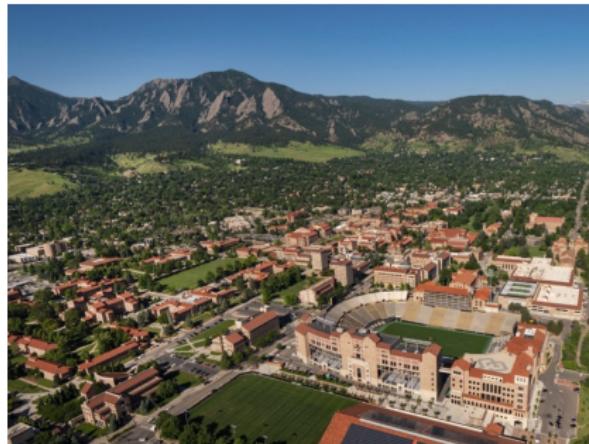




University of Colorado **Boulder**



CSCI 5622: Machine Learning

Lecture 2

Outline

- K-Nearest Neighbor (K-NN)
 - Basics
 - Example & Representation
- Practical use of K-NN
 - How to choose the right K and distance metric?
 - How to pre-process the data?
 - What to do in the case of a tie?
- Variations
 - Weighted Distance K-NN
 - Condensed Nearest Neighbor
 - KD-Trees

Note: Part of these slides is from CSCI567 Machine Learning (USC, Drs. Sha & Liu)

Outline

- K-Nearest Neighbor (K-NN)
 - Basics
 - Example & Representation
- Practical use of K-NN
 - How to choose the right K and distance metric?
 - How to pre-process the data?
 - What to do in the case of a tie?
- Variations
 - Weighted Distance K-NN
 - Condensed Nearest Neighbor
 - KD-Trees

Note: Part of these slides is from CSCI567 Machine Learning (USC, Drs. Sha & Liu)

K-Nearest Neighbor: Example

Recognizing types of Iris flowers (by R. Fisher)



setosa



versicolor

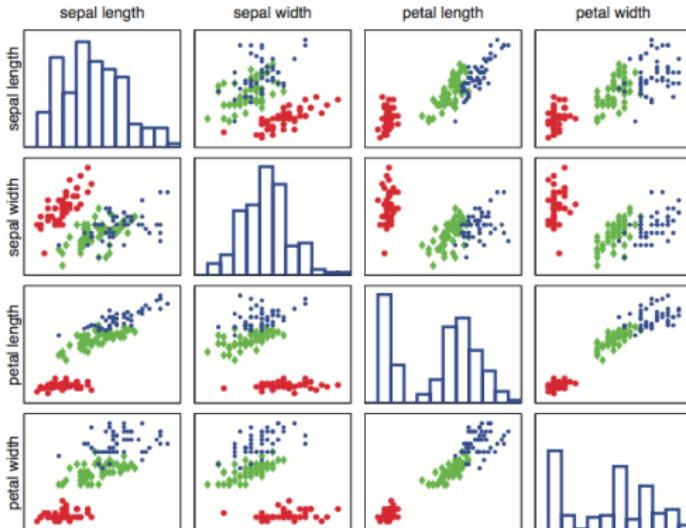


virginica

Features: the widths and lengths of sepal and petal

K-Nearest Neighbor: Example

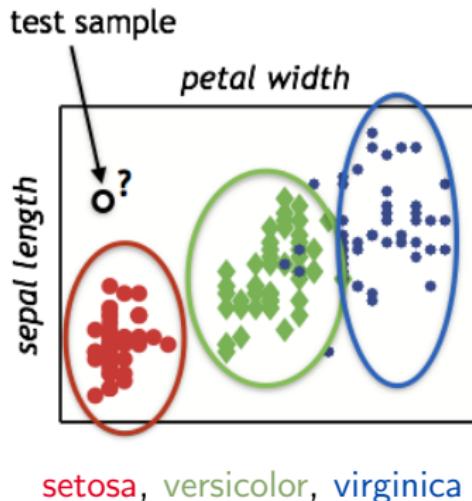
Visualizing features to get better intuition about our data



Each colored datapoint is one sample
setosa, versicolor, virginica

K-Nearest Neighbor: Example

Using two features: sepal length & petal width



Test sample is closer to red cluster → label it as **setosa**

K-Nearest Neighbor: Example

Recognizing types of Iris flowers (by R. Fisher)

Often data is organized in a table

Each row is one sample with 4 features and 1 label

```
5.1,3.5,1.4,0.2,Iris-setosa  
4.9,3.0,1.4,0.2,Iris-setosa  
4.7,3.2,1.3,0.2,Iris-setosa  
4.6,3.1,1.5,0.2,Iris-setosa  
5.0,3.6,1.4,0.2,Iris-setosa  
5.4,3.9,1.7,0.4,Iris-setosa  
4.6,3.4,1.4,0.3,Iris-setosa  
5.0,3.4,1.5,0.2,Iris-setosa  
4.4,2.9,1.4,0.2,Iris-setosa  
4.9,3.1,1.5,0.1,Iris-setosa  
5.4,3.7,1.5,0.2,Iris-setosa  
4.8,3.4,1.6,0.2,Iris-setosa  
4.8,3.0,1.4,0.1,Iris-setosa  
4.3,3.0,1.1,0.1,Iris-setosa
```

Attribute Information:
1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

[Source: <https://archive.ics.uci.edu/ml/datasets/iris>]

K-Nearest Neighbor: Hyperparameter tuning using a validation set

Training Data (or training set)

- N samples/datapoints/instances: $\mathcal{D}^{train} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$
- Used for learning representation $f : \mathbf{x} \rightarrow y$

Testing Data

- M samples/datapoints/instances: $\mathcal{D}^{test} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_M, y_M)\}$
- Used to assess how well $f(\cdot)$ will do in predicting an unseen sample

Validation Data (or validation/development set)

- L samples/datapoints/instances: $\mathcal{D}^{dev} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_L, y_L)\}$
- Used to optimize hyperparameter(s)

Train, test, validation data should **not** overlap

$$\mathcal{D}^{train} \cap \mathcal{D}^{test} \cap \mathcal{D}^{dev} = \emptyset$$

K-Nearest Neighbor: Representation

Classify data into one out of multiple classes

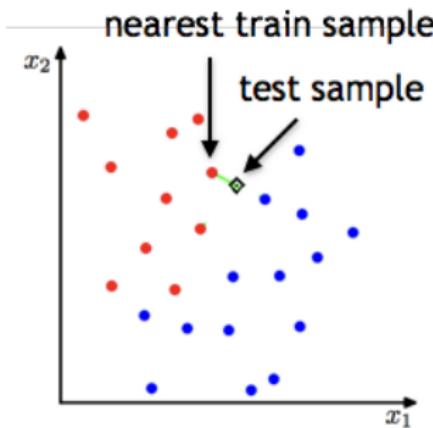
- **Input:** $\mathbf{x} \in \mathbb{R}^D$ (features, attributes, etc.)
- **Output:** $y \in \{1, 2, \dots, C\}$ (labels)
- **Model:** $f : \mathbf{x} \rightarrow y$

Special case: binary classification ($C=2$)

- **Output:** $y \in \{1, 2\}$ or $\{0, 1\}$ or $\{-1, 1\}$, etc.

K-Nearest Neighbor: Representation

Nearest Neighbor (or 1-Nearest Neighbor, 1-NN): Example



The nearest point to test sample x is a red training instance, therefore x will be labeled as **red**.

