

Recap

- Classifiers are better understood geometrically
- Decision boundary for most $w^T x + b = 0$
- Non linear classifiers use kernels or transform space
- First primitive classifier – Nearest Centroid
- View clustering geometrically

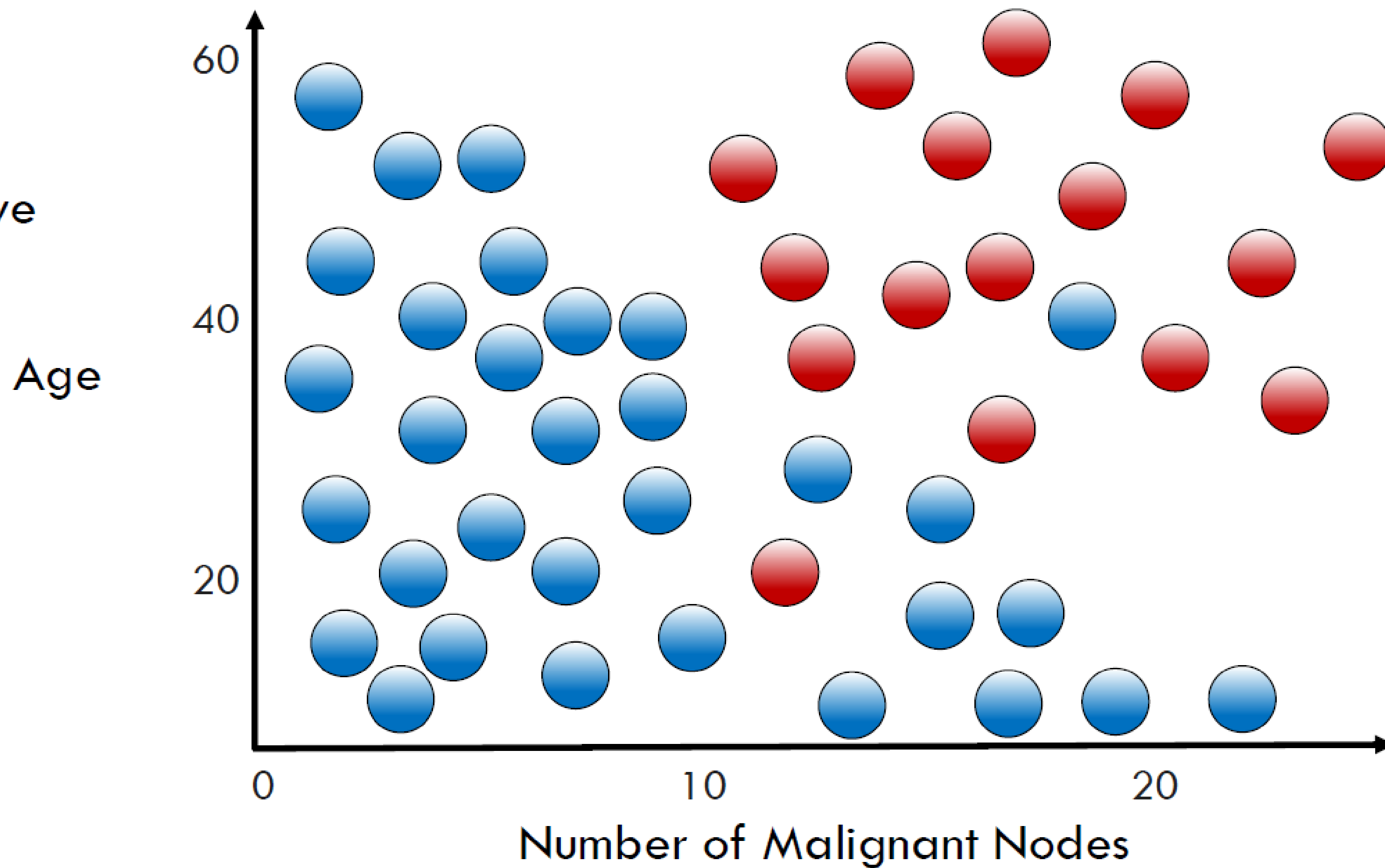


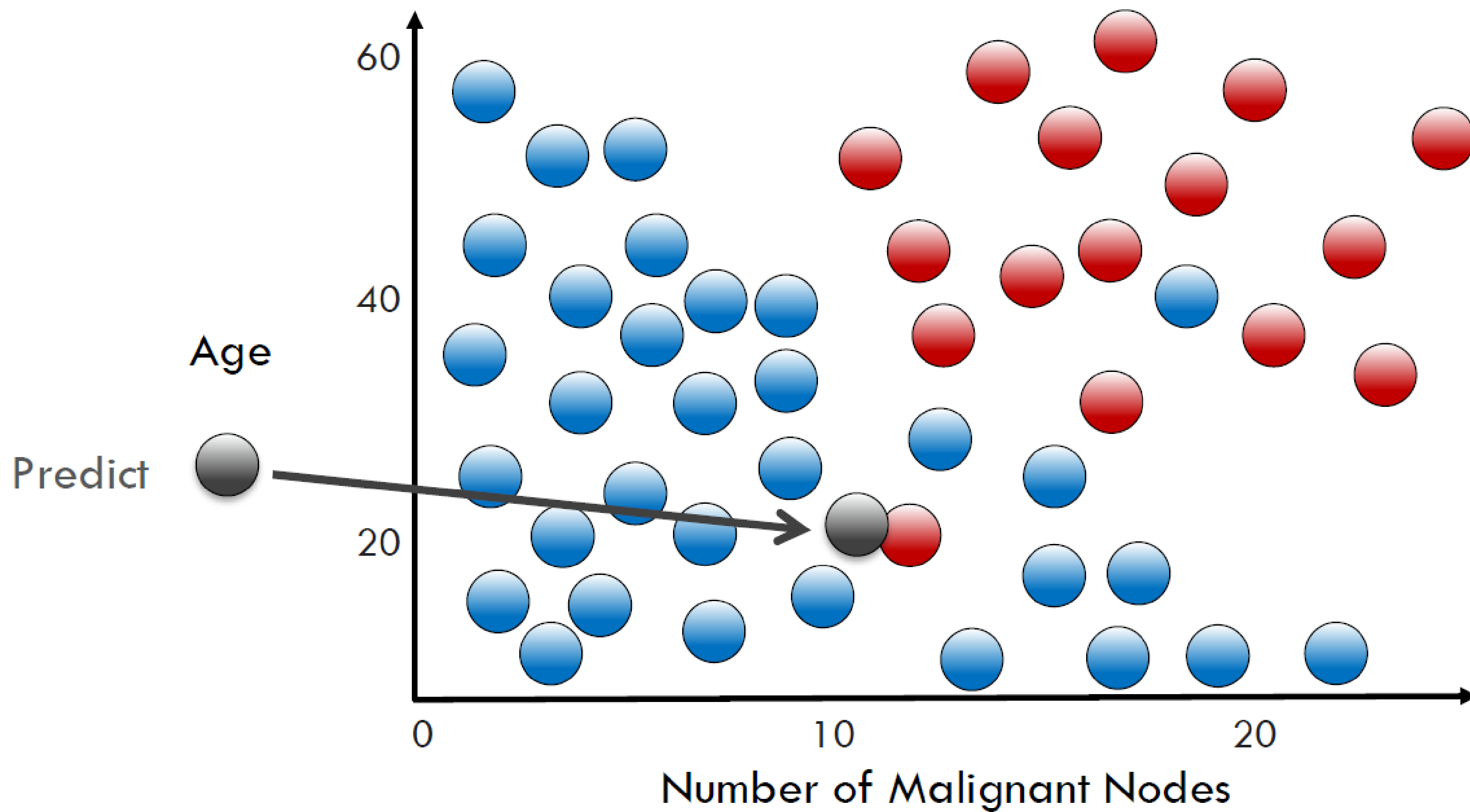
LOVE YOUR
NEIGHBOR



K Nearest Neighbors

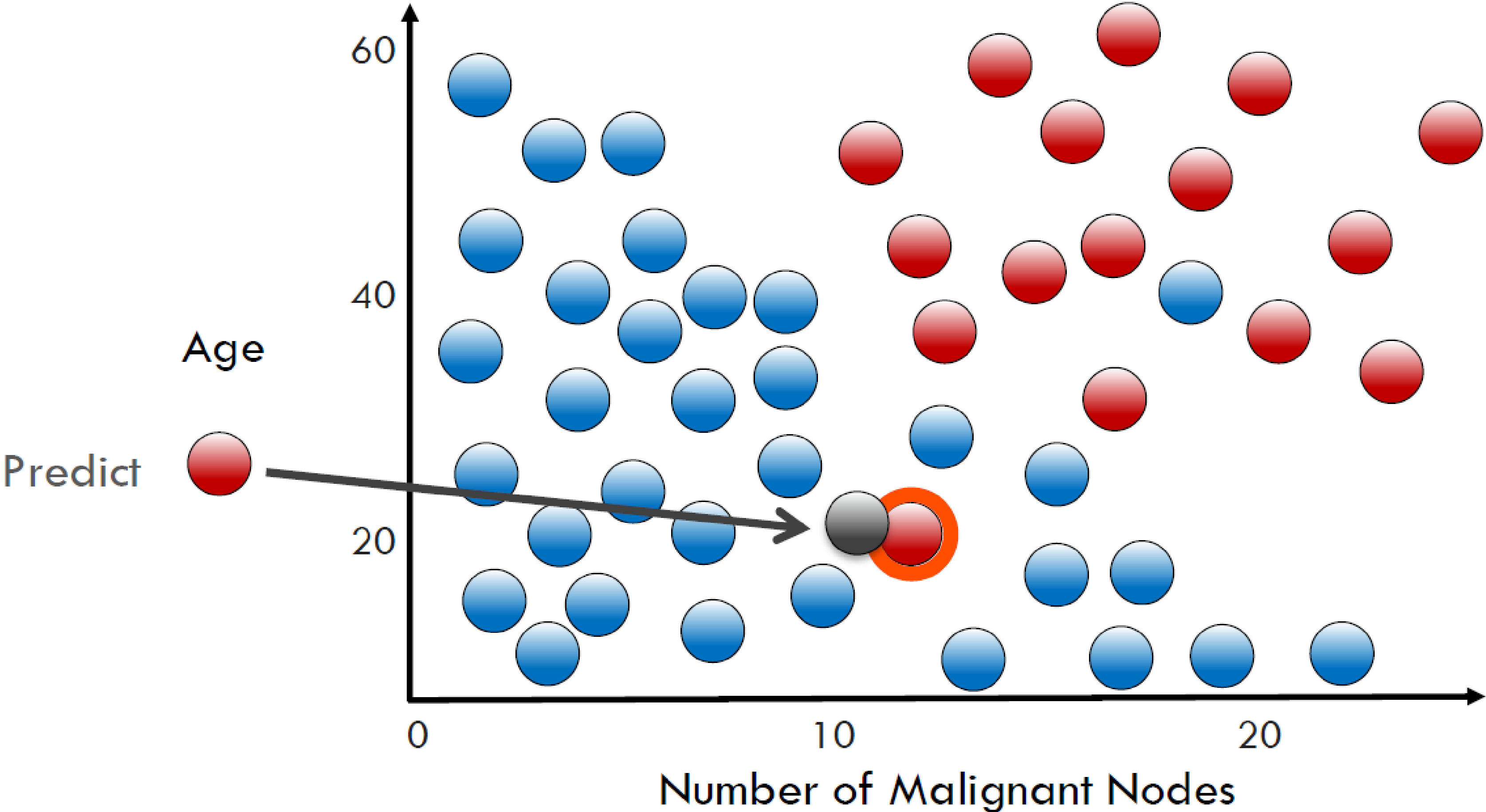
- Survived
- Did not survive



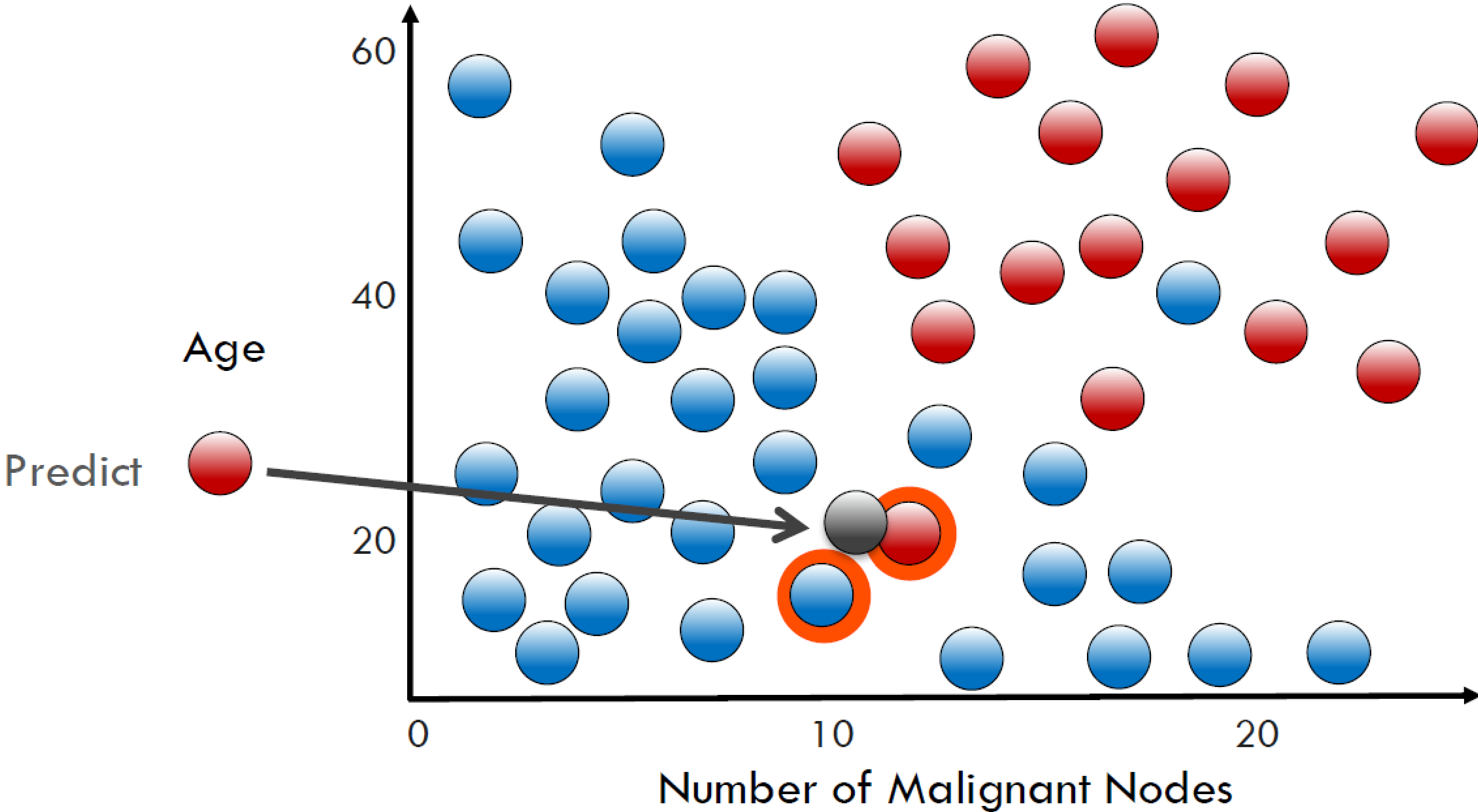
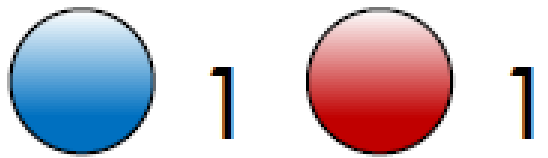



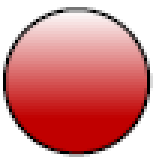
Neighbor Count (K = 1):

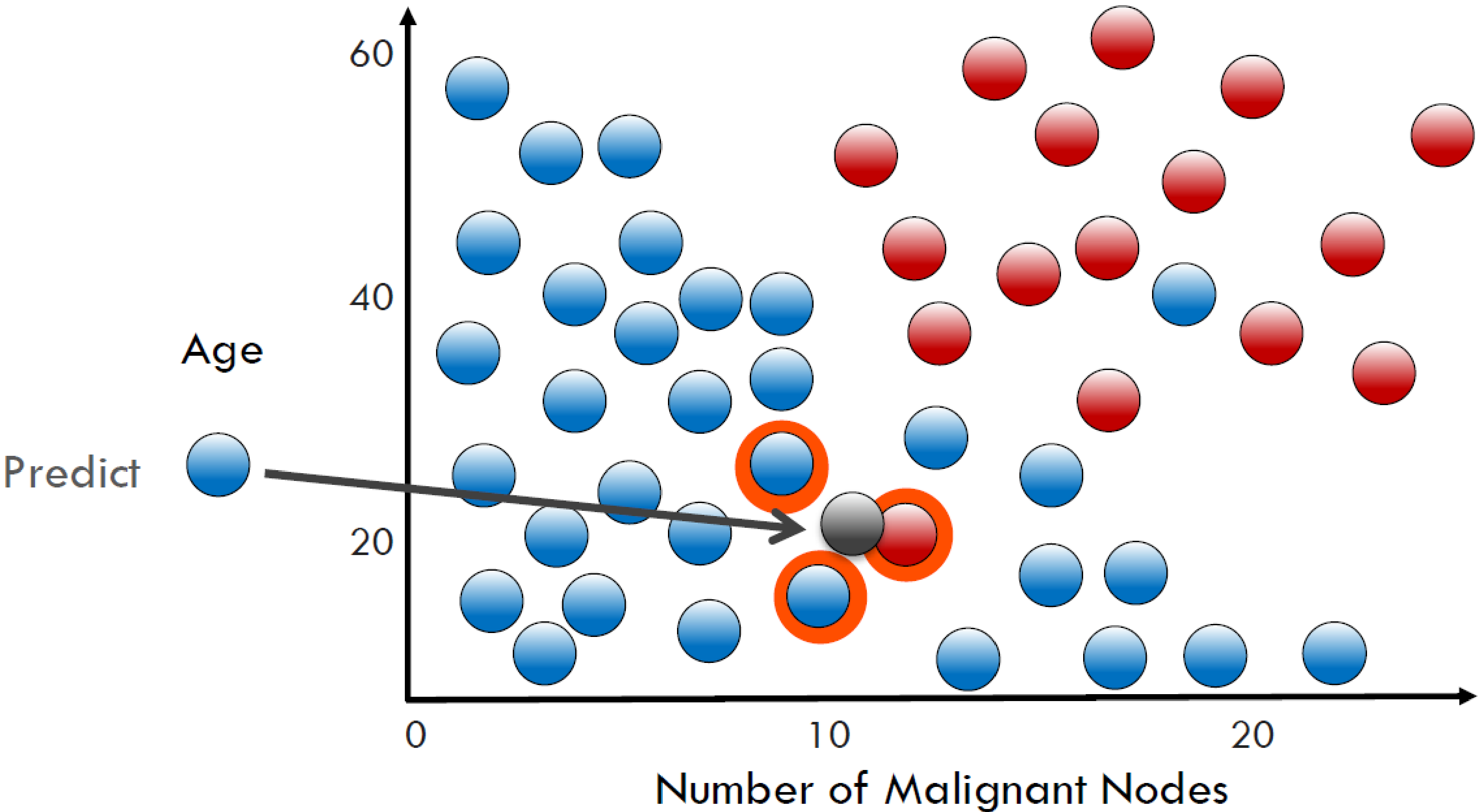
● 0 ● 1

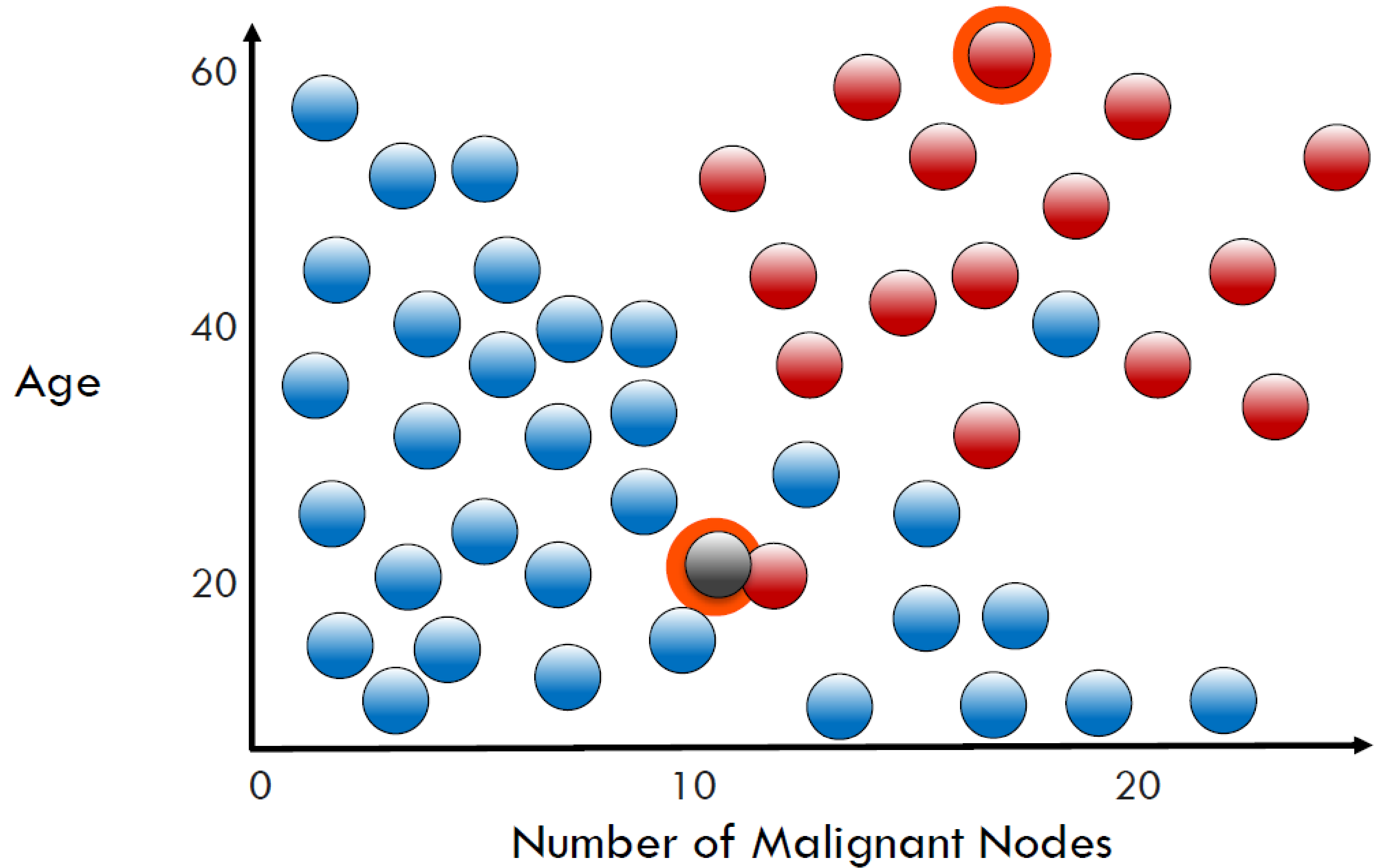


Neighbor Count (K = 2):



