Machine Learning CS342

Lecture 4: Instance-based Learning: The k-NN algorithm

Dr. Theo Damoulas
T.Damoulas@warwick.ac.uk

Office hours: Mon & Fri 10-11am @ CS 307

Last week summary:

Linear regression (OLS)
Ch1. R&G book

Machine Learning

- **Supervised Learning**
  - Regression ← Linear regression (OLS), Ch1. R&G book
  - Classification
  - Ranking
  - Structured Prediction

$$\mathcal{D} = \{\mathbf{X}, \mathbf{t}\}, \quad e.g. \quad \mathbf{X} \in \Re^{N \times D}, \mathbf{t} \in \Re^{N}$$

- **Unsupervised Learning**
  - Clustering
  - Dimensionality Reduction
  - Manifold Learning

$$\mathcal{D} = \{\mathbf{X}\}, \quad \mathbf{X} \in \Re^{N \times D}$$

- **Reinforcement Learning**
  - Markov Decision Process (e.g. Robotics)
  - Multi-Agent Systems
  - TD/Q-Learning (State-Action-Reward)

$$\mathcal{D} = \{\mathbf{S}, \mathbf{A}, \mathbf{R}, \pi^*, \mathbf{V}\}$$

Last week summary: Inputs X - Outputs t

$$\mathcal{D} = \{\mathbf{X}, \mathbf{t}\}$$

Further Training & Validation splits on this

Attributes, Dimensions, Features

D

Observations

Samples N

Instances

Student reg. no.	ML grade	P. Skills grade	final degree
1	92%	84%	78%
2	54%	100%	62%
3	58%	50%	52%
4	85%	96%	72%
5	67%	98%	68%
6	75%	86%	72%
7	52%	100%	61%
8	82%	90%	85%

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \\ x_{51} & x_{52} \\ x_{61} & x_{62} \\ x_{71} & x_{72} \\ x_{81} & x_{82} \end{bmatrix} \qquad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \end{bmatrix}$$

$$\mathbf{X} \in \Re^{N \times D} \qquad \mathbf{t} \in \Re^{N}$$

Last week summary: Linear model

$$\hat{t}_n = \hat{w}_0 + \hat{w}_1 x_{n1} + \hat{w}_2 x_{n2} = \mathbf{x}_n \hat{\mathbf{w}} \qquad \hat{\mathbf{t}} = \mathbf{X}\hat{\mathbf{w}}$$

Squared Error Loss $\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N}(t_n - f(x_n; w_0, w_1))^2$

Find the parameters that minimise the Loss

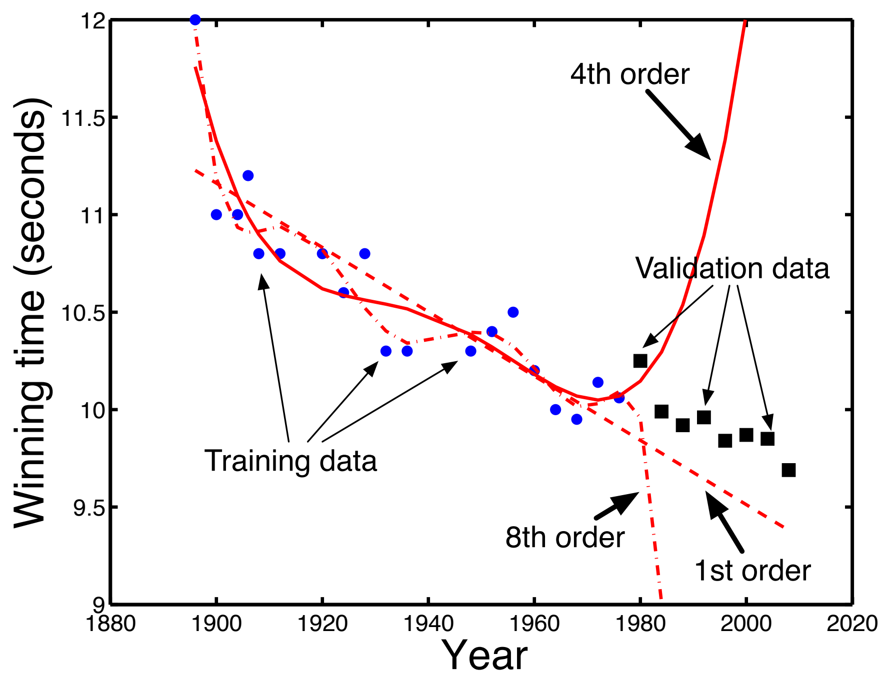
$$\widehat{w_0}, \widehat{w_1} \leftarrow \underset{w_0, w_1}{\operatorname{argmin}} \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_n(t_n, f(x_n; w_0, w_1))$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} \to 0 \qquad \text{if} \quad \frac{\partial^2 \mathcal{L}}{\partial^2 \mathbf{w}} > 0 \quad \text{we are at a minima}$$

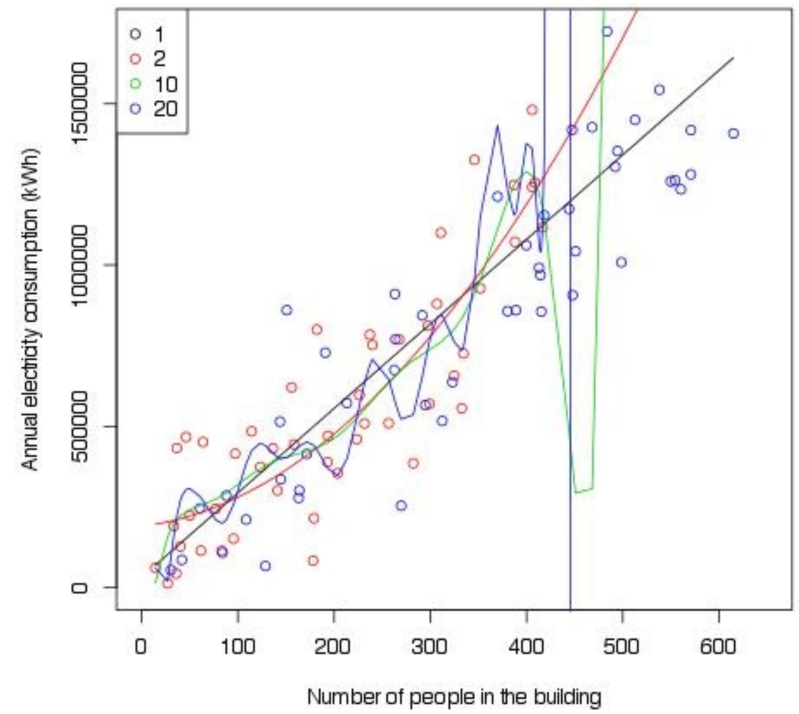
$$\widehat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{t}$$ where $$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1D} \\ 1 & x_{21} & \dots & x_{2D} \\ 1 & x_{N1} & \dots & x_{ND} \end{bmatrix}$$

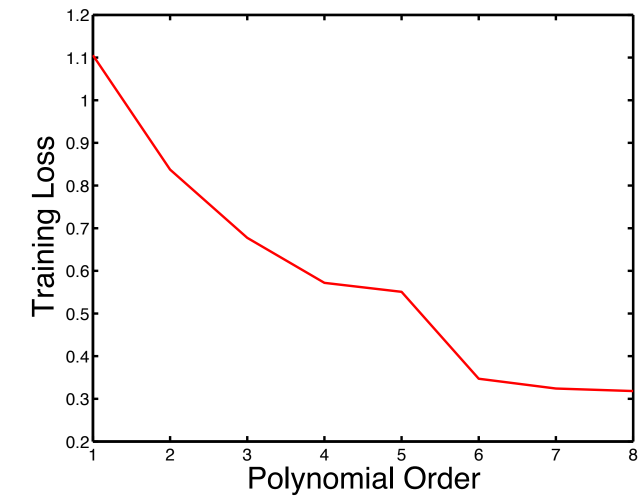
Last week summary: Overfitting & Cross-validation

