K-Nearest Neighbor

Supervised Learning

- Given training data $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$

Supervised Learning

- Given training data $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$
- N input/output pairs; $\mathbf{x}_i$ - input, $\mathbf{y}_i$ - output/label

Supervised Learning

- Given training data $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$
- N input/output pairs; $\mathbf{x}_i$ - input, $\mathbf{y}_i$ - output/label
- $\mathbf{x}_i$ is a vector consisting of D **features**
  - Also called **attributes** or **dimensions**
  - Features can be discrete or continuous
  - x_{im} denotes the m-th feature of $\mathbf{x}_i$

Supervised Learning

- Given training data $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$
- N input/output pairs; $\mathbf{x}_i$ - input, $\mathbf{y}_i$ - output/label
- $\mathbf{x}_i$ is a vector consisting of D **features**
    - Also called **attributes** or **dimensions**
    - Features can be discrete or continuous
    - x_{im} denotes the m-th feature of $\mathbf{x}_i$
- Forms of the output:
    - $\mathbf{y}_i \in \{1, \ldots, C\}$ for classification; a discrete variable
    - $\mathbf{y}_i \in \mathbb{R}$ for regression; a continuous (real-valued) variable

Supervised Learning

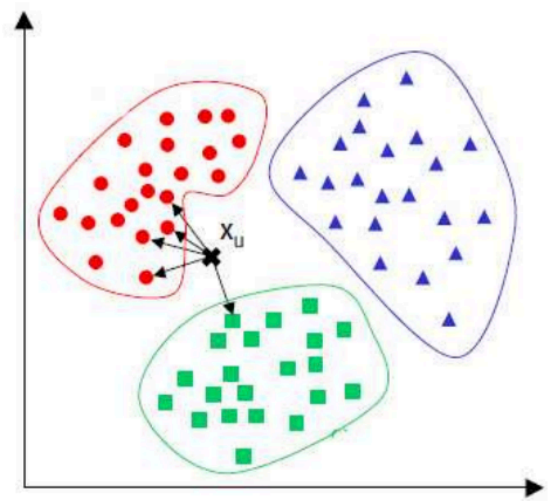
- Given training data $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$
- N input/output pairs; $\mathbf{x}_i$ - input, $\mathbf{y}_i$ - output/label
- $\mathbf{x}_i$ is a vector consisting of D **features**
    - Also called **attributes** or **dimensions**
    - Features can be discrete or continuous
    - x_{im} denotes the m-th feature of $\mathbf{x}_i$
- Forms of the output:
    - $\mathbf{y}_i \in \{1, \ldots, C\}$ for classification; a discrete variable
    - $\mathbf{y}_i \in \mathbb{R}$ for regression; a continuous (real-valued) variable
- **Goal:** predict the output $\mathbf{y}$ for an **unseen** test example $\mathbf{x}$

K-Nearest Neighbor

- Given training data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ and a test point
- Prediction Rule: Look at the K most similar training examples

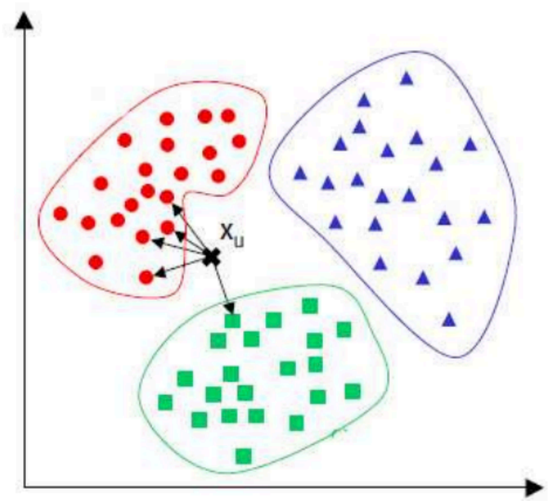
K-Nearest Neighbor

- Given training data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ and a test point
- Prediction Rule: Look at the K most similar training examples



