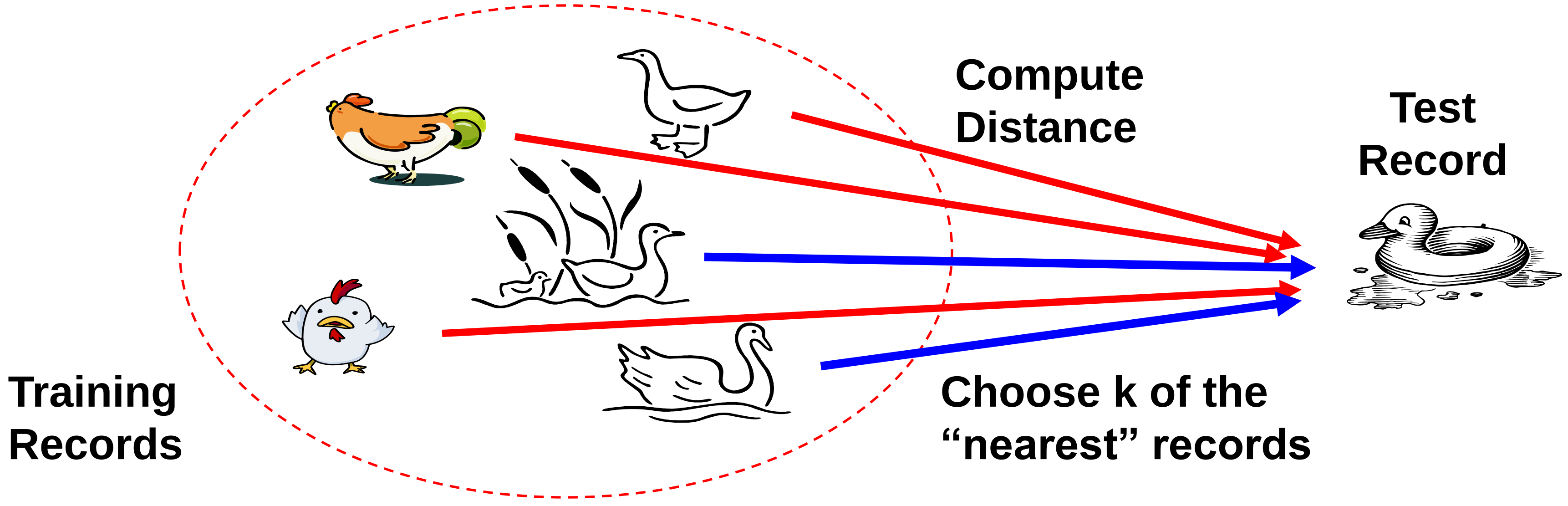
KNN

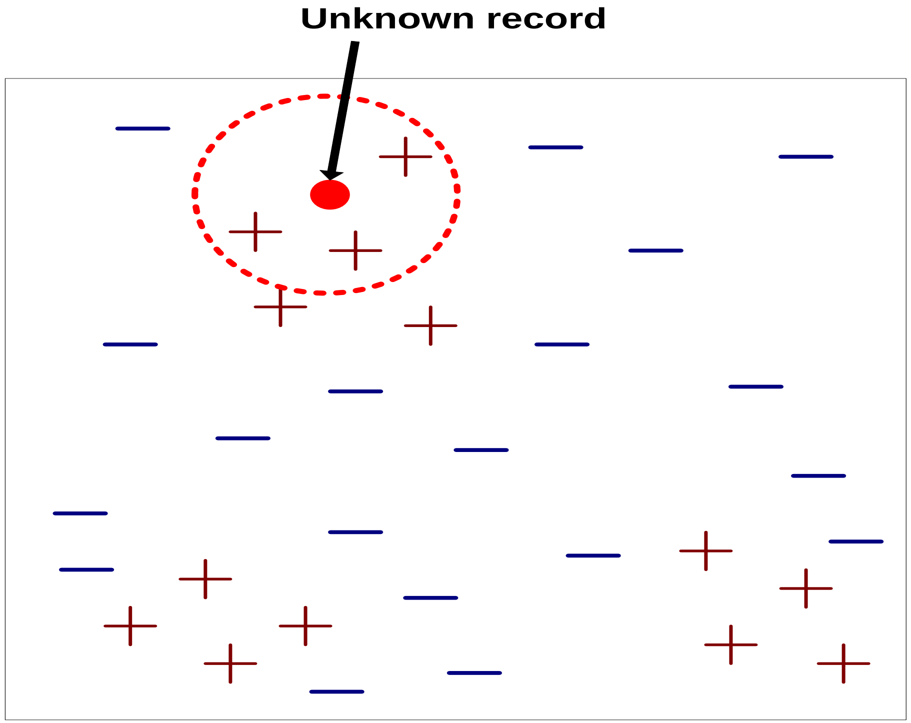
CS277

Nearest Neighbor Classifiers

- Basic idea:
  - If it walks like a duck, quacks like a duck, then it's probably a duck



Nearest-Neighbor Classifiers



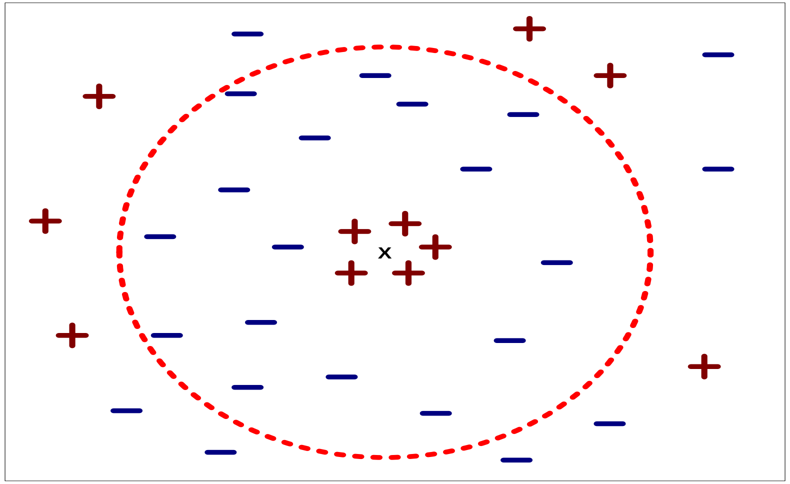
- Requires the following:
  - A set of labeled records
  - Proximity metric to compute distance/similarity between a pair of records
    - e.g., Euclidean distance
  - The value of *k*, the number of nearest neighbors to retrieve
  - A method for using class labels of K nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

How to Determine the class label of a Test Sample?