Generalisation & Validation data

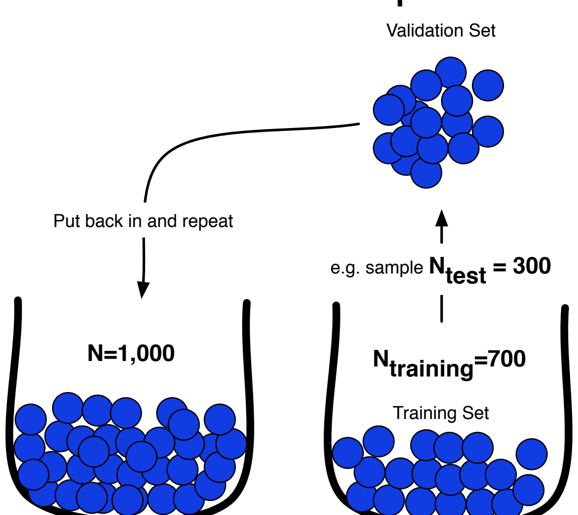


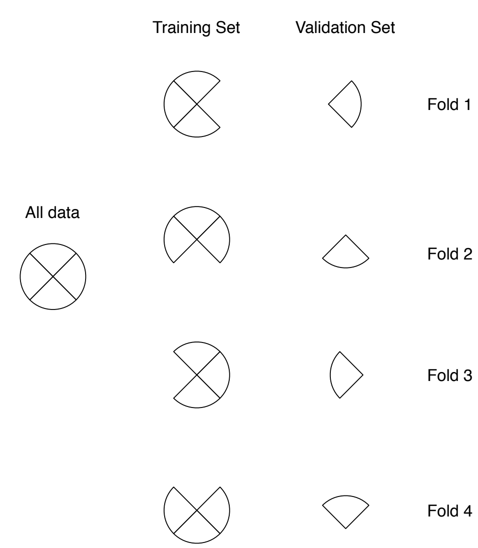
Overfitting and Curse of Dimensionality



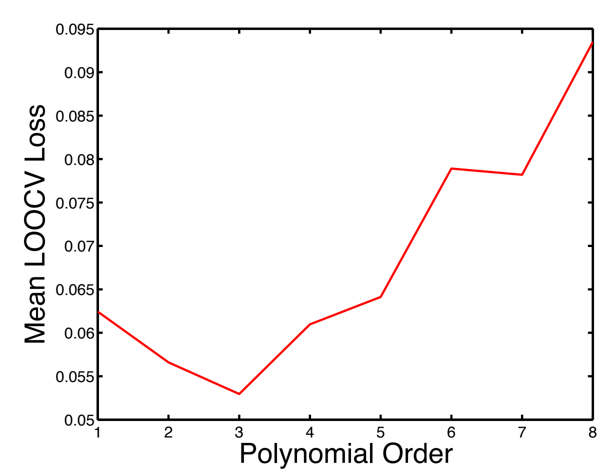


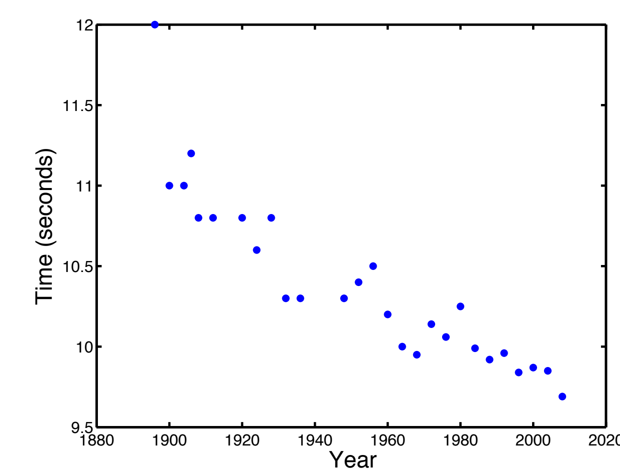
Bootstrap & Cross-validation





Model Selection and IID assumption





This week

Non-linear regression & Classification (k-NN, DTs)

Machine Learning

Supervised Learning
- Regression
- Classification
- Ranking
- Structured Prediction

$$\mathcal{D} = \{\mathbf{X}, \mathbf{t}\}, \quad e.g. \quad \mathbf{X} \in \Re^{N \times D}, \mathbf{t} \in \Re^{N}$$

Unsupervised Learning
- Clustering
- Dimensionality Reduction
- Manifold Learning

$$\mathcal{D} = \{\mathbf{X}\}, \quad \mathbf{X} \in \Re^{N \times D}$$

Reinforcement Learning
- Markov Decision Process (e.g. Robotics)
- Multi-Agent Systems
- TD/Q-Learning (State-Action-Reward)

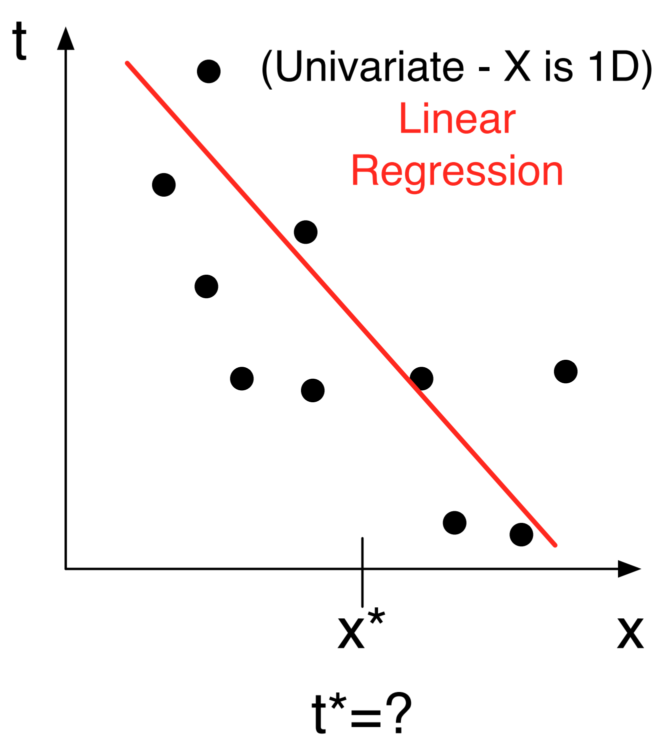
$$\mathcal{D} = \{\mathbf{S}, \mathbf{A}, \mathbf{R}, \pi^*, \mathbf{V}\}$$

Instance-based Learning (Regression & Classification)

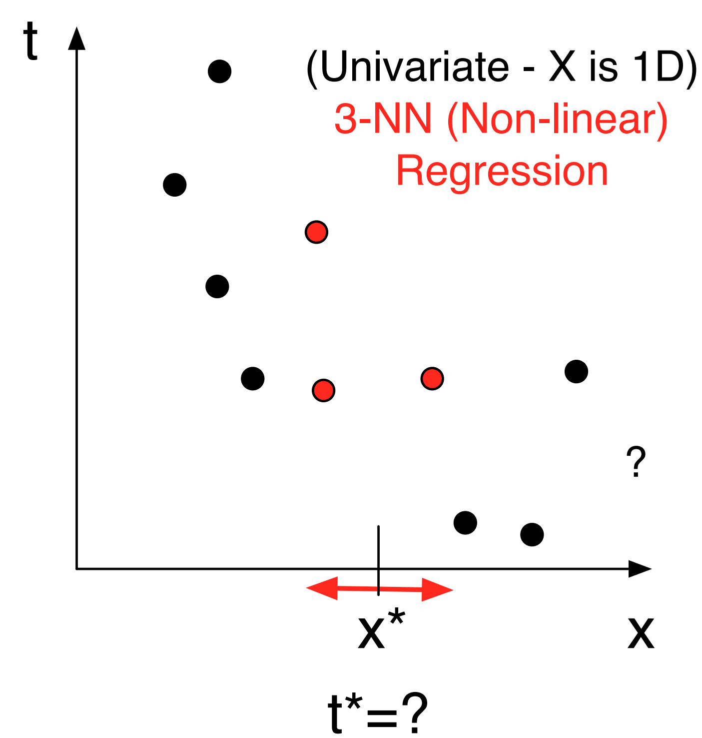
T. Mitchell book, Ch.8

Instance-based learners are all the machine learning algorithms that ***do not*** *construct an explicit description of the target function (like OLS) but store the training examples (instances) and use them,* ***and a notion of distance from them****, to generalise to unseen data.*

