

APL 405: Machine Learning in Mechanics

Lecture 3: k -Nearest Neighbour

by

Rajdip Nayek

Assistant Professor,
Applied Mechanics Department,
IIT Delhi

Instructor email: rajdipn@am.iitd.ac.in

Supervised Learning: Recap

Supervised Learning

Learning (training, estimating) a function (or model) f so that it best fits the relationship between

- the input \mathbf{x} , and
- the output y

from observed training data (the individual data points are assumed to be (probabilistically) **independent**)

$$D_{\text{train}} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$$

End goal will be to construct an output prediction $\hat{y}(\mathbf{x}_*)$ for unseen input \mathbf{x}_* so that it is close to

Types of Supervised learning: **Classification** and **Regression**

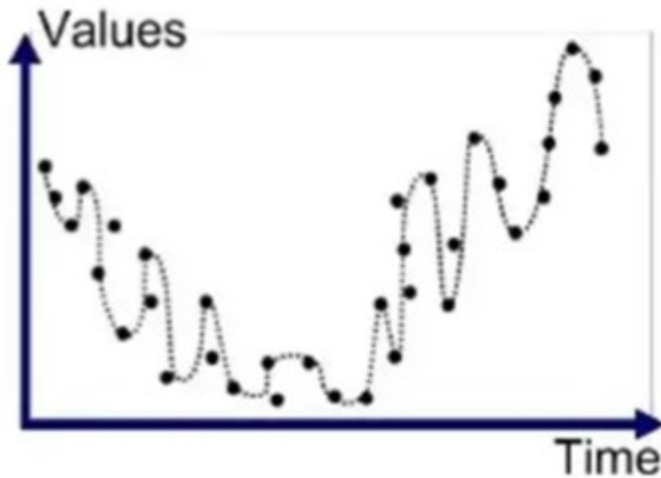
- Output variable y ? \rightarrow **categorical** \rightarrow **Classification**
- Output variable y ? \rightarrow **numerical** \rightarrow **Regression**
- Input variable can be categorical or numerical or mix of both

Supervised Learning: Recap

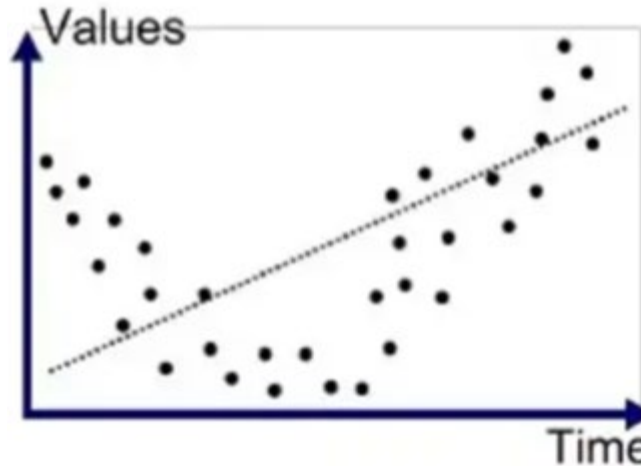
- **Parametric vs Non-parametric models**
- Prediction errors caused due to **bias**, **variance** and **irreducible** errors
- Bias make algorithms easier to understand but are generally less flexible
 - **Low bias**: Suggests less assumptions about the function f
 - **High bias**: Suggests more assumptions about the function f
- Machine learning algorithms that have a high variance are strongly influenced by the specifics of the training data
 - **Low variance**: Suggests small changes to the estimated function f with changes to the training dataset
 - **High variance**: Suggests large changes to the estimated function f with changes to the training dataset

