AI and Machine Learning

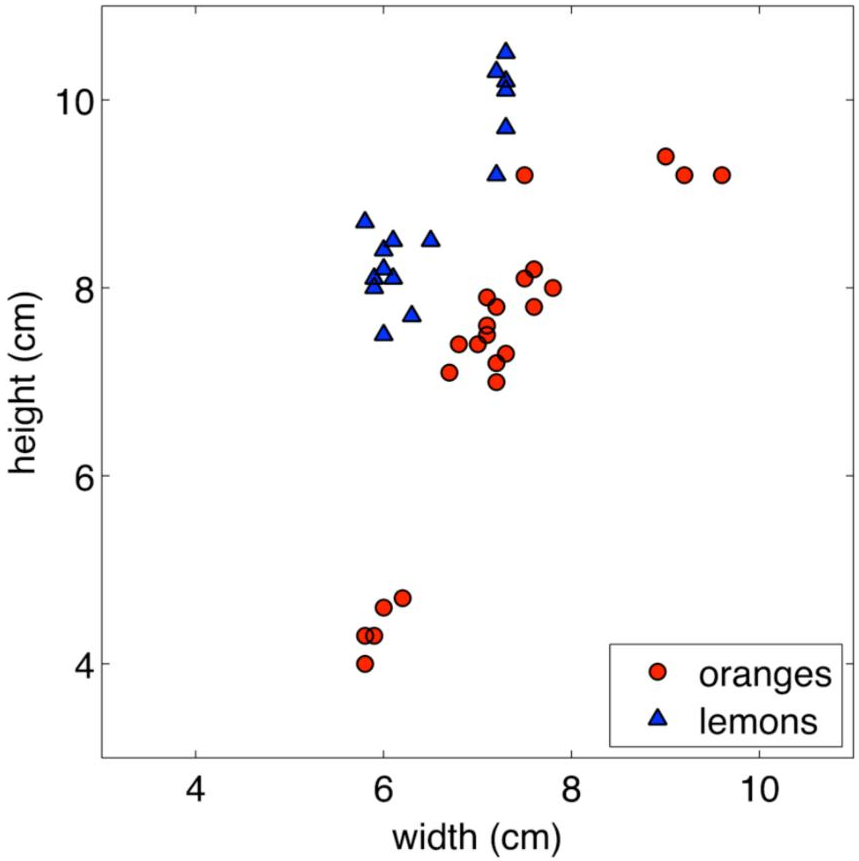
Zhiyun Lin

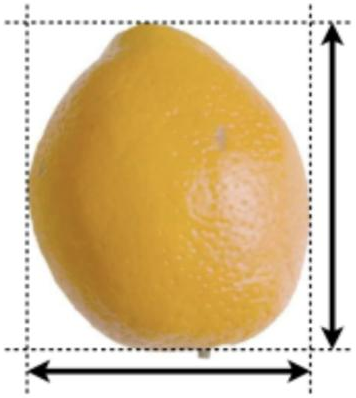
南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Nearest Neighbors

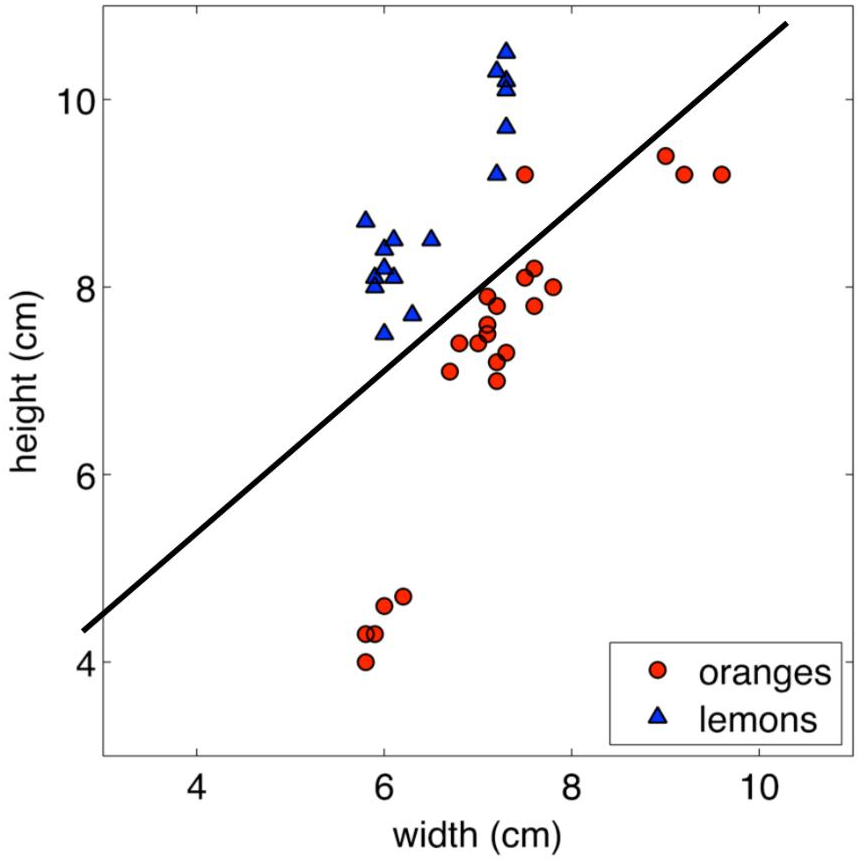
- Non-parametric Models
- Distance
- Nonlinear Decision Boundaries

