

Machine Learning

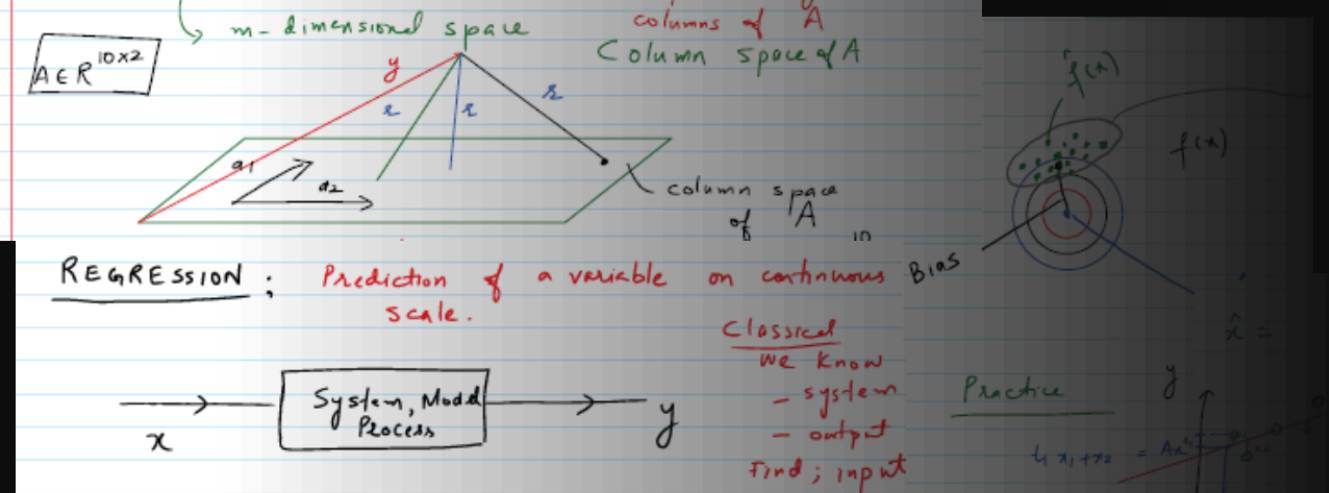
EE514 – CS535

kNN Algorithm: Overview, Analysis, Convergence and Extensions

Zubair Khalid

School of Science and Engineering
Lahore University of Management Sciences

https://www.zubairkhalid.org/ee514_2021.html



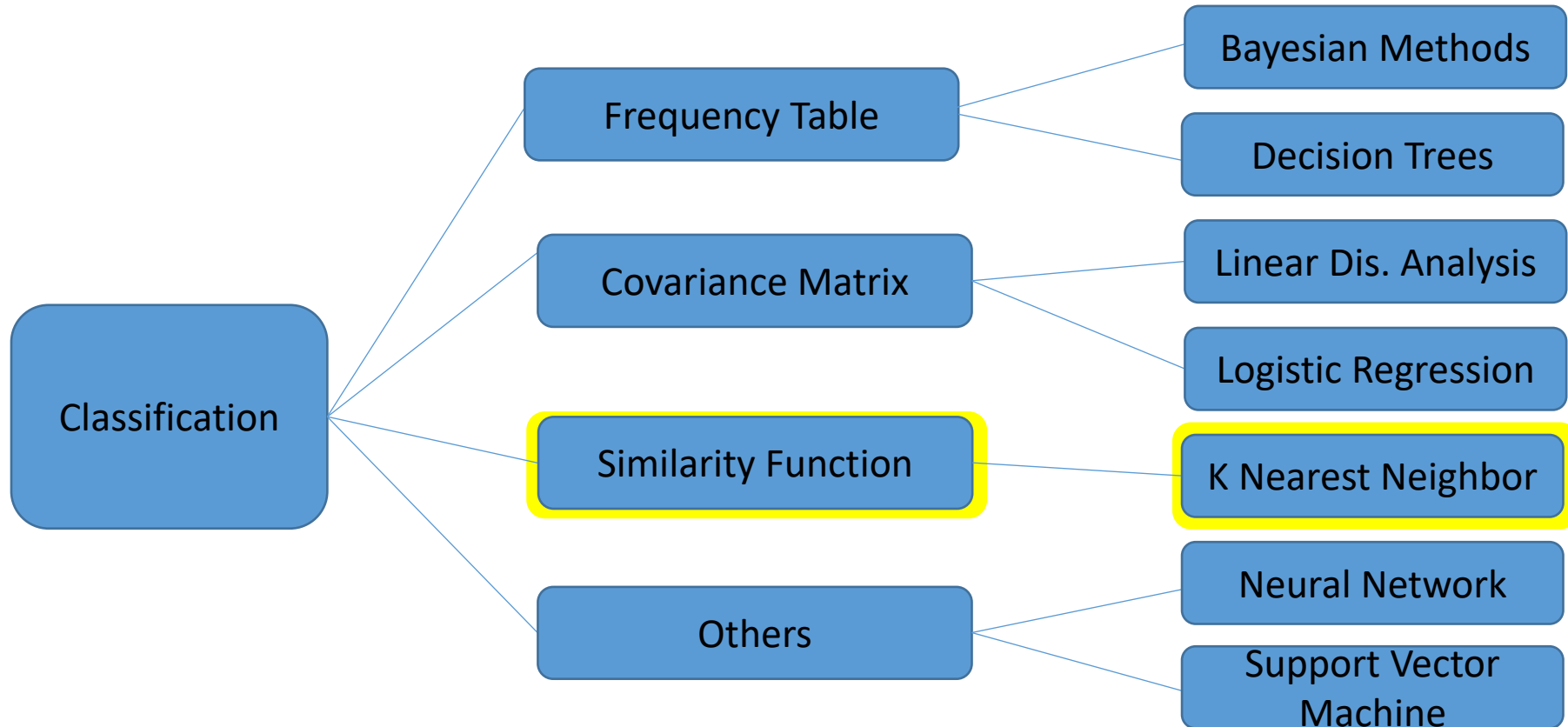
Outline

- *k-Nearest Neighbor (kNN) Algorithm Overview*
- *Algorithm Formulation*
- *Distance Metrics*
- *Choice of k*
- *Algorithm Convergence*
- *Storage, Time Complexity Analysis*
- *Fast kNN*
- *The Curse of Dimensionality*

Supervised Learning

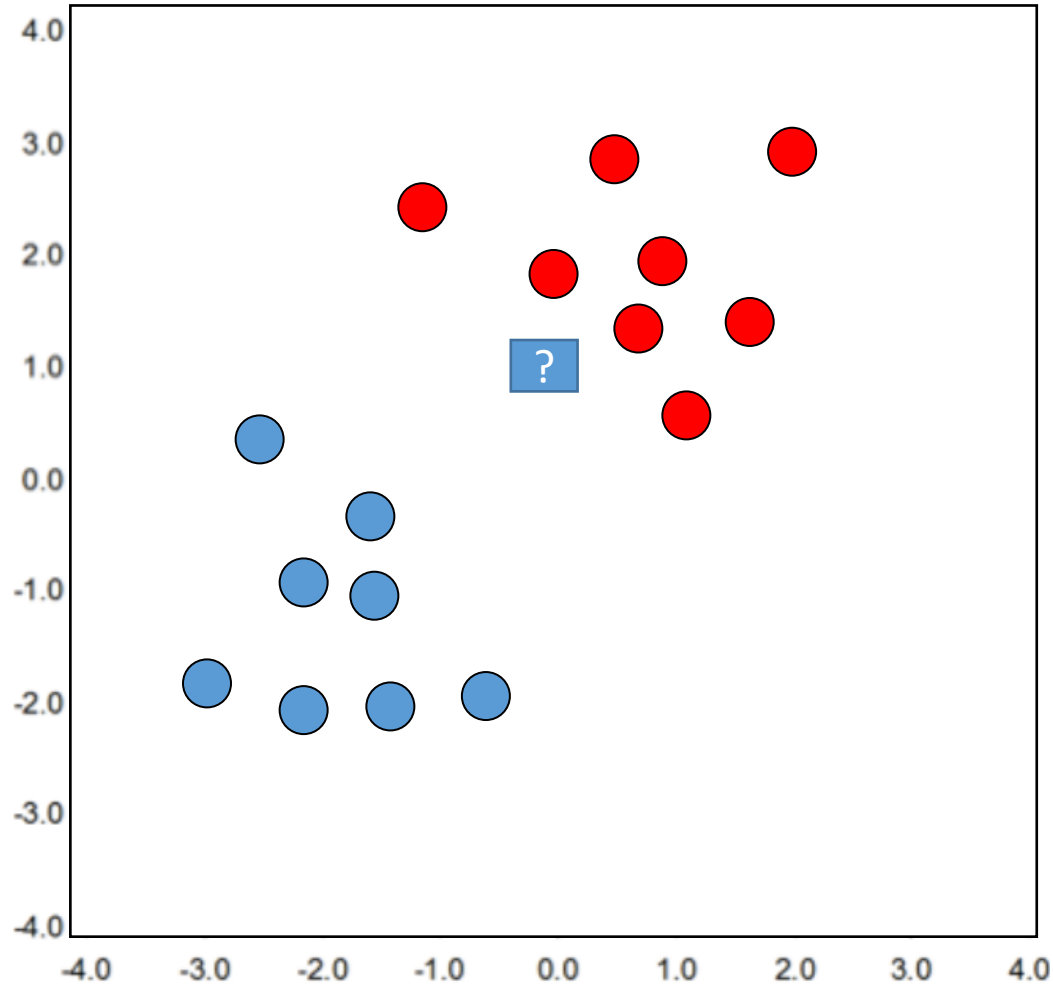
Classification Algorithms or Methods

Predicting a categorical output is called classification



k-Nearest Neighbor (kNN) Algorithm

Idea:



- Two classes, two features
- We want to assign label to unknown data point?
- Label should be *red*.

k-Nearest Neighbor (kNN) Algorithm

Idea:

- We have similar labels for similar features.
- We classify new test point using similar training data points.

Algorithm overview:

- Given some new test point x for which we need to predict the class y .
- Find **most similar** data-points in the training data.
- Classify x “like” these **most** similar data points.

Questions:

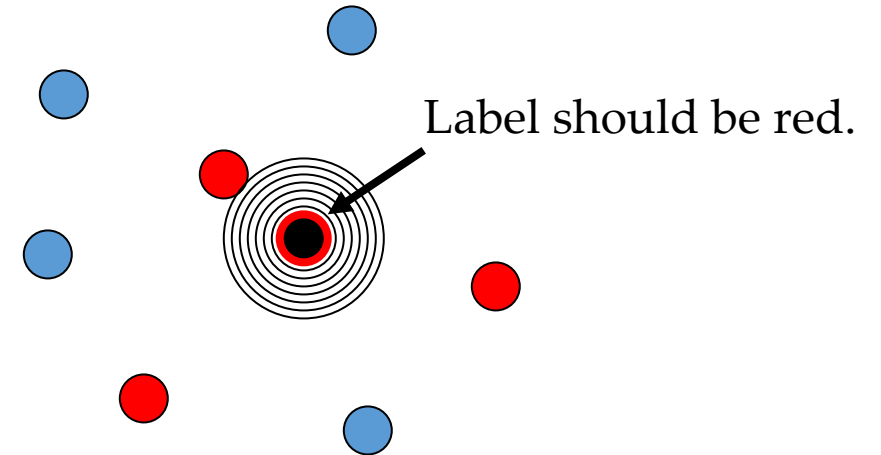
- How do we determine the similarity?
- How many similar training data points to consider?
- How to resolve inconsistencies among the training data points?

k-Nearest Neighbor (kNN) Algorithm

1-Nearest Neighbor:

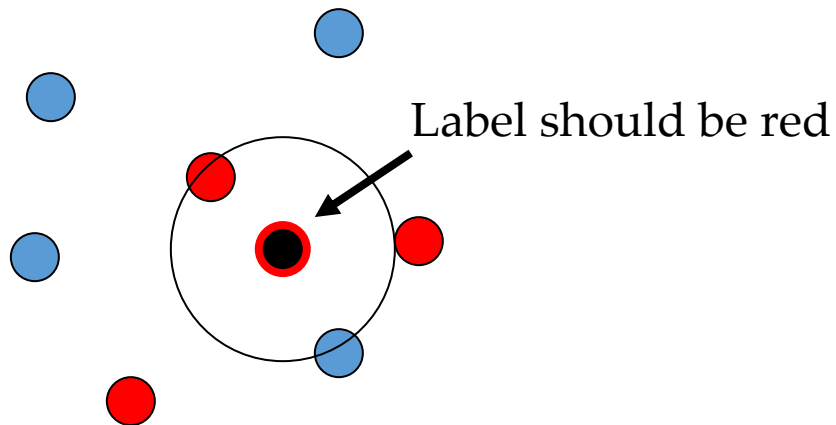
Simplest ML Classifier

Idea: Use the label of the *closest* known point

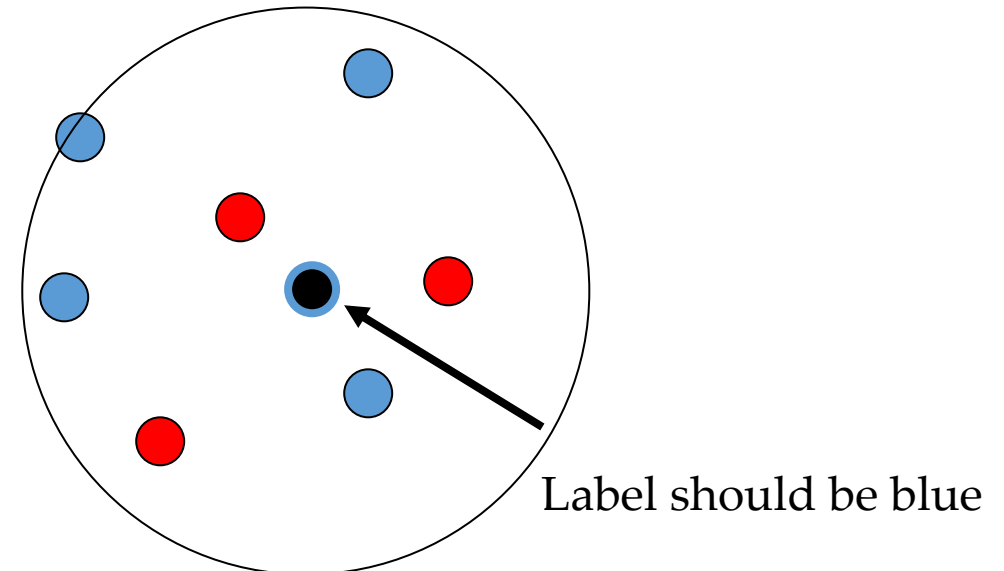


Generalization:

Determine the label of k nearest neighbors and assign the most frequent label



k=3



k=7

k-Nearest Neighbor (kNN) Algorithm

Formal Definition:

- We assume we have training data D given by

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

- $\mathcal{Y} = \{1, 2, \dots, M\}$ (M-class classification)
- For a point $\mathbf{x} \in \mathcal{X}^d$, we define a set $S_{\mathbf{x}} \subseteq D$ as a set of k neighbors.
- Using the function ‘dist’ that computes the distance between two points in \mathcal{X}^d , we can define a set $S_{\mathbf{x}}$ of size k as

$$\text{dist}(\mathbf{x}, \mathbf{x}') \geq \max_{(\mathbf{x}'', y'') \in S_{\mathbf{x}}} \text{dist}(\mathbf{x}, \mathbf{x}''), \quad \forall (\mathbf{x}', y') \in D \setminus S_{\mathbf{x}}$$

Interpretation:

Every point in D but not in $S_{\mathbf{x}}$ is at least as far away from \mathbf{x} as the furthest point in $S_{\mathbf{x}}$.

k-Nearest Neighbor (kNN) Algorithm

Formal Definition:

- Using the $S_{\mathbf{x}}$, we can define a classifier as a function that gives us most frequent label of the data points in $S_{\mathbf{x}}$

$$h(\mathbf{x}) = \text{mode}(\{y'' : \underbrace{(x'', y'') \in S_{\mathbf{x}}}\}) = y$$

- *Instance-based learning algorithm; easily adapt to unseen data*

k-Nearest Neighbor (kNN) Algorithm

Decision Boundary:

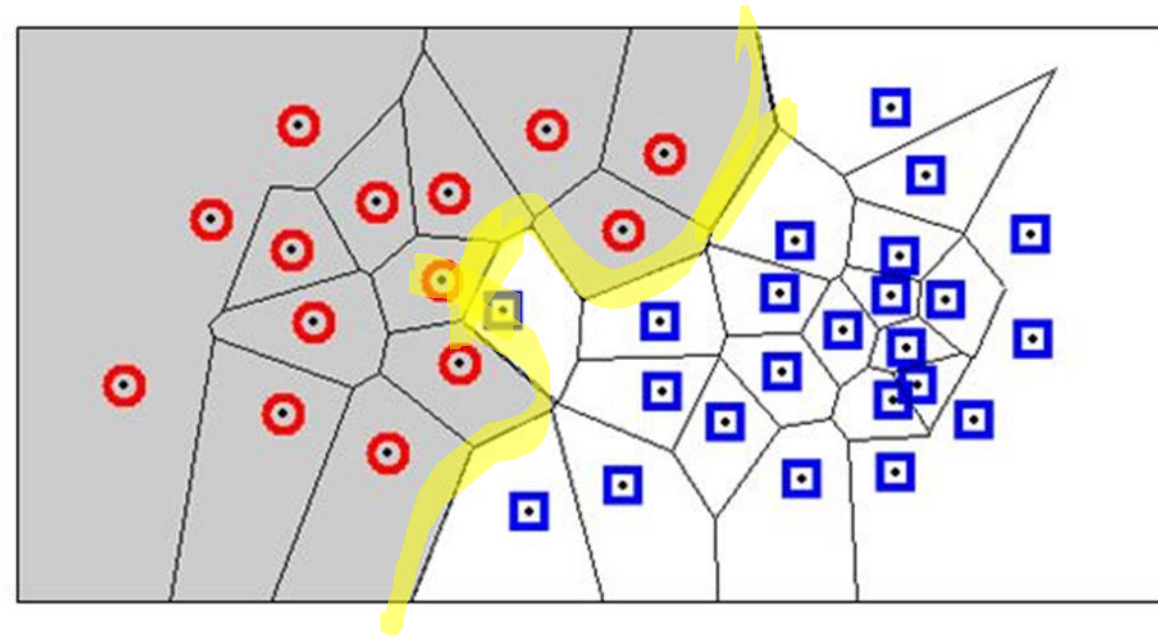
For $k = 1$, kNN defines a region, called decision boundary or region, in the space. Such division of the feature space is referred to as Voronoi partitioning.

We can define a region R_i associated with the feature point \mathbf{x}_i as

$$R_i = \{\mathbf{x} : \text{dist}(\mathbf{x}, \mathbf{x}_i) < \text{dist}(\mathbf{x}, \mathbf{x}_j), i \neq j\}$$

For example, Voronoi partitioning using Euclidean distance in two-dimensional space.

Classification boundary changes with the change in k and the distance metric.



k-Nearest Neighbor (kNN) Algorithm

Decision Boundary:

Demonstration

<https://demonstrations.wolfram.com/KNearestNeighborKNNClassifier/>

k-Nearest Neighbor (kNN) Algorithm

Characteristics of kNN:

- No assumptions about the distribution of the data
- Non-parametric algorithm
 - No parameters
- Hyper-Parameters
 - k (number of neighbors)
 - Distance metric (to quantify similarity)

k-Nearest Neighbor (kNN) Algorithm

Characteristics of kNN:

- Complexity (both time and storage) of prediction increases with the size of training data.
- Can also be used for regression (average or inverse distance weighted average)

- For example,

$$y = \frac{1}{k} \sum_{i=1}^k y_i, \quad (\mathbf{x}_i, y_i) \in S_{\mathbf{x}}$$

k-Nearest Neighbor (kNN) Algorithm

Practical issues:

- For binary classification problem, use odd value of k . Why?
- In case of a tie:
 - Use prior information
 - Use 1-nn classifier or $k-1$ classifier to decide
- Missing values in the data
 - Average value of the feature.

Outline

- *k*-Nearest Neighbor (kNN) Algorithm Overview
- Algorithm Formulation
- **Distance Metrics**
- Choice of *k*
- Algorithm Convergence
- Storage, Time Complexity Analysis
- Fast kNN
- The Curse of Dimensionality

k-Nearest Neighbor (kNN) Algorithm

We need to define distance metric to find the set of k nearest neighbors, S_x

- We defined a set S_x of size k as

$$\text{dist}(\mathbf{x}, \mathbf{x}') \geq \max_{(\mathbf{x}'', y'') \in S_x} \text{dist}(\mathbf{x}, \mathbf{x}''), \quad \forall (\mathbf{x}', y') \in D \setminus S_x$$

k-Nearest Neighbor (kNN) Algorithm

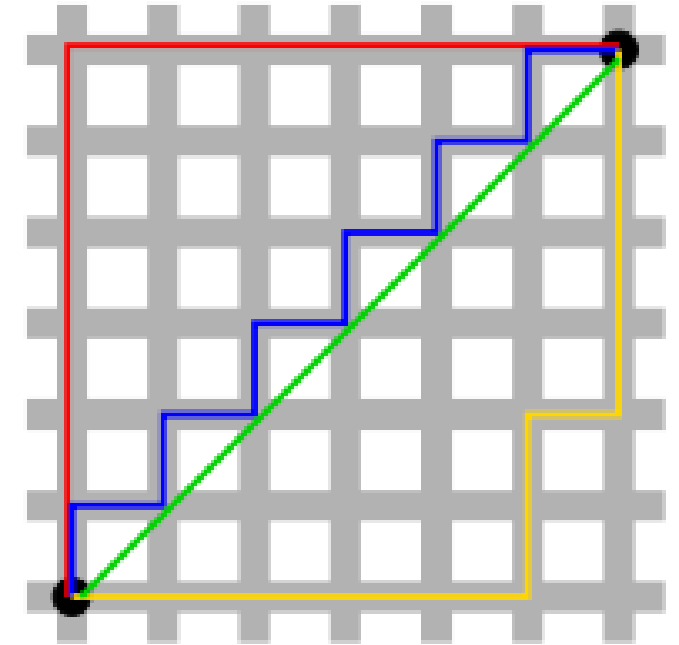
Distance Metric:

- Euclidean

$$\text{dist}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2 = \sqrt{\sum_{i=1}^d (x_i - x'_i)^2}$$

- Manhattan

$$\text{dist}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_1 = \sum_{i=1}^d |x_i - x'_i|$$



Euclidean

Manhattan

Manhattan

Manhattan

k-Nearest Neighbor (kNN) Algorithm

Norm of a vector

- p -norm of a vector $\mathbf{x} \in \mathbf{R}^d$

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^d |x_i|^p \right)^{1/p}, \quad p \geq 1$$

Properties of Norm

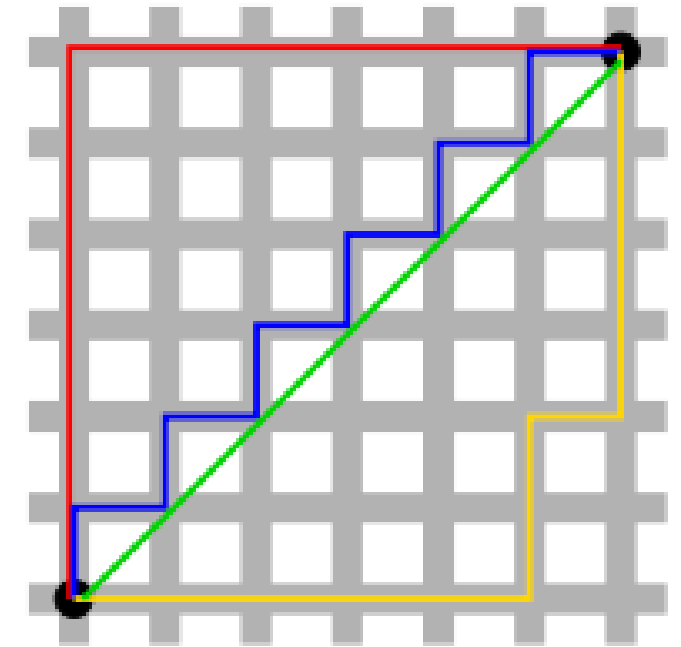
Non-negative, $\|\mathbf{x}\|_p \geq 0$

Absolutely homogenous: $\|\alpha\mathbf{x}\|_p = |\alpha| \|\mathbf{x}\|_p$

$\|\alpha\mathbf{x}\|_p = 0 \iff \mathbf{x} = 0$

Triangular inequality, $\|\mathbf{x} + \mathbf{x}'\|_p \leq \|\mathbf{x}\|_p + \|\mathbf{x}'\|_p$

$$\|\mathbf{x}\|_q \leq \|\mathbf{x}\|_p, \quad p \leq q$$



Euclidean $6\sqrt{2}$

Manhattan
Manhattan 12

Manhattan

k-Nearest Neighbor (kNN) Algorithm

Distance Metric:

- Euclidean $\text{dist}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2 = \sqrt{\sum_{i=1}^d (x_i - x'_i)^2}$

- Manhattan $\text{dist}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_1 = \sum_{i=1}^d |x_i - x'_i|$