Neighbor Count (K = 3):  2  1

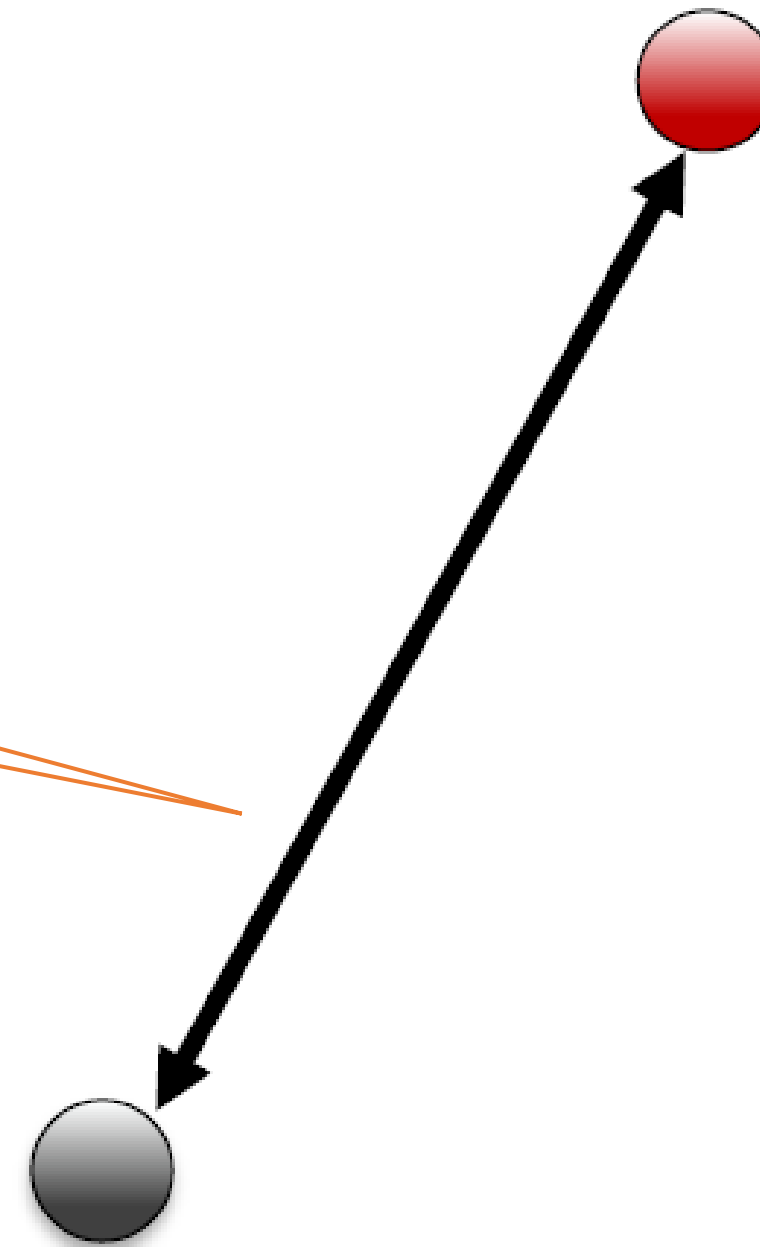


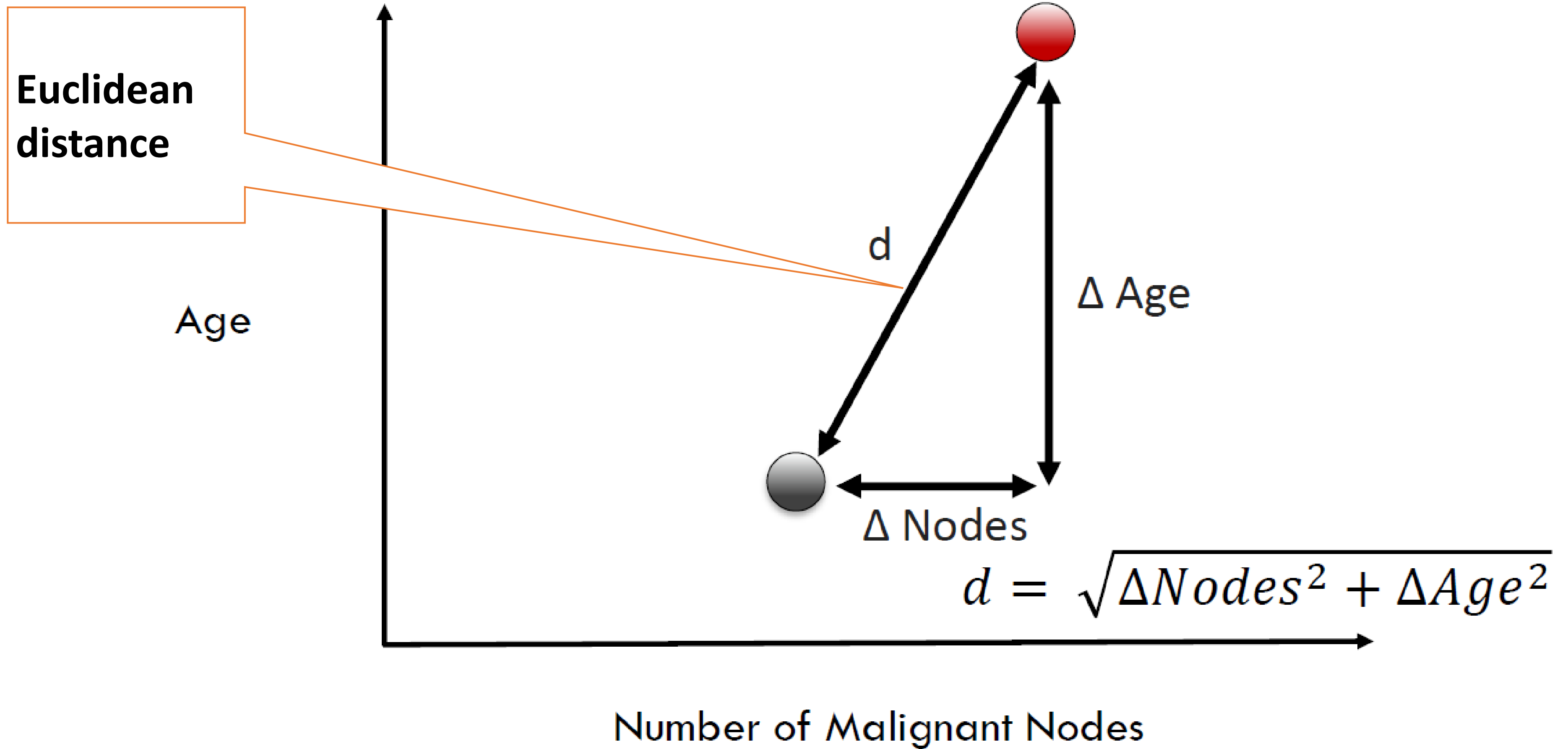


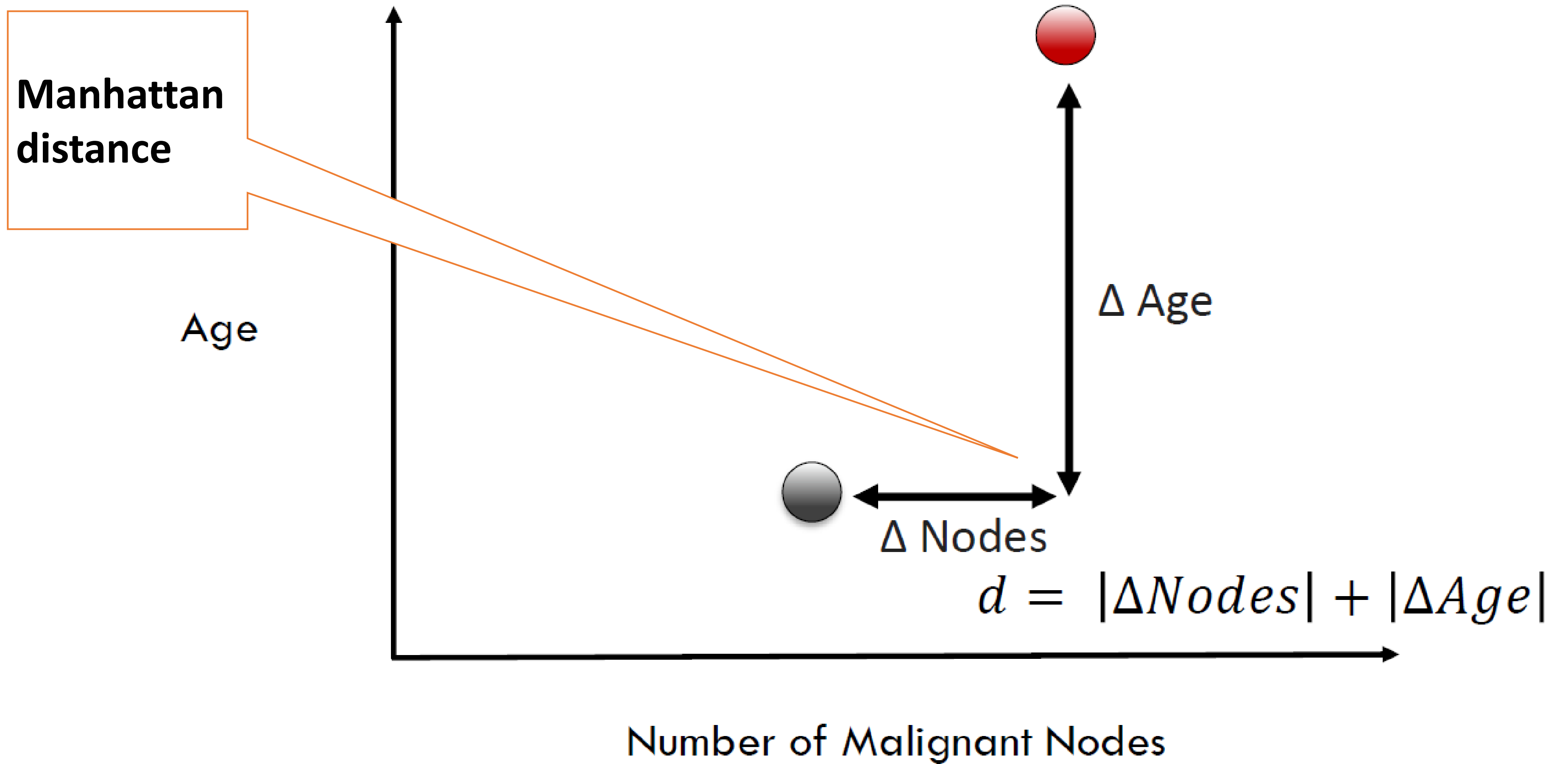
**How to find
distance
between 2
points**

