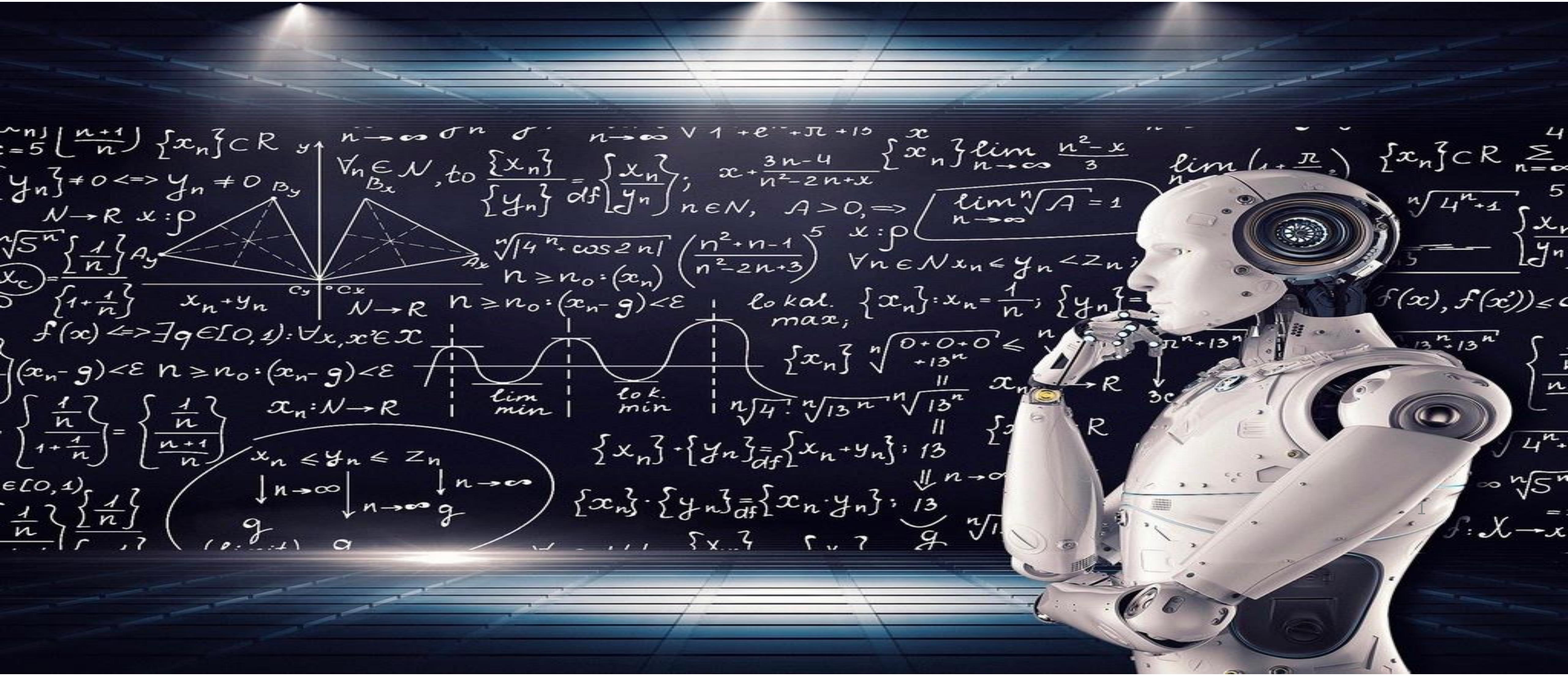
K-Nearest Neighbor

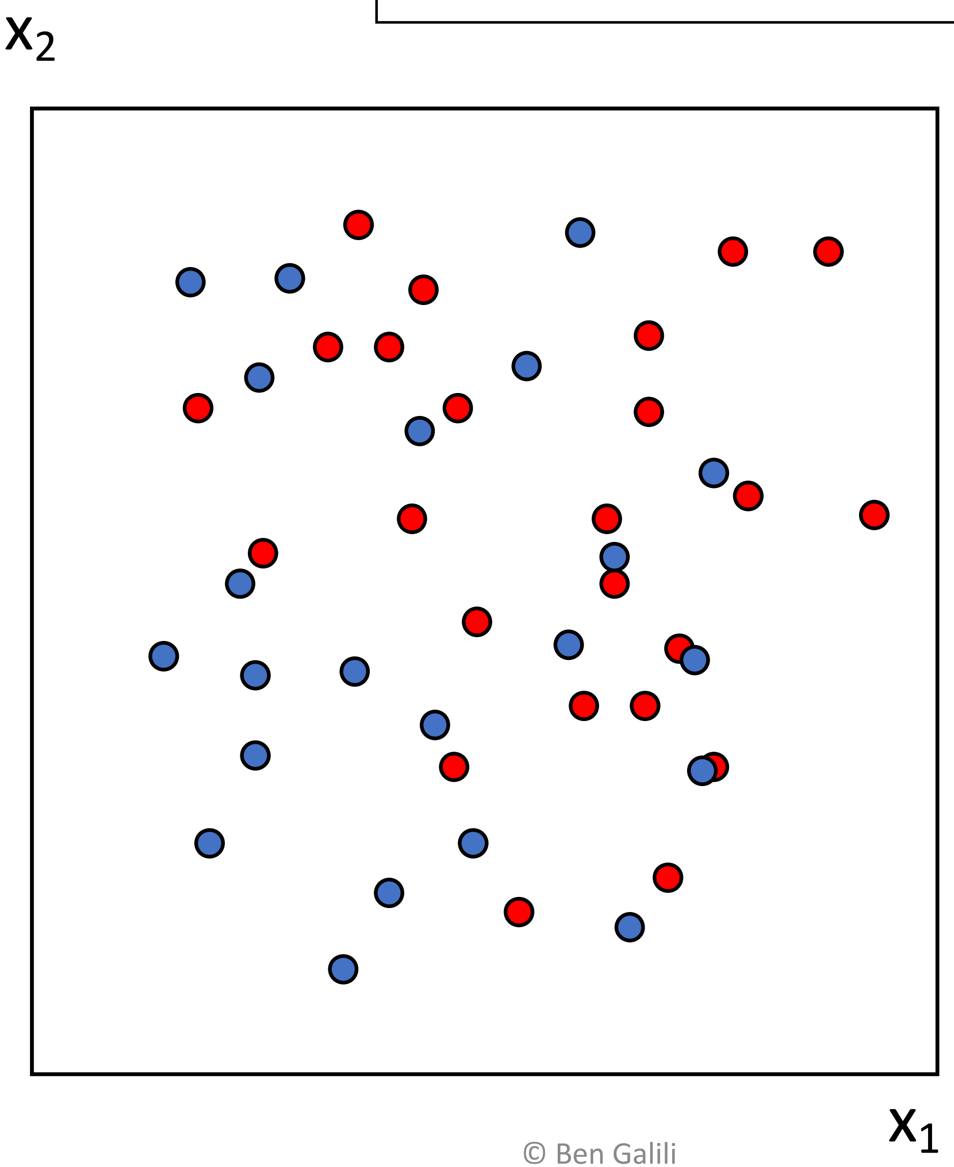


IDC HERZLIYA

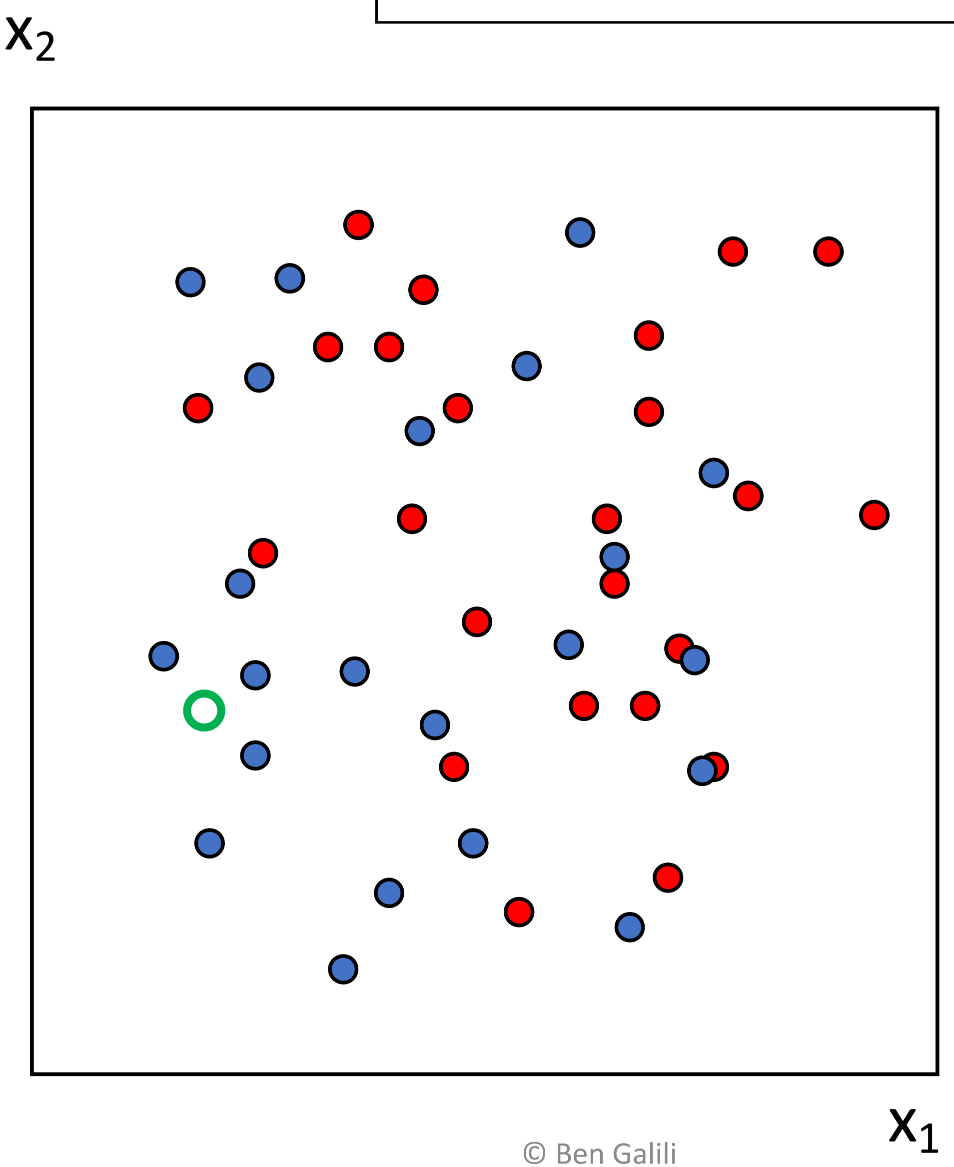
Instance Based Learning

- How do you know if the rent of the house is reasonable?
  - You compare it to known example’s
- This is how people naturally use Instance Based Learning algorithm

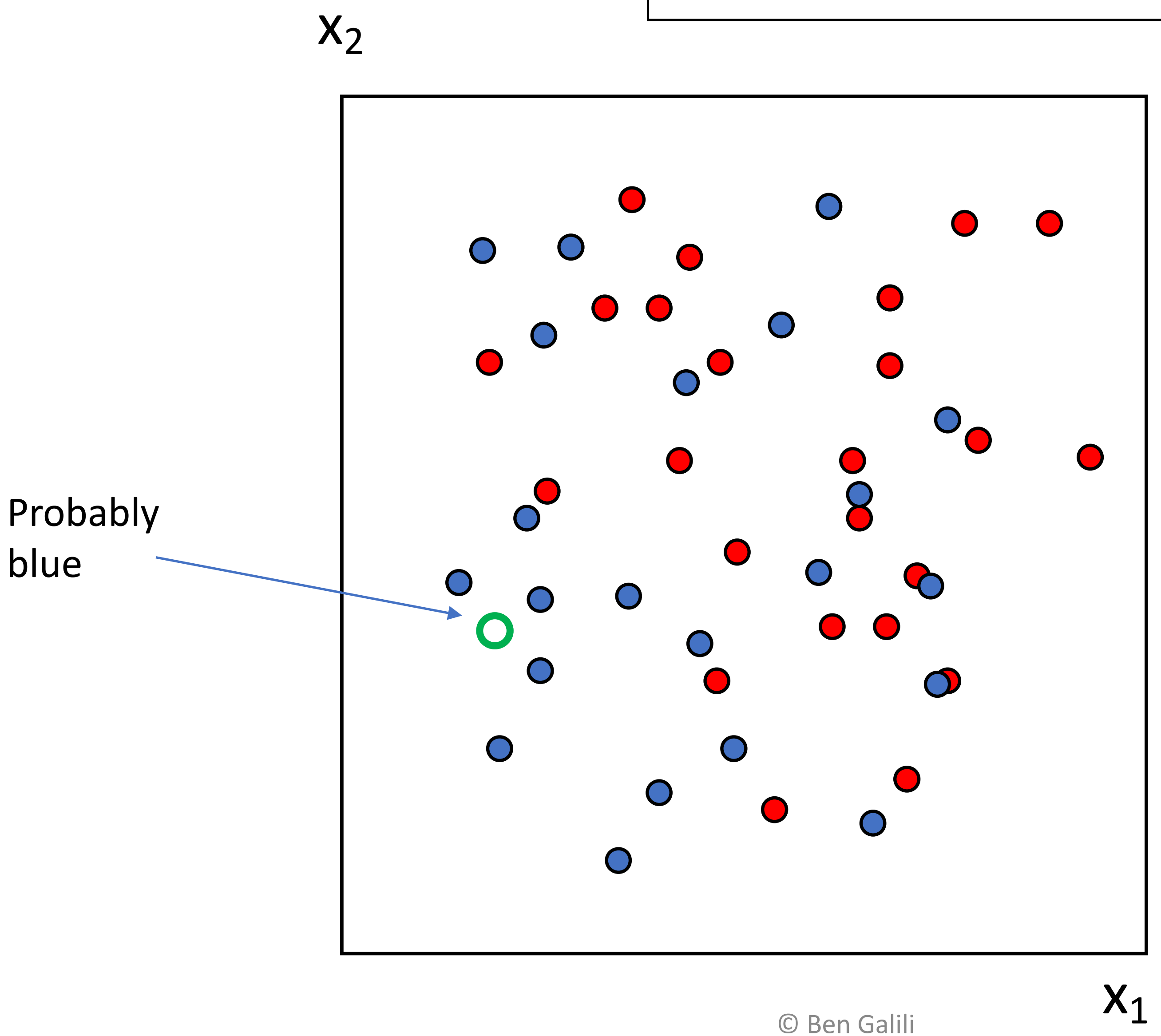
Model Based Learning vs Instance Based Learning



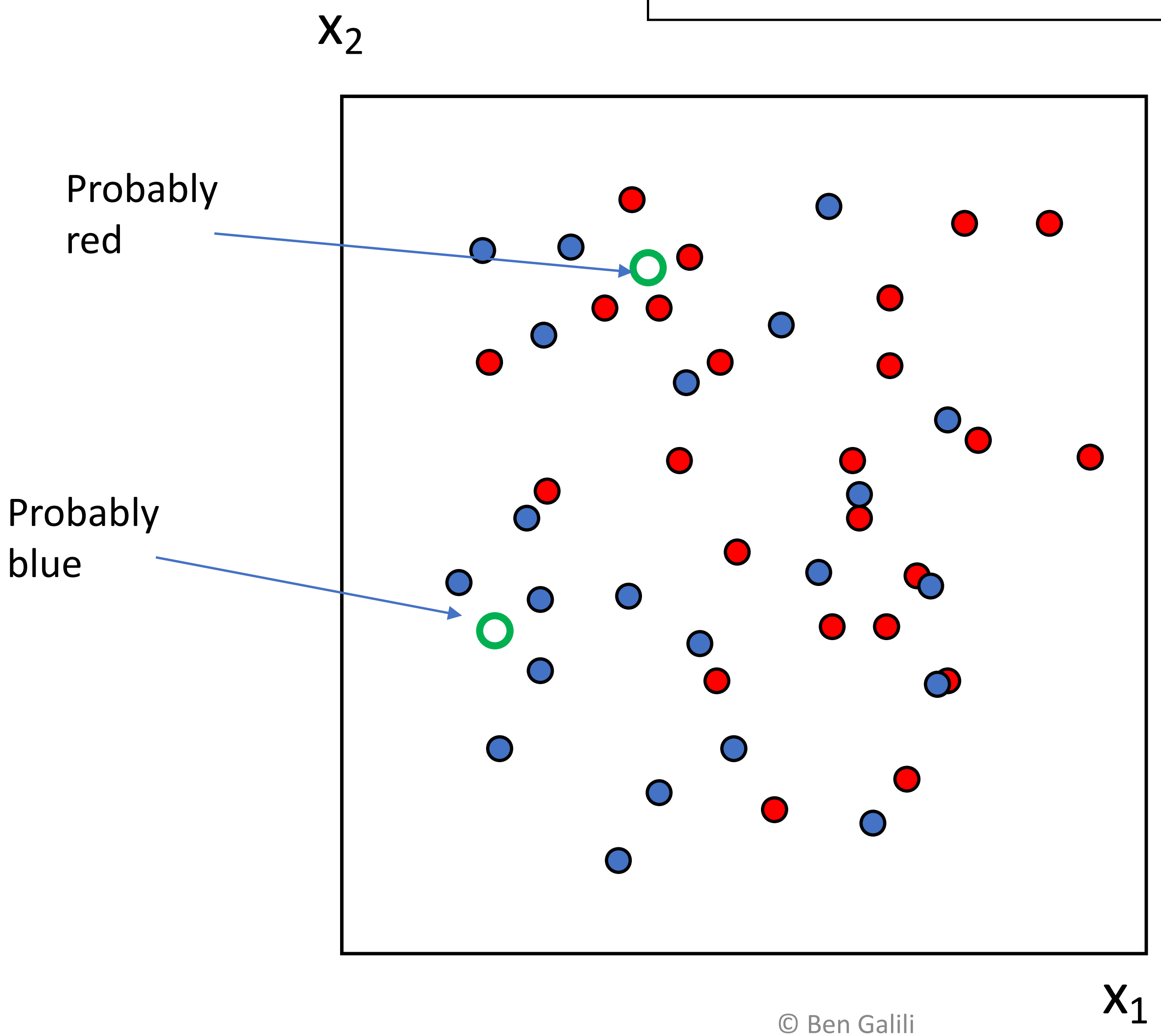
Model Based Learning vs Instance Based Learning