- Minkowski

$$\text{dist}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_p = \left(\sum_{i=1}^d (|x_i - x'_i|)^p \right)^{1/p}, \quad p \geq 1$$

$$p = \infty$$

$$\text{dist}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_\infty = \max_{i=1,2,\dots,d} (|x_i - x'_i|) \quad \text{Chebyshev Distance}$$

k-Nearest Neighbor (kNN) Algorithm

Distance Metric:

Properties of Distance Metrics:

Non-negative, $\text{dist}(\mathbf{x}, \mathbf{x}') \geq 0$

Symmetric, $\text{dist}(\mathbf{x}, \mathbf{x}') = \text{dist}(\mathbf{x}', \mathbf{x})$

$\text{dist}(\mathbf{x}, \mathbf{x}') = 0 \iff \mathbf{x} = \mathbf{x}'$

Triangular inequality, $\text{dist}(\mathbf{x}, \mathbf{x}') \leq \text{dist}(\mathbf{x}', \mathbf{x}'') + \text{dist}(\mathbf{x}'', \mathbf{x})$

k-Nearest Neighbor (kNN) Algorithm

Distance Metric:

- For categorical variable, use Hamming Distance

$$\text{dist}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^d 1 - \delta_{x_i - x'_i}$$

k-Nearest Neighbor (kNN) Algorithm

Cosine Distance

- Cosine distance, though does not satisfy the properties we defined for distance metric, is however used to measure the angular distance between the vectors.
- It follows from the standard definition of inner (dot) product between the vectors, that is,

$$\mathbf{x}^T \mathbf{x}' = \|\mathbf{x}\|_2 \|\mathbf{x}'\|_2 \cos \theta$$

or

$$\cos \theta = \frac{\mathbf{x}^T \mathbf{x}'}{\|\mathbf{x}\|_2 \|\mathbf{x}'\|_2}$$

What is the range of values of angular distance and what is the interpretation of these values?

k-Nearest Neighbor (kNN) Algorithm

Practical issues in computing distance:

- Mismatch in the values of data
 - Issue: Distance metric is mapping from d -dimensional space to a scalar. The values should be of the same order along each dimension.
- Solution: Data Normalization

Outline

- *k*-Nearest Neighbor (kNN) Algorithm Overview
- Algorithm Formulation
- Distance Metrics
- **Choice of *k***
- Algorithm Convergence
- Storage, Time Complexity Analysis
- Fast kNN
- The Curse of Dimensionality

k-Nearest Neighbor (kNN) Algorithm

Choice of k:

- $k=1$

Sensitive to noise

High variance

Increasing k makes algorithm less sensitive to noise

- $k=n$

Decreasing k enables capturing finer structure of space

Idea: Pick k not too large, but not too small (depends on data)

How?

k-Nearest Neighbor (kNN) Algorithm

Choice of k:

- Learn the best hyper-parameter, k using the data.
- Split data into training and validation.
- Start from $k=1$ and keep iterating by carrying out (5 or 10, for example) cross-validation and computing the loss on the validation data using the training data.
- Choose the value for k that minimizes validation loss.
- This is the only learning required for kNN.

Outline

- *k*-Nearest Neighbor (kNN) Algorithm Overview
- Algorithm Formulation
- Distance Metrics
- Choice of *k*
- **Algorithm Convergence**
- Storage, Time Complexity Analysis
- Fast kNN
- The Curse of Dimensionality

k-Nearest Neighbor (kNN) Algorithm

Error Convergence:

We wish to analyze the error rate of the kNN classifier.

We will show that the error of 1-NN classifier converges as number of points in D increases.

To show the convergence, we will derive that 1-NN classifier is only a factor 2 worse than the best possible classifier.

k-Nearest Neighbor (kNN) Algorithm

Learning Problem

We represent the entire training data as

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

Recall a problem in hand. We want to develop a model that can predict the label for the input for which label is unknown using kNN.

We assume that the data points (\mathbf{x}_i, y_i) are drawn from some (unknown) distribution $P(X, Y)$.

k-Nearest Neighbor (kNN) Algorithm

Bayes Optimal Classifier

If we assume that we know $P(y|\mathbf{x})$, we can predict the most likely label as follows:

$$y^* = h(\mathbf{x}) = \max_y P(y|\mathbf{x})$$

Error Rate:

Probability of misclassification or error rate can be computed as

$$\epsilon_{\text{Bayes Classifier}} = 1 - P(h(\mathbf{x})|\mathbf{x}) = 1 - P(y^*|\mathbf{x})$$

k-Nearest Neighbor (kNN) Algorithm

Error Convergence:

We want to determine 1-NN classification error as $n \rightarrow \infty$.

For a test-point \mathbf{x} , we assume 1-NN classifier assigns label of \mathbf{x}_{NN} to \mathbf{x} .

We can easily show that (consequence of filling of space)

$$\lim_{n \rightarrow \infty} \text{dist}(\mathbf{x}, \mathbf{x}_{NN}) \rightarrow 0$$

Error Rate:

We want to determine probability of misclassification, that is, the probability of having different labels of \mathbf{x} and \mathbf{x}_{NN} .

k-Nearest Neighbor (kNN) Algorithm

Error Convergence:

Probability that y is the correct label of \mathbf{x} but \mathbf{x}_{NN} has a different label:

$$P(y|\mathbf{x})(1 - P(y|\mathbf{x}_{NN}))$$

Probability that y is the incorrect label of \mathbf{x} but \mathbf{x}_{NN} has y label:

$$P(y|\mathbf{x}_{NN})(1 - P(y|\mathbf{x}))$$

Error Rate:

Probability of misclassification or error rate can be computed using the law of total probability

$$\epsilon_{NN} = P(y|\mathbf{x}_{NN})(1 - P(y|\mathbf{x})) + P(y|\mathbf{x})(1 - P(y|\mathbf{x}_{NN}))$$

k-Nearest Neighbor (kNN) Algorithm

Error Convergence:

Bound on Error Rate:

$$\epsilon_{\text{NN}} = P(y|\mathbf{x}_{\text{NN}})(1 - P(y|\mathbf{x})) + P(y|\mathbf{x})(1 - P(y|\mathbf{x}_{\text{NN}}))$$

Using

$$\lim_{n \rightarrow \infty} \text{dist}(\mathbf{x}, \mathbf{x}_{\text{NN}}) \rightarrow 0 \Rightarrow \mathbf{x} \rightarrow \mathbf{x}_{\text{NN}}$$

We obtain

$$\epsilon_{\text{NN}} = 2P(y|\mathbf{x})(1 - P(y|\mathbf{x}))$$

$$\epsilon_{\text{NN}} \leq 2(1 - P(y|\mathbf{x}))$$

Noting $P(y|\mathbf{x}) \leq 1$

$$\epsilon_{\text{NN}} \leq 2\epsilon_{\text{Bayes Classifier}}$$

1-NN classifier is only a factor 2 worse than the best possible classifier.

Outline

- *k*-Nearest Neighbor (kNN) Algorithm Overview
- Algorithm Formulation
- Distance Metrics
- Choice of *k*
- Algorithm Convergence
- **Storage, Time Complexity Analysis**
- Fast kNN
- The Curse of Dimensionality

k-Nearest Neighbor (kNN) Algorithm

Algorithm Computational and Storage Complexity:

Input/Output:

- We have a feature vector, \mathbf{x} for which we want to predict label y .
- We have k and dist function.