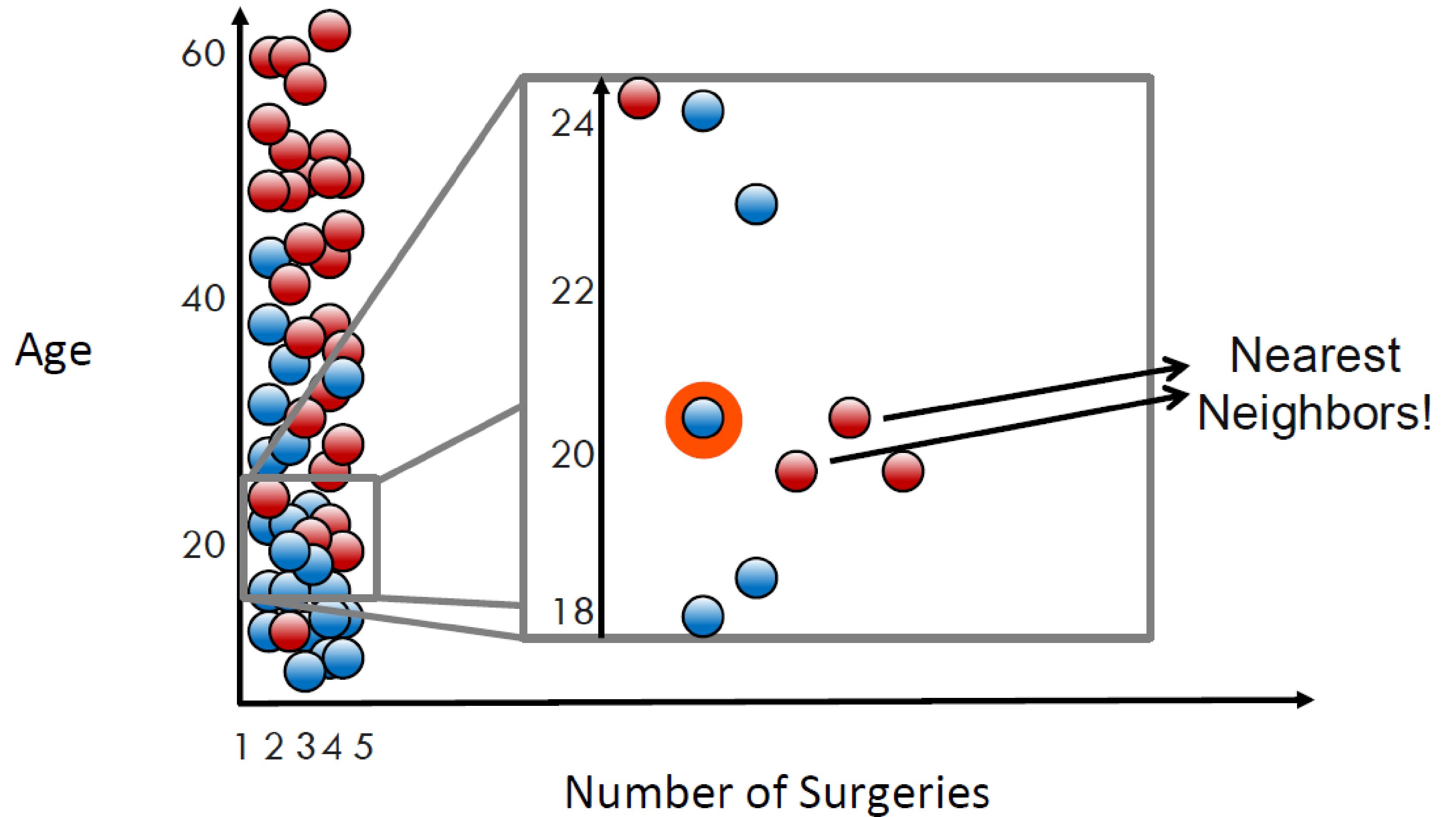
Age

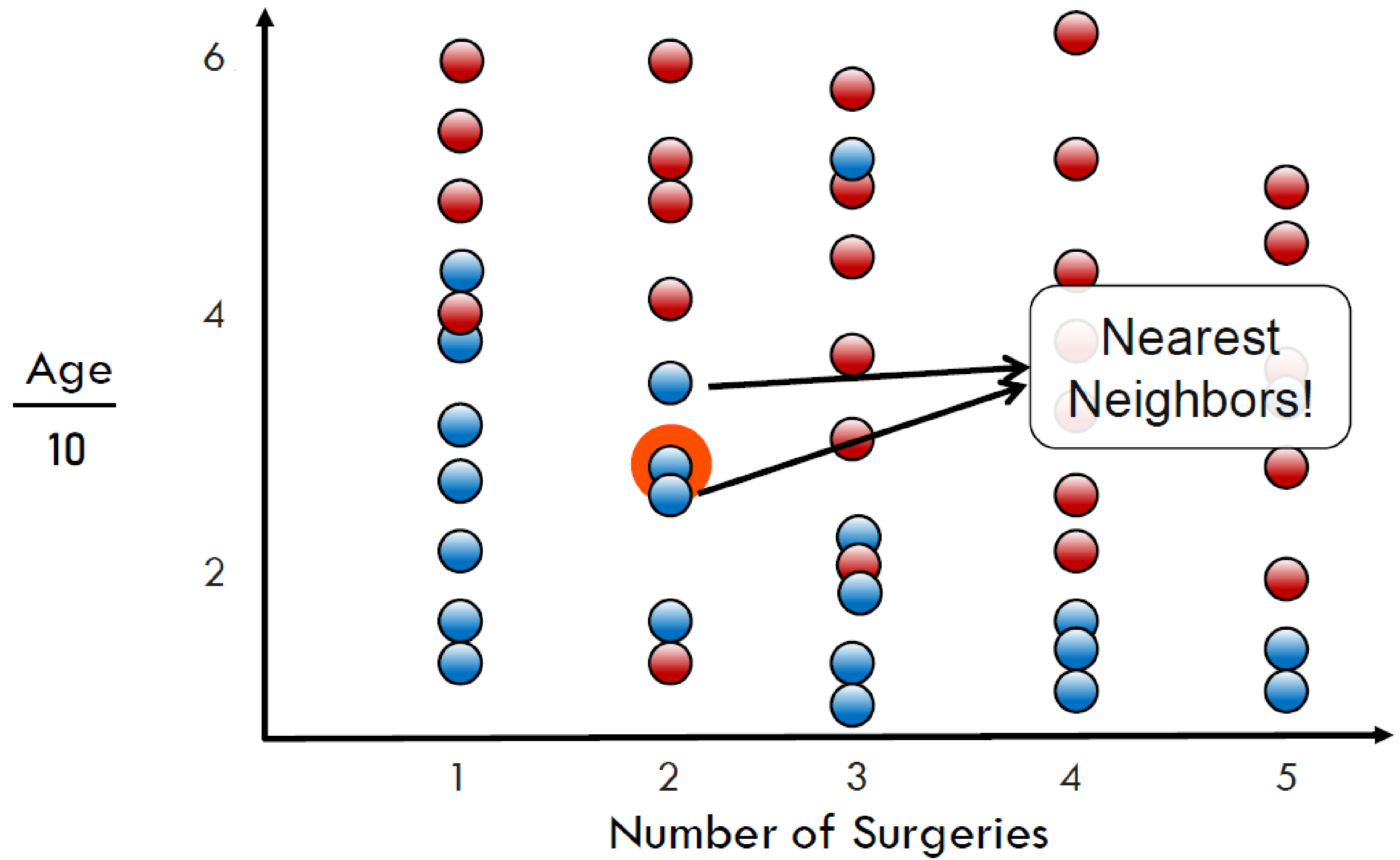
Number of Malignant Nodes

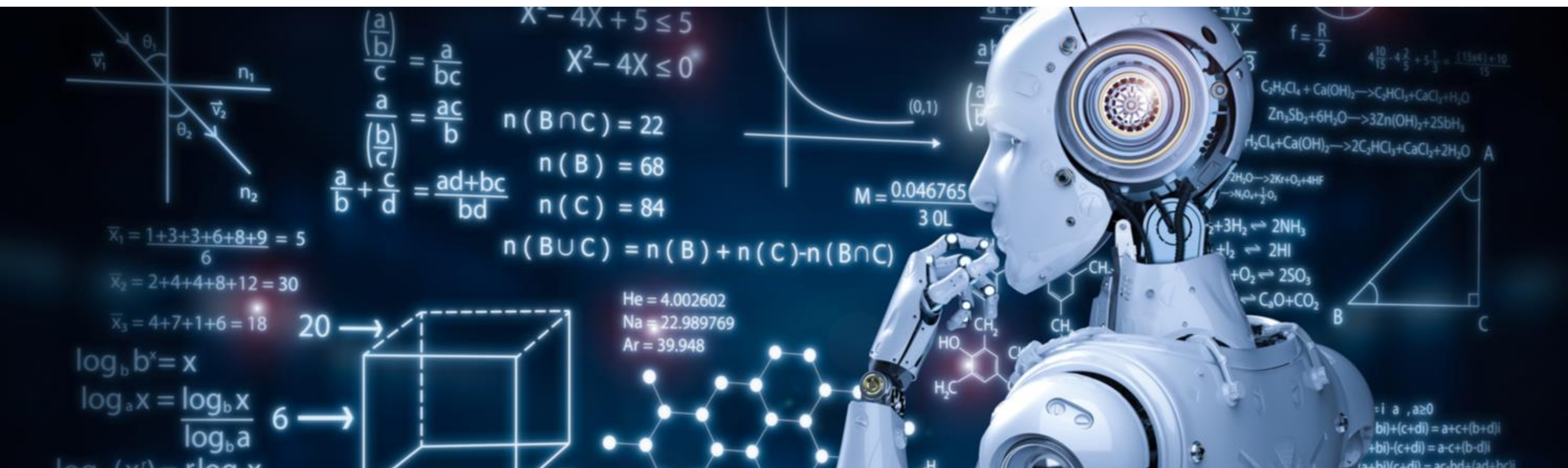








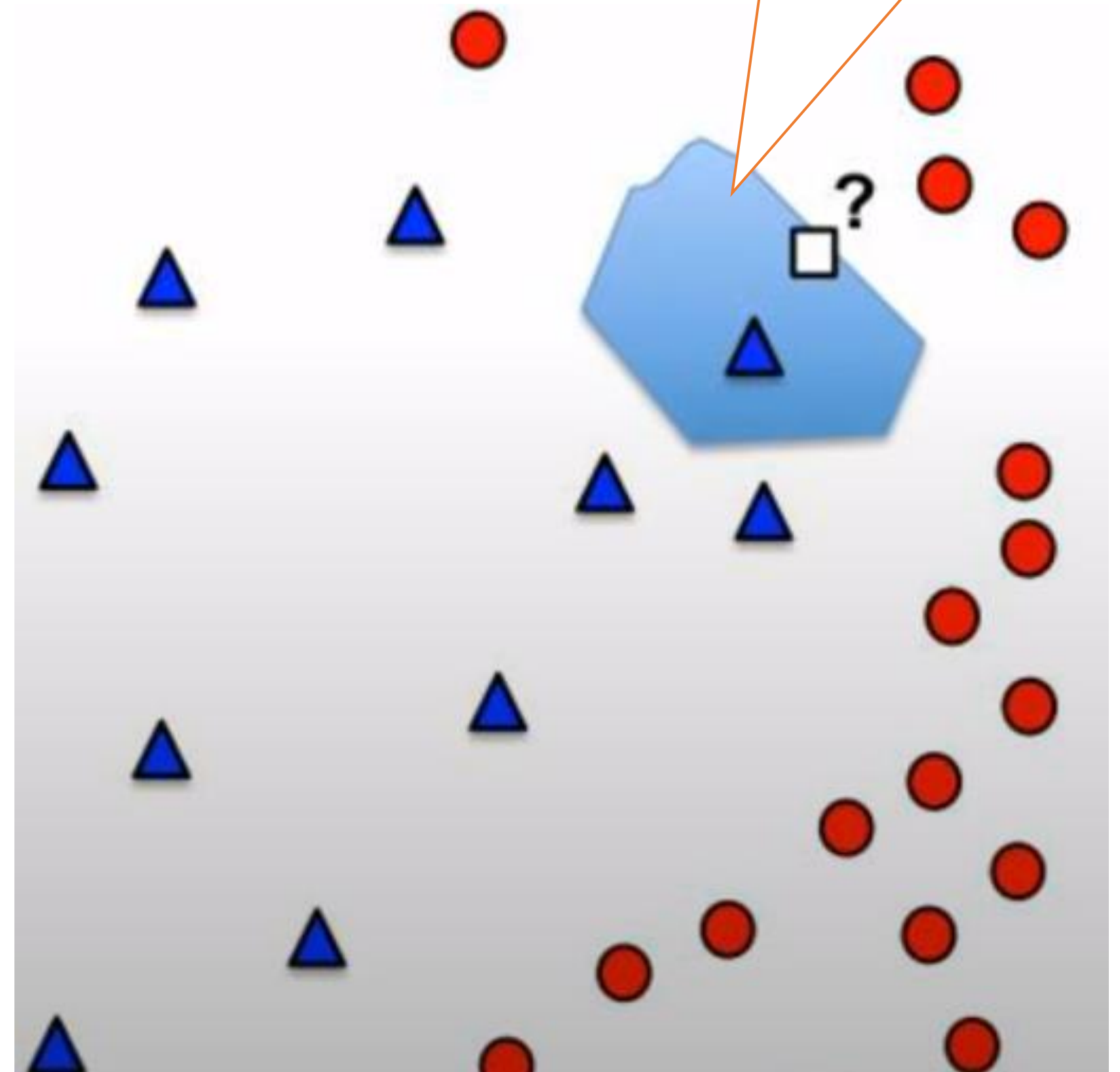
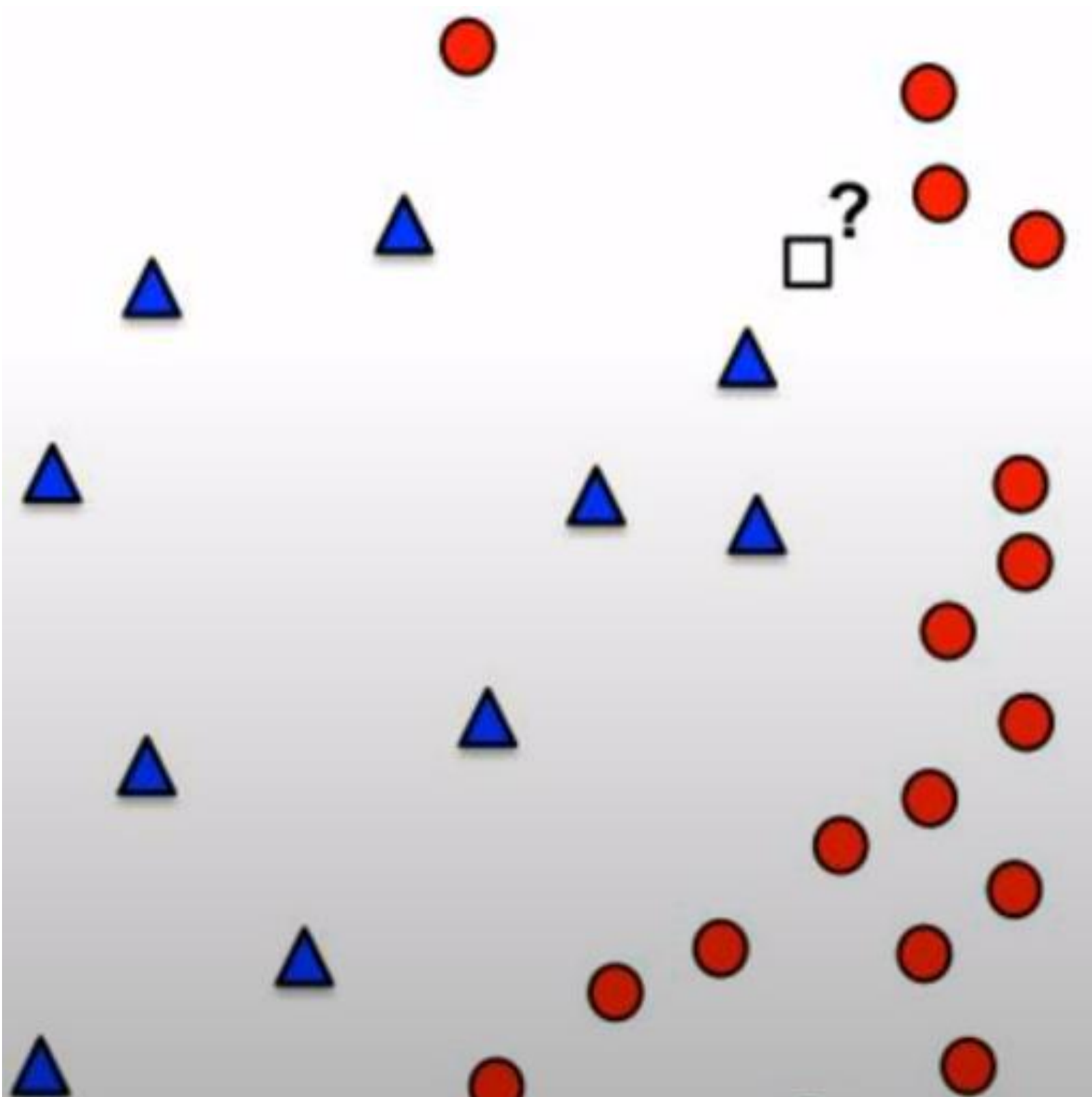




Mitigating overfitting in kNN

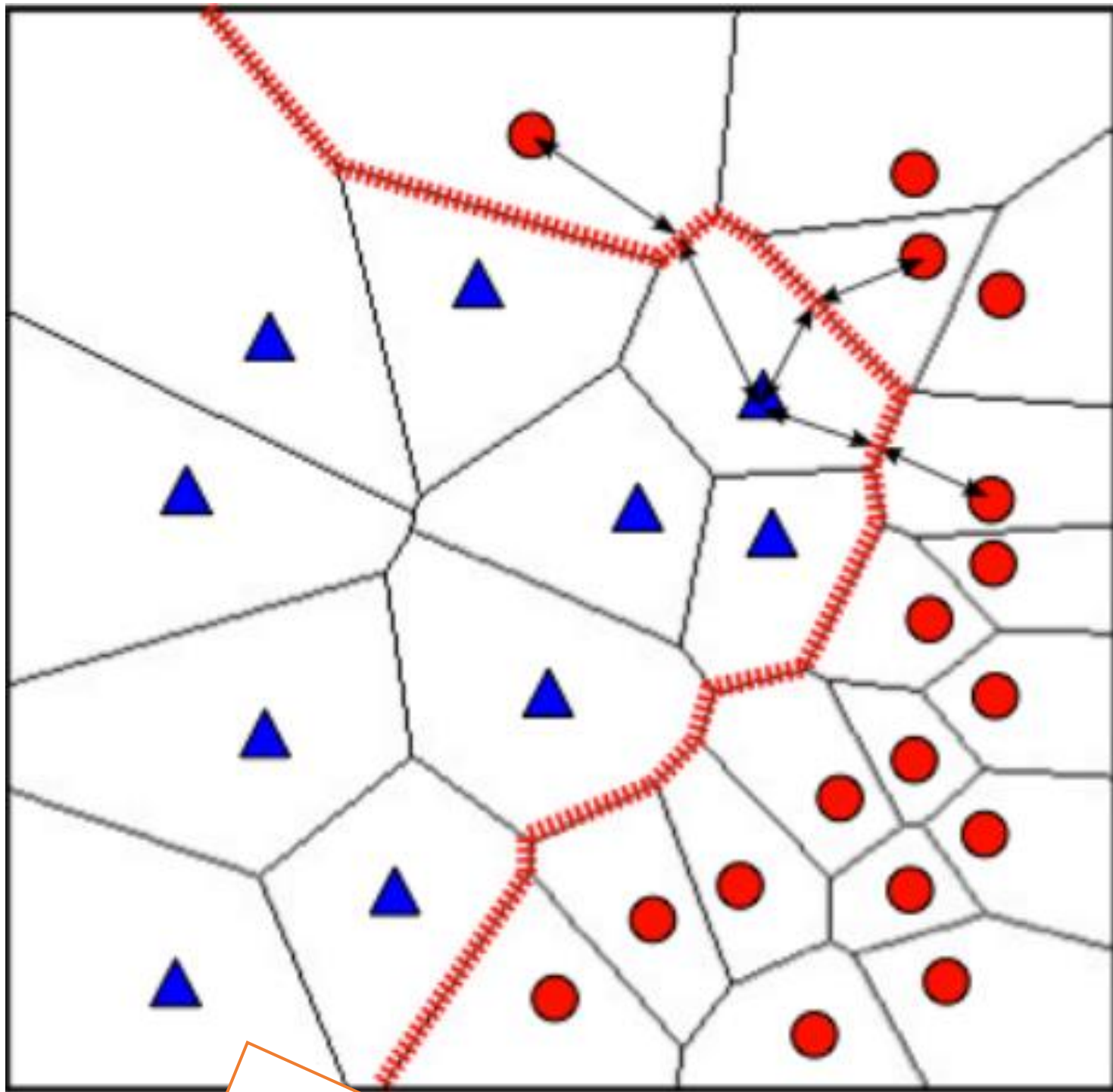
Decision Boundary in kNN

Voronoi cell: contains points closer to blue example than any other

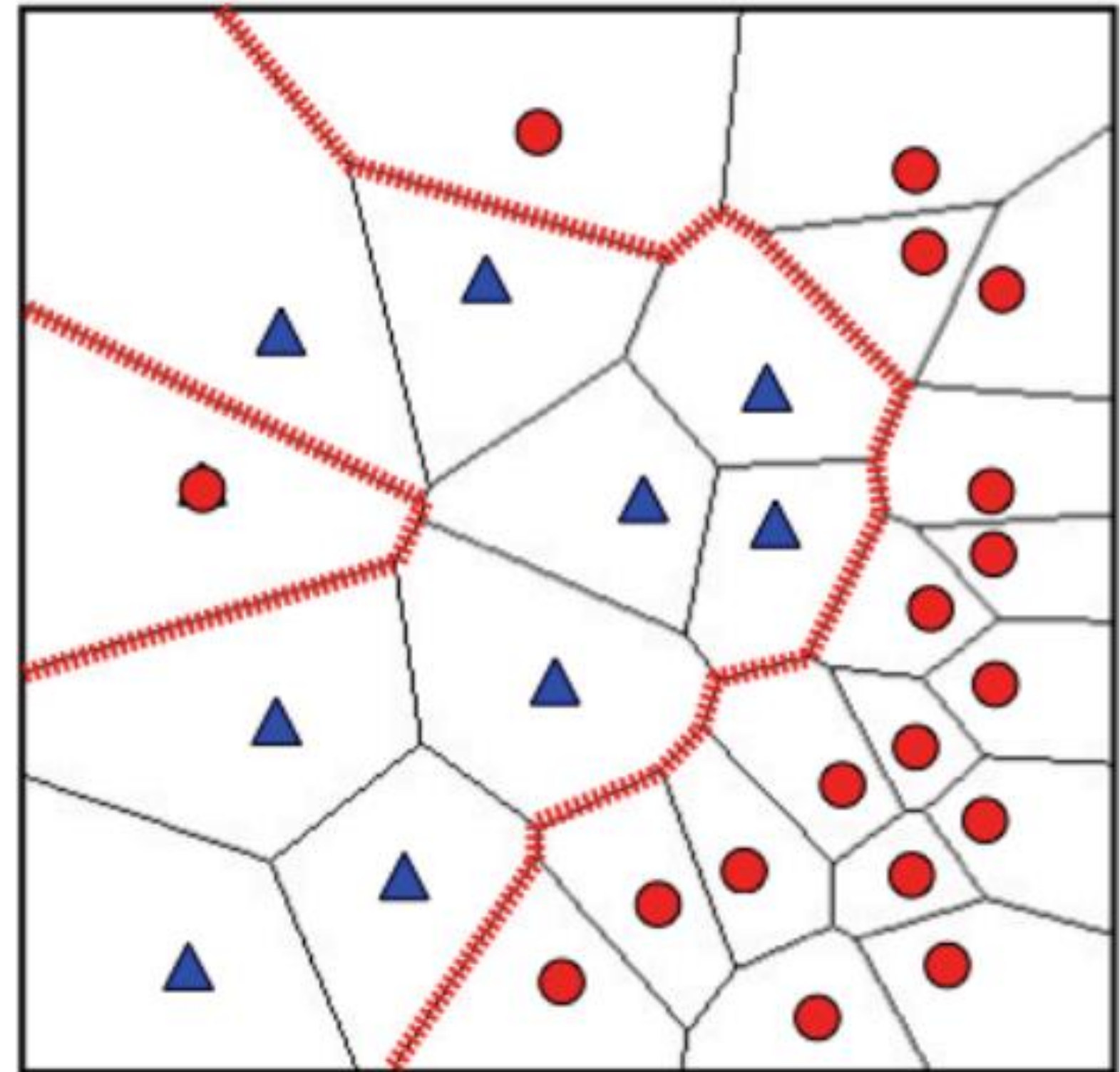


Decision Boundary in kNN (contd.)

- Pretty impressive non-linear boundary
- Flexible implies risk for overfitting

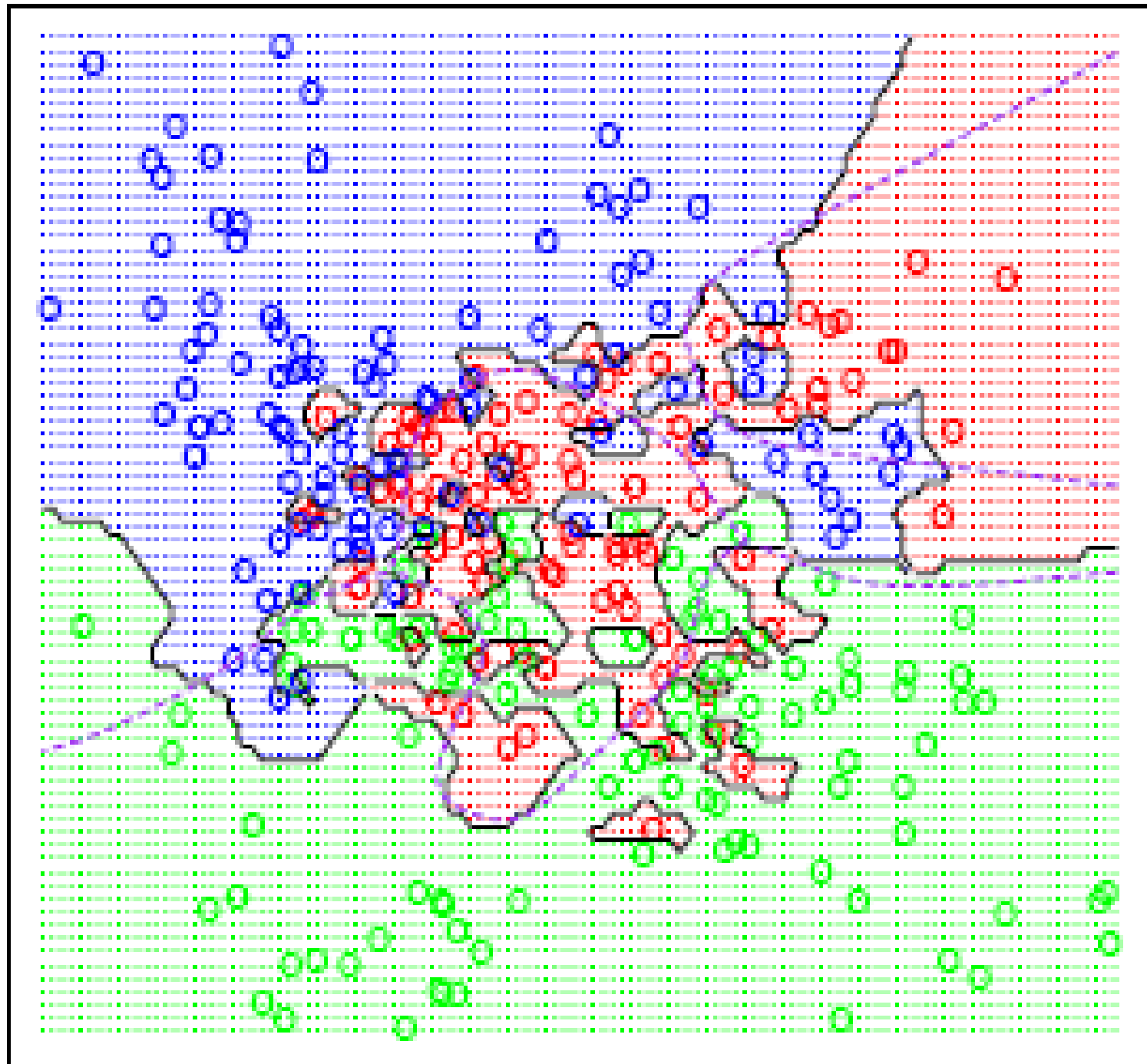


Voronoi Tessellation: Partitioned non overlapping space dominated by one type of training class

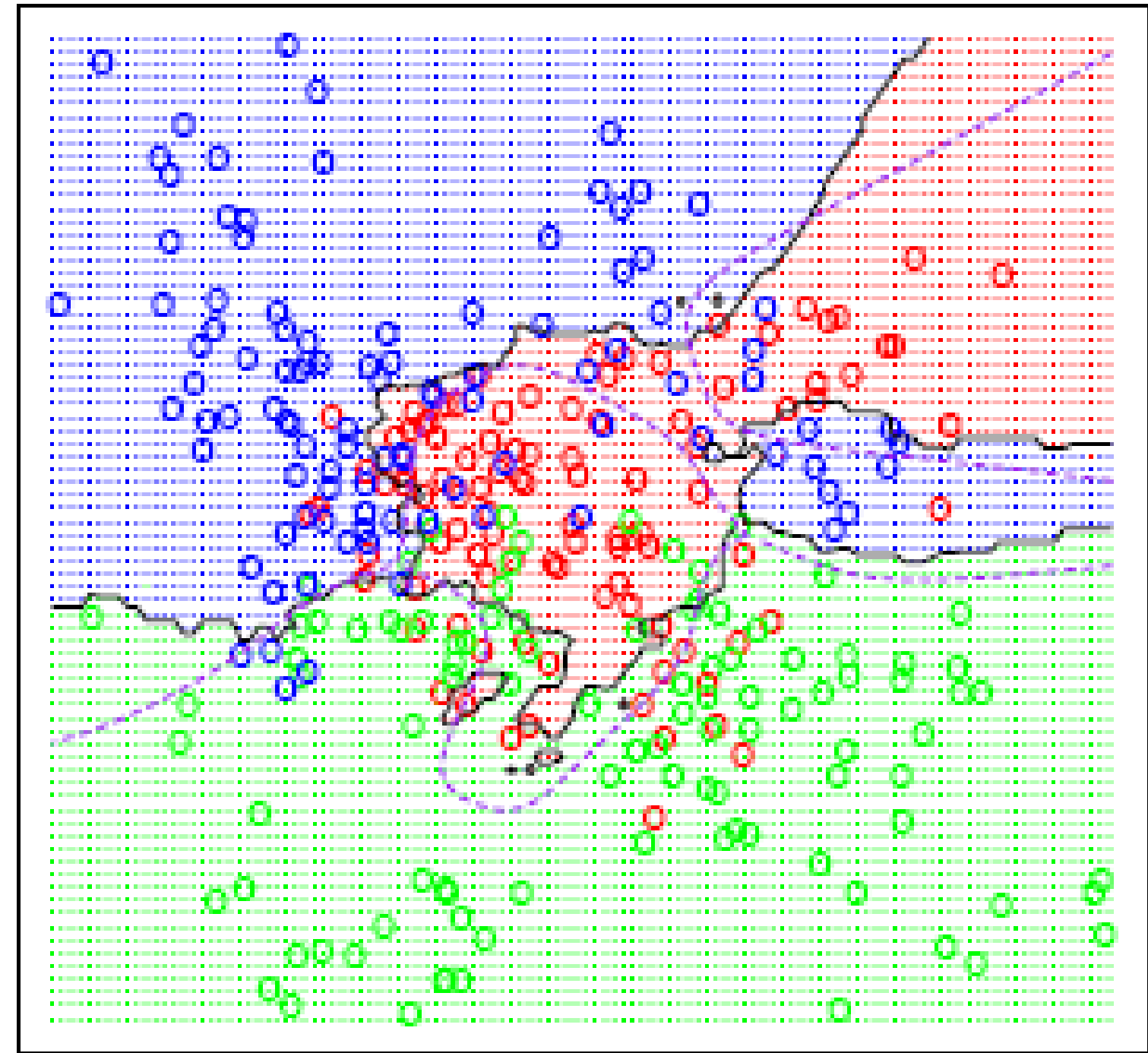


Decision Boundary in kNN (contd.)