Regression: t is continuous



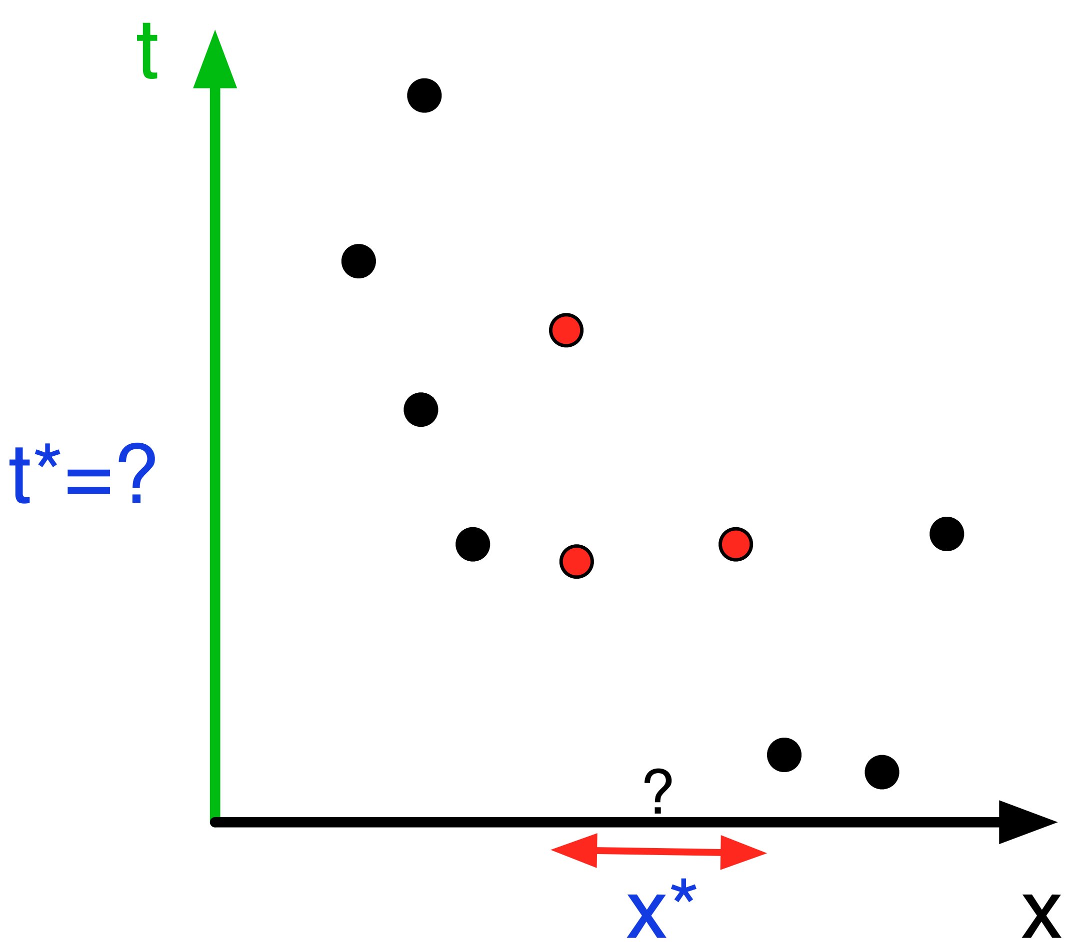
Training: Learn the best line/plane
Predict: Use the line/plane



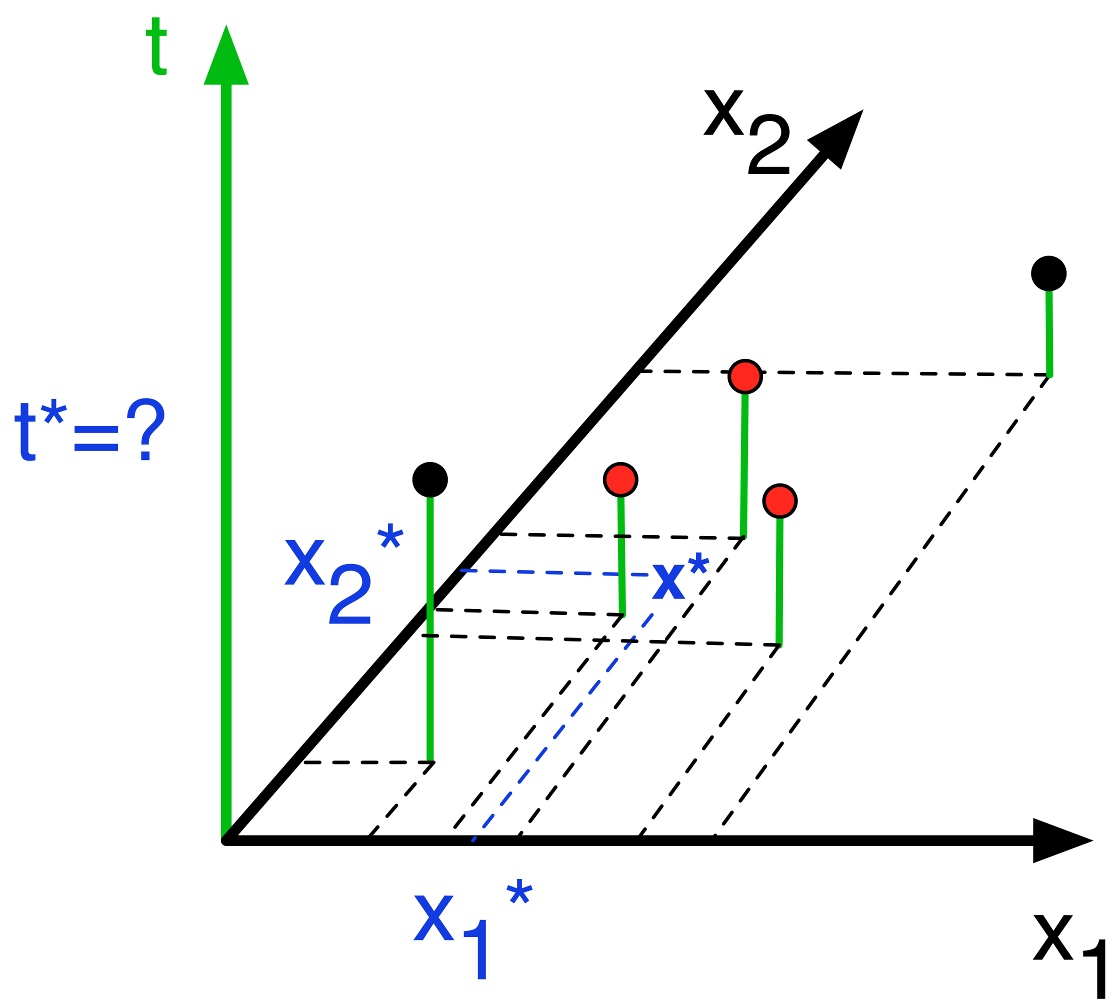
Training: Store the training data
Predict: Use the "closest" observations

What if we have 2 or more attributes (dimensions) for X?

Univariate (x is 1D)



Multivariate (x is 2D)



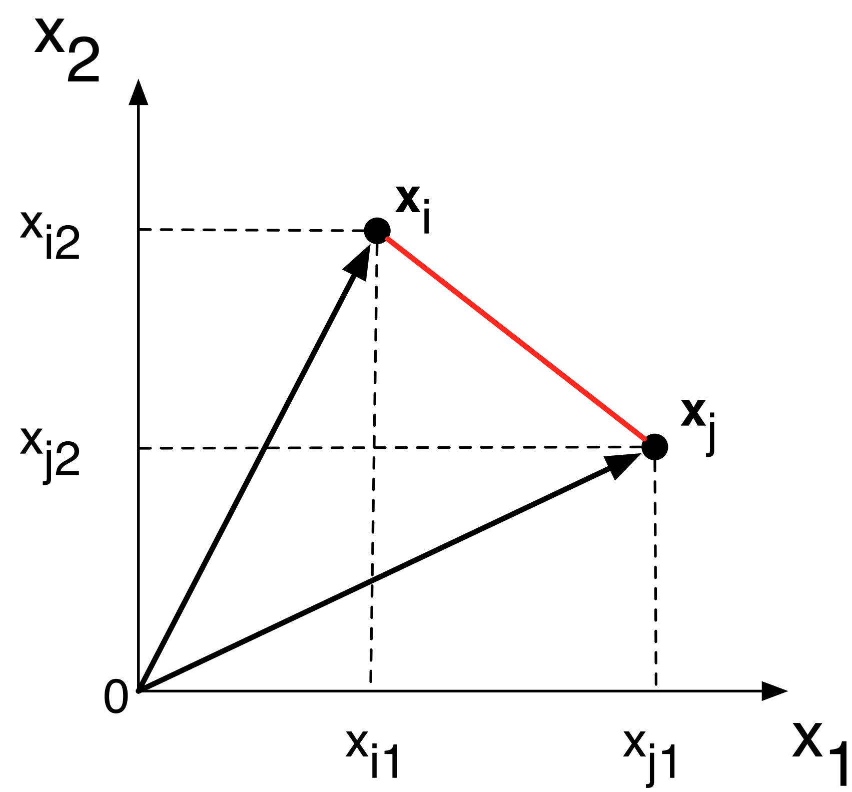
Need a distance metric

Euclidean Distance

Euclidean (L2) Distance
between x_i and x_j?

Input space
Univariate (x is 1-D scalar)

Input space
Multivariate (**x** is 2-D vector)



$$\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{iD}] \quad \in \quad \Re^D$$

$$\mathbf{x}_j = [x_{j1}, x_{j2}, \ldots, x_{jD}] \quad \in \quad \Re^D$$

For any D

$$\overset{\text{Euclidean}}{d}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{d=1}^{D} (x_{id} - x_{jd})^2}$$

Remember Lp norm?