- Take the majority vote of class labels among the k-nearest neighbors
- Weight the vote according to distance
  - weight factor, $w = 1/d^2$

Nearest Neighbor Classification…

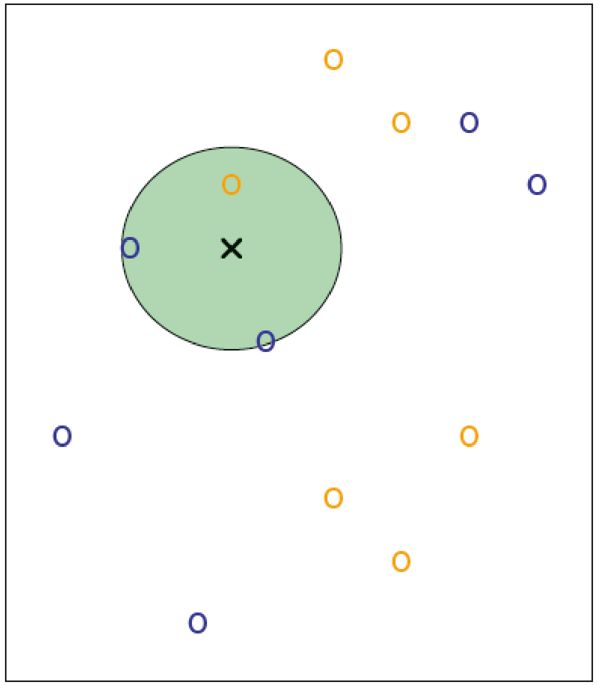
- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



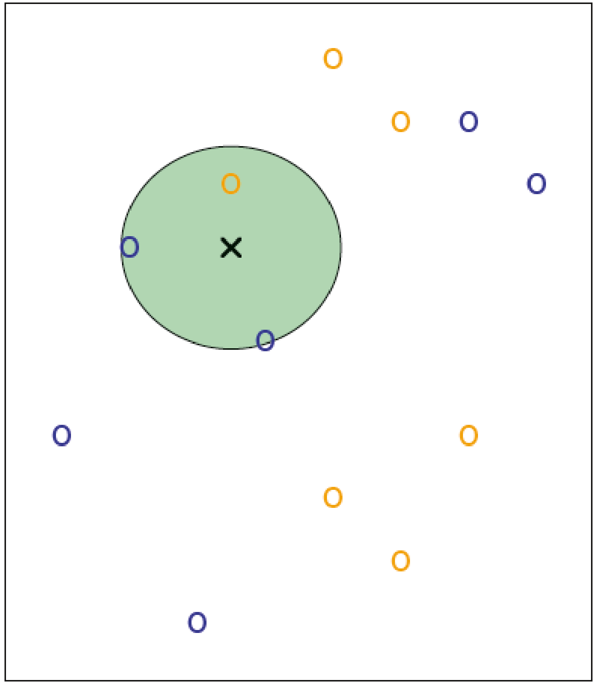
Nearest Neighbor Classification…

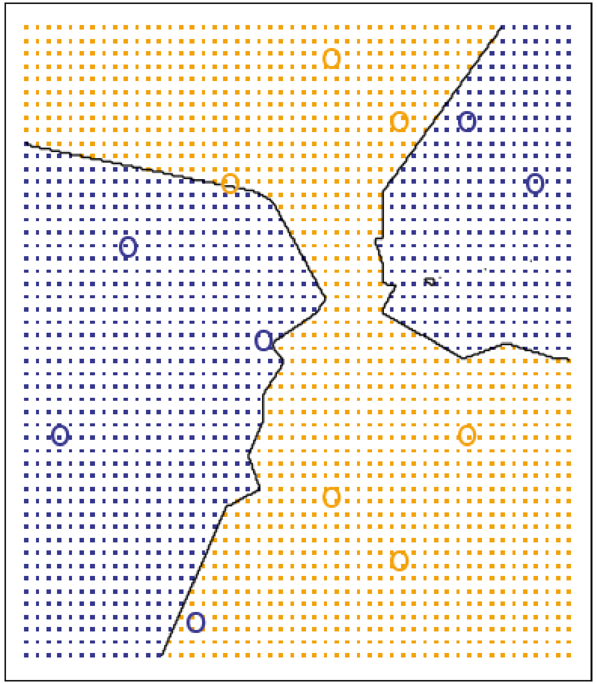
- **How to handle missing values in training and test sets?**
  - Proximity computations normally require the presence of all attributes
  - Some approaches use the subset of attributes present in two instances
    - This may not produce good results since it effectively uses different proximity measures for each pair of instances
    - Thus, proximities are not comparable