Steps:

- We defined a set $S_{\mathbf{x}}$ of size k as

$$\text{dist}(\mathbf{x}, \mathbf{x}') \geq \max_{(\mathbf{x}'', y'') \in S_{\mathbf{x}}} \text{dist}(\mathbf{x}, \mathbf{x}''), \quad \forall (\mathbf{x}', y') \in D \setminus S_{\mathbf{x}}$$

- Classifier that gives us most frequent label of the data points in $S_{\mathbf{x}}$

$$h(\mathbf{x}) = \text{mode}(\{y'' : (x'', y'') \in S_{\mathbf{x}}\})$$

k-Nearest Neighbor (kNN) Algorithm

Algorithm:

Steps:

Computational Complexity

1. Find distance between given test point and feature vector of every point in D .

Noting n number of data points we have and each feature vector \mathbf{x} is d -dimensional. $\mathcal{O}(dn)$

2. Find k points in D closest to the given test point vector to form a set S_x .

Finding k -th smallest distance using median of medians method. $\mathcal{O}(n)$

Finding k data-points in D with distance less than the k -th smallest distance $\mathcal{O}(n)$

3. Find the most frequent label in the set S_x and assign it to the test point. $\mathcal{O}(k)$

Computational Complexity: $\mathcal{O}(dn)$

Space Complexity: $\mathcal{O}(dn)$

Outline

- *k*-Nearest Neighbor (kNN) Algorithm Overview
- Algorithm Formulation
- Distance Metrics
- Choice of *k*
- Algorithm Convergence
- Storage, Time Complexity Analysis
- **Fast kNN**
- The Curse of Dimensionality

k-Nearest Neighbor (kNN) Algorithm

Fast kNN:

- kNN Computational complexity: $O(nd)$
- How to make it faster?
 - Dimensionality Reduction
 - Feature Selection (to be covered later)
 - PCA (to be covered)
 - Use efficient method to find nearest neighbors
 - KD Tree

k-Nearest Neighbor (kNN) Algorithm

K-D Tree:

- *k-Dimensional tree*
 - *Extended version of binary search tree in higher dimension*
- *Pick the splitting dimension*
 - *Randomly*
 - *Large variance dimension*
- *Pick the middle value of the feature along the selected dimension after sorting along that dimension.*
- *Use this value as the root node and construct a binary tree and keep going.*

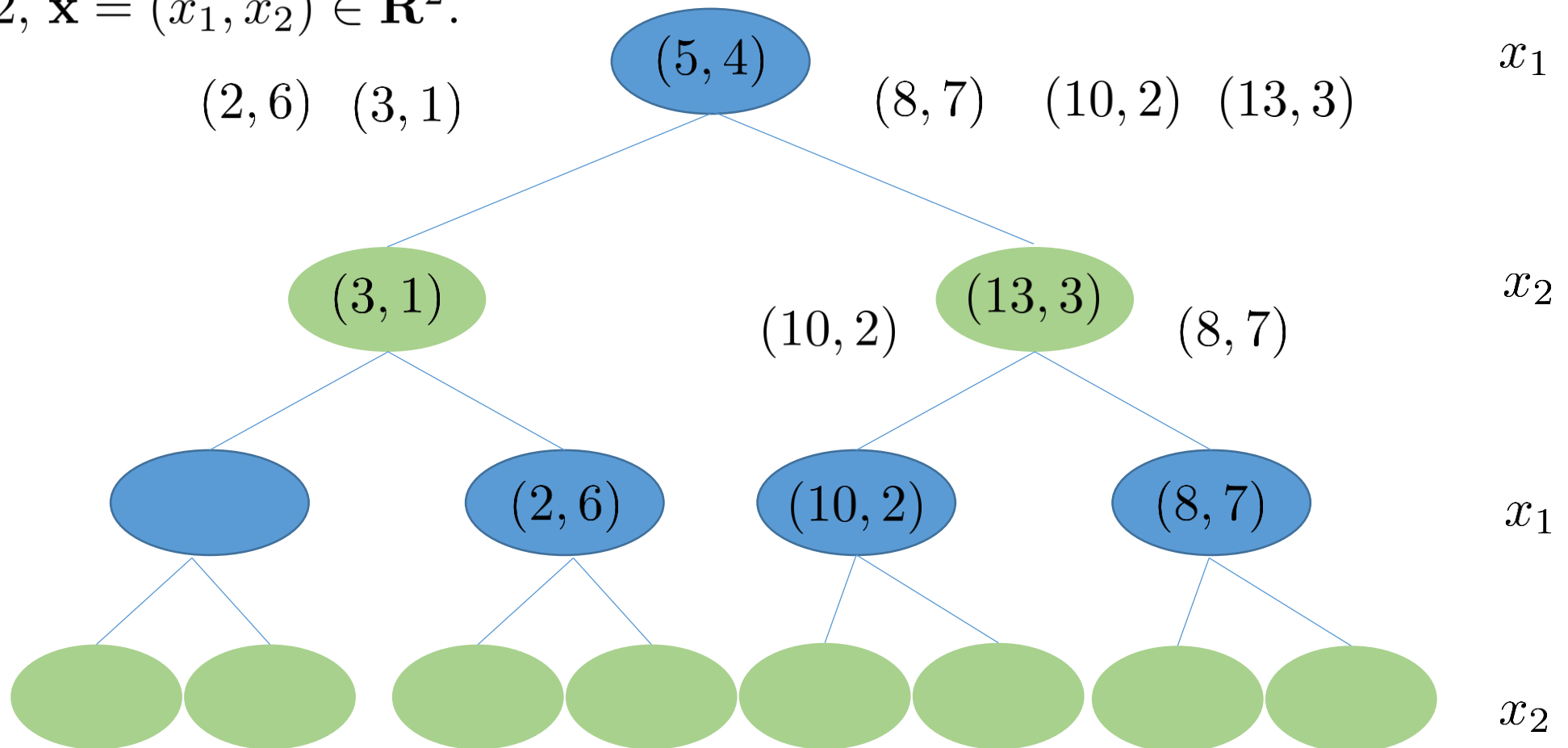
k-Nearest Neighbor (kNN) Algorithm

K-D Tree:

Example:

Splitting dimension

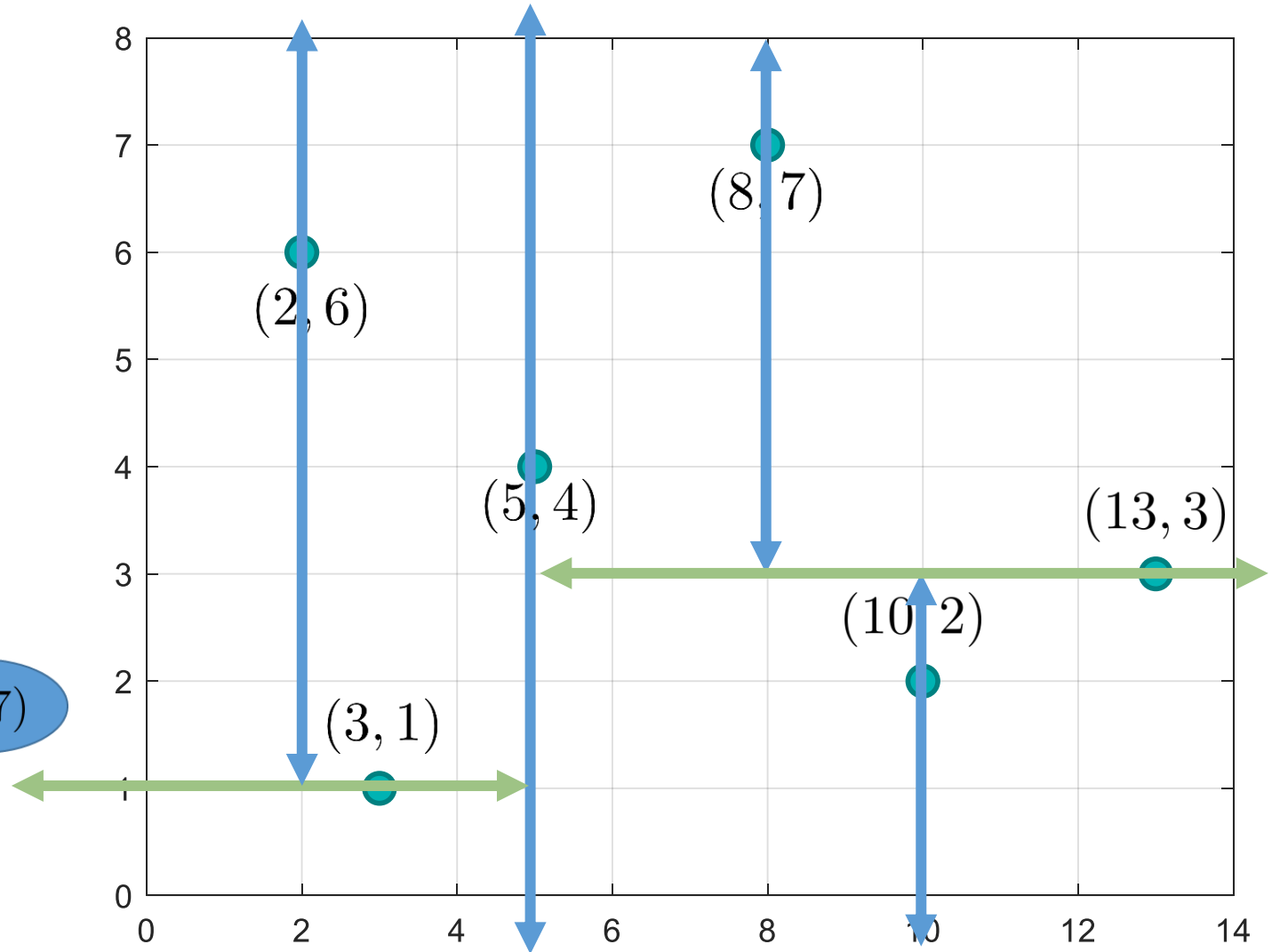
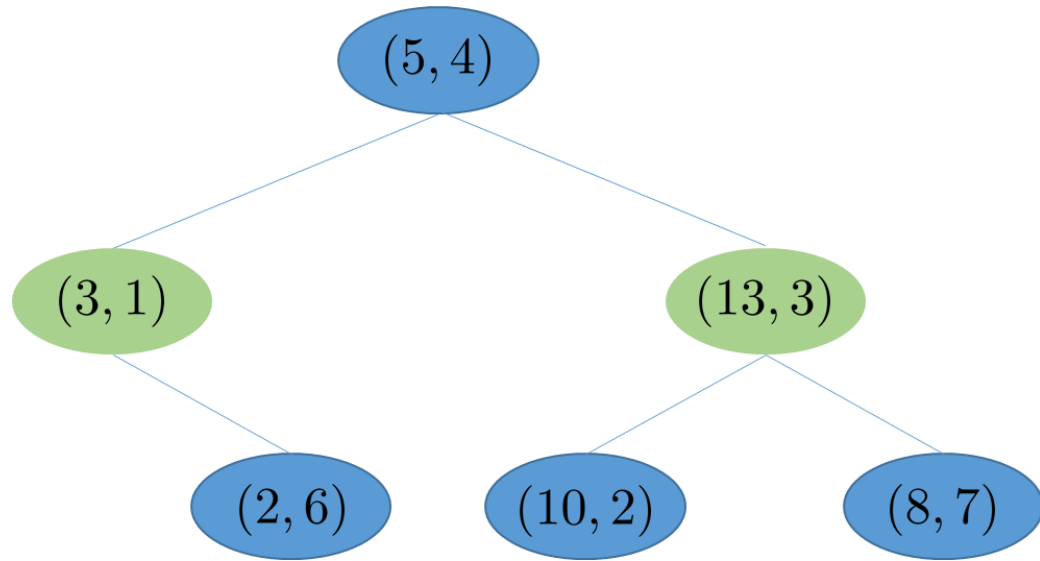
Consider $d = 2$, $\mathbf{x} = (x_1, x_2) \in \mathbf{R}^2$.



