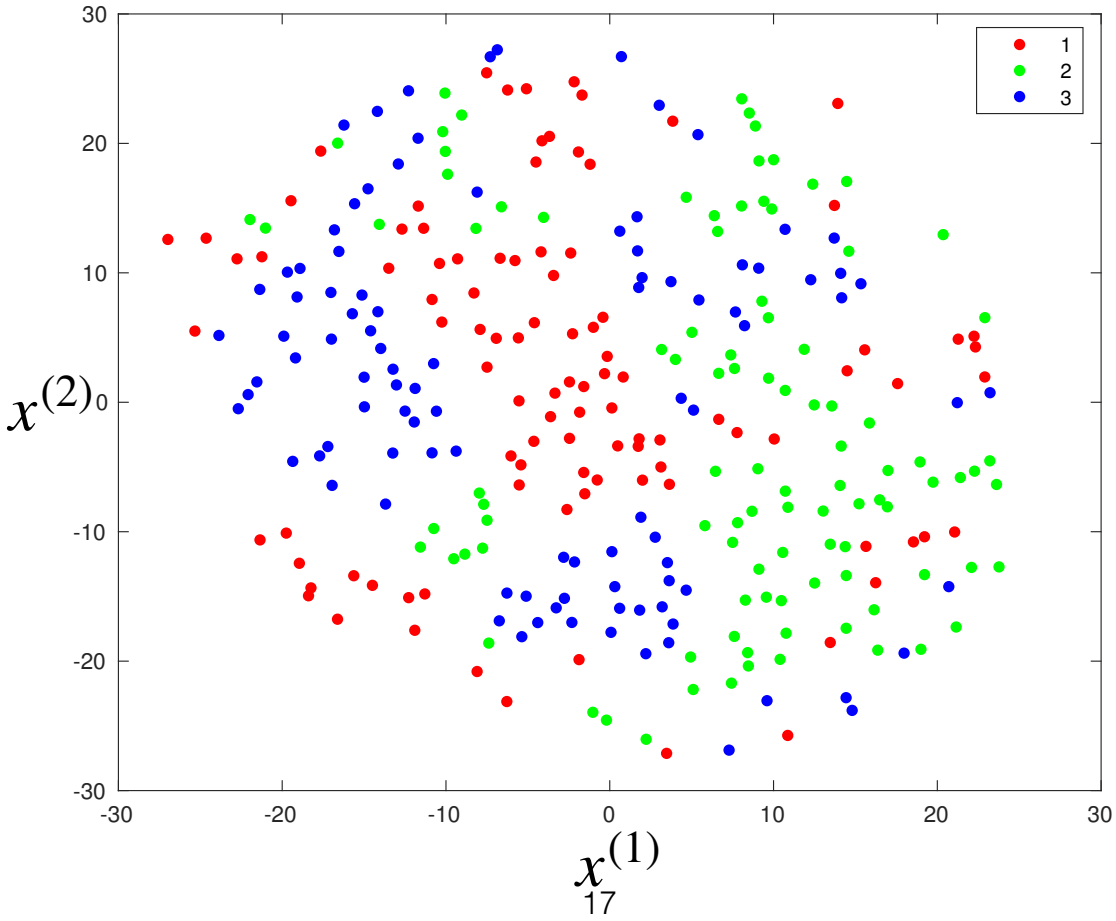
- Solution: Let us assumed that the classes are ordered as cat, dog and car. Then, I would expect:
  - $\hat{\mathbf{y}}_1 \approx \begin{bmatrix} 0.65 \\ 0.3 \\ 0.05 \end{bmatrix}$, because cats and dogs are more similar than cats and cars
  - $\hat{\mathbf{y}}_2 \approx \begin{bmatrix} 0.05 \\ 0.05 \\ 0.9 \end{bmatrix}$, because a car looks very different from both cats and dogs

Goals of today's lecture

- Move from linear to nonlinear machine learning
- Introduce the k-Nearest Neighbor method
- Introduce the idea of feature expansion

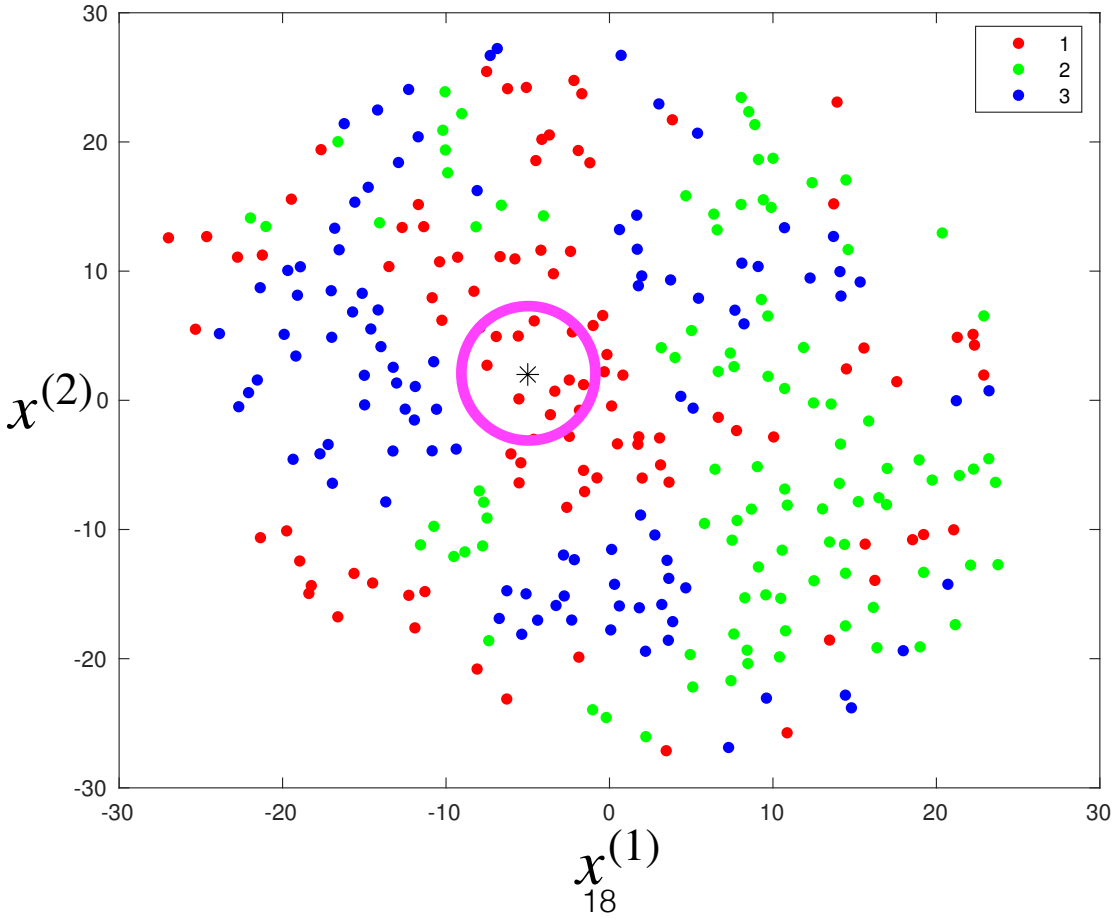
From linear to nonlinear

- Unfortunately, no linear model can directly fit this data



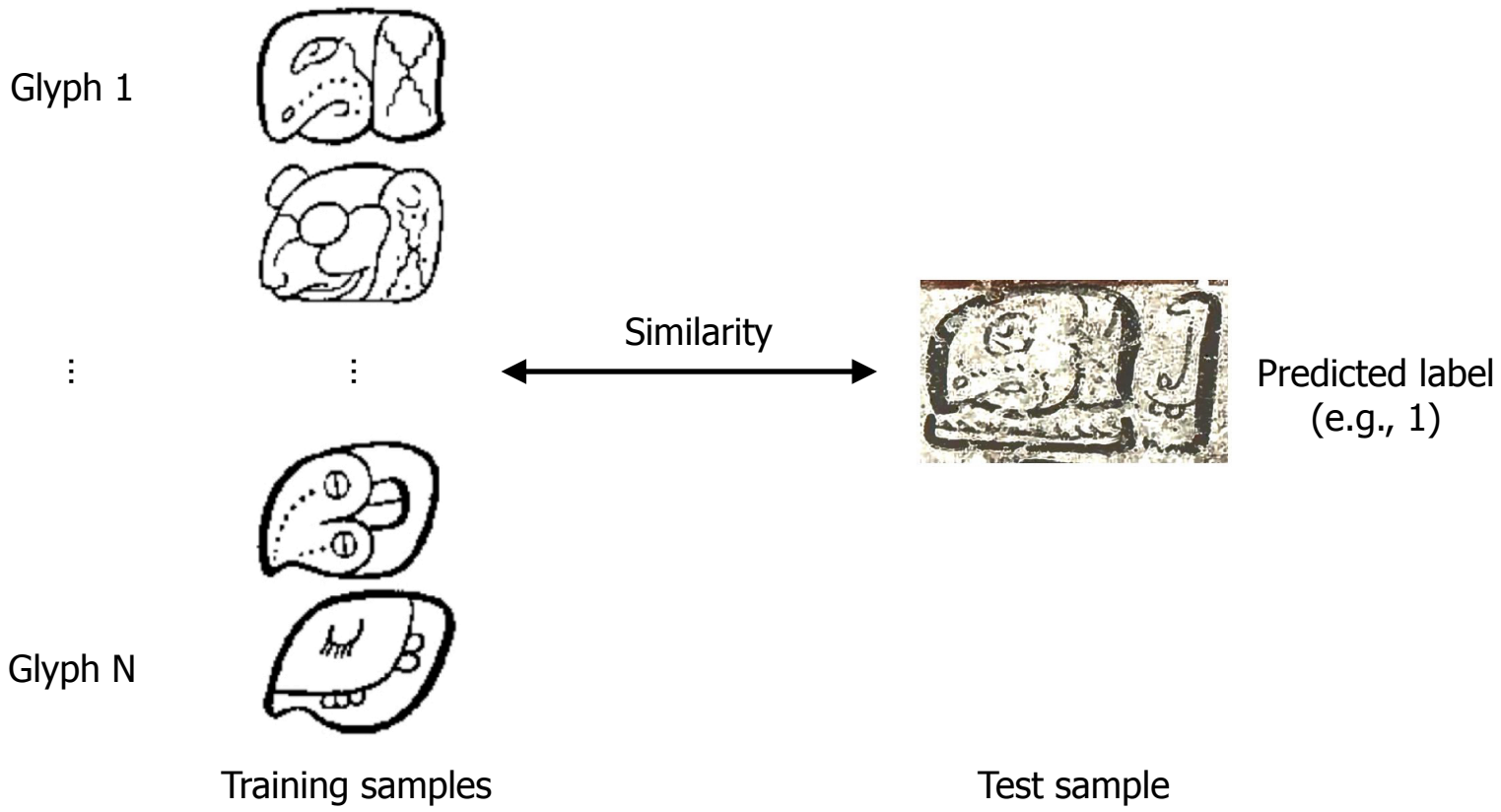
The simplest nonlinear ML algorithm

- However, assigning a class to this test sample (black star) seems very intuitive