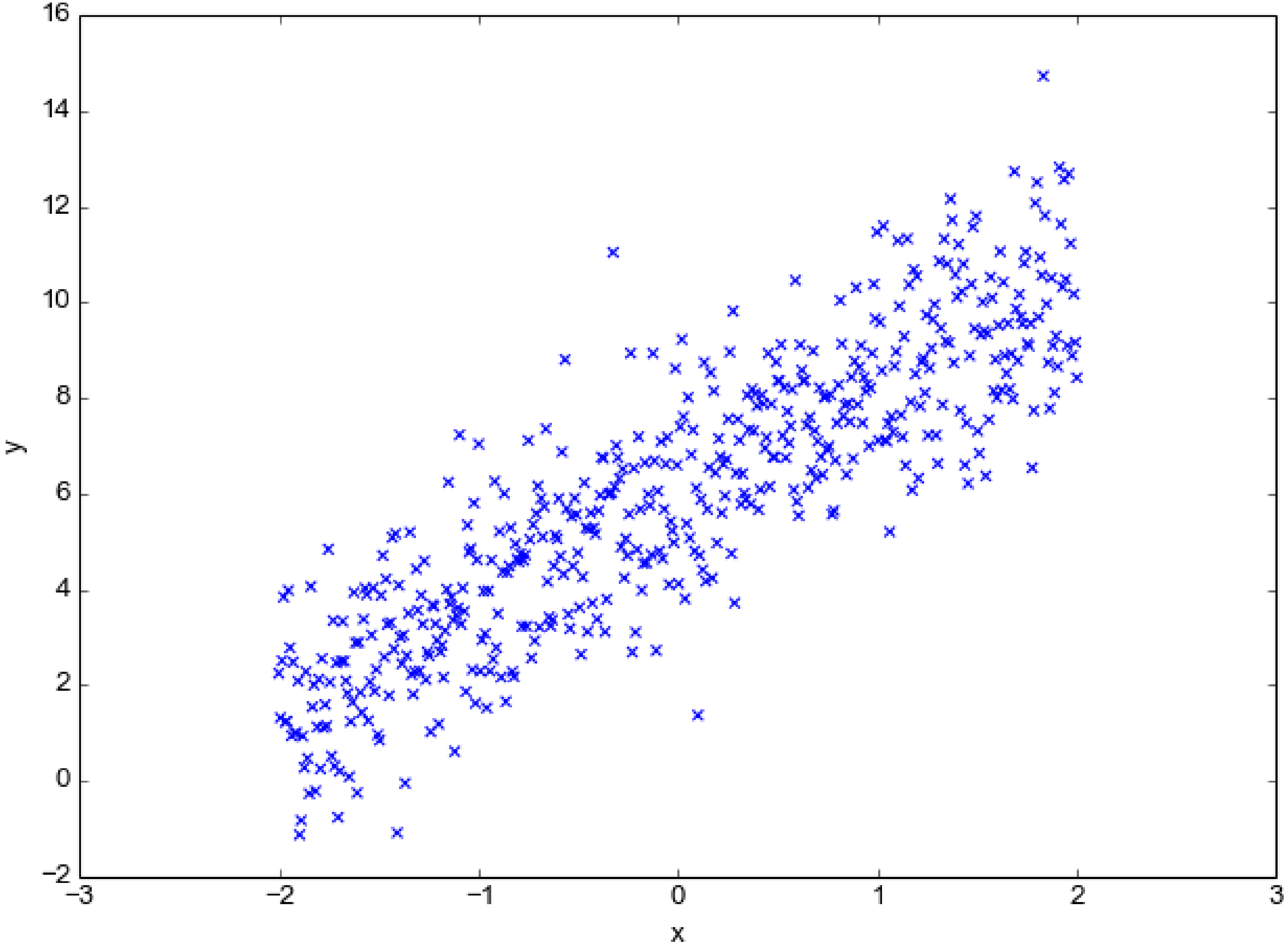
Model Based Learning vs Instance Based Learning



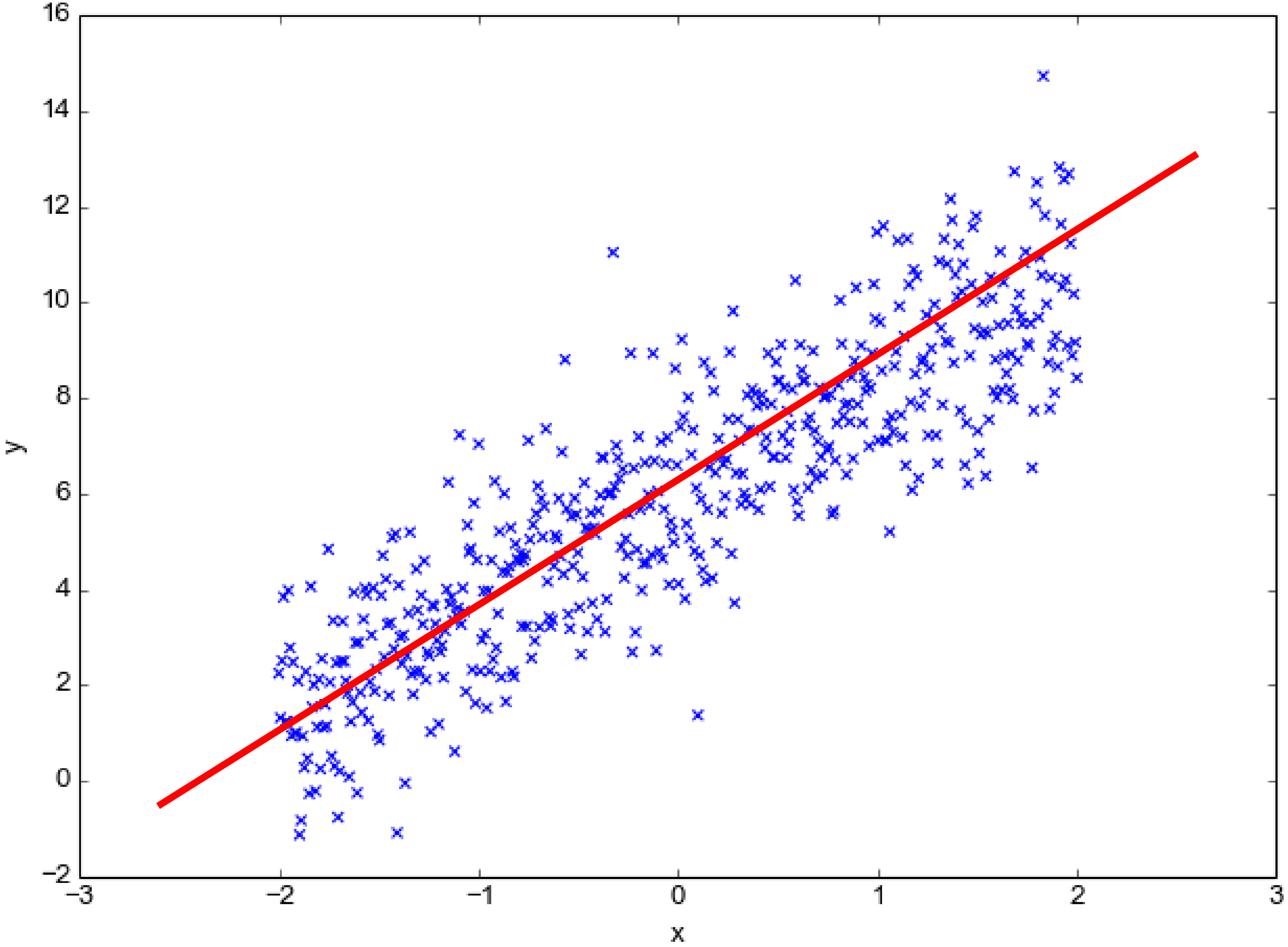
Model Based Learning vs Instance Based Learning



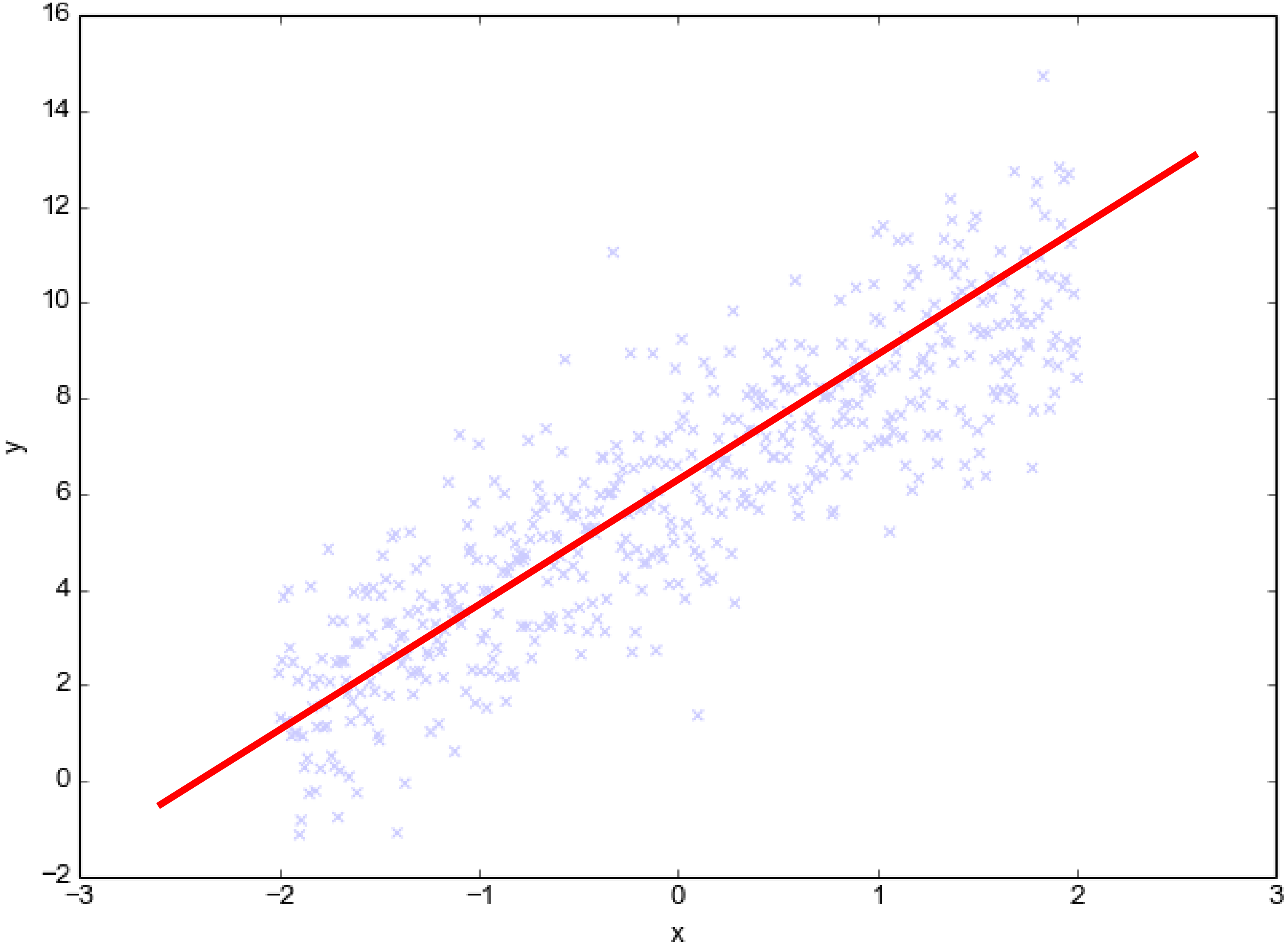
Model Based Learning vs Instance Based Learning



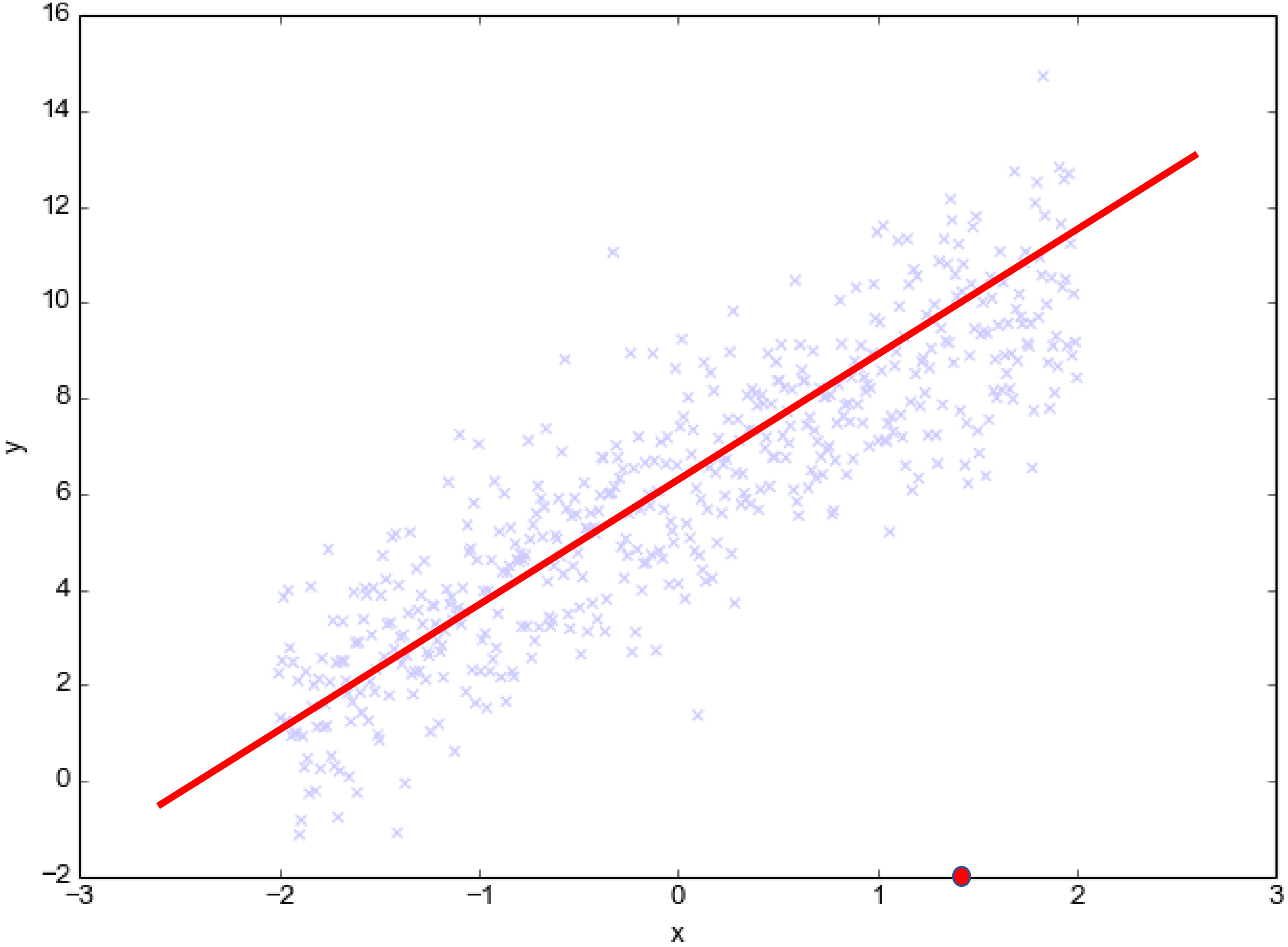
Model Based Learning vs Instance Based Learning



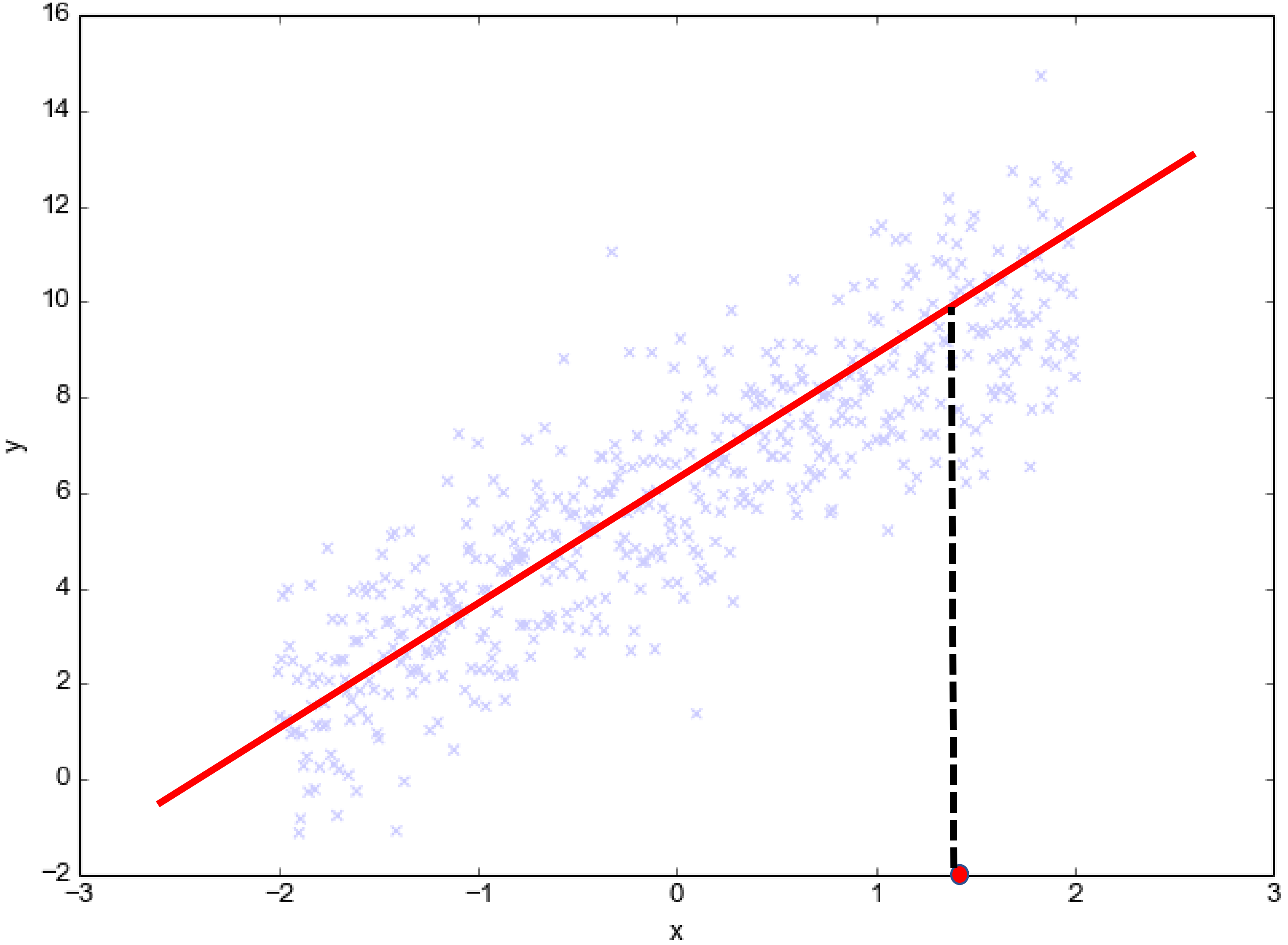
Model Based Learning vs Instance Based Learning



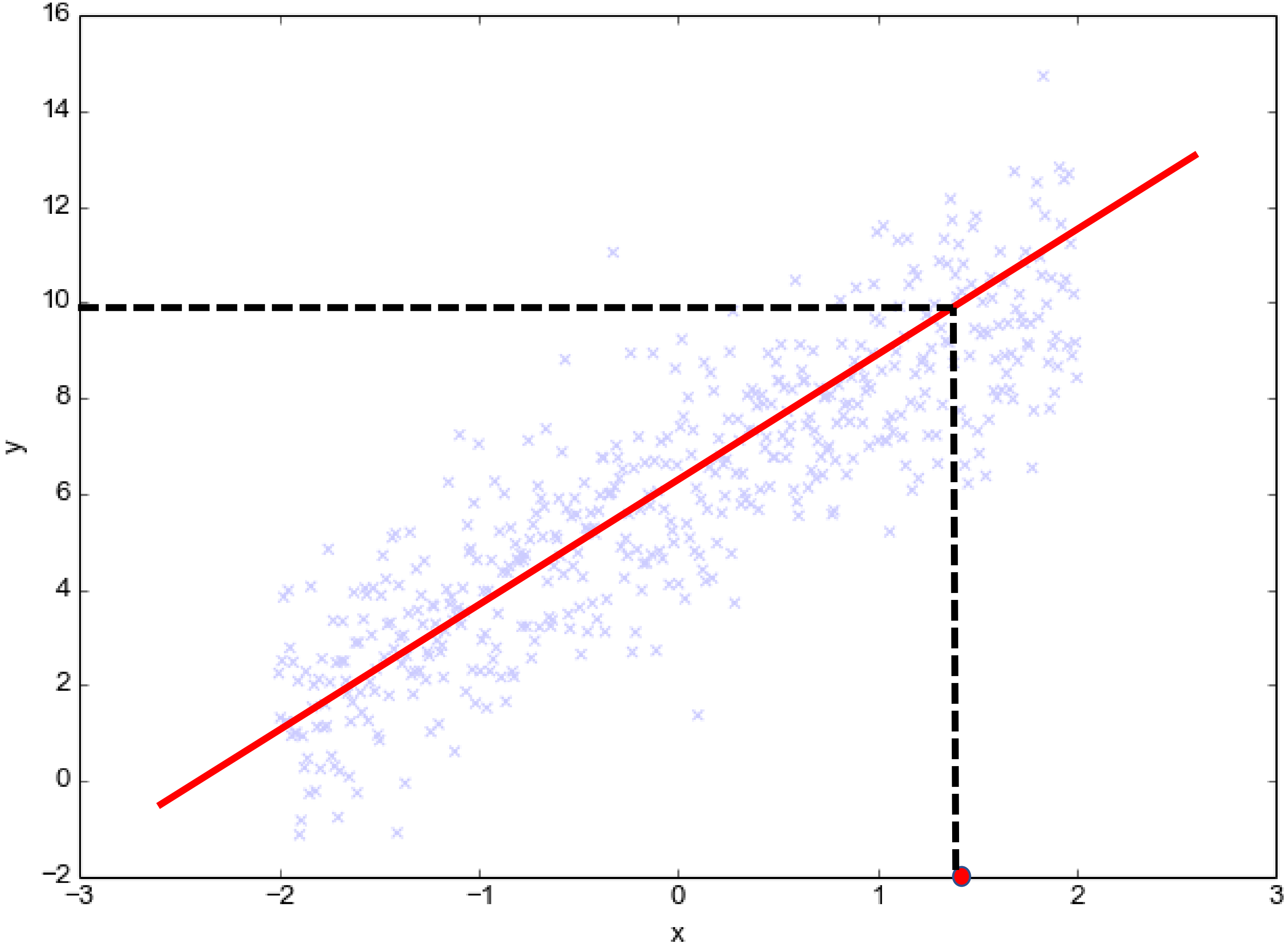
Model Based Learning vs Instance Based Learning