K-Nearest Neighbor

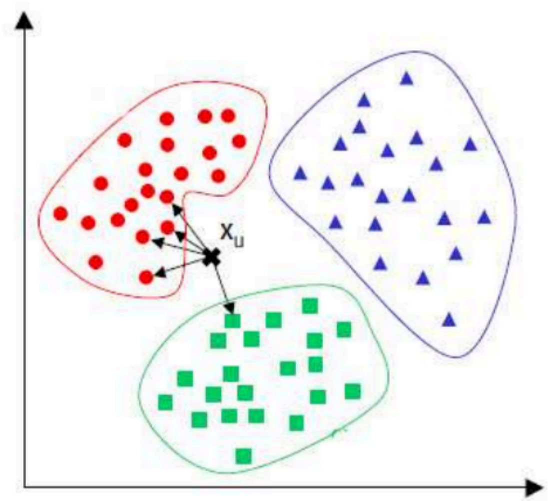
- Given training data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ and a test point
- Prediction Rule: Look at the K most similar training examples



  - For classification: assign the majority class label (majority voting)
  - For regression: assign the average response

K-Nearest Neighbor

- Given training data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ and a test point
- Prediction Rule: Look at the K most similar training examples



  - For classification: assign the majority class label (majority voting)
  - For regression: assign the average response
- The algorithm requires:
  - Parameter K: number of nearest neighbors to look for
  - Distance function: To compute the similarities between examples

K-Nearest Neighbor Algorithm

- Compute the test point's distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the K nearest neighbors
- Use majority rule (for classification) or averaging (for regression)

K-Nearest Neighbor Algorithm

- Compute the test point's distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the K nearest neighbors
- Use majority rule (for classification) or averaging (for regression)

Note: K-Nearest Neighbors is called a *non-parametric* method

- Unlike other supervised learning algorithms, K-Nearest Neighbors doesn't learn an explicit mapping f from the training data

K-Nearest Neighbor Algorithm

- Compute the test point's distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the K nearest neighbors
- Use majority rule (for classification) or averaging (for regression)

Note: K-Nearest Neighbors is called a *non-parametric* method

- Unlike other supervised learning algorithms, K-Nearest Neighbors doesn't learn an explicit mapping f from the training data
- It simply uses the training data at the test time to make predictions

K-NN: Compute Distance

- The K-NN algorithm requires computing distances of the test example from each of the training examples

K-NN: Compute Distance

- The K-NN algorithm requires computing distances of the test example from each of the training examples
- Several ways to compute distances
- The choice depends on the type of the features in the data

K-NN: Compute Distance

- The K-NN algorithm requires computing distances of the test example from each of the training examples
- Several ways to compute distances
- The choice depends on the type of the features in the data
- Real-valued features ($\mathbf{x}_i \in \mathbb{R}^D$): Euclidean distance is commonly used

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{m=1}^{D}(x_{im} - x_{jm})^2} = \sqrt{||\mathbf{x}_i||^2 + ||\mathbf{x}_j||^2 - 2\mathbf{x}_i^T \mathbf{x}_j}$$

K-NN: Compute Distance

- The K-NN algorithm requires computing distances of the test example from each of the training examples
- Several ways to compute distances
- The choice depends on the type of the features in the data
- Real-valued features ($\mathbf{x}_i \in \mathbb{R}^D$): Euclidean distance is commonly used

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{||\mathbf{x}_i||^2 + ||\mathbf{x}_j||^2 - 2\mathbf{x}_i^T \mathbf{x}_j}$$

- Generalization of the distance between points in 2 dimensions

K-NN: Compute Distance

- The K-NN algorithm requires computing distances of the test example from each of the training examples
- Several ways to compute distances
- The choice depends on the type of the features in the data
- Real-valued features ($\mathbf{x}_i \in \mathbb{R}^D$): Euclidean distance is commonly used

