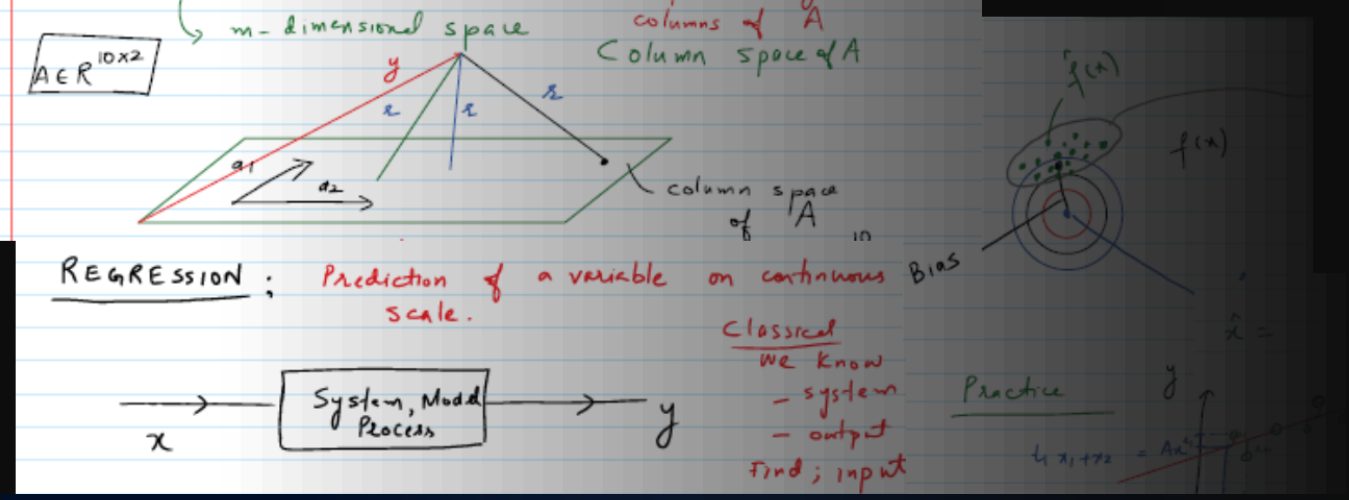


Machine Learning

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_2023.html

LUMS
A Not-for-Profit University

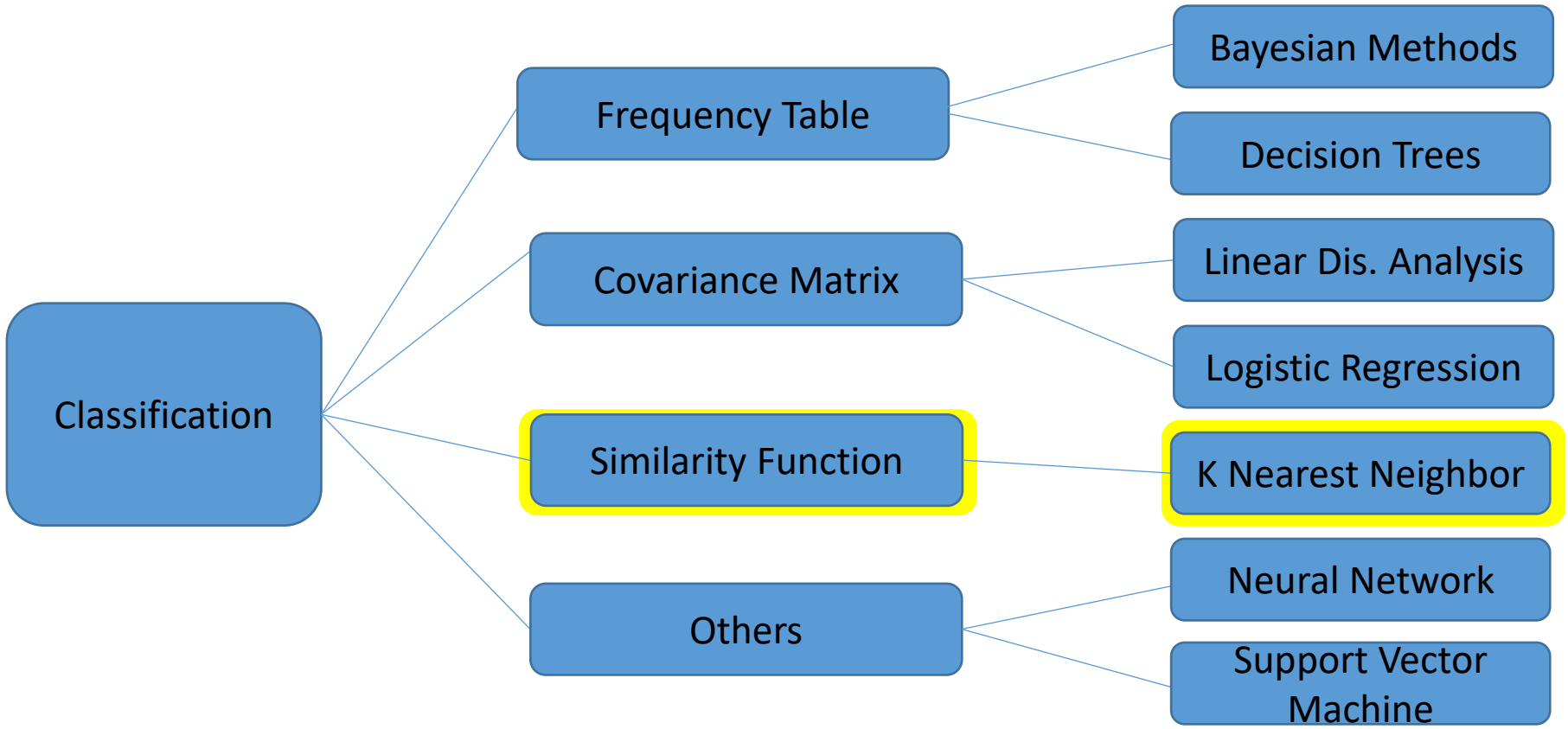
Outline

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

Supervised Learning

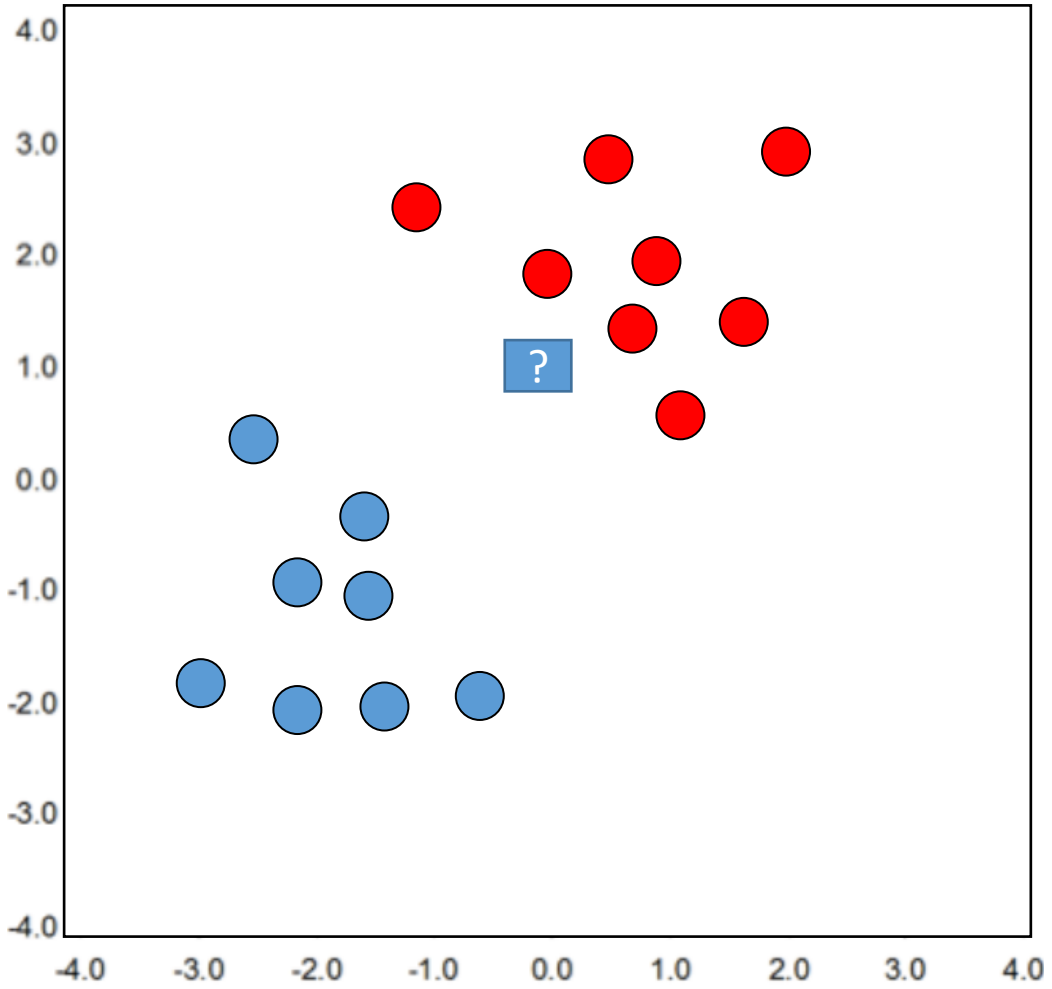
Classification Algorithms or Methods

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.