Classification: Oranges and Lemons



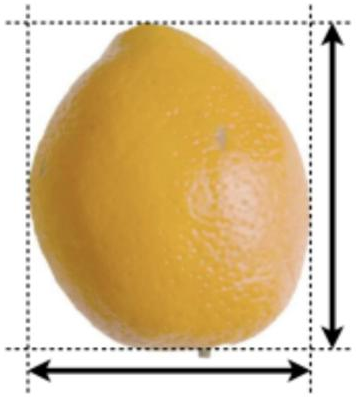


Classification: Oranges and Lemons



Can construct simple linear decision boundary:

$y = \text{sign}(w_0 + w_1 x_1 + w_2 x_2)$



Nonlinear Classification Nature

- Classification is intrinsically **non-linear**
  - It puts non-identical things in the same class, so a difference in the input vector sometimes causes zero change in the answer
- **Linear classification** means that the part that adapts is linear (just like linear regression)

$$z(x) = w^T x + w_0$$

- The adaptive part is followed by a non-linearity to make the **decision**

$$y(x) = f(z(x))$$

- What functions $f()$ have we seen so far in class?

Instance-based Learning

- Alternative to parametric models are **non-parametric models**
- These are typically simple methods for approximating discrete-valued or real-valued target functions (they work for classification or regression problems)
- **Learning** amounts to simply **storing** training data
- Test instances classified using **similar** training instances
- Embodies often sensible underlying **assumptions**:
  - Output varies smoothly with input
  - Data occupies sub-space of high-dimensional input space

Nearest Neighbors

- Training example in Euclidean space: $x \in \mathbb{R}^d$

> **Idea:** The value of the target function for a new query is estimated from the known value(s) of the **nearest training example(s)**

- Distance typically defined to be Euclidean:

$$||x^{(a)} - x^{(b)}||_2 = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - x_j^{(b)})^2}$$

Algorithm:

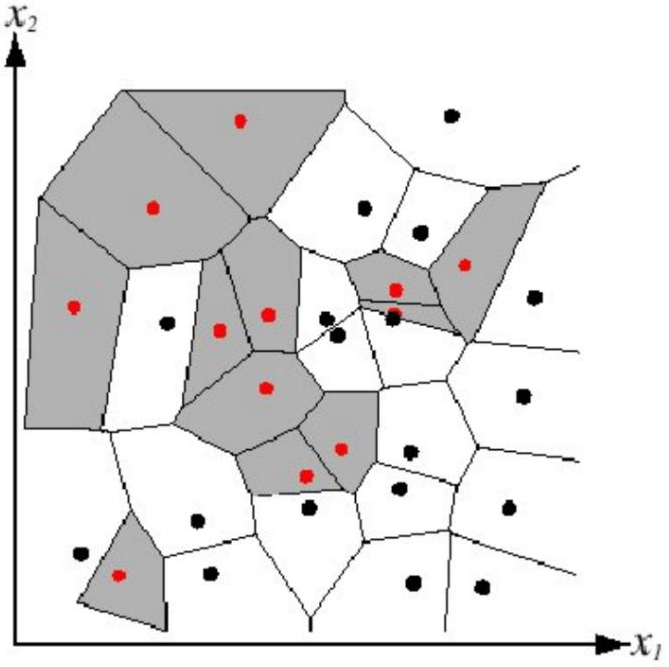
1. Find example $(\mathbf{x}^*, t^*)$ (from the stored training set) closest to the test instance $\mathbf{x}$. That is:

$$\mathbf{x}^* = \underset{\mathbf{x}^{(i)} \in \text{train. set}}{\text{argmin}} \ \text{distance}(\mathbf{x}^{(i)}, \mathbf{x})$$