K-Nearest Neighbor: Representation

1-Nearest Neighbor (1-NN)

- Assigns test sample $\mathbf{x} = [x_1, x_2, \dots, x_D]^T$ to the closest training sample
- Training samples are in the set $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, where $\mathbf{x}_n = [x_{n1}, x_{n2}, \dots, x_{nD}]^T$
- N: # training samples, D: # features
- Model

$$y = f(\mathbf{x}) = y_{nn(\mathbf{x})}$$

$$nn(\mathbf{x}) = \arg \min_{n=1, \dots, N} \|\mathbf{x} - \mathbf{x}_n\|_2^2 = \arg \min_{n=1, \dots, N} \sum_{d=1}^D (x_d - x_{nd})^2$$

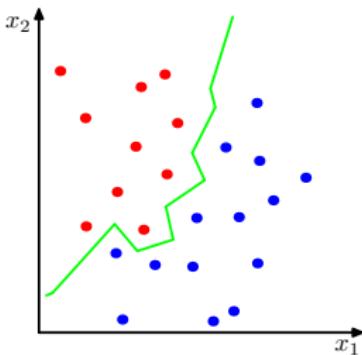
$\|\cdot\|_2$: l2 – norm

$\arg \min$: the index of the minimum element in a set or a vector

K-Nearest Neighbor: Representation

Nearest Neighbor (or 1-Nearest Neighbor, 1-NN): Example

Decision boundary: For every point in the space, we can determine its label using the nearest neighbor rule. This gives us a decision boundary that partitions the space into different regions.

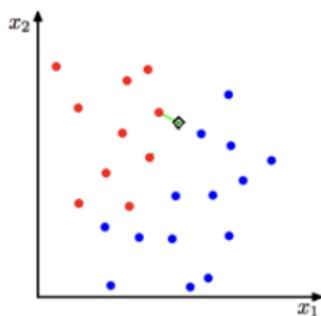


The above decision boundary is very sensitive to noise
What would be the solution for this?

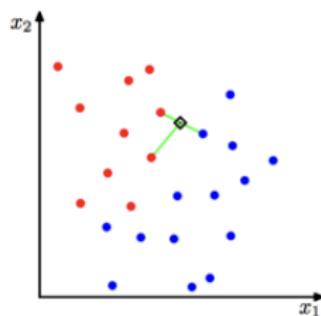
K-Nearest Neighbor: Representation

K-NN Example

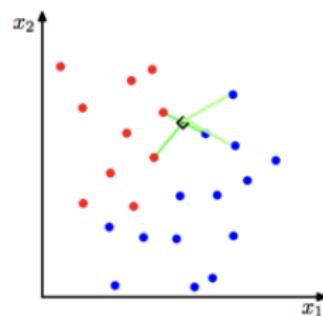
K=1, label=red



K=3 label=red



K=5, label=blue



K-Nearest Neighbor: Representation

Increase number of nearest neighbors to use

- 1-nearest neighbor: $nn_1(\mathbf{x}) = \arg \min_{n \in \{1, \dots, N\}} \|\mathbf{x} - \mathbf{x}_n\|_2^2$
- 2-nearest neighbor: $nn_2(\mathbf{x}) = \arg \min_{n \in \{1, \dots, N\} \setminus nn_1(\mathbf{x})} \|\mathbf{x} - \mathbf{x}_n\|_2^2$
- 3-nearest neighbor:
$$nn_3(\mathbf{x}) = \arg \min_{n \in \{1, \dots, N\} \setminus \{nn_1(\mathbf{x}), nn_2(\mathbf{x})\}} \|\mathbf{x} - \mathbf{x}_n\|_2^2$$

The set of K-nearest neighbors is

$$knn(\mathbf{x}) = \{nn_1(\mathbf{x}), \dots, nn_K(\mathbf{x})\}$$

Neighbors nn_1, \dots, nn_K in order of increasing distance from sample \mathbf{x}

K-Nearest Neighbor: Representation

K-NN Model

- Each neighbor in $knn(\mathbf{x}) = \{nn_1(\mathbf{x}), \dots, nn_K(\mathbf{x})\}$ votes one class
- Count the number of neighbors that have voted each class

$$v_c = \sum_{k \in knn(\mathbf{x})} \mathbb{I}(y_k = c), \quad c = 1, \dots, C$$

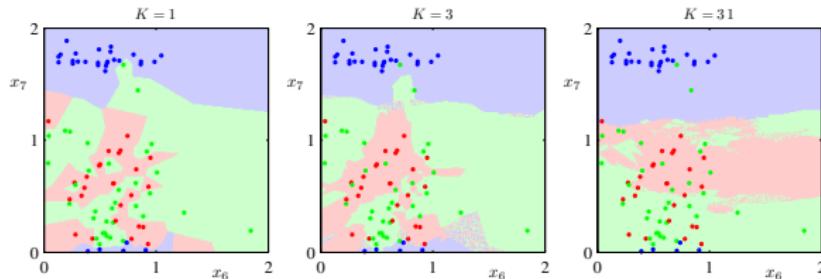
\mathbb{I} : indicator function ($\mathbb{I}\{A\} = 1$, if A true; $\mathbb{I}\{A\} = 0$, if A false)

- Assign test sample \mathbf{x} to the majority class membership of the K neighbors

$$y = f(\mathbf{x}) = \arg \max_{c=1, \dots, C} v_c$$

K-Nearest Neighbor: Representation

K-NN Decision Boundary



Number of neighbors K controls the degree of smoothing

$K \downarrow$: many small regions of each class

$K \uparrow$: fewer larger regions of each class

K-Nearest Neighbor: Computational Cost

Question: What is the computational cost of K-NN for labelling one test sample $\mathbf{x} \in \mathbb{R}^D$ given that we have N training data?

- A) $O(ND)$
- B) $O(KD)$
- C) $O(ND + NK)$
- D) $O(NKD)$



K-Nearest Neighbor: Computational Cost

Question: What is the computational cost of K-NN for labelling one test sample $\mathbf{x} \in \mathbb{R}^D$ given that we have N training data?

- A) $O(ND)$
- B) $O(KD)$
- C) $O(ND + NK)$
- D) $O(NKD)$

The correct answer is C.

The cost of measuring the distance between the test sample and each sample in the training data is $O(D)$

The cost of computing distances for all N train samples is $O(ND)$

The cost of finding the K closest samples is $O(NK)$ (can be optimized)

So the total cost is $O(ND + NK)$

[Nice video source: https://www.youtube.com/watch?v=UPAnUE_g5SQ]

Parametric v.s. non-parametric models

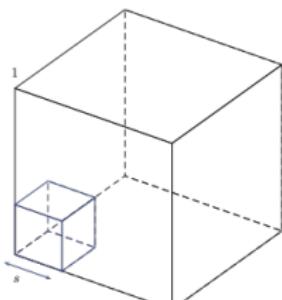
- Many possible ways to categorize learning models
- Non-parametric models (or instance/memory-based)
 - more flexible
 - computationally intractable for large datasets
 - e.g. K-NN
- Parametric models
 - faster to use
 - make strong assumptions about data
 - e.g. linear regression

Curse of dimensionality

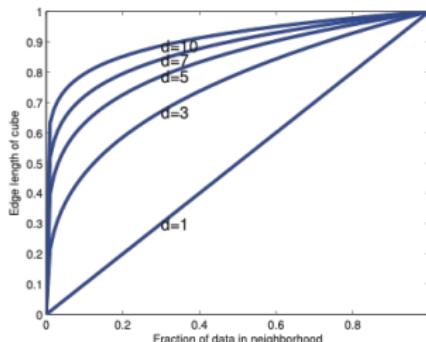
- In a high-dimensional space:
 - human intuition fails in higher dimensions
 - harder to generalize
 - harder to systematically search
 - harder to accurately approximate a target function
- K-NN is prone to high dimensions

Curse of dimensionality

- Assume that the entire space is a d-dimensional unit square
- Assuming that data is uniformly distributed within the 1-unit cube
- Assume a smaller cube within the space with edge length $s < 1$
- What should be the edge length of the cube s in order to include a fraction f of the data for each feature size $d = 1, \dots, 10$
- For $d = 10$, we need almost 90% of the original cube to cover 20% of the data



(a)



(b)