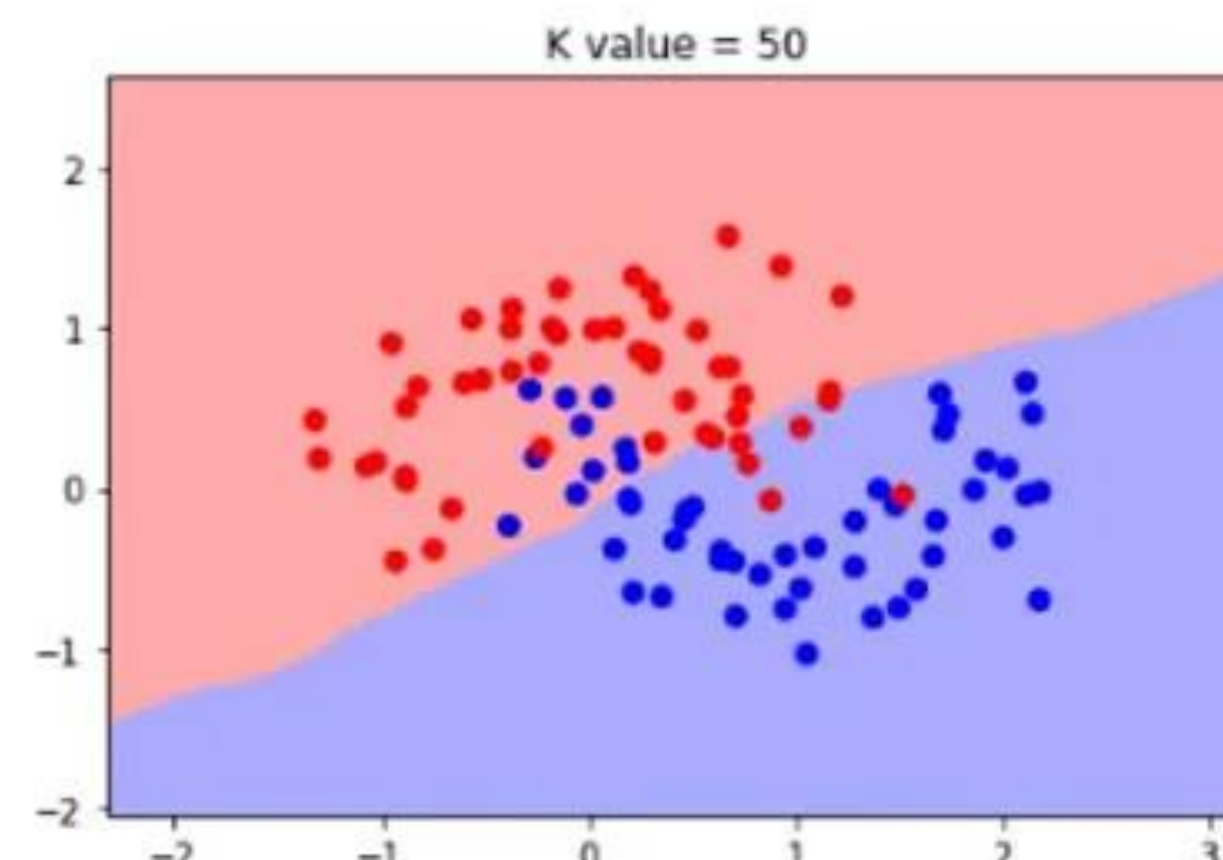
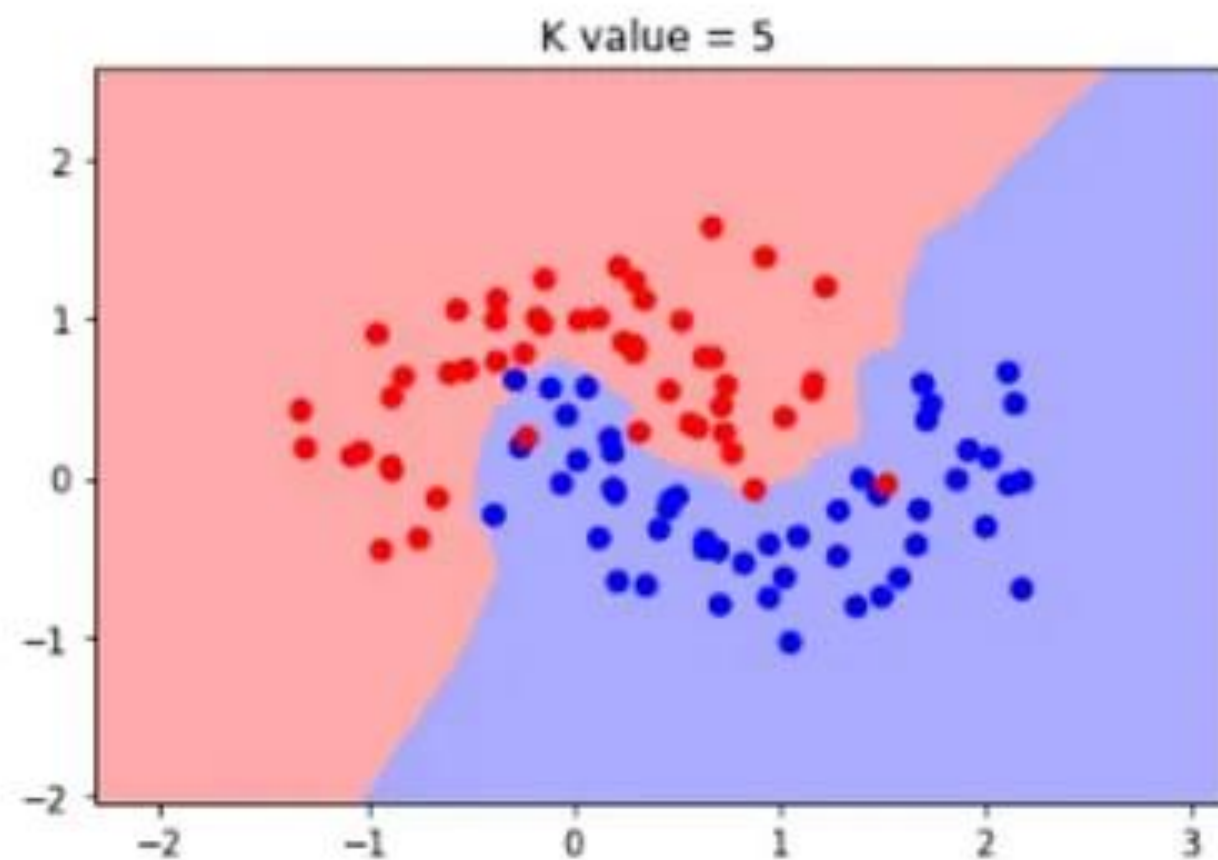
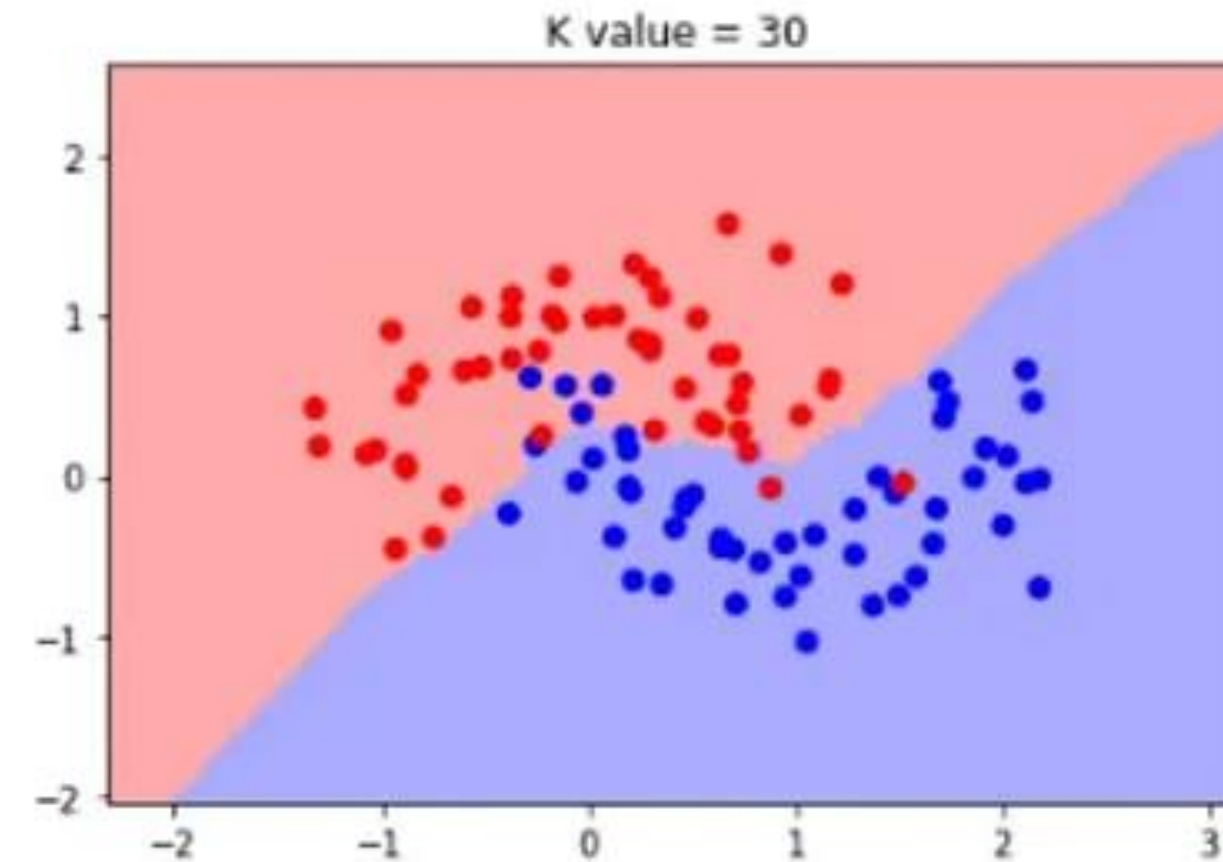
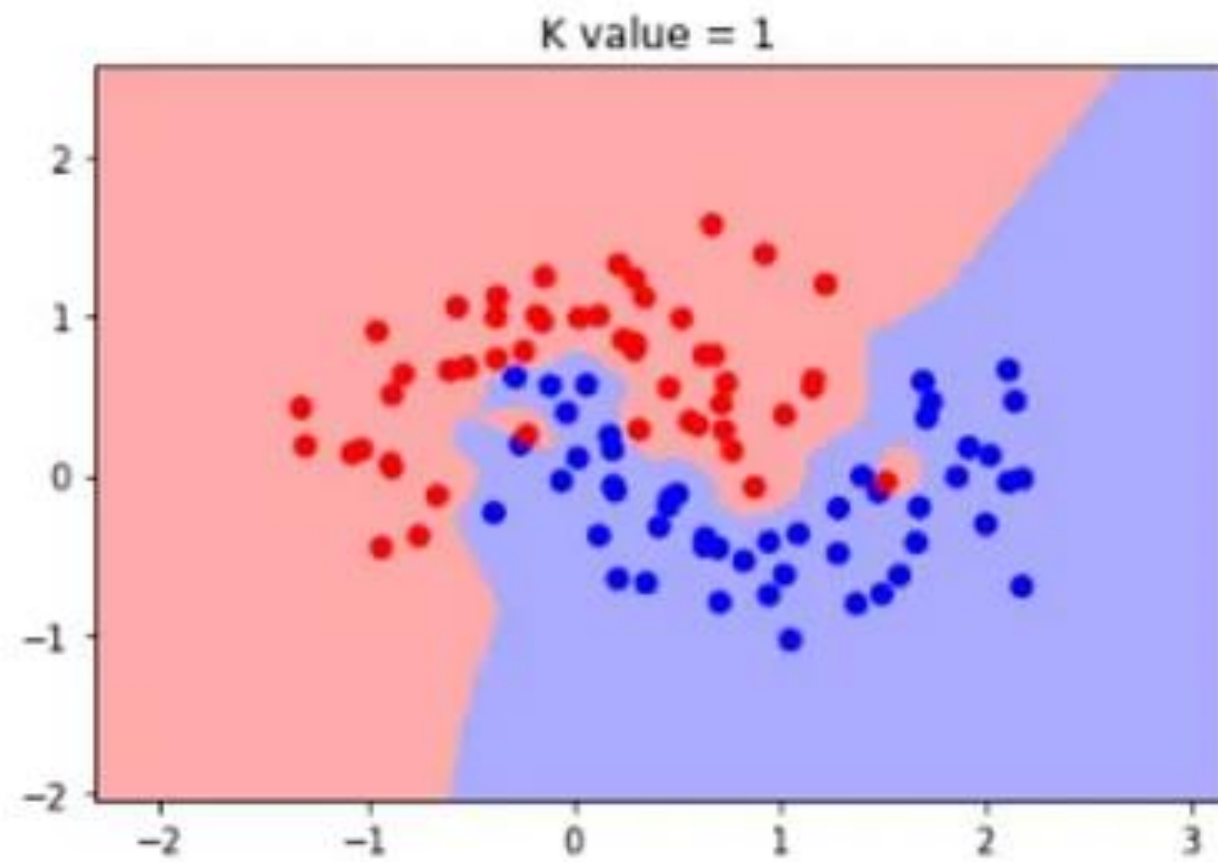
$k=1$



$k=15$

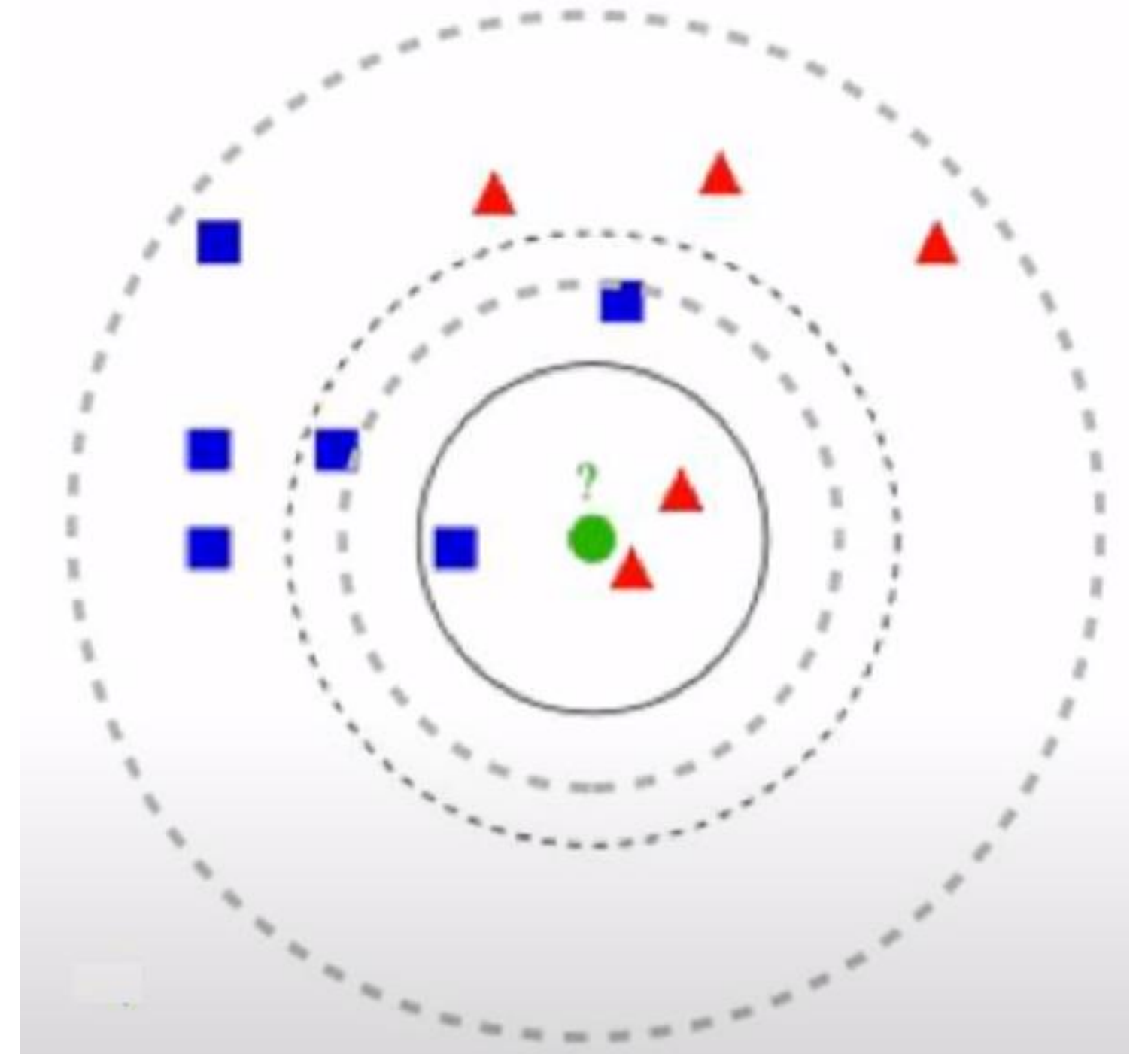


Decision Boundary in kNN (contd.)

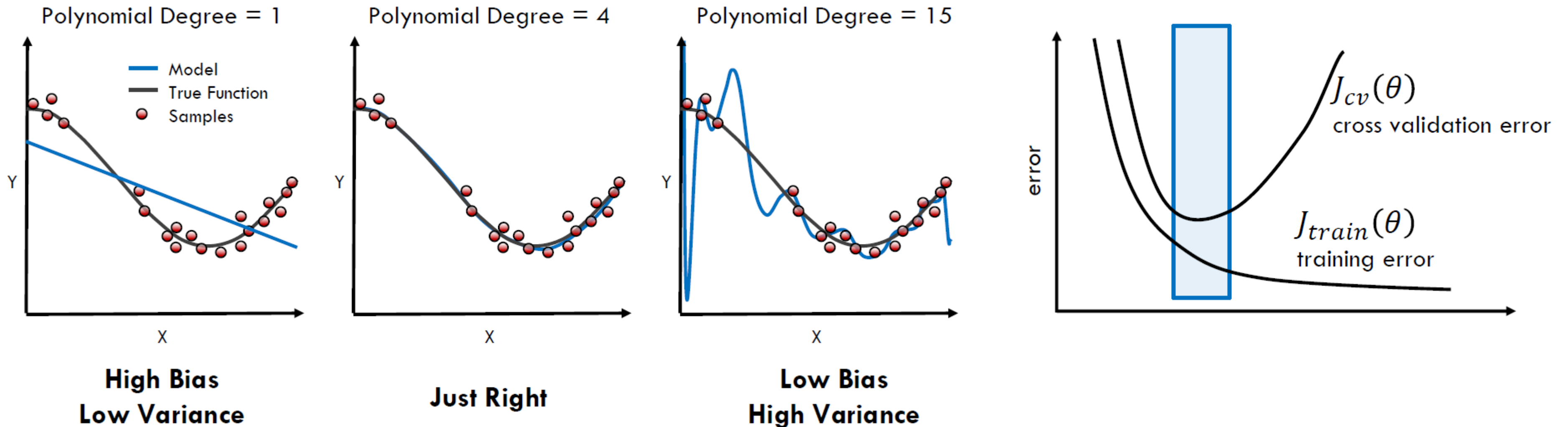


Impact of k

- Value of k has strong effect on kNN performance
- Small k -> unstable decision boundary
- Small change in training set leads to large change in classification
- Extreme case of large k: k=N
 - Every test point classified as the most probable class = $P(y_{\text{maxclass}}) = \text{Prior}$

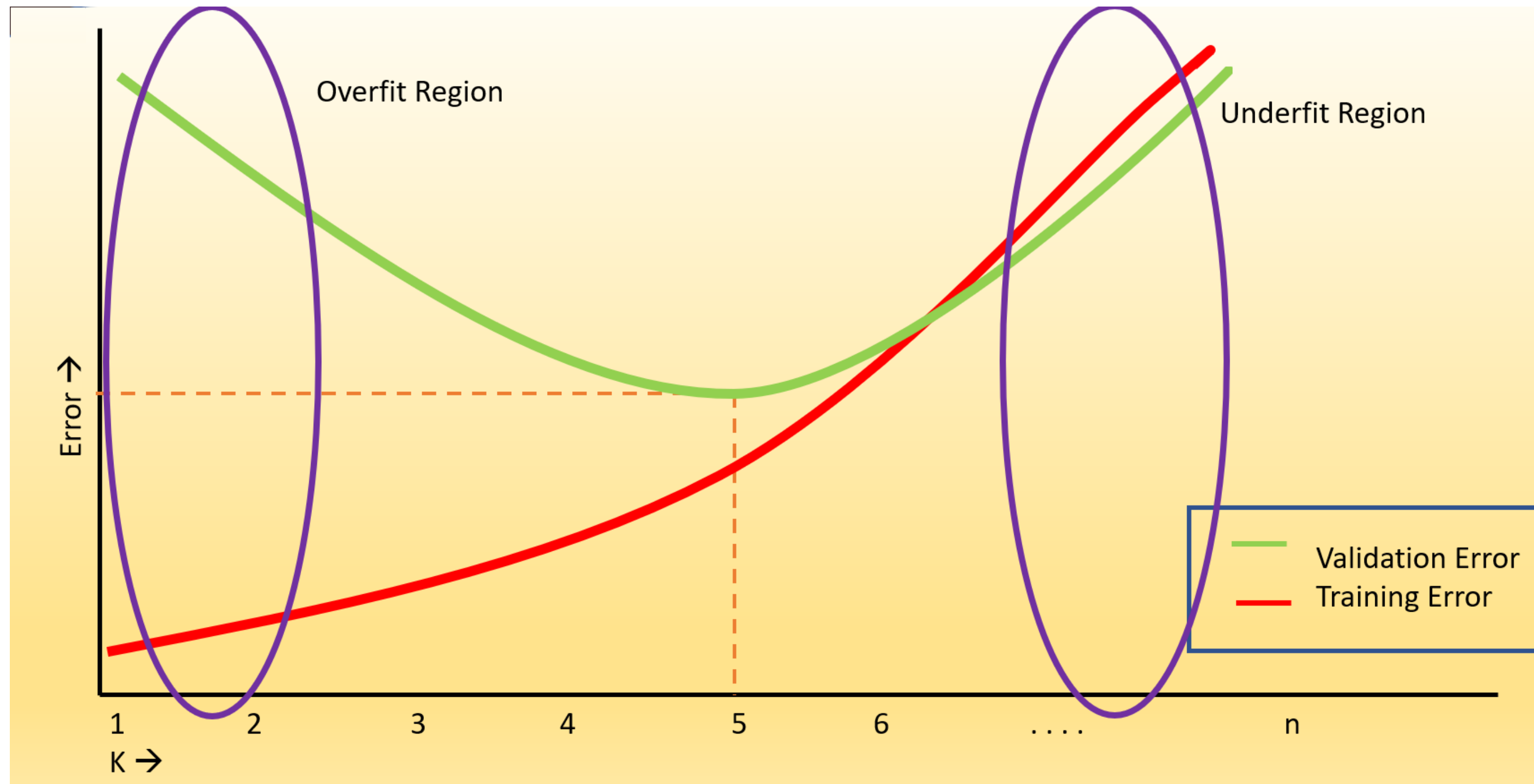


Decision Boundary in kNN (contd.)