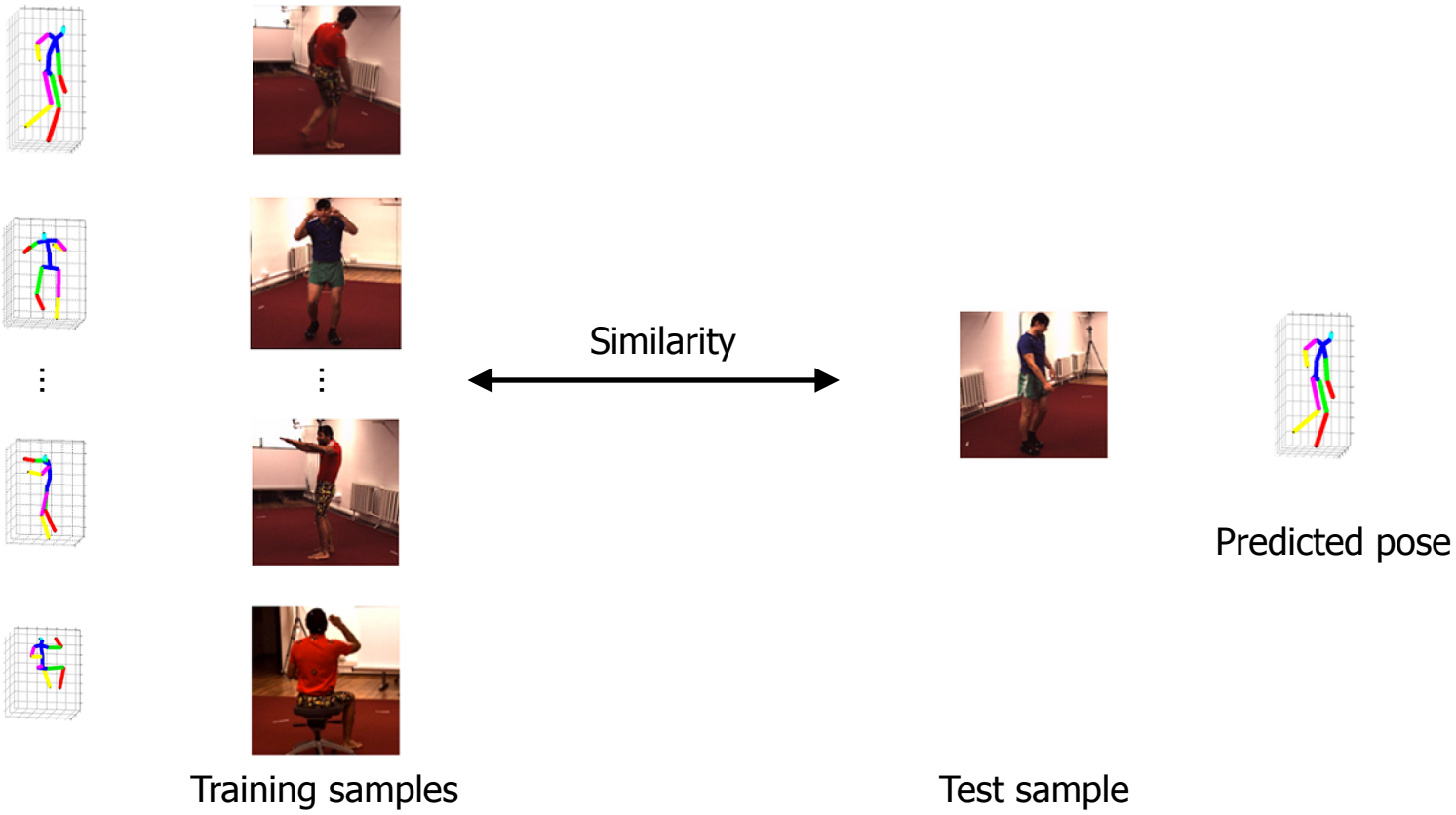
Nearest neighbor method (Bishop 2.5.2)

- Classification: Similar data samples have the same label
  - Example: Maya glyph recognition (Hu et al., 2017)



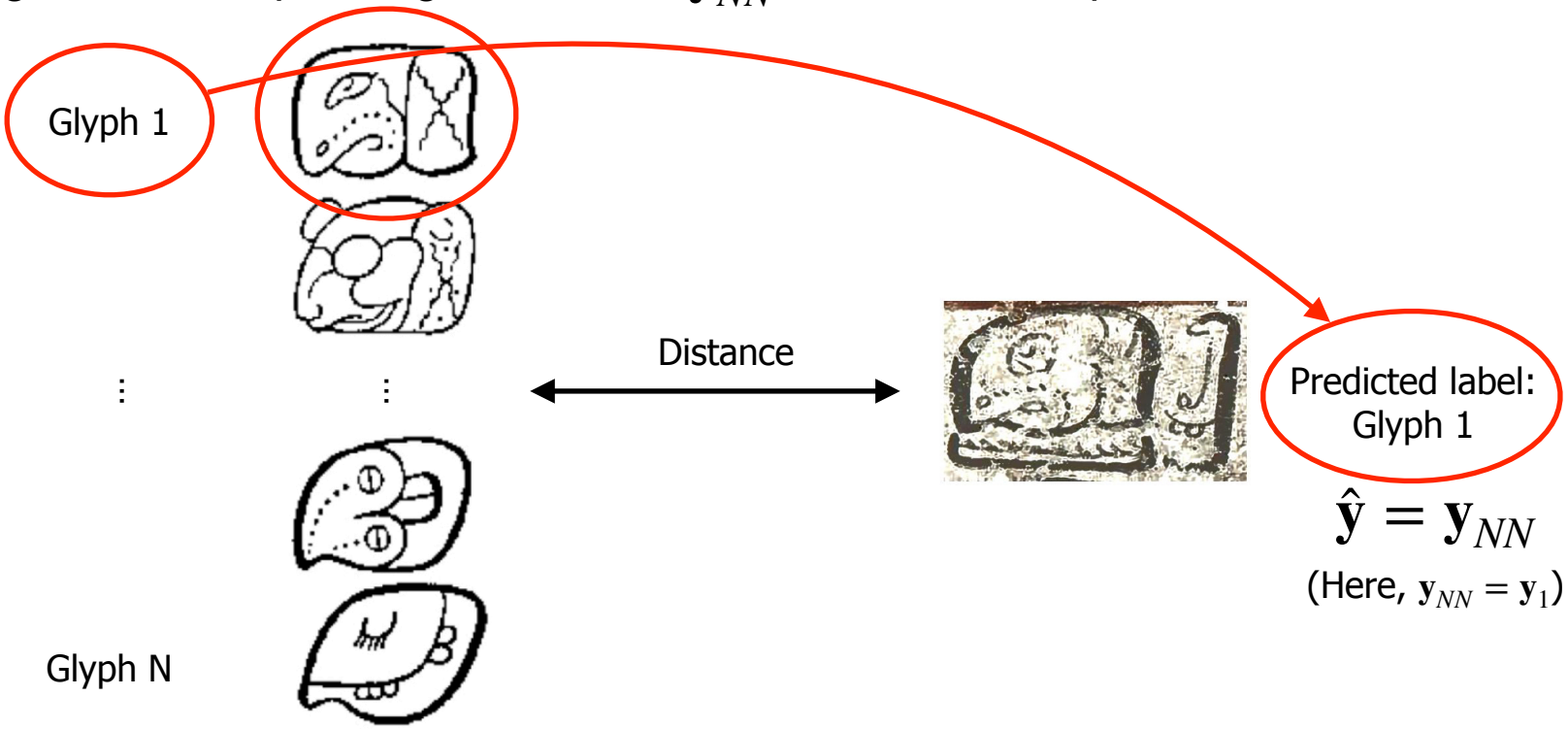
Nearest neighbor method

- Regression: Similar data samples have the same associated value
  - Example: Human pose estimation



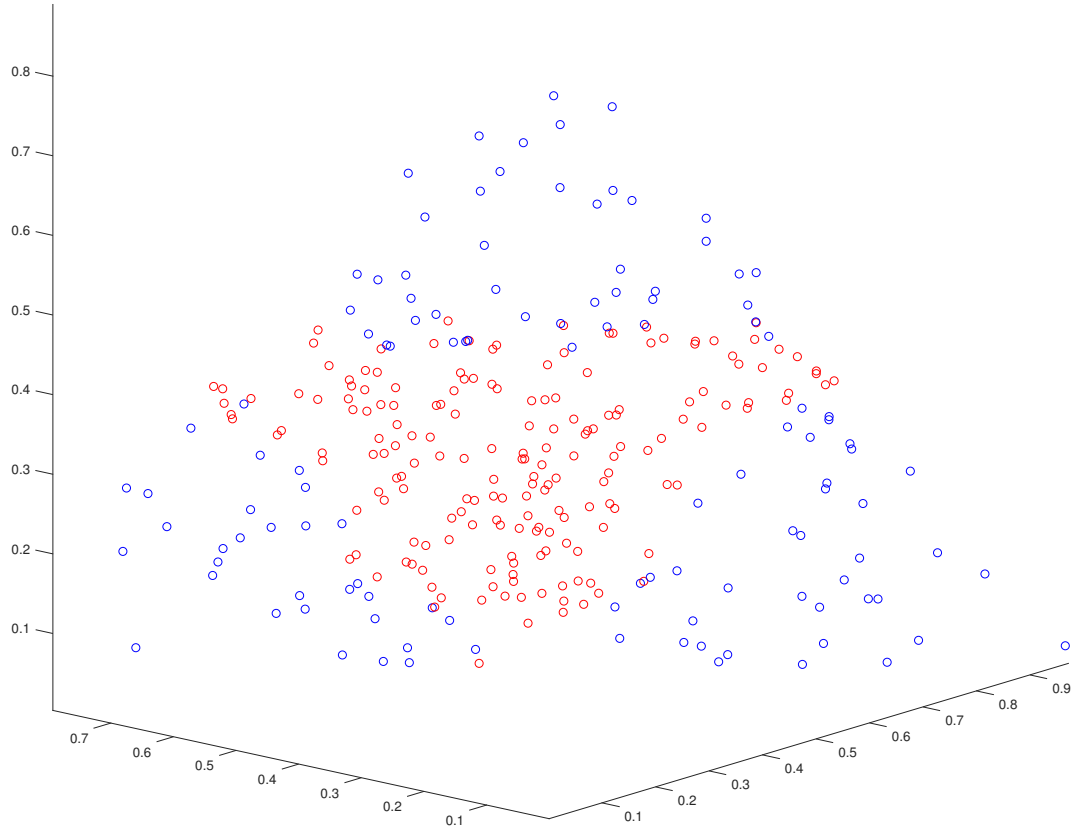
NN: Algorithm

1. Compute the distance between the test sample $\mathbf{x}$ and all training samples $\{\mathbf{x}_i\}$
2. Find the sample $\mathbf{x}_{NN}$ with minimum distance
3. Assign the corresponding label/value $\mathbf{y}_{NN}$ to the test sample



NN: Properties

- The results may change for different distance functions
  - Toy example with 3D histograms belonging to 2 classes (colors)