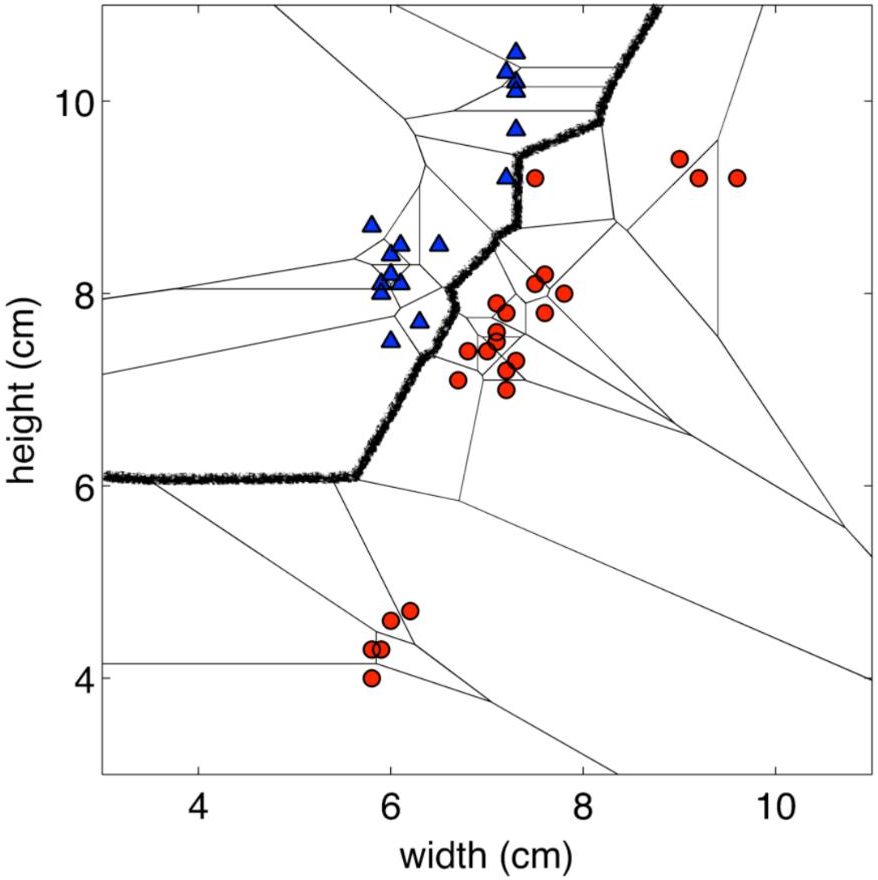
2. Output $y = t^*$

Nearest Neighbors: Decision Boundaries

- NN does not explicitly compute **decision boundaries** but can be inferred
- Decision boundaries: **Voronoi diagram** visualization
  - Show how input space divided into classes
  - Each line segment is equidistant between two points of opposite classes