Overfit vs Underfit

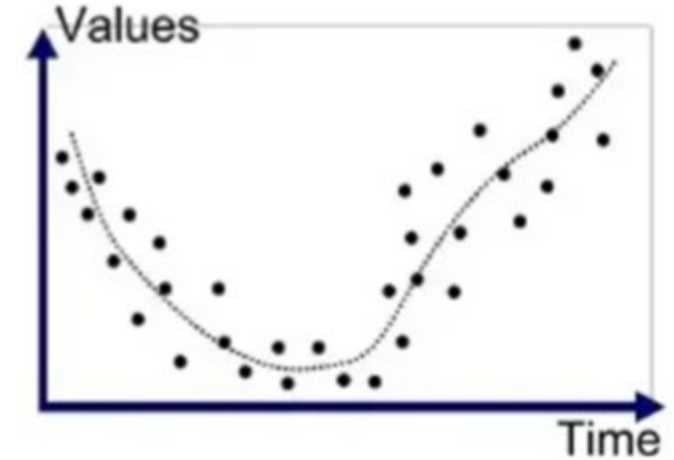
- **Overfitting** refers to the phenomenon when a model fits the training data “too well”
 - It happens when a **model learns the detail and noise in the training data**.
This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model.
 - Models that have **high variance** and **low bias** leads to overfitting
 - **Does not generalize to new unseen data well**



Overfitting



Underfitting



Balanced fit

- **Underfitting** refers to the phenomenon when a model is unable to fit to the training data
 - It happens when a model is “too rigid”
 - Models that have **high bias** and **low variance** leads to underfitting
 - **Does not generalize to new unseen data well**

Introduction to k -Nearest Neighbours (k -NN)

- We will start with the relatively simple *k -nearest neighbours* (k -NN) method.
- ✓ Can be used for *both* regression and classification
- Most ML algorithms are based on the intuition that *if the unseen data point \mathbf{x}_* is close to training data point \mathbf{x}_i , then the prediction $\hat{y}(\mathbf{x}_*)$ should be close to y_i .*
- A simple way to implement this idea is to find the “nearest” training data point
 - Compute the Euclidean† distance between the unseen input and all training inputs.

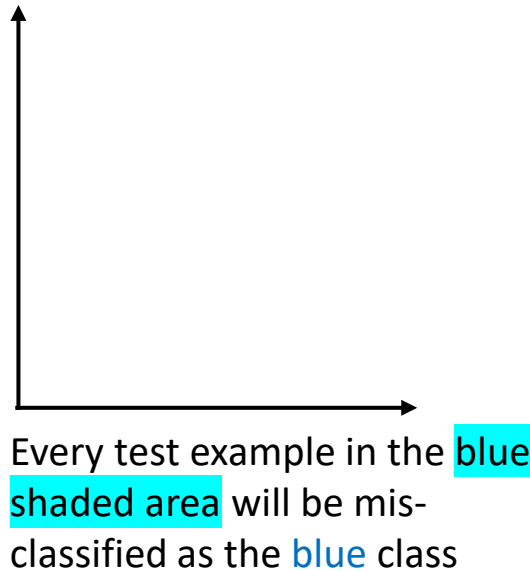
The i th Euclidean distance:
$$\|\mathbf{x}_i - \mathbf{x}_*\|_2 = \sqrt{(x_{i,1} - x_{*,1})^2 + (x_{i,2} - x_{*,2})^2 + \cdots + (x_{i,p} - x_{*,p})^2}$$

- Find the data point \mathbf{x}_j with the *shortest distance* to \mathbf{x}_* , and use its output as the prediction, $\hat{y}(\mathbf{x}_*) = y_j$
- This is the 1-nearest neighbour algorithm

† There are many other distances: Manhattan, Mahalanobis, cosine similarity, etc. Use Manhattan distance if inputs variables are not similar in type (such as age, gender, height, etc.)