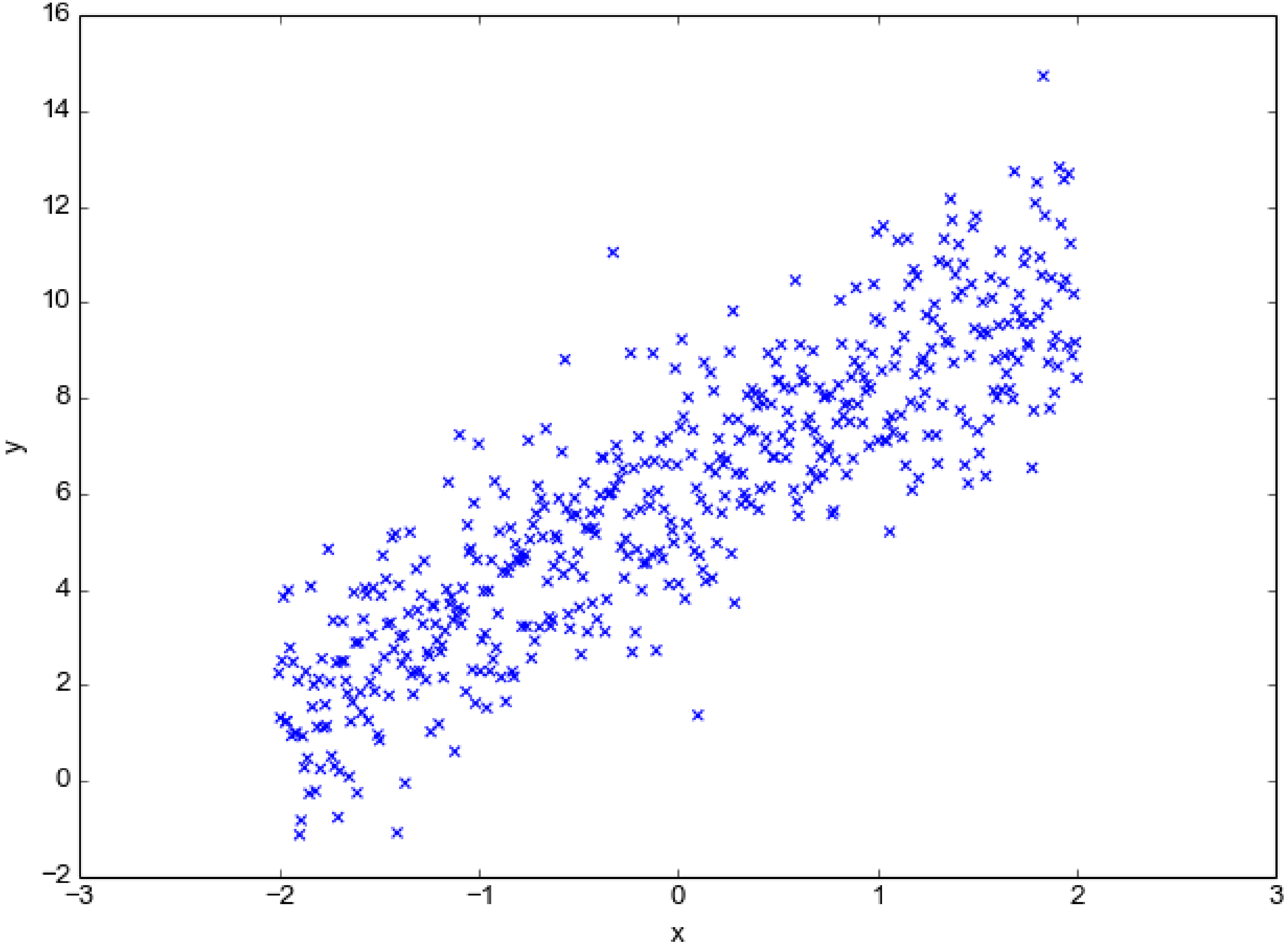
Model Based Learning vs Instance Based Learning



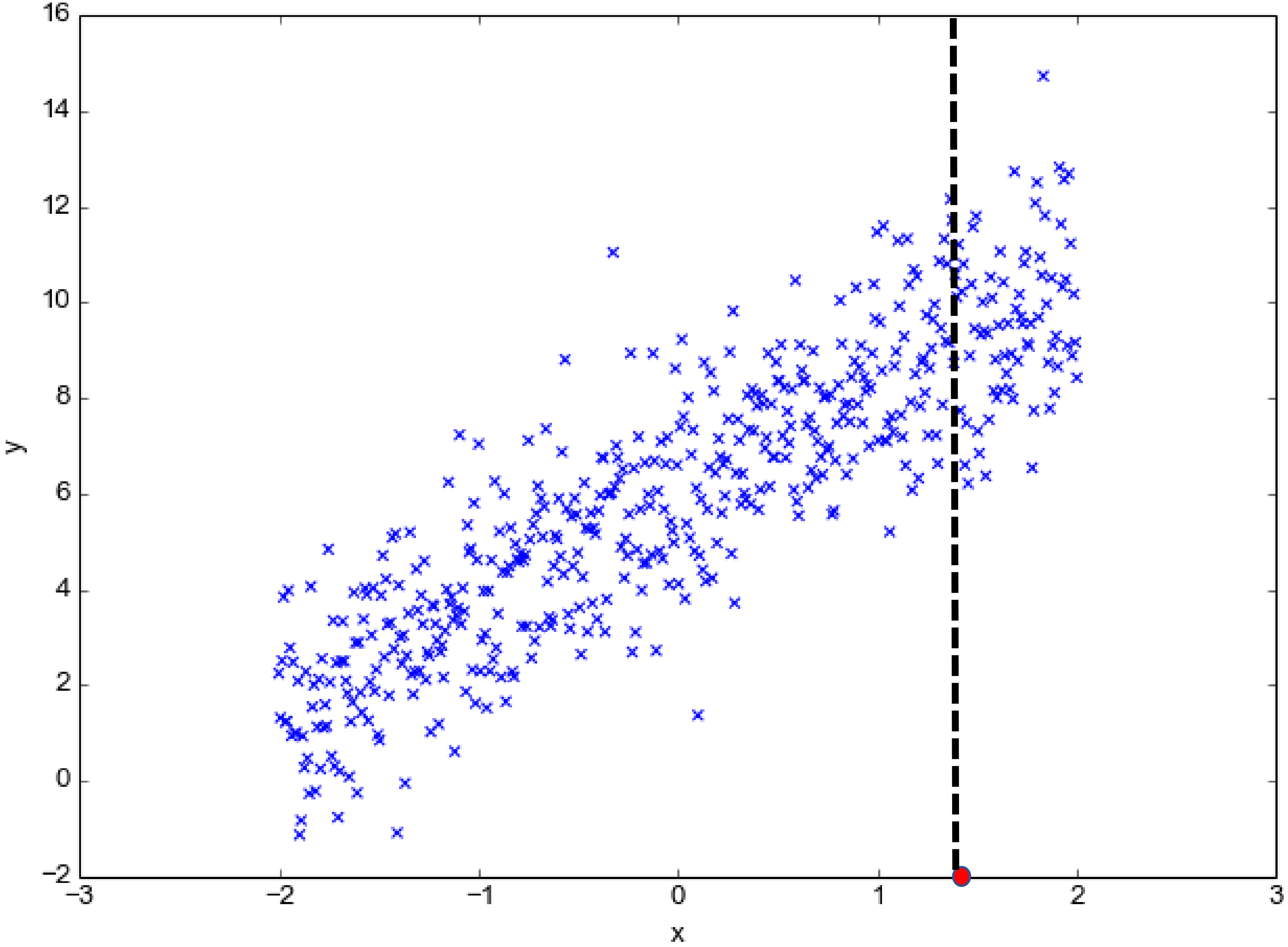
Model Based Learning vs Instance Based Learning



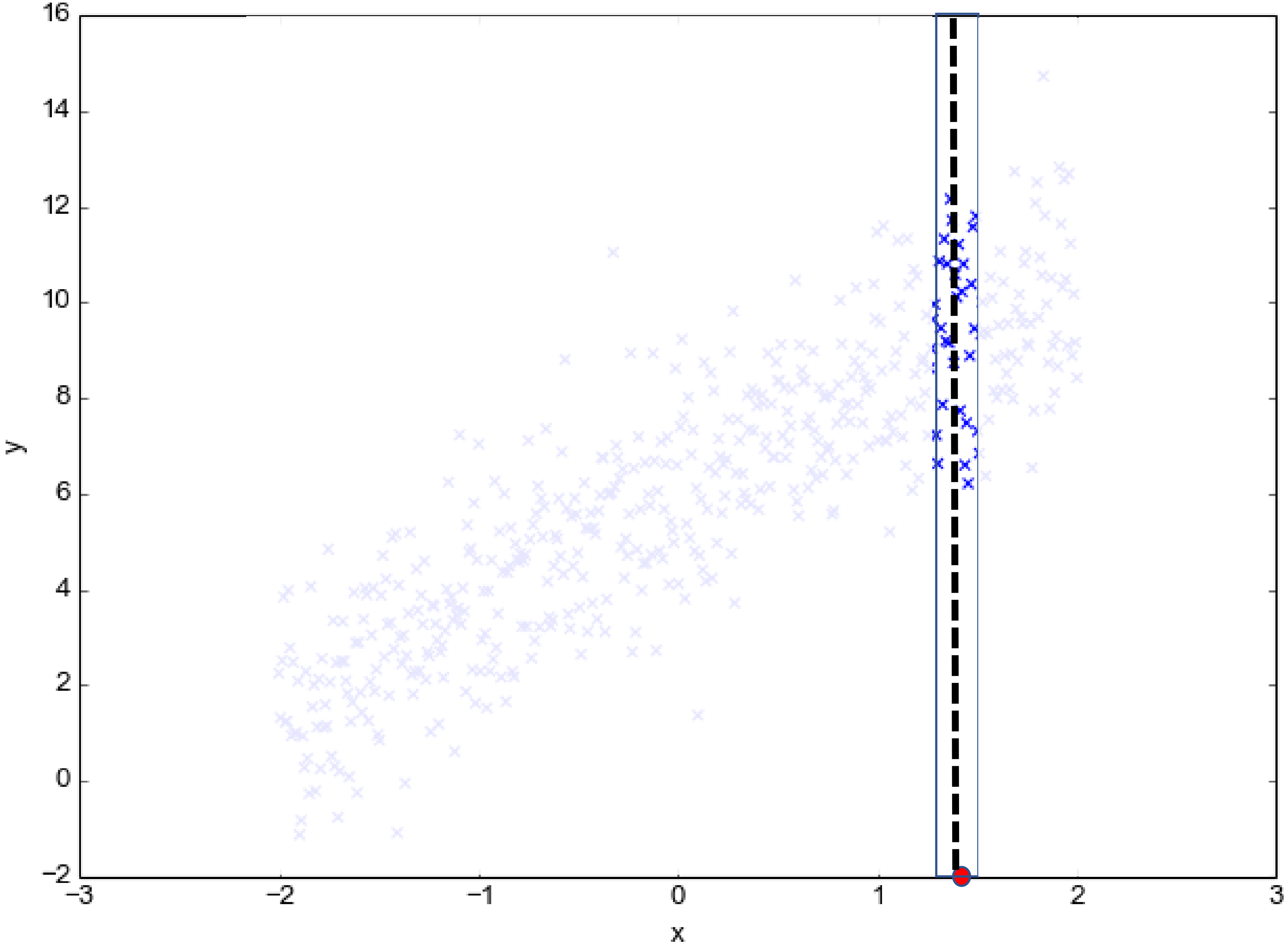
Model Based Learning vs Instance Based Learning



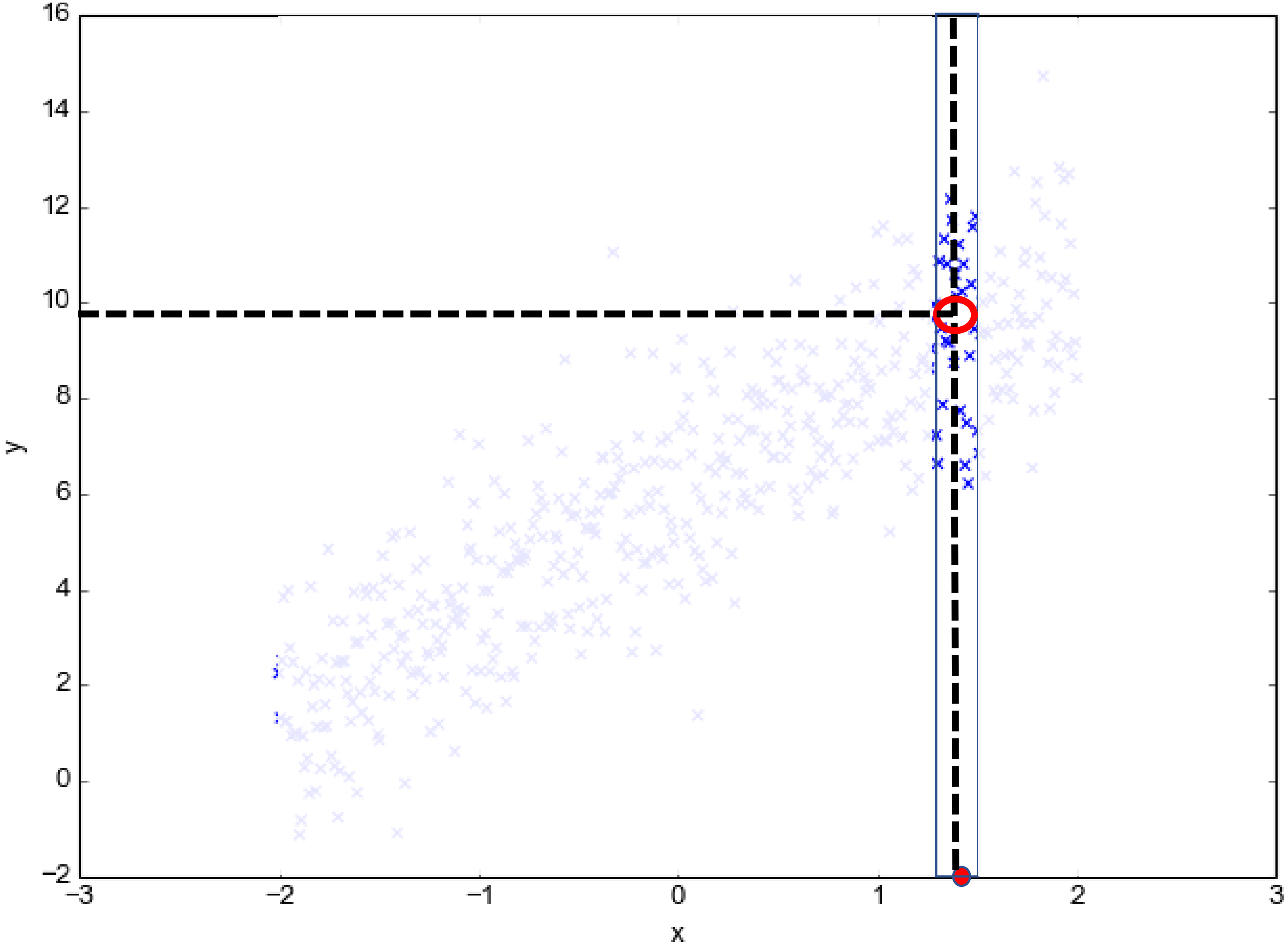
Model Based Learning vs Instance Based Learning



Model Based Learning vs Instance Based Learning



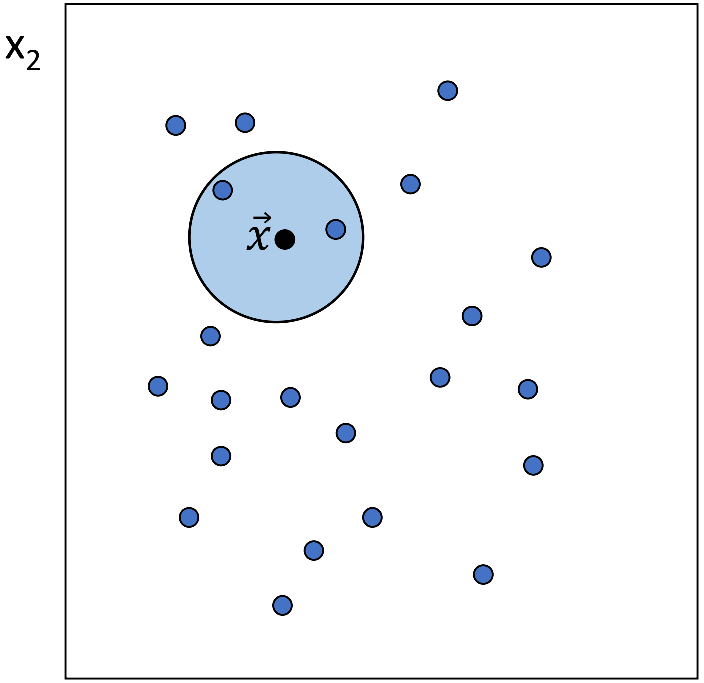
Model Based Learning vs Instance Based Learning



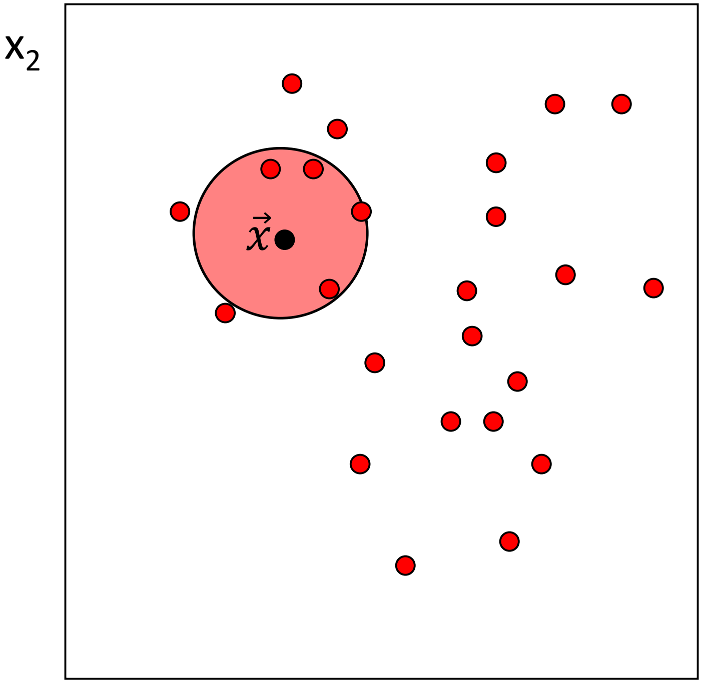
Instance Based Learning

- Family of learning algorithm that:
  - Doesn’t build a model to the data (like tree in Decision Tree)
  - Instead – compares new instance with instances seen in training
- Time complexity:
  - Fast learning (No learning…)
  - Potentially slow classification/prediction (O(n))
- Space complexity:
  - Store all in instances (O(n))
- Used in both Classification and Regression