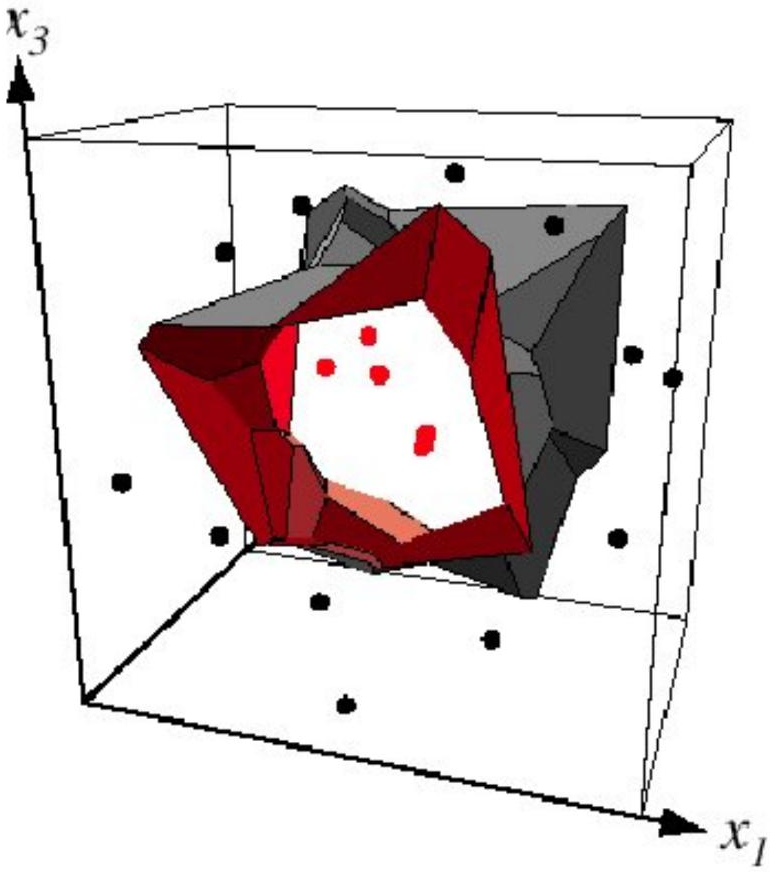


Nearest Neighbors: Decision Boundaries



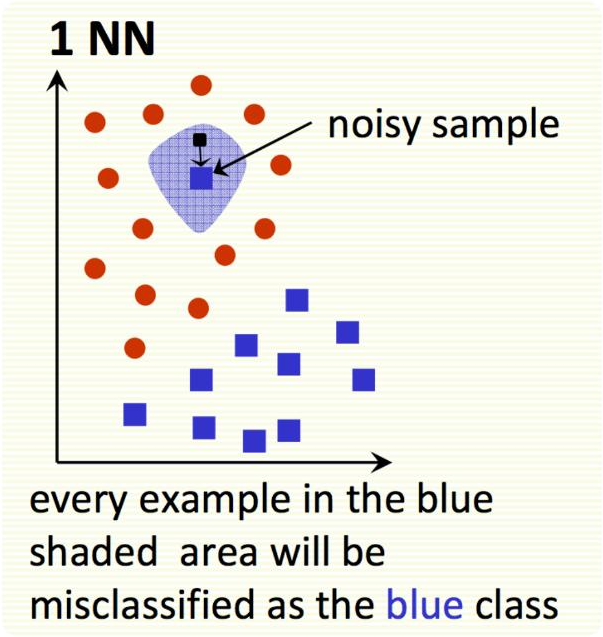
> Example: 2D decision boundary

Nearest Neighbors: Decision Boundaries



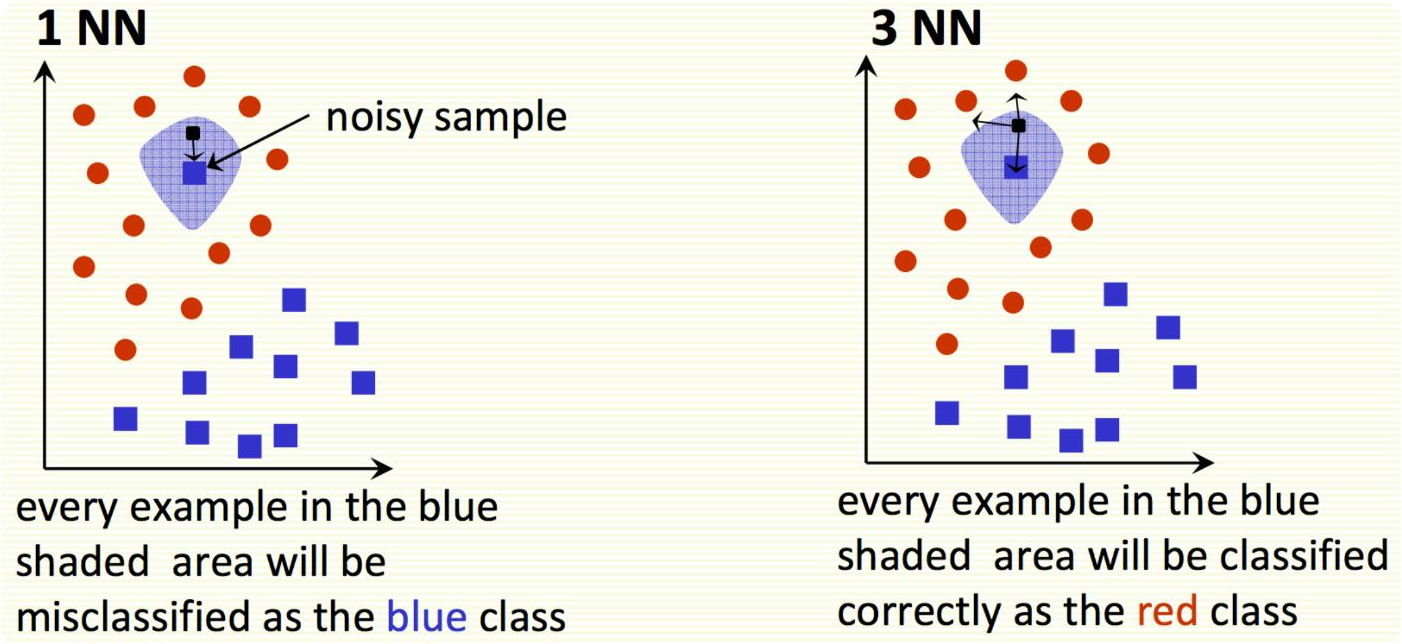
> Example: 3D decision boundary

Nearest Neighbors



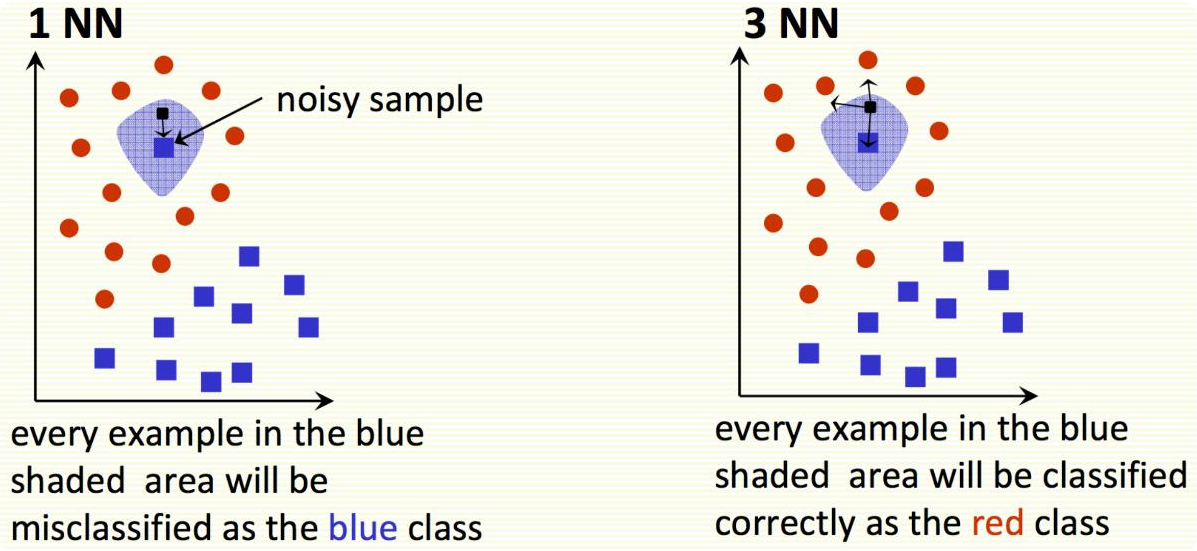
- Nearest neighbors sensitive to mis-labeled data ("class noise"). Solution?

k-Nearest Neighbors



- Nearest neighbors sensitive to mis-labeled data ("class noise"). Solution?
- Smooth by having k nearest neighbors vote

k-Nearest Neighbors



- Smooth by having k nearest neighbors vote

Algorithm (kNN):

1. Find k examples $\{\mathbf{x}^{(i)}, t^{(i)}\}$ closest to the test instance $\mathbf{x}$
2. Classification output is majority class

$$y = arg\max_{t^{(z)}} \sum_{r=1}^{k} \delta(t^{(z)}, t^{(r)})$$

k-Nearest Neighbors

- How do we **choose k**?
  - Larger k may lead to better performance
  - But if we set k too large we may end up looking at samples that are not neighbors (are far away from the query)
  - We can use cross-validation to find k
  - Rule of thumb is $k < \sqrt{n}$, where n is the number of training examples

k-Nearest Neighbors: Issues & Remedies

- If some attributes (coordinates of x) have larger **ranges**, they are treated as more important
  - normalize scale
    - **Simple option:** Linearly scale the range of each feature to be, e.g., in range $[0,\ 1]$
    - Linearly scale each dimension to have 0 mean and variance 1: Compute mean μ and variance σ^2 for an attribute x_j and scale: $(x_j - m)/\sigma$
  - be careful: sometimes scale matters
- **Irrelevant, correlated attributes** add noise to distance measure
  - eliminate some attributes
  - or vary and possibly adapt weight of attributes

k-Nearest Neighbors: Issues & Remedies

- Non-metric attributes (symbols)
  - Hamming distance

> Example: Suppose there are two strings $1101\ 1001$ and $1001\ 1101$.

