

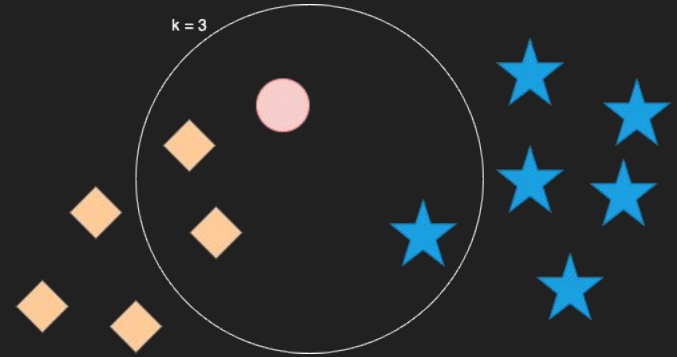
KNN Algorithm

K Nearest Neighbor

Preni Amijanian, Jessica Margala, Arsham Mehrani

Group #9

Group Representative: Arsham Mehrani



Introduction

The K-Nearest-Neighbors algorithm, also known as KNN algorithm, where k is a number greater than 0, is most known for its usefulness in Machine Learning Applications.

There are multiple applications of the KNN algorithm. For example, it can be used to solve classification problems, regression problems & missing data imputation problems.

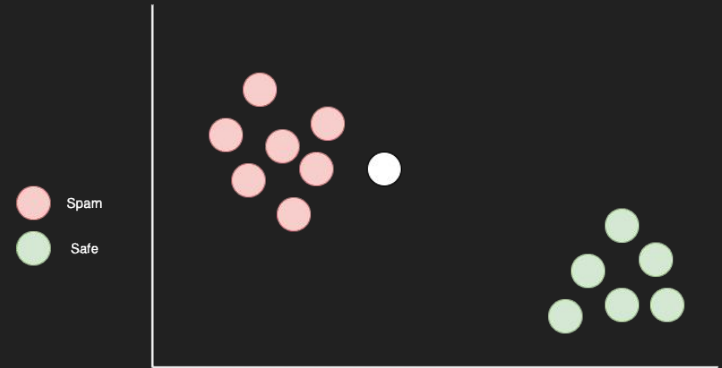
Today we'll be focusing on the KNN classification algorithm.

Introduction (cont.)

The KNN algorithm is considered an **instance-based learning** algorithm.

What is instance-based learning?

Instance-based learning is a Machine Learning model that utilizes memorization to “learn” from previous instances and then generalizes new instances based on some measures of similarity between the previously memorized instances.




Use Cases

- The recommendation feature on Amazon and other sites. (Using genres, keywords, ...)
- Determining eligibility of an applicant for a loan. (Using income, credit score, ...)
- Also, used in Finance to search for the best stocks.


List: \$45.99 (15% off)

Sponsored products related to this item


Sponsored




Satechi Type-C Aluminum USB Hub & Micro/SD Card Reader - Compatible with 2020...
★★★★☆ 176
\$34.99 ✓prime




Satechi Slim W1 Wired Backlit Keyboard - Illuminated Keys & Built-in USB-C Connecti...
★★★★☆ 16
\$59.99 ✓prime




Satechi Aluminum Type-C USB 3.0 3-in-1 Combo Hub with USB-C Pass-Through - Compatib...
★★★★☆ 337
\$32.99 ✓prime



Satechi Aluminum Type-C HDMI Adapter 4K (60Hz) - Compatible with 2016/2017/2018 Mac...
★★★★☆ 46
\$34.99 ✓prime



Satechi Aluminum Wireless Presenter Pointer Remote Control - Compatible with 2019...
★★★★☆ 310
\$39.99 ✓prime




Satechi Aluminum Multi-Port Adapter V2 - 4K HDMI (60Hz), Gigabit Ethernet, USB-C Ch...
★★★★☆ 2,669
\$79.99 ✓prime

Special offers and product promotions

- **Business Prime** : For Fast, FREE shipping, premium procurement benefits, and member-only offers on Amazon Business. **Try Business Prime free.**
- **Business Prime** : Fast, FREE shipping for small businesses with plans starting at \$69. **Start your FREE 30-day trial**

Have a question?