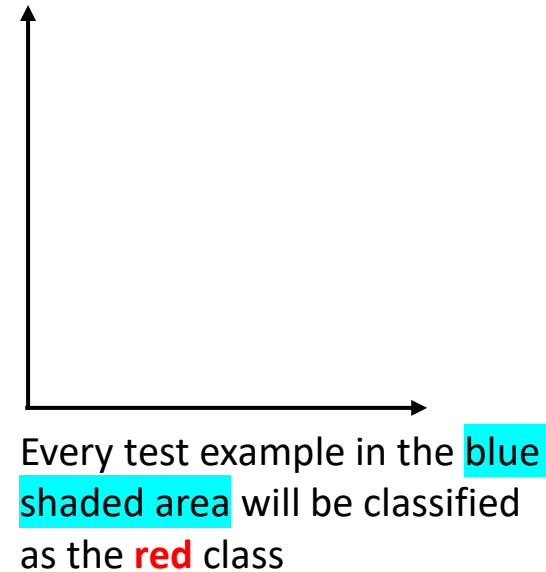
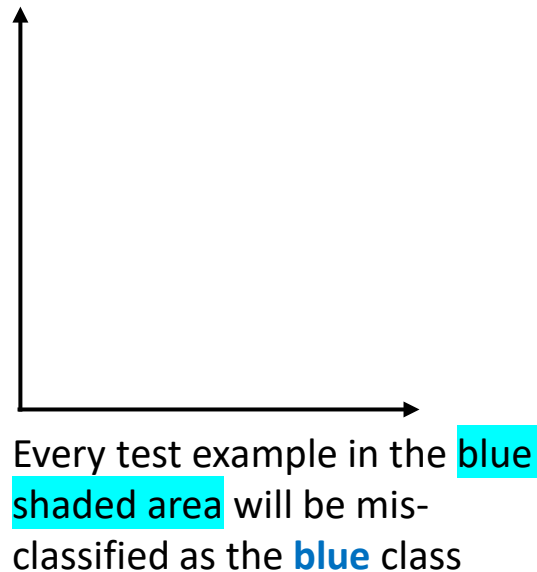
Introduction to k -Nearest Neighbours (k -NN)

- In practice, however, we can rarely say *for certain* what the output value y will be!
- Mathematically, we handle this by describing y as a **random variable**. That is, we consider the data as *noisy*, meaning that it is affected by **random errors** referred to as **noise**.
- **Shortcoming:** 1-nearest neighbour algorithm is sensitive to **noise** in data and **mis-labelled** data



Introduction to k -Nearest Neighbours (k -NN)

- In practice, however, we can rarely say *for certain* what the output value y will be!
- Mathematically, we handle this by describing y as a **random variable**. That is, we consider the data as *noisy*, meaning that it is affected by **random errors referred to as noise**.
- **Shortcoming:** 1-nearest neighbour algorithm is sensitive to **noise** in data and **mis-labelled** data



- **How to improve:** Use **k -nearest neighbours** to obtain a majority vote (or take an average)

Introduction to k -Nearest Neighbours (k -NN)

k -NN algorithm

Data: Training data $(\mathbf{x}_i, y_i)_{i=1}^N$ and unseen (test) input \mathbf{x}_*

Result: Predicted test output $\hat{y}(\mathbf{x}_*)$

1. Compute the distances $\|\mathbf{x}_i - \mathbf{x}_*\|_2$ for all training data points $i = 1, 2, \dots, N$
2. Find k examples (\mathbf{x}_i, y_i) closest to the test instance \mathbf{x}_*
3. Compute the prediction $\hat{y}(\mathbf{x}_*)$

$$\hat{y}(\mathbf{x}_*) = \begin{cases} \text{Mean (or median) of } k \text{ closest examples} & \text{(Regression)} \\ \text{Majority vote (mode) of } k \text{ closest examples} & \text{(Classification)} \end{cases}$$

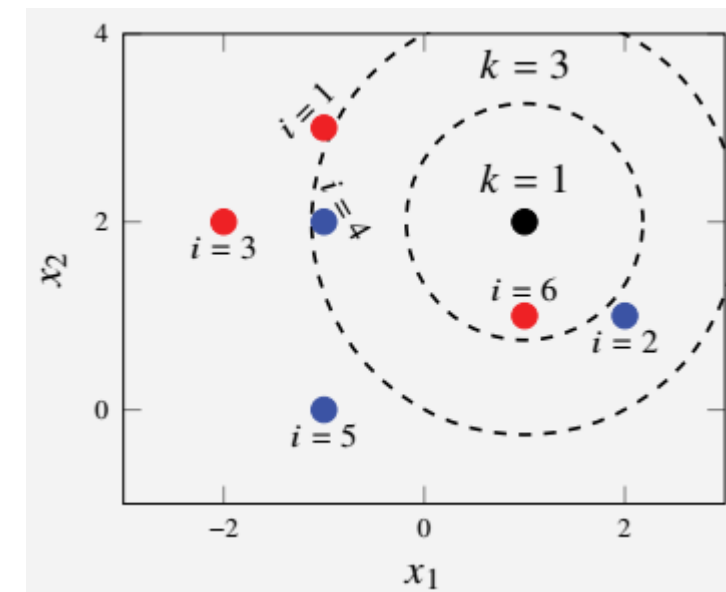
- Is k -NN a **parametric** or **non-parametric** algorithm?
- It makes no assumptions about the functional form and has no fixed set of parameters. Uses the entire training data when making predictions