Other distances? Lp norms and Distances

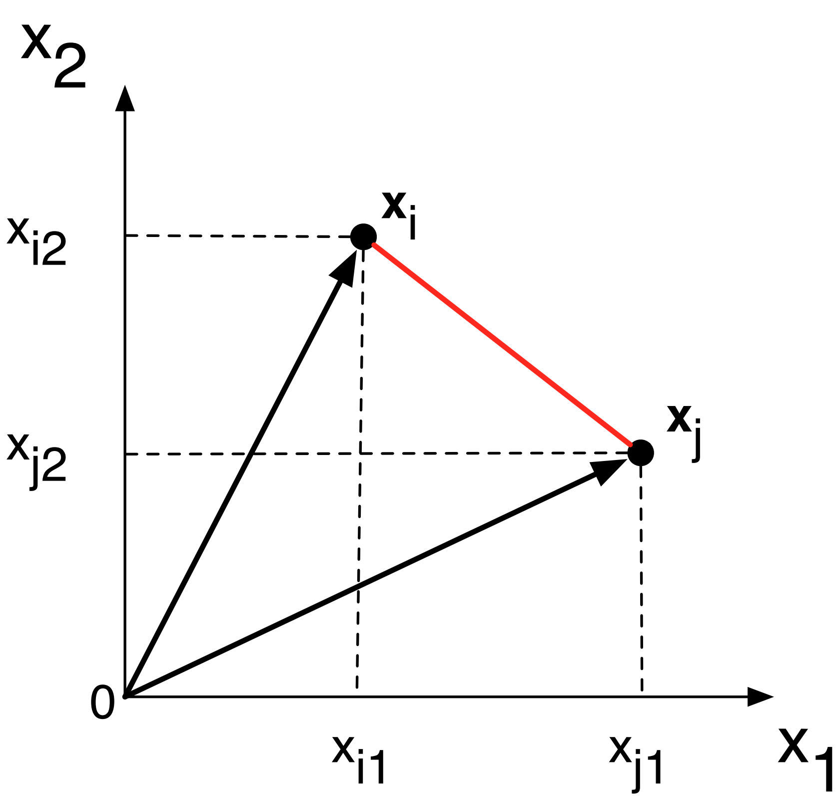
for a D-dimensional vector $\mathbf{x}_n$ the Lp norm is:

$$\mathrm{L}_p = \left(\sum_{d=1}^{D} |x_{nd}|^p \right)^{\frac{1}{p}}$$

Every norm (e.g. L1 for p=1, L2 for p=2) induces a *metric distance*

for p=2, Euclidean (L2) norm:

$$\mathrm{L}_2 = \left(\sum_{d=1}^{D} |x_{nd}|^2 \right)^{\frac{1}{2}} = \sqrt{\sum_{d=1}^{D} |x_{nd}|^2}$$



So L2 distance between $\mathbf{x}_i$ and $\mathbf{x}_j$:

$$\mathrm{L}_2(\mathbf{x}_i, \mathbf{x}_j) = \overset{\text{Euclidean}}{d(\mathbf{x}_i, \mathbf{x}_j)} = \sqrt{\sum_{d=1}^{D} (x_{id} - x_{jd})^2}$$

Manhattan distance (L1)

Can you use the Lp norm definition to write the L1 (p=1) distance between **x**$_i$ and **x**$_j$?

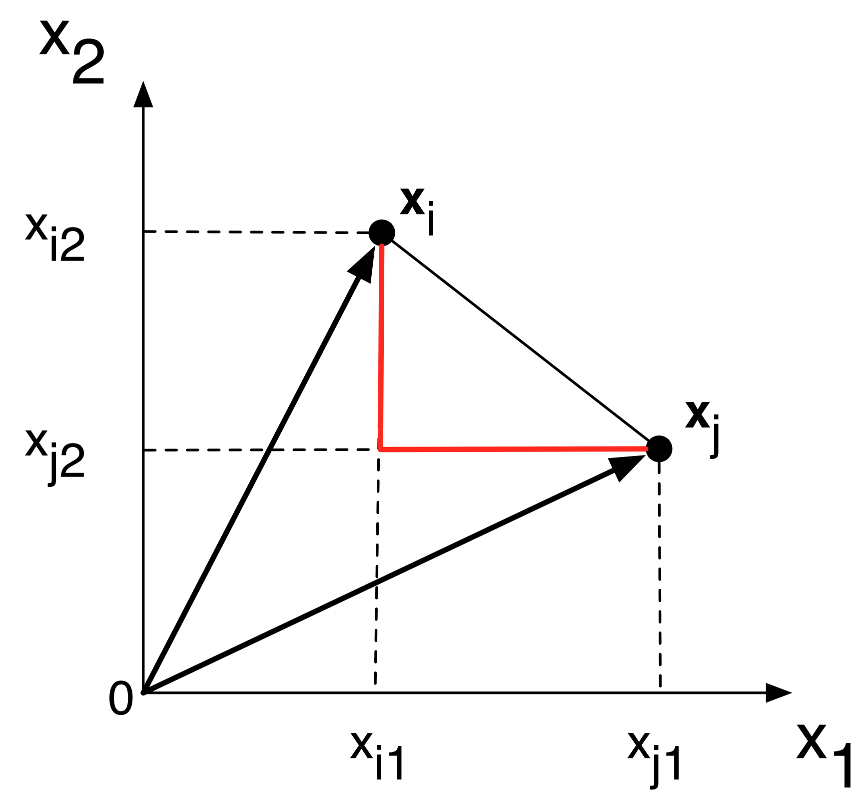
$$\mathrm{L}_p = \left(\sum_{d=1}^{D} |x_{nd}|^p \right)^{\frac{1}{p}}$$

$$d^{\text{Manhattan}}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{d=1}^{D} |x_{id} - x_{jd}|$$

Manhattan (L1) Distance between x$_i$ and x$_j$?

Input space
Univariate (x is 1-D scalar)

Input space
Multivariate (**x** is 2-D vector)



Distance for categorical data?

$\mathbf{x}_i$	1	0	1	1
$\mathbf{x}_j$	0	1	1	1

Hamming distance

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{d=1}^{D} \begin{cases} 1 & \text{if} \quad x_{id} \neq x_{jd} \\ 0 & \text{if} \quad x_{id} = x_{jd} \end{cases}$$

basic k-NN algorithm for regression

a.k.a. Lazy learning: No real training step..

Training:

- For each training example (input-output pair $\mathbf{x}_n, t_n$), add the example to the list *training_examples*

Regression:

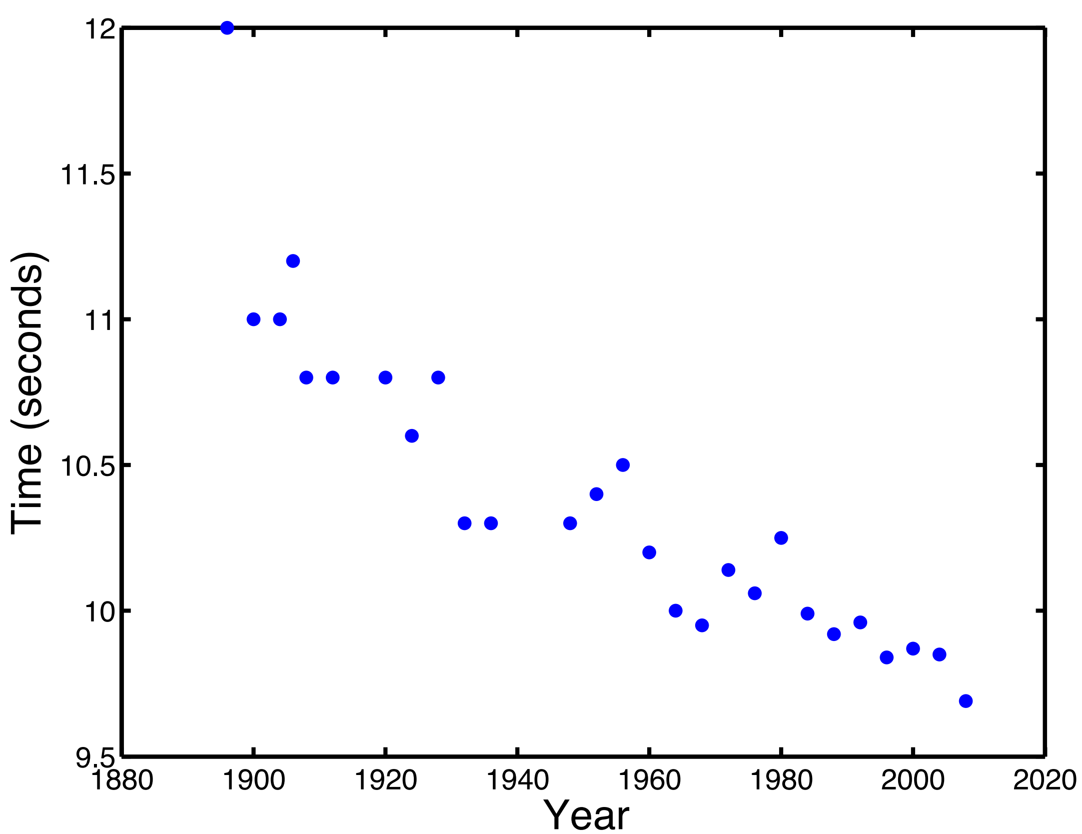
- Choose k, the number of neighbours we want
- Choose the distance function (e.g. Euclidean distance)
- Given a query instance $\mathbf{x}^*$ to predict its output t^*
  - Find $\mathbf{x}_1 \ldots \mathbf{x}_k$ the k instances that are **nearest** to $\mathbf{x}^*$ using the selected distance
  - Return prediction: $t^* \leftarrow \frac{\sum_{k=1}^{K} t_k}{k}$