Find answers in product info, Q&As, reviews



Roll over image to zoom in

- **PROTECTS YOUR DEVICES** - equipped with the perfect height and angle of traction areas, the Aluminum Desk prevents unwanted scratches on your aluminum devices.
- **SLEEK & SECURE DESIGN** - its solid, securely holds all iPad sizes with a secure grip, making it easy to set up in your home office setup.
- **DESIGNED FOR IPAD** - 2020/2019 iPad Max/12 Pro/12 Mini/12, iPhone 11 to 12" smartphones & tablet devices.

Compare with similar items
Report incorrect product information

Similar item to consider

Amazon Basics Multi-Angle Desktop Stand for Tablet and Phone - Black
★★★★☆ (14711)
\$10.49 ✓prime

Thoughtful Design with...
★★★★★
Shop now

Frequently bought together

Some of these items ship sooner than the others. Show details

This Item: Satechi Aluminum Desktop Stand - Adjustable Tablet Mount with Protective Grips - Compatible with... \$44.99

Satechi Aluminum M1 Bluetooth Wireless Mouse with Rechargeable Type-C Port - Compatible with Mac... \$29.99

Satechi Compact Backlit Bluetooth Keyboard - Wireless Bluetooth 5.0 & Multi-Device Sync - Compatible... \$79.99

Total price: **\$154.97**
Add all three to Cart
Add all three to List

Use Cases (cont)



- KNN Matting: KNN algorithm is used to separate an object in the foreground from its background. (used for unknown regions)



Description

- KNN is a supervised Machine Learning algorithm
- KNN is a non-parametric algorithm
- Is called a lazy algorithm

Example:

training set $\rightarrow \{[3,3, \text{'ClassA'}], [7,7, \text{'ClassB'}]\}$

test item $\rightarrow [6,6]$

\Rightarrow the result $[6,6, \text{'ClassB'}]$

Algorithm (Brute force)

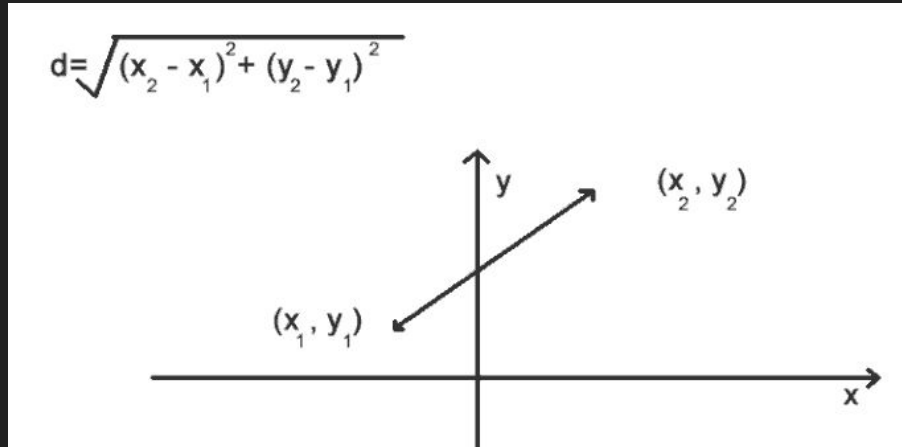
1. Load the data and prep the data. (ex: standardize, check for false entries, etc.)
2. Choose the value of K which determines the number of neighbors to be analyzed (K should be an odd number to prevent undetermined results).
3. For each classified point in the data set:
 - Determine the distance to the current unclassified query.
 - Store these distances in an ordered list (non-decreasing).
4. From the ordered set choose the first K items. (These are K Nearest items).
5. For classification assign the node of these K items to the unclassified input.