Example of k NN for binary classification

i	$x_{i,1}$	$x_{i,2}$	y_i
1	-1	3	Red
2	2	1	Blue
3	-2	2	Red
4	-1	2	Blue
5	-1	0	Blue
6	1	1	Red

i	$\ \mathbf{x}_i - \mathbf{x}_*\ _2$	y_i
6	$\sqrt{1}$	Red
2	$\sqrt{2}$	Blue
4	$\sqrt{4}$	Blue
1	$\sqrt{5}$	Red
5	$\sqrt{8}$	Blue
3	$\sqrt{9}$	Red

- Predict the output for $\mathbf{x}_* = [1 \ 2]^T$
- Consider two different k NN classifiers
 - one using $k = 1$, and (result is red)
 - another using $k = 3$ (result is blue)

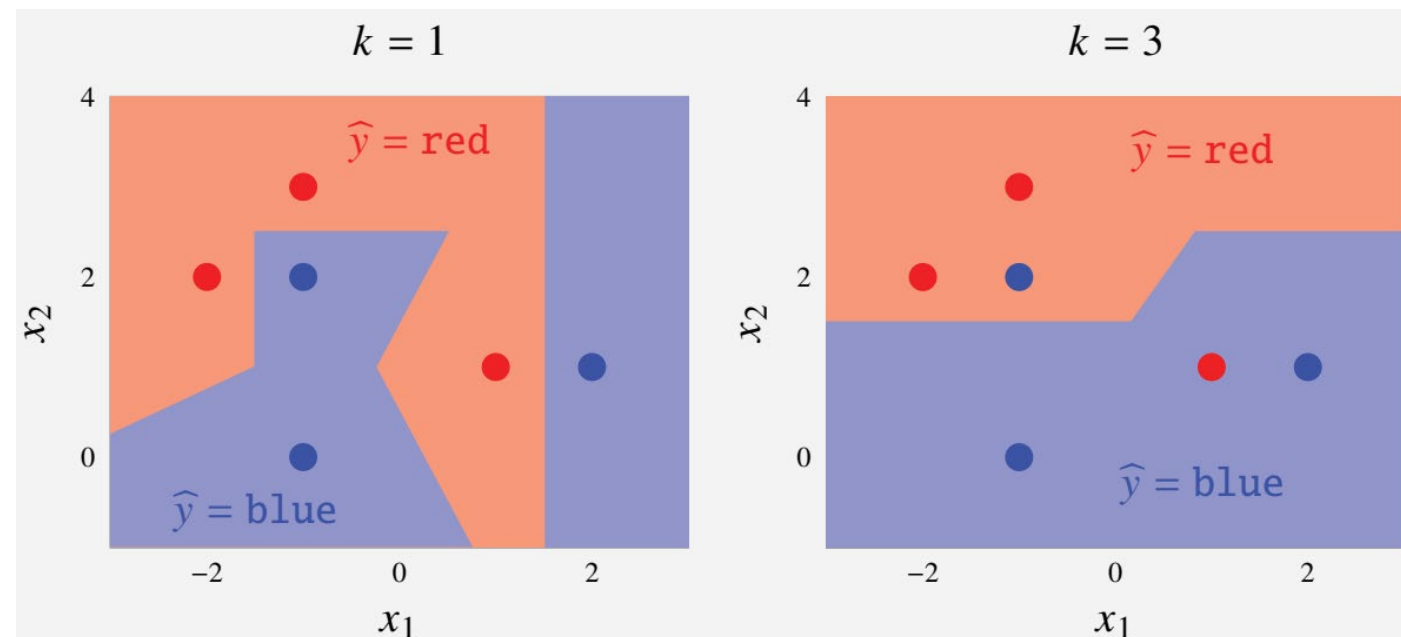


Decision boundary of a classifier

i	$x_{1,i}$	$x_{2,i}$	y_i
1	-1	3	Red
2	2	1	Blue
3	-2	2	Red
4	-1	2	Blue
5	-1	0	Blue
6	1	1	Red