K=3

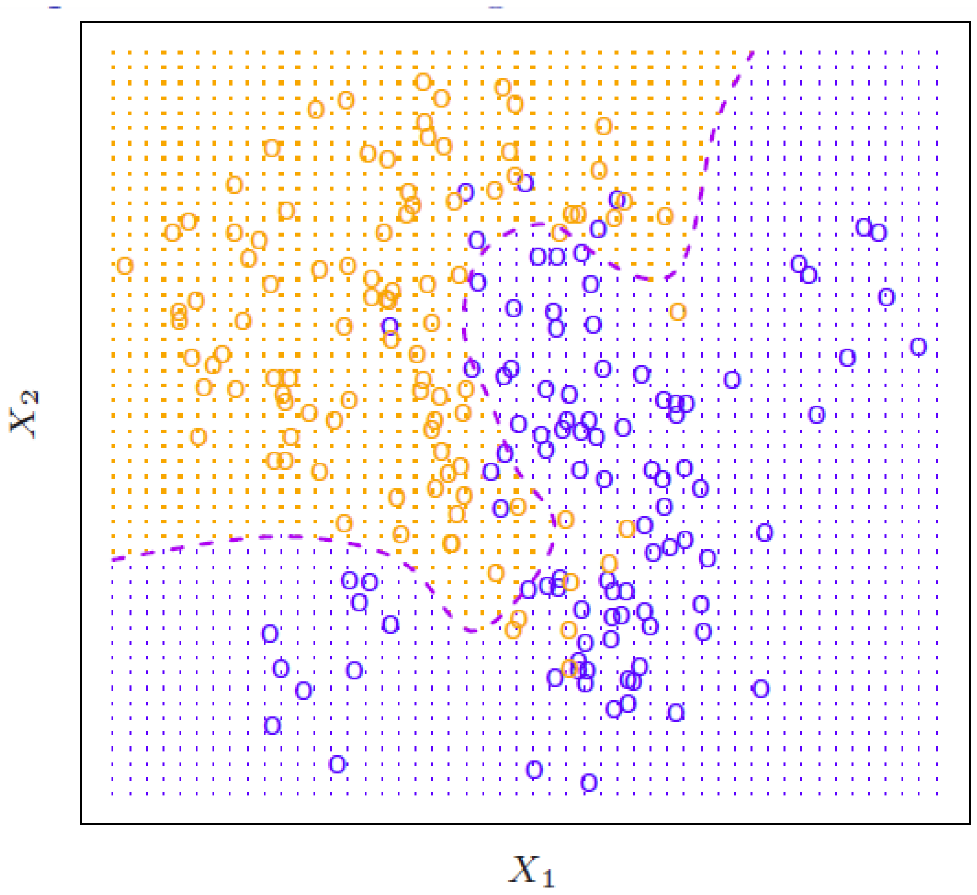


K=3

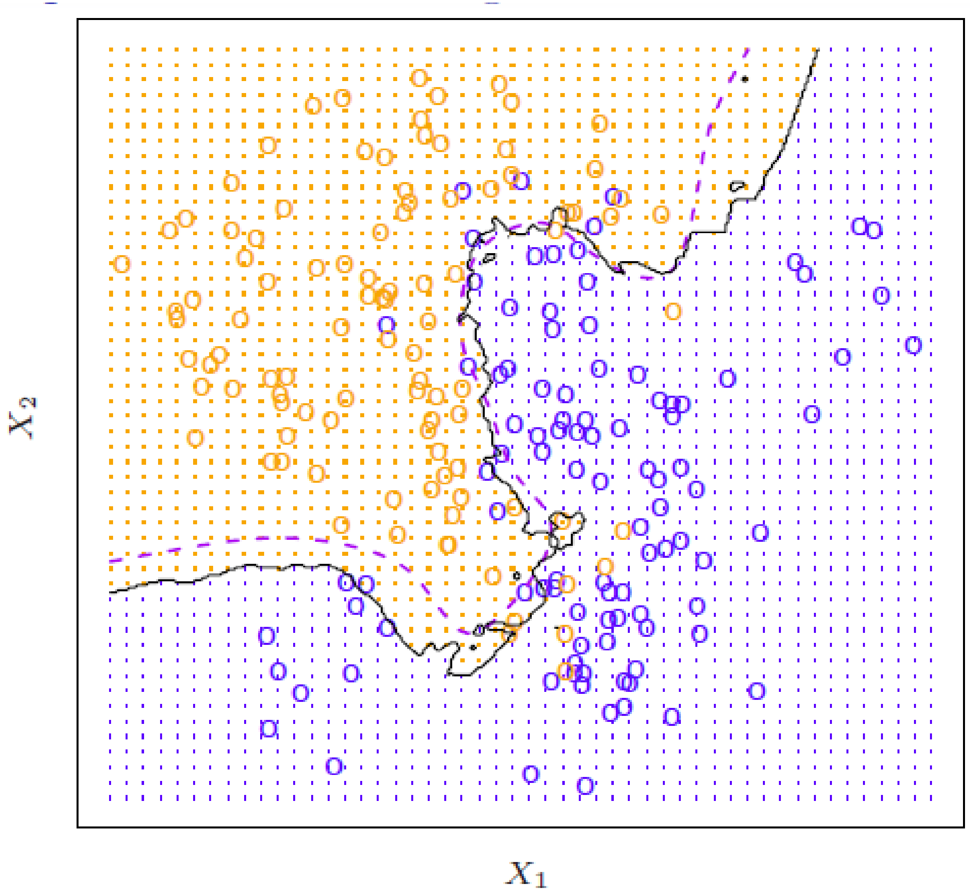




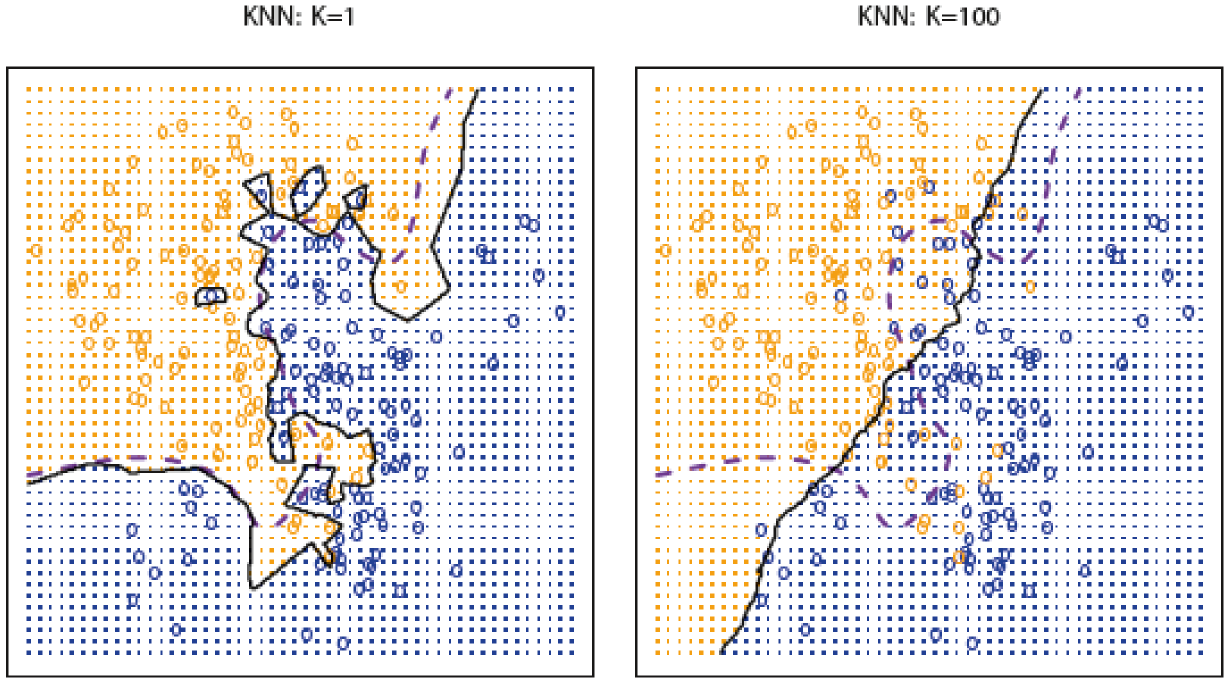
KNN



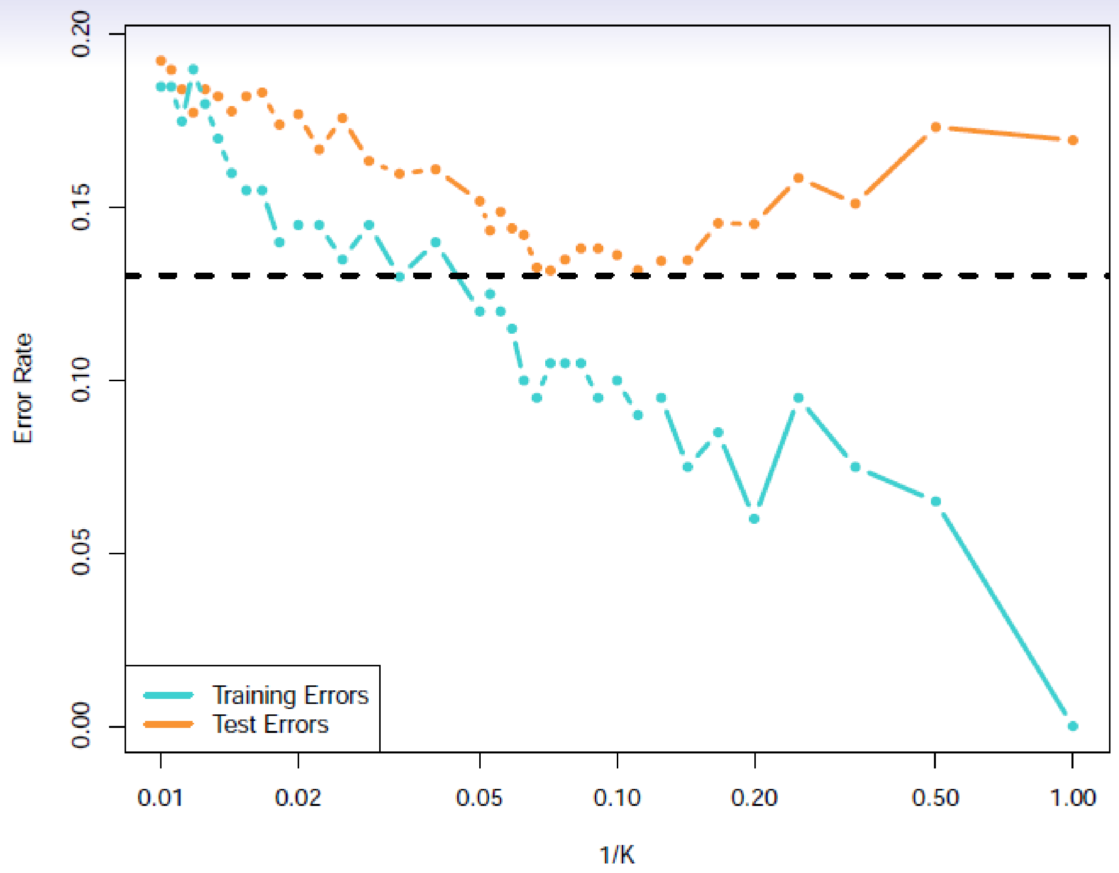
K=10



A Comparisons of K=1 vs K=100



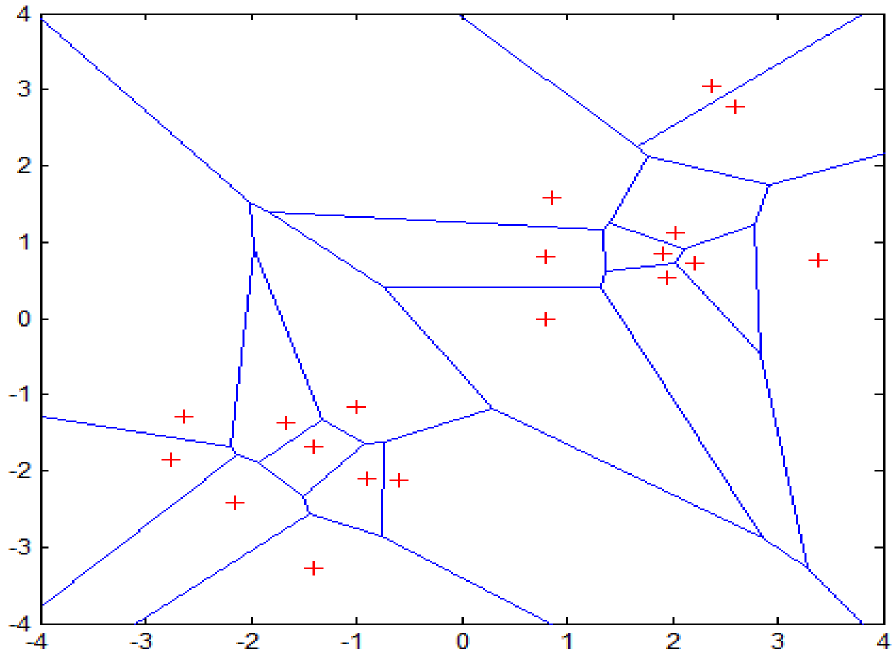
Error rate vs 1/k



Nearest-neighbor classifiers

- Nearest neighbor classifiers are local classifiers

- They can produce decision boundaries of arbitrary shapes.



1-NN decision boundary is a Voronoi Diagram

Distance Metrics

Minkowsky:

$$D(x,y) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r}$$

Euclidean:

$$D(x,y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(x,y) = \sum_{i=1}^{m} |x_i - y_i|$$

Nearest Neighbor Classification…

- **Data preprocessing is often required**
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
    - Example:
      - height of a person may vary from 1.5m to 1.8m
      - weight of a person may vary from 90lb to 300lb
      - income of a person may vary from $10K to $1M

  - Time series are often standardized to have 0 means a standard deviation of 1

Standardization

- Transform raw feature values into z-scores $z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$
  - x_{ij} is the value for the *i*th sample and *j*th feature
  - $\mu_j$is the average of all x_{ij} for feature *j*
  - σ_i is the standard deviation of all x_{ij} over all input samples
- Range and scale of z-scores should be similar (providing distributions of raw feature values are alike)
- Z-scores allow you to take data points drawn from populations with different means and standard deviations and place them on a common scale.

Choice of proximity measure matters

Mathematical technique to calculation similarity and dissimilarity of data points

- For documents, cosine is better than correlation or Euclidean

Cosine similarity is not a popular proximity measure but it is more important for comparing documents.

1 1 1 1 1 1 1 1 1 1 1 0	vs	0 0 0 0 0 0 0 0 0 0 0 1
0 1 1 1 1 1 1 1 1 1 1 1		1 0 0 0 0 0 0 0 0 0 0 0

Euclidean distance = 1.4142 for both pairs, but the cosine similarity measure has different values for these pairs.

Cosine Similarity

•If **A** and **B** are two vectors, then

cos(**A**, **B**) = <**A**,**B**> / ||**A**|| ||**B**|| ,

where <**A**,**B**> indicates inner product or vector dot product of vectors, **A** and **B**, and **||A||** and ||B|| are the L2 norms

- The result of the cosine similarity ranges from -1 to 1.
  - Value 1: indicates that the vectors are identical
  - Value 0: means that the vectors are orthogonal (not similar at all)
  - Value -1: implies opposite vectors
- The cosine similarity is often used in text analysis to determine the similarity between documents represented as vectors in a high-dimensional space, where each dimension corresponds to a specific term or word.