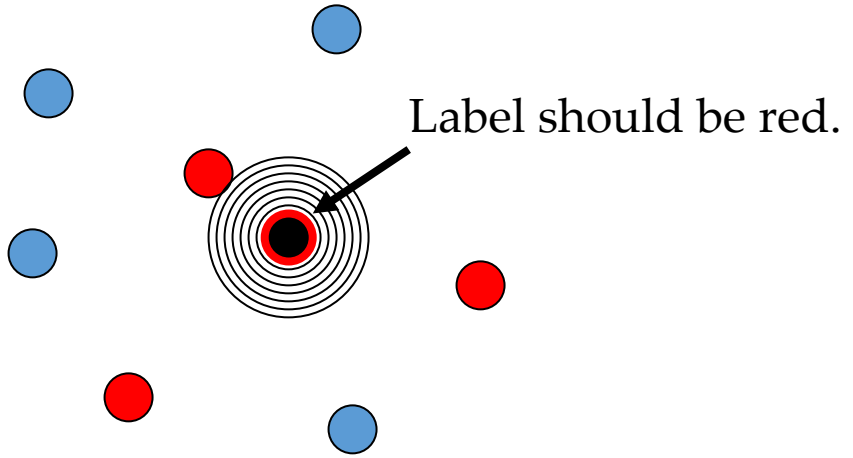
Questions:

- 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

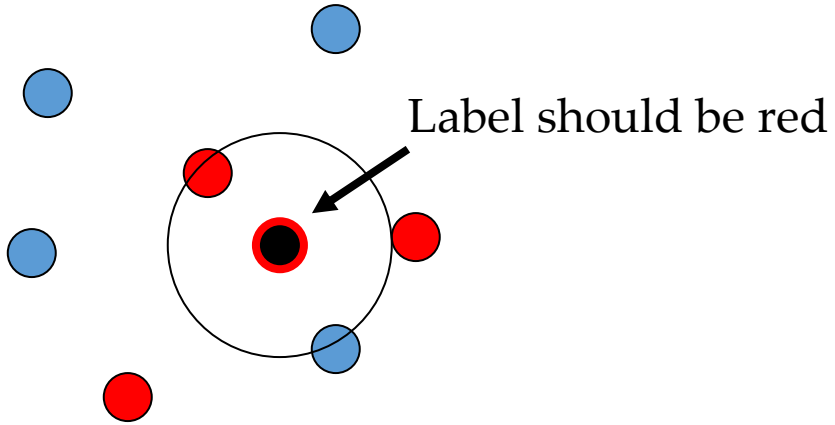
1-Nearest Neighbor:

Simplest ML Classifier
Idea: Use the label of the **closest** known point

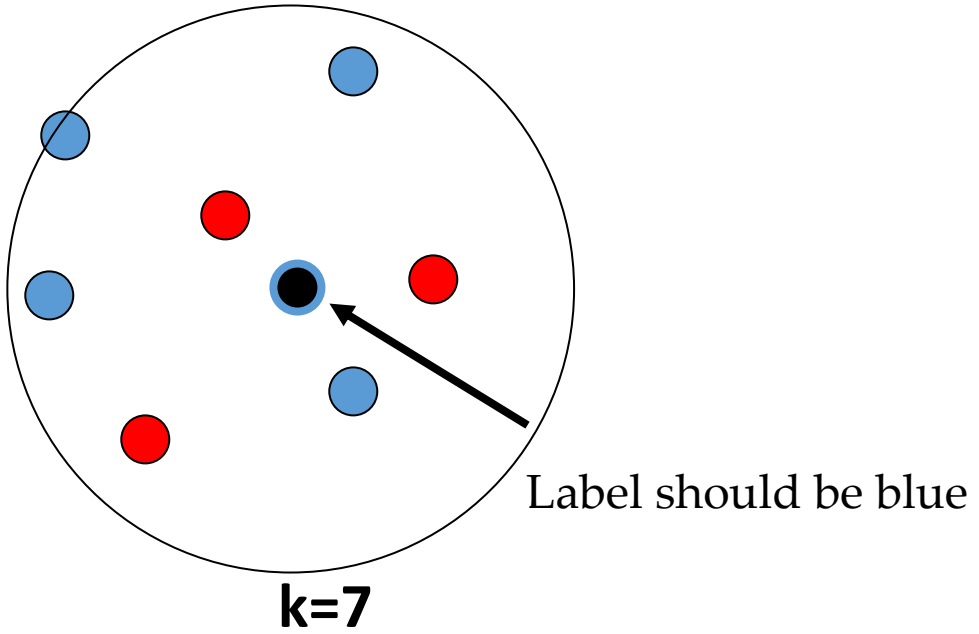


Generalization:

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

$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

- $\mathcal{Y} = \{1, 2, \ldots, M\}$ (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

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

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 $S_{\mathbf{x}}$

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

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

k-Nearest Neighbor (kNN) Algorithm

Decision Boundary:

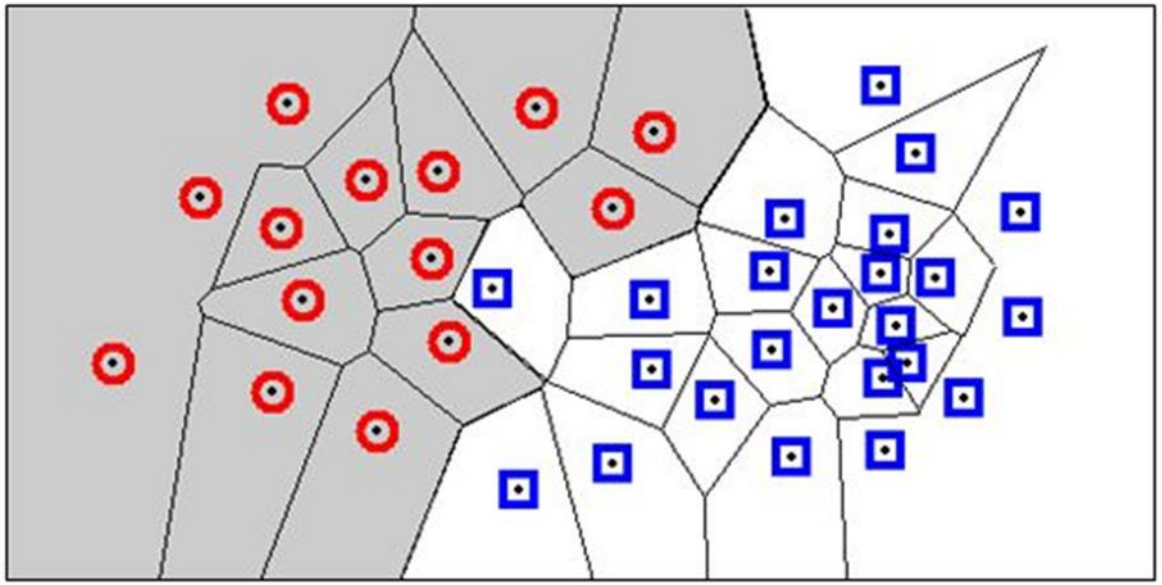
For $k = 1$, kNN defines a region, called decision boundary or region, in the space. Such division of the featue space is referred to s Voronoi partitioning.

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) Algroithm 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

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

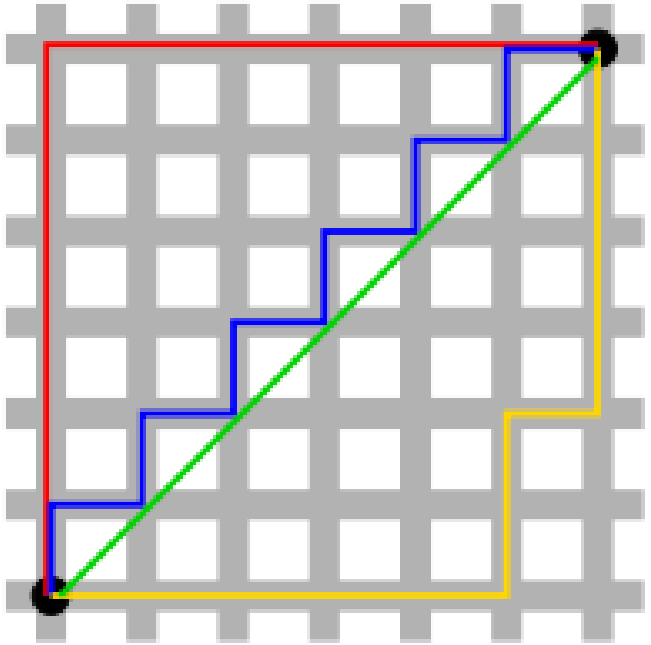
$$\text{dist}(\mathbf{x}, \mathbf{x}') \geq \max_{(\mathbf{x}'', y'') \in S_{\mathbf{x}}} \text{dist}(\mathbf{x}, \mathbf{x}''), \quad \forall (\mathbf{x}', y') \in D \backslash S_{\mathbf{x}}$$

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 $\mathbf{x} \in \mathbf{R}^d$

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

Properties of Norm

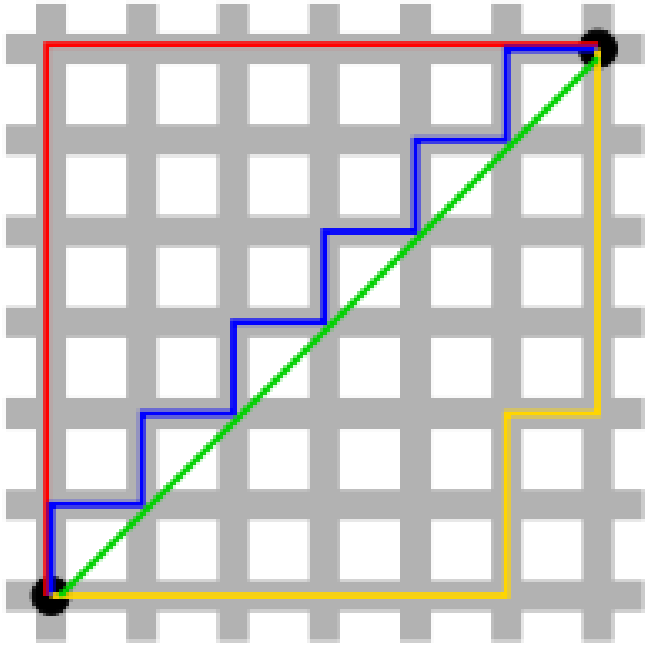
Non-negative, $\|\mathbf{x}\|_p \geq 0$

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

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

- Minkowski

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

$p = 2$

$p = 1$

$p = \infty$

$$\mathrm{dist}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_\infty = \max_{i=1,2,\ldots,d} (|x_i - x_i'|)$$ 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 vaiable, 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 meaasure 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 scaler. The values should be of the same order along each dimension.

  - Solution: Data Normalization

Outline

- k-Nearest Neighbor (kNN) Algroithm 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) Algroithm 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 classifer 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

$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

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 $n \to \infty$.