- **Decision boundaries** are the points in input space where the class prediction changes, that is, the borders between different classes
- They can help to understand a classifier and given a concise summary of a classifier

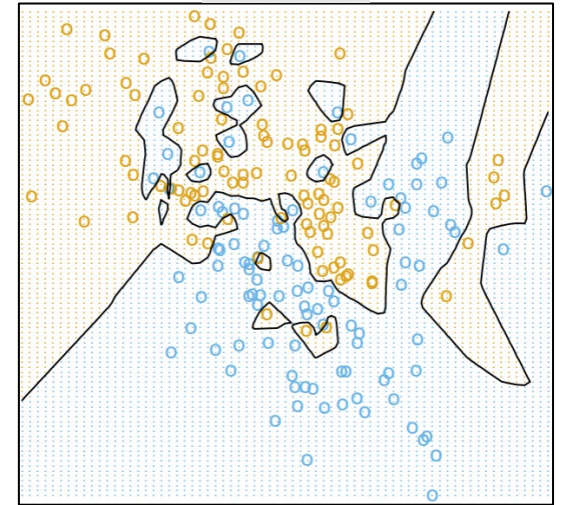
- Predict the output for $\mathbf{x}_* = [1 \ 2]^T$
- Consider two different k NN classifiers
 - one using $k = 1$, and
 - another using $k = 3$



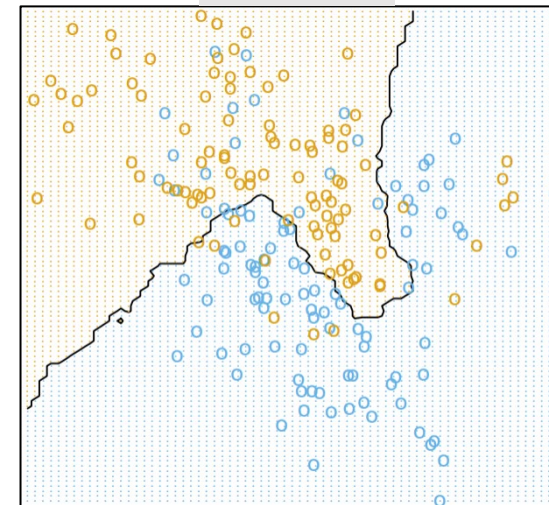
How to choose k ?

- The number of neighbours k is chosen by the user
- Since it is not learned, it is not a parameter, and we refer to it as the **hyperparameter**
- The choice of hyperparameter k has a big impact on the predictions made by k -NN
 - Small k
 - Good at capturing fine-grained patterns
 - May **overfit**, i.e. be sensitive to random errors in the training data
 - Large k
 - Makes stable predictions by averaging over lots of samples
 - May **underfit**, i.e. fail to capture important patterns
 - Balancing k (trade-off between flexibility and rigidity)
 - Optimal choice of k depends on the number of data points N
 - Rule of thumb: choose $k < \sqrt{N}$
 - We can choose k using **cross-validation**