KNN: Advantages vs Disadvantages

Advantages

- No training phase
- It can learn complex models easily
- It is robust to noisy training data

Disadvantages

- Determining the value of parameter K can be difficult as different K values can give different results.
- It is hard to apply on High Dimensional data
- Computation cost is high as each query has to go through all the records, which takes O(N) time, where N is the number of records.

Nearest Neighbour : Computational Complexity

- Expensive

  To determine the k - nearest neighbours it calculates the distances from all other N trianing points

  - To determine the nearest neighbour of a query point *q*, must compute the distance to all *N* training examples
    - + Pre-sort training examples into fast data structures (kd-trees)
    - + Compute only an approximate distance (LSH)
    - + Remove redundant data (condensing)
- Storage Requirements

  It is required to store all the trianing points to predict the class of new data point

  - Must store all training data **P**
    - + Remove redundant data (condensing)
    - - Pre-sorting often increases the storage requirements
- High Dimensional Data

  With the increase in the dimension the computational cost increases dramatically

  - "Curse of Dimensionality"
    - Required amount of training data increases exponentially with dimension
    - Computational cost also increases dramatically
    - Partitioning techniques degrade to linear search in high dimension

Reduction in Computational Complexity

- Reduce size of training set
  - Condensation, editing

Reducing the size of training data set can decrease the computational complexity

- Use geometric data structure for high dimensional search

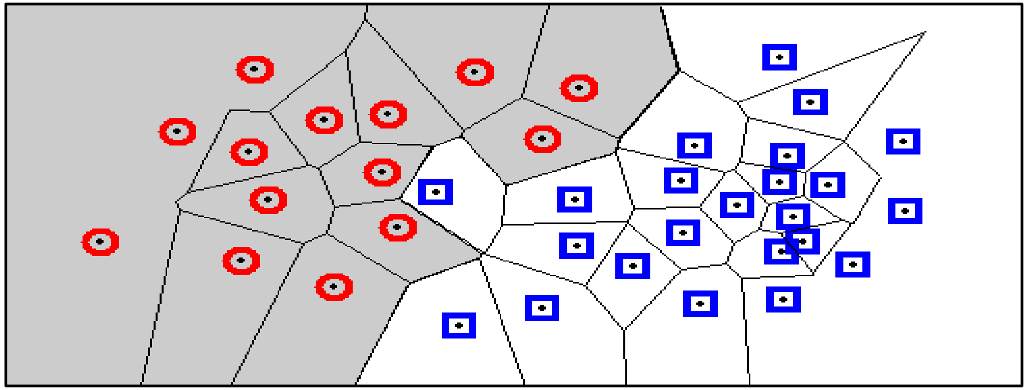
Geometric data structure ??

What is an Imbalanced dataset?

A dataset is said to be imbalanced if the instances of a class are either very less or very high in number as compared to the instances of other class.

CNN basically removes the majority class instances in such a manner that there is no information loss and the dataset becomes balanced.

Condensation: Decision Regions



Each cell contains one sample, and every location within the cell is closer to that sample than to any other sample.

A Voronoi diagram divides the space into such cells.

Every query point will be assigned the classification of the sample within that cell. The *decision boundary* separates the class regions based on the 1-NN decision rule.

Knowledge of this boundary is sufficient to classify new points.

The boundary itself is rarely computed; many algorithms seek to retain only those points necessary to generate an identical boundary.