Description

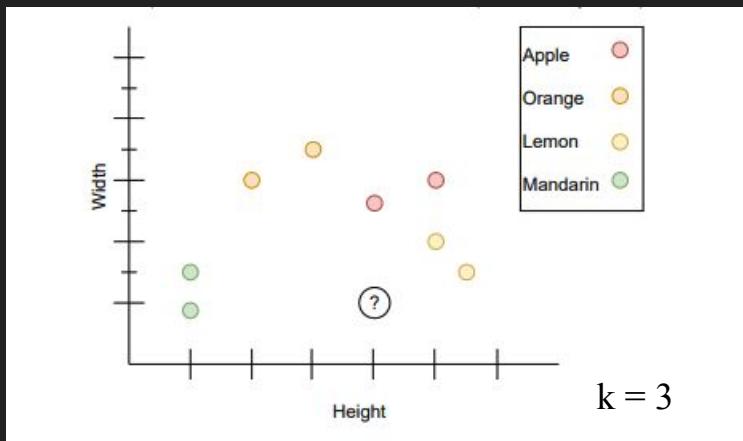
For Classification, this algorithm uses discrete values and finds the distance between the new item and the existing items.

- This is done by finding the least distance using Euclidean Distance

* If $k = 1$, to classify the item, we use the nearest neighbor (the smallest distance).



Example using Brute Force: Step 1



Step 1: Get the distance from classification point to every other point in dataset and store them in a list

$$\text{distance}((x, y), (a, b)) = \sqrt{(x - a)^2 + (y - b)^2}$$

$$\text{unknownFruit}(4,1) \text{ to mandarin1}(1,1) \quad \sqrt{(4 - 1)^2 + (1 - 1)^2} = 3$$

$$\text{unknownFruit}(4,1) \text{ to mandarin2}(1,1.5) \quad \sqrt{(4 - 1)^2 + (1 - 1.5)^2} = 3.04$$

$$\text{unknownFruit}(4,1) \text{ to orange1}(2,3) \quad \sqrt{(4 - 2)^2 + (1 - 3)^2} = 2.83$$

$$\text{unknownFruit}(4,1) \text{ to orange2}(3,3.5) \quad \sqrt{(4 - 3)^2 + (1 - 3.5)^2} = 2.69$$

$$\text{unknownFruit}(4,1) \text{ to apple1}(4,2.5) \quad \sqrt{(4 - 4)^2 + (1 - 2.5)^2} = 1.5$$

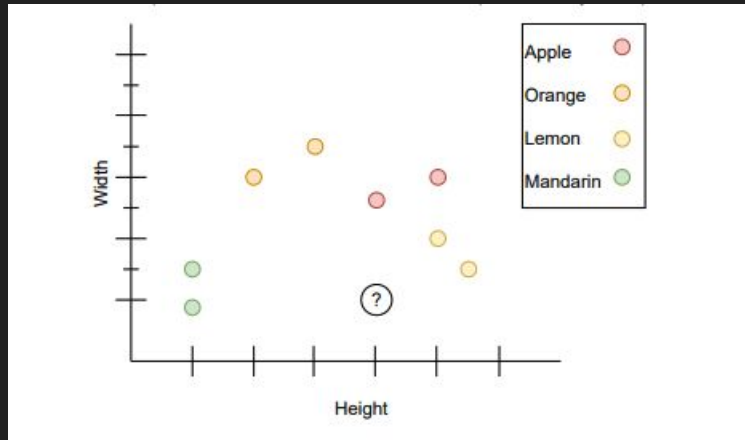
$$\text{unknownFruit}(4,1) \text{ to apple2}(5,3) \quad \sqrt{(4 - 5)^2 + (1 - 3)^2} = 2.24$$

$$\text{unknownFruit}(4,1) \text{ to lemon1}(5.5,1.5) \quad \sqrt{(4 - 5.5)^2 + (1 - 1.5)^2} = 1.58$$