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)

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

Error Rate:

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

Reference: Cover, Thomas, and, Hart, Peter. Nearest neighbor pattern classification[J]. IEEE Transactions on Information Theory, 1967, 13(1): 21-27

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:

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

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

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

Error Rate:

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

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

k-Nearest Neighbor (kNN) Algorithm

Error Convergence:

Bound on Error Rate:

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

Using

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

We obtain

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

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

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) Algroithm 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

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

- 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) Algroithm 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 later)
  - 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

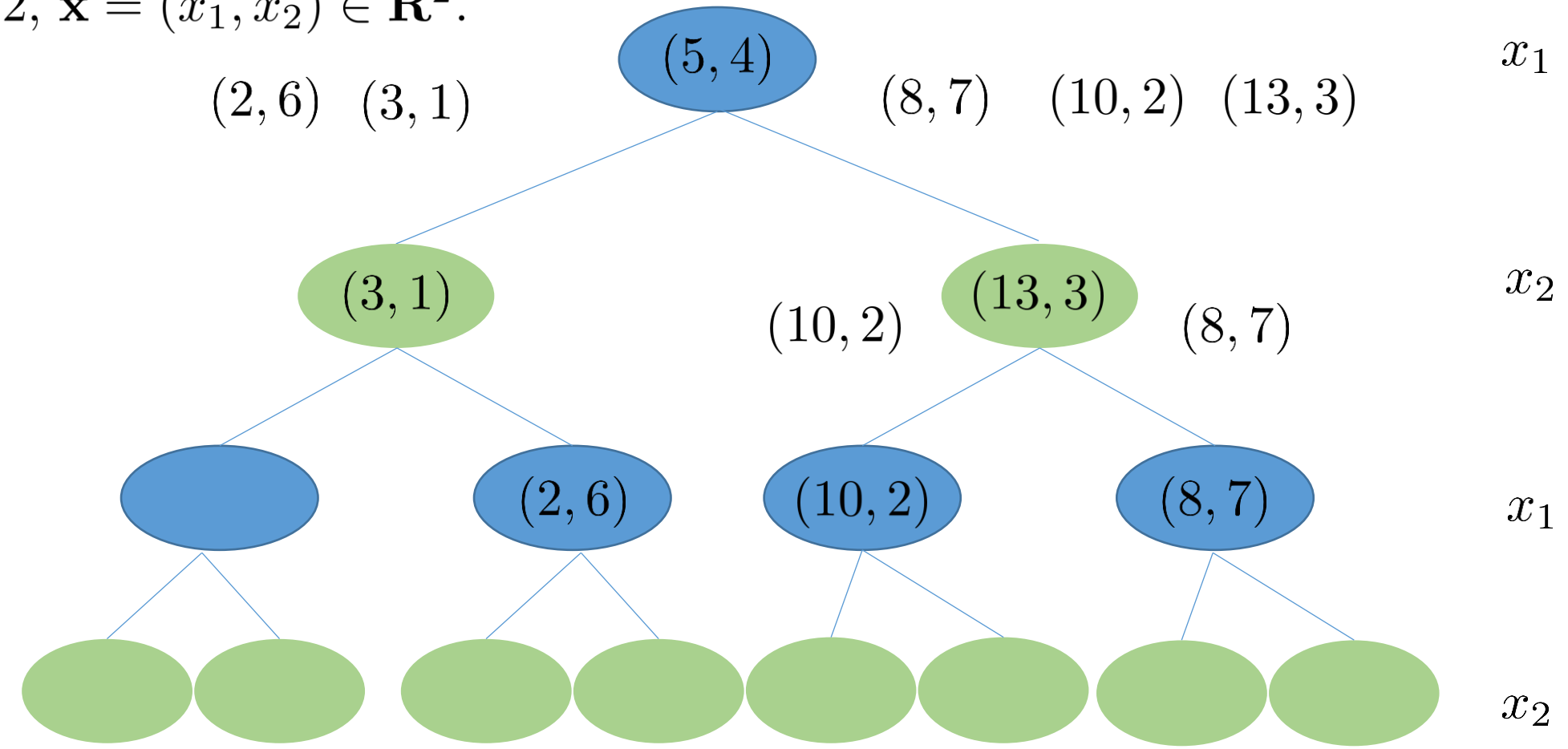
K-D Tree:

Example:

Splitting dimension

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

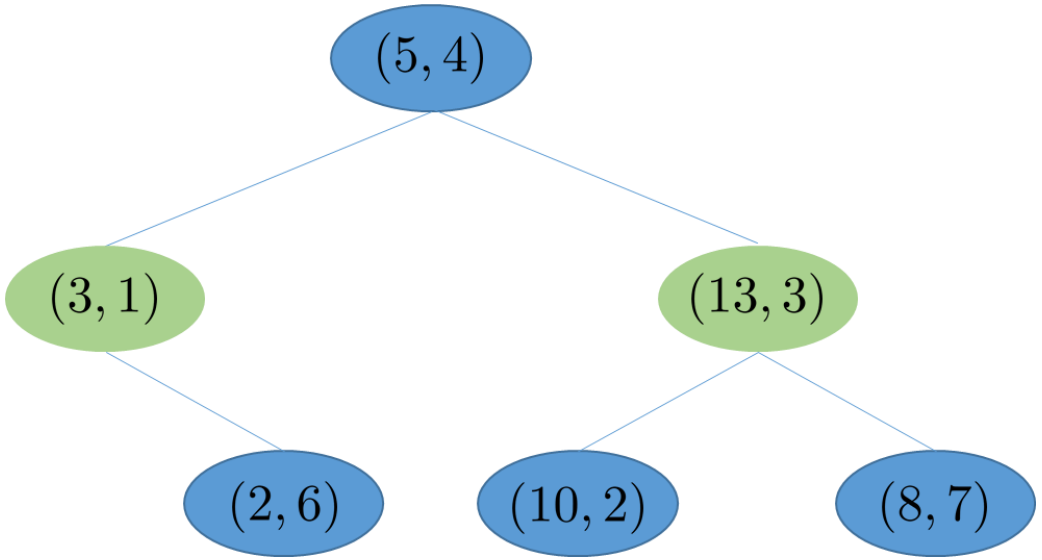
$(3,1)$
$(5,4)$
$(2,6)$
$(10,2)$
$(13,3)$
$(8,7)$

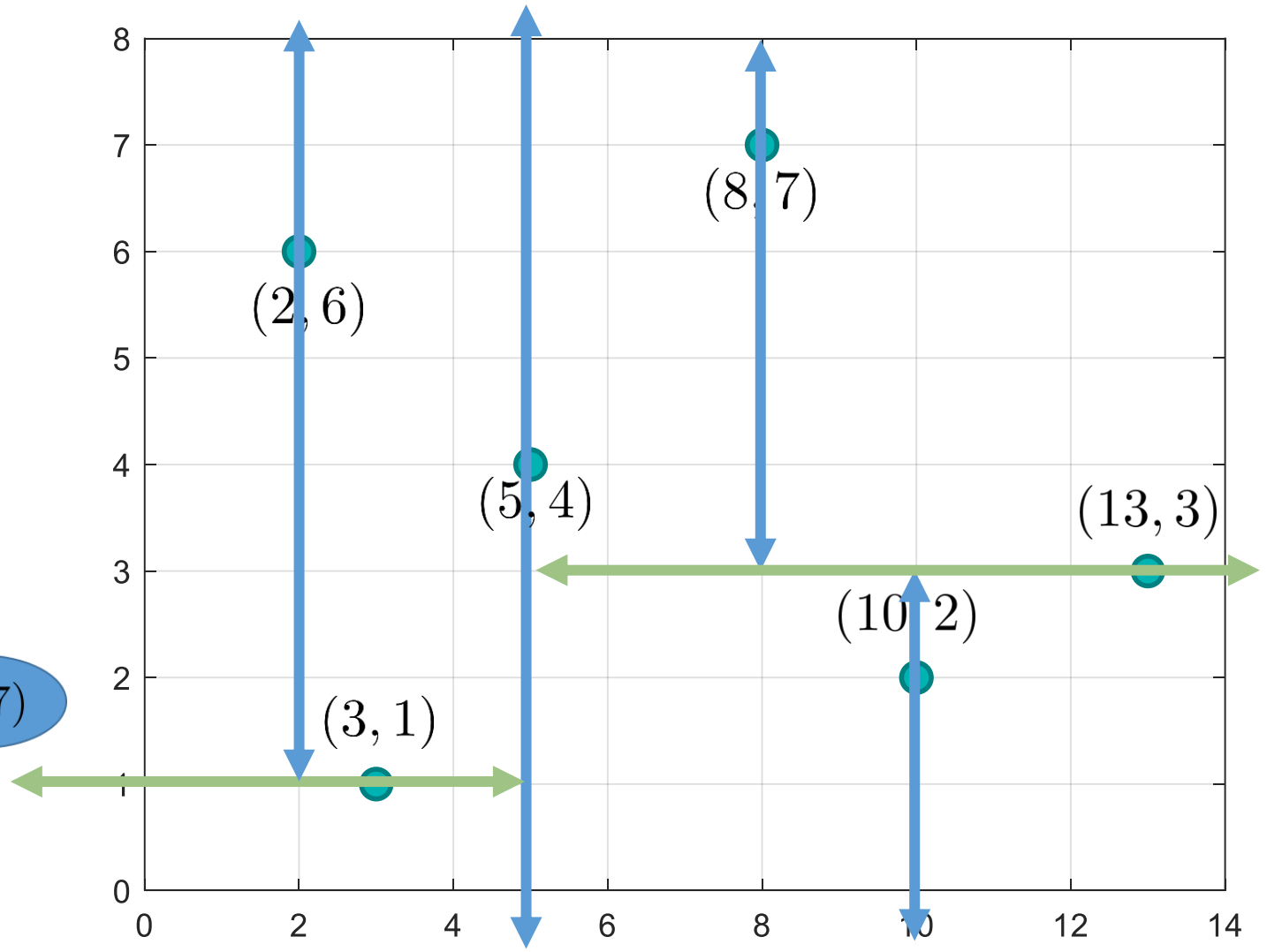


k-Nearest Neighbor (kNN) Algorithm

K-D Tree:

Example:

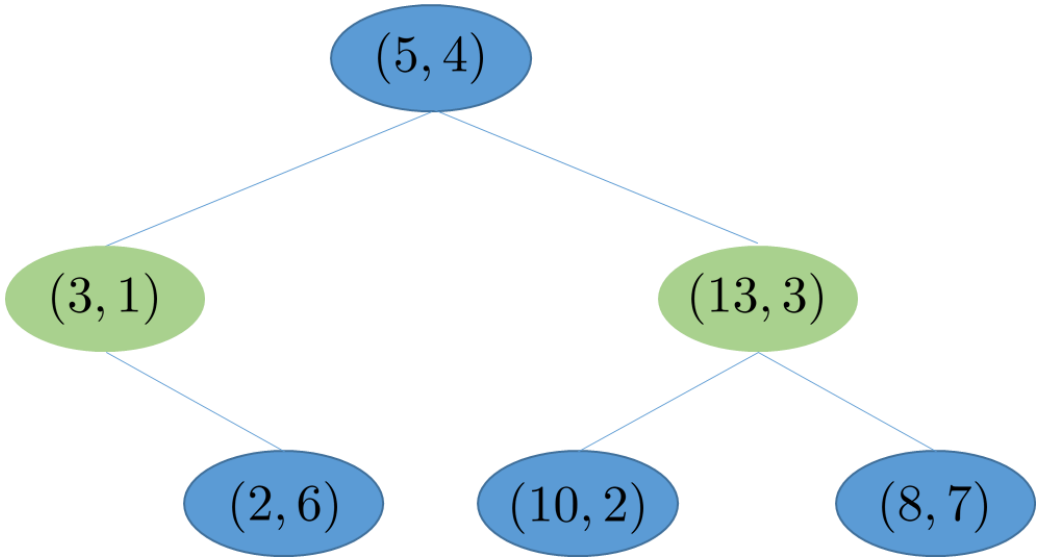


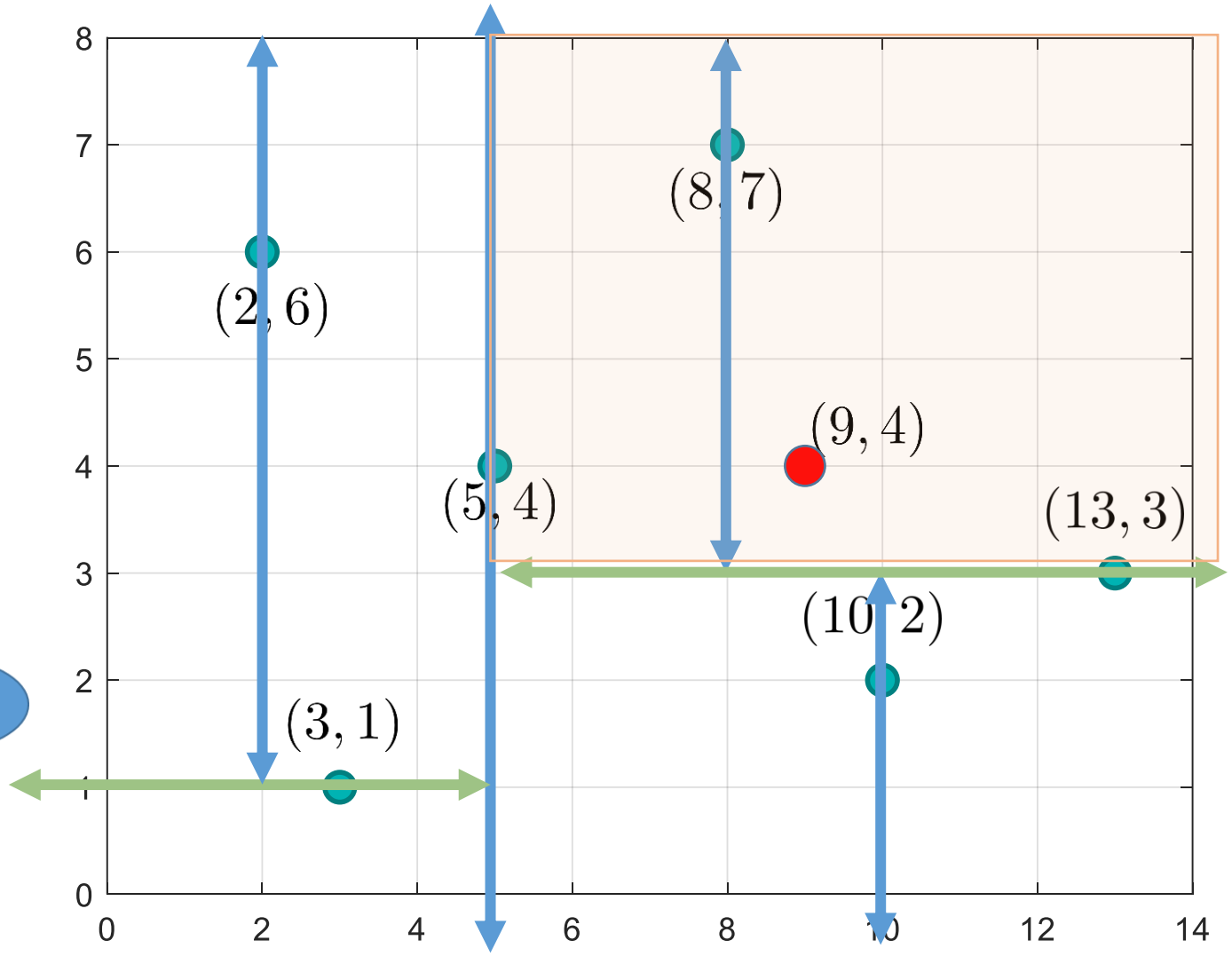


k-Nearest Neighbor (kNN) Algorithm

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(logn).

- Trade-offs:
  - Computational overhead to construct a tree O(n logn).
  - 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) Algroithm 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.

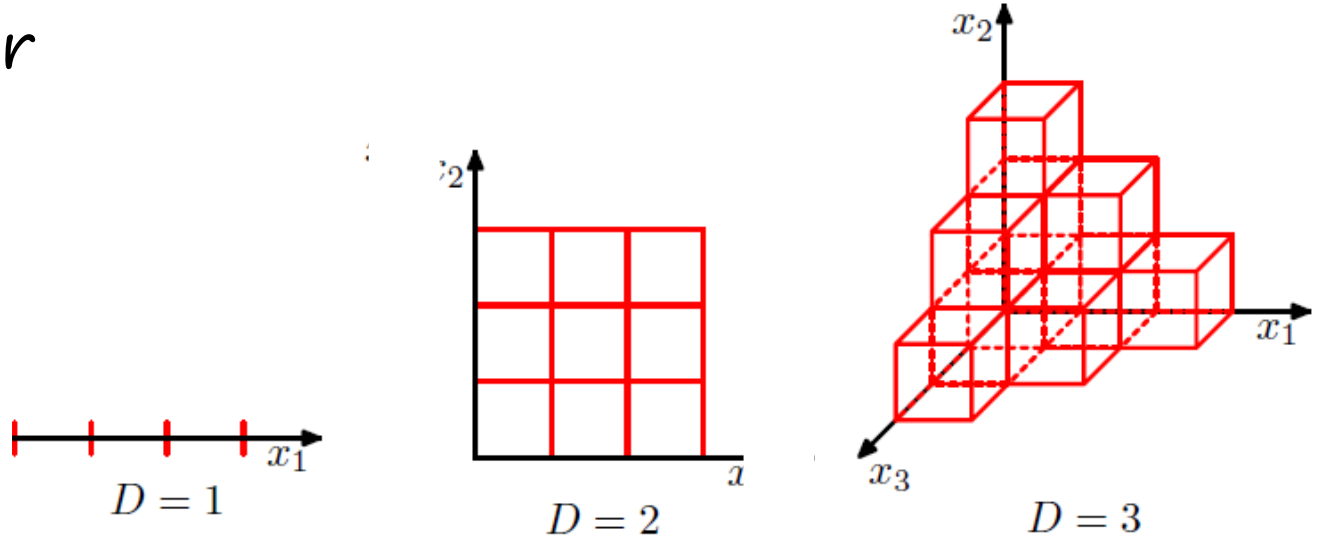
- To ensure the points stay close to each other, the size (n) of the data set must also have exponential growth. That means, we need a very large dataset to maintain the density of points in the high dimensional space.

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.



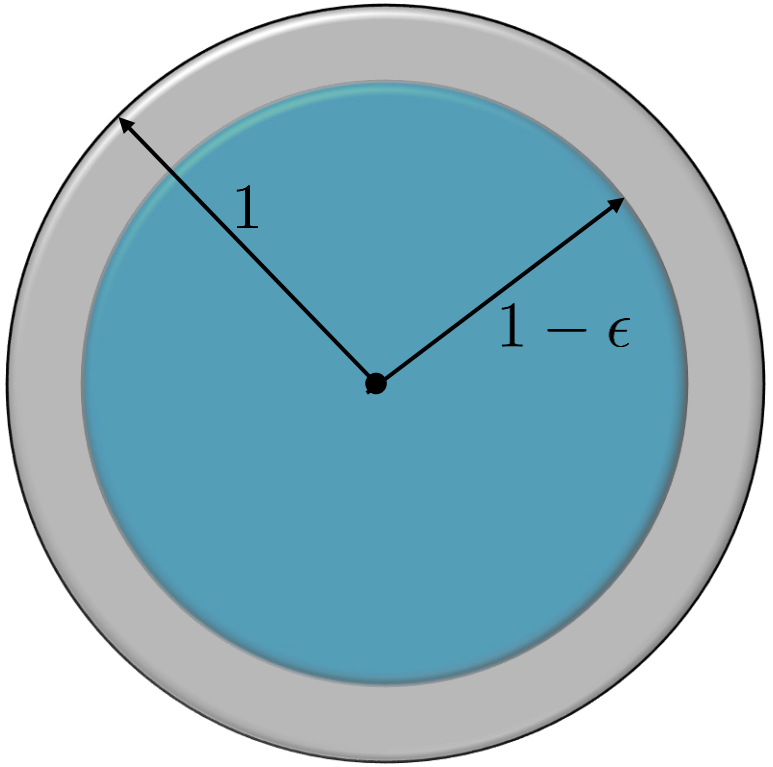
Ref: CB

k-Nearest Neighbor (kNN) Algorithm

The Curse of Dimensionality:

Consider a ball of radius r defined as

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



Volume of a ball of radius r

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

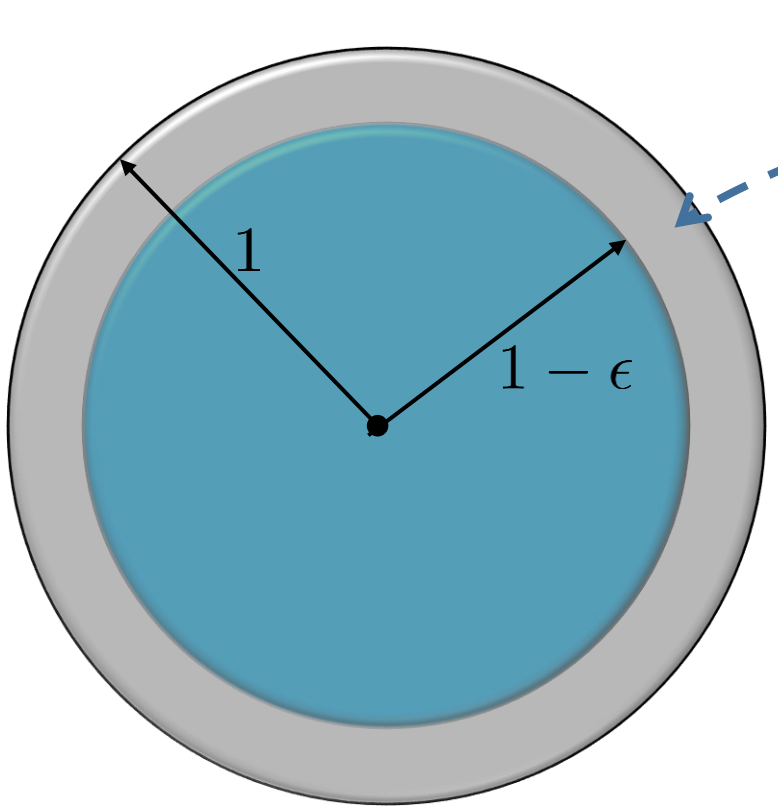
Fraction of a volume between the balls of radius 1 and radius $1-\epsilon$