K-Nearest Neighbor: Theoretical guarantees

Modeling expected mistakes

- Assume data (\mathbf{x}, y) is drawn from the joint distribution $p(\mathbf{x}, y)$
 - In practice, $p(\mathbf{x}, y)$ is **unknown**
- Assume we use classifier $f(\cdot)$ (2 classes for simplicity)
- Classification mistake on \mathbf{x} with ground truth y , or "**0/1 loss function**"
$$L(f(\mathbf{x}), y) = \begin{cases} 0, & f(\mathbf{x}) = y \\ 1, & f(\mathbf{x}) \neq y \end{cases}$$
- Expected classification mistake on \mathbf{x} , or "**expected conditional risk**"
$$\begin{aligned} R(f, \mathbf{x}) &= \mathbb{E}_{y \sim p(y|\mathbf{x})} L(f(\mathbf{x}), y) \\ &= P(y = 1|\mathbf{x}) \mathbb{I}(f(\mathbf{x}) = 0) + P(y = 0|\mathbf{x}) \mathbb{I}(f(\mathbf{x}) = 1) \end{aligned}$$

K-Nearest Neighbor: Theoretical guarantees

Bayes Optimal Classifier (2 class)

- For any labeling function $f(\cdot)$, we have $R(f^*, \mathbf{x}) \leq R(f, \mathbf{x})$
- Namely, $f^*(\cdot)$ is optimal, i.e. the Bayes optimal classifier always yields the lowest expected conditional risk
- In practice
 - The Bayes optimal classifier is generally not computable as it assumes the knowledge of $p(\mathbf{x}, y)$ and $p(y|\mathbf{x})$
 - However, it is useful as a conceptual tool to formalize how well a classifier can do without knowing the joint distribution

K-Nearest Neighbor: Theoretical guarantees

How well does K-NN do wrt Bayes Optimal Classifier? (2-class)

Theorem

For the K-NN labeling function f^{K-NN} for binary classification, we have:

$$R(f^*) \leq R(f^{K-NN}) \leq 2R(f^*)(1 - R(f^*)) \leq 2R(f^*)$$

Given infinite data, K-NN is guaranteed to approach the Bayes error rate under ideal conditions.

In short K-NN seems to do a reasonable thing.

Outline

- K-Nearest Neighbor (K-NN)
 - Basics
 - Example & Representation
- Practical use of K-NN
 - How to choose the right K and distance metric?
 - How to pre-process the data?
 - What to do in the case of a tie?
- Variations
 - Weighted Distance K-NN
 - Condensed Nearest Neighbor
 - KD-Trees

Note: Part of these slides is from CSCI567 Machine Learning (USC, Drs. Sha & Liu)

K-Nearest Neighbor: Practical Use

Two practical issues about K-NN

- Choosing number of neighbors K
- Choosing distance metric between test sample \mathbf{x} and training data $\mathbf{x}_n, n = 1, \dots, N$

$$\|\mathbf{x} - \mathbf{x}_n\|_p = \left(\sum_{d=1}^D |x_d - x_{nd}|^p \right)^{1/p}, \quad p \geq 1$$

The above are called **hyperparameters**. They are not estimated through the learning process, but are externally set before the beginning of the learning process.

K-Nearest Neighbor: Model parameters and Hyperparameters

- **Hyperparameters:** Parameters set before the beginning of the learning process
 - For K-NN: number of neighbors K , distance metric
- **Hyperparameter tuning:** The process of choosing a set of optimal hyperparameters for the learning process
- **Model parameters:** The parameters learned during the learning process
 - For K-NN: As we said, KNN is non-parametric. Next lecture, parameters will be the weights of the linear perceptron or the linear regression.

K-Nearest Neighbor: Hyperparameter tuning using a validation set

Training Data (or training set)

- N samples/datapoints/instances: $\mathcal{D}^{train} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$
- Used for learning representation $f : \mathbf{x} \rightarrow y$

Testing Data

- M samples/datapoints/instances: $\mathcal{D}^{test} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_M, y_M)\}$
- Used to assess how well $f(\cdot)$ will do in predicting an unseen sample

Validation Data (or validation/development set)

- L samples/datapoints/instances: $\mathcal{D}^{dev} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_L, y_L)\}$
- Used to optimize hyperparameter(s)

Train, test, validation data should **not** overlap

$$\mathcal{D}^{train} \cap \mathcal{D}^{test} \cap \mathcal{D}^{dev} = \emptyset$$

Recipe for hyperparameter tuning with dev/validation set

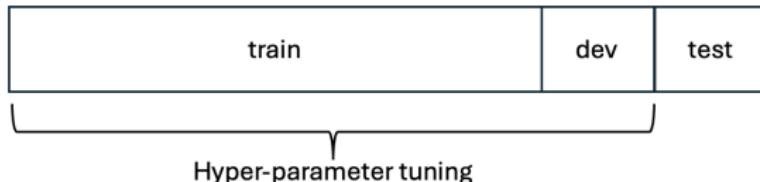
- For each hyperparameter value (or combination of values)
 - Train model using \mathcal{D}^{train}
 - Evaluate model performance on \mathcal{D}^{dev}

For K-NN: For $K = 1, 3, \dots$

 - For all $x \in \mathcal{D}^{dev}$, assign each sample to the majority class determined by its K neighbors from the train data \mathcal{D}^{train} (see previous slide, K-Nearest Neighbor: Representation)
 - Evaluate accuracy/error rate, etc. using all $x \in \mathcal{D}^{dev}$
- Choose the model with the best performance on \mathcal{D}^{dev}

For K-NN: Choose K with the best accuracy/error rate, etc.
- Evaluate the model on the test set \mathcal{D}^{test}

For K-NN: Similarly as we did for evaluating on \mathcal{D}^{dev}



Recipe for hyperparameter tuning with dev/validation set

Question: Assume that you have a set of training, validation, and test data, $\mathbf{X}^{\text{train}} \in \mathbb{R}^{D \times N_1}$, $\mathbf{X}^{\text{val}} \in \mathbb{R}^{D \times N_2}$, and $\mathbf{X}^{\text{test}} \in \mathbb{R}^{D \times N_3}$, respectively, with corresponding label vectors $\mathbf{y}^{\text{train}} \in \mathbb{R}^{N_1}$, $\mathbf{y}^{\text{dev}} \in \mathbb{R}^{N_2}$, and $\mathbf{y}^{\text{test}} \in \mathbb{R}^{N_3}$. You would like to perform K-NN classification and use the validation set to determine the best value of $K = \{1, 3, 5, 7, 9\}$. Please provide a basic pseudocode to do this. In the pseudocode, please also include the last evaluation step on the test data after having identified the best K based on the train data.

You can assume a function $\text{pred}=\text{K-NN}(X, y, Z, K)$, which provides a decision for test samples Z using training data X , training labels y and number of neighbors K , and a function $\text{acc}=\text{ComputeAcc}(\text{pred}, \text{lab})$, which computes the classification accuracy between predicted class pred and actual labels lab .

$$K = [1, 3, 5, 7, 9]$$

$\text{Acc} = []$ (vector to store the accuracies on the validation set from all values of K)
for $\alpha \in K$:

$$\text{pred}=\text{K-NN}(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}}, \mathbf{X}^{\text{val}}, \alpha)$$

$$\text{Acc} = [\text{Acc}, \text{ComputeAcc}(\text{pred}, \mathbf{y}^{\text{val}})] \text{ (storing accuracy for } \alpha\text{)}$$

$$K^{\text{best}} = K(\arg \max(\text{Acc})) \text{ (obtaining the best K)}$$

$$\text{pred}=\text{K-NN}(\mathbf{X}^{\text{train}} \cup \mathbf{X}^{\text{val}}, \mathbf{y}^{\text{train}} \cup \mathbf{y}^{\text{val}}, \mathbf{X}^{\text{test}}, K^{\text{best}})$$

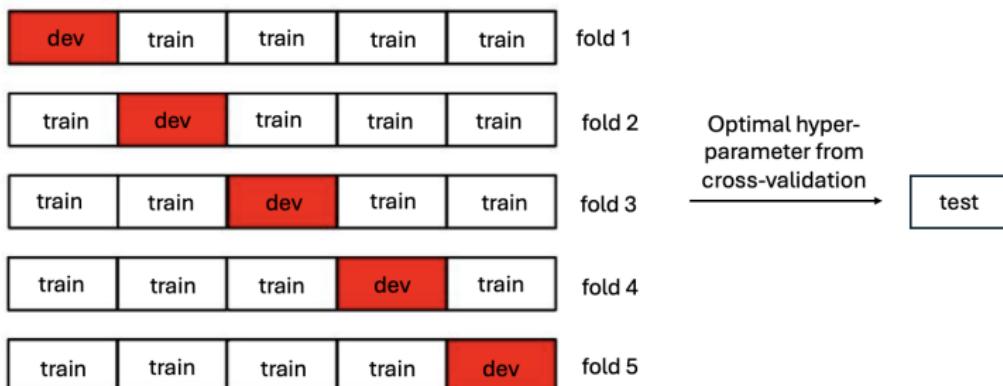
$$\text{acc}=\text{ComputeAcc}(\text{pred}, \mathbf{y}^{\text{test}})$$

Recipe for hyperparameter tuning with cross-validation

What if we don't have a lot of data in the validation set?