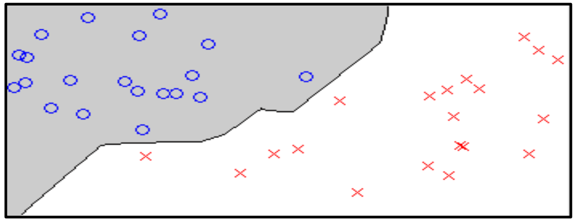
Condensing

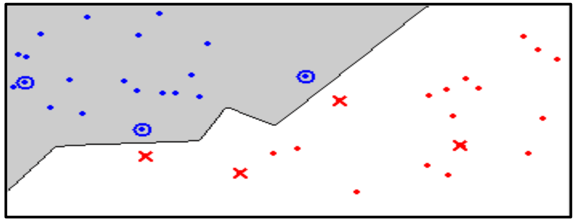
- Aim is to reduce the number of training samples
- Retain only the samples that are needed to define the decision boundary

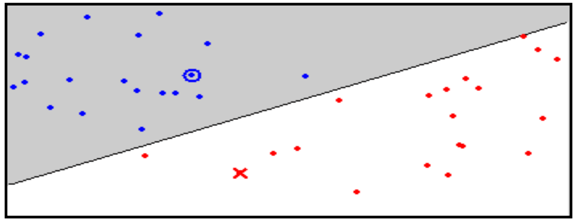
  Condensing targets to retains only those samples that are needed to define the decision boundaries

- Decision Boundary Consistent – a subset whose nearest neighbour decision boundary is identical to the boundary of the entire training set

- Minimum Consistent Set – the smallest subset of the training data that correctly classifies all of the original training data


Original data

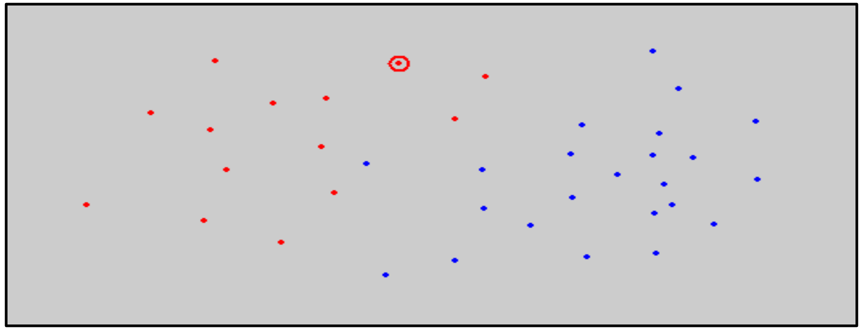

Condensed data


Minimum Consistent Set

Condensing

- Condensed Nearest Neighbour (CNN)

1. Initialize subset with a single (or K) training example
2. Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
3. Return to 2 until no transfers occurred or the subset is full

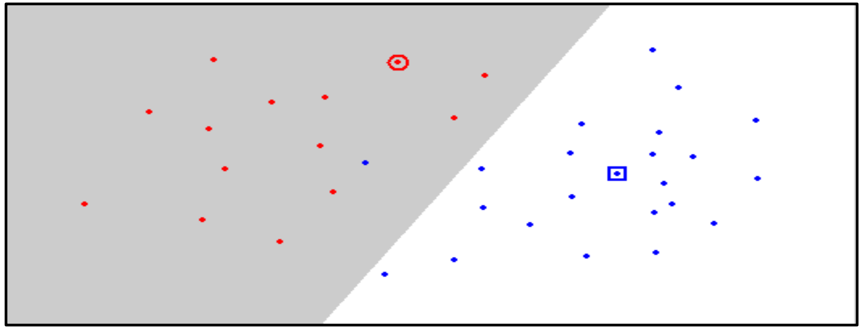


- Incremental
- Order dependent
- Neither minimal nor decision boundary consistent
- $O(n^3)$ for brute-force method

Condensing

- Condensed Nearest Neighbour (CNN)

1. Initialize subset with a single training example
2. Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
3. Return to 2 until no transfers occurred or the subset is full

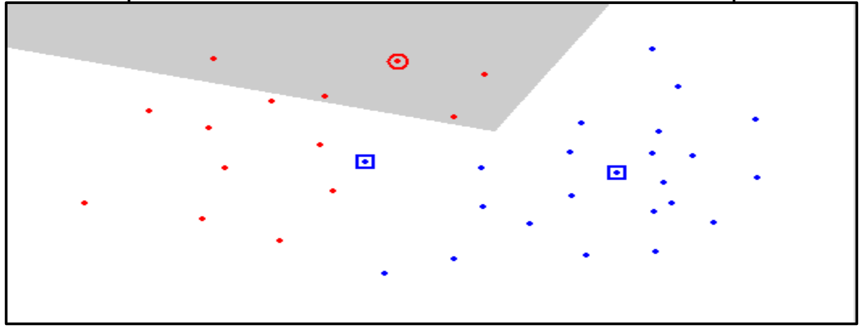


See there is one point in the classification that is falsely classified.

Condensing

- Condensed Nearest Neighbour (CNN)

1. Initialize subset with a single training example
2. Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
3. Return to 2 until no transfers occurred or the subset is full

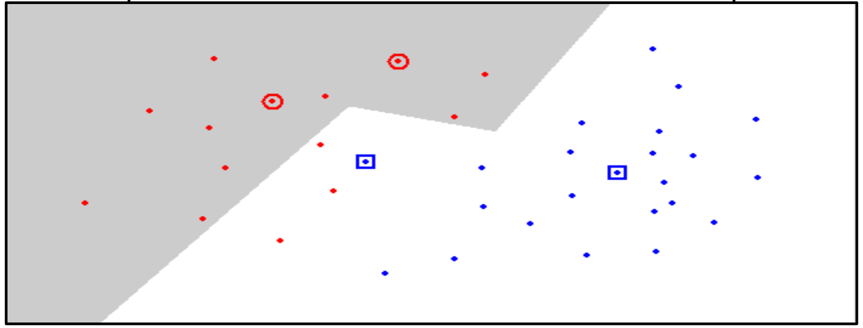


After transferring the incorrectly classified sample point, we got a number of points that became falsely classified.

Condensing

- Condensed Nearest Neighbour (CNN)

1. Initialize subset with a single training example
2. Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
3. Return to 2 until no transfers occurred or the subset is full

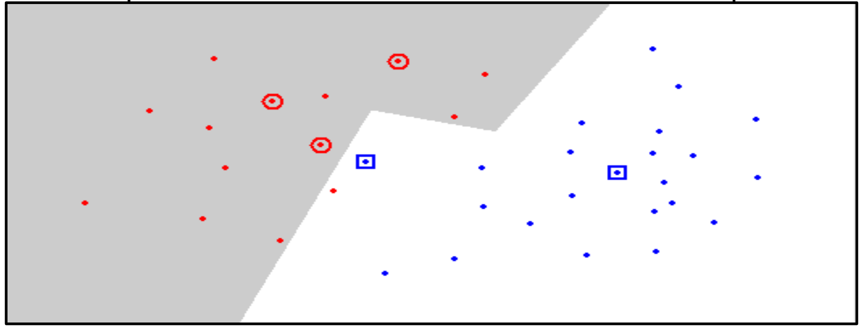


Condensing

- Condensed Nearest Neighbour (CNN)

1. Initialize subset with a single training example
2. Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
3. Return to 2 until no transfers occurred or the subset is full

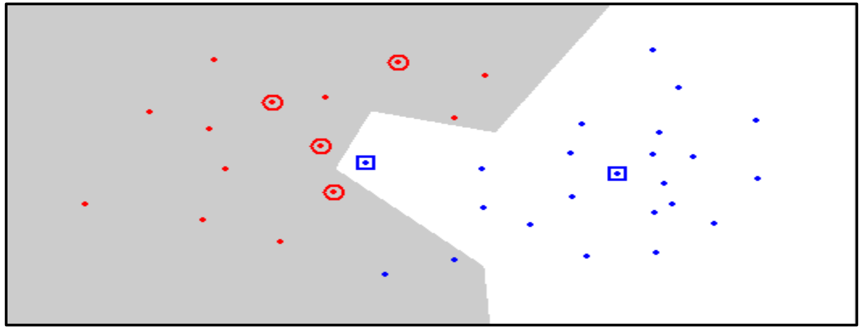
Repeat till the classification becomes correct
ie, all data points are correctly classified