k-Nearest Neighbor (kNN) Algorithm

K-D Tree:

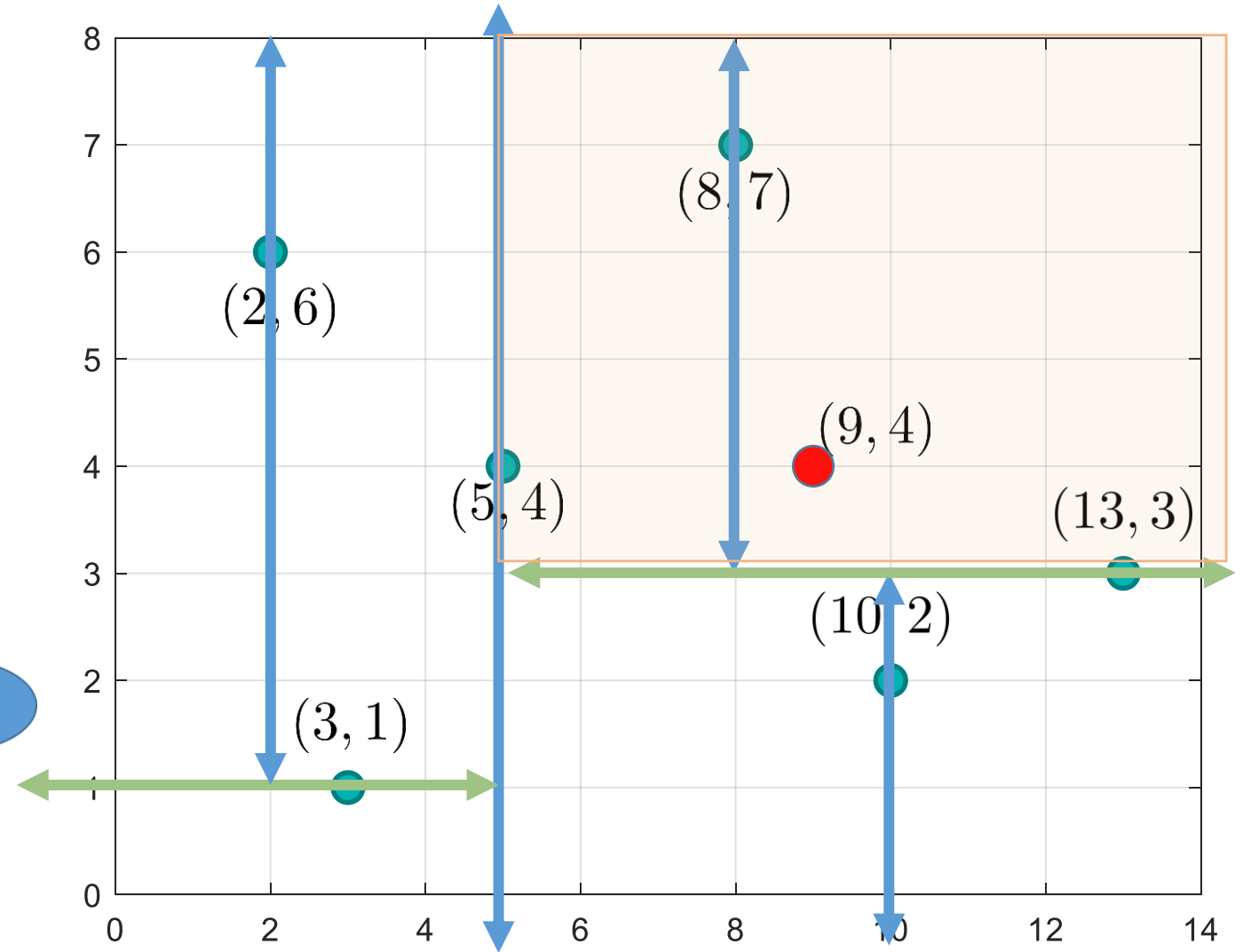
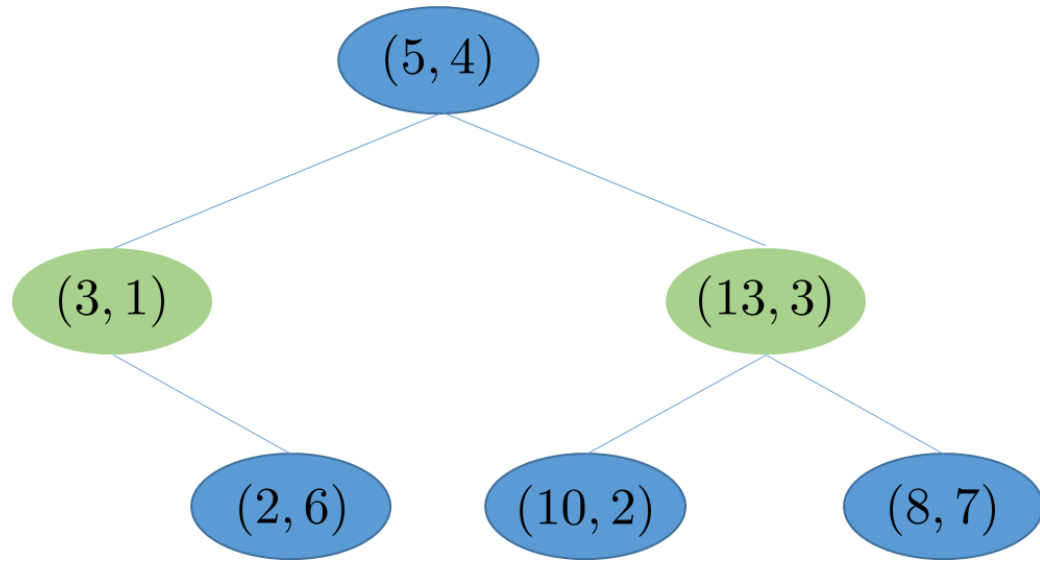
Example:



k-Nearest Neighbor (kNN) Algorithm

K-D Tree:

Connection with kNN:
Finding nearest neighbor



Issue: May miss neighbors! Trick to handle this.

k-Nearest Neighbor (kNN) Algorithm

K-D Tree - Summary:

- Enables significant reduction in the time complexity to support nearest neighbor algorithm.
 - Search to $O(\log n)$.
- Trade-offs:
 - Computational overhead to construct a tree $O(n \log n)$.
 - Space complexity: $O(n)$.
 - May miss neighbors.
 - Performance is degraded with the increase in the dimension of future space (Curse of Dimensionality).