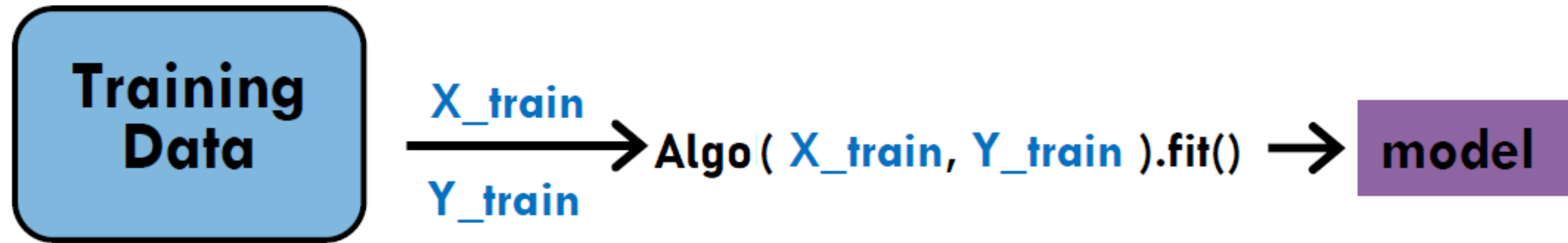


- In other ML algorithms, higher value (such as degree of polynomial in regression) leads to overfitting
- Reverse in kNN
- Overfitting for low k. Underfitting for high k

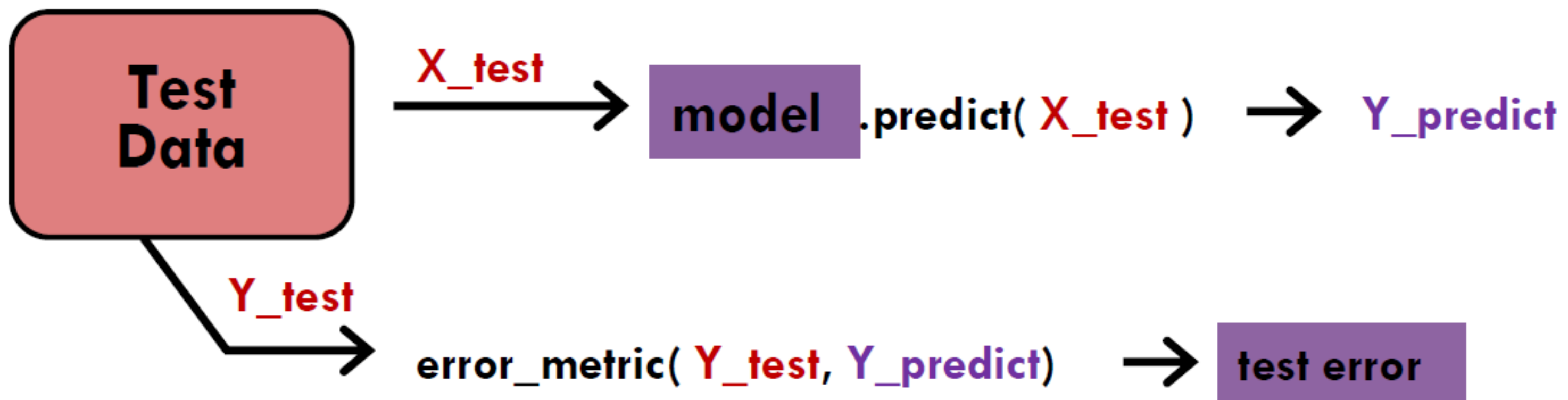
kNN Overfitting v/s Underfitting



Typical train-test steps



- Nearest Centroid training: extract centroids & classes
- Throw away training data after that



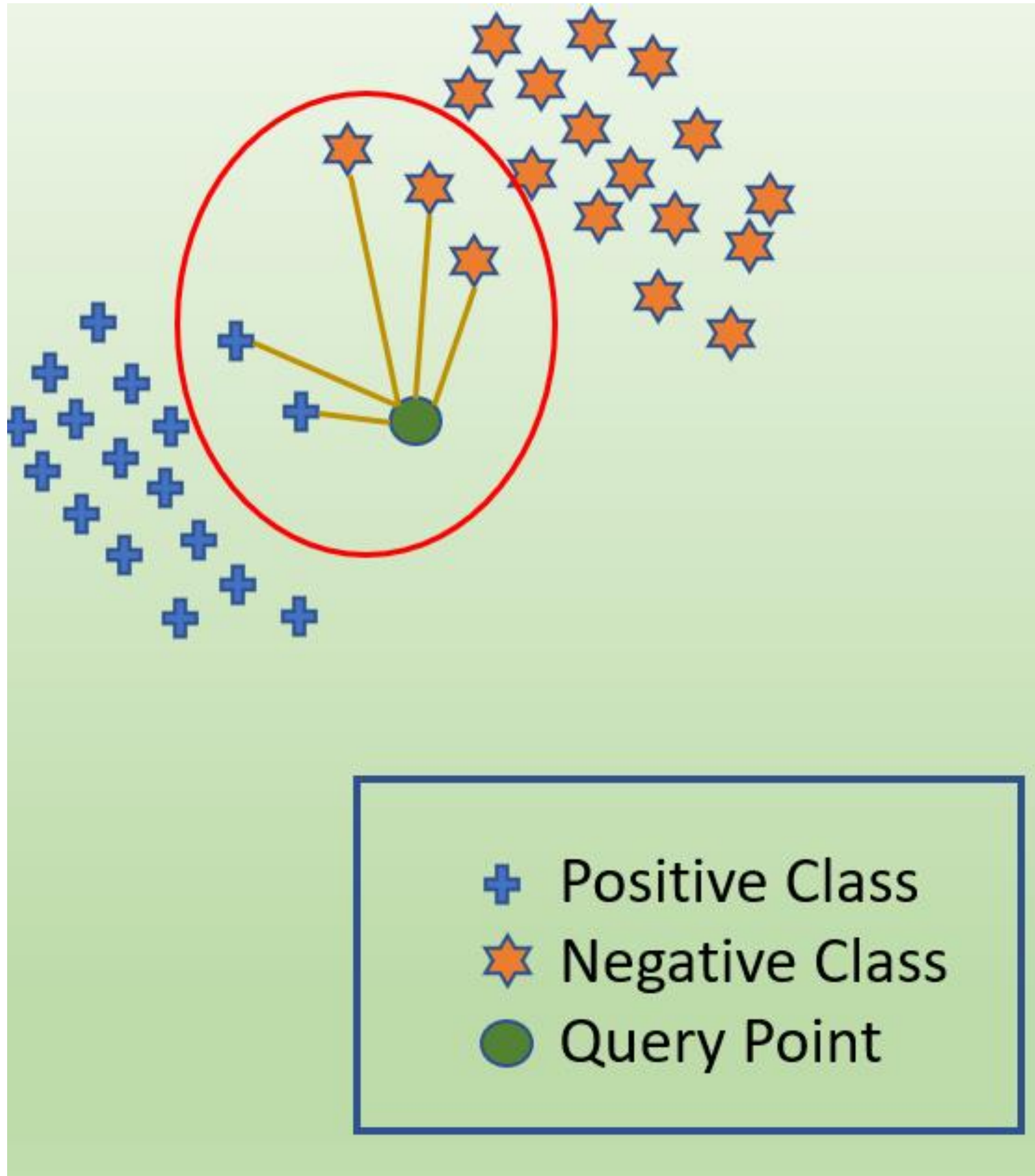
kNN train-test steps

- Train:
 - kNN training phase does nothing
 - Just holds onto the provided hyper params & data
- Test:
 - Multiple test points are provided
 - Runs pairwise distance between all train & test points ($m \times n$ distance computations)
 - Reports score
 - Computationally expensive at prediction time

Nearest Centroid v/s kNN comparison

Nearest Centroid	kNN
Eager	Lazy
Batch	Instance
Parametric	Non parametric
Discriminative	Discriminative

Weighted kNN



$$weight = \frac{1}{distance}$$

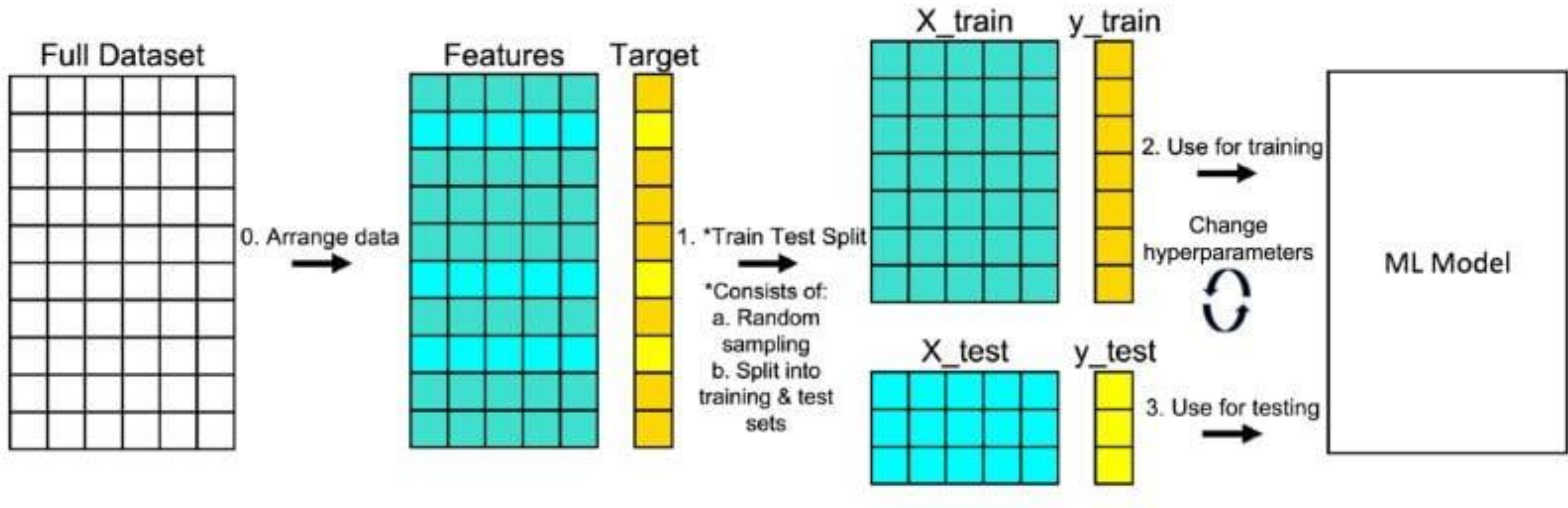
Neighbour	True Label	Distance	Weight	Sum of Weights
X_1	Positive	0.1	10	13.3
X_2	Positive	0.3	3.3	
X_3	Negative	1	1	1.8
X_4	Negative	2	0.5	
X_5	Negative	3	0.3	



How to select a kNN model?

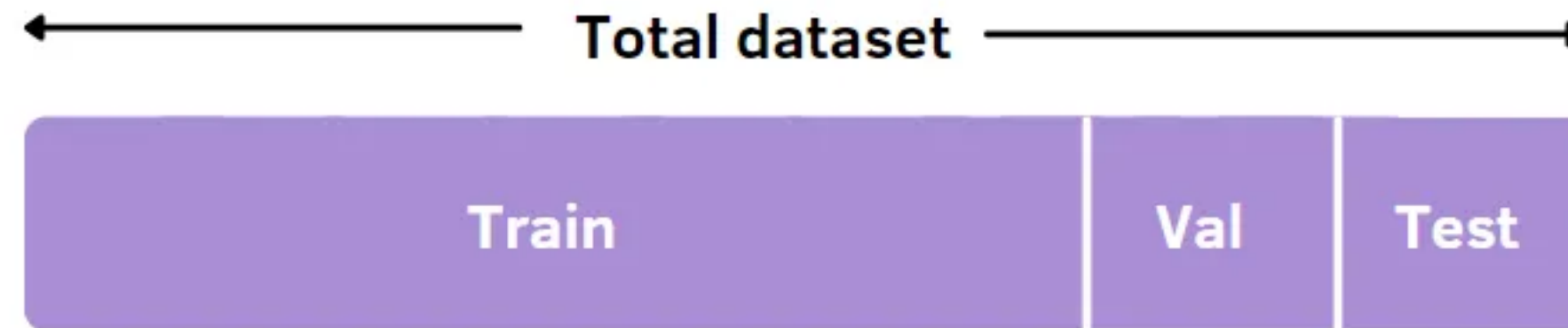
- Basic kNN - Two hyperparameters to choose
 - k – The number of nearest neighbors
 - Distance measure (Euclidean, Manhattan)
- Many more hyperparameters exist
 - How about weighted kNN?
- Parameters & hyperparams – What's the difference
- Hyperparameter tuning

Option 1: Use train-test split



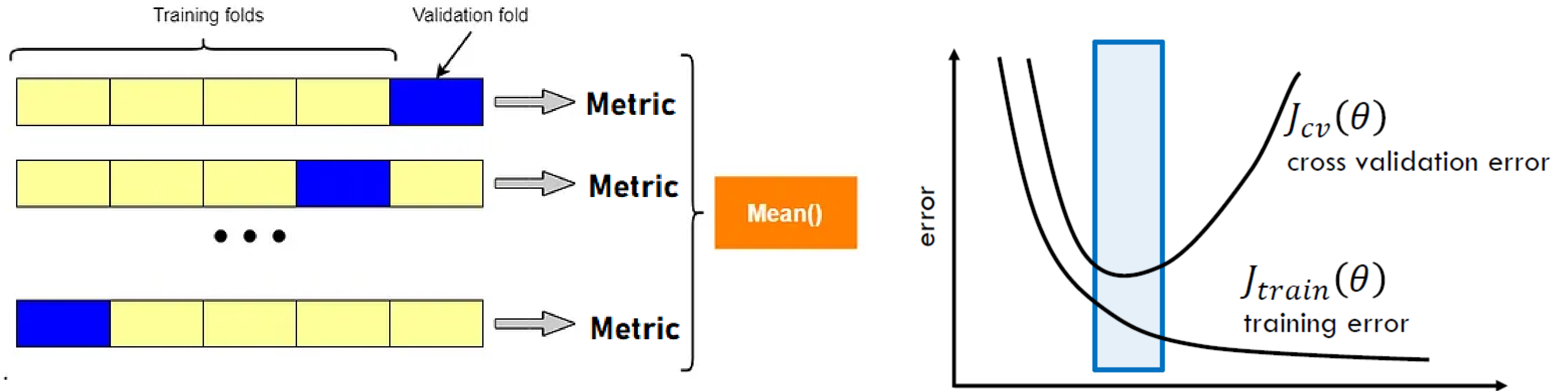
- Issue:
 - We will never know if a model is good without running on test split

Option 2: train-validation-test split



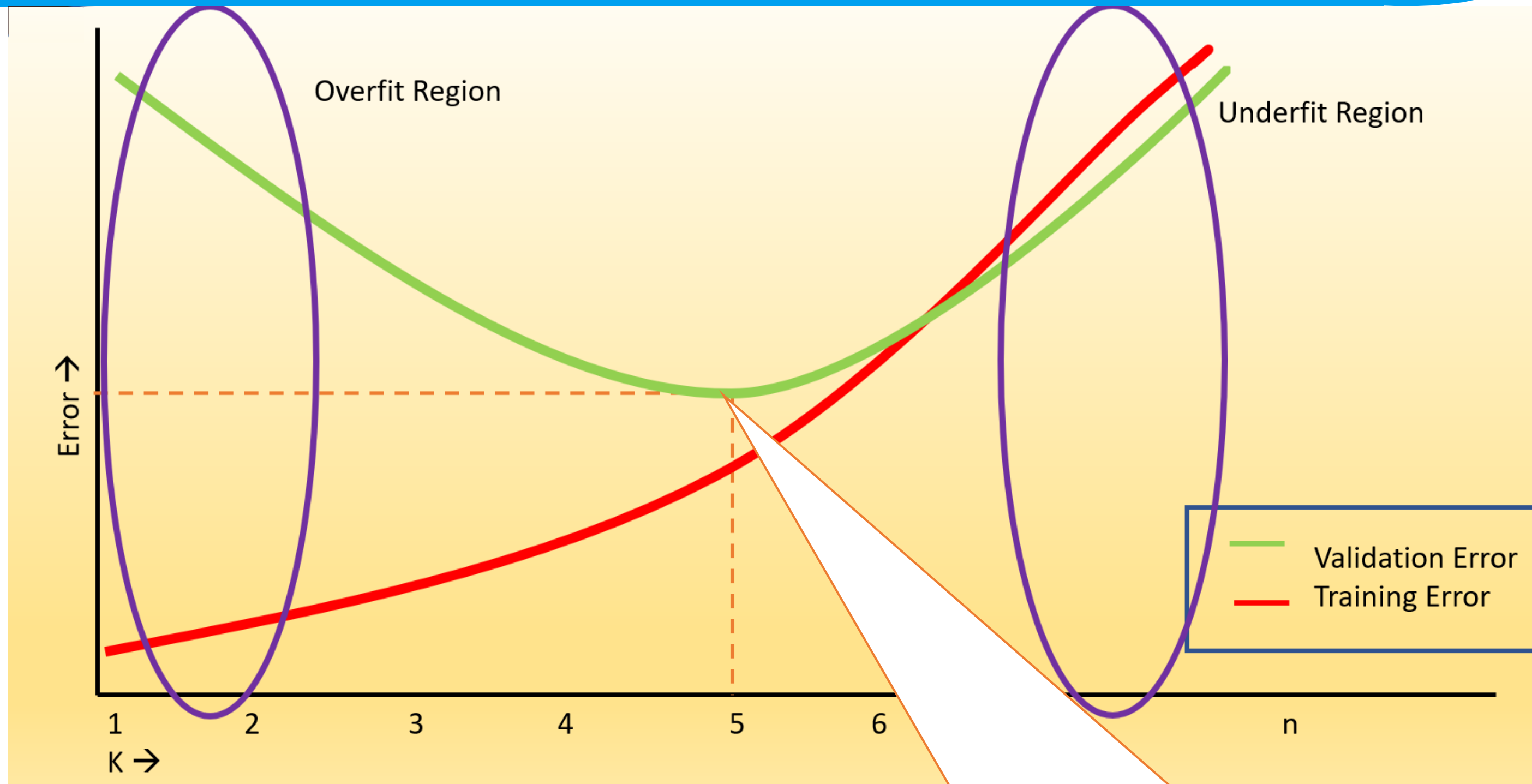
- Classical, old fashioned approach: 60:20:20 split
- Acceptable when there is lot of data
- Issue:
 - Model does not generalize when less data
 - Results in overfitting

K-Fold CV (contd.)