Parzen window – Classification

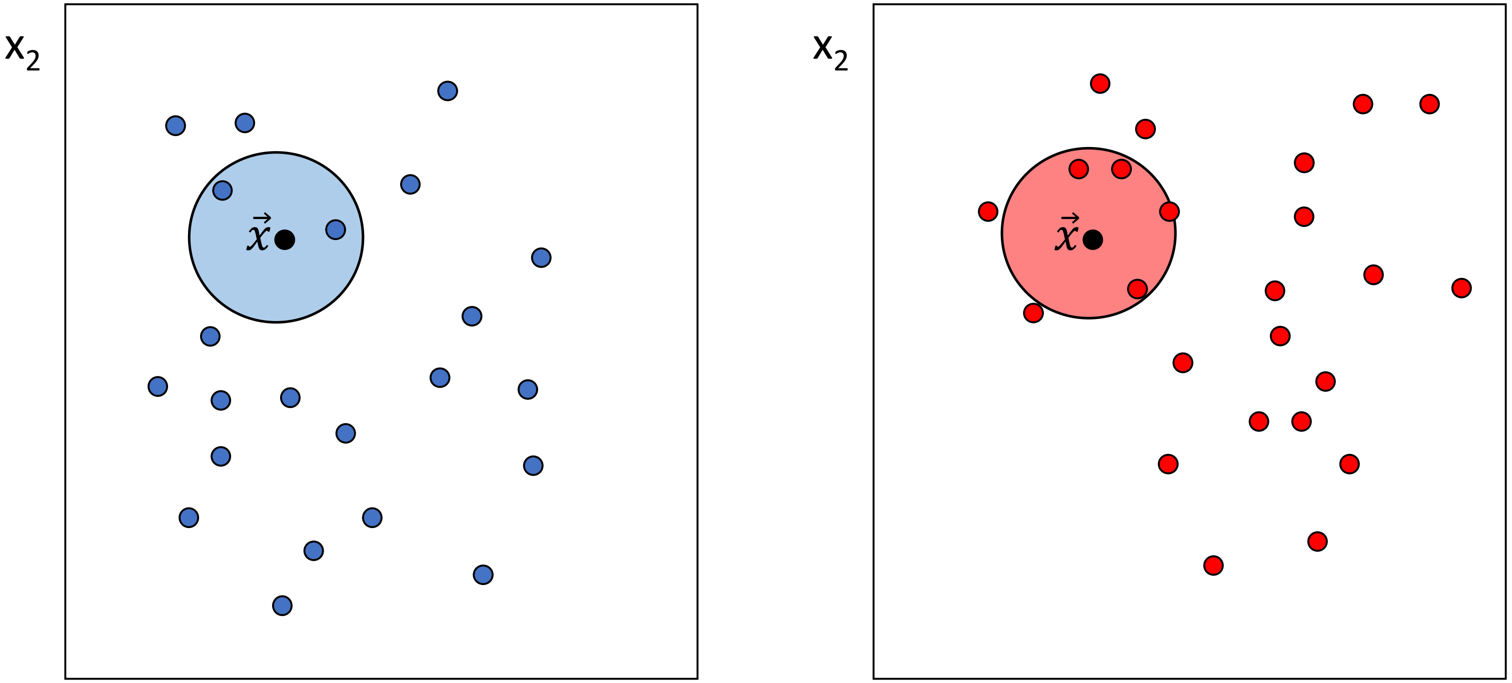


$P(\vec{\boldsymbol{x}}_{new} \mid A_1)$



$P(\vec{\boldsymbol{x}}_{new} \mid A_2)$

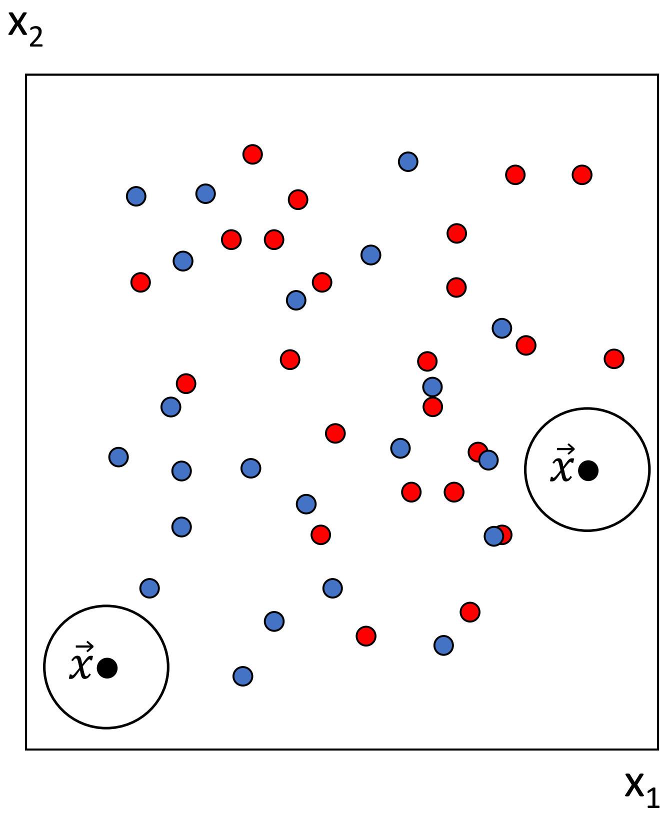
Parzen window – Classification



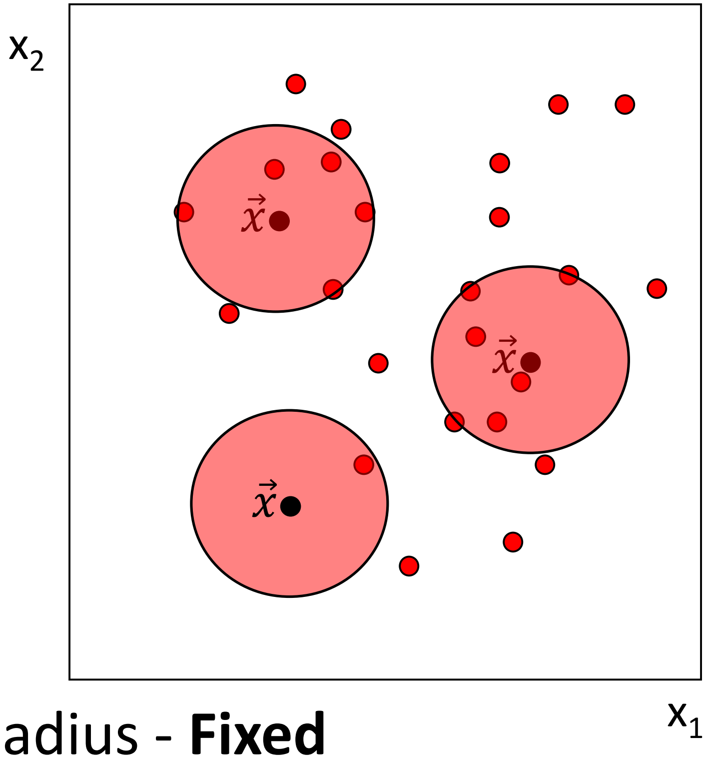
$P(\vec{\boldsymbol{x}}_{new} \,|\, A_1)$ $\qquad$ $P(\vec{\boldsymbol{x}}_{new} \,|\, A_2)$

$$p(\vec{x}_{new}|A_i) = \frac{1}{n_i} \sum_{\vec{x} \in A_i} \frac{1}{h^d} K\left(\frac{\vec{x}_{new} - \vec{x}}{h}\right)$$

Parzen window – problems



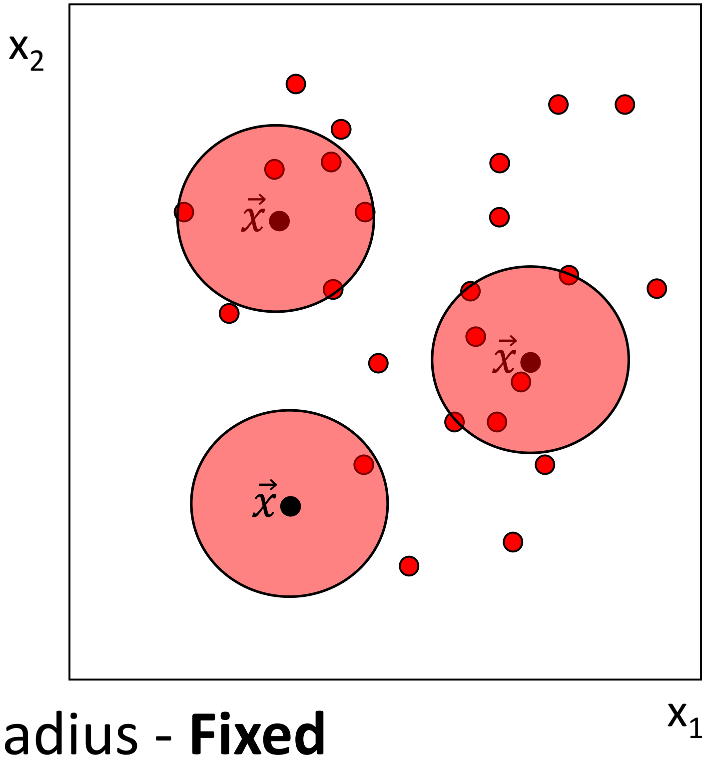
From Parzen window to kNN



Radius - **Fixed**

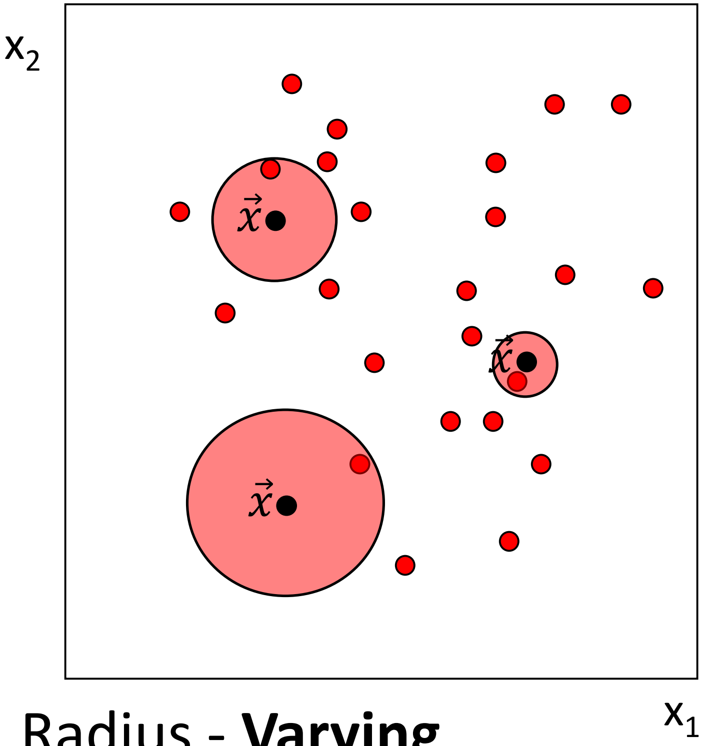
Number of samples in window - **Varying**

From Parzen window to kNN



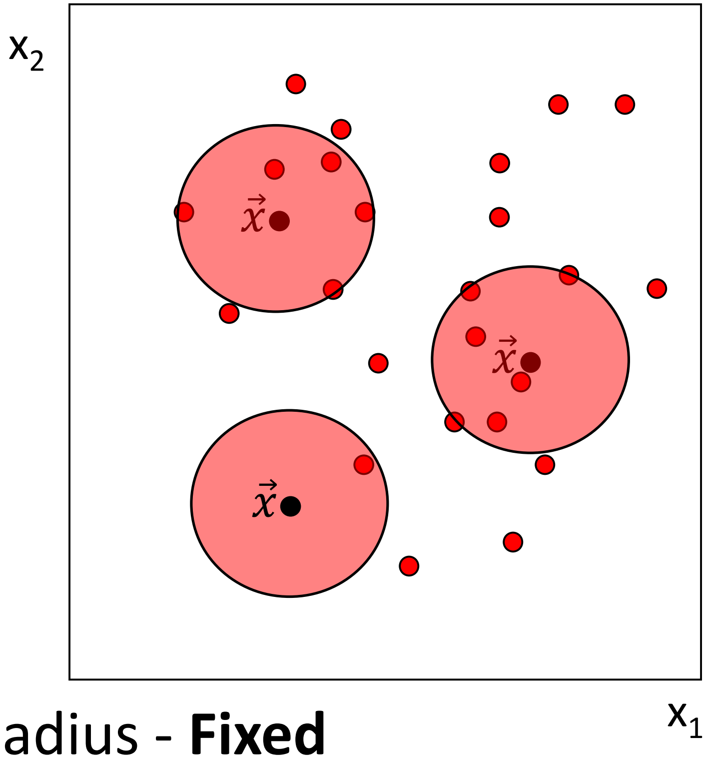
Radius - **Fixed**
Number of samples in window - **Varying**

kNN, K=1



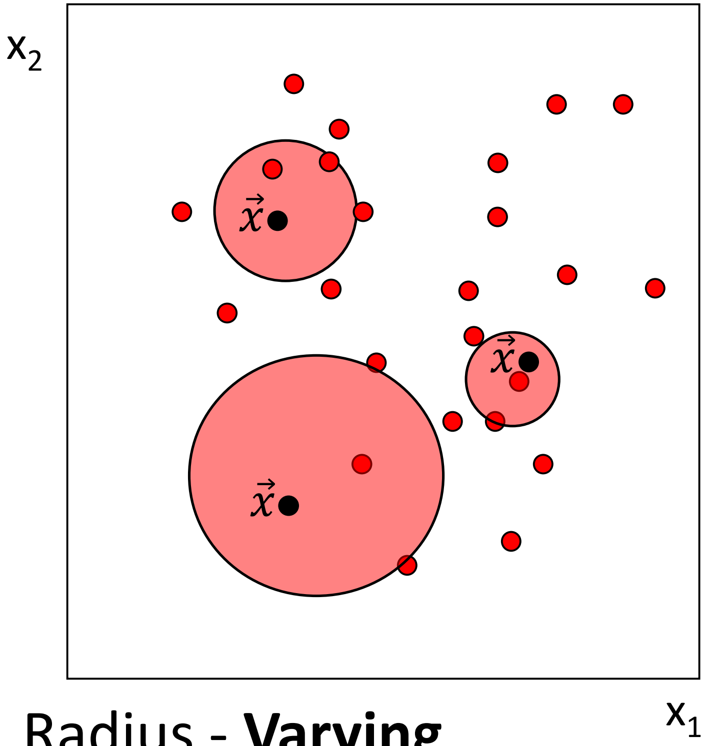
Radius - **Varying**
Number of samples in window - **Fixed**

From Parzen window to kNN



Radius - **Fixed**
Number of samples in window - **Varying**

kNN, **K=2**



Radius - **Varying**
Number of samples in window - **Fixed**

K-Nearest Neighbors – kNN

- Nearest Neighbor prediction:
  - On input instance, find the “nearest” training instance and predict whatever the neighbor’s target value is
- K-Nearest Neighbor prediction:
  - On input instance, find the k “nearest” instances and estimate the majority (for discrete) or average (for continuous) of their target values

Prediction

- On input instance x find k nearest neighbors $\{x^{(i)}\}$ for $i \in \{1, \dots, k\}$ and predict:
  - For regression [**average**] :

$$\hat{f}(x) = \frac{1}{k}\sum_{i=1}^{k} f(x^{(i)})$$

  - For classification [**majority vote**]:

$$\hat{f}(x) = MAJ_i\left(\{f\left(x^{(i)}\right)\}\right)$$

* Where MAJ is the majority function over all i

Pros and Cons

- Advantages:
  - Training is fast
  - Can learn very complex target functions easily
  - You don't lose information
- Disadvantages
  - Slow at prediction time
  - Lots of memory storage
  - Easily fooled by irrelevant attributes\instances

Questions

- How to find nearest? ***What is near?***
- Slow query & Large space
- How to choose k?

Questions

- How to find nearest? ***What is "near"?***

- Slow query & Large space

- How to choose k?

Distance For Numeric Features

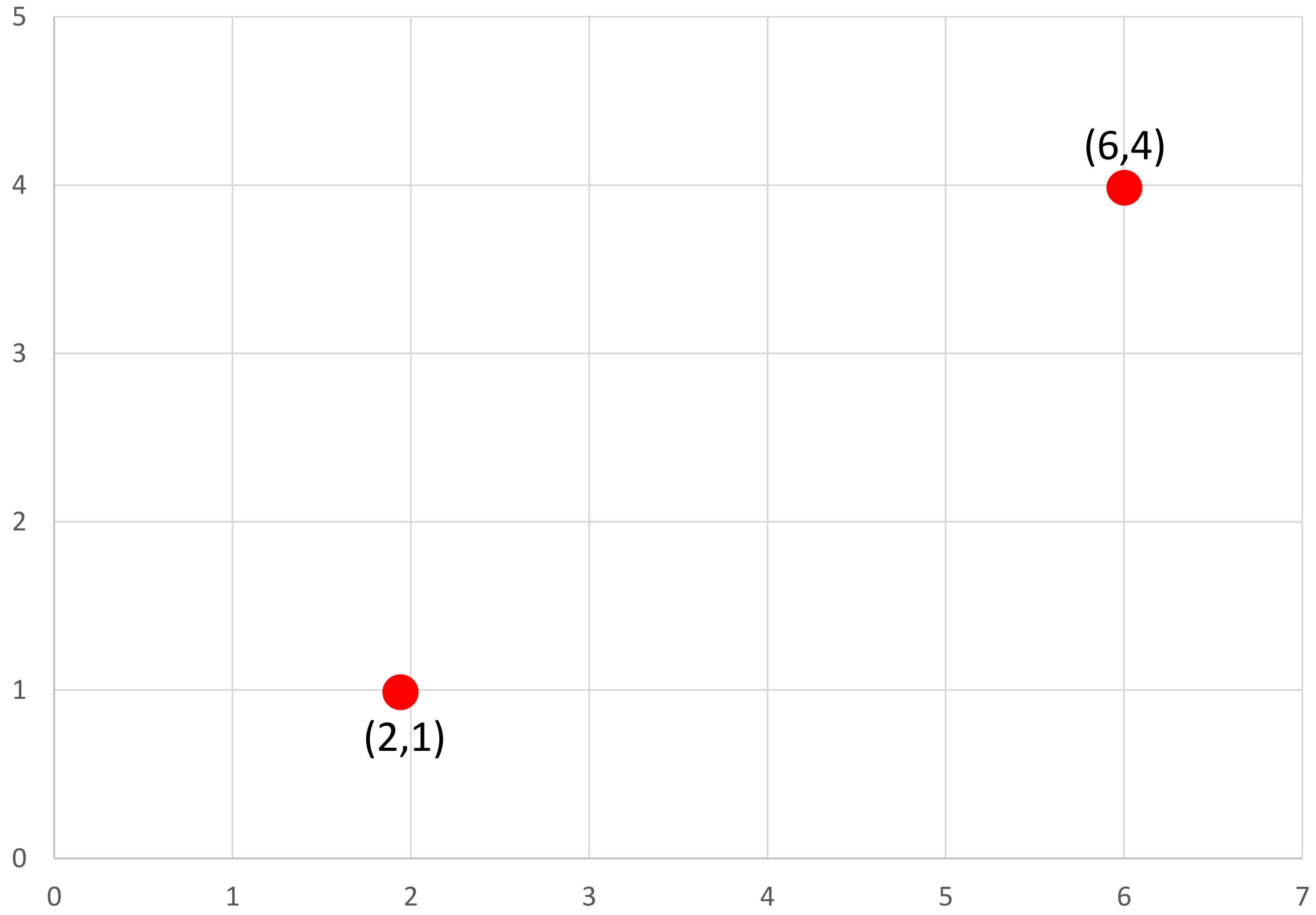
- L-p distance:

$$Lp\left(x^{(i)}, x^{(j)}\right) = \sqrt[p]{\sum_{l=1}^{d} \left| x_l^{(i)} - x_l^{(j)} \right|^p}$$