Regression vs Classification

Regression: targets **t** are continuous values

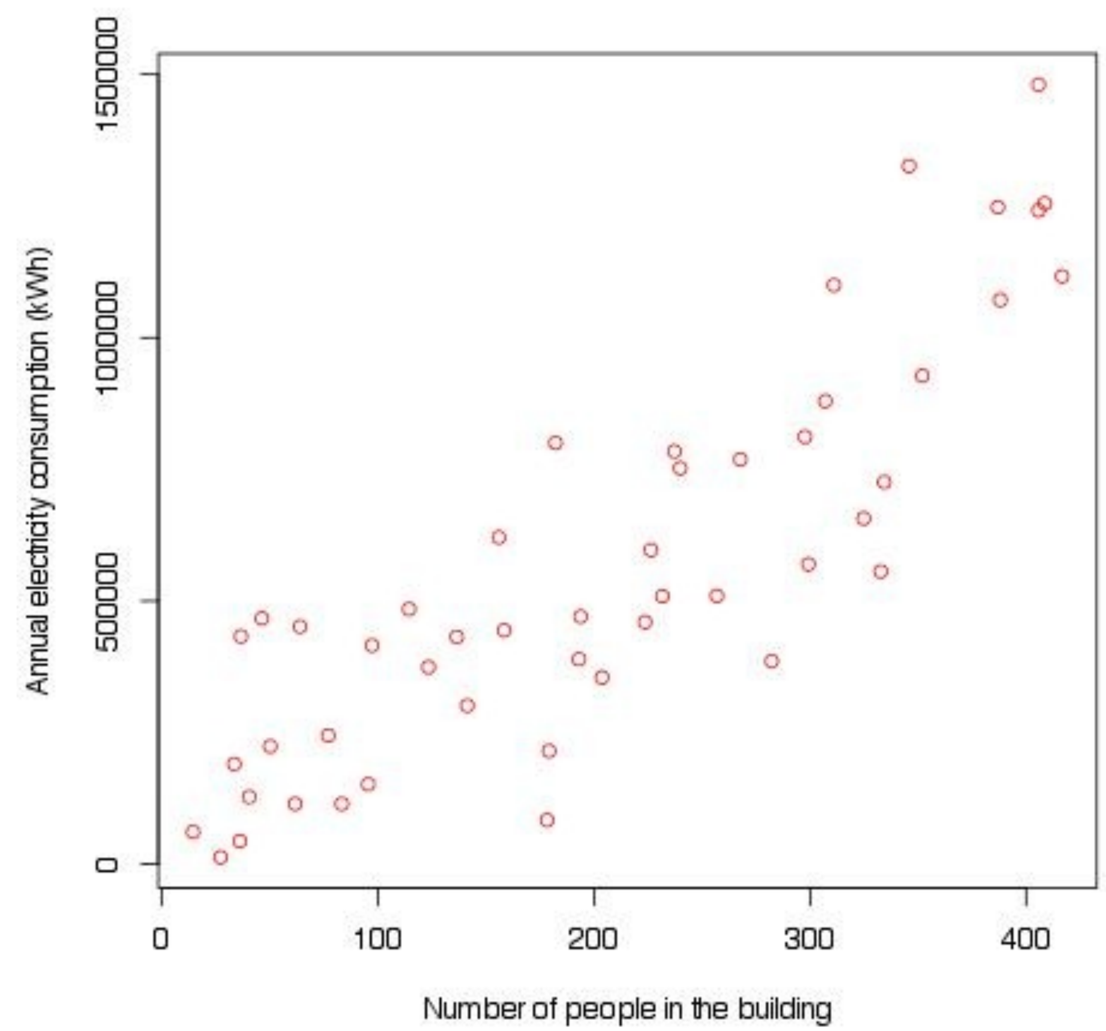
We can visualise the target as an additional dimension

t



x

t



x

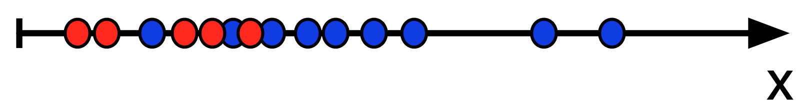
In both cases we have one attribute so x is 1D

Regression vs Classification

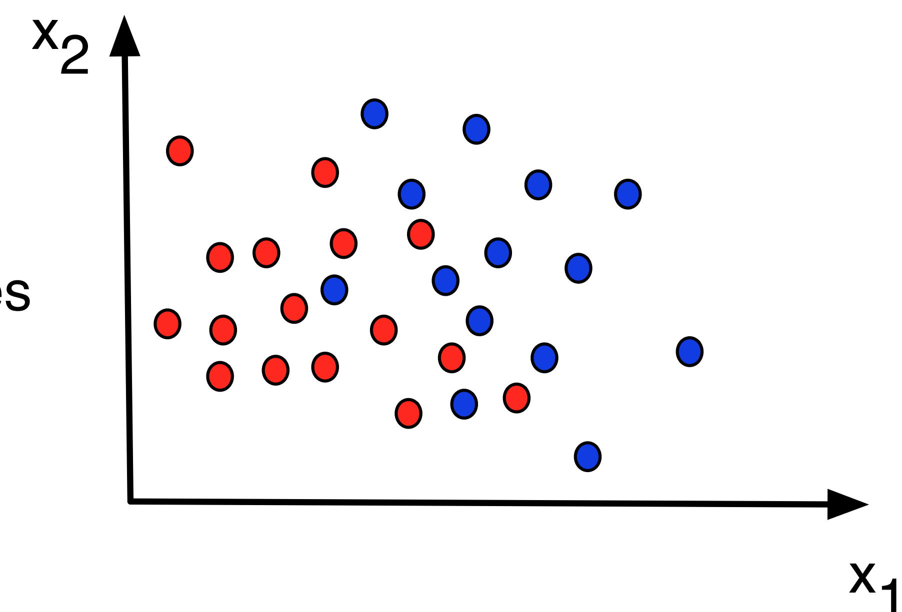
Classification: targets **t** are **discrete** values

We can visualise the target as different colour for each class

Inputs have one attribute
so 1D input space



Inputs have two attributes
so 2D input space



Binary Classification

$$t_n \in \{-1, 1\}$$

Multiclass/Multinomial Classification

$$t_n \in \{1, 2, \ldots, C\}$$

k-NN for classification?

Classification

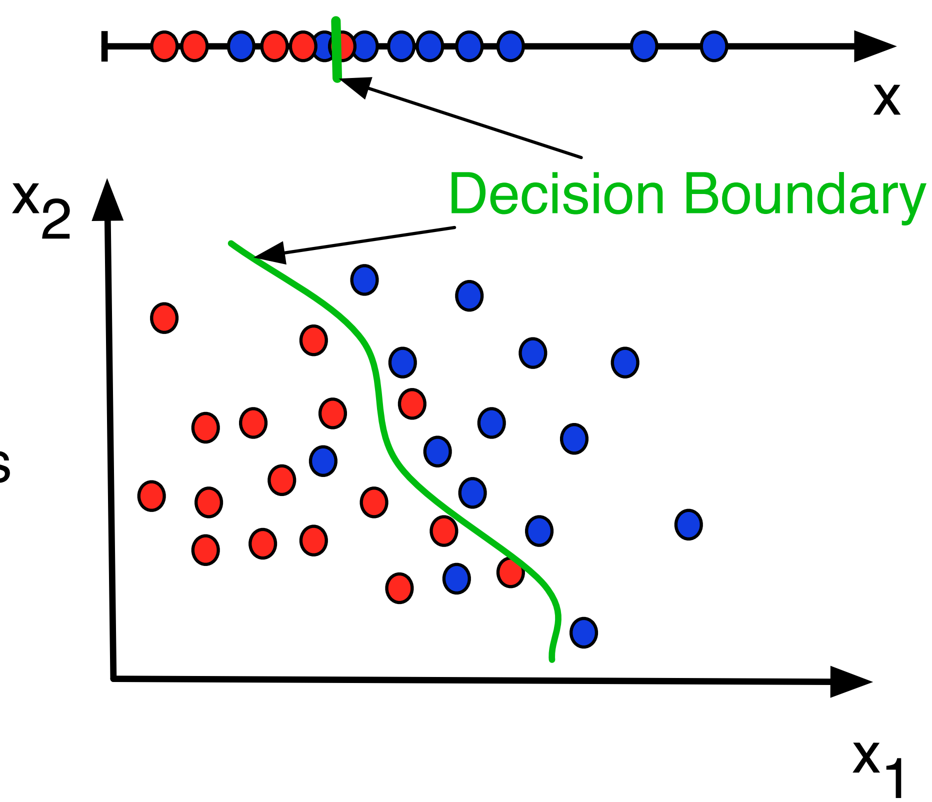
- The goal is to assign instances/inputs to target classes

$$t_n \in \{-1, 1\} \qquad\qquad t_n \in \{1, 2, \ldots, C\}$$

- The boundary between the classes where it is ***equiprobable to belong to either class*** is called the **decision boundary**