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

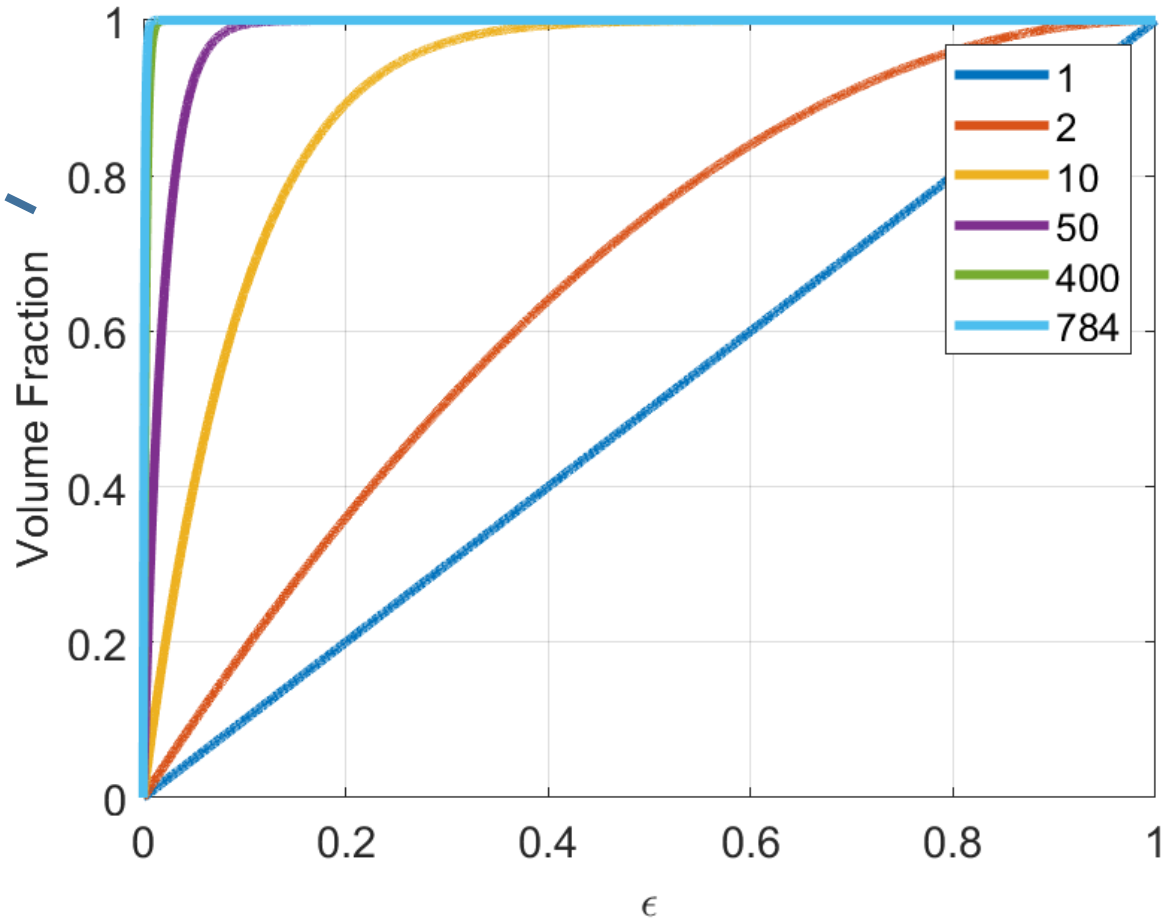
$$K_D = \frac{\pi^{D/2}}{\Gamma(\frac{n}{2}+1)}$$

k-Nearest Neighbor (kNN) Algorithm

The Curse of Dimensionality:

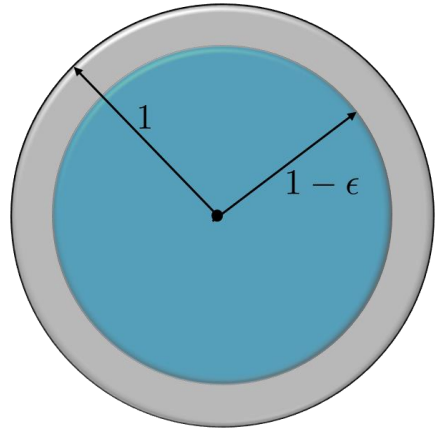


$$1-(1-\epsilon)^D$$



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

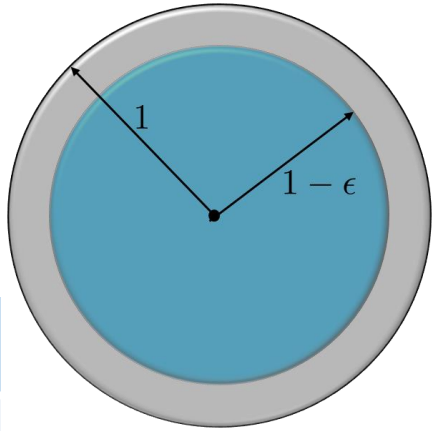
For $D = 50$, 5 out of 1000 data-points would be inside the inner ball.

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:

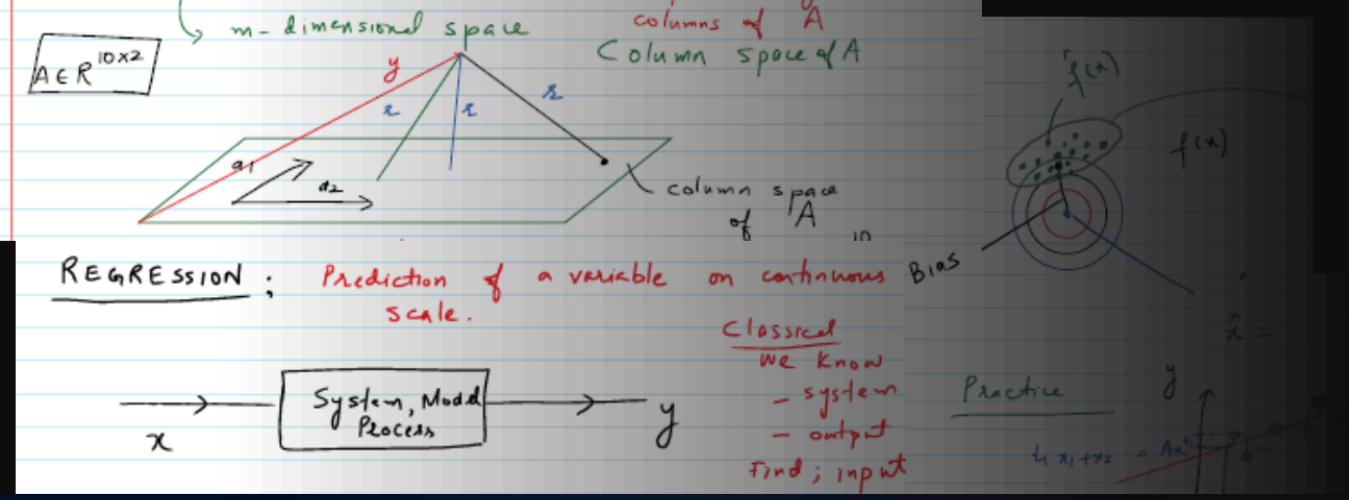
Overall:

- 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. Kouiroukidis and G. Evangelidis, "The Effects of Dimensionality Curse in High Dimensional kNN Search," 2011 15th Panhellenic Conference on Informatics, Kastonia, 2011, pp. 41-45, doi: 10.1109/PCI.2011.45.



Machine Learning

EE514 – CS535

Dimensionality Reduction: Feature Selection and Feature Extraction (PCA)

Zubair Khalid

School of Science and Engineering
Lahore University of Management Sciences

https://www.zubairkhalid.org/ee514_2023.html

LUMS
A Not-for-Profit University

Outline

- Dimensionality Reduction
- Feature Selection
- Feature Extraction – PCA

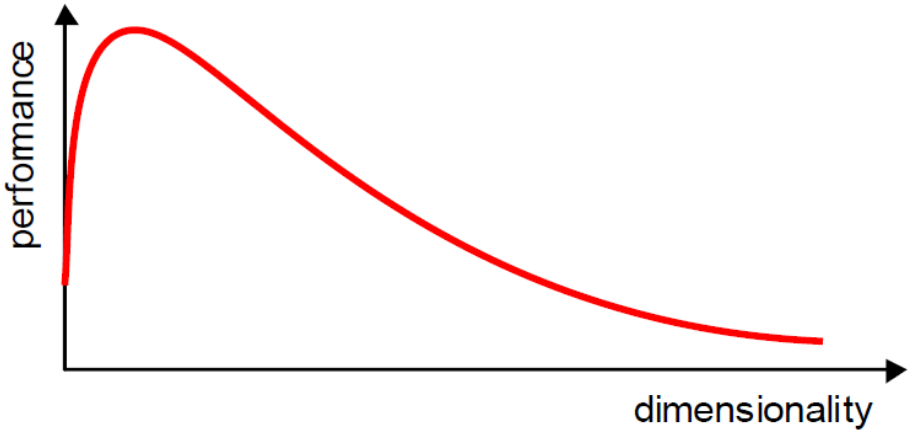
Dimensionality Reduction

Why?

- Increasing the number of inputs or features does not always improve accuracy of classification.

- Performance of classifier may degrade with the inclusion of irrelevant or redundant features.

- Curse of dimensionality; "Intrinsic" dimensionality of the data may be smaller than the actual size of the data.



Benefits:

- Improve the classification performance.

- Improve learning efficiency and enable faster classification.

- Better understanding of the underlying process mapping inputs to output.

Dimensionality Reduction

Feature Selection and Feature Extraction:

Given a set of features, reduce the number of features such that "the learning ability of the classifier" is maximized.

$$\mathbf{x} = [x_1, x_2, \ldots, x_d]$$

Feature Selection:

Select a subset of the existing features.

$$\mathbf{x} = [x_1, x_2, \ldots, x_d]$$

$$\mathbf{z} = [x_{i_1}, x_{i_2}, \ldots, x_{i_k}]$$

Feature Extraction:

Transform existing features to obtain a set of new features using some mapping function.

$$\mathbf{x} = [x_1, x_2, \ldots, x_d]$$

$$\mathbf{z} = f(\mathbf{x})$$

$$\mathbf{z} = [z_1, z_2, \ldots, z_k]$$

$$k \ll d$$

Dimensionality Reduction

Feature Selection:

Select a subset of the existing features.

$$\mathbf{x} = [x_1, x_2, \ldots, x_d]$$

$$\mathbf{z} = [x_{i_1}, x_{i_2}, \ldots, x_{i_k}]$$

Select the features in the subset that either improves classification accuracy or maintain same accuracy.

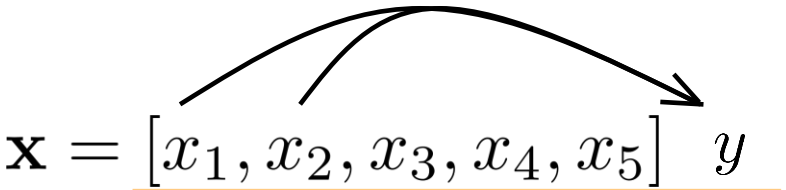
How many subsets do we have?