Interlude

Histograms

Histograms

- A histogram in $\mathbb{R}^D$ is a D-dimensional vector such that

$$x^{(d)} \in [0,1] \ , \forall 1 \leq d \leq D$$

$$\sum_{d=1}^{D} x^{(d)} = 1$$

- Given a vector in $\mathbb{R}^D$ whose values are all non-negative (and at least one is strictly positive), one can obtain a histogram by dividing each value by the sum of values

Histograms: Numerical example

- Let the original vector in $\mathbb{R}^9$ be

$$\tilde{\mathbf{x}} = [1 \quad 2 \quad 1 \quad 2 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0]^T$$

- The sum of its values is

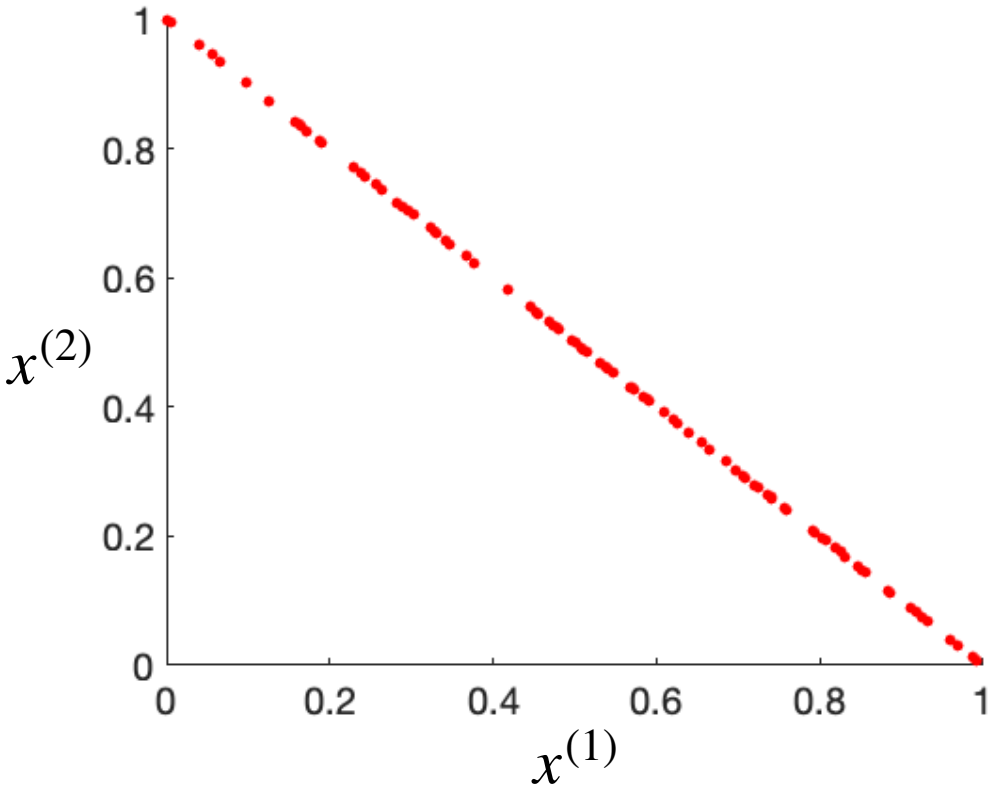
$$\sum_{d=1}^{D} \tilde{x}^{(d)} = 8$$

- Then, a histogram can be computed as

$$\mathbf{x} = [1/8 \quad 1/4 \quad 1/8 \quad 1/4 \quad 1/8 \quad 1/8 \quad 0 \quad 0 \quad 0]^T$$

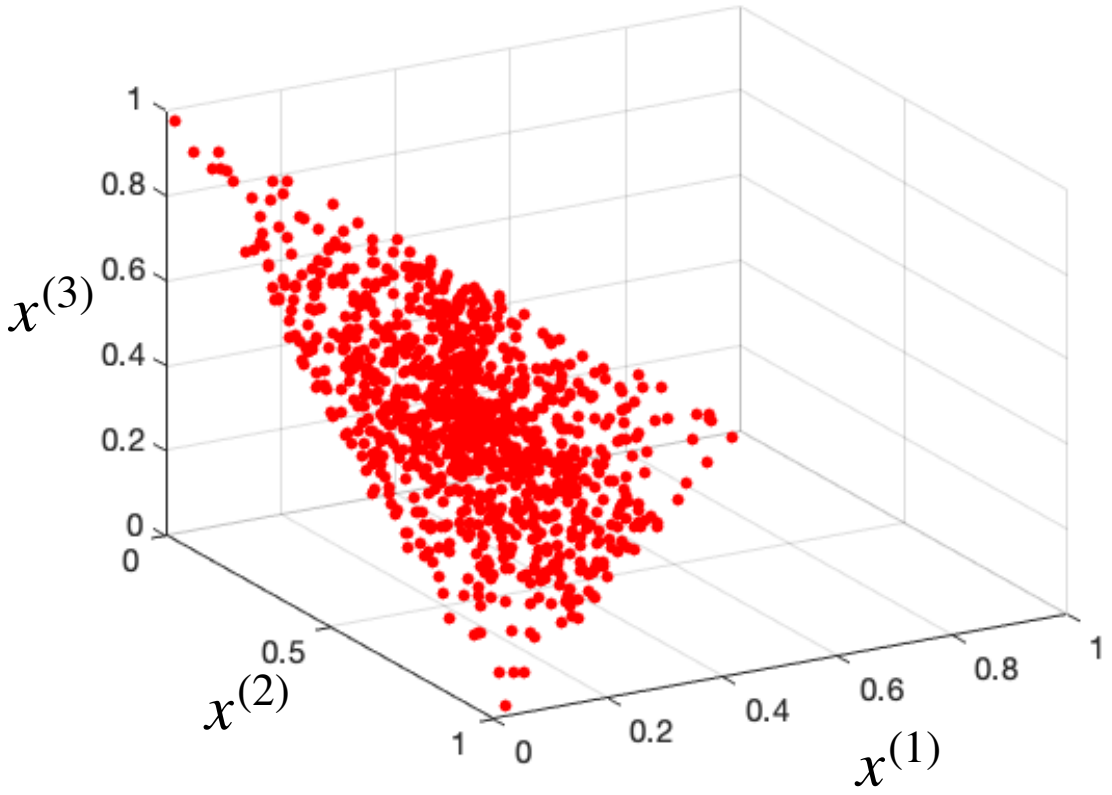
Histograms in 2D

- $N = 100$ samples in dimension $D = 2$



Histogram in 3D

- $N = 1000$ samples in dimension $D = 3$

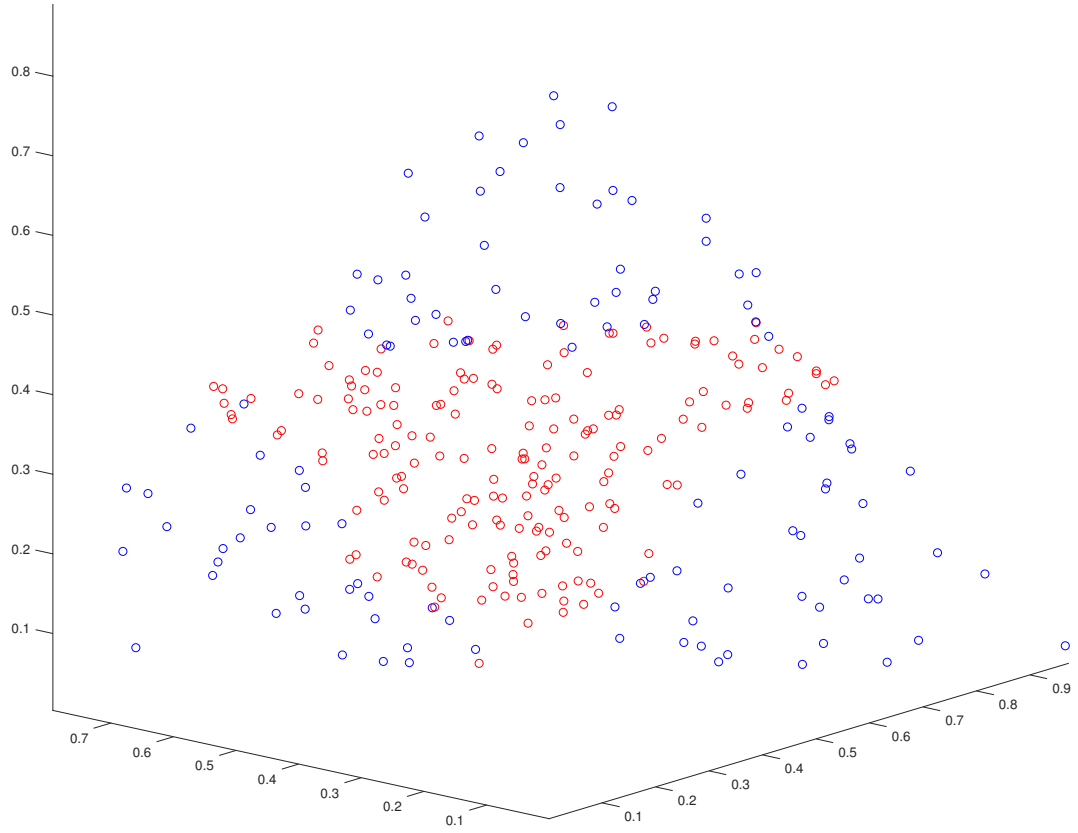


End of the interlude

Back to the NN algorithm

NN: Properties

- The results may change for different distance functions
  - Toy example with 3D histograms belonging to 2 classes (colors)