Inputs have one attribute so 1D input space

Inputs have two attributes so 2D input space



Classification with k-NN

Training:

- For each training example (input-output pair $\mathbf{x}_n, t_n$), add the example to the list *training_examples*

(Binary) Classification:

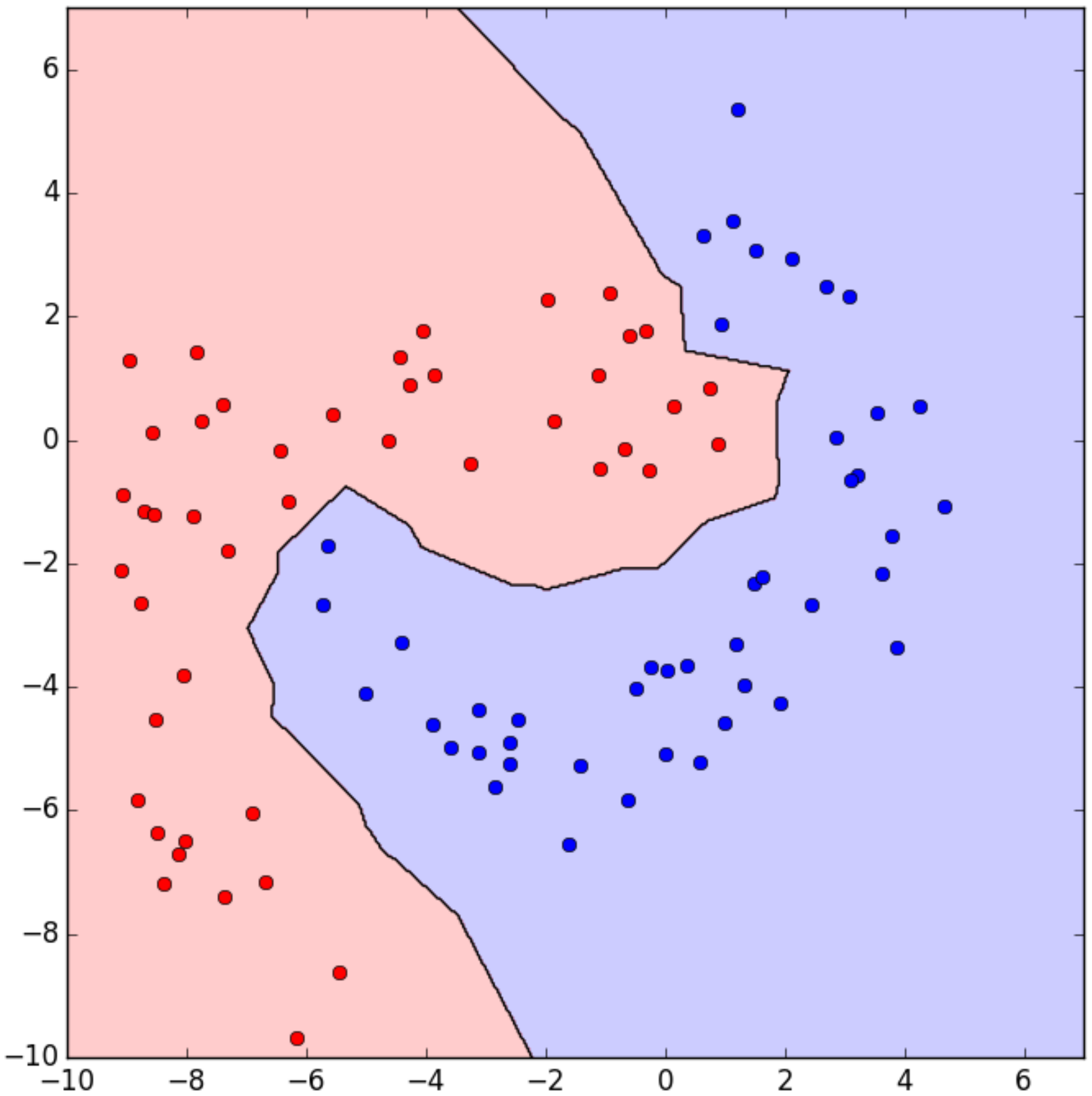
- Choose k, the number of neighbours we want
- Choose the distance function (e.g. Euclidean distance)
- Given a query instance **x*** to predict its output t*
  - Find $\mathbf{x}_1 \ldots \mathbf{x}_k$ the k instances that are **nearest** to **x*** using the selected distance
  - Return prediction: $t_n^* \leftarrow \text{majority}(t_1, \ldots, t_k)$

or more formally:

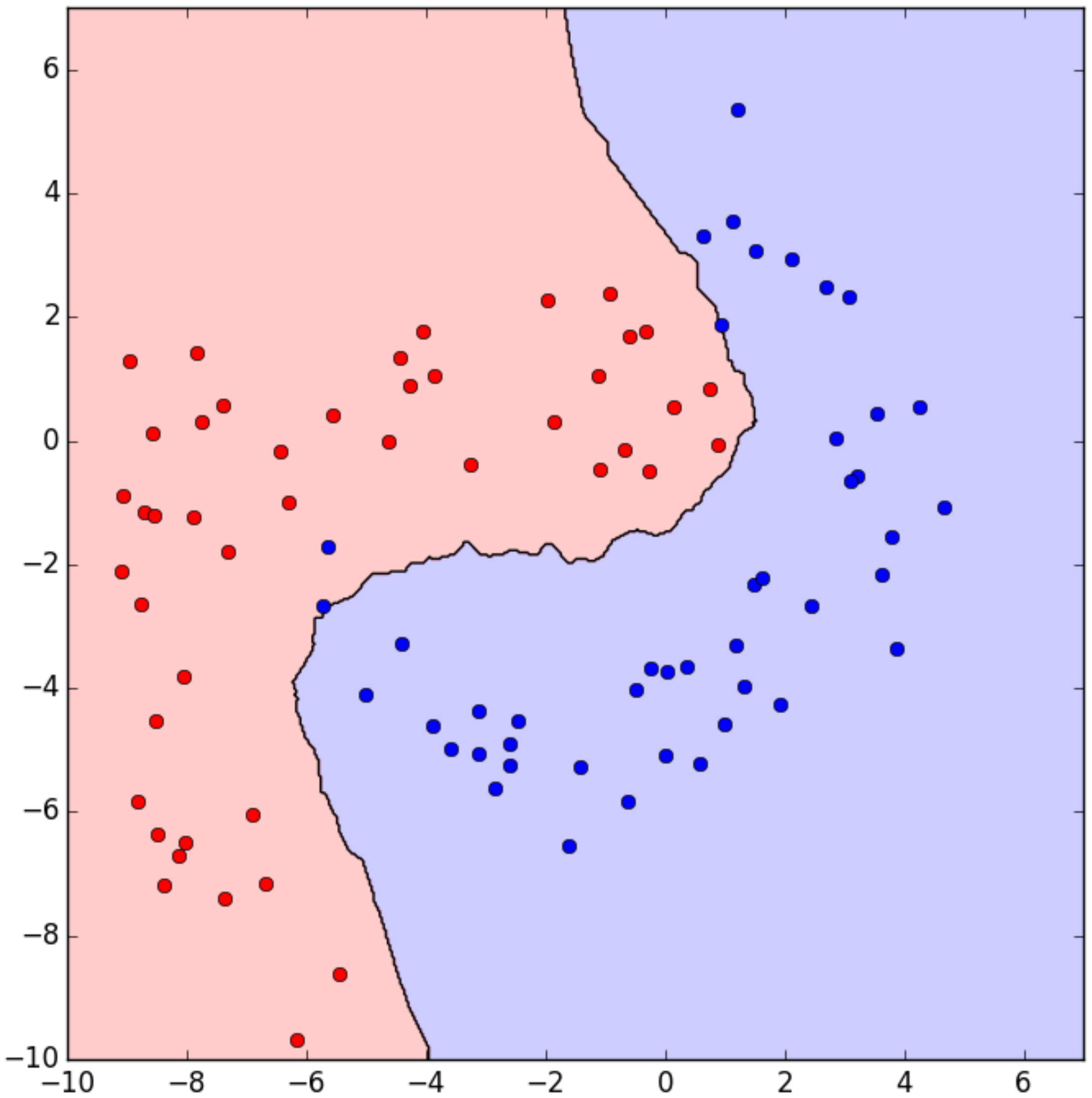
$$t_n^* \leftarrow \underset{u \in \{-1,1\}}{\text{argmax}} \sum_{i=1}^{k} \delta(u, t_i) \qquad \text{where } \delta(a,b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$