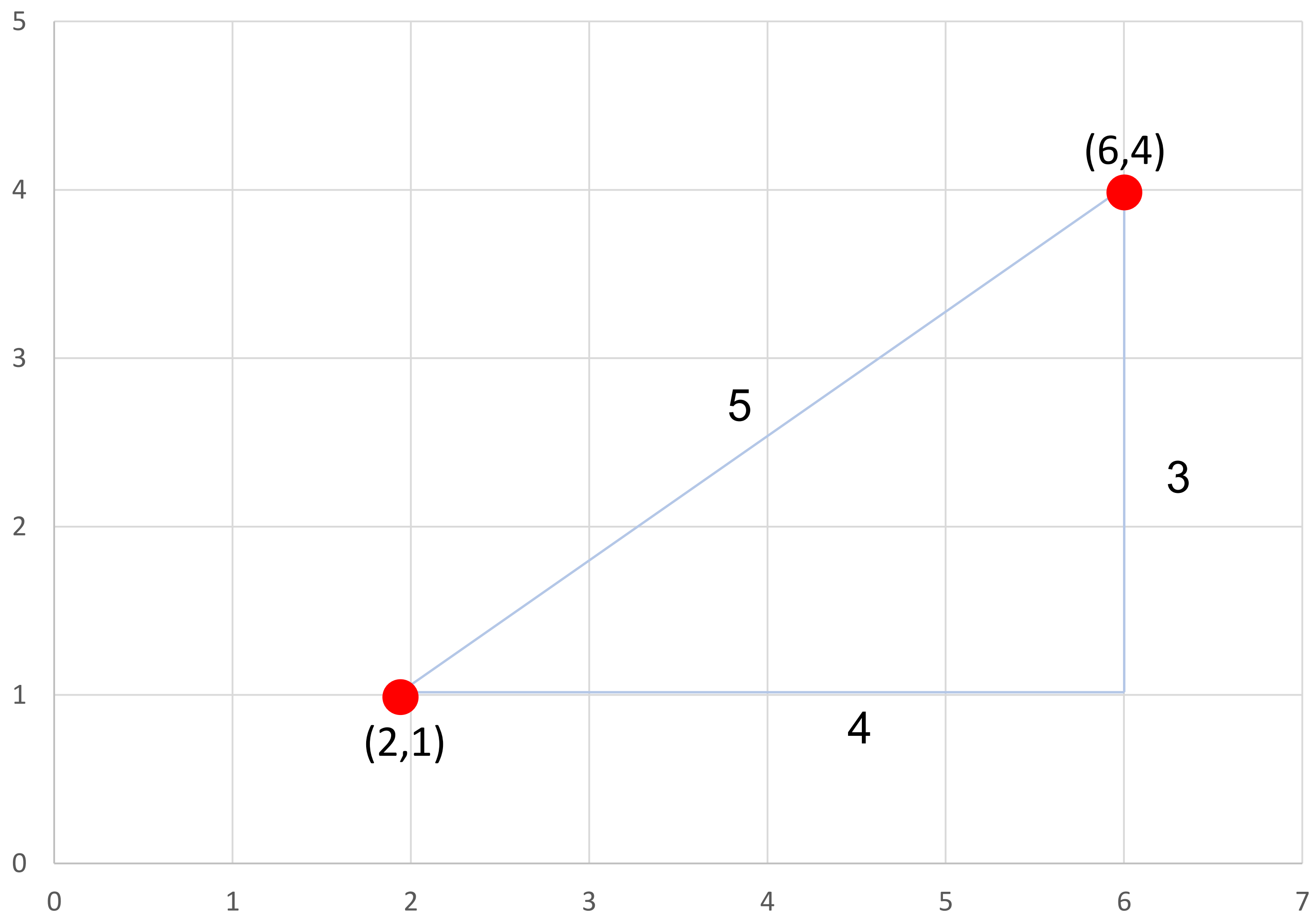
l – the index of the vector dimension

d – the dimension of the vector

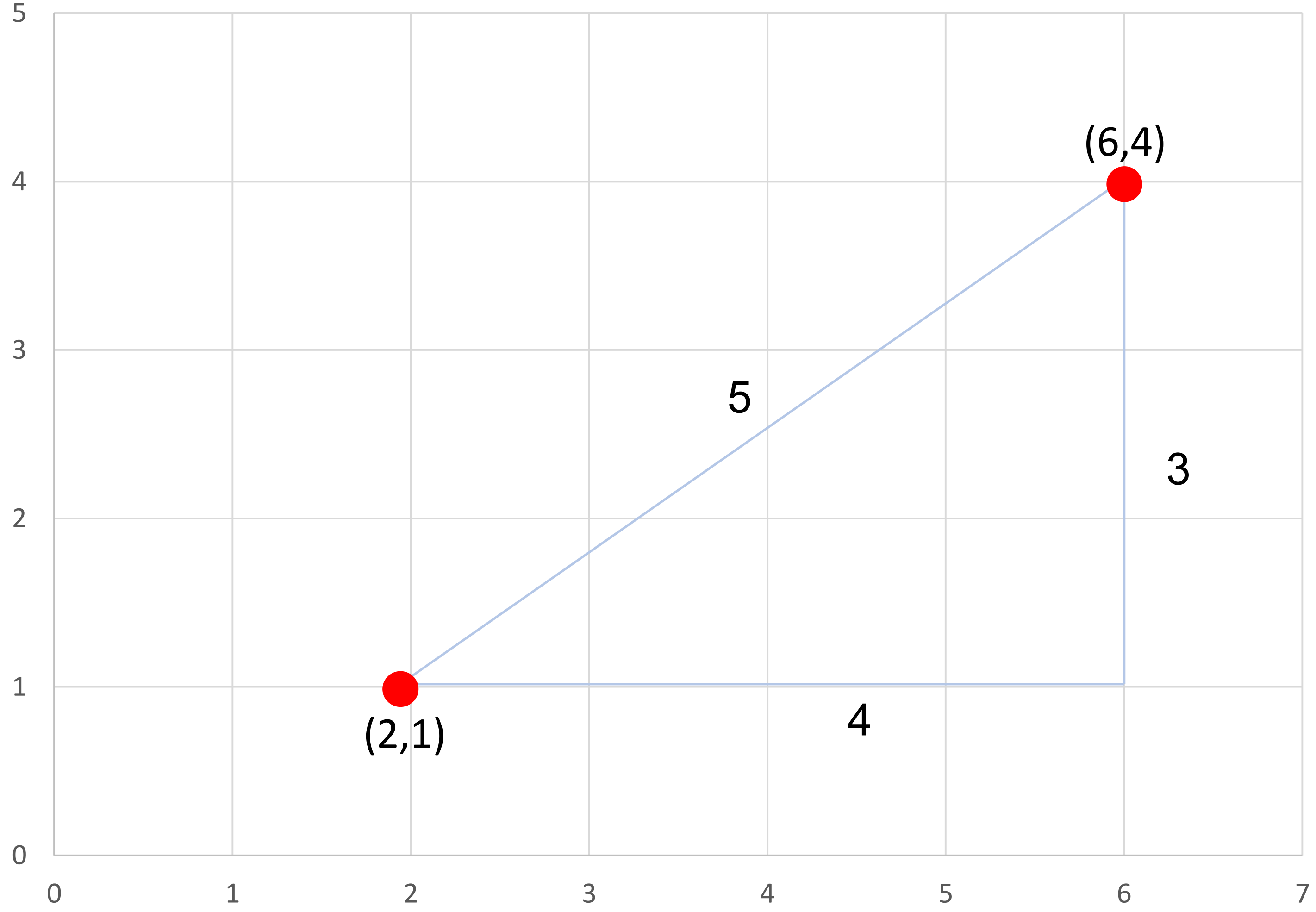
LP distance - Example



LP distance - Example

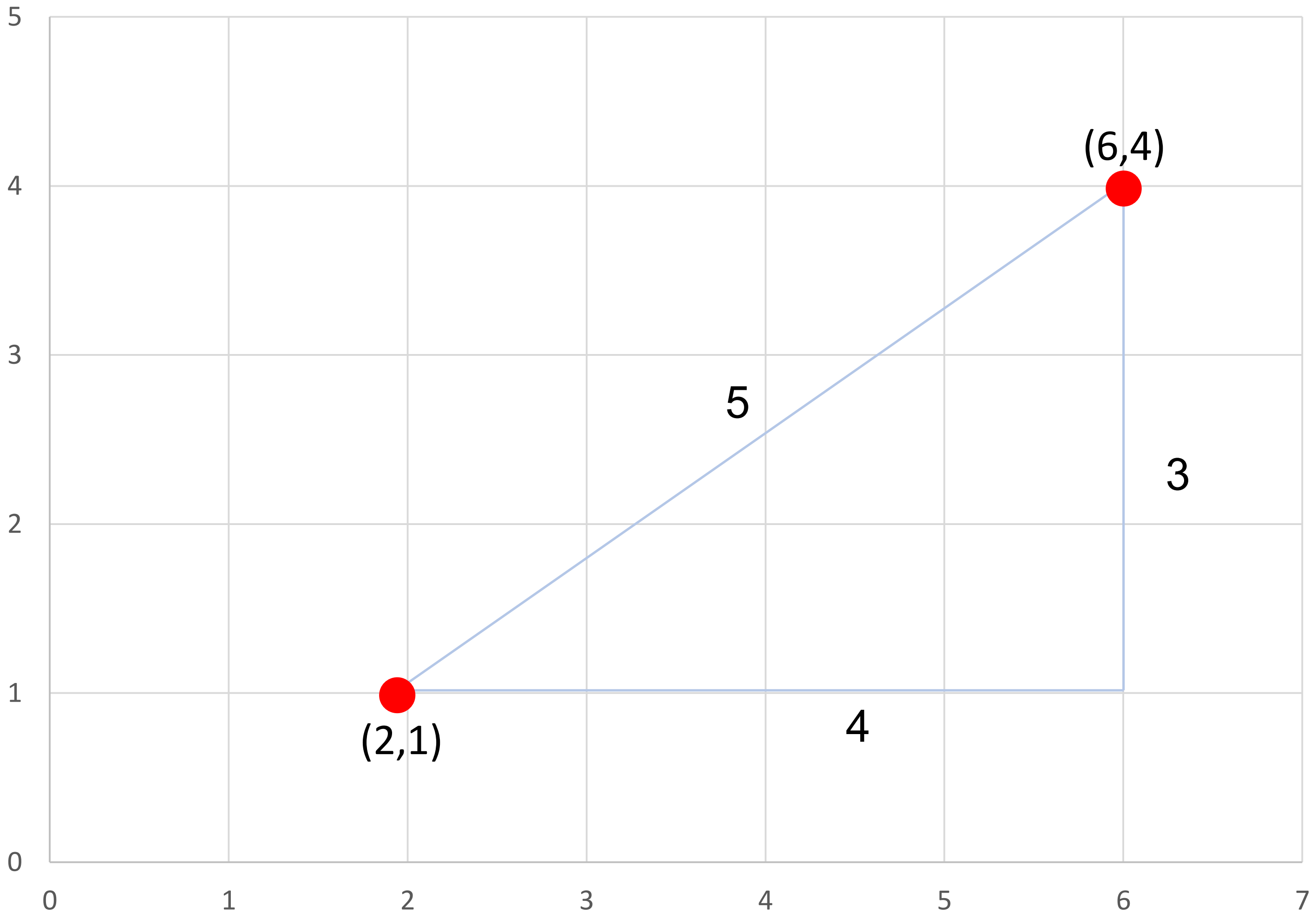


LP distance - Example



L1 distance = 3+4=7

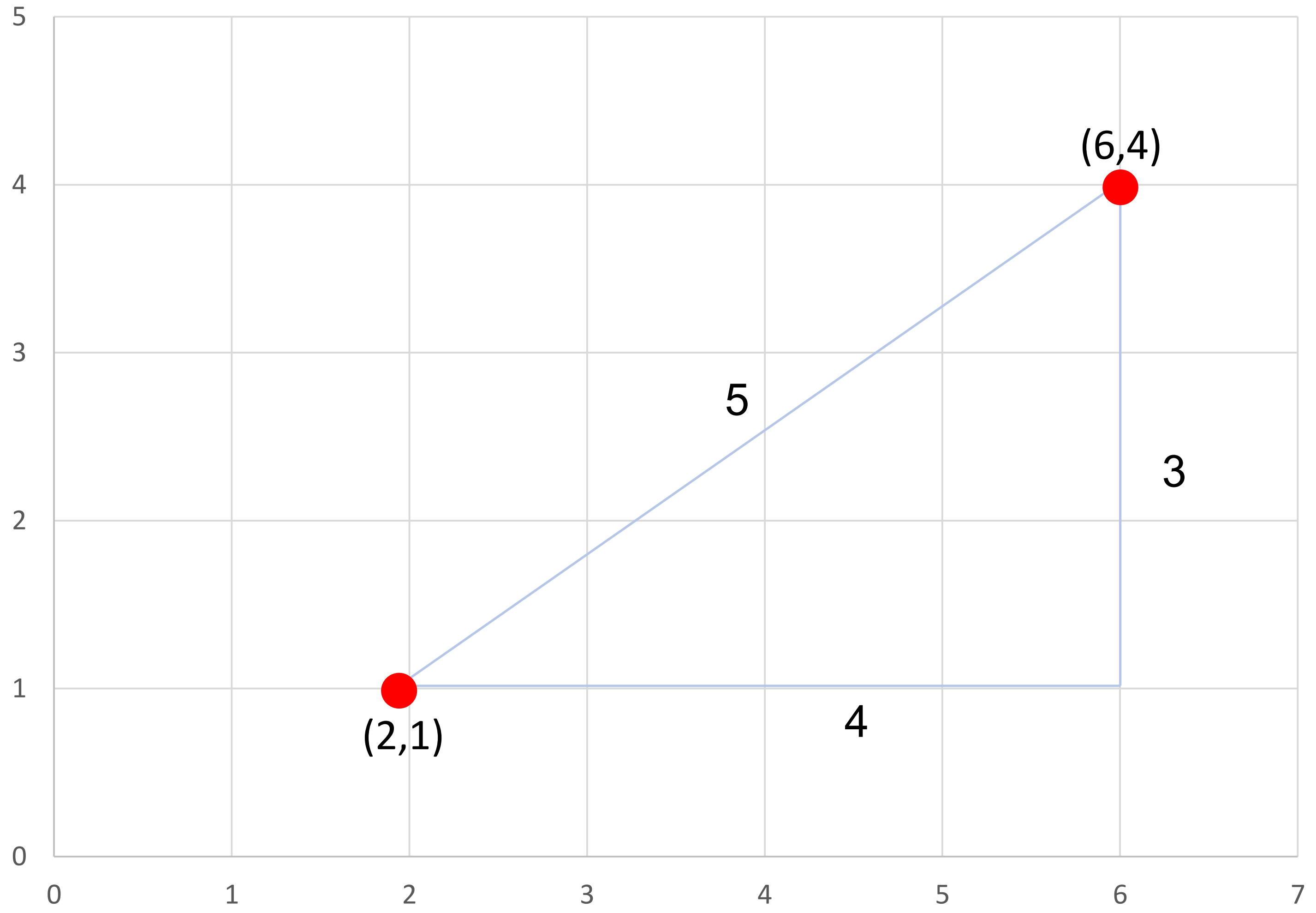
LP distance - Example



L1 distance = 3+4=7

L2 distance = sqrt(16+9)=5

LP distance - Example



L1 distance = 3+4=7

L2 distance = sqrt(16+9)=5

L_inf distance = max(3,4) = 4

Distance For Numeric Features

- When $p = 2$ the Lp distance is called the Euclidean distance
- When $p = 1$ the Lp distance is called the Manhattan distance
- When $p = \infty$ we define this function as follow:

$$L\infty\left(x^{(i)}, x^{(j)}\right) = MAX_l |x_l^{(i)} - x_l^{(j)}|$$

Examples

- $x^{(1)} = (1, 2, 4),\ \ x^{(2)} = (4, 0, 3)$
- When $p = 2$:

$$L2\left(x^{(1)}, x^{(2)}\right) = \sqrt[2]{\sum_{l=1}^{3}\left(x_l^{(1)} - x_l^{(2)}\right)^2} = \sqrt[2]{(-3)^2 + (2)^2 + (1)^2} = \sqrt[2]{14}$$

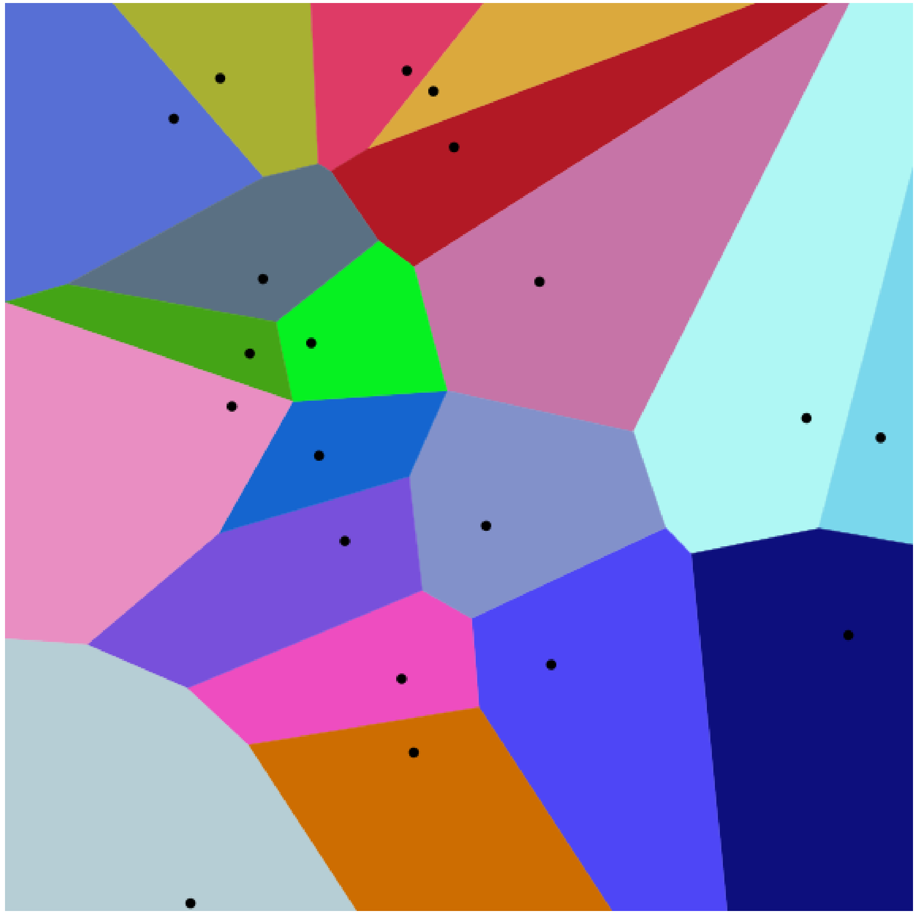
- When $p = 1$:

$$L1\left(x^{(1)}, x^{(2)}\right) = \sum_{l=1}^{3}\left|x_l^{(1)} - x_l^{(2)}\right| = 3 + 2 + 1 = 6$$