NN Example: 3D Histograms

- Default distance: Euclidean distance

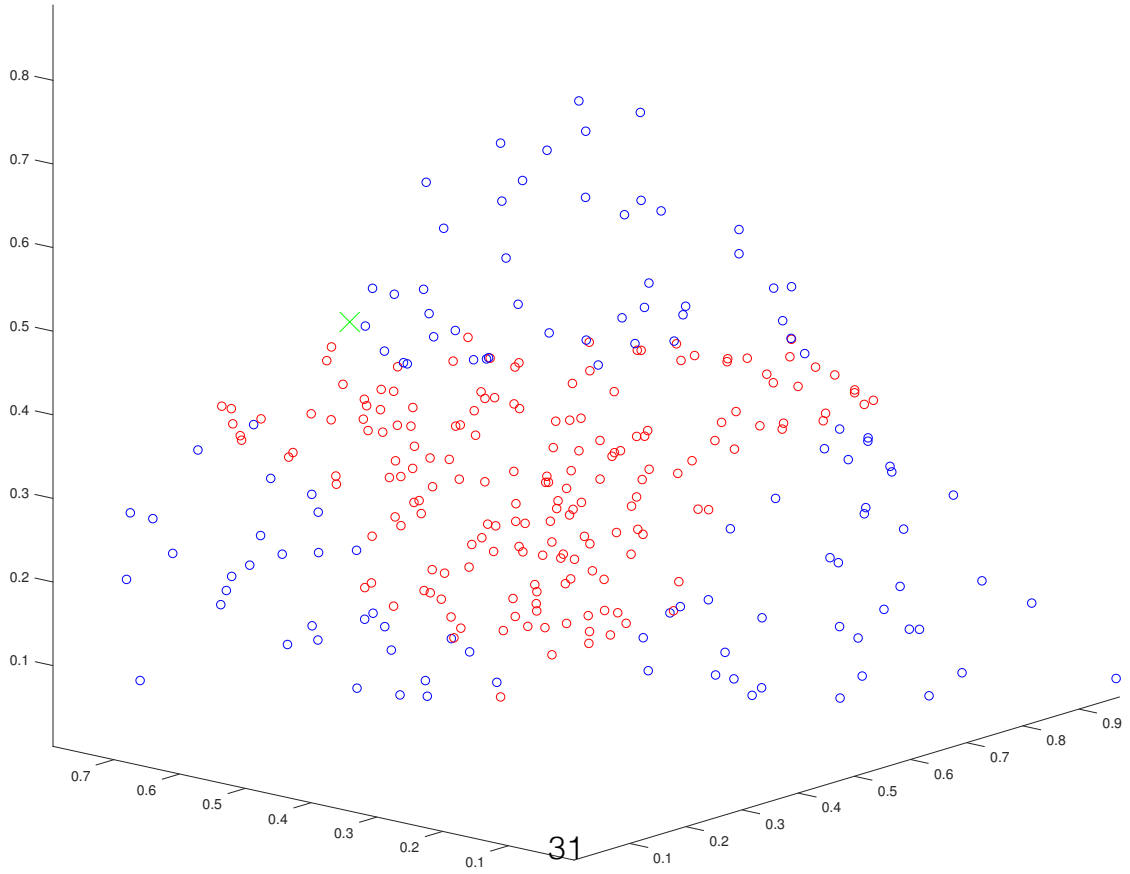
$$d(\mathbf{x}_i, \mathbf{x}) = \sqrt{\sum_{d=1}^{D} (x_i^{(d)} - x^{(d)})^2}$$

- Distance often used for histograms: Chi-square distance

$$d(\mathbf{x}_i, \mathbf{x}) = \sqrt{\chi^2(\mathbf{x}_i, \mathbf{x})} = \sqrt{\sum_{d=1}^{D} \frac{(x_i^{(d)} - x^{(d)})^2}{x_i^{(d)} + x^{(d)}}}$$

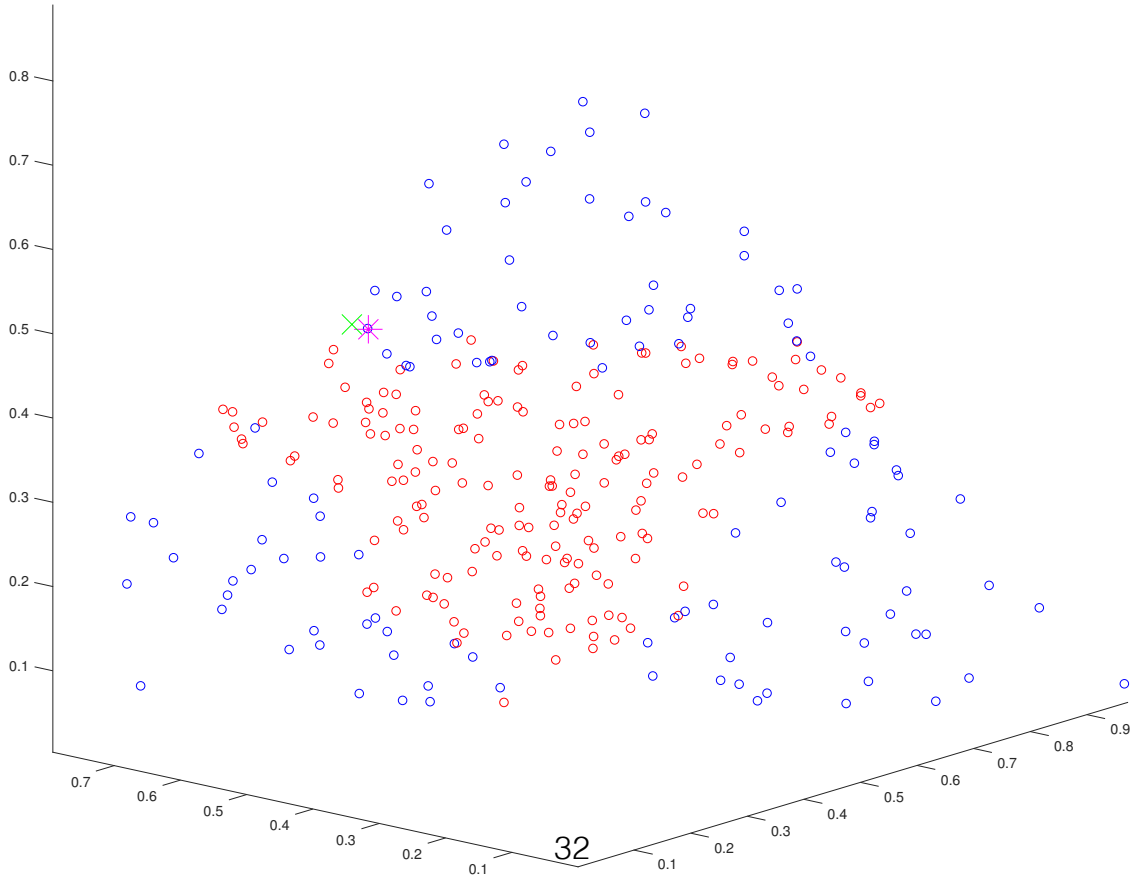
NN Example: 3D Histograms

- Test point: Green cross



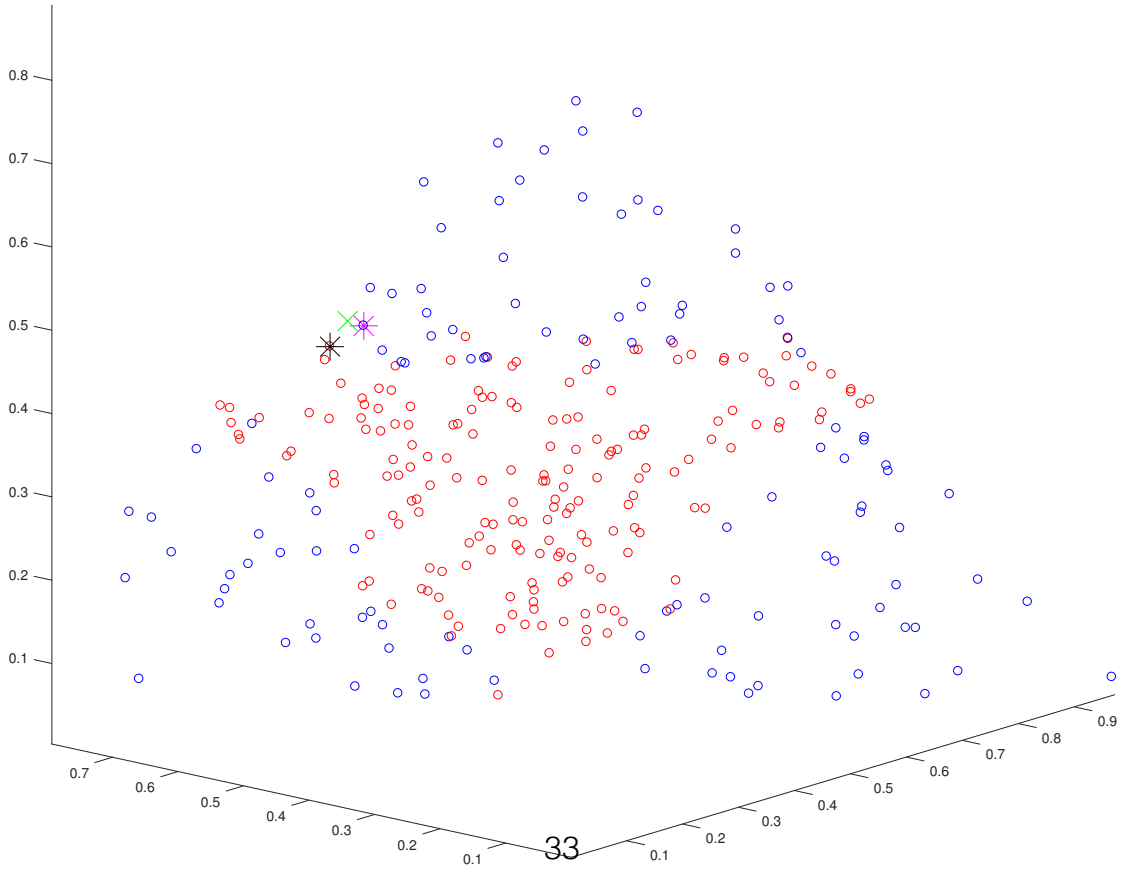
NN Example: 3D Histograms

- Test point: Green cross
- NN with Euclidean distance: Magenta star (class blue)



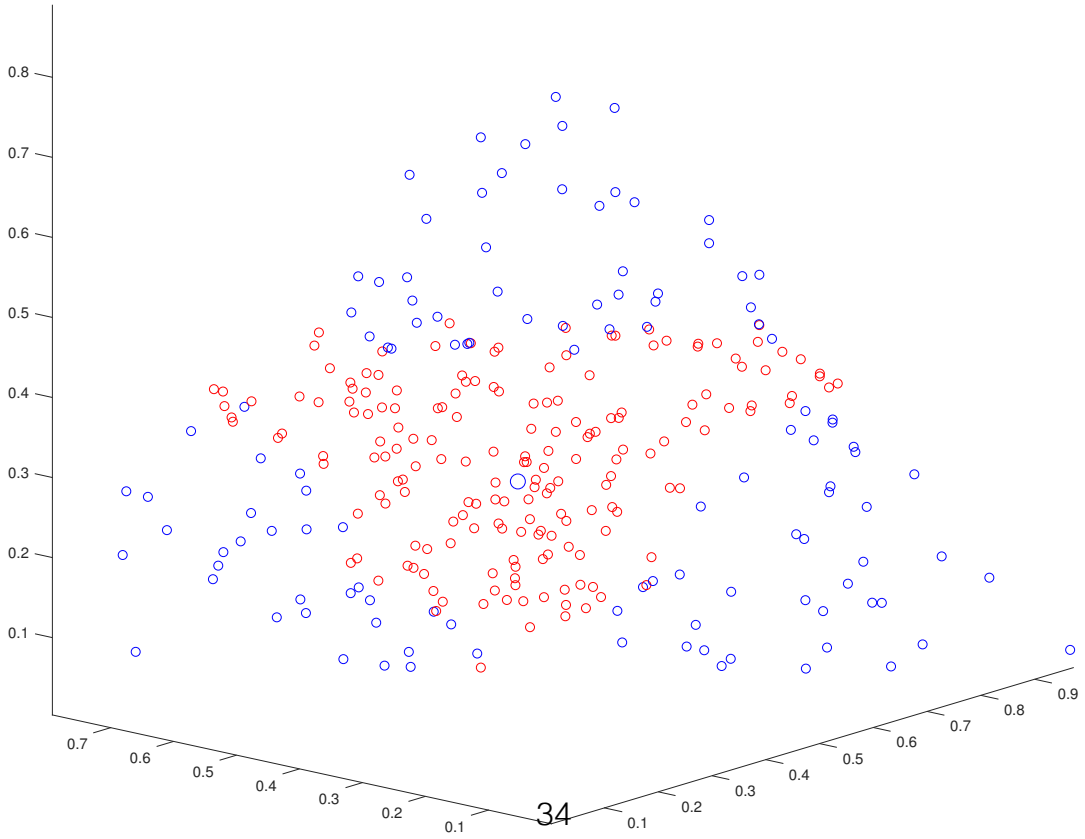
NN Example: 3D Histograms

- Test point: Green cross
- NN with Euclidean distance: Magenta star (class blue)
- NN with Chi-square distance: Black star (class red)