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{||\mathbf{x}_i||^2 + ||\mathbf{x}_j||^2 - 2\mathbf{x}_i^T \mathbf{x}_j}$$

- Generalization of the distance between points in 2 dimensions
- $||\mathbf{x}_i|| = \sqrt{\sum_{m=1}^{D} x_{im}^2}$ is called the **norm** of $\mathbf{x}_i$
  - Norm of a vector $\mathbf{x}$ is also its **length**

K-NN: Compute Distance

- The K-NN algorithm requires computing distances of the test example from each of the training examples
- Several ways to compute distances
- The choice depends on the type of the features in the data
- Real-valued features ($\mathbf{x}_i \in \mathbb{R}^D$): Euclidean distance is commonly used

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{m=1}^{D}(x_{im} - x_{jm})^2} = \sqrt{||\mathbf{x}_i||^2 + ||\mathbf{x}_j||^2 - 2\mathbf{x}_i^T \mathbf{x}_j}$$

- Generalization of the distance between points in 2 dimensions
- $||\mathbf{x}_i|| = \sqrt{\sum_{m=1}^{D} x_{im}^2}$ is called the **norm** of $\mathbf{x}_i$
  - Norm of a vector $\mathbf{x}$ is also its **length**

K-NN: Feature Normalization

- Note: Features should be on the same scale
- Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize

K-NN: Feature Normalization

- Note: Features should be on the same scale
- Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize
- One way is:
  - Replace x_{im} by $z_{im} = \frac{(x_{im} - \bar{x_m})}{\sigma_m}$ (make them zero mean, unit variance)

K-NN: Feature Normalization

- Note: Features should be on the same scale
- Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize
- One way is:
  - Replace x_{im} by $z_{im} = \frac{(x_{im} - \bar{x_m})}{\sigma_m}$ (make them zero mean, unit variance)
  - $\bar{x_m} = \frac{1}{N} \sum_{i=1}^{N} x_{im}$: empirical mean of m^{th} feature

K-NN: Feature Normalization

- Note: Features should be on the same scale
- Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize
- One way is:
  - Replace x_{im} by $z_{im} = \frac{(x_{im} - \bar{x_m})}{\sigma_m}$ (make them zero mean, unit variance)
  - $\bar{x_m} = \frac{1}{N}\sum_{i=1}^{N} x_{im}$: empirical mean of m^{th} feature
  - $\sigma_m^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{im} - \bar{x_m})^2$: empirical variance of m^{th} feature

K-NN: Other Distance Measure

- Binary-valued features
  - Use Hamming distance: $d(x_i, x_j) = \sum_{m=1}^{D} \mathbb{I}(x_{im} \neq x_{jm})$
  - Hamming distance counts the number of features where the two examples disagree