We perform **cross-validation**

- Split train data into S equal parts
- Use each part as a validation set and all others as train set
- Choose the hyperparameter value (or combination of values) that results in best **average** performance across all folds.



Recipe for cross-validation

- Split train data into S equal parts, each noted as \mathcal{D}_s^{train} , $s = 1, \dots, S$
- For each hyperparameter value (e.g. $K = 1, 3, \dots$)
 - For each $s = 1, \dots, S$
 - Train model using $\mathcal{D}^{train} \setminus \mathcal{D}_S^{train}$
 - Evaluate model performance (noted as E_s) on \mathcal{D}_s^{train}
 - Compute average performance for current hyperparameter
$$E = \frac{1}{s} \sum_{s=1}^S E_s$$
- Choose the hyperparameter corresponding to best average performance E
- Use the best hyperparameter to train on a model using all \mathcal{D}^{train}
- Evaluate the last model trained on all \mathcal{D}^{train} using the best hyperparameter on \mathcal{D}^{test}

Recipe for cross-validation

Question: Assume that you have a set of training and test data, $\mathbf{X}^{\text{train}} \in \mathbb{R}^{D \times N_1}$ and $\mathbf{X}^{\text{test}} \in \mathbb{R}^{D \times N_2}$, respectively, with corresponding label vectors $\mathbf{y}^{\text{train}} \in \mathbb{R}^{N_1}$ and $\mathbf{y}^{\text{test}} \in \mathbb{R}^{N_2}$. You would like to perform K-NN classification and use cross-validation on the training set to determine the best value of $K = \{1, 3, 5, 7, 9\}$. Please provide a basic pseudocode to do this. In the pseudocode, please also include the last evaluation step after having identified the best K based on the train data.

You can assume a function $\text{pred}=\text{K-NN}(X, y, Z, K)$, which provides a decision for test samples Z using training data X , training labels y and number of neighbors K , and a function $\text{acc}=\text{ComputeAcc}(\text{pred}, \text{lab})$, which computes the classification accuracy between predicted class pred and actual labels lab .

Split $\{\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}}\}$ into S equal parts $\{\mathbf{X}_S^{\text{train}}, \mathbf{y}_S^{\text{train}}\}_{s=1}^S$

$K = [1, 3, 5, 7, 9]$

$CVAcc = []$ (vector to store the accuracies from the cross-validation)

for $\alpha \in K$:

$E = []$ (vector to store accuracies within each fold)

for $s \in 1, \dots, S$:

$\text{pred}=\text{K-NN}(\mathbf{X}^{\text{train}} \setminus \mathbf{X}_S^{\text{train}}, \mathbf{y}^{\text{train}} \setminus \mathbf{y}_S^{\text{train}}, \mathbf{X}_S^{\text{train}}, \alpha)$

$E(s)=\text{ComputeAcc}(\text{pred}, \mathbf{y}_S^{\text{train}})$

$CVAcc = [CVAcc, \frac{1}{S} (E(1) + \dots + E(S))]$ (storing average accuracy from all folds for α)

$K^{\text{best}} = K(\arg \max(CVAcc))$ (obtaining the best K)

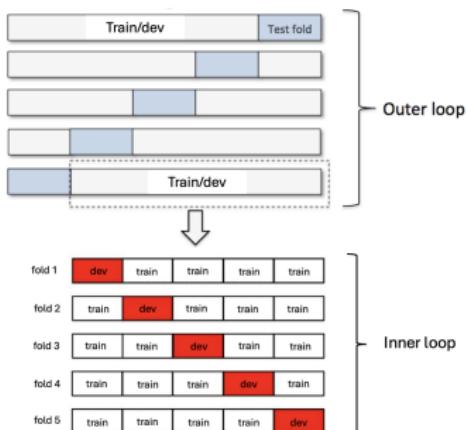
$\text{pred}=\text{K-NN}(\mathbf{X}^{\text{train}}, \mathbf{y}^{\text{train}}, \mathbf{X}^{\text{test}}, K^{\text{best}})$

$\text{acc}=\text{ComputeAcc}(\text{pred}, \mathbf{y}^{\text{test}})$

Evaluation with a cross-validation set

Nested cross-validation (Small datasets)

- The dataset is partitioned into k (e.g., $k = 5, 10$) equal size subsets, called **outer folds**.
- In each outer loop, perform cross-validation, i.e., split into s subsets, etc.
- The above is repeated k times.



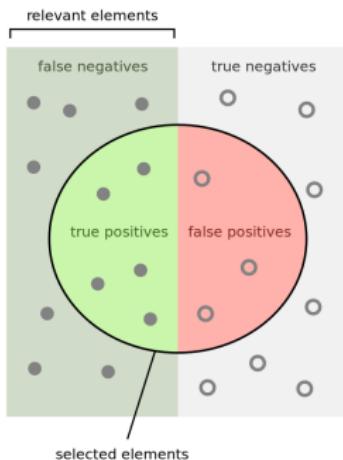
Evaluation metrics

Classification metrics

- Simple classification accuracy: $\frac{\text{#correctly classified samples}}{\text{#total samples}}$
- Balanced (macro) classification accuracy (for unbalanced classes):
$$\frac{1}{K} \cdot \frac{\text{#correctly classified samples for class 1}}{\text{#total samples for class 1}} + \dots + \frac{1}{K} \cdot \frac{\text{#correctly classified samples for class K}}{\text{#total samples for class K}}$$
- Precision (for binary classification, e.g., class 0/1):
$$\frac{\text{#correctly classified samples to class 1}}{\text{#all samples classified as class 1}}$$
- Recall (for binary classification, e.g., class 0/1):
$$\frac{\text{#correctly classified samples to class 1}}{\text{#all actual samples from class 1}}$$
- F1-score: $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Evaluation metrics

Classification metrics



$\text{Precision} = \frac{\text{How many selected items are relevant?}}{\text{How many selected items are selected?}}$	$\text{Recall} = \frac{\text{How many relevant items are selected?}}{\text{How many relevant items are there?}}$
--	--

- Precision (for binary classification, e.g., class 0/1):

$$\frac{\text{\#correctly classified samples to class 1}}{\text{\#all samples classified as class 1}}$$
- Recall (for binary classification, e.g., class 0/1):

$$\frac{\text{\#correctly classified samples to class 1}}{\text{\#all actual samples from class 1}}$$
- F1-score: $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

How to measure “neighbor nearness” with other distances?

Previously we have used the euclidean distance

(dominated by noise if #useful features is much smaller than #noisy features)

$$nn(\mathbf{x}) = \arg \min_{n=1, \dots, N} \|\mathbf{x} - \mathbf{x}_n\|_2^2 = \arg \min_{n=1, \dots, N} \sum_{d=1}^D (x_d - x_{nd})^2$$

We can also use alternative distances

- L_1 , Manhattan, or city block distance

$$nn(\mathbf{x}) = \arg \min_{n=1, \dots, N} \|\mathbf{x} - \mathbf{x}_n\|_1 = \arg \min_{n=1, \dots, N} \sum_{d=1}^D |x_d - x_{nd}|$$

- Hamming distance (discrete data, # non-matching attributes)

$$nn(\mathbf{x}) = \arg \min_{n=1, \dots, N} \sum_{d=1}^D 1(x_d \neq x_{nd})$$

where $1(a) = 1$ if a true, 0 otherwise

How to measure “neighbor nearness” with other distances?