$k = 1$

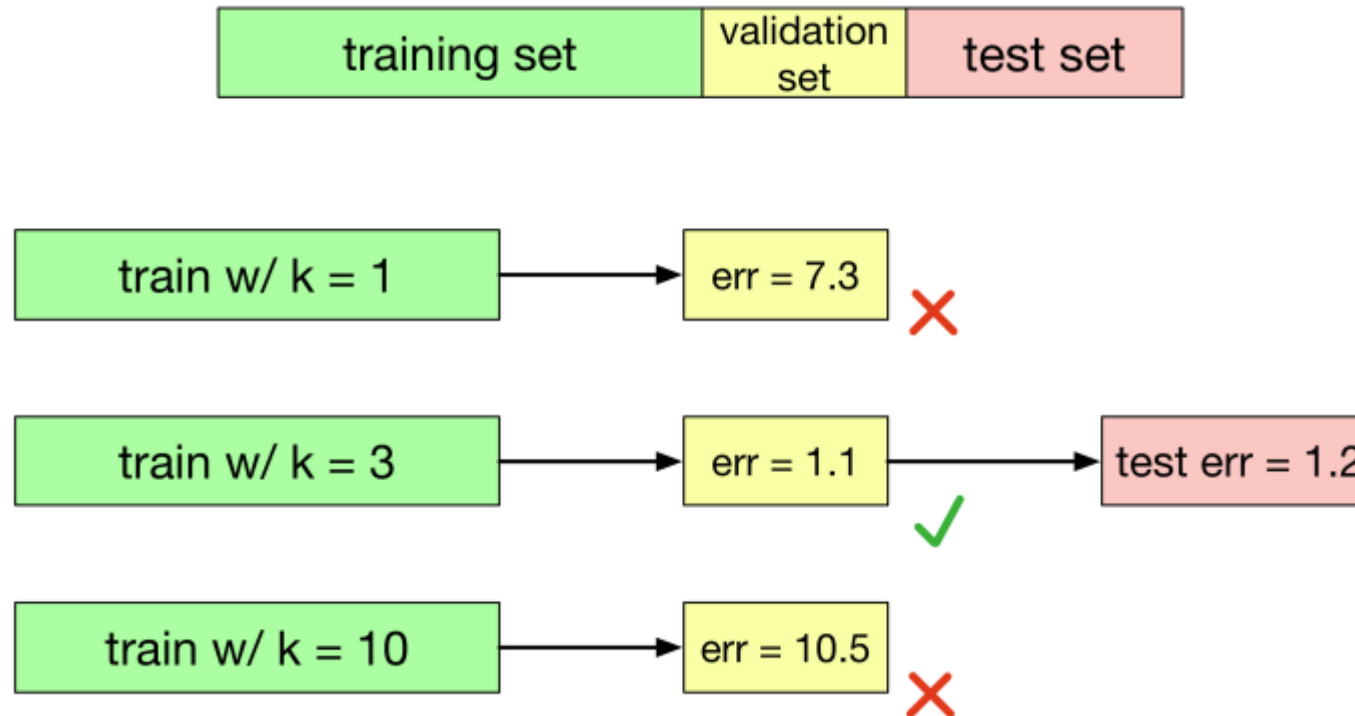


$k = 15$



Validation and Test sets

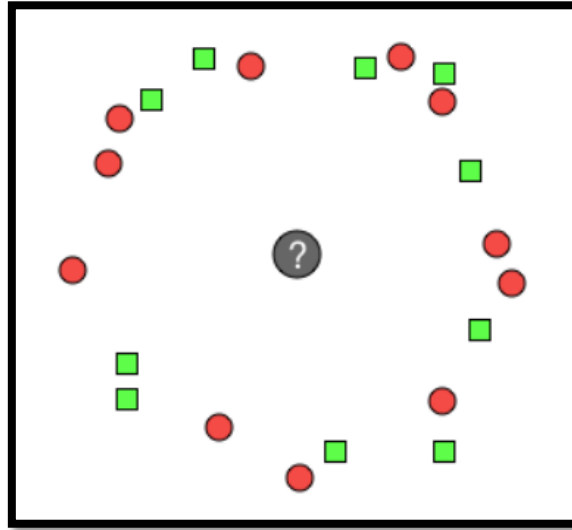
- We can tune the hyperparameters using a **validation set**:



- The **test set** is used only at the very end, to measure the generalization performance of the algorithm.

Pitfalls of k NN: Curse of dimensionality

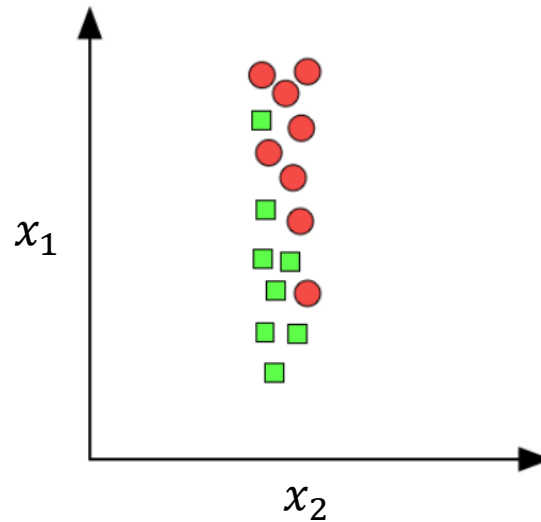
- k NN works well with a small dimension of inputs (e.g. 2-3), but struggles when the input dimension is high
- In high dimensions, “most” points are far apart and are approximately at the same distance
 - Hence, our intuition that works for distances in 2- and 3- dimensional spaces breaks down in higher dimensions