- For each of $k=1,3,5,7,10$ in kNN:
 - Run 5-Fold CV
 - Get mean train and val error. Plot
- In general, k with lowest val error is best k

kNN K-Fold CV

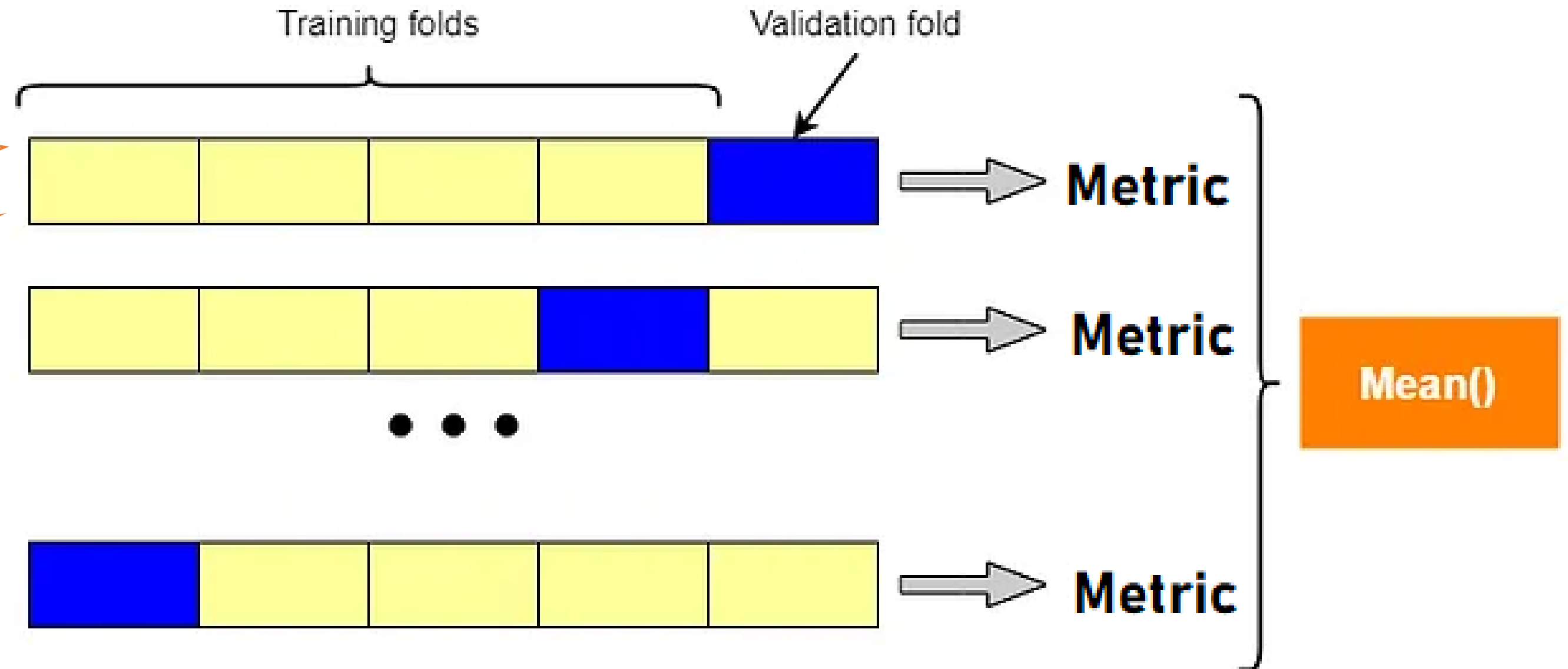


**Classifier metric may measure accuracy
& higher the better.
We look for best score in CV**

Option 3: K-Fold Cross Validation (CV)

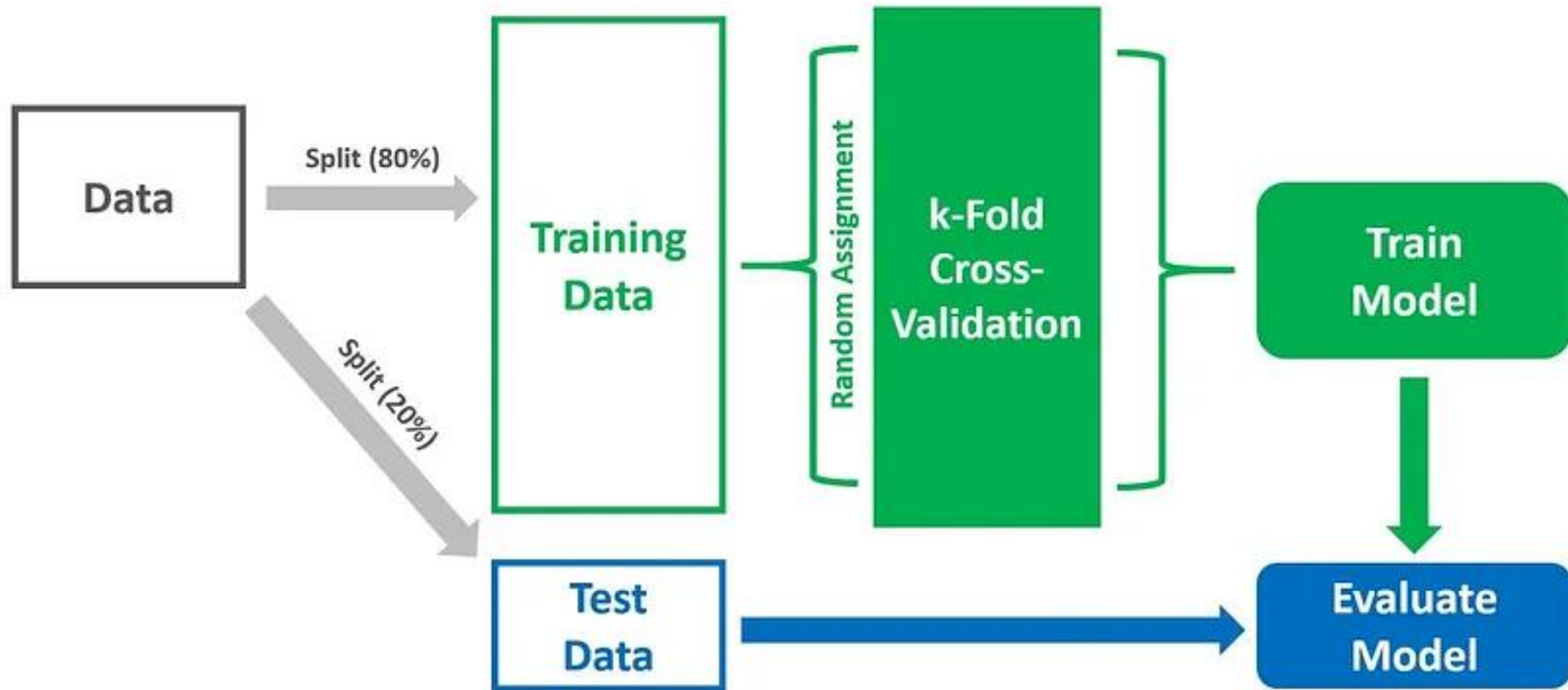
Choose $k = 1$ for
kNN (1 neighbor)

Repeat with $k = 3,$
 $5, 7$ for kNN

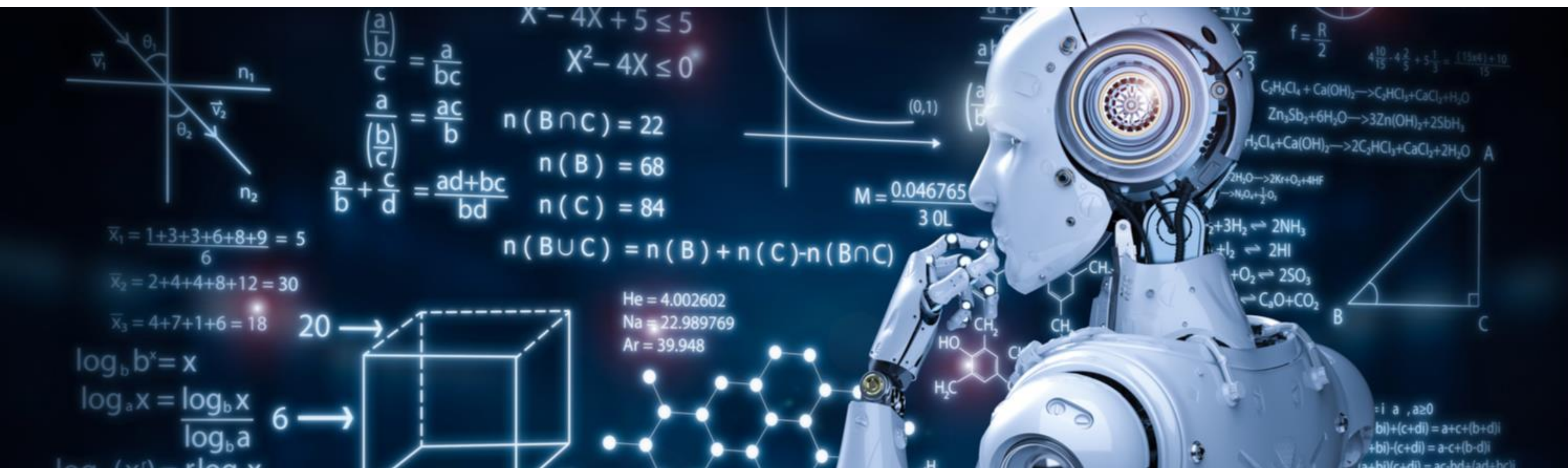


- Mean errors for different $k = [10.5, 12, 12, 8.5, 18]$
- K-Fold CV is used to select best k for kNN
- Note: K in K-Fold CV got nothing to do with k in kNN
- Choose $K = 5$ (or 10) in K-Fold CV. This is constant

Example: k-Fold Cross-Validation



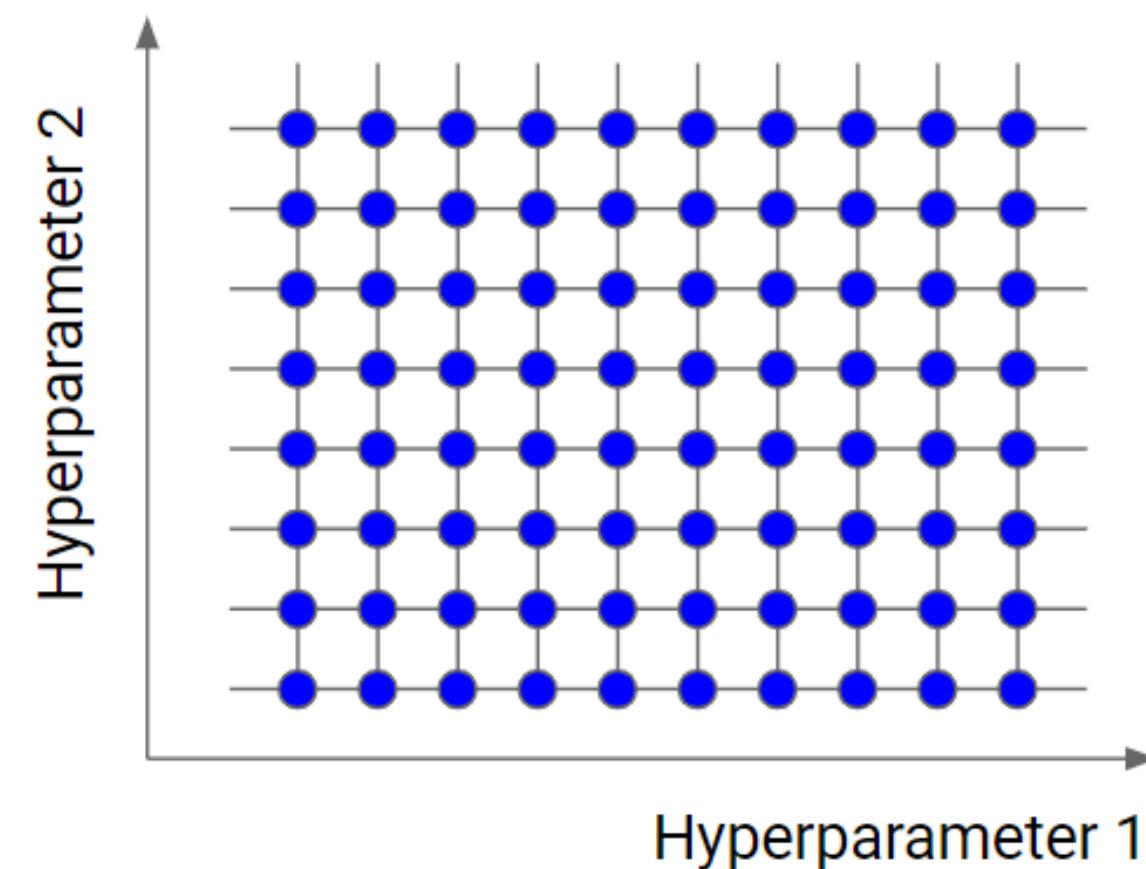
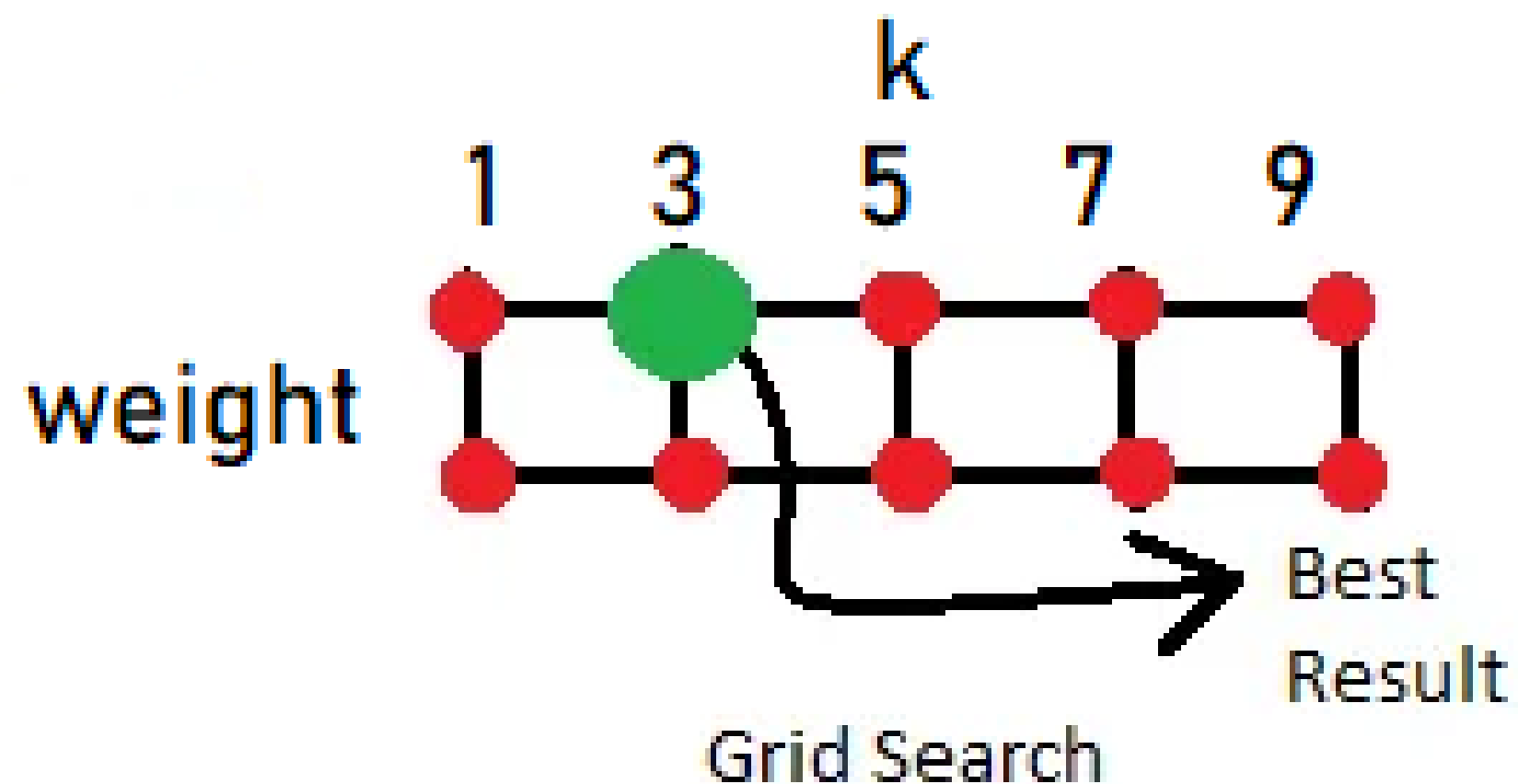
- Can we use K-Fold CV for Nearest Centroid?



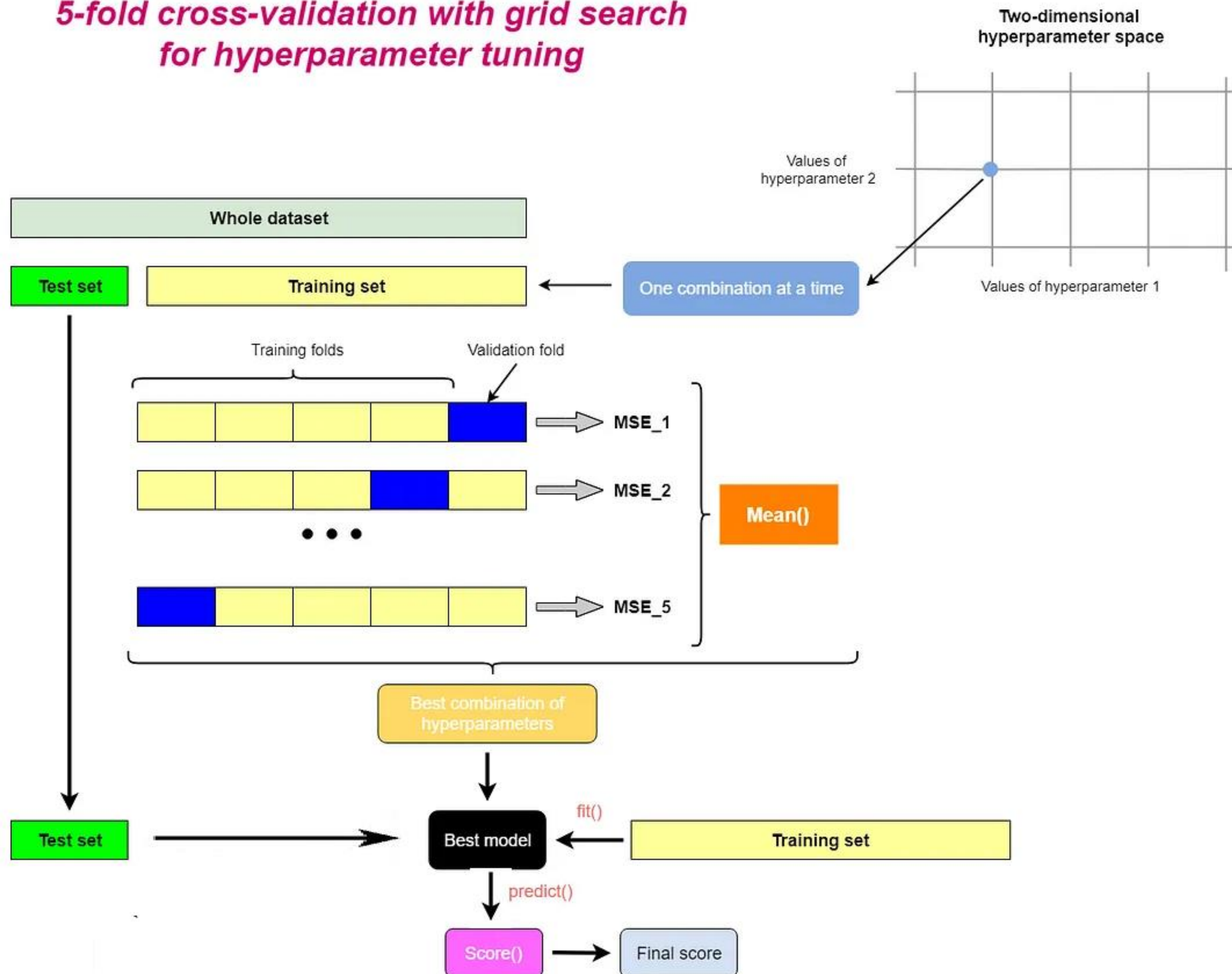
Hyperparameter tuning with Grid Search and Bayesian Optimization