Different distances might result in different sorting between samples

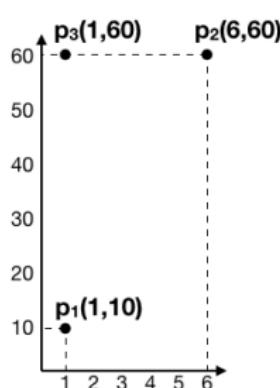
Example 1

- train data: $A(3, 1, 1)$ belonging to class 1, $B(2, 2, 2)$ belonging to class 2
- test data: $X(1, 1, 1)$
- $l_1(A, X) = |2| + |0| + |0| = 2$
 $l_1(B, X) = |1| + |1| + |1| = 3$
→ X in class 1
- $l_2(A, X) = \sqrt{|2|^2 + |0|^2 + |0|^2} = 2$
 $l_2(B, X) = \sqrt{|1|^2 + |1|^2 + |1|^2} = \sqrt{3}$
→ X in class 2

How to measure “neighbor nearness” with other distances?

Distances depend on the units of the features

Example: one feature in cm and one in mm



$$\text{dist}(p_1, p_2) = \sqrt{(60 - 10)^2 + (6 - 1)^2} = 2525$$

$$\text{dist}(p_1, p_3) = \sqrt{(60 - 10)^2 + (1 - 1)^2} = 2500$$

$$\text{rel_change}_{\text{x-axis}} = 100 \cdot \frac{6-1}{6}\% = 83\%$$

$$\text{rel_change}_{\text{dist}} = 100 \cdot \frac{2525-2500}{2500}\% = 1\%$$

Proximity can change due to change in feature scales
 Larger-scale features tend to inflate distance measure

How to measure “neighbor nearness” with other distances?

Normalize data so that they have comparable range

Value changes across any feature can be equally reflected to the distance metric, when features are normalized

$$x_{nd} := \frac{x_{nd} - \bar{x}_d}{\sqrt{s_d}}$$

$$\bar{x}_d = \frac{1}{N} \sum_n x_{nd}, \quad s_d = \frac{1}{N-1} \sum_n (x_{nd} - \bar{x}_d)^2$$

How to measure “neighbor nearness” with other distances?

Question

Assume the following train samples:

Class 1: $y_1 = (10, 4)$, $y_2 = (10, 13)$,
 $y_3 = (10, 16)$

Class 2: $z_1 = (6, 10)$, $z_2 = (14, 10)$

And the following distance metrics:

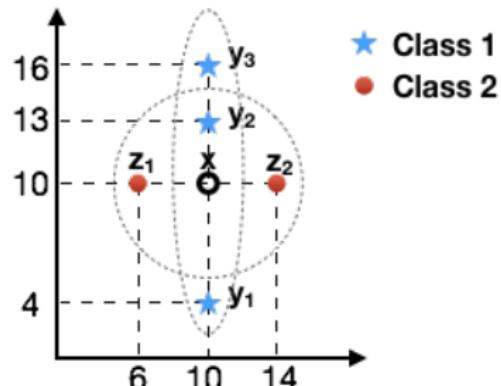
$$D_1 = (\mathbf{x} - \mathbf{x}_k)^T \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} (\mathbf{x} - \mathbf{x}_k)$$

$$D_2 = (\mathbf{x} - \mathbf{x}_k)^T \begin{bmatrix} 1 & 0 \\ 0 & 1/3 \end{bmatrix} (\mathbf{x} - \mathbf{x}_k)$$

In which class would K-NN (K=3) classify sample $\mathbf{x} = (10, 10)$ using distances D_1 and D_2 ?

Hint: Assume $\mathbf{x} = [x_1, x_2]^T$ and $\mathbf{x}_k = [x_{k1}, x_{k2}]^T$.

Expand the matrix-vector expression.



- A) Class 1 for both D_1 and D_2
- B) Class 2 for both D_1 and D_2
- C) Class 1 for D_1 , Class 2 for D_2
- D) Class 2 for D_1 , Class 1 for D_2

How to measure “neighbor nearness” with other distances?



How to measure “neighbor nearness” with other distances?

The correct answer is D

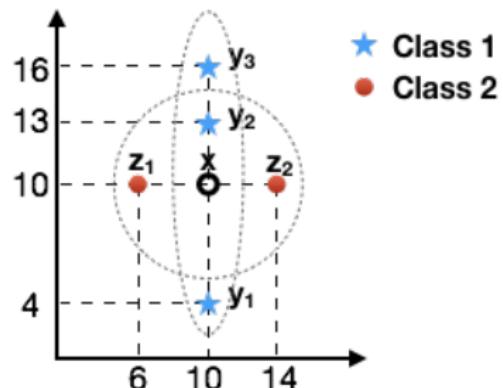
$$D_1 = (x_1 - x_{k1})^2 + (x_2 - x_{k2})^2$$

$$D_2 = (x_1 - x_{k1})^2 + \frac{1}{3}(x_2 - x_{k2})^2$$

Train Sample	D_1	D_2
y_1	36	12
y_2	9	3
y_3	36	12
z_1	16	16
z_2	16	16

The 3 nearest neighbors of x based on D_1 are y_2, z_1, z_2 , so K-NN decides Class 2

The 3 nearest neighbors of x based on D_2 are y_1, y_2, y_3 , so K-NN decides Class 1



- A) Class 1 for both D_1 and D_2
- B) Class 2 for both D_1 and D_2
- C) Class 1 for D_1 , Class 2 for D_2
- D) Class 2 for D_1 , Class 1 for D_2

What to do in the case of a tie?

- Randomly assign a class
- Assign the class with the highest frequency in the training set
- Classify the test sample according to the closest training sample (1-NN), or according to the K-1 or K+1 training samples
- Decide by assigning a weight to each vote according to its distance from the test sample (i.e. weighted K-NN)

K-Nearest Neighbor: Advantages & Disadvantages

Advantages	Disadvantages
<ul style="list-style-type: none">1. Intuitive2. Simple & easy to implement3. Guarantees that it “works”	<ul style="list-style-type: none">1. Computationally expensive for large datasets2. We need to keep the training data in memory3. Choosing the right K can be tricky4. Sensitive to noisy features5. May perform badly in high dimensions

Outline

- K-Nearest Neighbor (K-NN)
 - Basics
 - Example & Representation
- Practical use of K-NN
 - How to choose the right K and distance metric?
 - How to pre-process the data?
 - What to do in the case of a tie?
- Variations
 - Weighted Distance K-NN
 - Condensed Nearest Neighbor
 - KD-Trees

Note: Part of these slides is from CSCI567 Machine Learning (USC, Drs. Sha & Liu)

Distance Weighting

- K-NN assumes that similar instances mean similar things
- Hence samples closest to the query point might matter most
- Weight each neighbor by their closeness to the test sample

$$w_k = \frac{\exp(-\text{dist}(\mathbf{x}, nn_k(\mathbf{x})))}{\sum_{k \in knn(\mathbf{x})} \exp(-\text{dist}(\mathbf{x}, nn_k(\mathbf{x})))}$$

$$v_c = \sum_{k \in knn(\mathbf{x})} w_k \cdot \mathbb{I}(y_k = c), \quad c = 1, \dots, C$$