Condensing

- Condensed Nearest Neighbour (CNN)

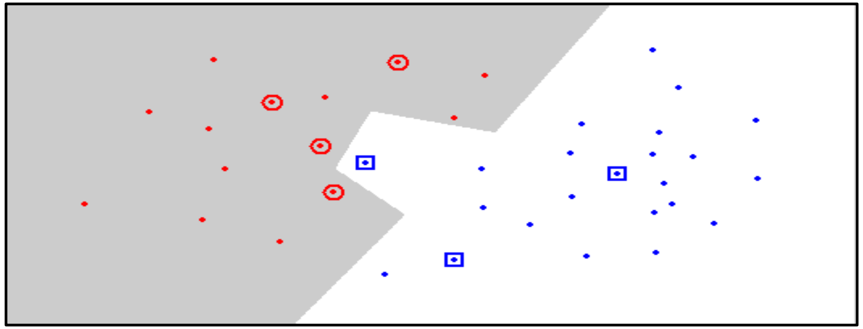
1. Initialize subset with a single training example
2. Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
3. Return to 2 until no transfers occurred or the subset is full



Condensing

- Condensed Nearest Neighbour (CNN)

1. Initialize subset with a single training example
2. Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
3. Return to 2 until no transfers occurred or the subset is full



KD TREE axis selection

There are different strategies for choosing an axis when dividing, but the most common one would be to cycle through each of the K dimensions repeatedly and select a midpoint along it to divide the space. For instance, in the case of 2-dimensional points with x and y axes, we first split along the x-axis, then the y-axis, and then the x-axis again, continuing in this manner until all points are accounted for

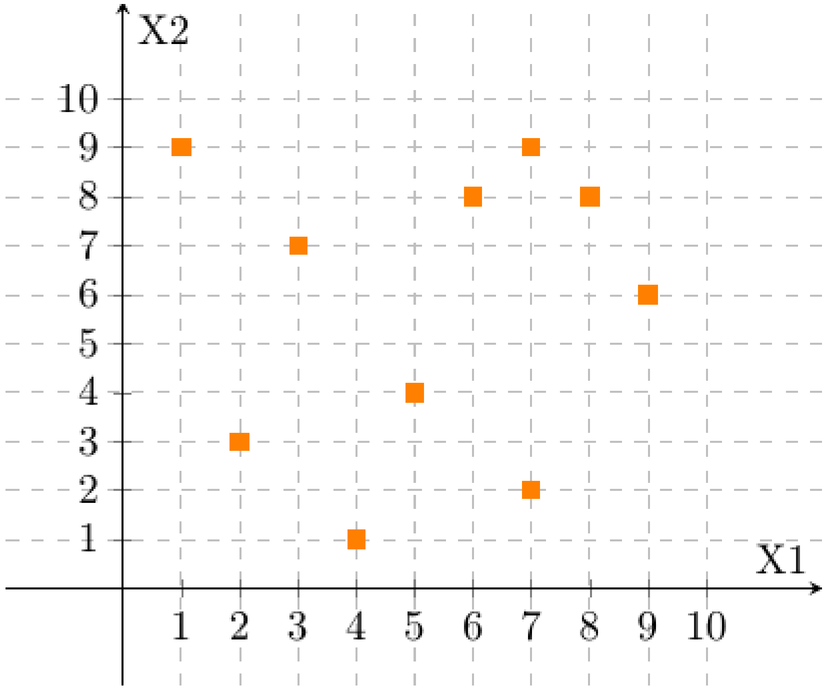
KD Tree

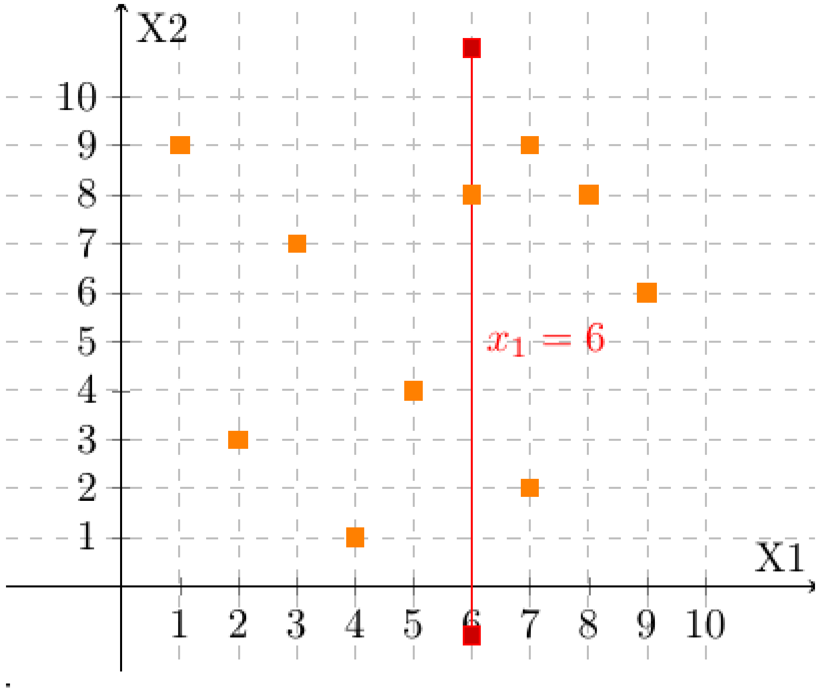
A data structure that organizes points in a k-dimensional space. It is also known as a K-Dimensional Tree.

- KD Tree is a space partitioning data structure for organizing points in K-Dimensional space.
- It is useful for representing data efficiently.
- In KD Tree the data points are organized and partitioned on the basis of some specific conditions.
- Some axis-aligned **cuts** are used to create different regions, keeping track of points that lie in these regions
  - Split the regions at the **median** of the observations.
- Each region is represented by a node in the tree

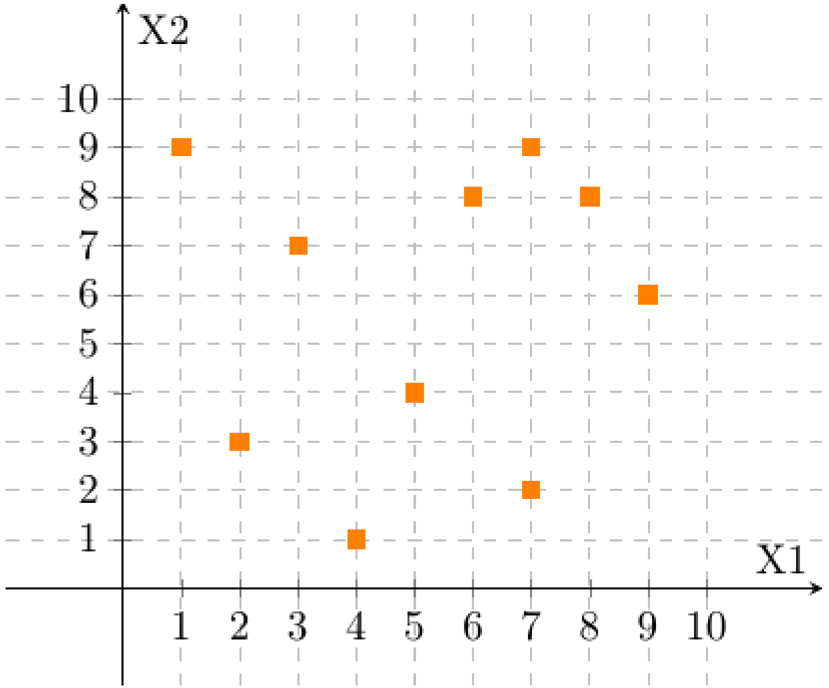
Every non-leaf node in the tree acts as a hyperplane, dividing the space into two partitions. This hyperplane is perpendicular to the chosen axis, which is associated with one of the K dimensions.