How do we choose this subset?

Dimensionality Reduction

Feature Selection:

Example:



$$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5] \quad y$$

0	0	1	0	1	0
0	1	0	0	1	1
1	0	1	0	1	1
1	1	0	0	1	1
0	0	1	1	0	0
0	1	0	1	0	1
1	0	1	1	0	1
1	1	0	1	0	1

* Source: A tutorial on genomics by Yu (2004).

Data set:

- Five Boolean features
- $y = x_1$ (or) x_2
- x_3 = (not) x_2
- x_4 = (not) x_5

Optimal subset:

$\{x_1, x_2\}$ or $\{x_1, x_3\}$

Optimization in space of all feature subsets would have

$$2^d \text{ possibilities}$$

Can't search over all possibilities and therefore we rely on heuristic methods.

Dimensionality Reduction

Feature Selection:

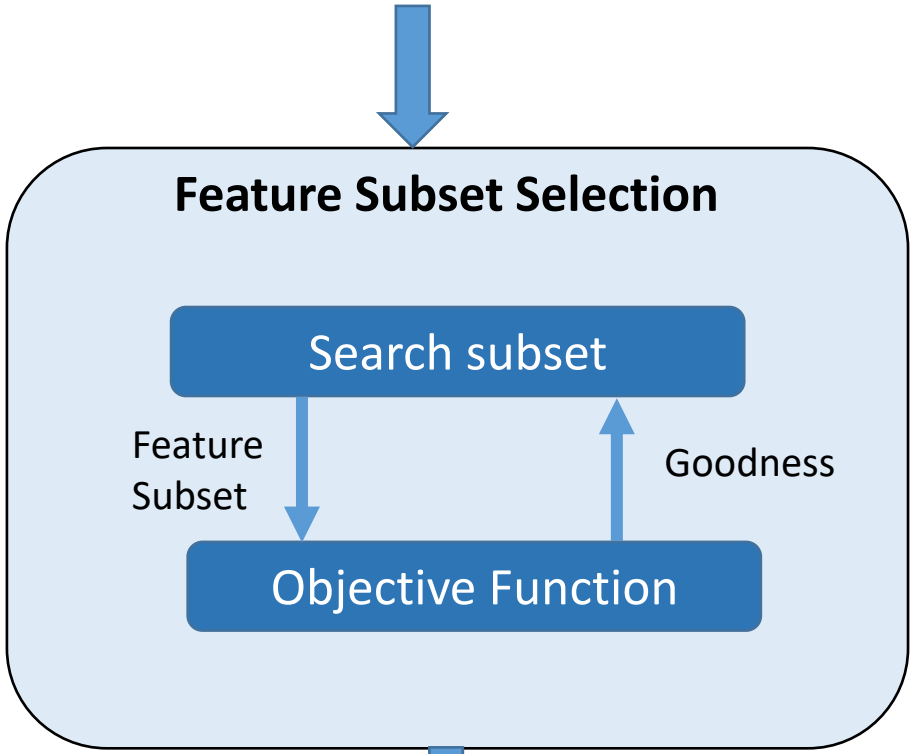
$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

How do we choose this subset?

- Feature selection can be considered as an optimization problem that involves
  - Searching of the space of possible feature subsets
  - Choose the subset that is optimal or near-optimal with respect to some objective function

- **Filter Methods** (unsupervised method)
  - Evaluation is independent of the learning algorithm
  - Consider the input only and select the subset that has the most information

- **Wrapper Methods** (supervised method)
  - evaluation is carried out using model selection the machine learning algorithm
  - Train on selected subset and estimate error on validation dataset



$$D = \{(\mathbf{z_1}, y_1), (\mathbf{z_2}, y_2), \ldots, (\mathbf{z_n}, y_n)\} \subseteq \mathcal{X}^k \times \mathcal{Y}$$

$$\mathbf{z} = [x_{i_1}, x_{i_2}, \ldots, x_{i_k}]$$

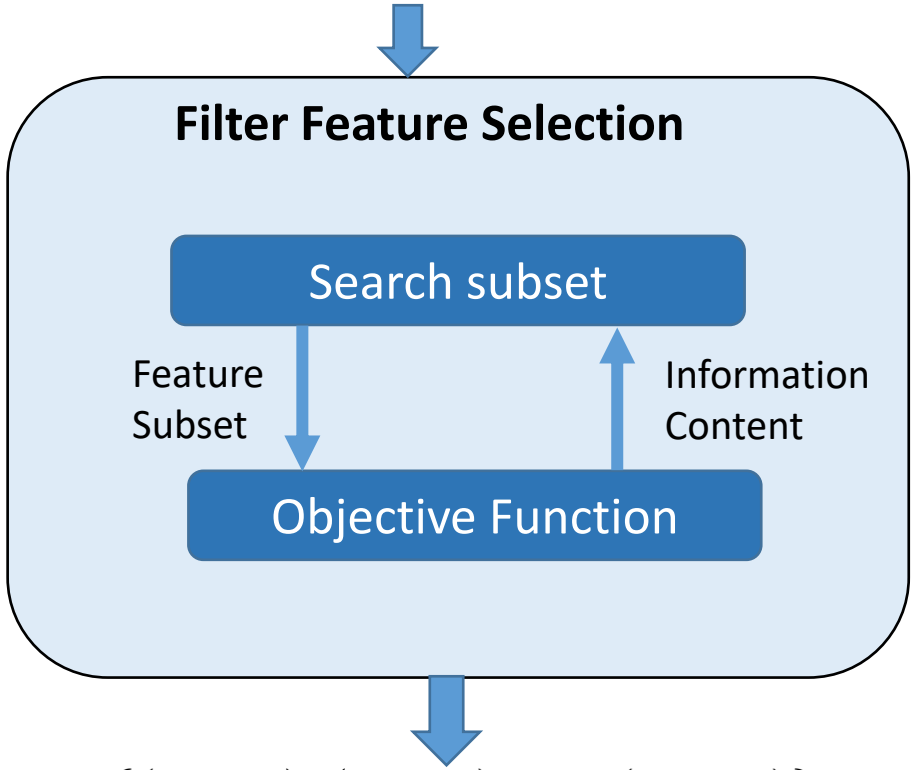
Dimensionality Reduction

Feature Selection:

How do we choose this subset?

Filter Methods

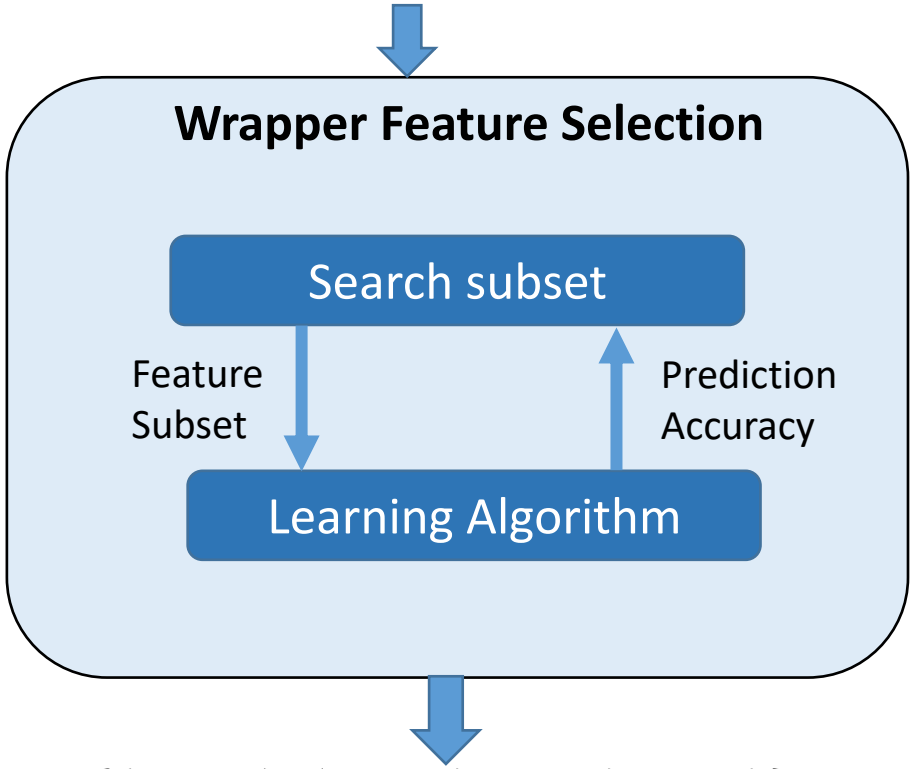
$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$



$$D = \{(\mathbf{z_1}, y_1), (\mathbf{z_2}, y_2), \ldots, (\mathbf{z_n}, y_n)\} \subseteq \mathcal{X}^k \times \mathcal{Y}$$

Wrapper Methods

$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$



$$D = \{(\mathbf{z_1}, y_1), (\mathbf{z_2}, y_2), \ldots, (\mathbf{z_n}, y_n)\} \subseteq \mathcal{X}^k \times \mathcal{Y}$$

Dimensionality Reduction

Feature Selection:

Filters Method:

- *Univariate Methods*
  - Treats each feature independently of other features

- Calculate score of each feature against the label using the following metrics:
  - Pearson correlation coefficient
  - Mutual Information
  - F-score
  - Chi-square
  - Signal-to-noise ratio (SNR), etc.

- Rank features with respect to the score

- Select the top k-ranked features (k is selected by the user)

Dimensionality Reduction

Feature Selection:

Filters Method – Ranking Metrics:

- *Pearson correlation coefficient (measure of linear dependence)*

Denote feature values by a vector $\mathbf{a} \in \mathbf{R}^n$ (Note n is the number of points).

Denote labels by a vector $\mathbf{y} = [y_1, y_2, \ldots, y_n]$.

Define Pearson correlation coefficient as

$$\rho = \frac{\tilde{\mathbf{a}}^T \tilde{\mathbf{y}}}{\|\tilde{\mathbf{a}}\|_2 \tilde{\mathbf{y}}\|_2}, \quad |\rho| \leq 1$$

Here

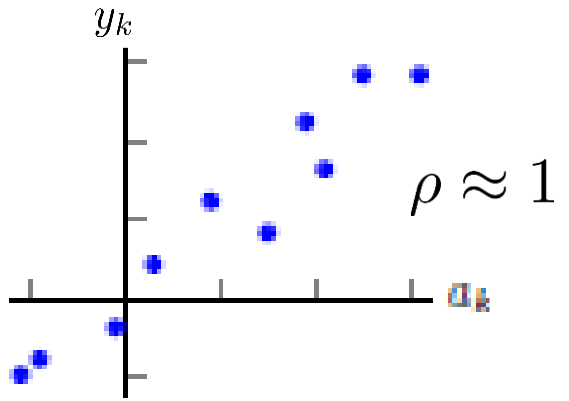
$$\tilde{\mathbf{a}} = \mathbf{a} - \text{avg}(\mathbf{a})\mathbf{1}$$

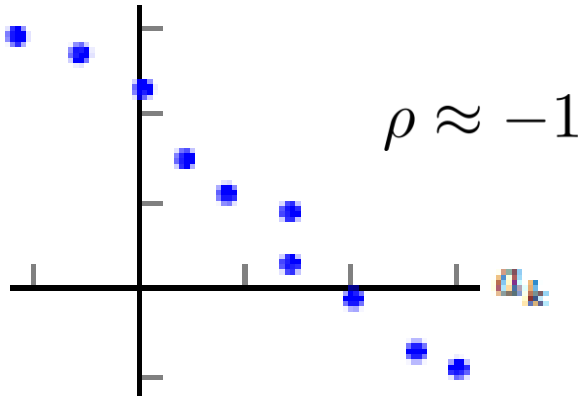
is a demeaned vector and is obtined by subtracting mean of a vector from it.