NN: Properties

- The results may be sensitive to outliers
  - A point close to the outlier (blue circle in the center) will be misclassified
  - Solution: Look at multiple neighbors instead of just one



k-Nearest Neighbors

- Classification:

1. Compute the distance between the test sample $\mathbf{x}$ and all training samples $\{\mathbf{x}_i\}$
2. Find the k samples $\{\mathbf{x}_{NN1}, \ldots, \mathbf{x}_{NNk}\}$ with minimum distances
3. Find the most common label among these k nearest neighbors (majority vote)
4. Assign the corresponding label $\mathbf{y}_{MV}$ to the test sample

- If several labels appear the same number of times among the k-NN, one strategy consists of taking the one whose samples have the smallest average distance to the test sample

k-Nearest Neighbors

- Regression:

1. Compute the distance between the test sample $\mathbf{x}$ and all training samples $\{\mathbf{x}_i\}$
2. Find the k samples $\{\mathbf{x}_{NN1}, \ldots, \mathbf{x}_{NNk}\}$ with minimum distances
3. Compute the value $\hat{\mathbf{y}}$ for the test sample based on that of these k-NN

- Two strategies:
  - Use the average of the k-NN values $\{\mathbf{y}_{NNi}\}$

$$\hat{\mathbf{y}} = \frac{1}{k}\sum_{i=1}^{k}\mathbf{y}_{NNi}$$

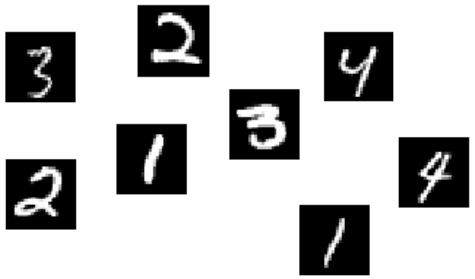
  - Question: How else would you compute the predicted value $\hat{\mathbf{y}}$?

k-Nearest Neighbors: Demo

- http://vision.stanford.edu/teaching/cs231n-demos/knn/

k-Nearest Neighbors: Example

- MNIST classification



Method	Preprocessing	Error	Reference
K-nearest-neighbors, Euclidean (L2)	none	5.0	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	none	3.09	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	none	2.83	Kenneth Wilder, U. Chicago
K-nearest-neighbors, Euclidean (L2)	deskewing	2.4	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	deskewing, noise removal, blurring	1.80	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring	1.73	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 1 pixel shift	1.33	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 2 pixel shift	1.22	Kenneth Wilder, U. Chicago
K-NN with non-linear deformation (IDM)	shiftable edges	0.54	Keysers et al. IEEE PAMI 2007
K-NN with non-linear deformation (P2DHMDM)	shiftable edges	0.52	Keysers et al. IEEE PAMI 2007
K-NN, Tangent Distance	subsampling to 16x16 pixels	1.1	LeCun et al. 1998
K-NN, shape context matching	shape context feature extraction	0.63	Belongie et al. IEEE PAMI 2002

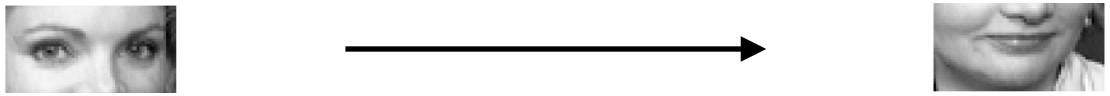
k-Nearest Neighbors: Example

- Regression on UCI datasets:
  - Howley & Madden, AICS 2007 (k-NN with data-driven distance metric)
  - Abalone dataset:
    - Predict the age of abalone from 11 attributes, such as sex, length, diameter, height,…

	No. Samples	No. Attributes	Average Test Error	
			Euclidean	KTree
Abalone*	4177	11	2.69	**2.60**

Recap: Multi-output linear regression: Example

- Face completion:
  - Example from https://scikit-learn.org/stable/auto_examples/plot_multioutput_face_completion.html
  - Task: Given the top half image of a face, predict the bottom half

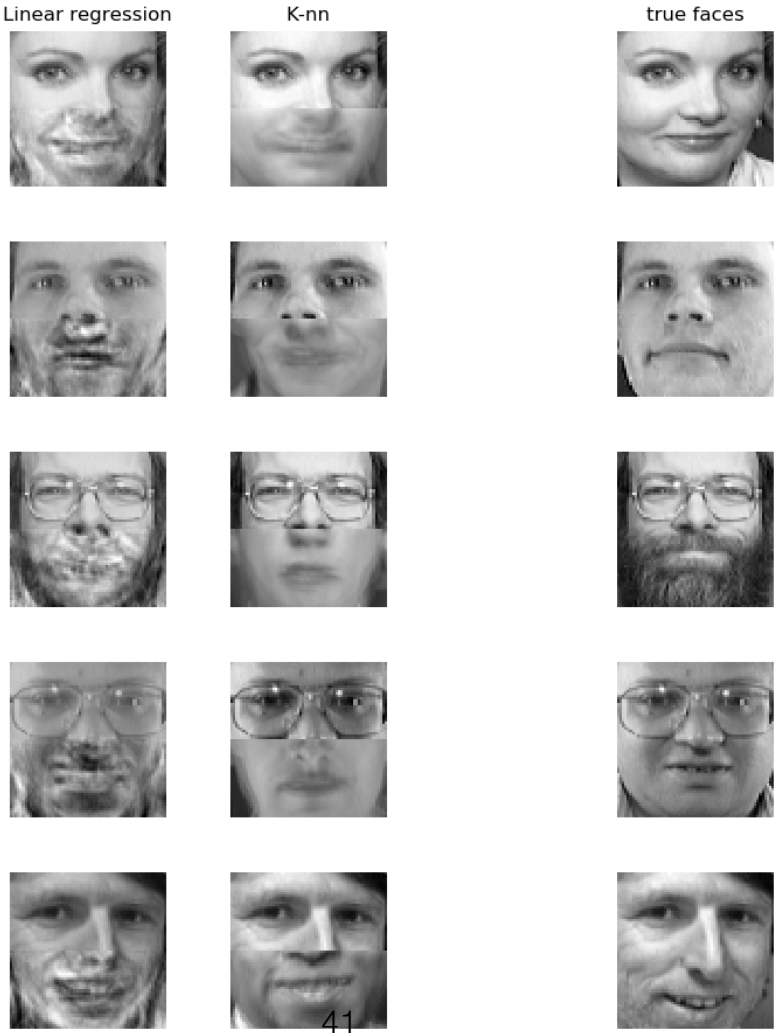


Input Model (e.g., linear regression) Output

  - Dataset: Olivetti faces
    - 40 subjects (35 for training, 5 for testing)
    - 10 images per subject

Multi-output linear regression: Example

- Face completion:
  - Results:



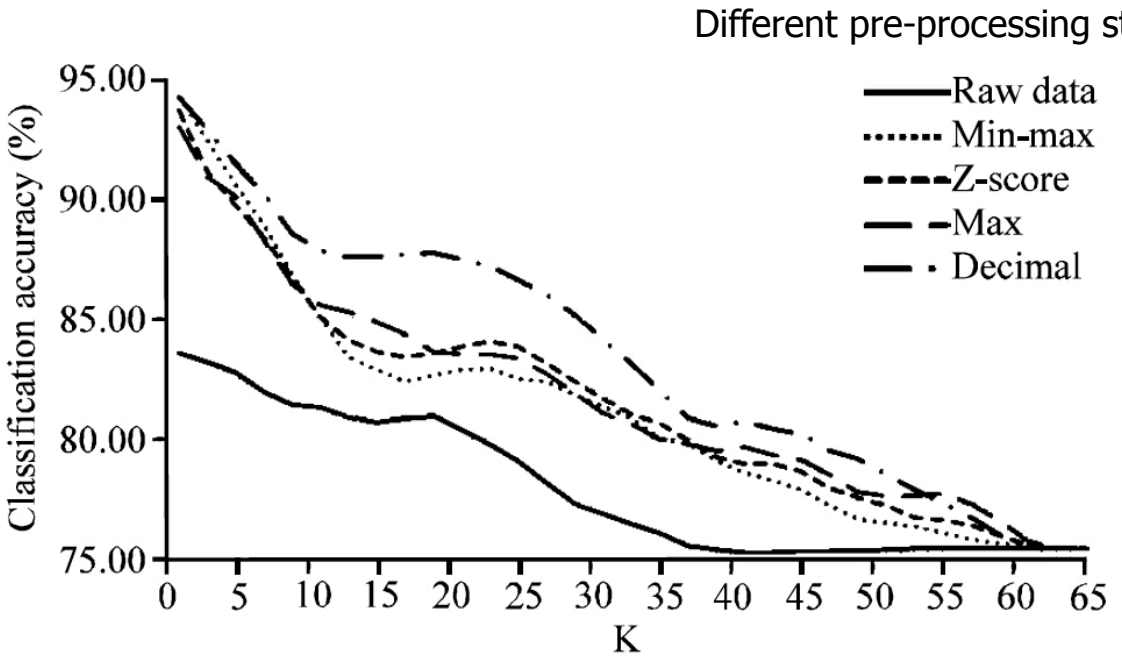
k-Nearest Neighbors: Properties

- No learning per se! What we need is
  - A good data representation
  - a distance function
  - a given value k
- This is referred to as a *non-parametric* learning method
  - No parameters to optimize during training
  - k is said to be a *hyper-parameter*, set manually (or by validation, which we will discuss in a future lecture)
- Nevertheless, k-NN can give very good results
  - E.g., MNIST digit recognition: 0.52% error

K-NN with non-linear deformation (P2DHMDM)	shiftable edges	0.52	Keysers et al. IEEE PAMI 2007