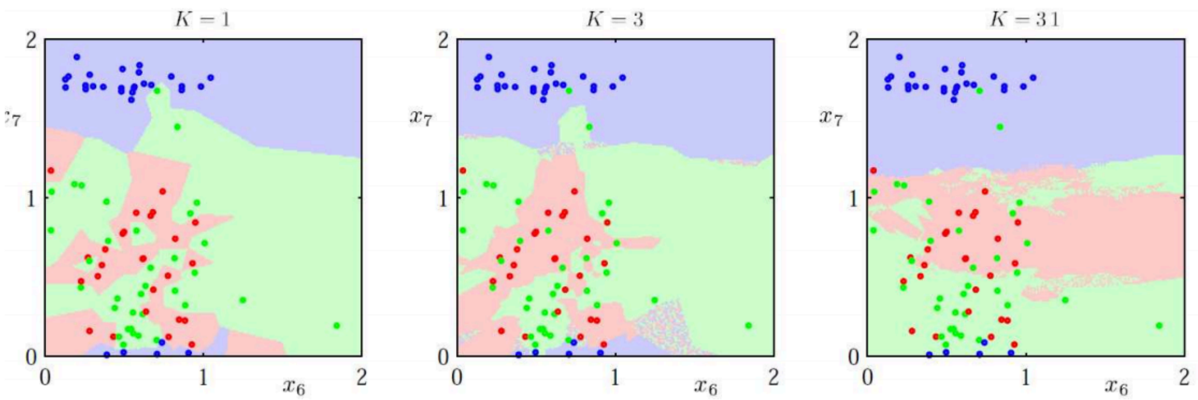
K-NN: Other Distance Measure

- Binary-valued features
  - Use Hamming distance: $d(x_i, x_j) = \sum_{m=1}^{D} \mathbb{I}(x_{im} \neq x_{jm})$
  - Hamming distance counts the number of features where the two examples disagree
- Mixed feature types (some real-valued and some binary-valued)?
  - Can use mixed distance measures
  - E.g., Euclidean for the real part, Hamming for the binary part

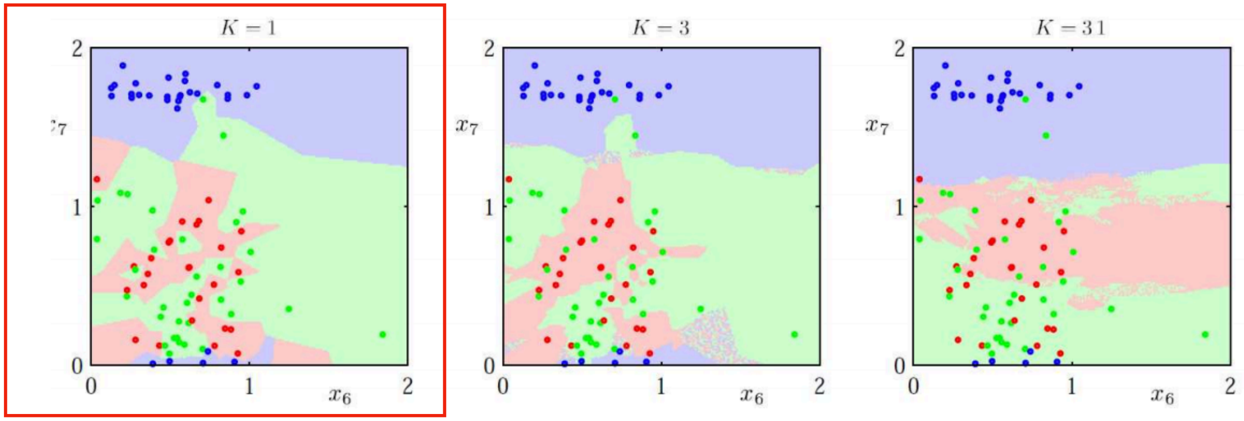
K-NN: Other Distance Measure

- Binary-valued features
  - Use Hamming distance: $d(x_i, x_j) = \sum_{m=1}^{D} \mathbb{I}(x_{im} \neq x_{jm})$
  - Hamming distance counts the number of features where the two examples disagree
- Mixed feature types (some real-valued and some binary-valued)?
  - Can use mixed distance measures
  - E.g., Euclidean for the real part, Hamming for the binary part
- Can also assign **weights** to features: $d(x_i, x_j) = \sum_{m=1}^{D} w_m d(x_{im}, x_{jm})$