- When $p = \infty$:

$$L\infty\left(x^{(1)}, x^{(2)}\right) = MAX(|-3|, |2|, |1|) = 3$$

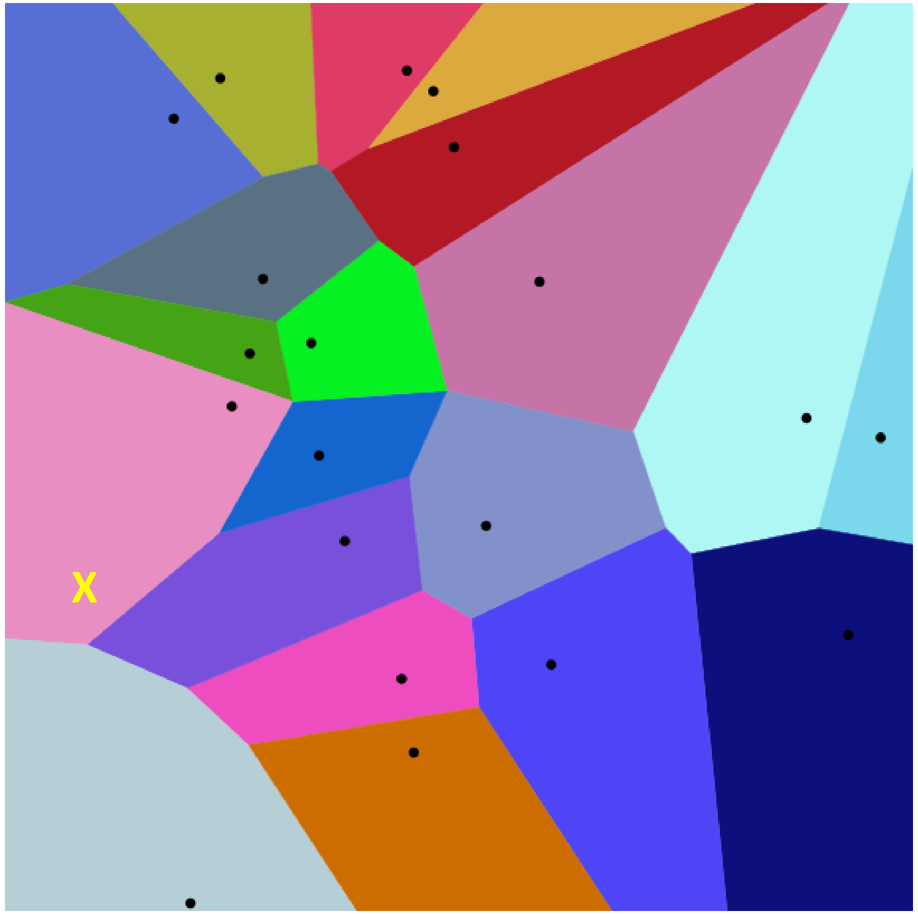
Examples – voronoi diagram



Euclidean distance

Manhattan distance

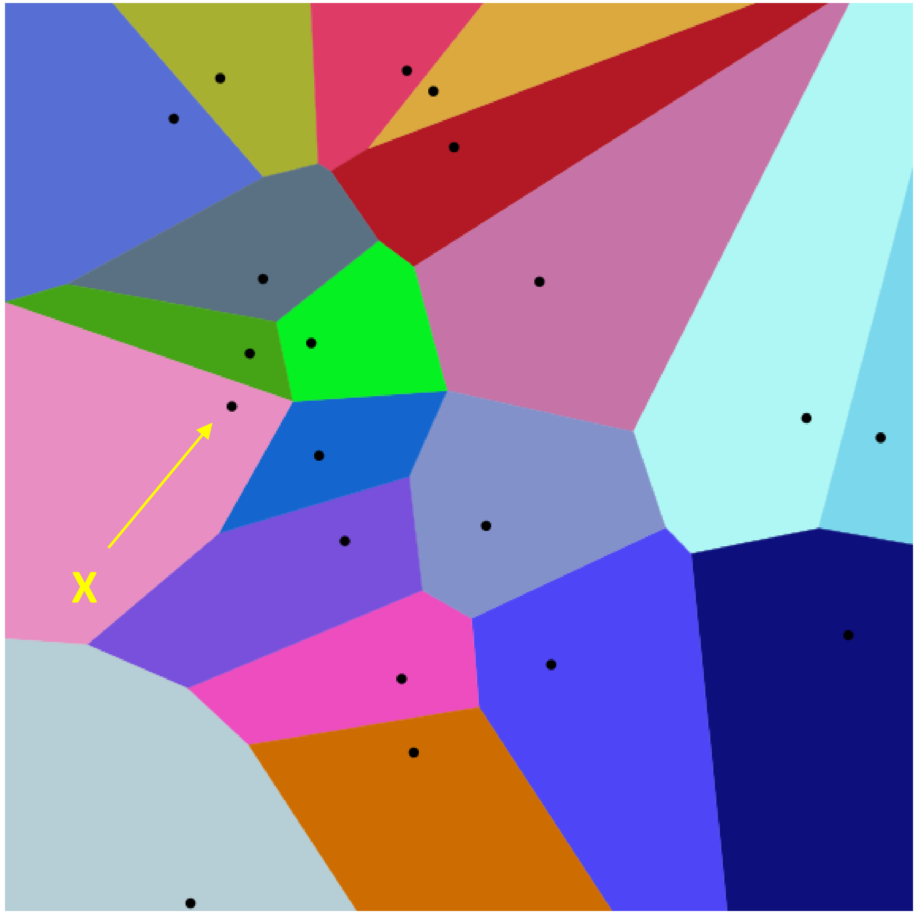
Examples – voronoi diagram



Euclidean distance

Manhattan distance

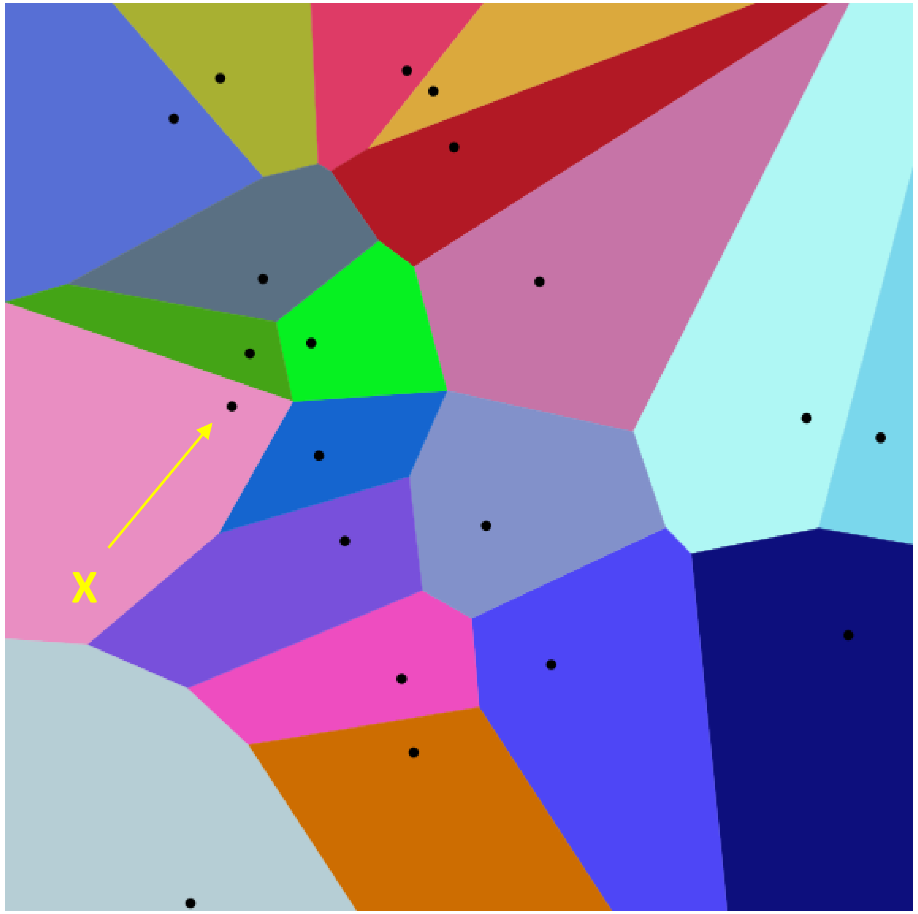
Examples – voronoi diagram



Euclidean distance

Manhattan distance

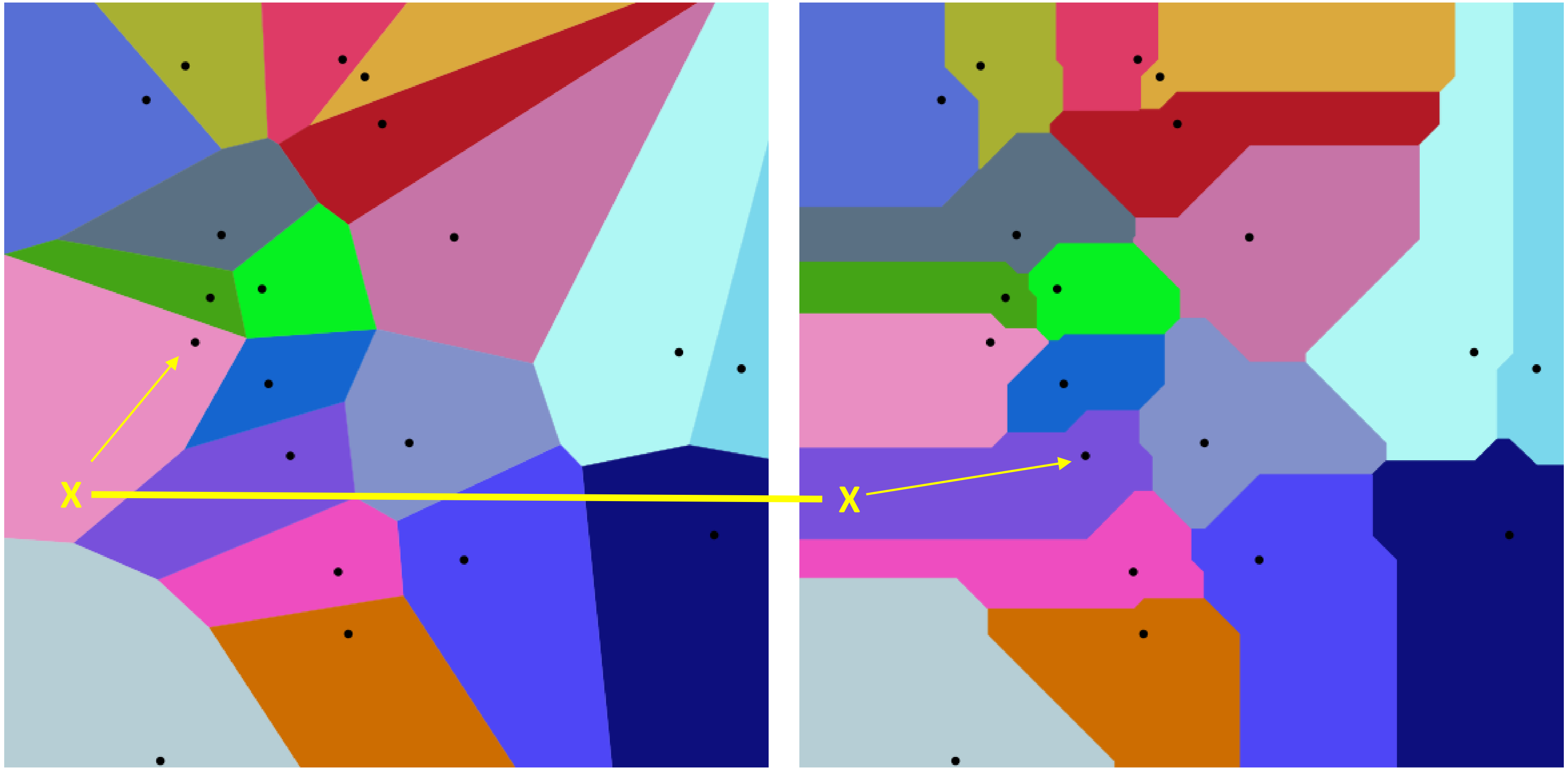
Examples – voronoi diagram



Euclidean distance

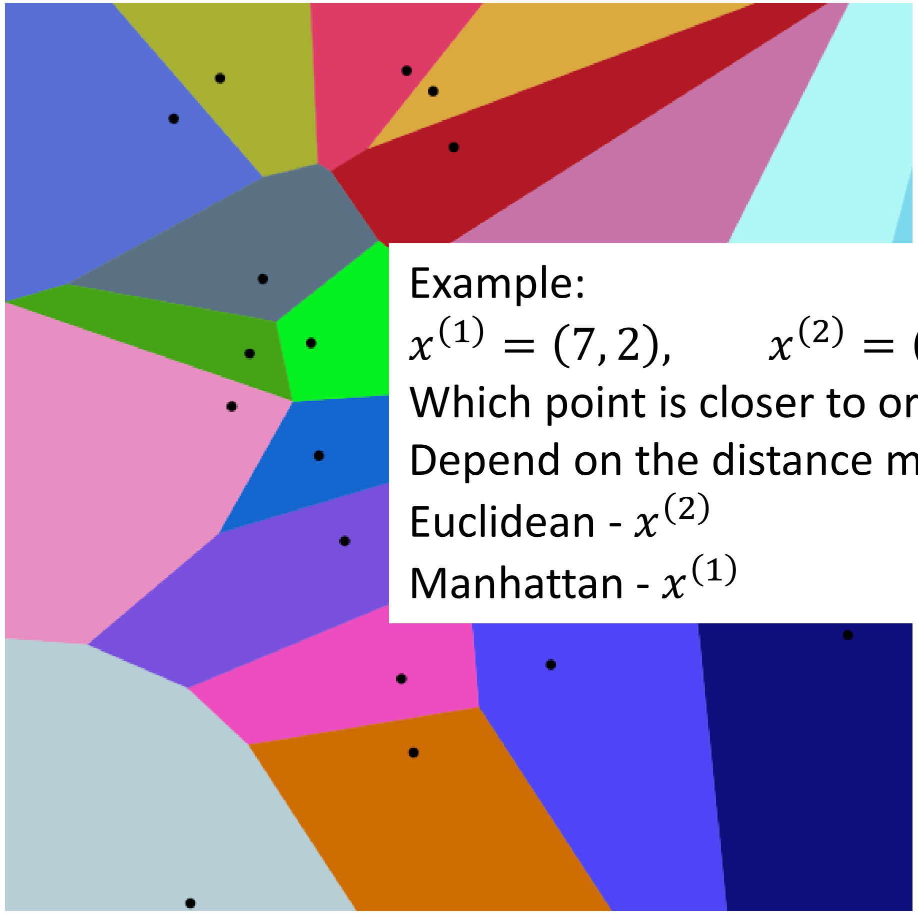
Manhattan distance

Examples – voronoi diagram



Euclidean distance Manhattan distance

Examples – voronoi diagram



Example:

$x^{(1)} = (7, 2), \qquad x^{(2)} = (5, 5)$

Which point is closer to origin?

Depend on the distance method:

Euclidean - $x^{(2)}$

Manhattan - $x^{(1)}$

Euclidean distance

Manhattan distance

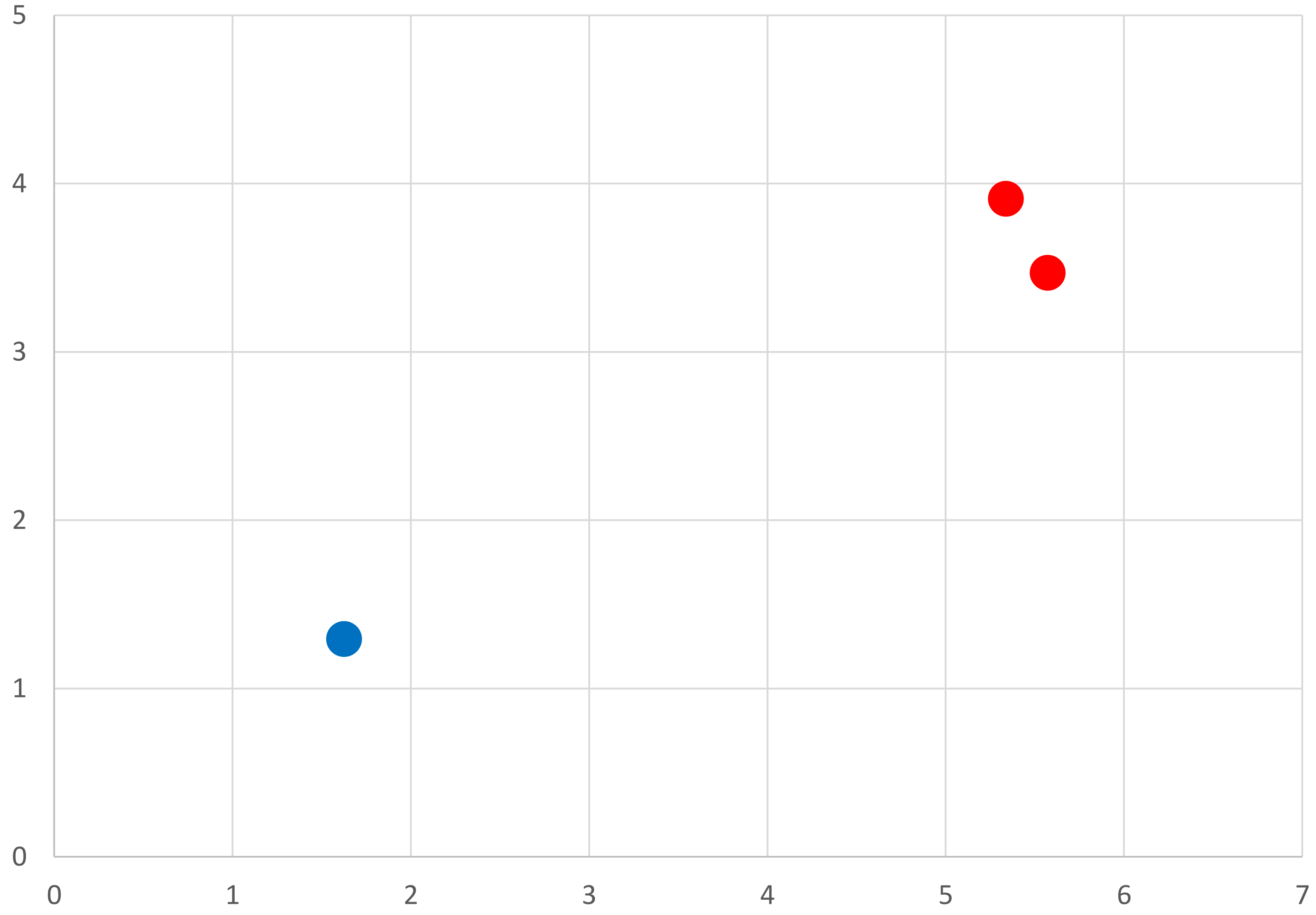
What about non Numeric?