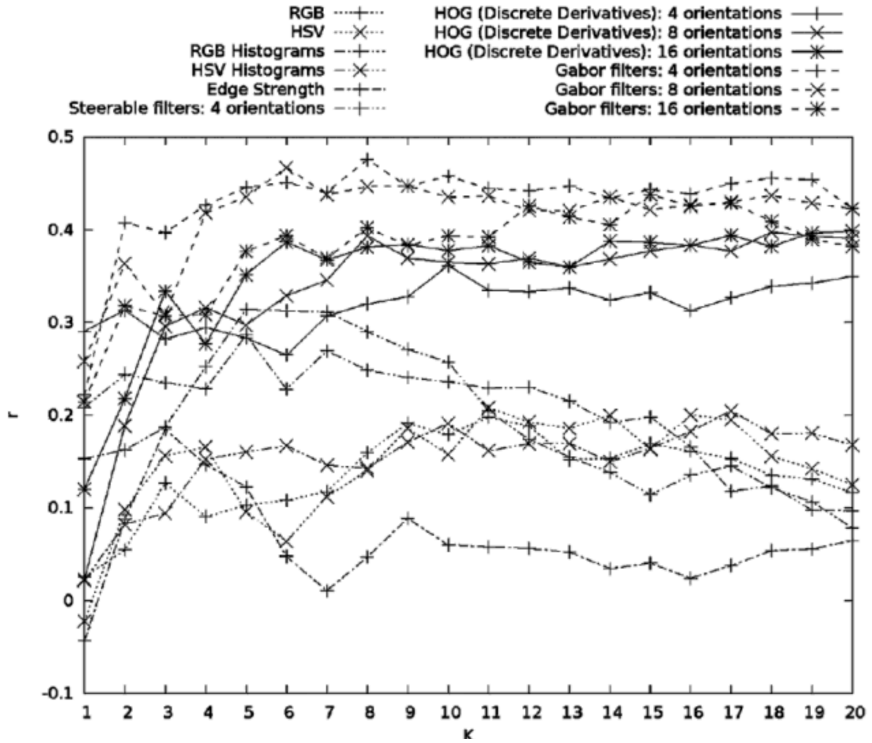
k-Nearest Neighbors: Properties

- The results depend on the value k
  - E.g., Ma et al., Journal of Applied Sciences, 2014
  - UCI Parkinson dataset: Predict if a patient suffers from Parkinson's disease from several biomedical voice measurements



k-Nearest Neighbors: Properties

- The results depend on the value k
  - E.g., Dee et al., “Visual digital humanities: using image data to derive approximate metadata”, 2016
  - Predict the year of a painting from an image



- Solution: Cross-validation —> Next week

k-Nearest Neighbors: Properties

- Computationally expensive
  - For each test sample, one needs to compute the distances to all training samples
- Solution: Approximate nearest neighbor search
  - Hashing
  - kd-trees

Approximate k-Nearest Neighbors: Example

- Human pose estimation: Shakhnarovitch et al., ICCV 2003
  - Use k-NN with hashing to retrieve the most similar training image

Synthetic training data (for which you know the true pose)

Real test sample

Top match

k-Nearest Neighbors: Properties

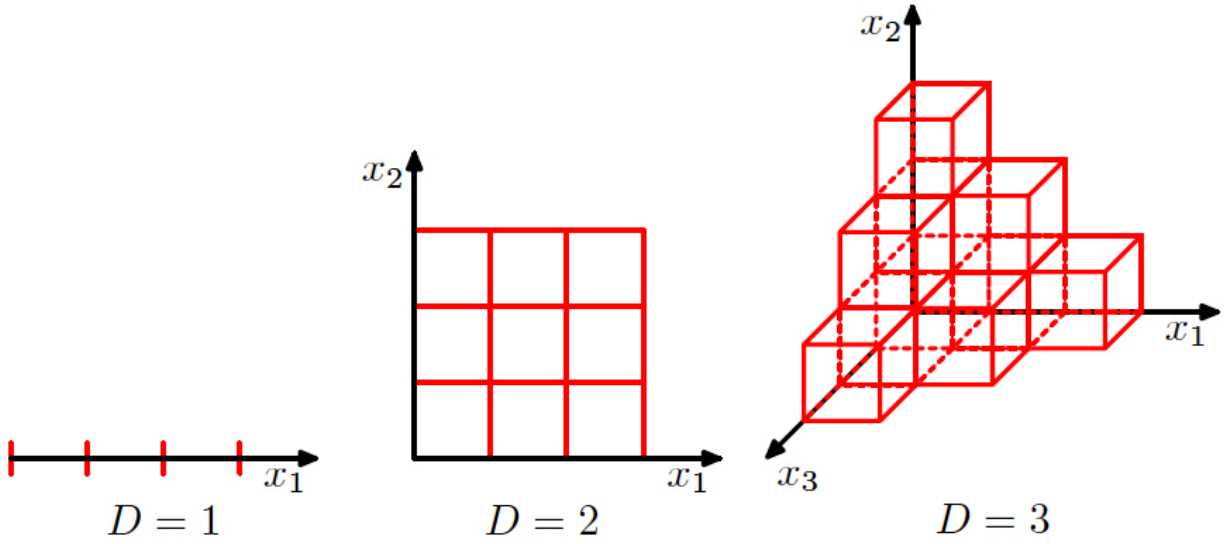
- Curse of dimensionality:
  - In the real world, data representations tend to be high dimensional:
    - E.g., vectorizing an image yields a huge vector

$$\mathbf{x}_i \in \mathbb{R}^{536 \cdot 356 \cdot 3} = \mathbb{R}^{572448}$$

  - In such high dimension, all points tend to be far apart

Curse of dimensionality (Bishop 1.4)

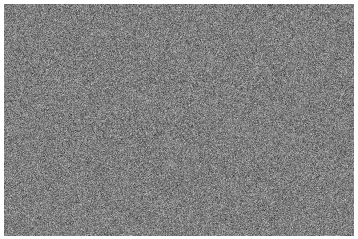
- Let us divide the input space into regular cells
- For dimensions $D = 1,2,3$ this gives



- The number of cells grows exponentially
- k-NN would then require an exponential growth of the data to cover the entire space

Curse of dimensionality

- Handling high-dimensional data also makes k-NN memory inefficient
  - We need to store all the training data to compute distances
- Fortunately, real data often lies in much smaller subspaces
  - E.g., the two images below lie in the same space, but we are more likely to observe the left one in the real world



- In a few weeks, we will talk about how to find the low-dimensional subspace of a given dataset

k-Nearest Neighbors: Summary

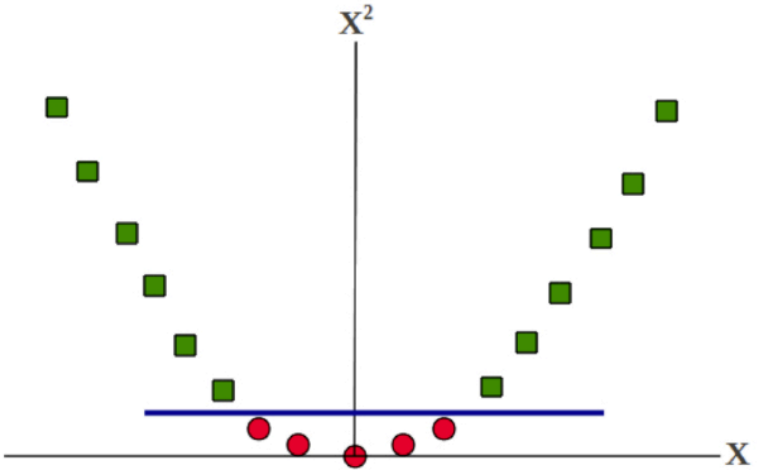
- Advantages:
  - Simple method
  - Effective to handle nonlinear data
  - Only requires defining one parameter (k)
- Drawbacks:
  - The results depend on k
  - Becomes expensive as N and/or D grow
  - May be unreliable with a large D
- Let us now look at a different way to handle nonlinear data

Nonlinear classification

- How can we handle this data?
  - 1D input, 2 classes (colors)

  - How about transforming the input?