Outline

- *k*-Nearest Neighbor (kNN) Algorithm Overview
- Algorithm Formulation
- Distance Metrics
- Choice of *k*
- Algorithm Convergence
- Storage, Time Complexity Analysis
- Fast kNN
- *The Curse of Dimensionality*

k-Nearest Neighbor (kNN) Algorithm

The Curse of Dimensionality:

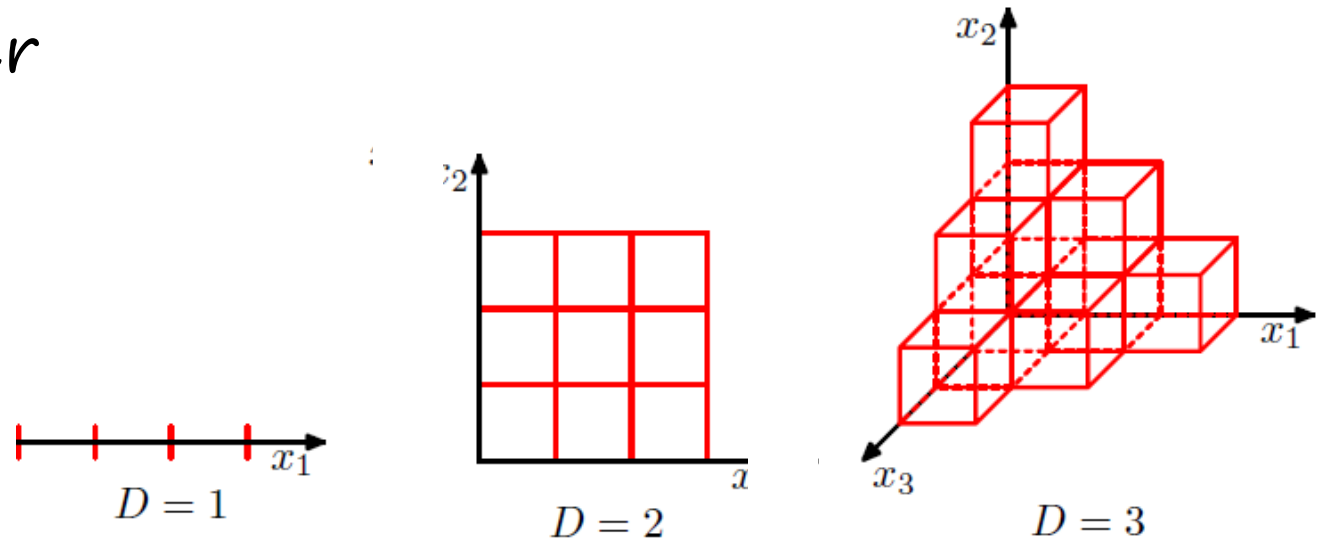
- Refers to the problems or phenomena associated with classifying, analyzing and organizing the data in high-dimensional spaces that do not arise in low-dimensional settings.
- For high-dimensional datasets, the size of data space is huge.
- In other words, the size of the feature space grows exponentially with the number of dimensions (d) of the data sets.
- To ensure the points stay close to each other, the size (n) of the data set must also have exponential growth. That means, we need a very large dataset to maintain the density of points in the high dimensional space.

k-Nearest Neighbor (kNN) Algorithm

The Curse of Dimensionality:

- For high-dimensional datasets, the size of data space is huge.

For an exponentially large number of cells, we need an exponentially large amount of training data to ensure that the cells are not empty.



Ref: CB

k-Nearest Neighbor (kNN) Algorithm

The Curse of Dimensionality:

Consider a ball of radius r defined as

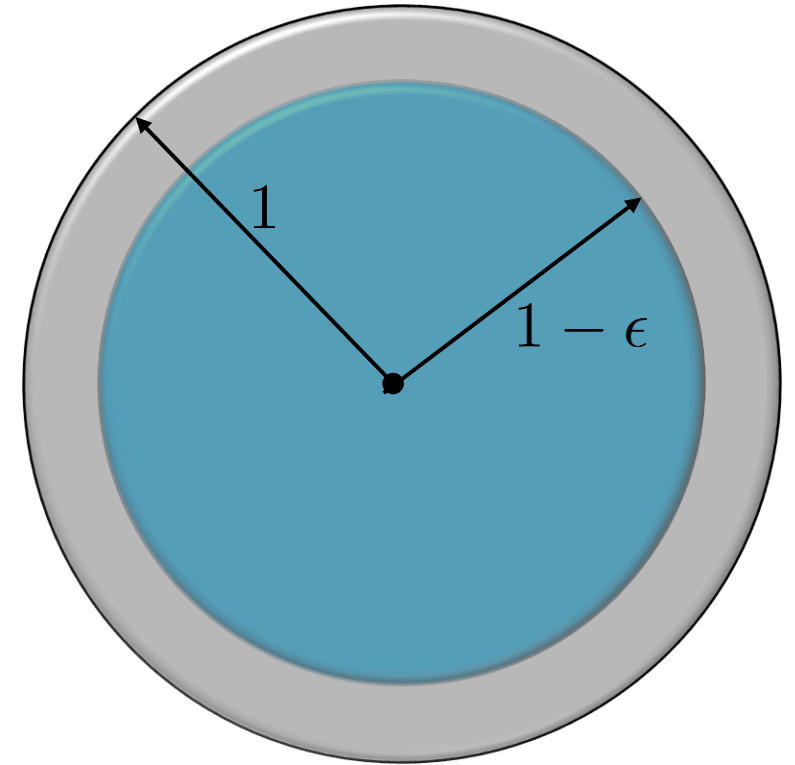
$$B(r) = \{\|\mathbf{x}\|_2 \leq r \mid \mathbf{x} \in \mathbf{R}^d\}$$