$$y = f(\mathbf{x}) = \arg \max_{c=1, \dots, C} v_c$$

Condensed Nearest Neighbor

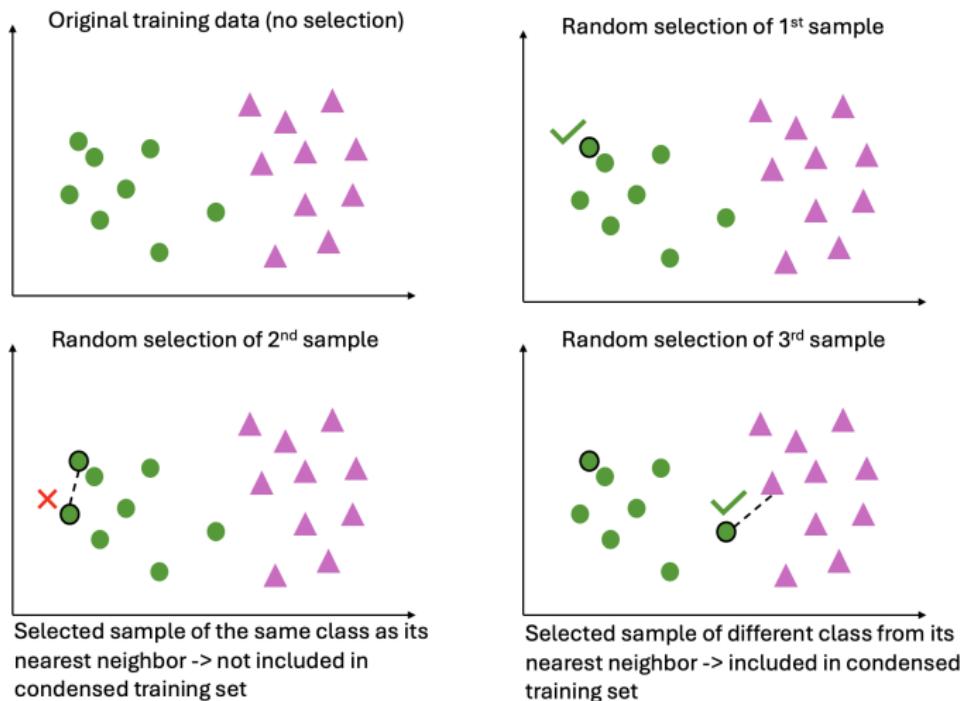
- Approximate way to reduce time/space complexity of K-NN
- Decrease number of stored training samples
- Select the smallest subset \mathcal{Z} of \mathcal{D}^{train} , such that when \mathcal{Z} is used, error does not increase
- Subset \mathcal{Z} is determined **once before running K-NN**, therefore the high computational cost is not during testing
- Local search that depends on the order of training samples
 - Non-unique solution

Condensed Nearest Neighbor

Algorithm Pseudocode for Condensed Nearest Neighbor

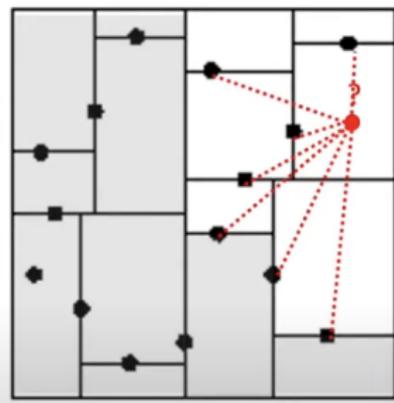
```
1: Randomly initialize  $\mathcal{Z}$  with a sample from  $\mathcal{D}^{train}$ 
2: repeat
3:   for  $x \in \mathcal{D}^{train}$  (in random order) do
4:     Find  $x' \in \mathcal{Z}$  closest to  $x$ 
5:     if  $\text{class}(x) \neq \text{class}(x')$  then
6:       add  $x$  to  $\mathcal{Z}$ 
7:     end if
8:   end for
9: until  $\mathcal{Z}$  does not change
```

Condensed Nearest Neighbor



KD-Trees

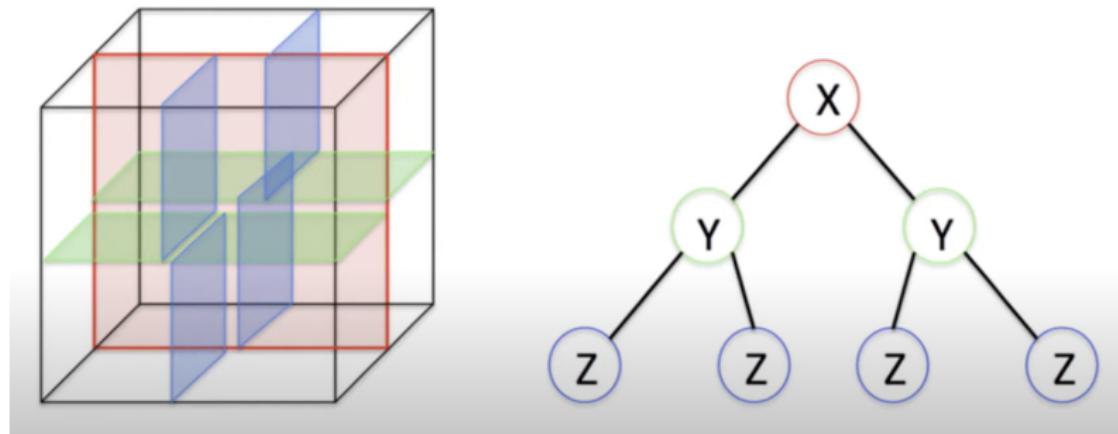
- We do not need to compute the distance of a test sample with all training samples
- We can use a data structure to store the training samples
- Then we can use this data structure for a more efficient search
- A KD-Tree is a data structure that partitions the space and organizes the D-dimensional training samples



Source: <https://www.youtube.com/watch?v=SnFqx3M28wQ>

KD-Trees: Construction

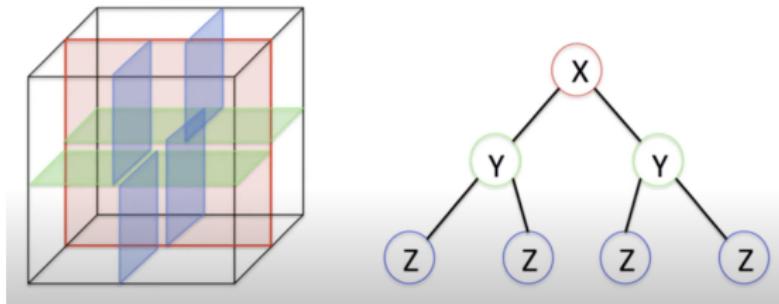
- Each cut in the D-dimensional space is represented as a tree node
- Cuts are created for each dimension separately and we iterate across all dimensions



Source: <https://www.youtube.com/watch?v=SnFqx3M28wQ>

KD-Trees: Construction

- ① Choose a feature dimension (or axis), e.g., x, y, z, \dots
- ② Sort data points according to axis and find the median
- ③ Set the median as the root
- ④ Assign the data points with smaller values than the median along that axis on the left side of the root
- ⑤ Assign the data points with larger values than the median along that axis on the right side of the root
- ⑥ Apply recursion using the data points on the left side and the right side of the root, and using the next feature dimension (or axis)



Source: <https://www.youtube.com/watch?v=SnFqx3M28wQ>

KD-Trees: Search

Object List = (2,3) (5,4) (9,6) (4,7) (8,1) (7,2)

1. axis = **x**
2. Sort Object List according to axis:

(**2,3**) (**4,7**) (**5,4**) (**7,2**) (**8,1**) (**9,6**)

3. Find the median (choose a default side for case where you have two medians): **Right**

(2,3) (4,7) (5,4) (**7,2**) (8,1) (9,6)

Source: <https://www.youtube.com/watch?v=SnFqx3M28wQ>

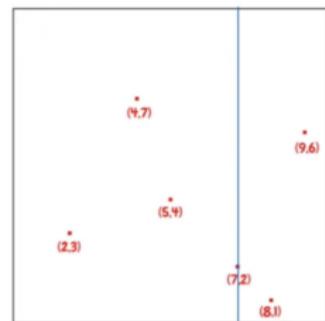
KD-Trees: Search

4. Set the median as root, and divide the other data points in two groups (before and after the median)

7, 2

(2,3) (4,7) (5,4)

(8,1) (9,6)