Multiple hyperparameter tuning

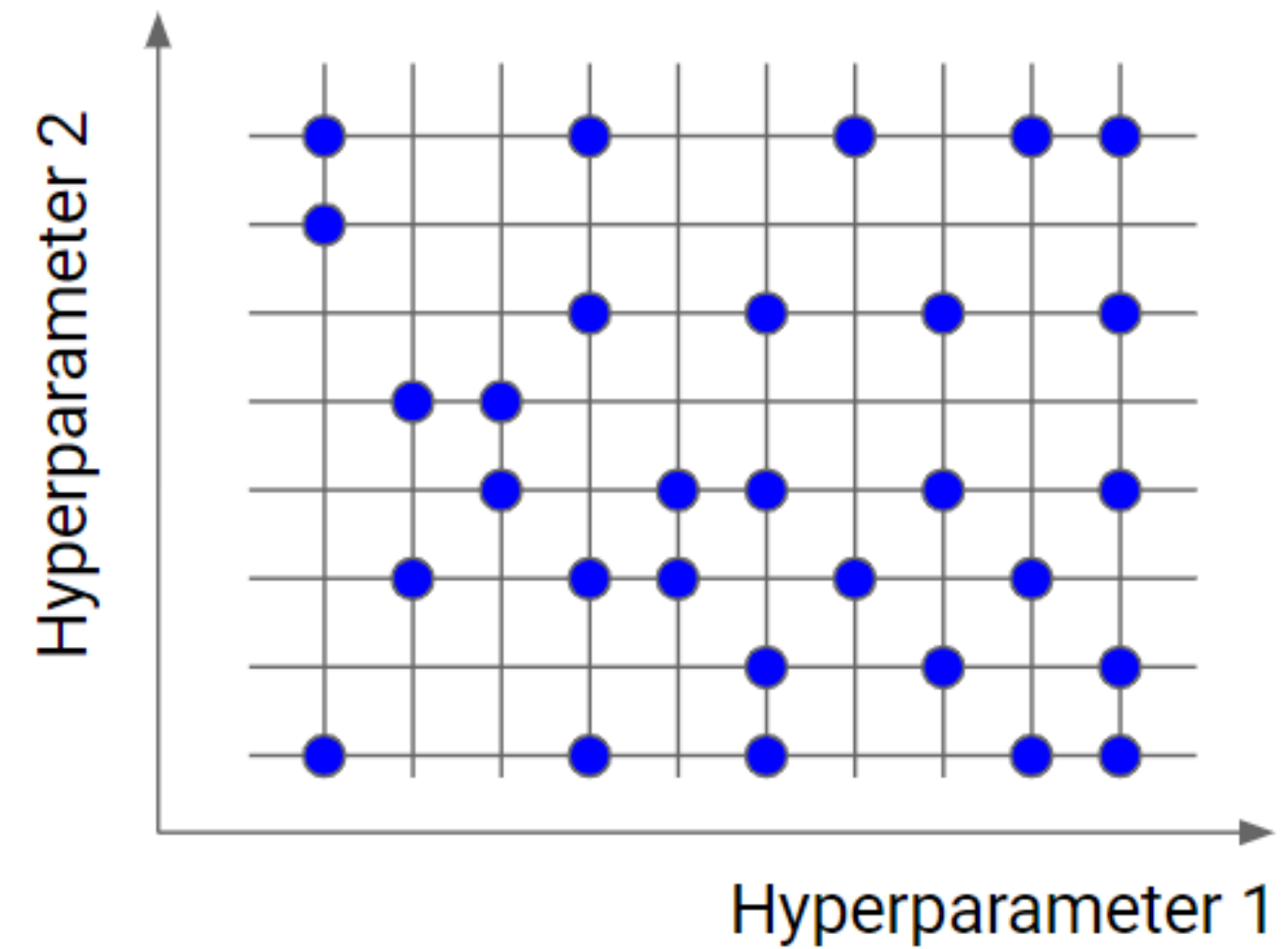
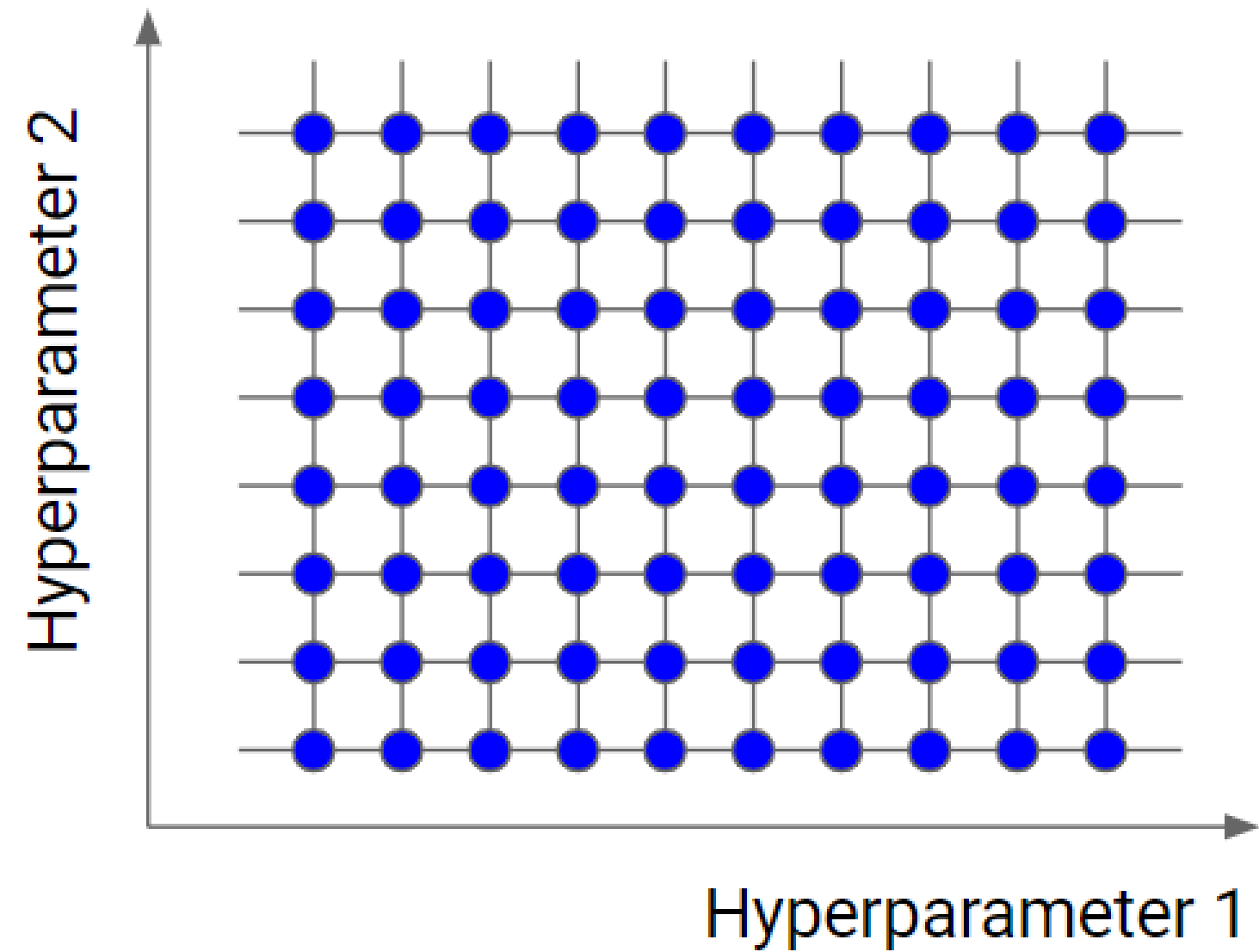
- kNN – k and weight
 - $k=\{1,3,5,7,9\}$
 - $\text{weight}=\{\text{"uniform"}, \text{"distance"}\}$
- K-Fold CV to be executed $5 \times 2 = 10$ times
- GridSearchCV



5-fold cross-validation with grid search for hyperparameter tuning



GridSearchCV & RandomSearchCV



Which option when? Dos and don'ts

- Regular machine learning
 - 1 hyperparam - K-Fold CV
 - Multiple hyperparams - GridSearchCV, RandomCV
- What about deep learning?
 - Lot's of data, lengthy training process
 - Training neural network K-times CV is not feasible
- KerasTuner, Hyperopt, Optuna for both regular & deep learning

Why not Grid search

- Too many eval - Objective function eval is expensive
 - Grid search is completely uninformed about past eval
- 3 spaces to be aware of:
 - Feature space
 - Parameter space
 - Hyperparameter space

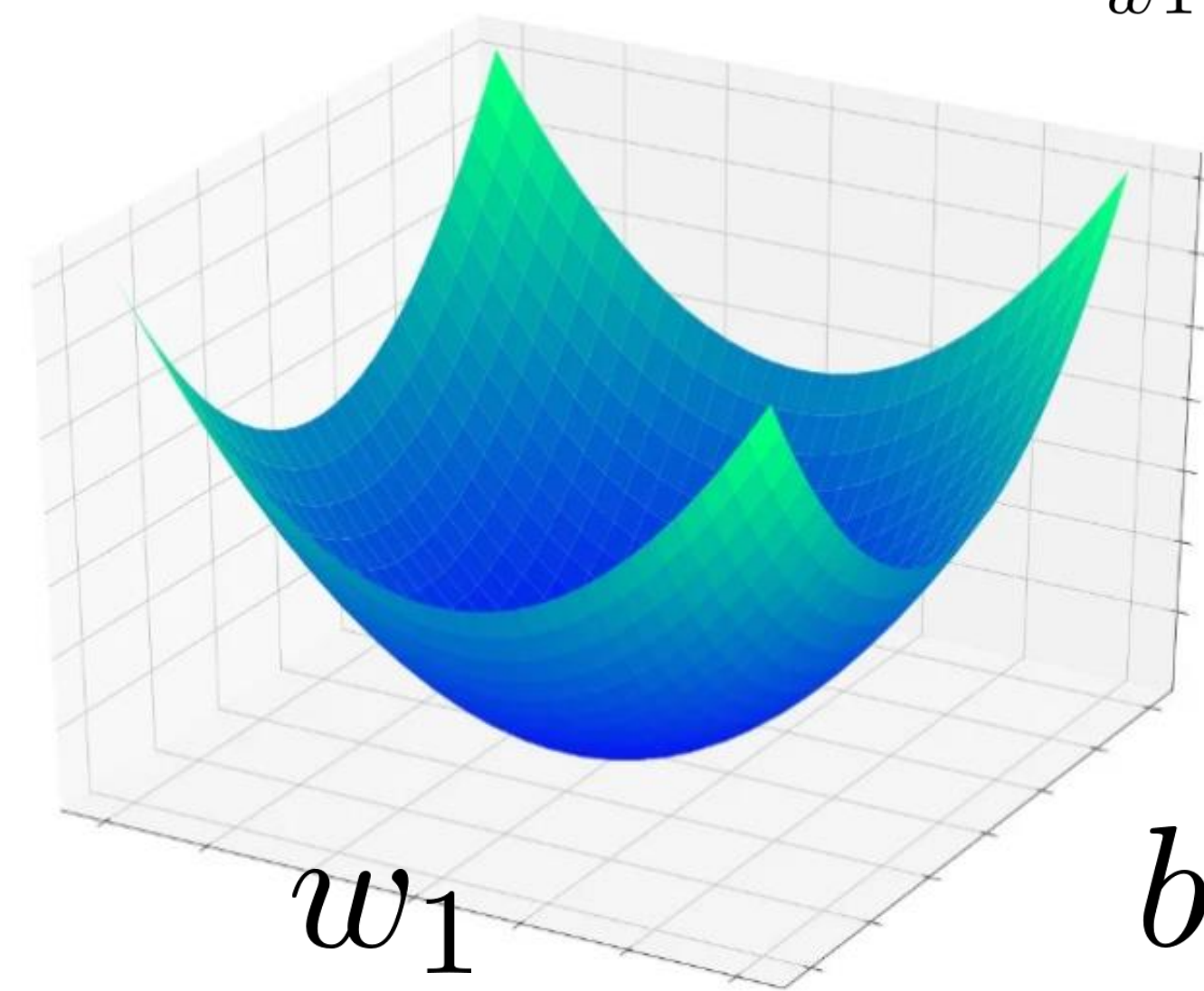
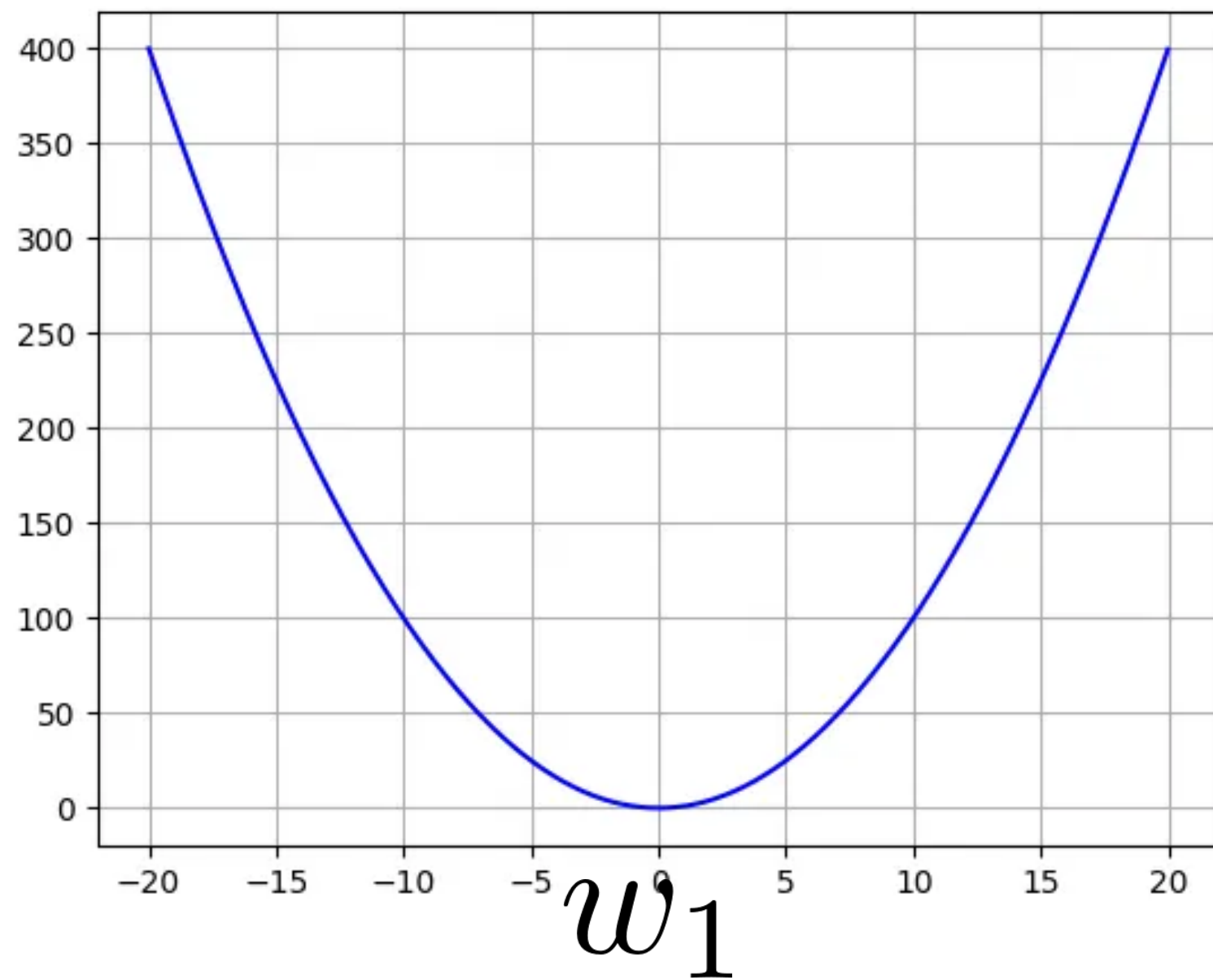
Understanding parameter space

- Objective function is evaluated in parameter space

$$\hat{y}_i = x_i^T w + b \quad w \in \mathbb{R}^d$$

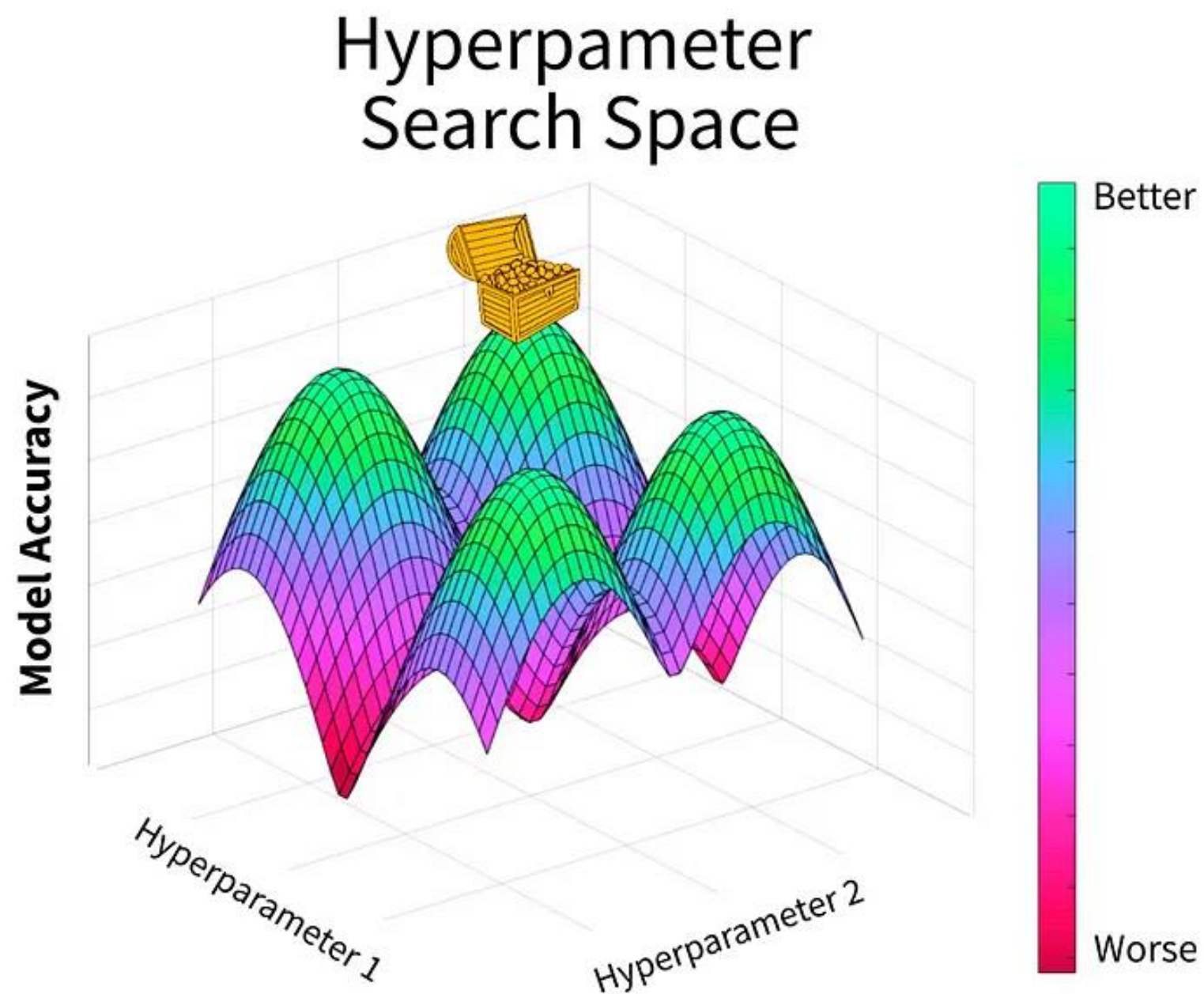
$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

$$\text{Objective} = \arg \min_{w, b} \text{MSE}$$



Why not Grid search

- Too many eval - Objective function eval is expensive
- Grid search is completely uninformed about past eval



• GridSearch happens in hyperparameter space

• There's also a hypothesis space. Don't worry about it

Hyperparameter tuning with Bayesian Optimization

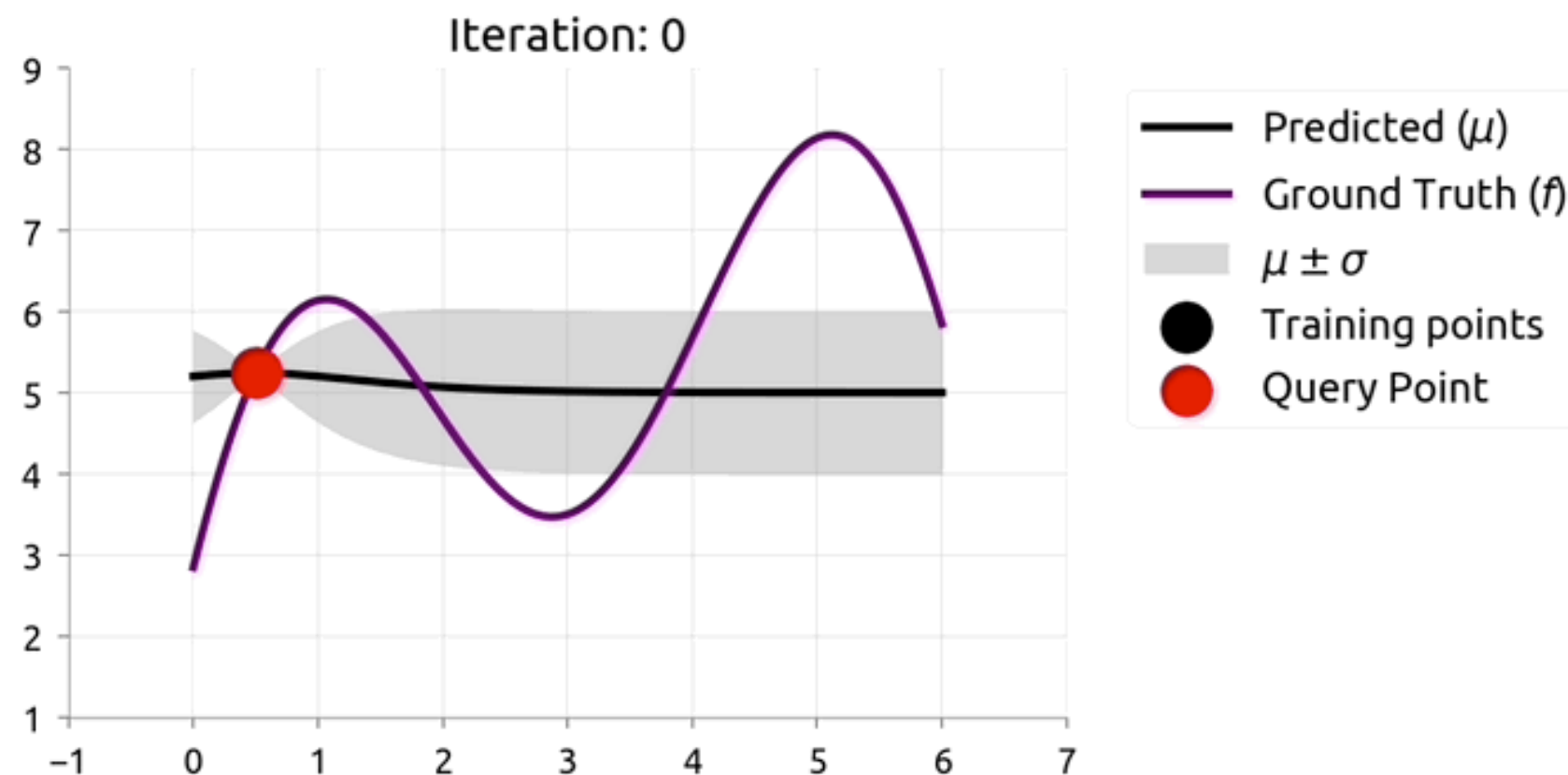
Surrogate function

Posterior

Likelihood

Prior

$$P(\text{score}|\text{hyperparam}) = \frac{P(\text{hyperparam}|\text{score})P(\text{score})}{P(\text{hyperparam})}$$



- Prior, Likelihood & Posterior
- Posterior evaluated with surrogate function
- Cheap to evaluate

- Surrogate function modeled as a Gaussian Process (GP)
- GP gives a distribution of functions
- Very good at interpolating, not good at extrapolating

Types of Cross Validation

- Hold out (Train-test split)
- K-Fold Cross Validation
- Stratified K-Fold
- Repeated K-Fold
- Nested k-Fold
- Leave One Out (LOOCV)
- Let's look at others in lab





Multiclass kNN & kNN Regression

Multi class kNN

- Majority v/s plurality
- For 3 class
 - Majority = 50%+
 - Plurality = 33.33%+


A
y:  

Majority vote: 