k-NN Classification

k=1



k=13

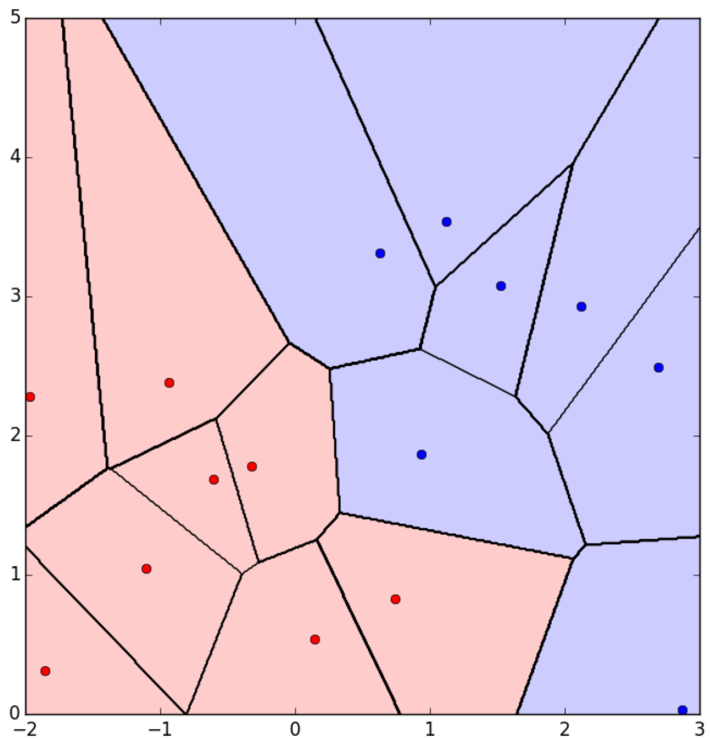


Piece-wise linear decision boundary

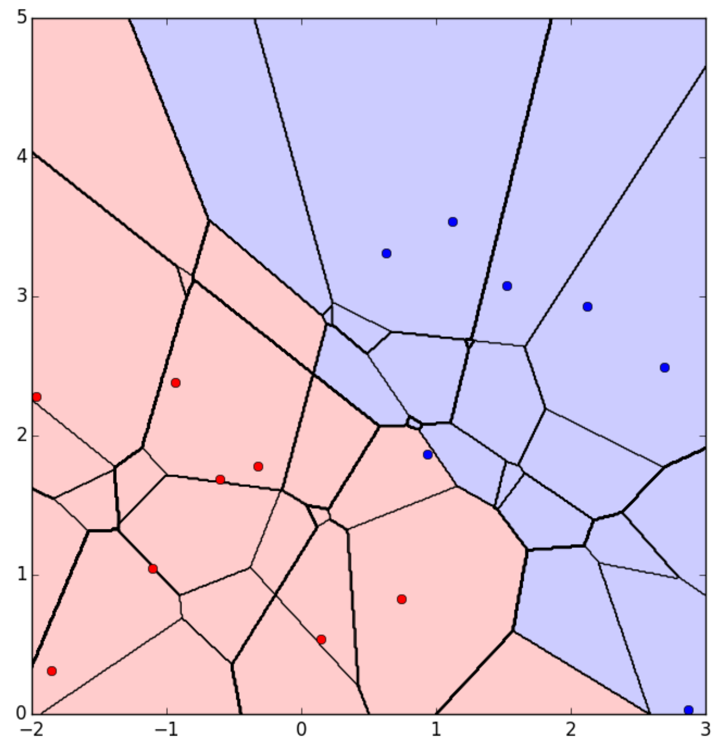
k-NN Classification: Effect of k

- k controls the complexity of the hypothesis we learn
- If k even then we need to resolve ties (in classification)
- As k increases we utilise more neighbours
- More neighbours = smoother decision boundary = less complex boundary
- k-NN creates Voronoy tessellations