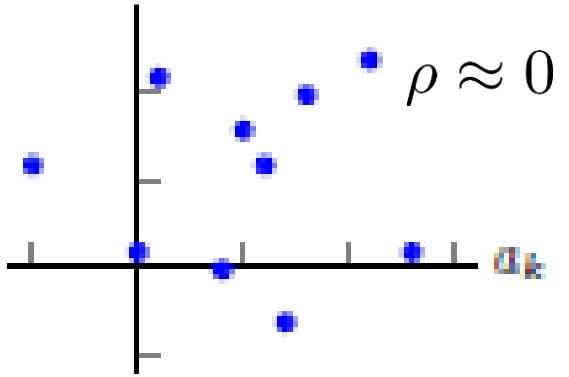
- *Signal-to-noise ratio (SNR)*

$$\text{SNR} = \frac{\text{avg}(\mathbf{a}) - \text{avg}(\mathbf{y})}{\text{std}(\mathbf{a}) - \text{std}(\mathbf{y})},$$

where std denotes the standard deviation of the vector.







Dimensionality Reduction

Feature Selection:

Wrappers Method:

- Forward Search Feature Subset Selection Algorithm (Super intuitive)

  - Start with empty set as feature subset
  - Try adding one feature from the remaining features to the subset
  - Estimate classification or regression error for adding each feature
  - Add feature to the subset that gives max improvement

- Backward Search Feature Subset Selection Algorithm (Super intuitive)

  - Start with full feature set as subset
  - Try removing one feature from the subset
  - Estimate classification or regression error for removing each feature
  - Remove/drop the feature that gives minimal impact on error or reduces the error

Outline

- Dimensionality Reduction
- Feature Selection
- Feature Extraction – PCA

Dimensionality Reduction

Feature Extraction:

Transform existing features to obtain a set of new features using some mapping function.

$$\mathbf{x} = [x_1, x_2, \ldots, x_d]$$

$$\mathbf{z} = f(\mathbf{x})$$

$$\mathbf{z} = [z_1, z_2, \ldots, z_k]$$

- The mapping function **z**=*f*(**x**) can be **linear** or non-linear.

- Can be interpreted as projection or mapping of the data in the higher dimensional space to the lower dimensional space.

- Mathematically, we want to find an optimum mapping **z**=*f*(**x**) that preserves the desired information as much as possible.

Dimensionality Reduction

Feature Extraction:

Idea:

- Finding optimum mapping is equivalent to optimizing an **objective** function.
- We use different objective functions in different methods;
  - Minimize Information Loss: Mapping that represent the data as accurately as possible in the lower-dimensional space, e.g., Principal Components Analysis (PCA).
  - Maximize Discriminatory Information: Mapping that best discriminates the data in the lower-dimensional space, e.g., Linear Discriminant Analysis (LDA).
- Here we focus on PCA, that is, a linear mapping.
- Why Linear: Simpler to Compute and Analytically Tractable.

Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

- Given features in d-dimensional space

- Project into lower dimensional space using the following linear transformation

$$\mathbf{z} = \mathbf{W}^T \mathbf{x}$$

- For example (can you tell me size of matrix **W** for the following cases),
  - find best planar approximation to 4D data
  - find best planar approximation to 100D data

- We want to find this mapping while preserving as much information as possible, and ensuring

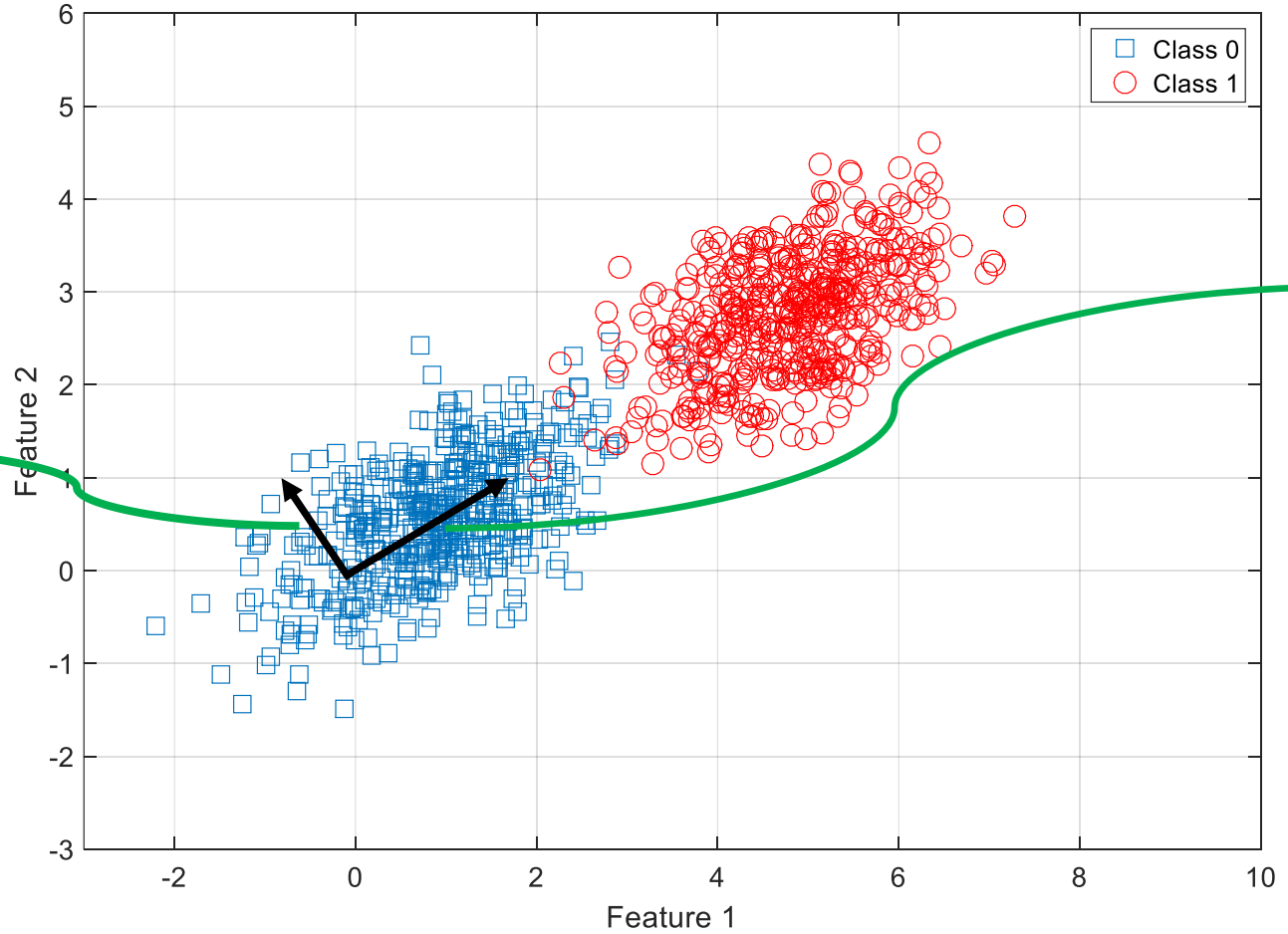
  - ***Objective 1***: the features after mapping are uncorrelated; cannot be reduced further

  - ***Objective 2***: the features after mapping have large variance

Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

Geometric Intuition:



Most contribution of each class lies in this direction

First Principal Component

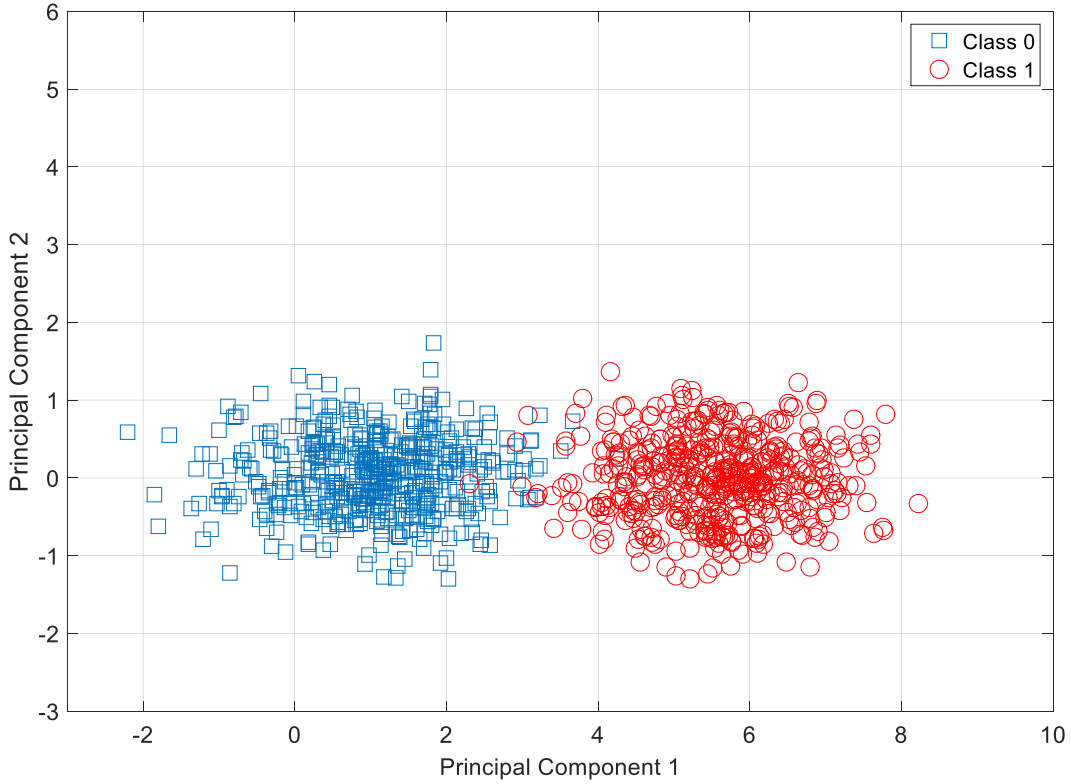
Second Principal Component

Toy Illustration in two dimensions

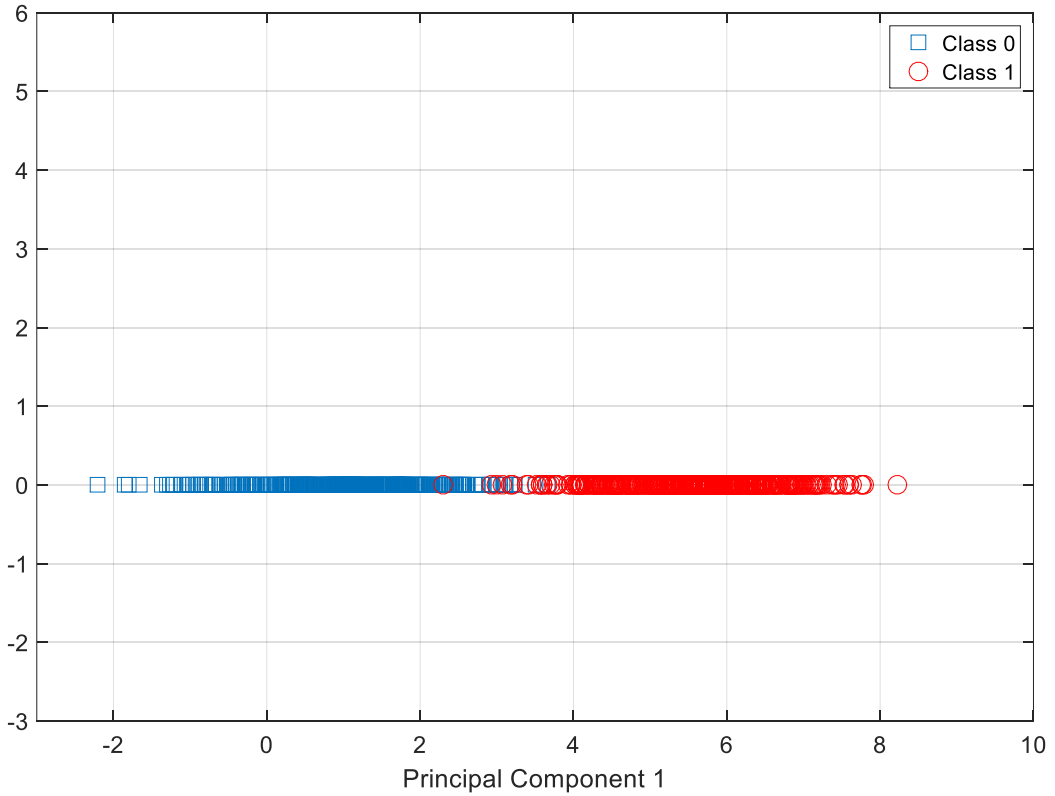
Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

Geometric Intuition:



Change of coordinates: Linear combinations of features



Ignoring the Second Component/Feature

Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

Mathematical Formulation:

We have n feature vectors of the form $\mathbf{x} \in \mathbf{R}^d$.

Note d represents the number of features.

In PCA, we want to represent $\mathbf{x}$ in a new space of lower dimensionality using only k basis vectors ($k < N$), that is,

$$\hat{\mathbf{x}} = \sum_{i=1}^{k} z_i \mathbf{v}_i$$

such that

$$\|\mathbf{x} - \hat{\mathbf{x}}\|_2$$

is minimized.

Here $\mathbf{v}_i \in \mathbf{R}^d$ for $i = 1, 2, \ldots, k$ represent the k number of orthogonal vectors that form the basis, referred to as principal components, of the subspace of dimensionality=k.

Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

Mathematical Formulation:

How do we find the basis vectors $\mathbf{v}_i \in \mathbf{R}^d$ for $i = 1, 2, \ldots, k$?

Steps to find Principal Components:

We have n feature vectors $\mathbf{x}_i \in \mathbf{R}^d, \quad i = 1, 2, \ldots, n.$

Step 1: Compute Sample Mean:

Sample mean (note summtion over the number of feature vectors n)

$$\overline{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$$

Step 2: Subtract Sample Mean:

Subtract sample mean from each feature vector $\mathbf{x}_i$ to obtain $\mathbf{s}_i$, that is,

$$\mathbf{s}_i = \mathbf{x}_i - \overline{\mathbf{x}}$$

Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

Mathematical Formulation:

Step 3: Calculate the Covariance Matrix:

Now we have n feature vectors $\mathbf{s}_i \in \mathbf{R}^d, \quad i = 1, 2, \ldots, n.$

What is special about these vectors?

Zero mean; taken along all feature vectors

Calculate the Covariance Matrix as follows

$$\Sigma = \frac{1}{n} \sum_{i=1}^{n} \mathbf{s}_i \mathbf{s}_i^T$$

This can also be expressed as

$$\Sigma = \frac{1}{n} \mathbf{S}\mathbf{S}^T$$

where

$$\mathbf{S} = [\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n]$$

How do you interpret the entries of the matrix? Spend some time and try to understand this!

For two vectors $\mathbf{f}, \mathbf{g} \in \mathbf{R}^n$, covariance is defined as

$$\sigma_{\mathbf{fg}} = \frac{1}{n} \sum_{i}^{n} (f_i - \text{avg}(\mathbf{f}))\,(g_i - \text{avg}(\mathbf{g}))$$

Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

Special about the Covariance Matrix:

The covarince matrix is symmetric, that is, $\Sigma^T = \Sigma$. (super easy to show)

The covarince matrix is positive semi-definite. (again, super easy)

Size of Σ is $d \times d$.

Step 4: Carry out Eigenvalue Decomposition of Covariance Matrix:

Carry out eigenvalue decomposition of the covarince matrix as

$$\Sigma = \mathbf{V}\mathbf{D}\mathbf{V}^T$$

Here the matrix $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_d]$ contains d orthogonal eigenvectors $\mathbf{v}_i \in \mathbf{R}^d$, referred to as principal components, that serve as the basis of $\mathbf{R}^d$.

Here the matrix $\mathbf{D}$ is a diagonal matrix with eigenvalues denoted by $\lambda_1, \lambda_2, \ldots, \lambda_d$.

Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

Step 5: Dimensionality Reduction

We wanted to find the basis vectors $\mathbf{v}_i \in \mathbf{R}^d$ for $i = 1, 2, \ldots, k$.

We have $\mathbf{v}_i \in \mathbf{R}^d$ for $i = 1, 2, \ldots, d$.

- *Q: How to select k out of d?*

- A: *Simple, select the ones corresponding to k largest eigenvalues.*

Construct the maapping matrix of size $d \times k$ as

$$\mathbf{W} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k]$$

to reduce the dimensionality of the feature space from $\mathbf{R}^d$ to $\mathbf{R}^k$ as

$$\mathbf{z} = \mathbf{W}^T \mathbf{x}$$

Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

Using $\mathbf{z}$, we can go back to $\mathbf{R}^d$ to obtain apprxoimation of $\mathbf{x}$ as

$$\hat{\mathbf{x}} = \sum_{i=1}^{k} z_i \mathbf{v}_i = \mathbf{W}\mathbf{z}$$

Connection with the Objectives:

- **Objective 1**: the features after mapping are uncorrelated; cannot be reduced further
  - Enabled by orthogonality of the principal components
- **Objective 2**: the features after mapping have large variance
  - We have used covariance matrix to define the mapping and used eigenvectors with largest eigenvalues, that is, those dimensions capturing the variations in the data.
  - **PCA maps the data along the directions where we have most of the variations in the data.**

Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

How do we choose k?

- It depends on the amount of information, that is variance, we want to preserve in the mapping process.

- We can define a variable T to quantify this preservation of information

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} > T$$

- T=1, when k=d; No reduction.
- T=0.8, interpreted as that 80% variation in the data has been preserved.

Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

Example: $d = 2,\ n = 10,\ k = 1$

Step 1: Compute sample mean:

$$\bar{\mathbf{x}} = [1.81, 1.91]$$

x_1	x_2	
2.5000	2.4000	$\mathbf{x}_1$
0.5000	0.7000	$\mathbf{x}_2$
2.2000	2.9000	
1.9000	2.2000	
3.1000	3.0000	
2.3000	2.7000	
2.0000	1.6000	
1.0000	1.1000	
1.5000	1.6000	
1.1000	0.9000	

Step 2: Subtract Sample Mean:

$$\mathbf{s}_i = \mathbf{x}_i - \overline{\mathbf{x}}$$

s_1	s_2	
0.6900	0.4900	$\mathbf{s}_1$
-1.3100	-1.2100	$\mathbf{s}_2$
0.3900	0.9900	
0.0900	0.2900	
1.2900	1.0900	
0.4900	0.7900	
0.1900	-0.3100	
-0.8100	-0.8100	
-0.3100	-0.3100	
-0.7100	-1.0100	

Step 3: Calculate the Covariance Matrix:

$$\mathbf{S} = [\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n]$$

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n} \mathbf{s}_i \mathbf{s}_i^T = \frac{1}{n}\mathbf{S}\mathbf{S}^T$$

$$\Sigma = \begin{bmatrix} 0.5549 & 0.5539 \\ 0.5539 & 0.6449 \end{bmatrix}$$

We have divided by n. Some authors divide by n-1. It won't change the principal components

Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

Example:

Step 4: Carry out Eigenvalue Decomposition of Covariance Matrix:

$$\Sigma = \mathbf{V}\mathbf{D}\mathbf{V}^T \qquad \mathbf{V} = \begin{bmatrix} -0.7352 & 0.6779 \\ 0.6779 & 0.7352 \end{bmatrix} \qquad \mathbf{D} = \begin{bmatrix} 0.0442 & 0 \\ 0 & 1.1556 \end{bmatrix}$$

Step 5: Dimensionality Reduction

Use $\mathbf{W} = [\mathbf{v}_2]$ (associated with the largest eigenvalue) to reduce the dimensionality of the feature space from $\mathbf{R}^2$ to $\mathbf{R}$ as

$$\mathbf{z} = \mathbf{W}^T\mathbf{x}$$

z
3.4591
0.8536
3.6233
2.9054
4.3069
3.5441
2.5320
1.4866
2.1931
1.4073

Dimensionality Reduction

Feature Extraction - Principal Component Analysis:

Practical Considerations and Limitations:

- Data should be normalized before using PCA for dimensionality reduction.

- Usually, we normalize every feature by subtracting mean of that feature followed by dividing with standard deviation of the feature.

- The covariance matrix of the reduced feature is projection along orthogonal components (directions) and therefore features are uncorrelated to each other. In other words, PCA decorrelates the features.

- **Limitation:**
  - PCA does not consider the separation of data with respect to class label and therefore we do not have a guarantee the mapping of the data along dimensions of maximum variance results in the new features good enough for class discrimination.

  Solution: Linear Discriminant Analysis (LDA) – Find mapping directions along which the classes are best separated.