$$1101\ 1001 \oplus 1001\ 1101 = 0100\ 0100$$

> It contains two 1s, the Hamming distance $d(1101\ 1001, 1001\ 1101) = 2$

k-Nearest Neighbors: Issues & Remedies

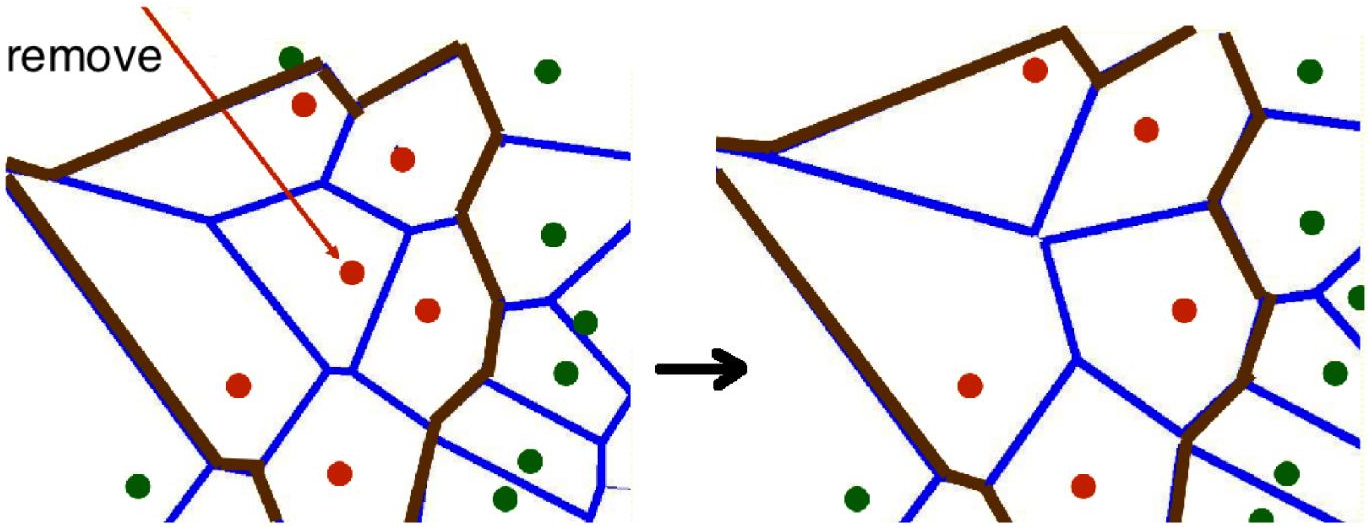
- **Expensive at test time:** To find one nearest neighbor of a query point x, we must compute the distance to all N training examples. **Complexity: $O(kdN)$ for k-NN**
  - Use subset of dimensions
  - Pre-sort training examples into fast data structures (e.g., kd-trees)
  - Compute only an approximate distance (e.g., LSH)
  - Remove redundant data (e.g., condensing).

k-Nearest Neighbors: Issues & Remedies

- **Storage Requirements:** Must store all training data
  - Remove redundant data (e.g., condensing).
  - Pre-sorting often increases the storage requirements
- **High Dimensional Data:** "Curse of Dimensionality"
  - Required amount of training data increases exponentially with dimension
  - Computational cost also increases

k-Nearest Neighbors: Issues & Remedies

> If all Voronoi neighbors have the same class, a sample is useless, remove it



Certainly! Here is the explanation of KD tree in English, formatted in Markdown:

KD Tree Concept

KD tree (K-Dimensional Tree) is a data structure used to organize points in a multi-dimensional space.

- It is similar to a **binary search tree** in two-dimensional space but extended to K-dimensional space.
- KD trees are primarily used for **efficiently solving multi-dimensional space search problems**, such as nearest neighbor search and range search.

Definition of KD Tree

- **Node**: Each node in a KD tree represents a point in K-dimensional space.
- **Axis Selection**: During the construction of the tree, a dimension (axis) is chosen each time and points are partitioned based on that dimension's value.
- **Partitioning**: A value is chosen along the selected axis to split the points into two parts: **one part** with points having values less than the chosen value on that axis, and **the other part** with points having values greater than or equal to the chosen value.
- **Recursive Construction**: The same process is applied recursively to each subset until each subset contains only one point or no points.

Steps to Build a KD Tree

1. Choose a dimension (axis).
2. Sort the points along that dimension.
3. Select the **median** as the partition point, dividing the points into two groups.
4. Recursively apply the above steps to each subset until each subset contains only one point or no points.