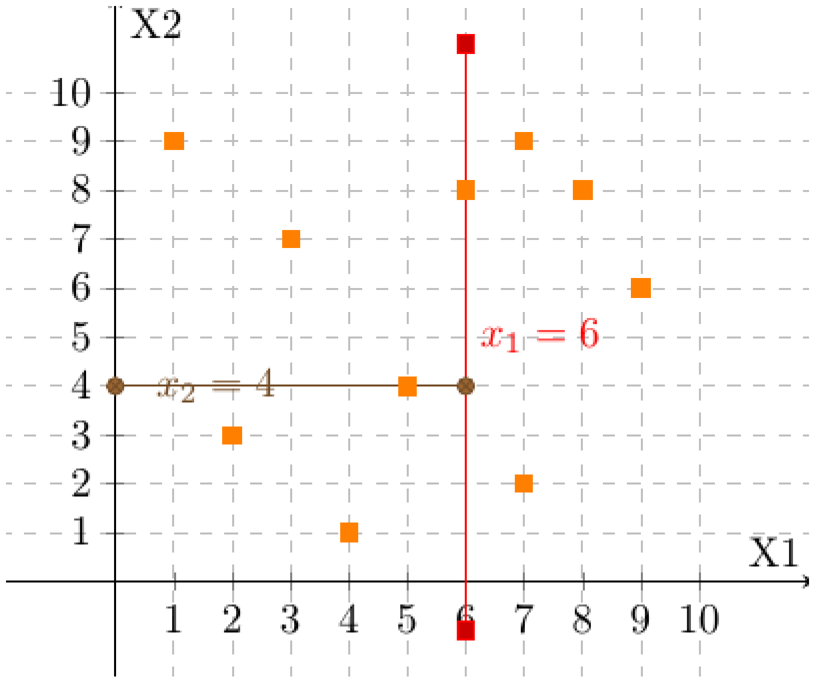
Example



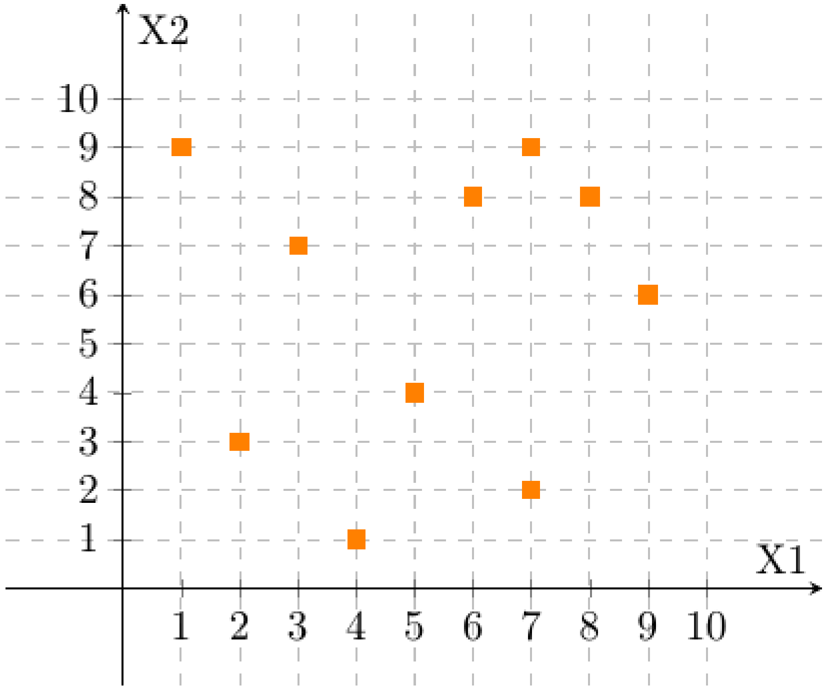


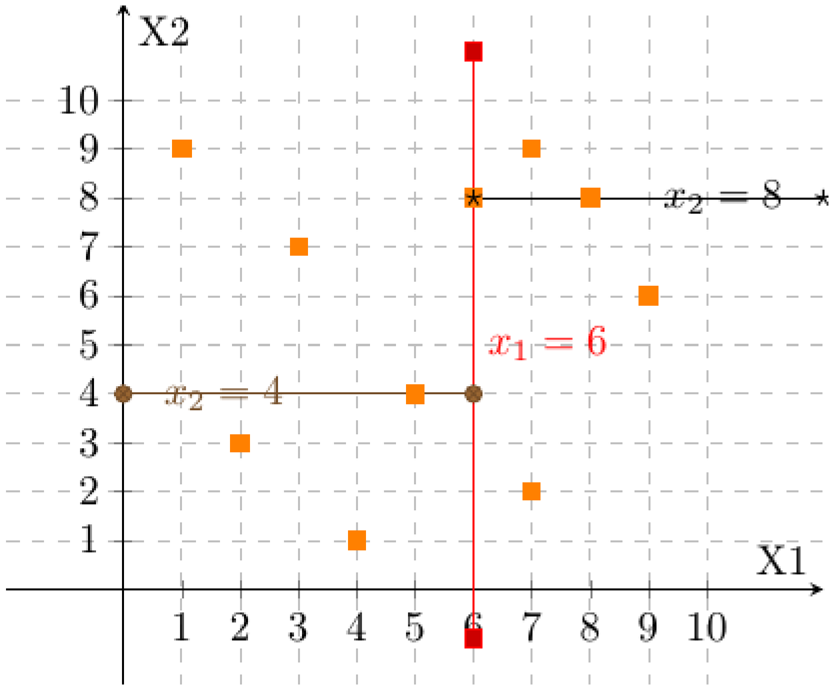
Example



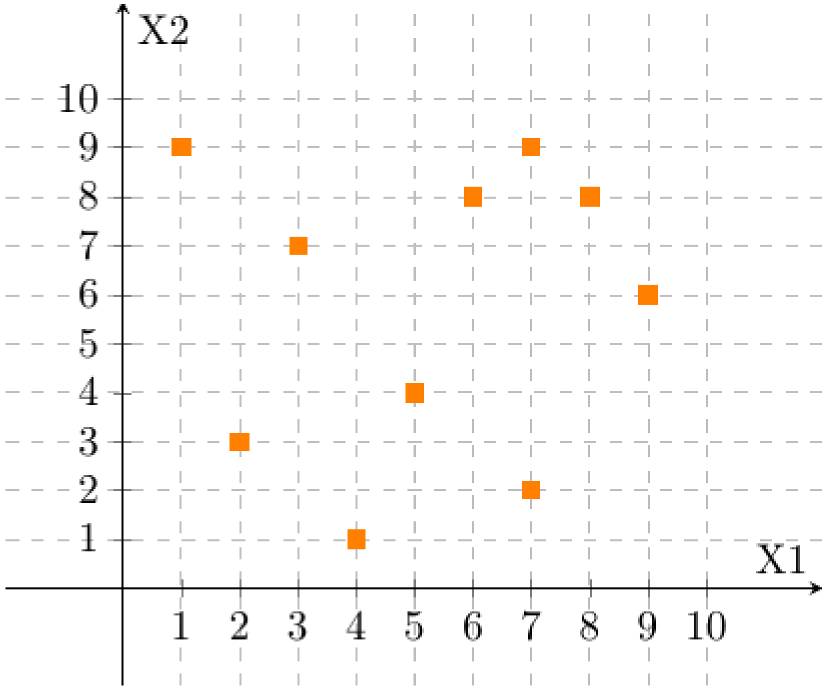


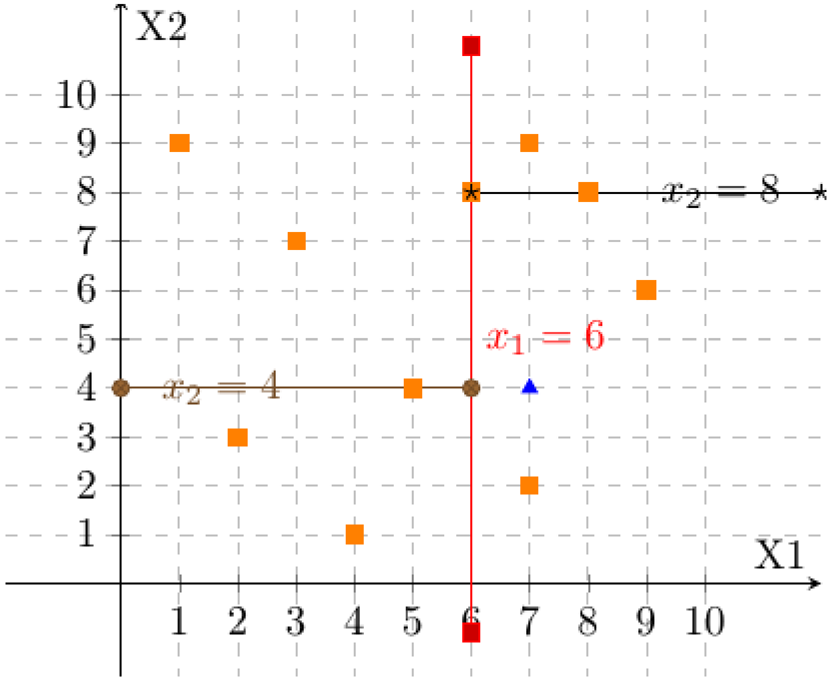
Example





Example





Advantages of KD Tree

- At each level of the tree, KDTree divides the range of the domain in half. Hence they are useful for performing range searches.
- The complexity lies in between O(log N) to O(N) where N is the number of nodes in the tree.

Disadvantages of KD Tree

- Degradation of performance when high dimensional data is used. The algorithm will need to visit many more branches. If the dimensionality of a dataset is K then the number of nodes $N \gg 2^k$.
- If the query point is far from all the points in the dataset then we might have to traverse the whole tree to find the nearest neighbors.

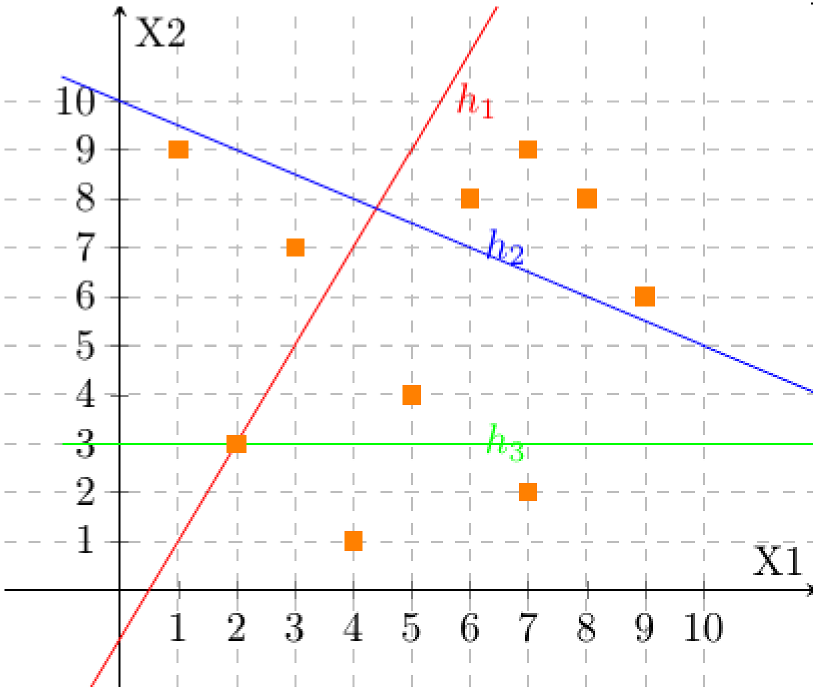
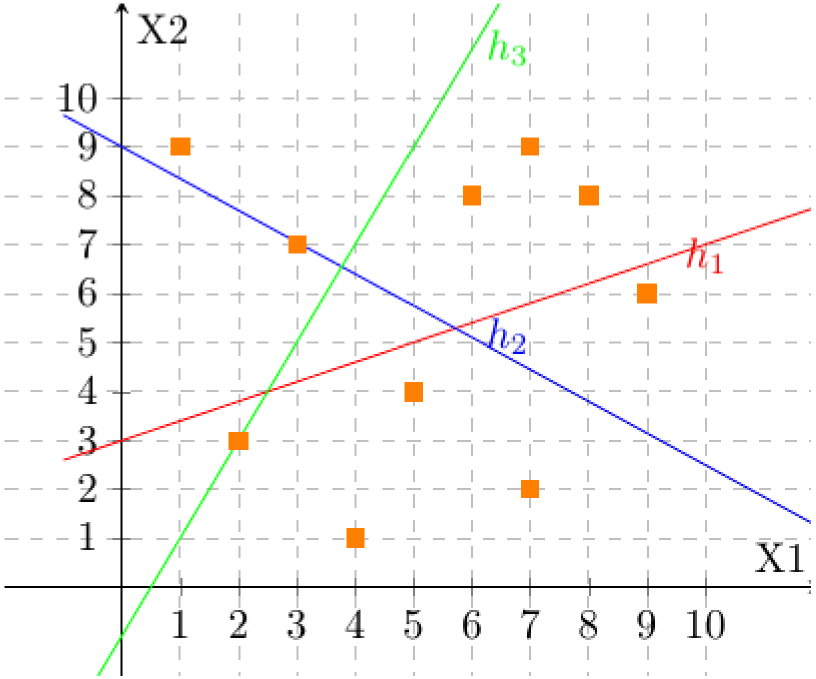
LSH: Locality Sensitive Hashing

- Locality Sensitive Hashing is a technique to enable creating a hash or putting items in buckets such that
  - Similar items go to the same bucket with high probability
  - Dissimilar items go into the same bucket with low probability

Underlying Idea

- Generate a series of *hyperplanes* that partition the space.
- Items lying in the same partition have the same hash value.
- Consider a dataset with N points, each with d dimensions.
- Generate a random set of K hyperplanes: $h_1, h_2, \ldots, h_K$
- For every point in the dataset ($1 \leq i \leq N$), generate a hash as follows:
- For $k \in 1 \ldots K$
  - kth bit of hash = 1 if $x_i . h_k < 0$
  - kth bit of hash = 0 if $x_i . h_k \geq 0$

- To maximize the probability of finding similar items in the same bucket/ same hash, repeat the process L times generating L different sets of K hyperplanes and corresponding hashes for each item
  - Effectively L independent hash tables are constructed for the dataset
- Given a point , in order to retrieve a similar item:
- For each repetition (each hash table constructed) $1 \leq l \leq L$
  - generate the corresponding hash
  - Compare with all other elements in the same bucket of the hash table – add items which are close in value to the output set.


X2
X1
h_1
h_2
h_3

X2
X1
h_1
h_2
h_3