- How do you measure the distance between Blue, Green and Red?
  - First convert to numeric, then measure the distance
  - Use other methods – Hamming, Value Difference Measure, etc…

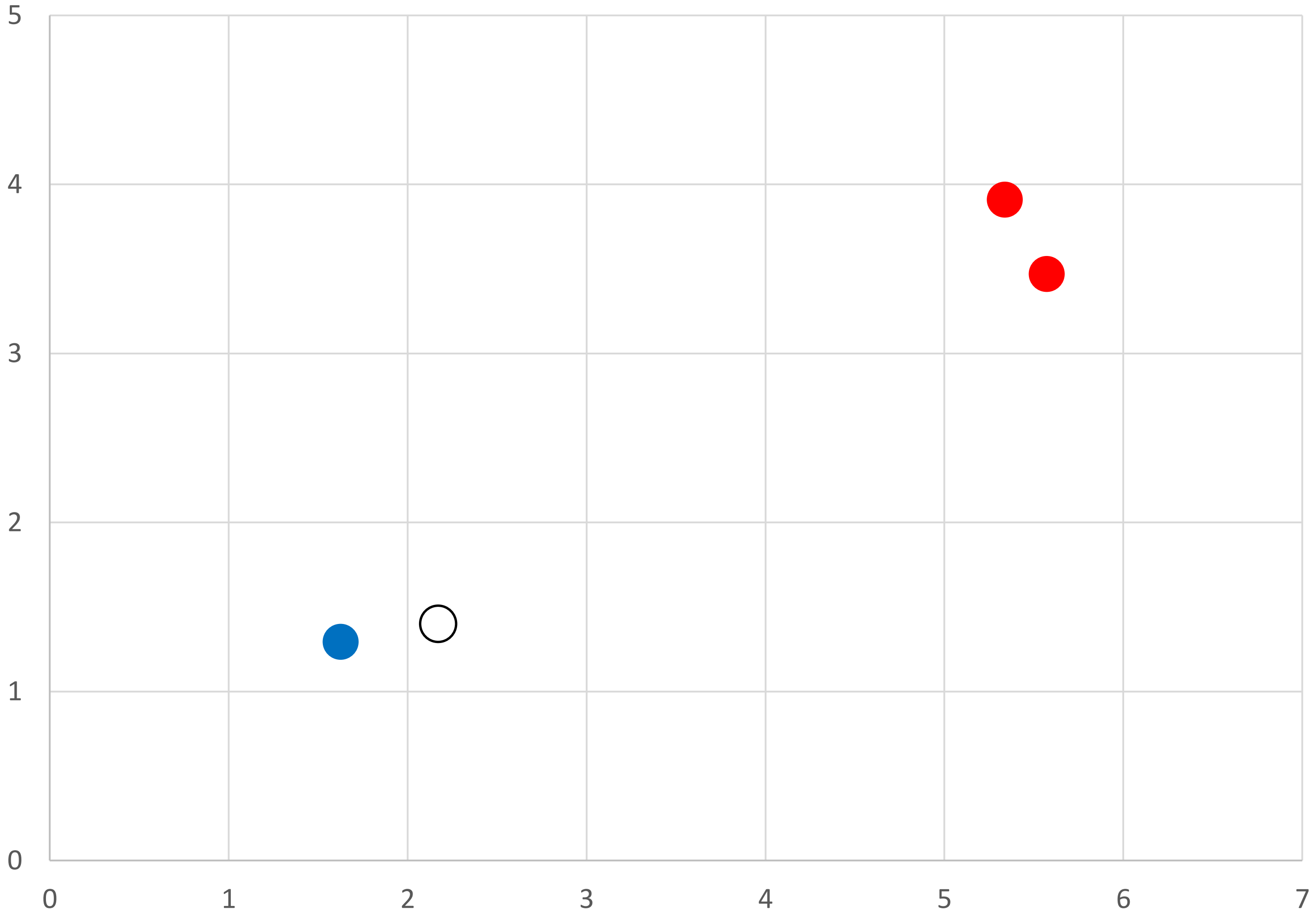
Hamming distance

- The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different:
  - "roses" and "toned" is 3
  - "karolin" and "kerstin" is 3
  - 1011101 and 1001001 is 2
  - 2143896 and 2233796 is 3

Weighted kNN - motivation

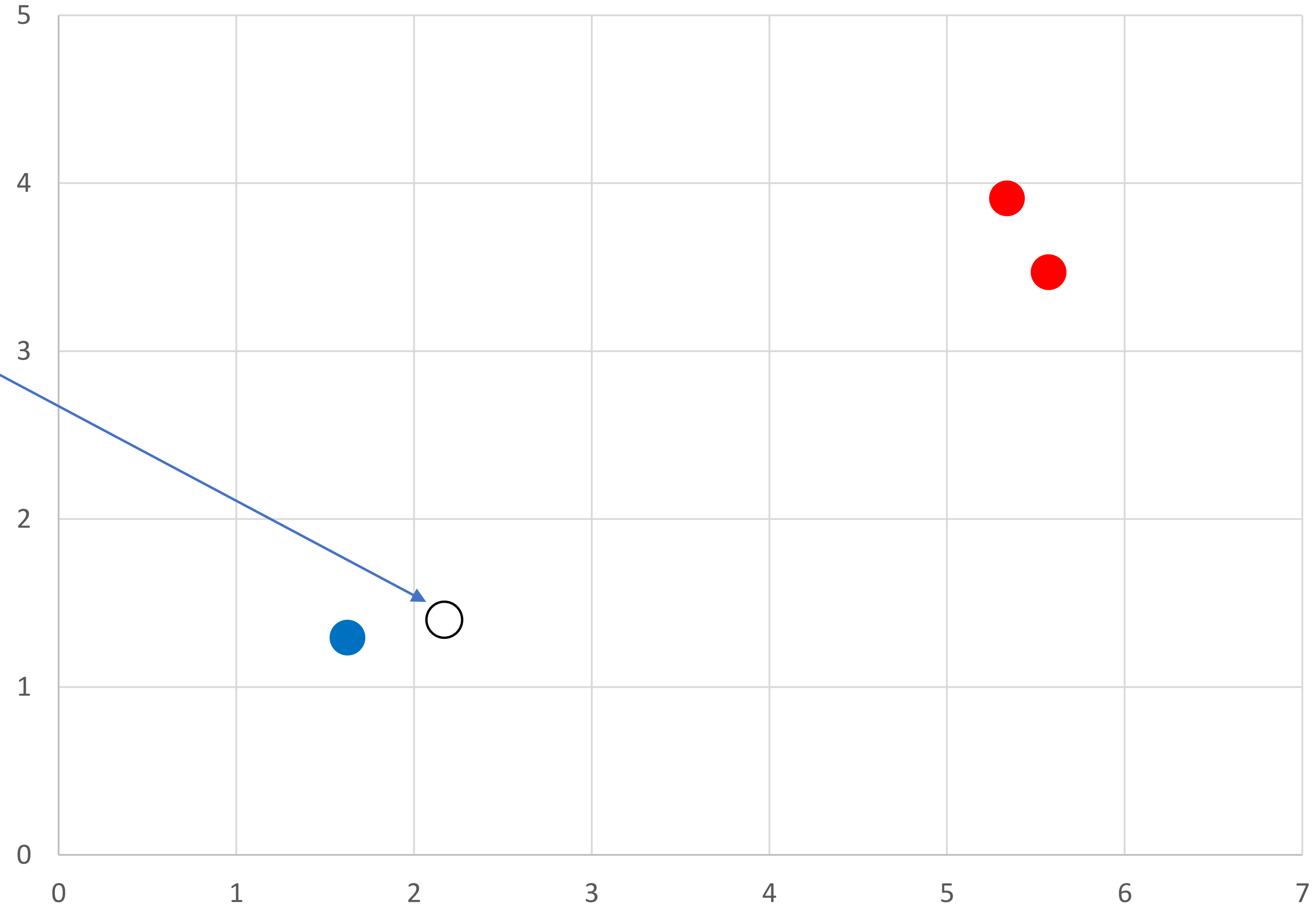


Weighted kNN - motivation



Weighted kNN - motivation

3NN: predicts "red"
We prefer: "blue"



Weighted kNN

- Is the neighbor with distance 5 has the same contribute like the neighbor with distance 10?
- We need a way to give the closer neighbors more weight
- This is called weighted kNN
- What is the simplest way to do it?
  - Divide the neighbor class in the distance
  - Now calculate the majority

Distance weighted Knn

- For Regression/continuous attributes instead of doing:

$$\hat{f}(x) = \frac{1}{k}\sum_{i=1}^{k} f(x^{(i)})$$

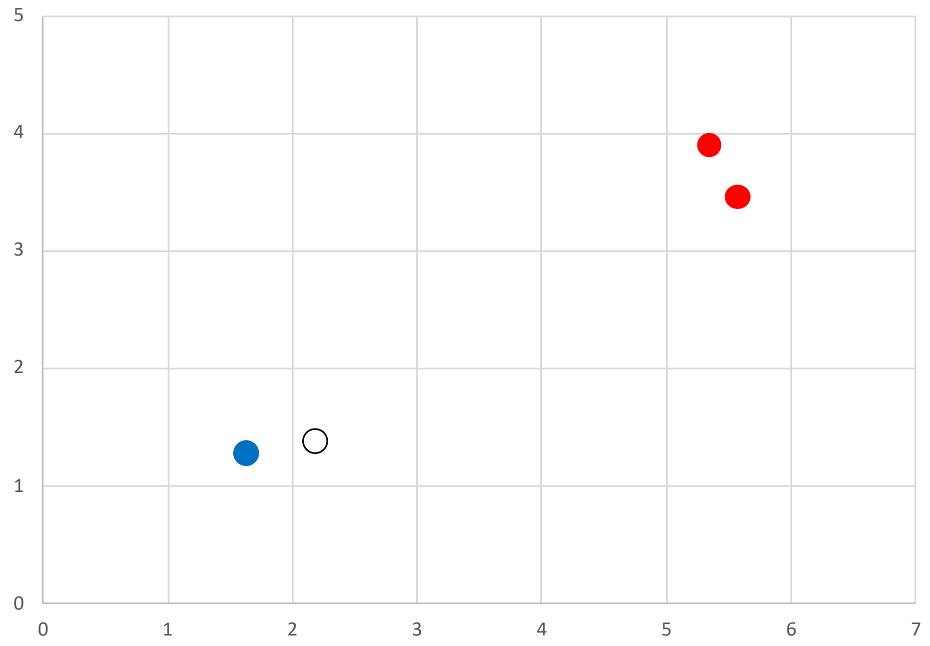
- We can do:

$$\hat{f}(x) = \frac{\sum_{i=1}^{k} w_i f\left(x^{(i)}\right)}{\sum_{i=1}^{k} w_i}$$

Where $w_i = \frac{1}{distance\left(x^{(i)},x\right)}$

Example

- K=3
- The 3 nearest neighbors:
  - X1 distance = 5, class = No
  - X2 distance = 2, class = Yes
  - X3 distance = 5, class = No
- Regular kNN will output ‘No’
- The weighted:
  - $MAJ\left(\frac{No}{5}, \frac{Yes}{2}, \frac{No}{5}\right) =$ ‘Yes’



Question

- How to find nearest? ? ***What is “near”?*** ~~V~~
  - We know the possible methods, but which one to choose?
- Slow query & Large space X
- How to choose k? X

Question

- How to find nearest? ? ***What is "near"?*** ~~V~~
  - We know the possible methods, but which one to choose?
- Slow query & Large space X
- How to choose k? X

Improving Efficiency

- Compute time

$$T_{predict\ sample} = N_{samples} * T_{compute\ distance}$$

- We want to reduce query time and space:
  - Reduce $N_{samples}$
    - Reducing search time using search structures – K-D Tree
    - Reducing number of points by filtering
  - Reducing distance calculation time
    - Interrupt calculation
    - Reduce number of features (feature selection)

Improving Efficiency

- Compute time

$$T_{predict\ sample} = N_{samples} * T_{compute\ distance}$$

- We want to reduce query time and space:
  - Reduce $N_{samples}$
    - Reducing search time using search structures – K-D Tree
    - Reducing number of points by filtering
  - Reducing distance calculation time
    - Interrupt calculation
    - Reduce number of features (feature selection)

A word on Curse of Dimensionality

- How many features do we want to consider in our algorithm – why not all:
  - The required number of samples grows exponentially with the number of variables
  - The relevant information is store in few features
  - In high dimension all instances are far from each other – this is bad for kNN
- In practice, beyond a certain point, the inclusion of additional features leads to worse rather than better performance!

Improving Efficiency

- Compute time

$$T_{predict\ sample} = N_{samples} * T_{compute\ distance}$$

- We want to reduce query time and space:
  - Reduce $N_{samples}$
    - Reducing search time using search structures – K-D Tree
    - Reducing number of points by filtering
  - Reducing distance calculation time
    - Interrupt calculation
    - Reduce number of features (feature selection)

Efficiency – Reducing distance calculation time

- We want less calculation on far (not relevant) instances, and full calculation on close instances
- Our distance built from sum of distances
- We will stop if the current sum is greater than some threshold

Efficiency – Reducing distance calculation time

- Example:
- $x^{(1)} = (1,2,4),\ x^{(2)} = (4,0,3),\ \ x^{(3)} = (10,0,3)$
- We want the nearest neighbor, where $x^{(1)}$ is the query instance – we choose 10 to be our threshold (l2-distance)
- How the computation look like?

$$Lp\left(x^{(i)}, x^{(j)}\right) = \sqrt[p]{\sum_{l=1}^{d} \left|x_l^{(i)} - x_l^{(j)}\right|^p}$$

Improving Efficiency

- Compute time

$$T_{predict\ sample} = N_{samples} * T_{compute\ distance}$$

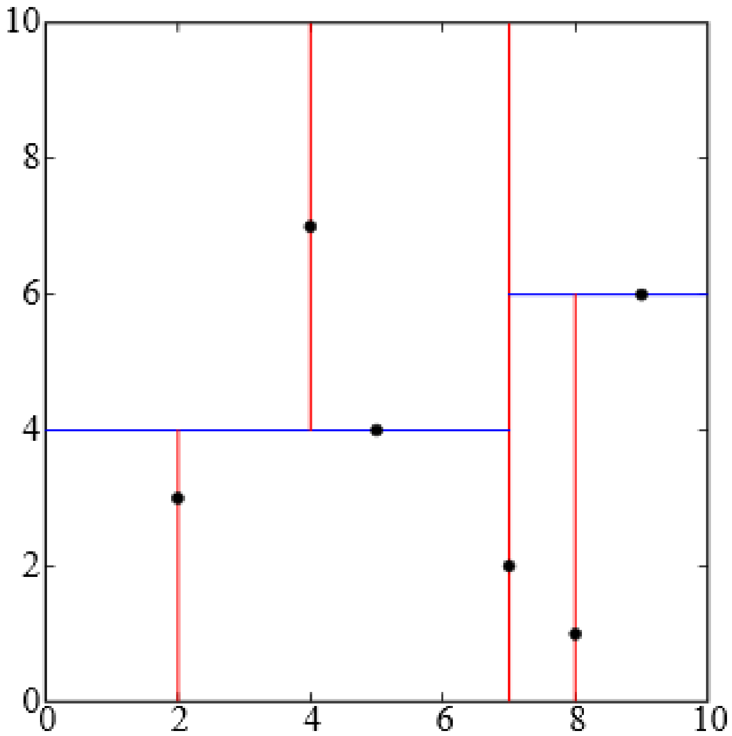
- We want to reduce query time and space:
  - Reduce $N_{samples}$
    - Reducing search time using search structures – K-D Tree
    - Reducing number of points by filtering
  - Reducing distance calculation time
    - Interrupt calculation
    - Reduce number of features (feature selection)

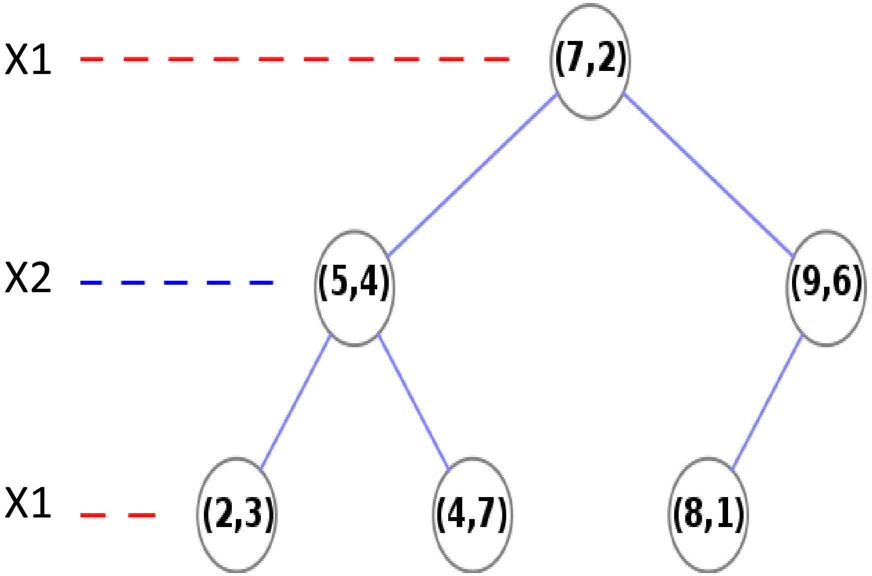
Efficiency – K-D Tree

- Instead of search the nearest neighbor on all training data we will construct an efficient search structure
- We divide the data to partitions, each time in different dimension
- The search for the neighbors, first will find the relevant partition and then will search only in this partition

Efficiency – K-D Tree

- Example:
  - Points set: (2,3), (5,4), (9,6), (4,7), (8,1), (7,2)





Improving Efficiency

- Compute time

$$T_{predict\ sample} = N_{samples} * T_{compute\ distance}$$

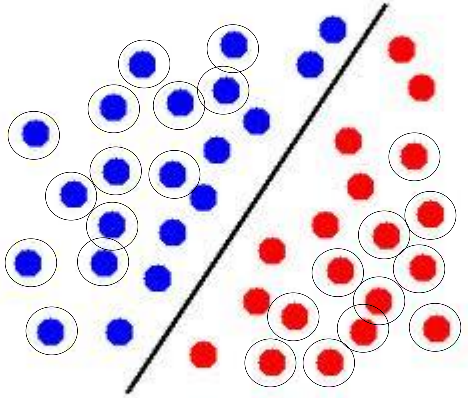
- We want to reduce query time and space:
  - Reduce $N_{samples}$
    - Reducing search time using search structures – K-D Tree
    - $\boxed{\text{Reducing number of points by filtering}}$
  - Reducing distance calculation time
    - Interrupt calculation
    - Reduce number of features (feature selection)

Efficiency - Reducing number of points by filtering

- Goal:

  remove points from the training set that don't effect the boundary

- **Forward:** insert training set points one by one but keep only those that are not classified correctly



- **Backward:** accept all points in the training set and then go through the points and remove those that are correctly classified by their (KNN) neighbors

Note: order dependent (**Greedy!**)

Efficiency - Reducing number of points by filtering

- This procedure called Edited kNN

- **Backward KNN**(S)

```
T = S
For each instance x in T
        if x is classified correctly by T-{x}
                remove x from T
Return T
```

- **Forward KNN**(S)

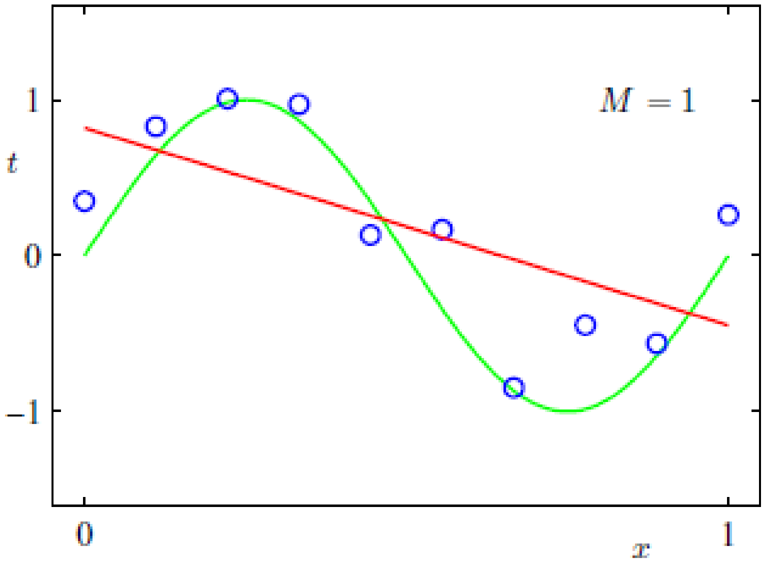
```
T = Ø
For each instance x in S
        if x is not classified correctly by T
                add x to T
Return T
```