Example of a KD Tree

> Suppose we have the following set of points in two-dimensional space:
>
> $$P = \{(1,2), (3,5), (2,3), (5,4), (3,2), (4,5)\}$$

- **Step 1:** Choose the first dimension (e.g., x-axis) and sort
  - Sorted point set: {(1,2), (2,3), (3,2), (3,5), (4,5), (5,4)} .
- **Step 2:** Choose the median as the partition point
  - The median is the third point ((3,2)), and we divide the points into two groups:
    - Left subtree: {(1,2), (2,3)}
    - Right subtree: {(3,5), (4,5), (5,4)}

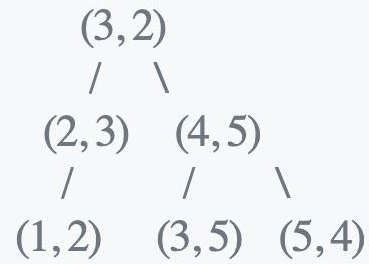
Example of a KD Tree

- **Step 3:** Recursively build
  - For the left subtree, we choose the y-axis as the next dimension, and the median is (2,3), dividing into:
    - Left subtree: {(1,2)}
    - Right subtree: Empty.
  - For the right subtree, we again choose the x-axis as the next dimension, and the median is (4,5) , dividing into:
    - Left subtree: {(3,5)}
    - Right subtree: {(5,4)} .

Example of a KD Tree

Result

The final KD tree is as follows:



> This tree structure allows us to efficiently perform spatial searches, such as finding the nearest neighbor of a given point or points within a certain range.

Python Codes for KD Tree

```
from scipy.spatial import KDTree
import numpy as np

class KNN:
    def __init__(self, k=3):
        """
        初始化KNN类
        :param k: 邻居的数量
        """
        self.k = k
        self.kdtree = None
```

Python Codes for KD Tree

```
def fit(self, X, y):
    """
    :param X: 训练数据的特征，形状为(n_samples, n_features)
    :param y: 训练数据的标签，形状为(n_samples,)
    """
    self.X_train = X
    self.y_train = y
    self.kdtree = KDTree(X)

def predict(self, X):
    """
    预测新数据点的标签
    :param X: 需要预测的数据点的特征，形状为(n_samples, n_features)
    :return: 预测的标签，形状为(n_samples,)
    """
    predictions = [self._predict(x) for x in X]
    return np.array(predictions)
```

Python Codes for KD Tree

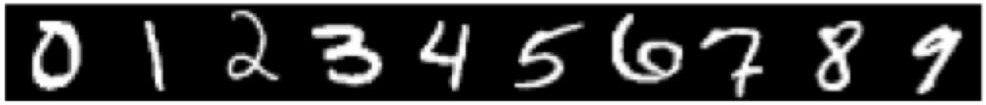
```
def _predict(self, x):
    """
    预测单个数据点的标签
    :param x: 单个数据点的特征，形状为(1, n_features)
    :return: 预测的标签
    """
    # 找到k个最近邻
    dist, idx = self.kdtree.query(x, k=self.k, p=2)
    # 如果idx是一维数组（即查询单个点），则将其转换为列表
    if idx.ndim == 1:
        idx = [idx] # 将一维数组转换为包含单个元素的列表
    # 获取这些邻居的标签
    neighbors_labels = [self.y_train[i] for i in idx[0]]
    # 计算出现次数最多的标签
    prediction = max(set(neighbors_labels), key=neighbors_labels.count)
    return prediction
```

Python Codes for KD Tree

```
# 使用示例
if __name__ == "__main__":
    # 假设我们有一些训练数据
    X_train = np.array([[1, 2], [2, 3], [3, 1], [6, 5], [7, 7], [8, 6]])
    y_train = np.array([0, 0, 0, 1, 1, 1])

    # 创建KNN实例并训练
    knn = KNN(k=3)
    knn.fit(X_train, y_train)

    # 假设我们有一些测试数据
    X_test = np.array([[1, 1], [3,1], [5, 5]])

    # 进行预测
    predictions = knn.predict(X_test)
    print("预测结果:", predictions)
```

Example: Digit Classification

- Decent performance when lots of data



- Yann LeCunn – MNIST Digit Recognition
  - Handwritten digits
  - 28x28 pixel images: $d = 784$
  - 60,000 training samples
  - 10,000 test samples
- Nearest neighbour is competitive

	Test Error Rate (%)
Linear classifier (1-layer NN)	12.0
K-nearest-neighbors, Euclidean	5.0
K-nearest-neighbors, Euclidean, deskewed	2.4
K-NN, Tangent Distance, 16x16	1.1
K-NN, shape context matching	0.67
1000 RBF + linear classifier	3.6
SVM deg 4 polynomial	1.1
2-layer NN, 300 hidden units	4.7
2-layer NN, 300 HU, [deskewing]	1.6
LeNet-5, [distortions]	0.8
Boosted LeNet-4, [distortions]	0.7

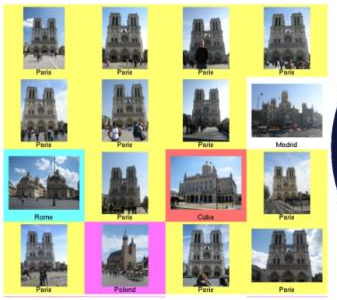
Fun Example: Where on Earth is this Photo From?