- We can show this by applying the rules of expectation and covariance of random variables (HW)

Pitfalls of k NN: Normalization

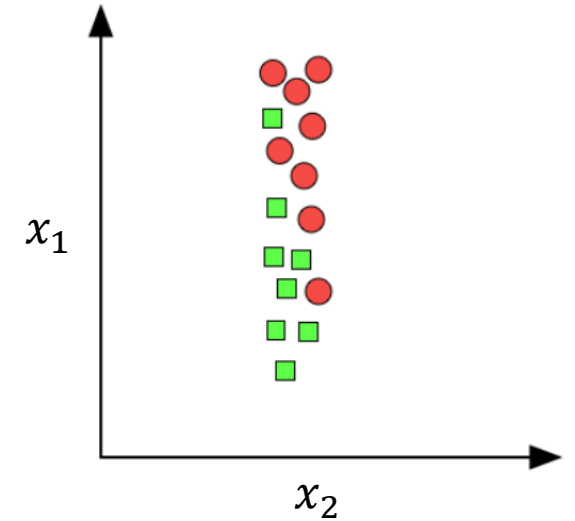
- k NN can be quite sensitive to the range of the input features
- Example, $\mathbf{x} = [x_1 \ x_2]^T$, where x_1 is in the range $[100, 1000]$ and the values of x_2 is in the range $[0, 1]$ (or vice-versa)



- The Euclidean distance between a test point \mathbf{x}_* and a training data point \mathbf{x}_i is $\sqrt{(x_{i,1} - x_{*,1})^2 + (x_{i,2} - x_{*,2})^2}$
- The Euclidean distance is dominated by the first term $(x_{i,1} - x_{*,1})^2$ simply due to the larger magnitude of x_1
- Thus, the variable x_1 gets considered much more important than x_2 by k NN

Pitfalls of k NN: Normalization

- k NN can be sensitive to the ranges of the input features



- **Simple fix:** **Normalize** each dimension to be in the range $[0, 1]$

- $\bar{x}_{i,j} = \frac{x_{i,j} - \min_i(x_{i,j})}{\max_i(x_{i,j}) - \min_i(x_{i,j})}$ for all $i = 1, 2, \dots, N$ and $j = 1, 2, \dots, p$

- **Simple fix:** Standardize each dimension using mean μ_j and standard deviation σ_j of data

- $\bar{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sigma_j}$ for all $i = 1, 2, \dots, N$ and $j = 1, 2, \dots, p$

Pitfalls of k NN: Computationally costly

- Computational cost for **training time**: 0
- Computational cost at **test time**, per test data point
 - Calculate p -dimensional Euclidean distances with N data points: $\mathcal{O}(Np)$
 - Sort the distances: $\mathcal{O}(N \log N)$
- This must be done for each test data point, which is very expensive by the standards of a learning algorithm!
- Need to store the entire dataset in memory!
- Gives decent accuracy when there is lots of data



MNIST digit classification

- Handwritten digits
- 28x28 pixel images: $p = 784$
- 60,000 training samples
- 10,000 test samples

	Test Error Rate (%)
Linear classifier (1-layer NN)	12.0
K-nearest-neighbors, Euclidean	5.0
K-nearest-neighbors, Euclidean, deskewed	2.4
K-NN, Tangent Distance, 16x16	1.1
K-NN, shape context matching	0.67
1000 RBF + linear classifier	3.6
SVM deg 4 polynomial	1.1
2-layer NN, 300 hidden units	4.7
2-layer NN, 300 HU, [deskewing]	1.6
LeNet-5, [distortions]	0.8
Boosted LeNet-4, [distortions]	0.7

Summary

- k -Nearest Neighbors algorithm is a non-parametric algorithm that can be used for both classification and regression
- k NN stores the entire training dataset in memory which it uses as its representation
 - k NN does not learn any model
- k NN makes predictions just-in-time by calculating the similarity between a test input and each training sample
 - There are many distance measures to choose from to match the structure of your input data
- It is a good idea to rescale your data, such as using normalization, when using k NN