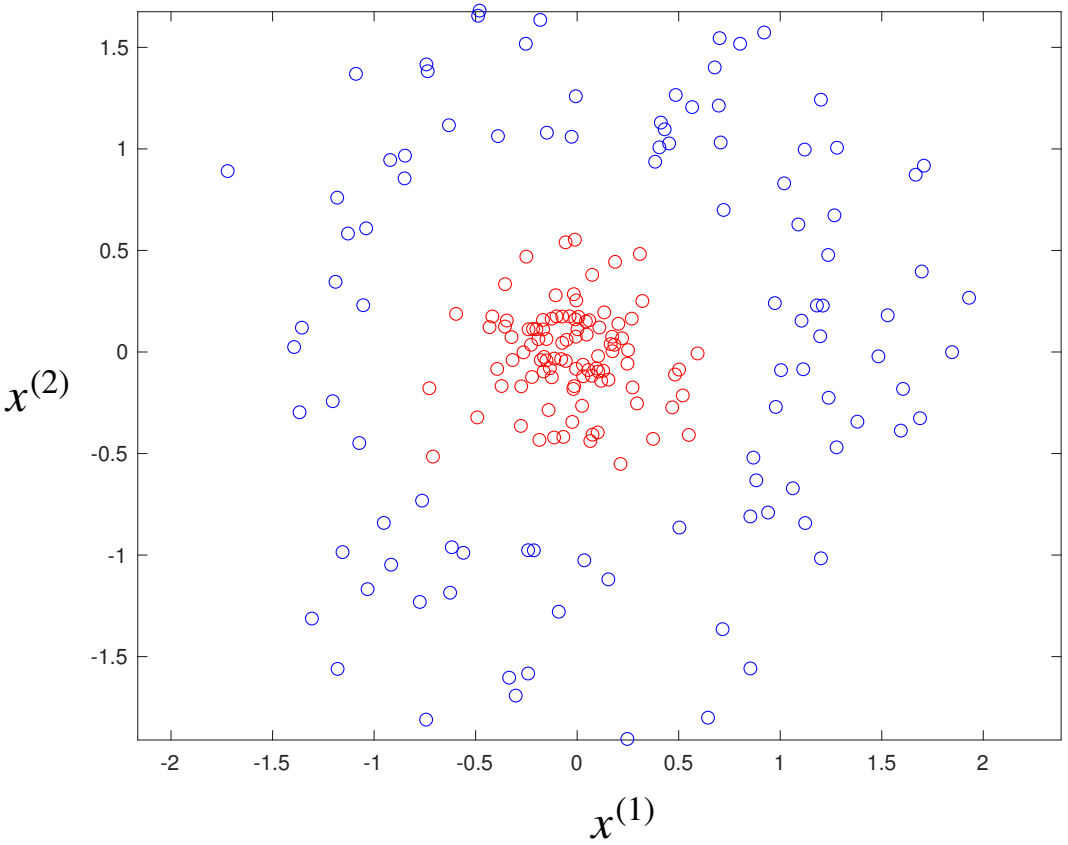
$$x \rightarrow [x, x^2]$$



Figures from Edith Law's slides

Nonlinear classification

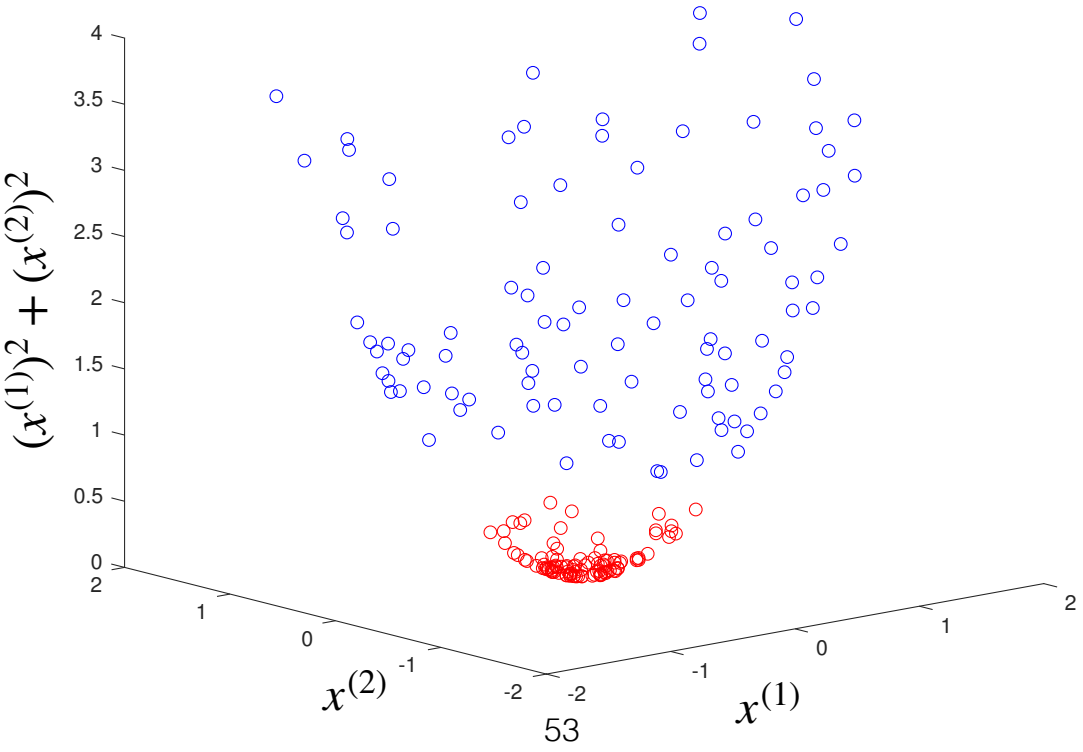
- How about this data?
  - 2D input, 2 classes (colors)



Nonlinear classification

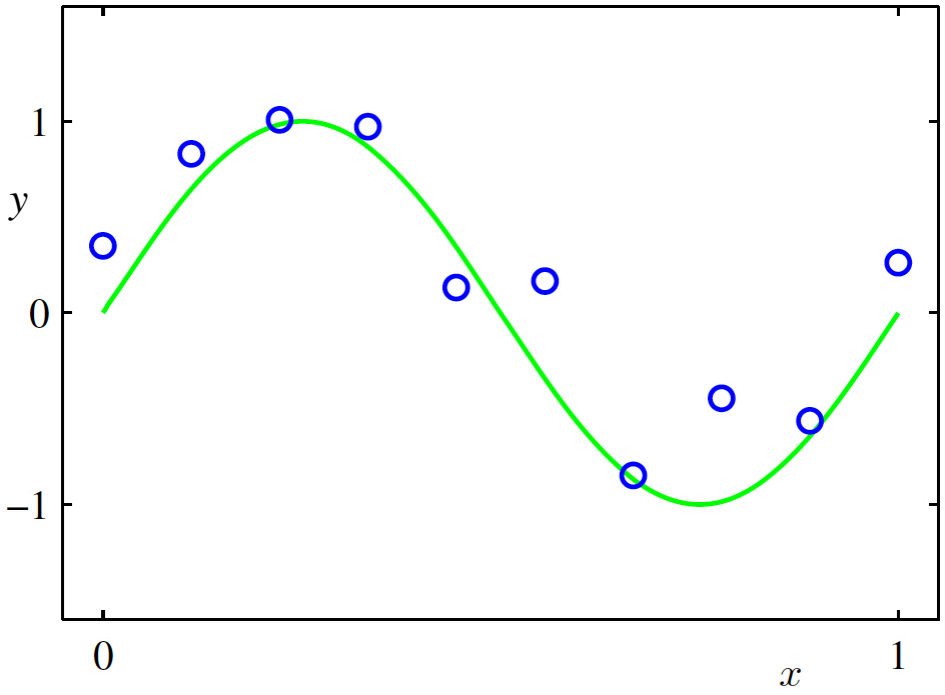
- Again, how about transforming the input?

$$\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} \rightarrow \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ (x^{(1)})^2 + (x^{(2)})^2 \end{bmatrix}$$



Polynomial curve fitting

- We have noisy training observations (blue circles, 1D input, 1D output) coming from the true green curve



Polynomial curve fitting

- We seek to find a polynomial function that approximates the true curve using these observations
- Such a polynomial function of degree M can be written as

$$y = w^{(0)} + w^{(1)}x + w^{(2)}x^2 + \cdots + w^{(M)}x^M = \sum_{j=0}^{M} w^{(j)}x^j$$

where the $\{w^{(j)}\}$ are the coefficients of the different terms, but x^j represents x to the power j

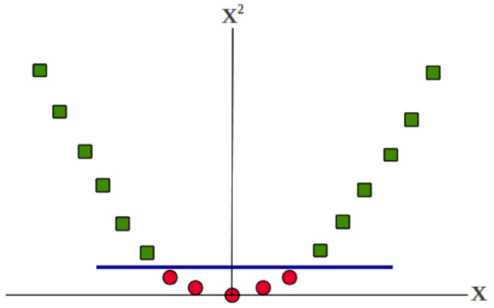
- For example:
  - For $M = 0$, we have the constant function $y = w^{(0)}$
  - For $M = 1$, we have the line $y = w^{(0)} + w^{(1)}x$
  - For $M = 2$, we have the quadratic function $y = w^{(0)} + w^{(1)}x + w^{(2)}x^2$

Polynomial feature expansion

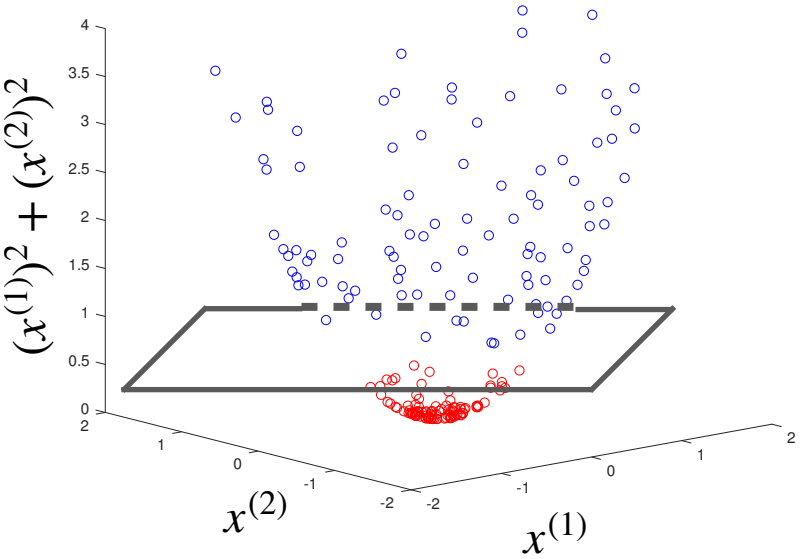
- In essence, in the three examples, we performed a change of variables
  - In the first case: $x \rightarrow \phi(x) = [x, x^2]$
  - In the second case: $\mathbf{x} \rightarrow \phi(\mathbf{x}) = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ (x^{(1)})^2 + (x^{(2)})^2 \end{bmatrix}$
  - In the last (polynomial) case: $x \rightarrow \phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix}$

Polynomial feature expansion

- We can then apply a linear model to the resulting variables
  - In the first case:



  - In the second case:



Polynomial feature expansion

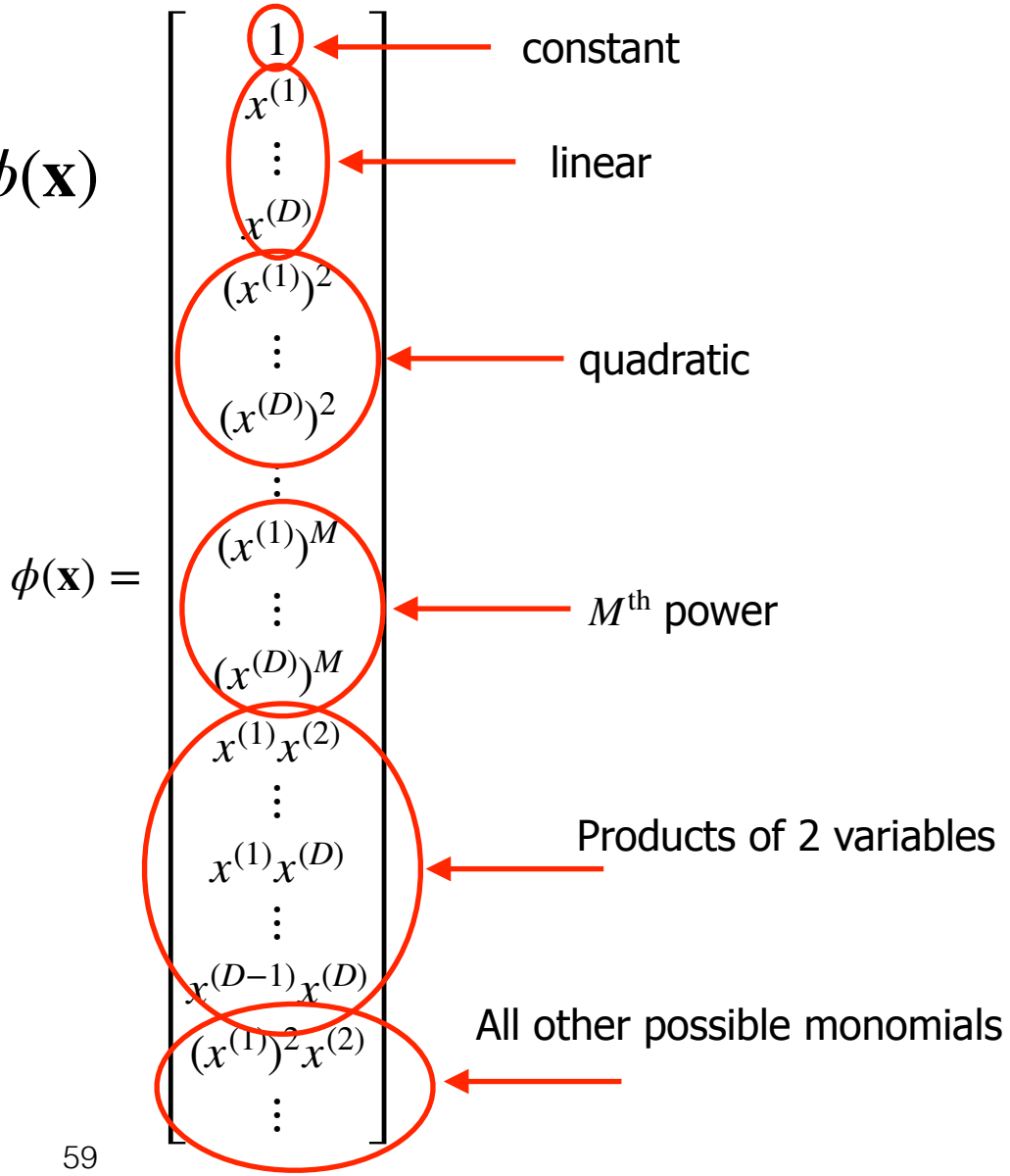
- We can then apply a linear model to the resulting variables
  - In the last case, we have

$$\sum_{j=0}^{M} w^{(j)} x^j = \mathbf{w}^T \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix}$$