Plurality vote: 

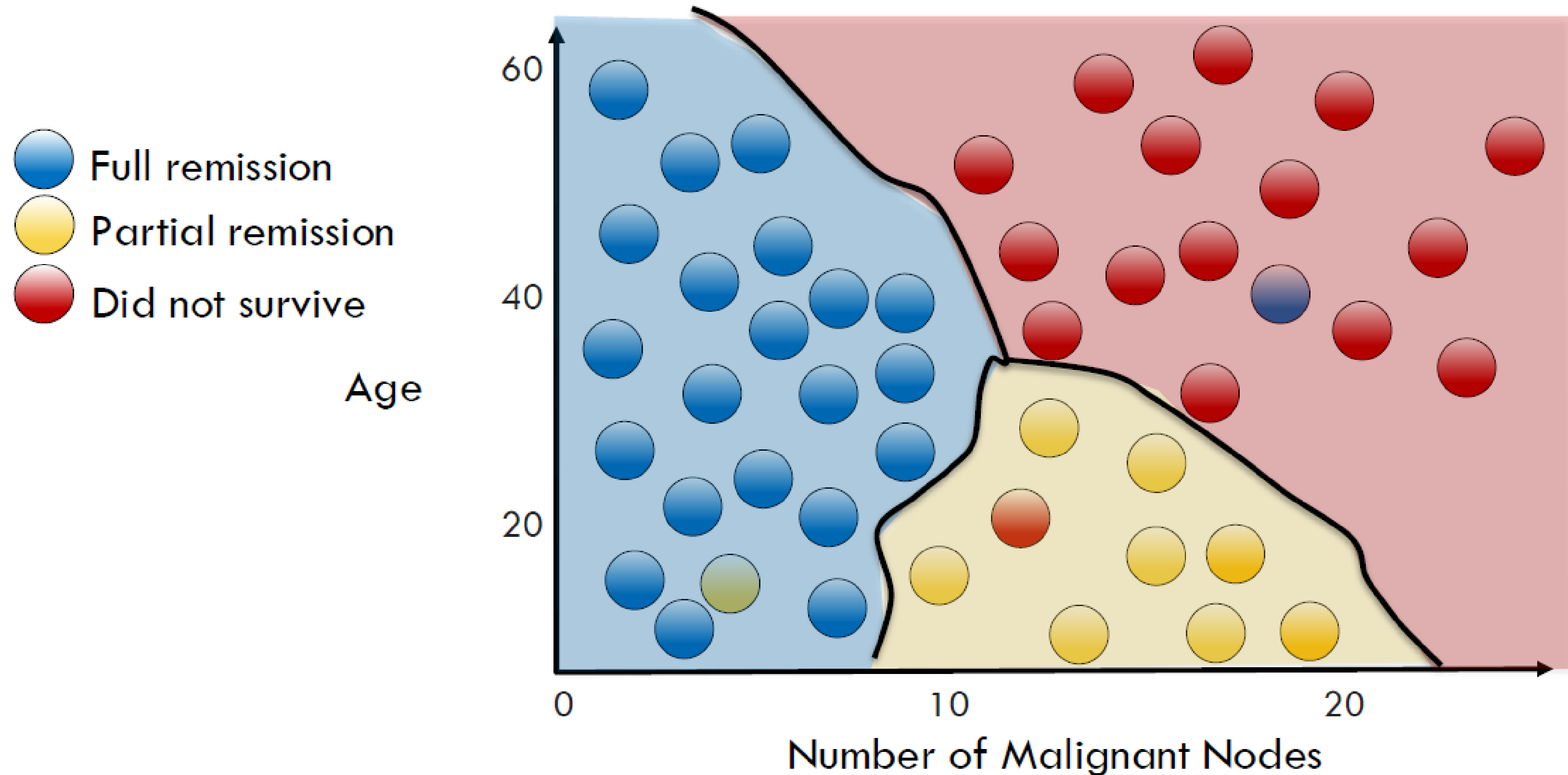
B
y:   

Majority vote: None

Plurality vote: 

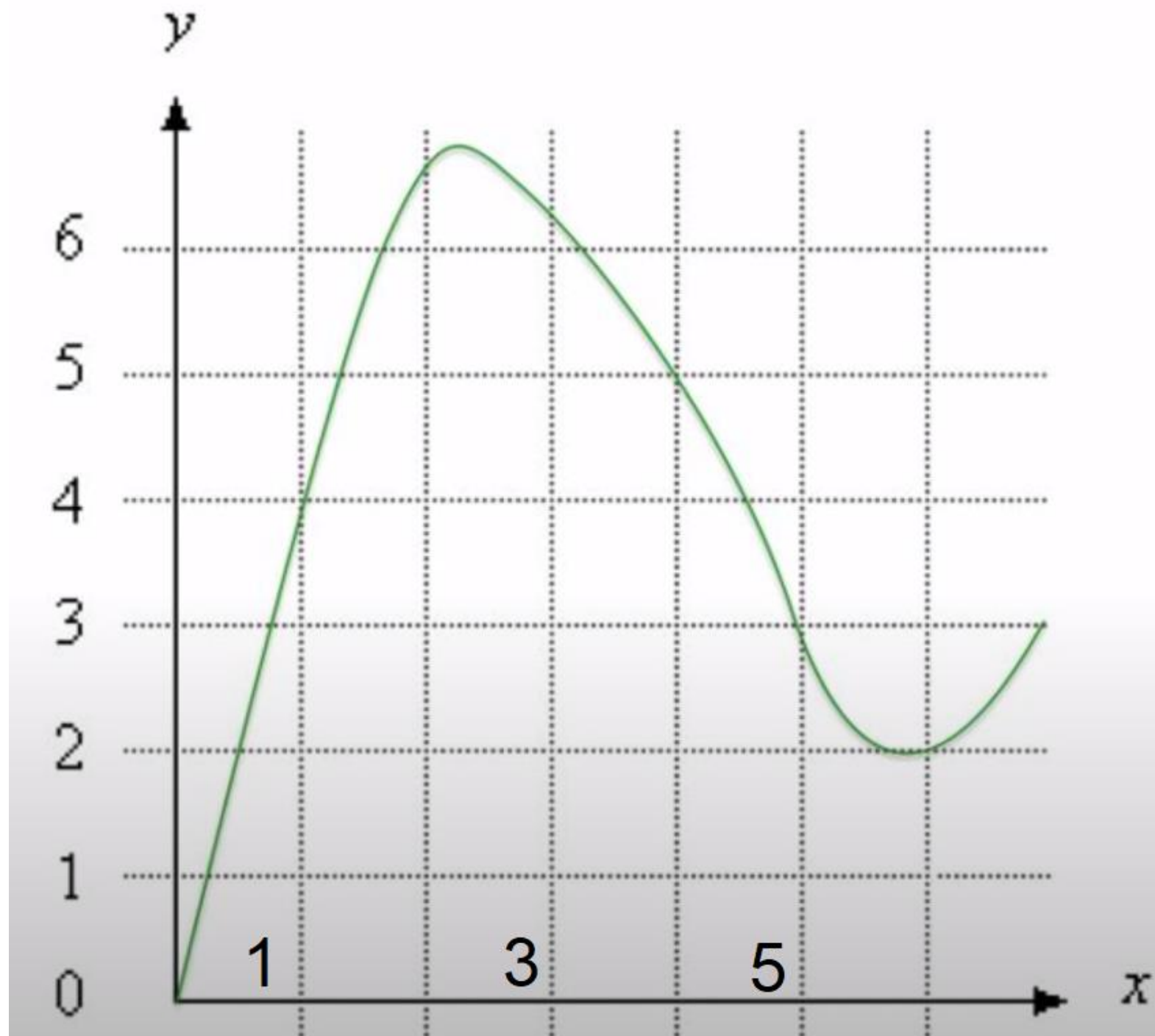
Multi class kNN decision boundary

$K = 5$

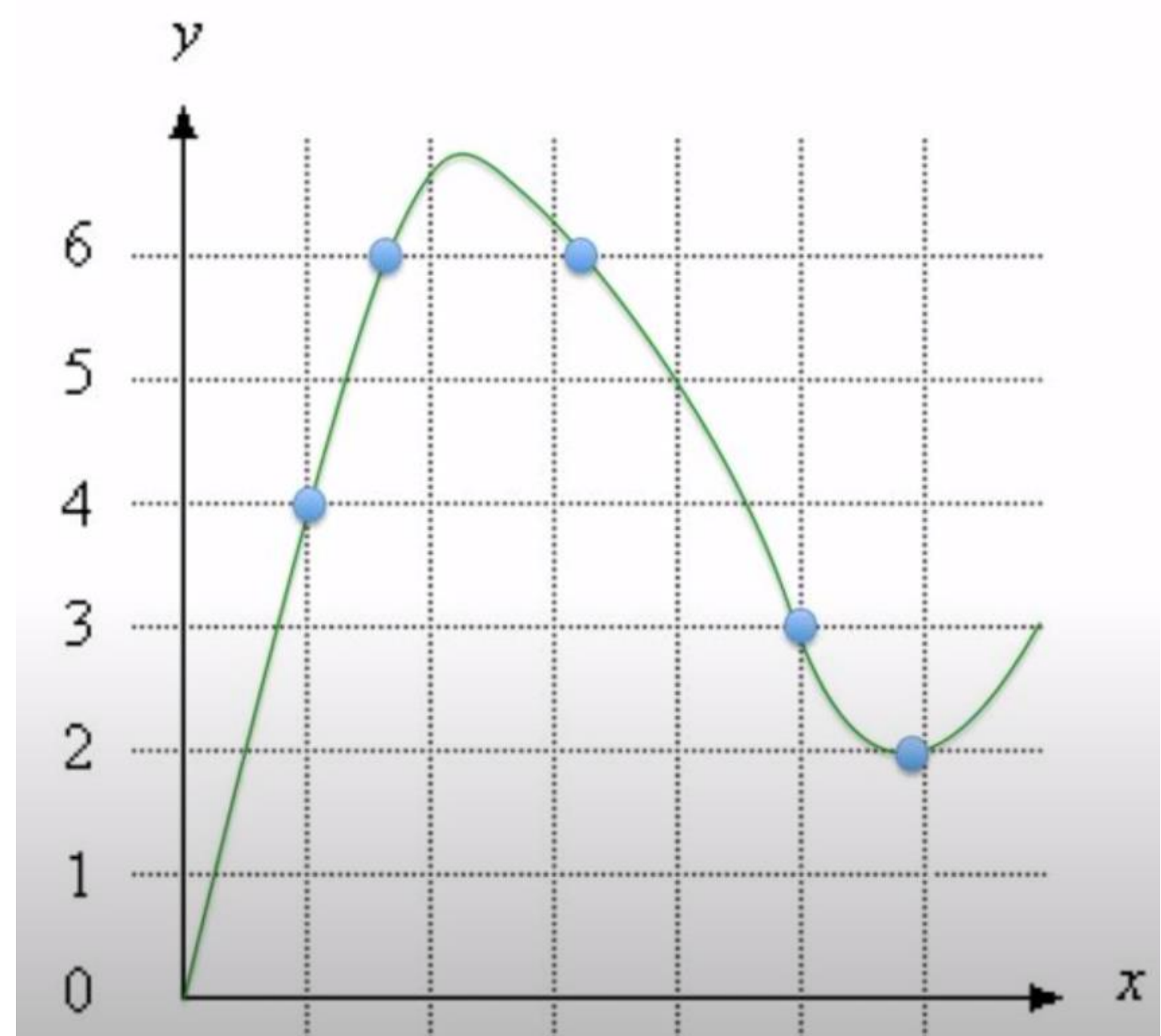


kNN Regression

Real function

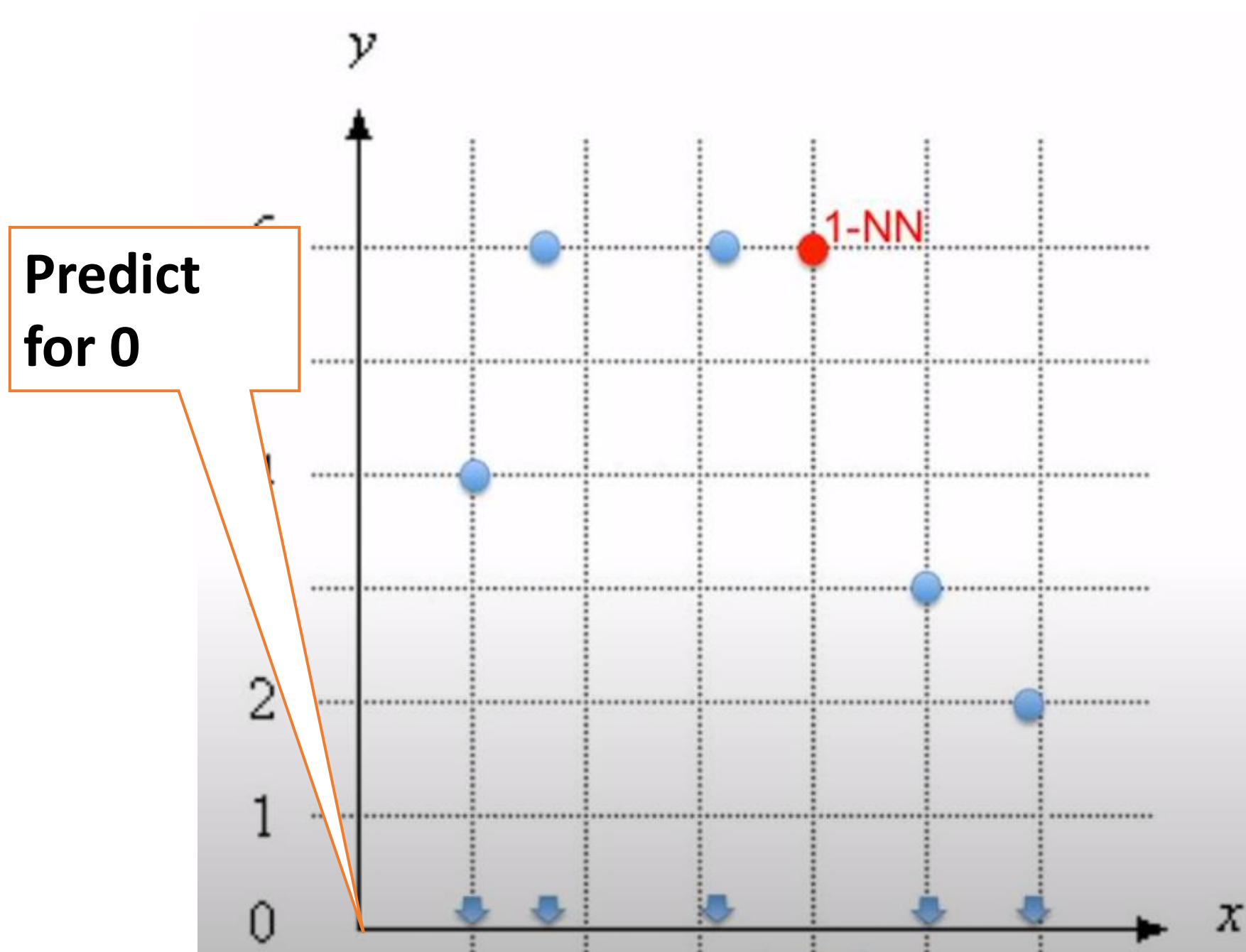


Actual data

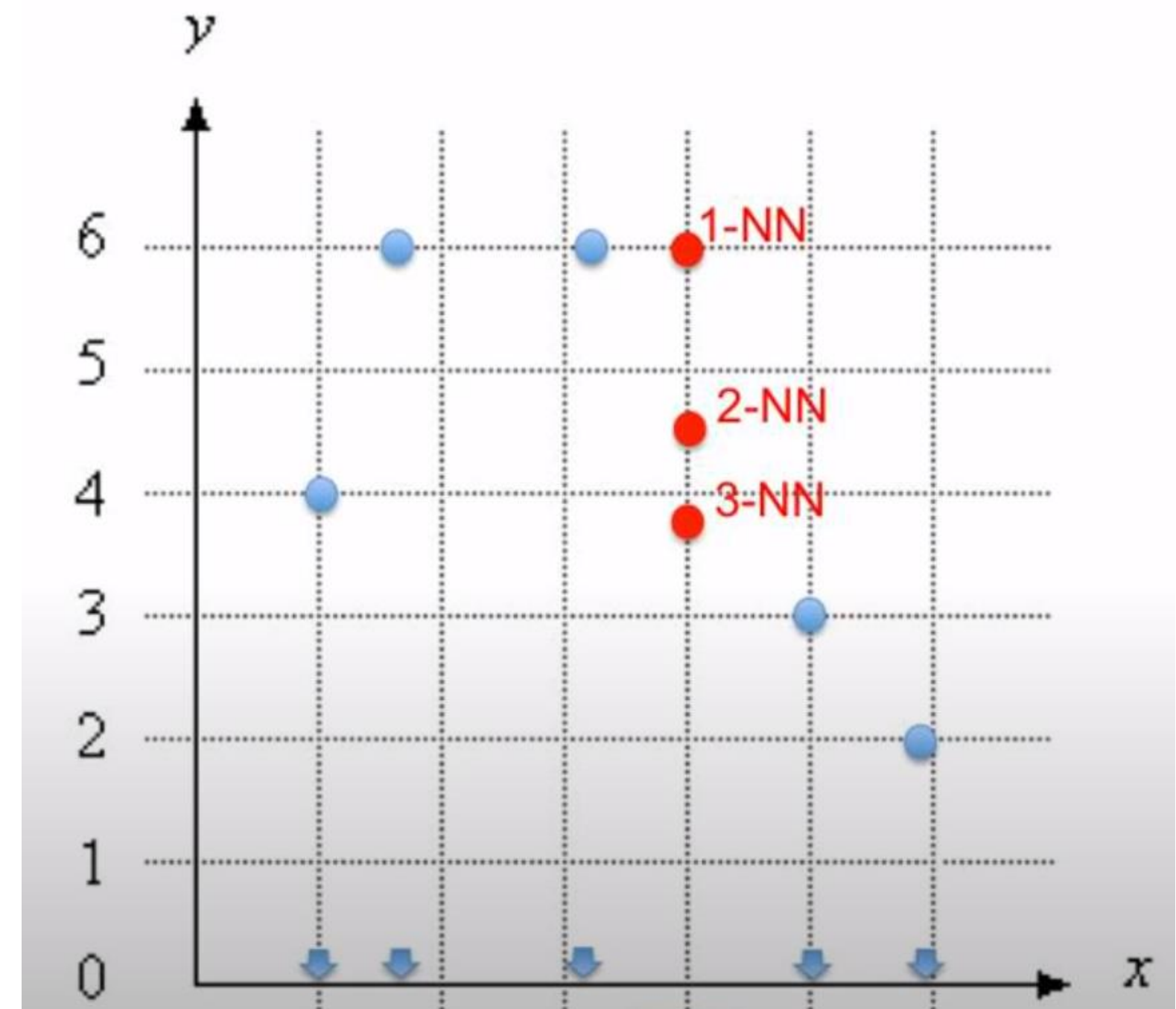


kNN Regression(contd.)

1-NN Regression

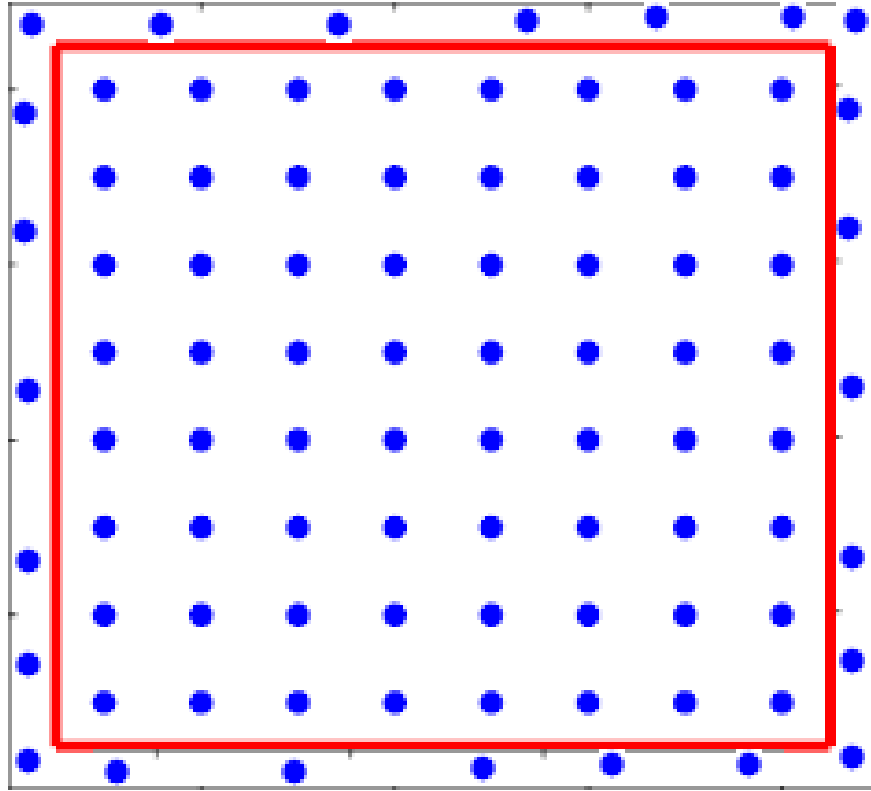


2-NN, 3-NN Regression



- kNN Regression good at interpolation bad for extrapolation

Curse of Dimensionality



- $P(\text{point} < 0.01 \text{ units from border}) =$
 - $1 - P(\text{point inside } 0.99)$
 - $P(\text{point inside } 0.99) = 0.99 * 0.99$
 - $1 - 0.9801 = 0.0199$

- $P(\text{point} < 0.01 \text{ units from border}) =$
 $= 1 - (0.99)^3 = 1 - 0.9703 = 0.0297$
- For 1000 dimensions, $P(\text{point in } 0.01 \text{ border}) =$
 $= 1 - (0.99)^{1000} = 1 - 0.00004317 = 0.99995683$



QUESTIONS



Thank You!