Source: <https://www.youtube.com/watch?v=SnFqx3M28wQ>

KD-Trees: Search

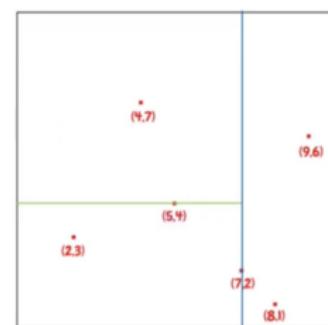
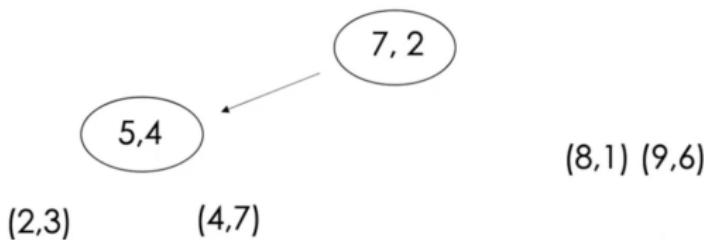
1. Sort according to axis y



Source: <https://www.youtube.com/watch?v=SnFqx3M28wQ>

KD-Trees: Search

2. Set the median as root:

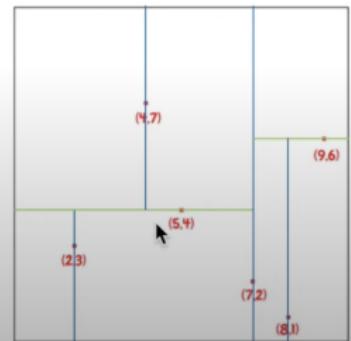
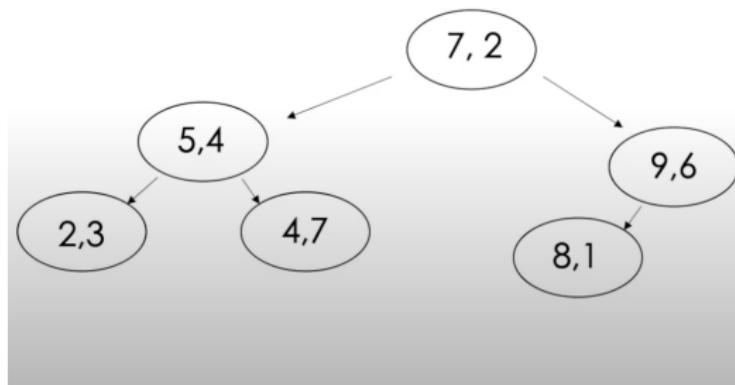


Source: <https://www.youtube.com/watch?v=SnFqx3M28wQ>

KD-Trees: Search

Example: Training samples

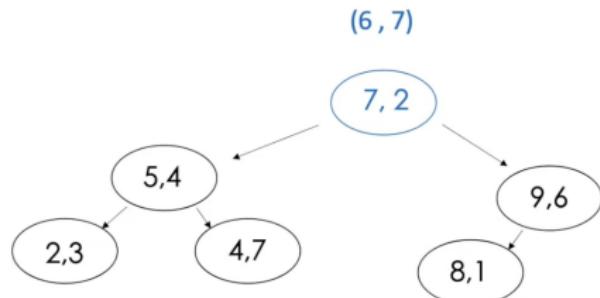
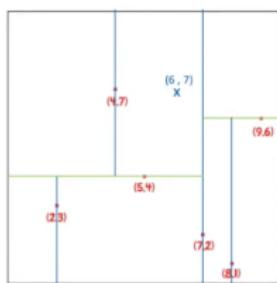
$\{(2, 3), (5, 4), (9, 6), (4, 7), (8, 1), (7, 2)\}$



Source: <https://www.youtube.com/watch?v=SnFqx3M28wQ>

KD-Trees: Search

Example: How to search for the nearest neighbor of a new data point $(6, 7)$?



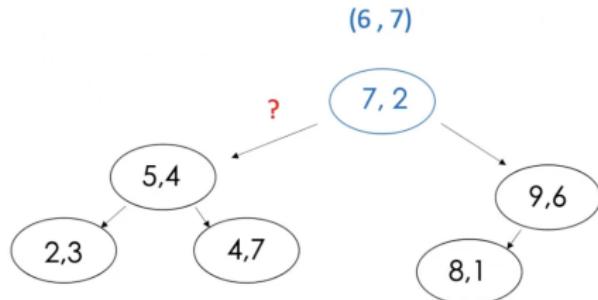
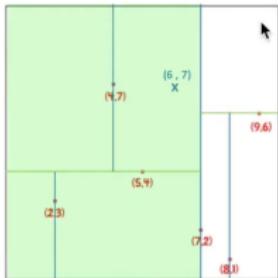
Best distance = 5.09 ↗
 Current best = (7, 2)

$$d([6, 7]; [7, 2]) = \sqrt{1 + 25} = \boxed{5.09}$$

Source: <https://www.youtube.com/watch?v=oQQrxJvnhw>

KD-Trees: Search

Example: How to search for the nearest neighbor of a new data point $(6, 7)$?

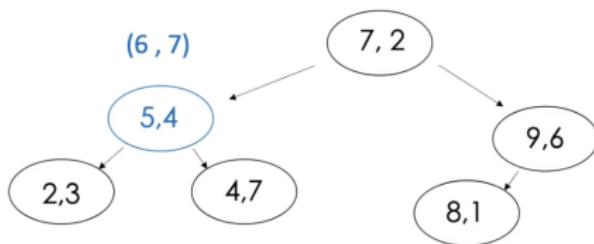
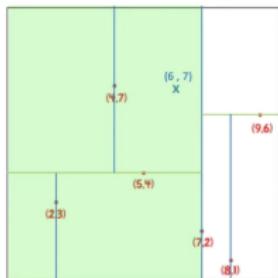


Best distance = 5.09
Current best = $(7, 2)$

Source: <https://www.youtube.com/watch?v=oQQrxJvnhw>

KD-Trees: Search

Example: How to search for the nearest neighbor of a new data point $(6, 7)$?



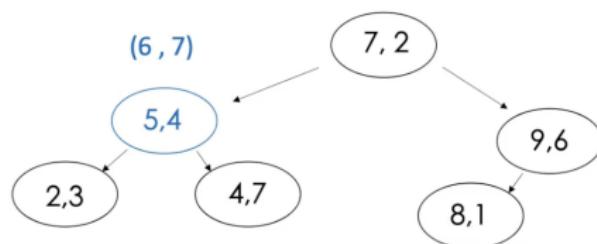
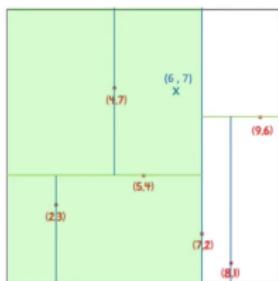
Best distance = 5.09
Current best = $(7, 2)$

$$d([6, 7]; [5, 4]) = \sqrt{1+9} = 3.16$$

Source: <https://www.youtube.com/watch?v=oQQrxijVnhw>

KD-Trees: Search

Example: How to search for the nearest neighbor of a new data point $(6, 7)$?



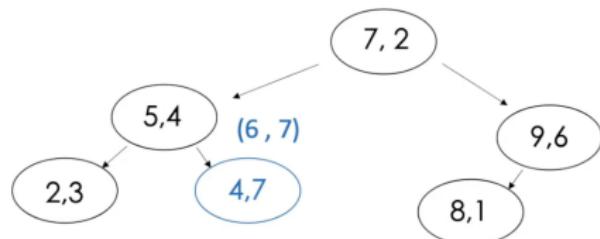
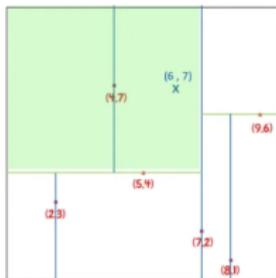
Best distance = 3.16
Current best = $(5, 4)$

$$d([6, 7]; [5, 4]) = \sqrt{1 + 9} = 3.16$$

Source: <https://www.youtube.com/watch?v=oQQrxijJvnhw>

KD-Trees: Search

Example: How to search for the nearest neighbor of a new data point $(6, 7)$?