Polynomial feature expansion

- In general, for D-dimensional inputs, we can obtain a large $\phi(\mathbf{x})$

- The resulting expanded features can then be used in any linear algorithm that we have seen previously
  - Let us look at how this would work for linear regression



Working with expanded features

- In essence, polynomial feature expansion uses a mapping from $\mathbf{x} \in \mathbb{R}^D$ to another representation $\phi(\mathbf{x}) \in \mathbb{R}^F$, where $F \gg D$

- We can then use a linear model in this new space and write

$$\hat{y}_i = \mathbf{w}^T \phi(\mathbf{x}_i)$$

where now $\mathbf{w} \in \mathbb{R}^F$

(assuming that the 1 accounting for the bias has been incorporated in $\phi(\mathbf{x}_i)$)

Working with expanded features: Training

- For linear regression, we used the least-square loss

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \left(\mathbf{w}^T \mathbf{x}_i - y_i\right)^2$$

- With our expanded features, we can re-write this as

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \left(\mathbf{w}^T \phi(\mathbf{x}_i) - y_i\right)^2$$

Note that the $\mathbf{w}$ vectors in the two equations above are different:

- In the first, $\mathbf{w} \in \mathbb{R}^{D+1}$
- In the second, $\mathbf{w} \in \mathbb{R}^{F}$

Working with expanded features: Gradient

- This new function is still convex, and its gradient is given by

$$\nabla R = 2\sum_{i=1}^{N} \phi(\mathbf{x}_i)\left(\phi(\mathbf{x}_i)^T\mathbf{w} - y_i\right)$$

- Setting it to zero means that

$$\left(\sum_{i=1}^{N} \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^T\right)\mathbf{w}^* = \sum_{i=1}^{N} \phi(\mathbf{x}_i)y_i$$

$F \times F$ $F \times 1$ vector $F \times 1$ vector

Working with expanded features: Solution

- We can group the transformed inputs $\{\phi(\mathbf{x}_i)\}$ and the outputs $\{y_i\}$ in a matrix and vector of the form

$$\Phi = \begin{bmatrix} \phi(\mathbf{x}_1)^T \\ \phi(\mathbf{x}_2)^T \\ \vdots \\ \phi(\mathbf{x}_N)^T \end{bmatrix} \in \mathbb{R}^{N \times F} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N$$

- Then, we have

$$\mathbf{w}^* = \left(\Phi^T \Phi\right)^{-1} \Phi^T \mathbf{y}$$

Polynomial feature expansion: Demo

- https://playground.tensorflow.org/#activation=linear&batchSize=10&dataset=gauss&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=&seed=0.49340&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

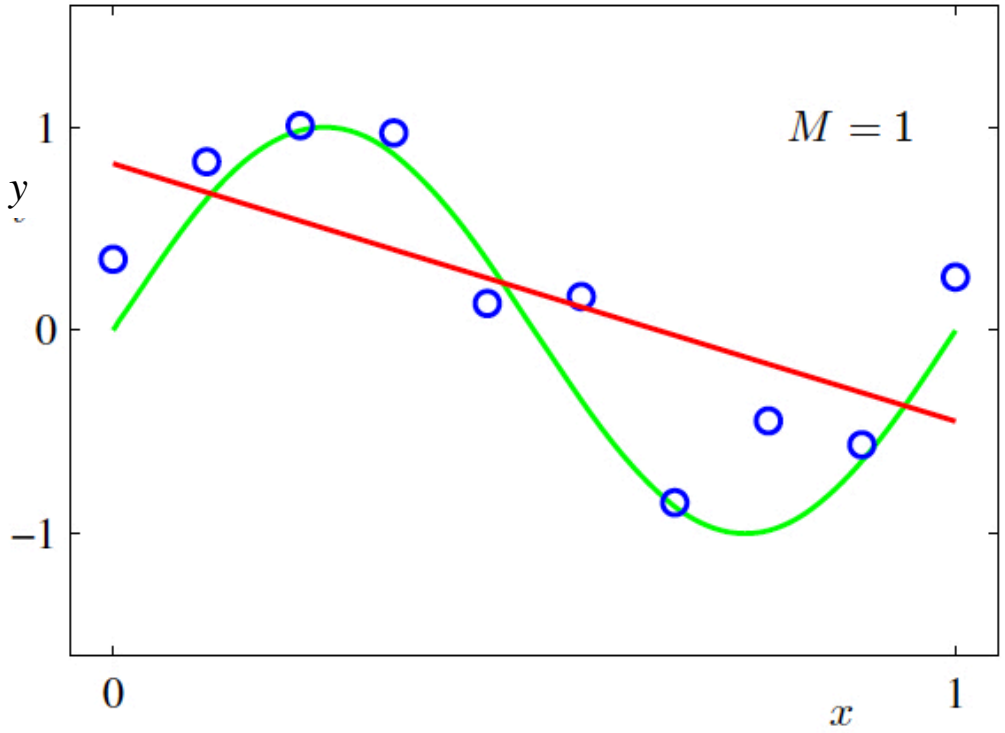
Mapping to a higher-dimensional space

- In all the examples considered before, the mapping $\phi(\cdot)$ increases the dimensionality of the input to the linear model
  - From 1 to 2 in the first toy example
  - From 2 to 3 in the second one
  - From 1 to $M+1$ in the polynomial fitting case
  - From 2 to 5 in the previous demo

- Discussion: Why is higher dimension good?

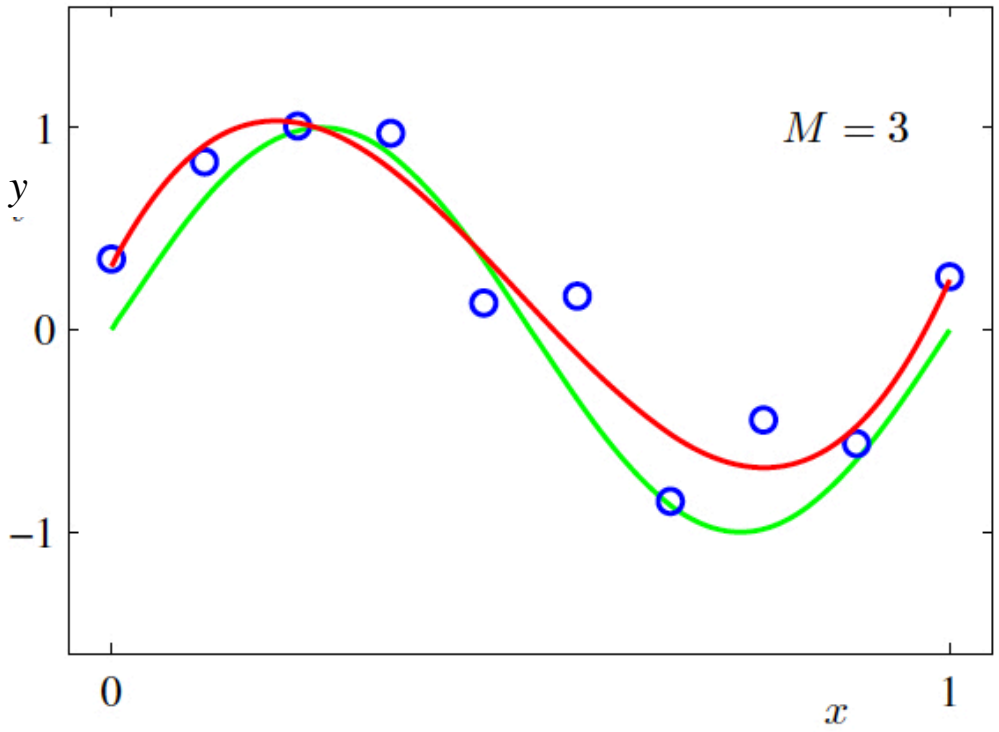
Feature expansion: Properties

- Effective, but requires choosing the degree of the polynomial
  - In Bishop's polynomial curve fitting example, with $M = 1$



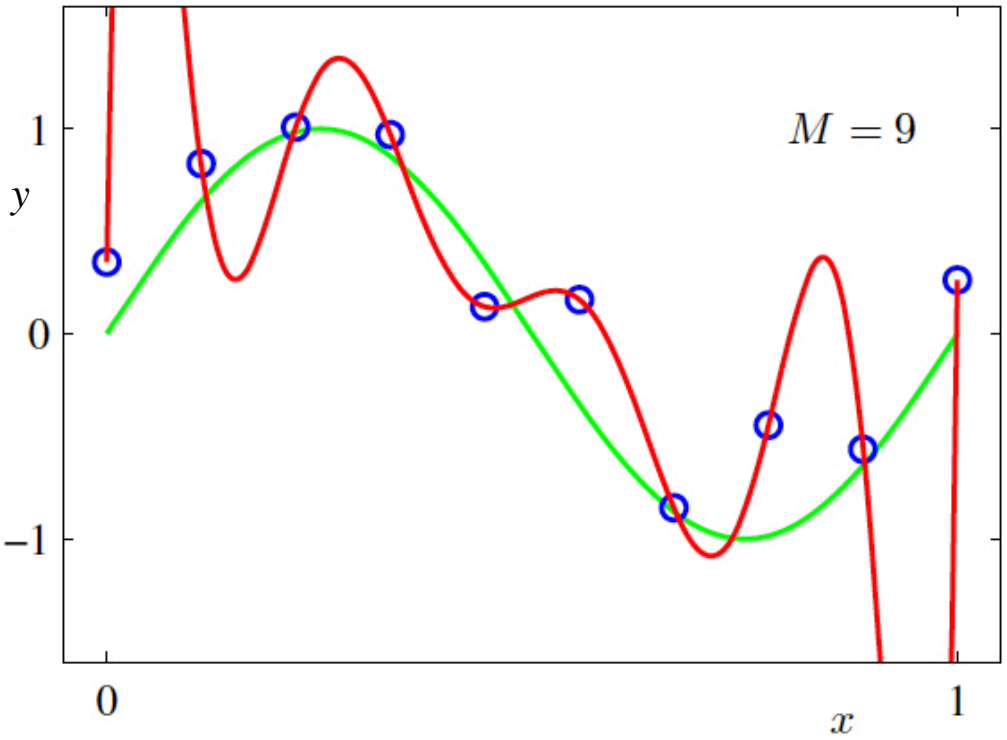
Feature expansion: Properties

- Effective, but requires choosing the degree of the polynomial
  - In Bishop's polynomial curve fitting example, with $M = 3$



Feature expansion: Properties

- Effective, but requires choosing the degree of the polynomial
  - In Bishop's polynomial curve fitting example, with $M = 9$



Feature expansion: Properties

- Why limit ourselves to polynomials?
  - Any function of the input could be used, e.g.,
    - sine and cosine
    - exponential and logarithm
    - …
- In fact, in the demo…
  - https://playground.tensorflow.org/#activation=linear&batchSize=10&dataset=gauss&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=&seed=0.49340&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false
- Then, how do we choose which functions to use?
  - Solution: Kernels: In 2 weeks