Volume of a ball of radius r

$$V(d) = K_D r^D$$

Fraction of a volume between the balls of radius 1
and radius $1 - \epsilon$

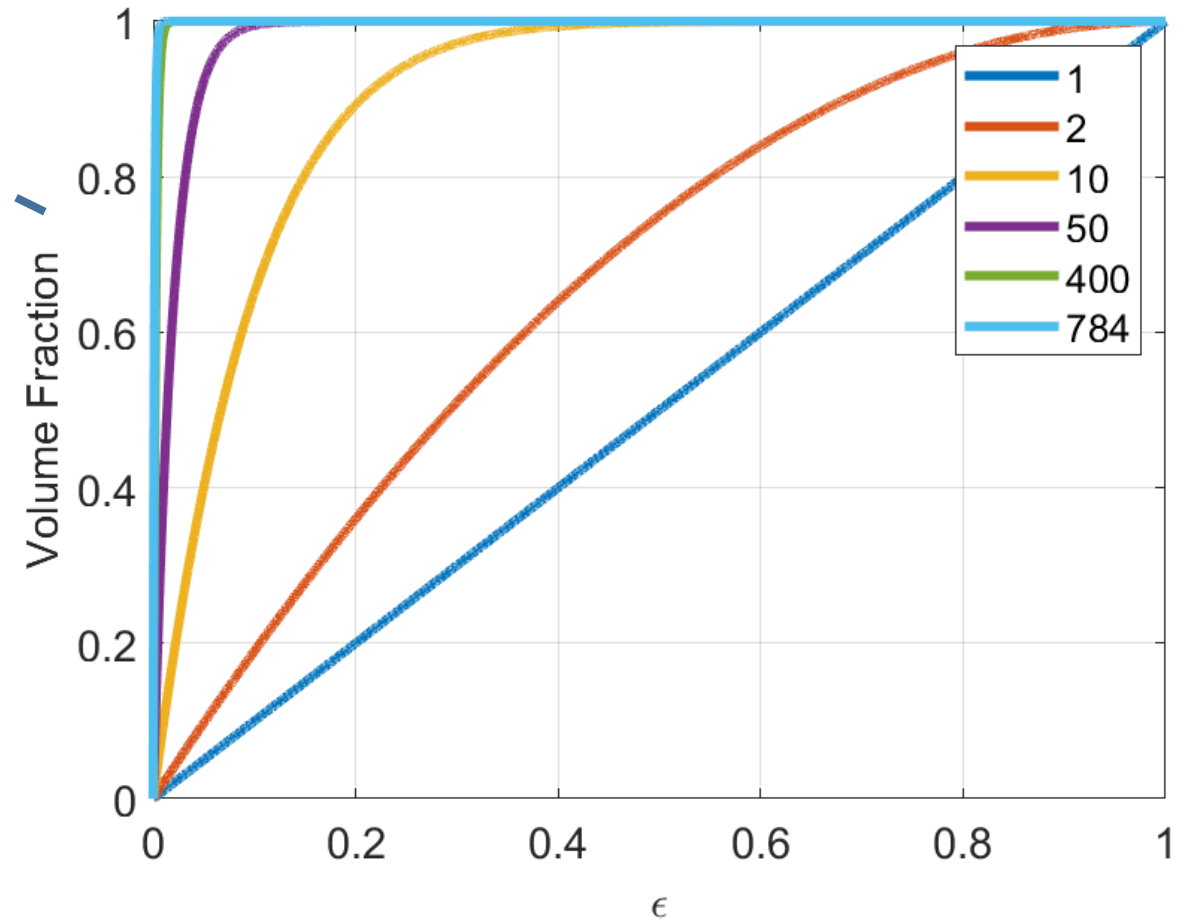
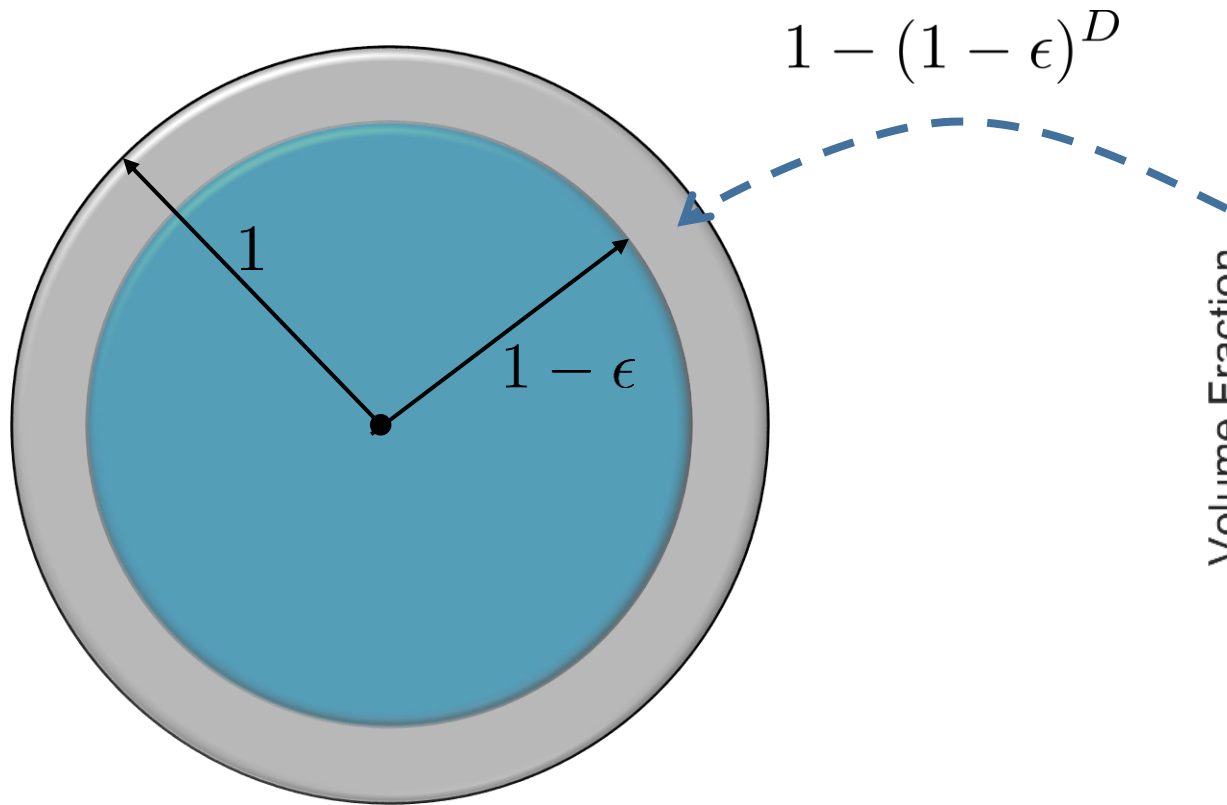
$$\frac{V(1) - V(1 - \epsilon)}{V(1)} = 1 - (1 - \epsilon)^D$$



$$K_D = \frac{\pi^{D/2}}{\Gamma(\frac{n}{2} + 1)}$$

k-Nearest Neighbor (kNN) Algorithm

The Curse of Dimensionality:



k-Nearest Neighbor (kNN) Algorithm

The Curse of Dimensionality (Another viewpoint):

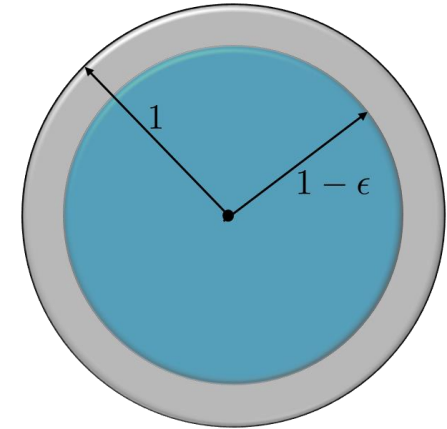
Calculate Probabilities that a uniformly distributed point is inside

$$\epsilon = 0.1$$

the shell: $1 - (1 - \epsilon)^D$

the inner ball: $(1 - \epsilon)^D$

D = 1	2	10	50	400	784
0.1	0.19	0.65	0.995	1.000	1.000
0.9	0.81	0.35	0.005	0.000	0.000



For $D = 50$, 5 out of 1000 data-points would be inside the inner ball.

For $D = 400$, $(1 - \epsilon)^D = 4.9774e - 19$; almost all points lie on the surface of the ball.

If you take a test point on the origin and $D = 400$, (almost) every point is at the same (Euclidean) distance from the origin.

k-Nearest Neighbor (kNN) Algorithm

The Curse of Dimensionality (Another viewpoint):

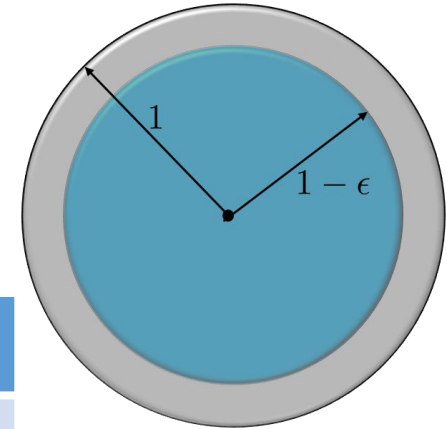
Calculate Probabilities that a uniformly distributed point is inside

$$\epsilon = 0.01$$

the shell: $1 - (1 - \epsilon)^D$

the inner ball: $(1 - \epsilon)^D$

D = 1	2	10	50	400	784
0.01	0.02	0.096	0.395	0.982	0.999
0.99	0.98	0.904	0.605	0.018	0.0004



k-Nearest Neighbor (kNN) Algorithm

The Curse of Dimensionality:

Connection with kNN:

- With the increase in the number of features or number of dimensions of the feature space, data-points are never near to one another.
- kNN algorithm carries out predictions about the test point assuming we have data-points near to the test point that are similar to the test point.
- As we do not have neighbors in the high dimensional space, kNN becomes vulnerable and sensitive to the Curse of Dimensionality.

k-Nearest Neighbor (kNN) Algorithm

The Curse of Dimensionality: Why does kNN work?

Two related explanations;

- Real-world data in the higher dimensional space is confined to a region with effective lower dimensionality.
 - **Dimensionality Reduction (to be covered later in the course)**
- Real-world data exhibits smoothness that enables us to make predictions exploiting interpolation techniques.
- For example,
 - Data along a line or a plane in higher dimensional space
 - detection of orientation of object in an image; data lies on effectively 1 dimensional manifold in probably 1million dimensional space.
 - Face recognition in an image (50 or 71 features).
 - Spam filter

k-Nearest Neighbor (kNN) Algorithm

Reference:

Overall:

- <https://www.cs.cornell.edu/courses/cs4780/2018fa/>
- CB: sec 1.1
- HTF: 13.3 up to end of 13.3.2
- The curse of dimensionality
 - CB: 1.4
 - KM: 1.4.3
 - N. Kouroukidis and G. Evangelidis, "The Effects of Dimensionality Curse in High Dimensional kNN Search," 2011 15th Panhellenic Conference on Informatics, Kastoria, 2011, pp. 41-45, doi: 10.1109/PCI.2011.45.

Feedback: Questions or Comments?

Email: zubair.khalid@lums.edu.pk