K-NN: Choice of K

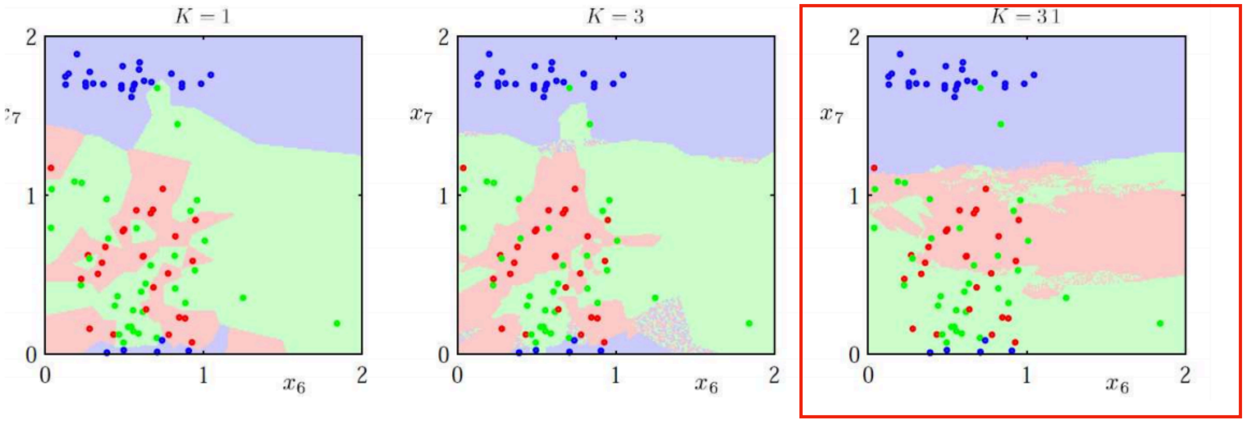


K-NN: Choice of K



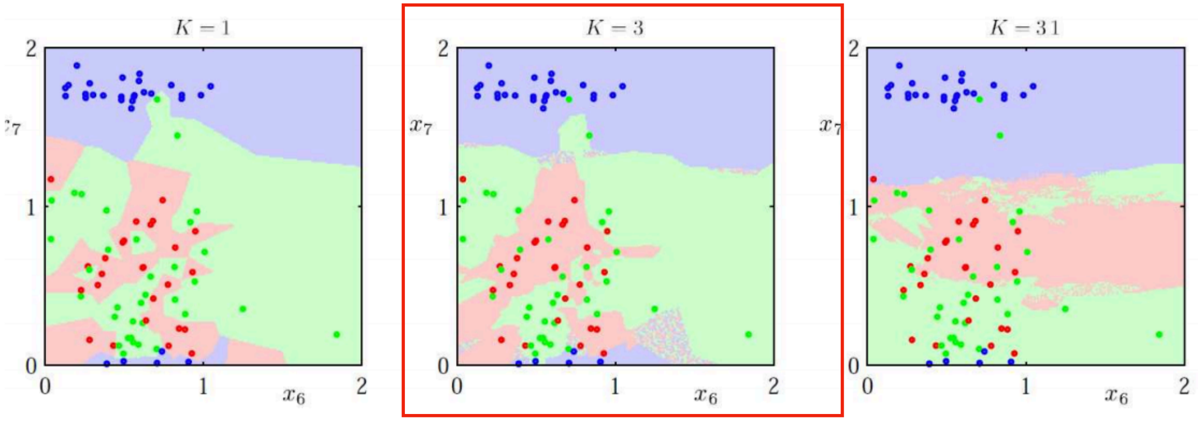
- Small K
  - Creates many small regions for each class
  - May lead to non-smooth) decision boundaries and overfit