Best distance = 3.16
Current best = (5, 4)

$$d([6, 7]; [4, 7]) = \sqrt{4 + 0} = \boxed{2}$$

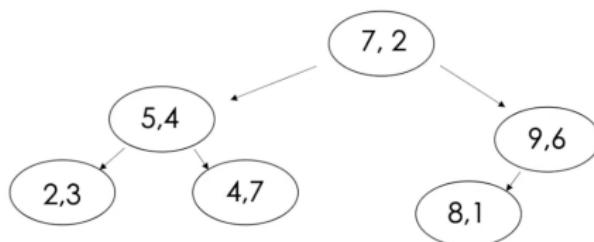
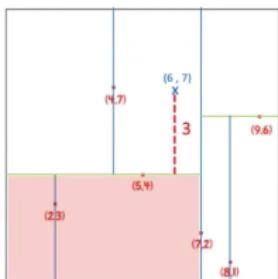


Source: <https://www.youtube.com/watch?v=oQQrxJvnhw>

KD-Trees: Search

Example: How to search for the nearest neighbor of a new data point $(6, 7)$?

Is there a way to find a neighbor in the shaded area that is closer to $(6, 7)$ than the current best? **No**

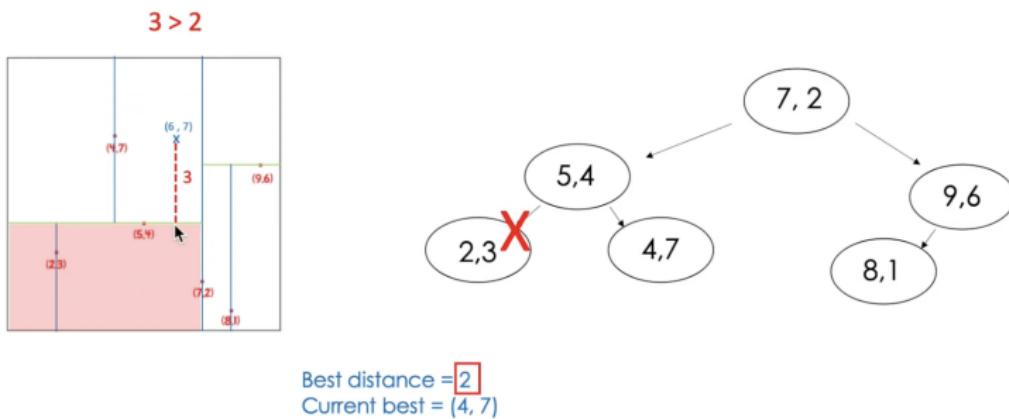


Best distance = 2
Current best = $(4, 7)$

Source: <https://www.youtube.com/watch?v=oQQrxJvnhw>

KD-Trees: Search

Example: How to search for the nearest neighbor of a new data point $(6, 7)$?

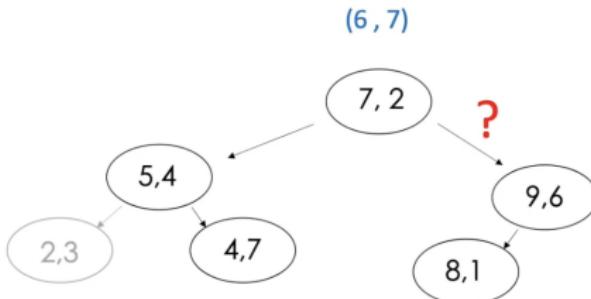
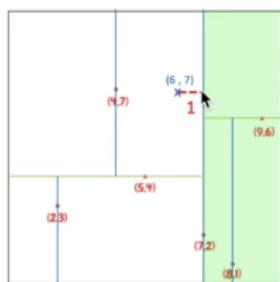


Source: <https://www.youtube.com/watch?v=oQQrxijJvnhw>

KD-Trees: Search

Example: How to search for the nearest neighbor of a new data point $(6, 7)$?

Is there a way to find a neighbor in the shaded area that is closer to $(6, 7)$ than the current best? **Yes**

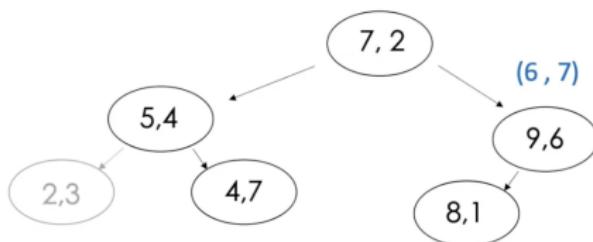
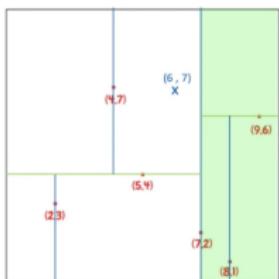


Best distance = 2
Current best = $(4, 7)$

Source: <https://www.youtube.com/watch?v=oQQrxijJvnhw>

KD-Trees: Search

Example: How to search for the nearest neighbor of a new data point (6, 7)?



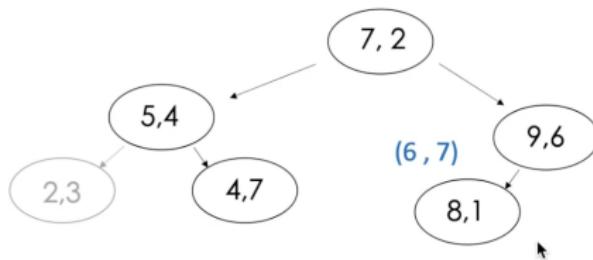
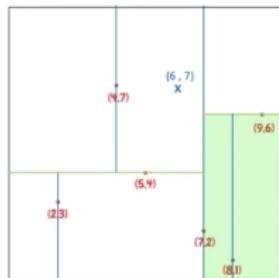
Best distance = 2
Current best = (4, 7)

$$d([6, 7]; [9, 6]) = \sqrt{9 + 1} = 3.16$$

Source: <https://www.youtube.com/watch?v=oQQrxijVnhw>

KD-Trees

Example: How to search for the nearest neighbor of a new data point (6, 7)?



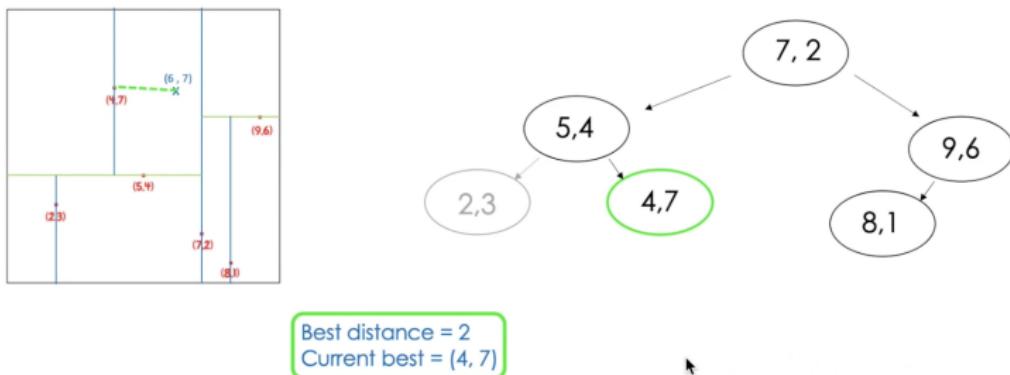
Best distance = 2
Current best = (4, 7)

$$d([6, 7]; [8, 1]) = \sqrt{4 + 36} = 6.32$$

Source: <https://www.youtube.com/watch?v=oQQrxJvnhw>

KD-Trees: Search

Example: How to search for the nearest neighbor of a new data point $(6, 7)$?



Source: <https://www.youtube.com/watch?v=oQQrxJvnhw>

KD-Trees

Computational cost

- KD-Trees will have $\log N$ depth
- Searching the nearest neighbor: $O(N \log N)$
- Building the tree: $O(DN \log N)$ (sorting the points in each dimension)

Summary

- Described a simple learning algorithm (K-NN)
 - non-parametric (instance-based) learning algorithm
 - “similar inputs will have similar outputs”
 - high computational cost for large data
 - guaranteed to approach Bayes error rate under ideal conditions
- Practical issues
 - number of neighbors, type of distance → (cross-)validation
- Computationally “cheaper” options (e.g. condensed K-NN)
- Reading materials
 - Alpaydin 8.1-8.5