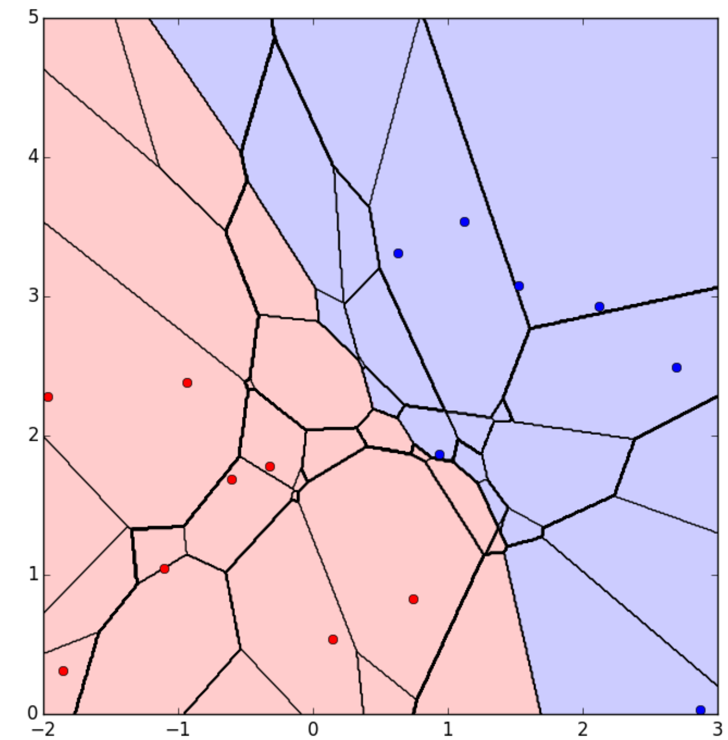
k=1



k=3

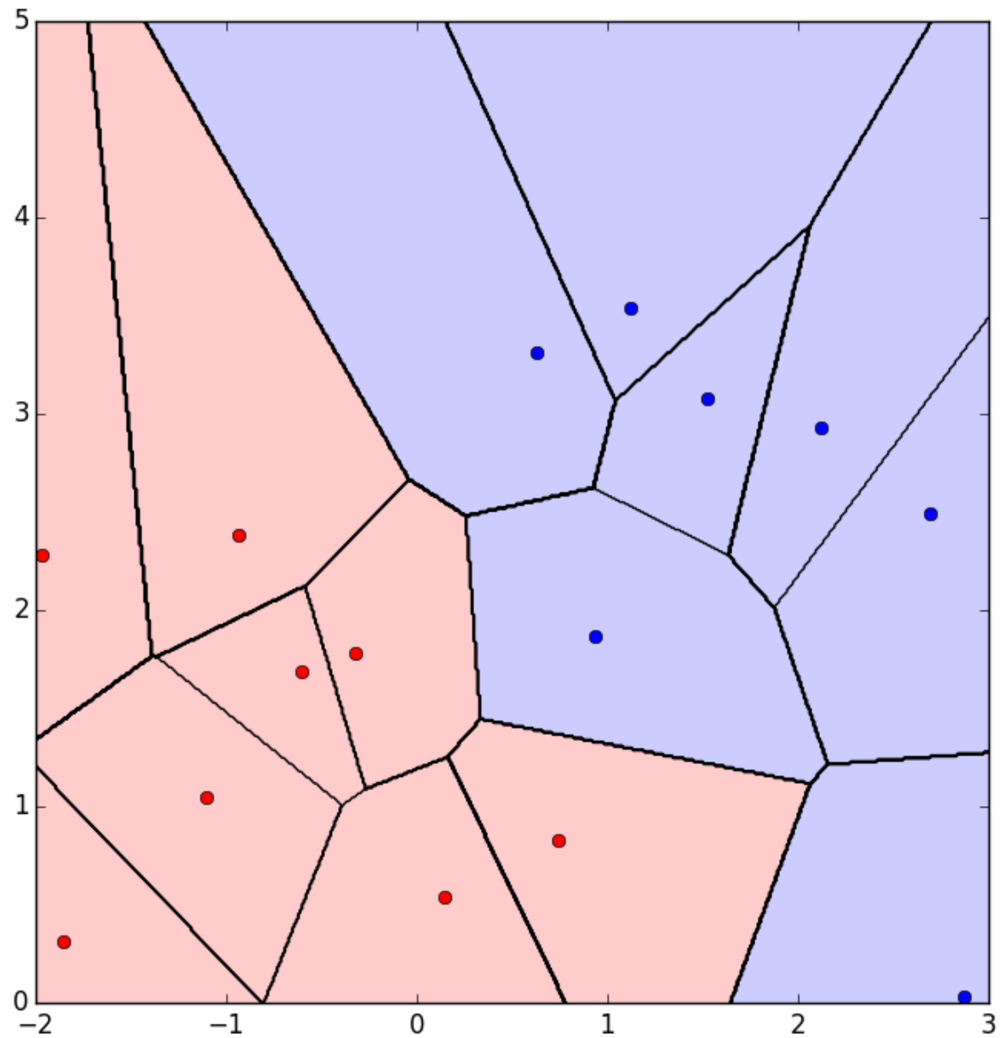


k=5

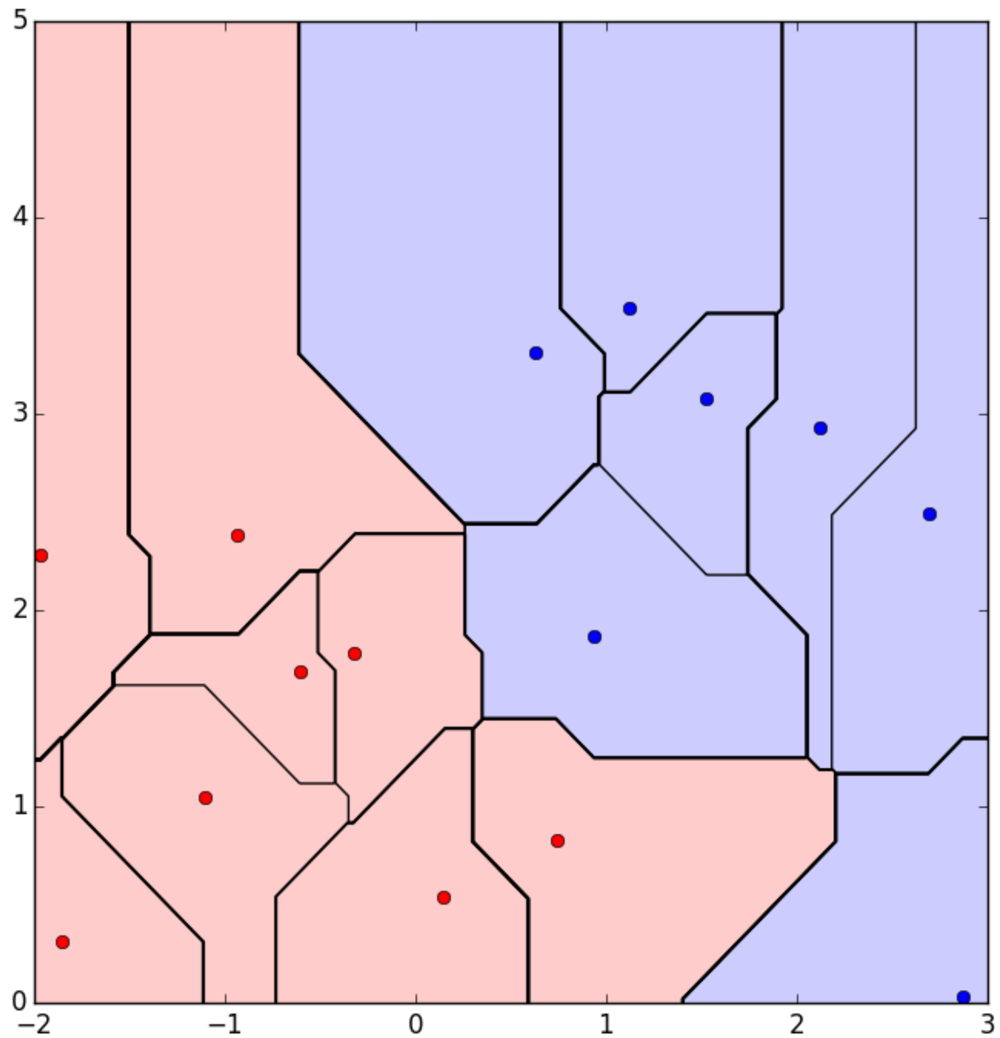


k-NN Classification: Effect of distance metric (L1 vs L2)

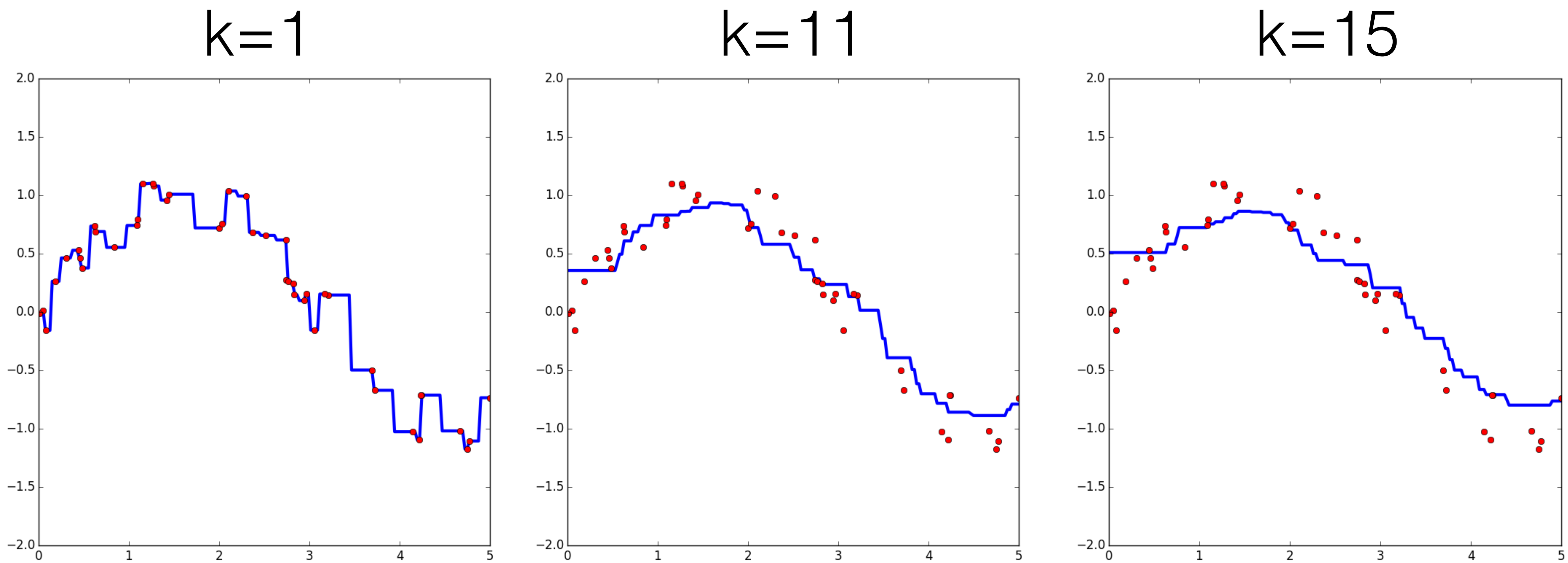
k=1, L2 (Euclidean) distance



k=1, L1 (Manhattan) distance



k-NN Regression: Effect of k



- As we increase k, smoother piece-wise linear function
- Small k can lead to severe **overfitting!**
- Boundary effects (no neighbours on some sides)
- Use CV to choose K

Distance-weighted k-NN

Any extensions? Equal vote?

- Weigh the vote of each neighbour by its distance to the observation
- Can help break ties when k is even
- Can help deal with noisy data and outliers

Regression:

$$t^* \leftarrow \frac{\sum_{k=1}^{K} w_i t_k}{\sum_{i=1}^{k} w_i}$$

$$w_i = \frac{1}{d(\mathbf{x}_i, \mathbf{x}^*)^2}$$

Classification:

$$t_n^* \leftarrow \underset{u \,\in\, \{-1,1\}}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(u, t_i)$$

$$\text{where } \delta(a,b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$

Remarks on k-NN

- Vanilla k-NN will not perform well in high-D as distances "break" in high-D
- In high-D, data concentrates so distances go to extremes
- Every dimension = an attribute. Some attributes are useless…
- Learn which attributes are important and weight these dimensions more
- Assign weights for every dimension and learn via e.g. cross-validation
- Can use tree data structures to improve search time for neighbours
- High computational cost to store all the training data in big data settings

	Naive	k-d tree
NN Search:	$O(ND)$	$O(\log N)$

Very simple algorithm but very successful over the years!