K-NN: Choice of K



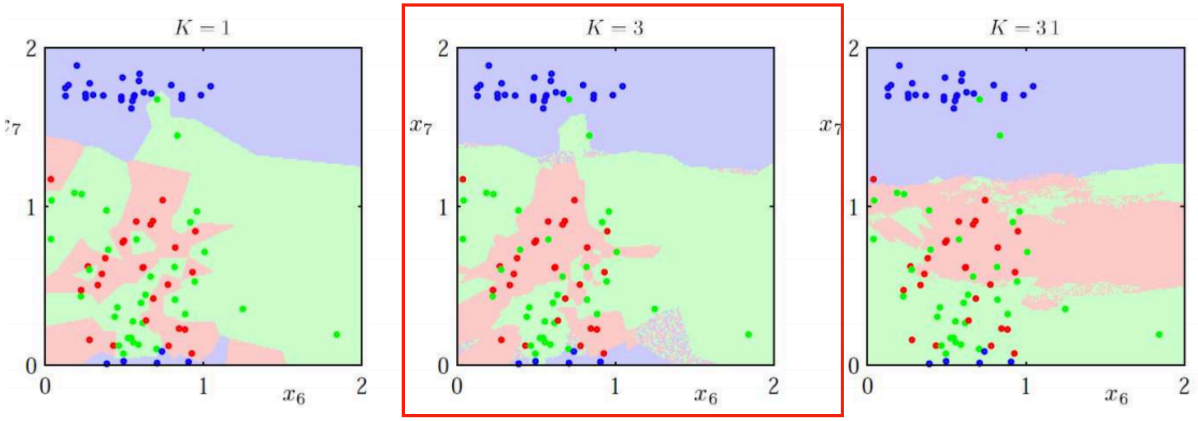
- Small K
  - Creates many small regions for each class
  - May lead to non-smooth) decision boundaries and overfit
- Large K
  - Creates fewer larger regions
  - Usually leads to smoother decision boundaries (caution: too smooth decision boundary can underfit)

K-NN: Choice of K



- Small K
  - Creates many small regions for each class
  - May lead to non-smooth) decision boundaries and overfit
- Large K
  - Creates fewer larger regions
  - Usually leads to smoother decision boundaries (caution: too smooth decision boundary can underfit)
- Choosing K
  - Often data dependent and heuristic based
  - Or using cross-validation (using some **held-out data**)