- Problem: Where (e.g., which country or GPS location) was this picture taken?

> [Paper: James Hays, Alexei A. Efros. im2gps: estimating geographic information from a single image. http://graphics.cs.cmu.edu/projects/im2gps/

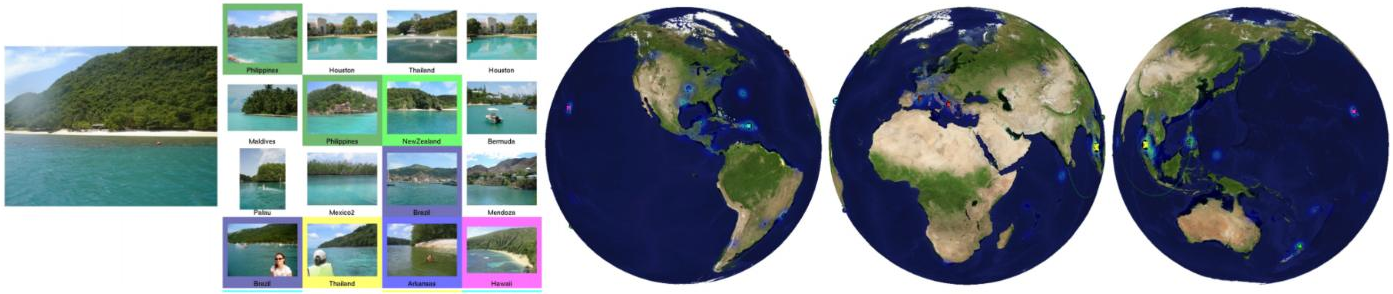
Fun Example: Where on Earth is this Photo From?

- Problem: Where (e.g., which country or GPS location) was this picture taken?
  - Get 6M images from Flickr with GPS info (dense sampling across world)
  - Represent each image with meaningful features
  - Do k-NN!

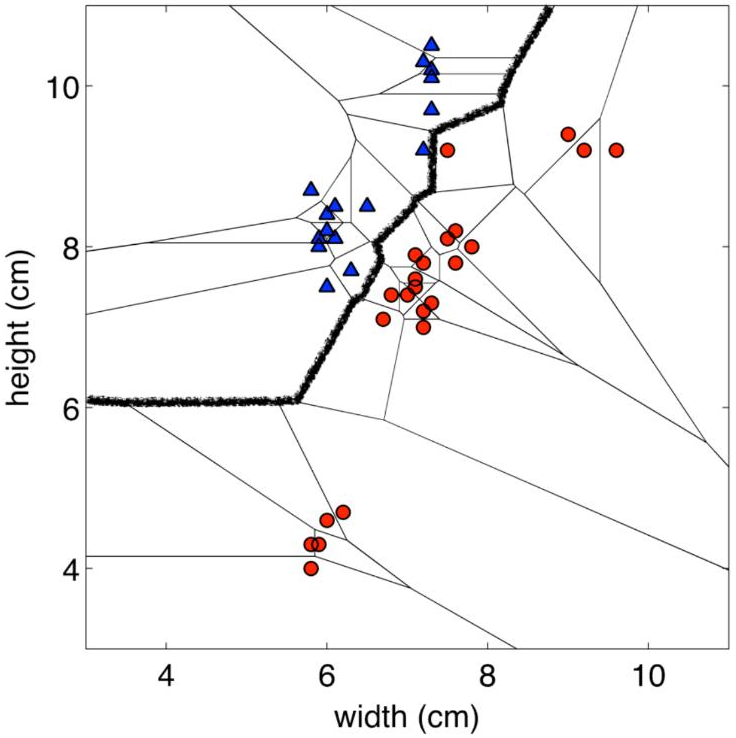


Fun Example: Where on Earth is this Photo From?

- Problem: Where (eg, which country or GPS location) was this picture taken?
  - Get 6M images from Flickr with GPS info (dense sampling across world)
  - Represent each image with meaningful features
  - Do k-NN (large k better, they use $k = 120$)!



k-NN Summary



- Naturally forms **complex decision boundaries**; adapts to data density
- If we have lots of samples, k-NN typically works well

k-NN Summary

- Problems:
  - Sensitive to class noise
  - Sensitive to scales of attributes
  - Distances are less meaningful in high dimensions
  - Scales linearly with number of examples