Open question

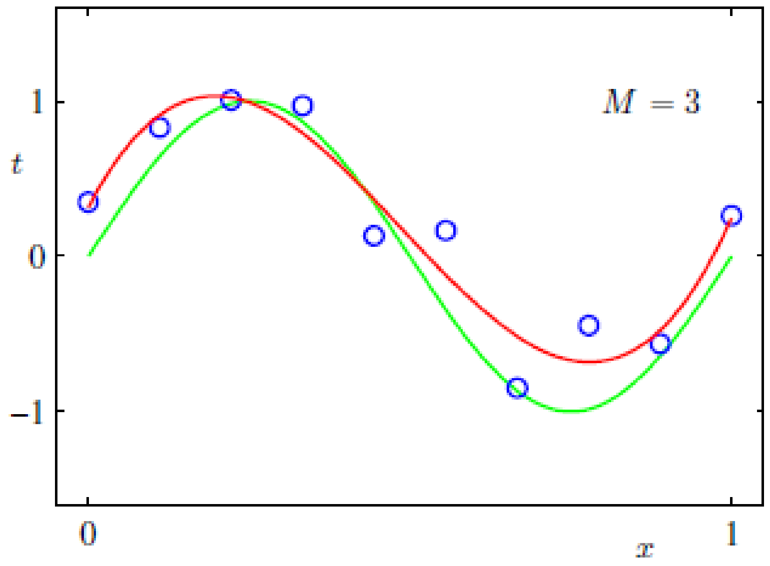
- How to find nearest? ~~√~~
  - We know the possible methods, but which one to choose?
- Slow query & Large space √
  - We now able to reduce space (irrelevant points) & accelerate query time (K-D tree, reducing calculation time)
- How to choose k? X

Overfitting

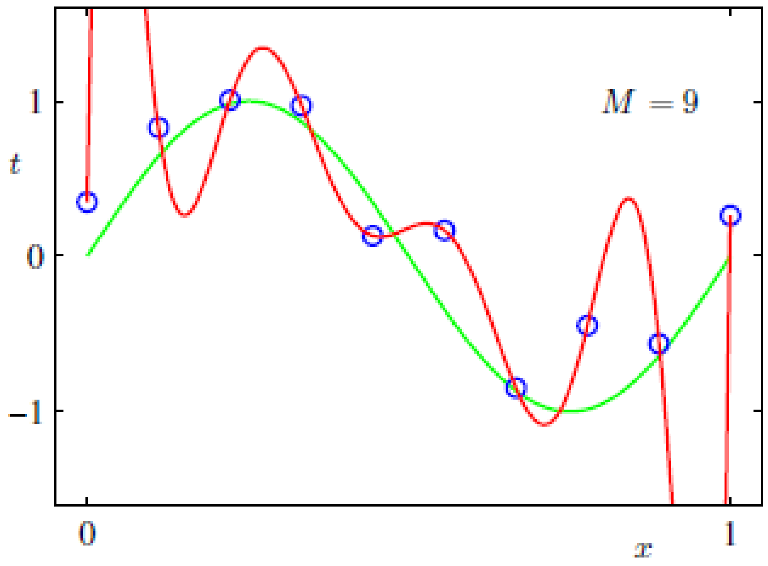
- Recall: polynomial regression



Too simple



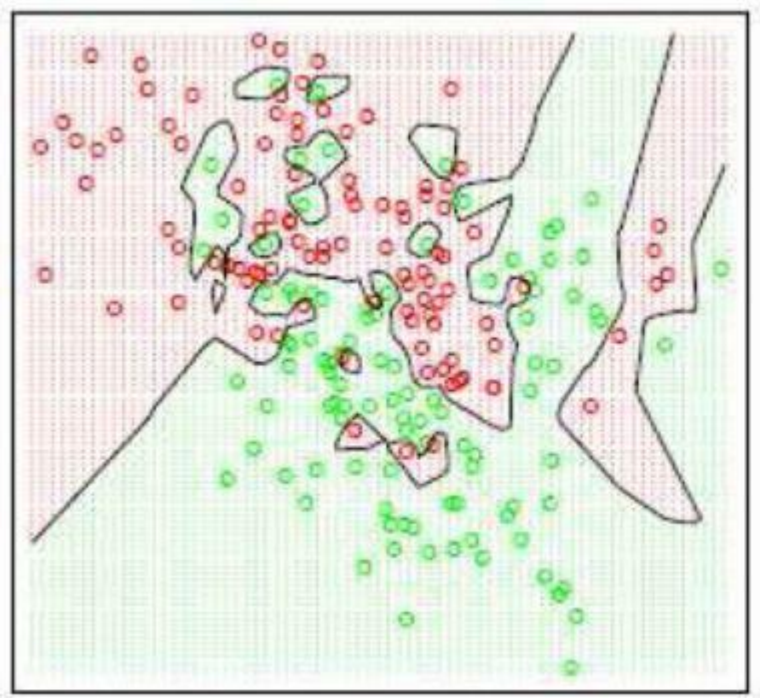
Just right



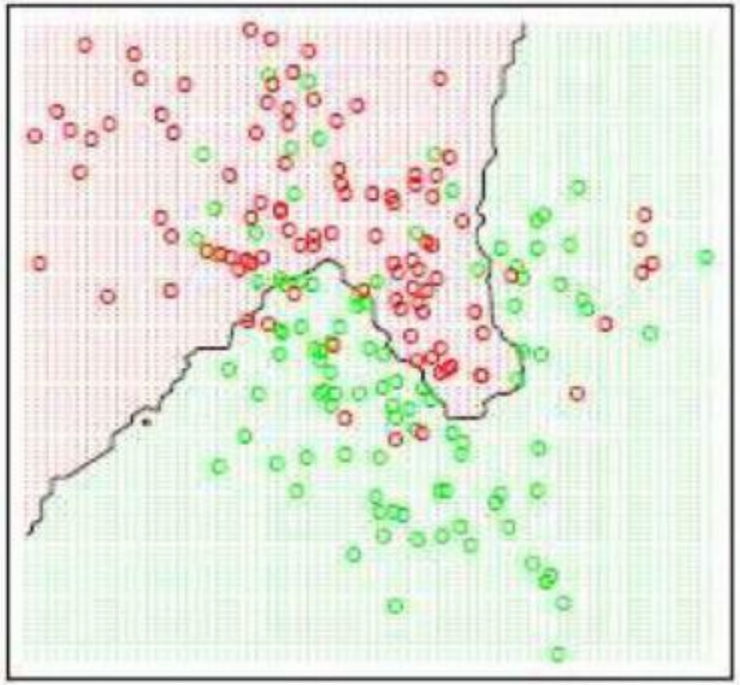
Too complex (“memorize”)

Overfitting

K=1

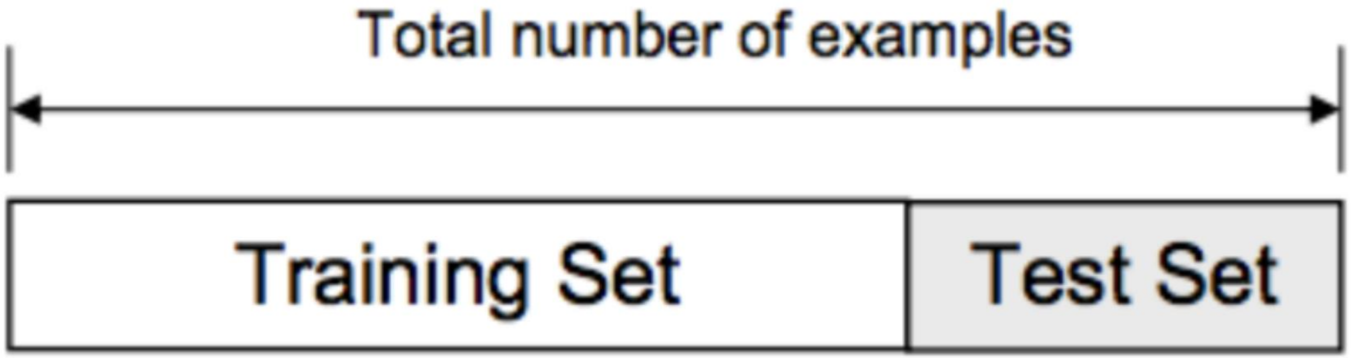


K=15



Dataset splitting & Cross Validation

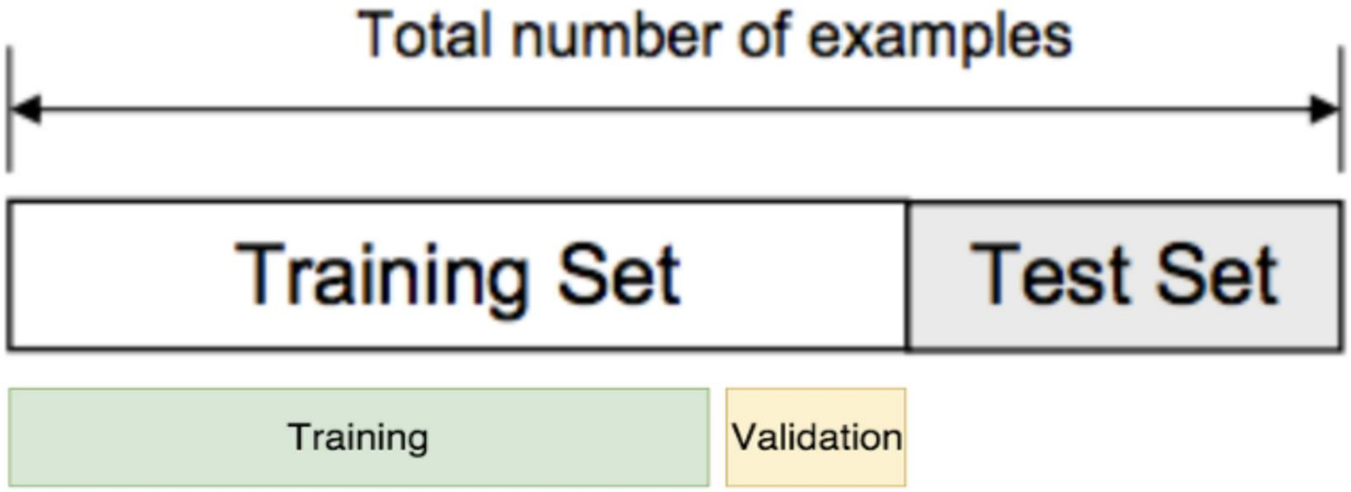
- When we’re using a statistical model (like linear regression, for example), we fit the model on a training set in order to make predications on a data that wasn’t trained (general data)
- In order to do that we need to split the data to training and test sets



- Is this enough?

Dataset splitting & Cross Validation

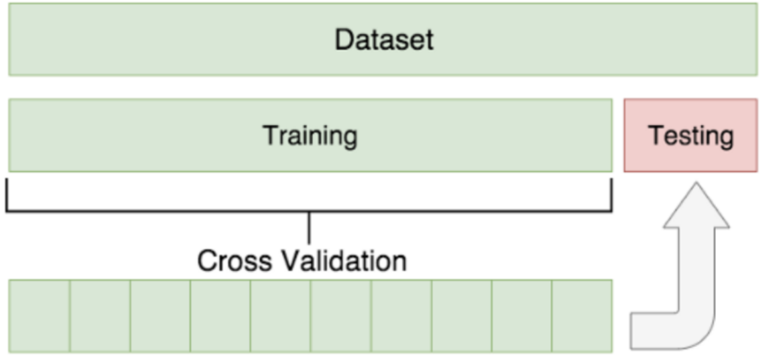
- How do we fine tune our model (choose the best hyper parameters for the model)?
- We can’t use the test set for choosing the hyper parameters - why?
- We need another set – validation set



Dataset splitting & Cross Validation

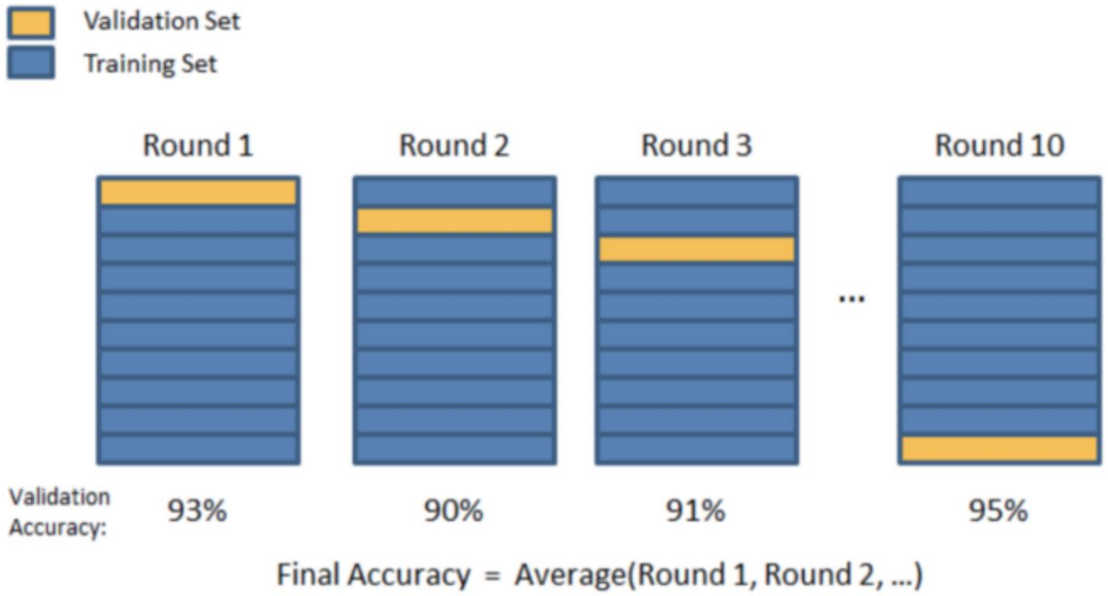
- Splitting the data only once has some drawbacks:
  - If the dataset is “sparse” then we need all the data we can get
  - If we get an unfortunate split then this method might not work (we can reduce the probability for that by shuffling the data)

- The second method for fine tune our model is cross-validation
- It’s very similar to train/val split, but it’s applied to more subsets
- Meaning, we split our data into k subsets, and train on k-1 one of those subset
- What we do is to hold the last subset for test
- We’re able to do it for each of the subsets



Dataset splitting & Cross Validation

- There are two main methods for executing the cross validation:
  - K-folds cross validation:
    - In K-Folds Cross Validation we split our data into k different subsets (or folds)
    - We use k-1 subsets to train our data and leave the last subset (or the last fold) as test data
    - We then average the model against each of the folds and then finalize our model
    - After that we test it against the test set



Dataset splitting & Cross Validation

- There are two main methods for executing the cross validation:
  - Leave One Out cross validation (LOOCV):
    - In this type of cross validation, the number of folds (subsets) equals to the number of observations we have in the dataset
    - We then average ALL of these folds
    - Because we would get a big number of training sets (equals to the number of samples), this method is very computationally expensive and should be used on small datasets
    - If the dataset is big, it would most likely be better to use a different method, like k-fold

Dataset splitting & Cross Validation

- There are two main methods for executing the cross validation:
  - So, what method should we use? How many folds?
  - The more folds we have
    - We will be reducing the error due the bias but increasing the error due to variance
    - The computational price would go up too, obviously — the more folds you have, the longer it would take to compute it and you would need more memory
  - With a lower number of folds
    - we’re reducing the error due to variance, but the error due to bias would be bigger
    - It’s would also computationally cheaper
  - Therefore, in big datasets, k=10 is usually advised
  - In smaller datasets, as I’ve mentioned before, it’s best to use LOOCV

Cross Validation

- For kNN – need to choose
  - K=?
  - P=?
- Cross validation – a method for hyper parameter optimization

parameters	Mean error
K=1, p=2	0.78
K=5, p=2	0.25
K=3, p=1	0.48

Source: http://karlrosaen.com/ml/learning-log/2016-06-20/

Open question

- How to find nearest? √
  - We know the possible methods & we use X-Fold Cross Validation to chose best one?
- Slow query & Large space √
  - We now able to reduce space (irrelevant points) & accelerate query time (K-D tree, reducing calculation time)
- How to choose k? √
  - We use X-Fold Cross Validation to chose best one

Questions

?

Parzen window – from the beginning

- We first want to find a measure for $p(x)$ – at the end we will convert it to $p(x|A_i)$
  - The mathematical definition for pdf, $p(x)$:
    - The probability that x is between 2 points a and b

$$P(a < x < b) = \int_a^b p(x)dx$$

    - The probability is non negative for all real x
    - For all possible x the integral is 1:

$$\int_{-\infty}^{\infty} p(x)dx = 1$$

Parzen window – from the beginning

- If we look at a region $\mathcal{R}$
  - The probability P that x is inside a region $\mathcal{R}$:

$$P = \int_{\mathcal{R}} p(x)dx$$

  - If we assume that $\mathcal{R}$ is so small that $p(x)$ does not vary much within it, we can write:

$$P = \int_{\mathcal{R}} p(x)dx \approx p(x) \int_{\mathcal{R}} dx = p(x)V$$

Where V is the volume of $\mathcal{R}$

Parzen window – from the beginning

- Suppose that n samples are drawn independently according to some pdf $p(x)$
- If there are m out of n samples falling within $\mathcal{R}$, we have:

$$P = \frac{m}{n}$$

- We got that:

$$P = \frac{m}{n} = p(x)V$$

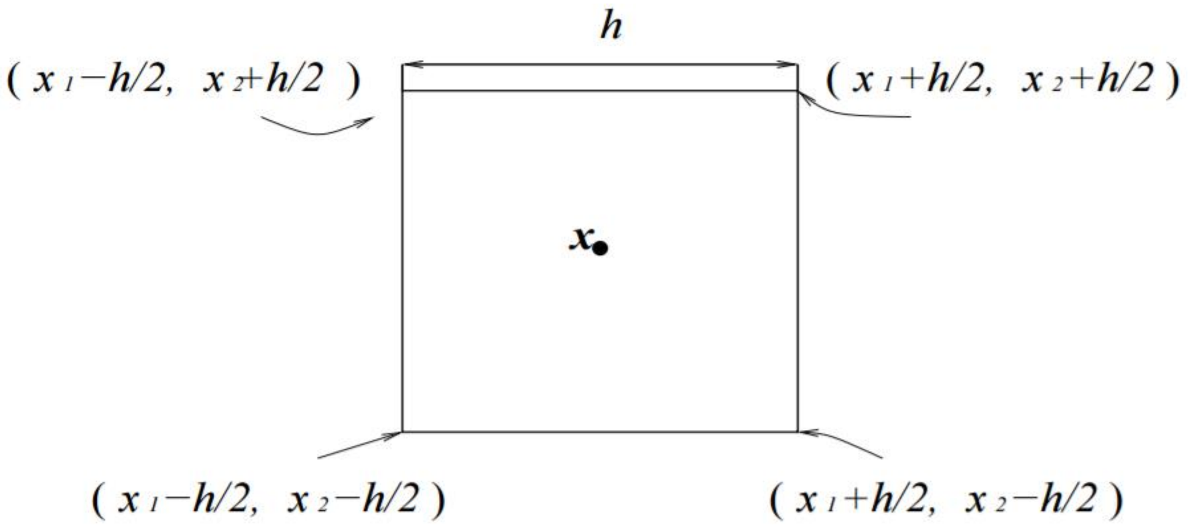
$$p(x) = \frac{m/n}{V}$$

Parzen window – from the beginning

- If $\mathcal{R}$ is hypercube centered at x, and h is the length of the edge of the hypercube we get:

$$V = h^d$$

Where d is the dimension of the hypercube



Parzen window – from the beginning

- We can now define a kernel function:

$$K(u) = \begin{cases} 1, & if\ |u| < \frac{1}{2} \\ 0, & otherwise \end{cases}$$

- And in our case, m , the total number of samples falling within $\mathcal{R}$, out of n samples, is given by:

$$m = \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

Parzen window – from the beginning

- And we finally got the probability of x:

$$p(x) = \frac{m/n}{V} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h^d}K\left(\frac{x - x_i}{h}\right)$$

- We can change the kernel (window) function to yield other parzen window density estimation methods

Parzen window – classification

- Now that we can estimate the instance probability, we can use the likelihood, $p(x|A_i)$, to classify:

$$p(\vec{x}_{new}|A_i) = \frac{1}{n_i}\sum_{\vec{x}\in A_i}\frac{1}{h^d}K\left(\frac{\vec{x}_{new}-\vec{x}}{h}\right)$$