K-NN: Choice of K

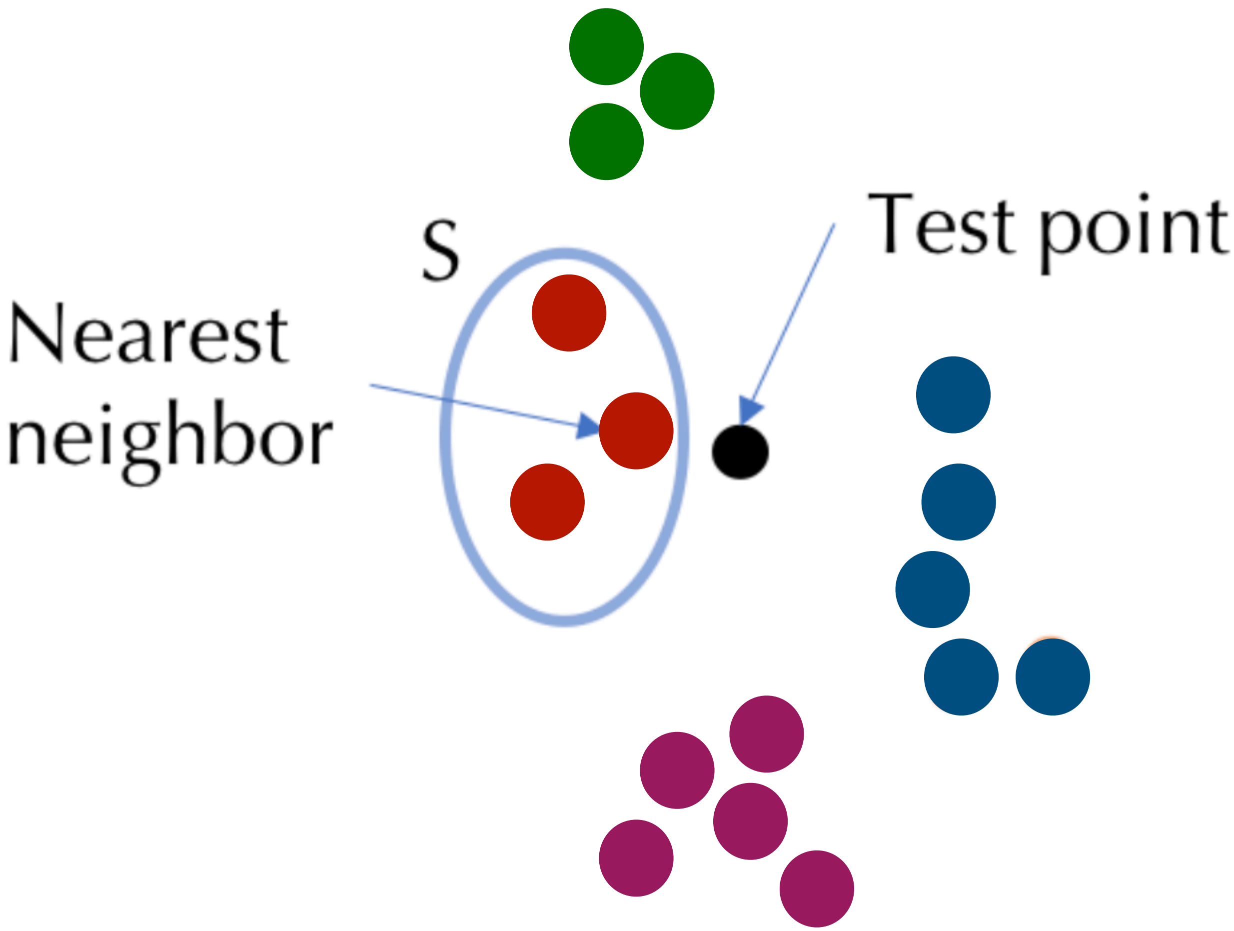


- Small K
  - Creates many small regions for each class
  - May lead to non-smooth) decision boundaries and overfit
- Large K
  - Creates fewer larger regions
  - Usually leads to smoother decision boundaries (caution: too smooth decision boundary can underfit)
- Choosing K
  - Often data dependent and heuristic based
  - Or using cross-validation (using some **held-out data**)
  - In general, a K too small or too big is bad!

K-NN: Properties

- What's nice
  - Simple and intuitive; easily implementable
  - Asymptotically consistent (a theoretical property)
    - With infinite training data and large enough K, K-NN approaches the best possible classifier (Bayes optimal)
- What's not so nice..
  - Store all the training data *in memory* even at test time
    - Can be memory intensive for large training datasets
    - An example of non-parametric, or **memory/instance-based** methods
    - Different from parametric, model-based learning models
  - Expensive at test time: $O(ND)$ computations for each test point
    - Have to search through all training data to find nearest neighbors
    - Distance computations with N training points (D features each)
  - Sensitive to noisy features
  - May perform badly in high dimensions (curse of dimensionality)
    - In high dimensions, distance notions can be counter-intuitive!

K-NN: Example



K-NN: Pseudocode

1. Calculate "$d(x, x_i)$" $i = 1, 2, \ldots, \mathbf{n}$; where **d** denotes the Euclidean distance between the points.
2. Arrange the calculated **n** Euclidean distances in non-decreasing order.
3. Let **k** be a +ve integer, take the first **k** distances from this sorted list.
4. Find those **k**-points corresponding to these **k**-distances.
5. Let $\mathbf{k}_i$ denotes the number of points belonging to the i^{th} class among **k** points i.e. $k \geq 0$
6. If $k_i > k_j \ \forall \ i \neq j$ then put x in class i.

Let (X_i, C_i) where $i = 1, 2, \ldots, n$ be data points. X_i denotes feature values & C_i denotes labels for X_i for each i.
Assuming the number of classes as 'c'
$c_i \in \{1, 2, 3, \ldots, c\}$ for all values of i

Let x be a point for which label is not known, and we would like to find the label class using k-nearest neighbor algorithms.