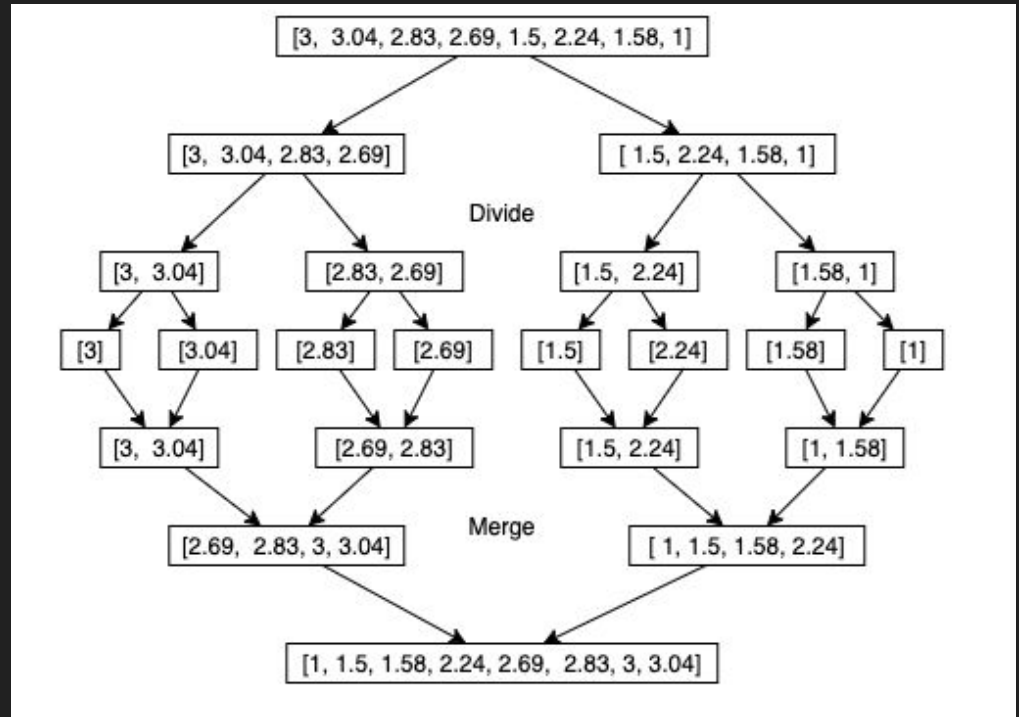
$$\text{unknownFruit}(4,1) \text{ to lemon2}(4,2) \quad \sqrt{(4 - 4)^2 + (1 - 2)^2} = 1$$

List of Distances: [3, 3.04, 2.83, 2.69, 1.5, 2.24, 1.58, 1]

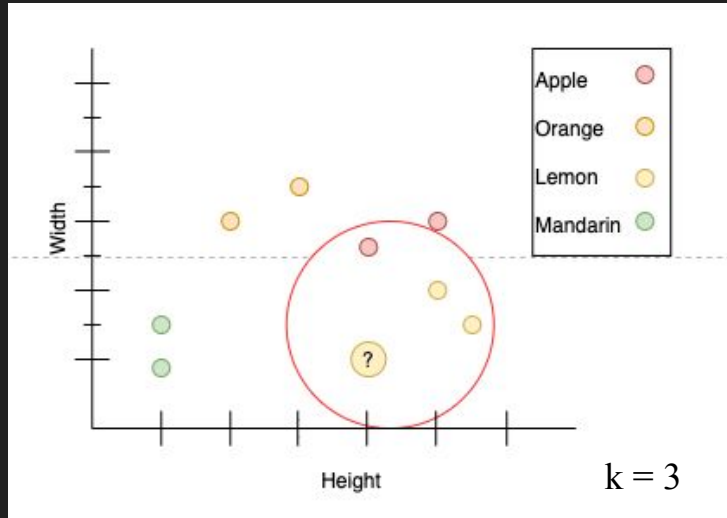
Example using Brute Force: Step 2



Step 2: Sort the list of distances in a non-decreasing order using MergeSort



Example using Brute Force: Step 3



Step 3: Make predictions using k closest neighbors

[1, 1.5, 1.58, 2.24, 2.69, 2.83, 3, 3.04]

K Nearest Neighbors:

1 -> lemon2 (4,2)

1.5 -> apple1 (4,2.5)

1.58 -> lemon1 (5.5,1.5)

Frequency of K Nearest Neighbors:

Apple: 1

Orange: 0

Lemon: 2

Mandarin: 0



Final Prediction: Lemon

k-d tree Data Structure used for KNN algorithm

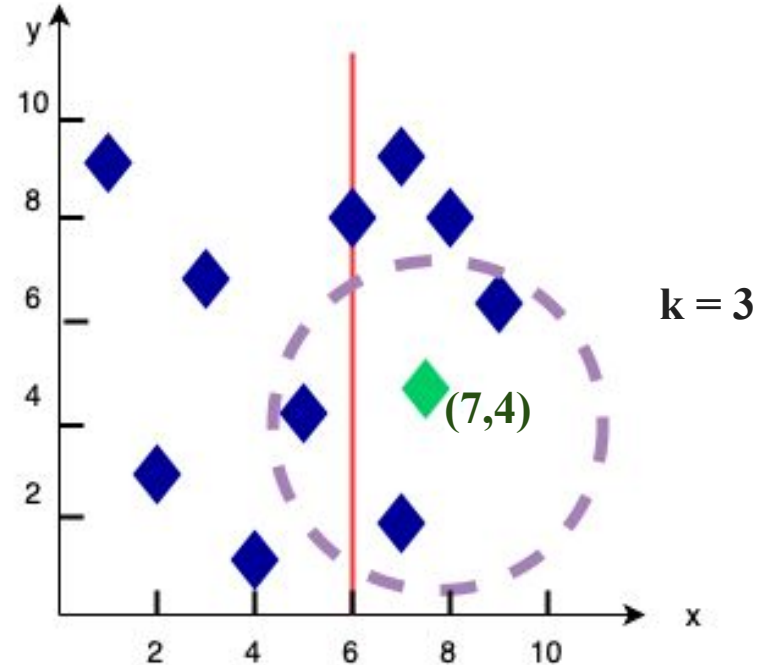
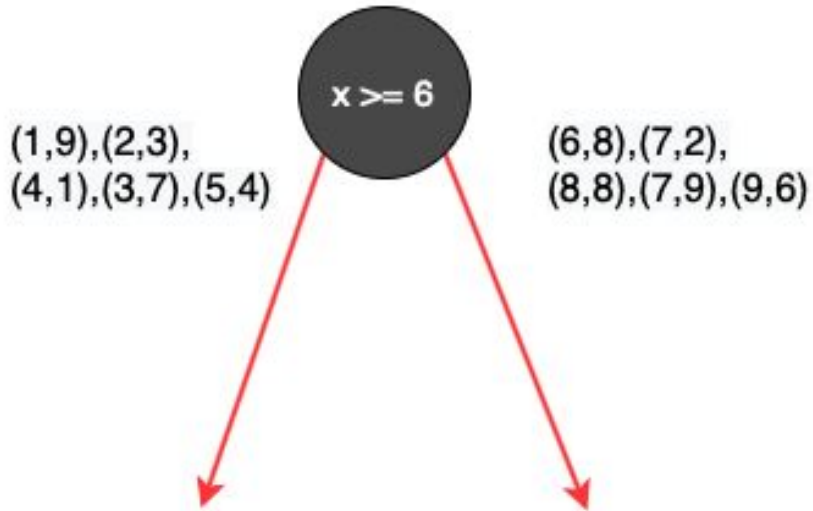
k-d tree is a binary tree. To organize the dataset into a tree, we take all the training instances and follow these steps:

1. Start with the root node
2. Take a random attribute (in this case either x or y components)
3. Find the median of that attribute
4. Split the data based on the median value
5. Repeat these steps by using the other (unused) attribute

The test point would start from the root and move down. Depending on if the test point is greater or less than the current node it's in, it would go to the right or left child.

Step 1:

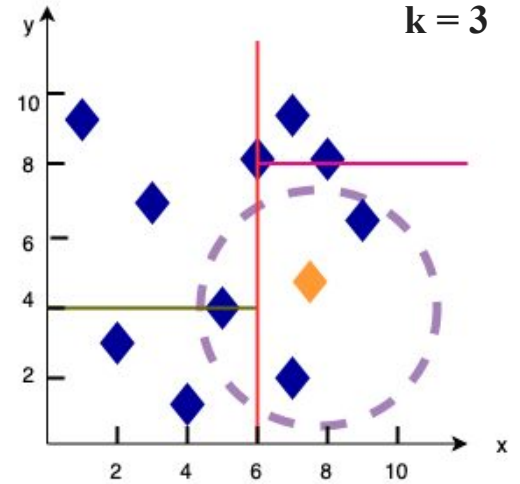
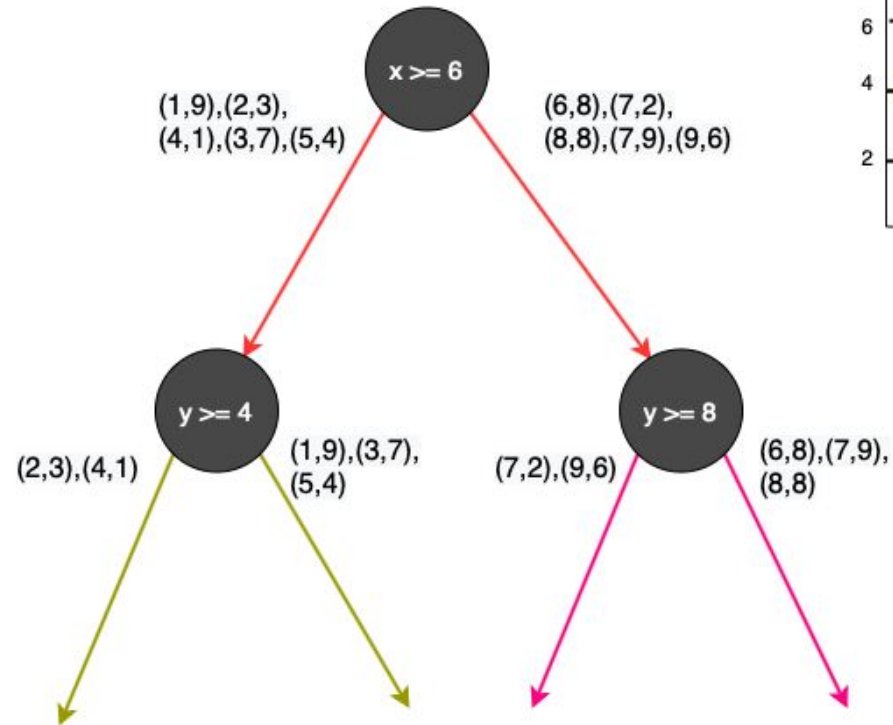
training dataset: $\{(1,9),(2,3),(4,1),(3,7),(5,4),(6,8),(7,2),(8,8),(7,9),(9,6)\}$



Step 2:

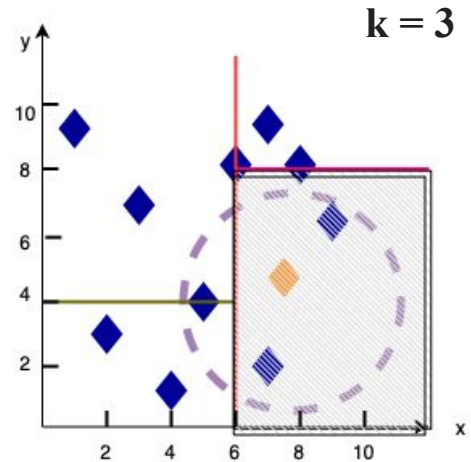
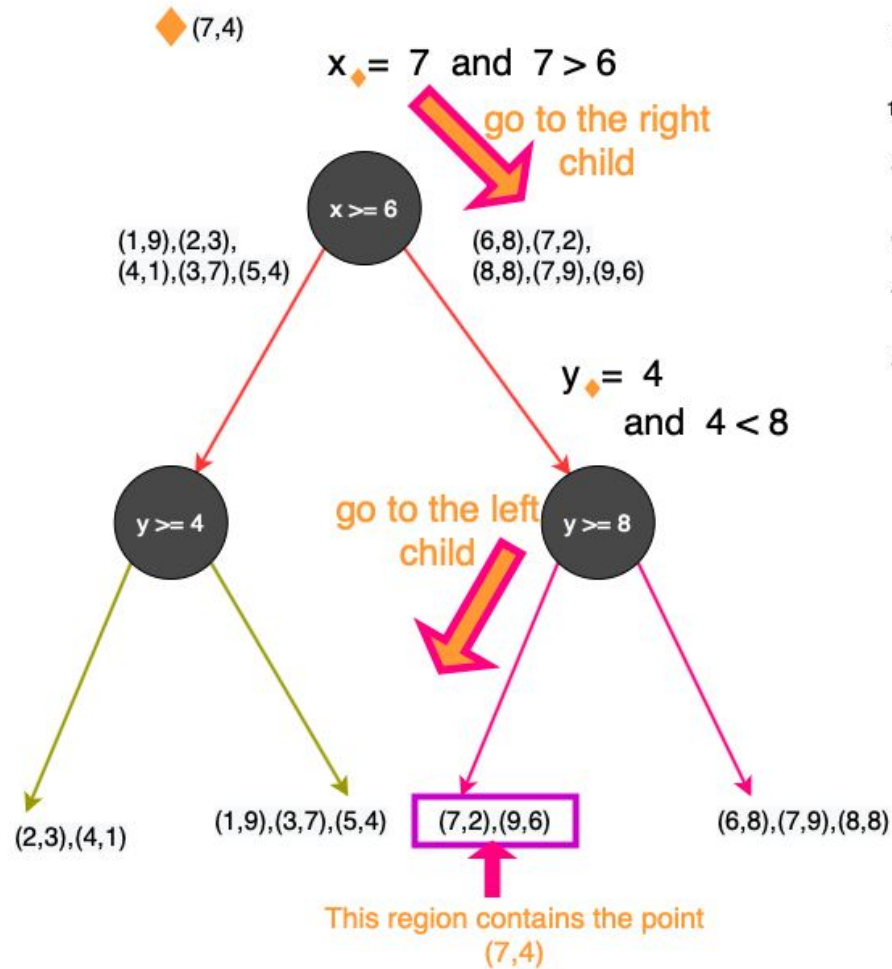
testing point: ◆ (7,4)

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



Step 3:

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



Design Techniques

- Data prepping is the most important initial step of this algorithm.
- Obtain a data to train and test the algorithm on. Use 70% for training and 30% for testing.
- Calculate the Euclidean distance between the query node to all other nodes.
- These values are stored using the following ways:
 - Hashmap
 - Two value tuple in python
 - Or a list of objects with distance of the nodes as their field in java.
- Then the array is sorted based on these distance and K smallest distances are selected, then these nodes are returned to guess the classification of the unknown node.

Pseudocode (Brute-Force)

```
//TrainingObject contains the x and y coordinates, the class category, and distance
//assuming the dimension is 2

void knn (int k, number unkownDataX, number unkownDataY, TrainingObject trainingArr[])
for (index i = 0; i less than trainingArr.length; i++)
    //method to calculate the distance between the new input and training data
    trainingArr[i].distance = euclideanDistance (trainingArr[i].x, trainingArr[i].y,
                                                unkownDataX, unkownDataY)

Sort the distances (in a non-decreasing fashion) //can use merge sort or quick sort for time
                                                complexity of nlogn

for (index i = 0; i less than k; i++) //select the first k values from the distance (the k
                                    closest objects to the unknown data)

    //the data from which class do we see the most surrounding the unknown data

    calculate the frequency of each class category

    based on the majority vote, it is determined which class the undecided item belongs to
```

Implementation of kNN Algorithm using Brute Force Method

Time Complexity

The time complexity of the kNN algorithm for classification can depend on the size of the data set that is being used, similar to all instance-based learning algorithms.

Another factor which may impact our time complexity is which data structure we use to implement it.

For the kNN algorithm we can either just use **brute force method**, or we can implement different **tree** data structures.

Time Complexity using Brute Force

If **n** is equal to the number of points in the data set, **k** is equal to the number of neighbors we consider and **d** is the number of dimensions in the data set:

1 - Calculate distance to every other point:

$$\Theta(n)$$

2 - Sort all the distances using **Merge Sort**:

$$\Theta(n \log n)$$

3 - Calculate the frequency of each class to find k nearest neighbors:

$$\Theta(1)$$

Overall Time Complexity:

$$\Theta(d * n^2 \log n)$$

Note: There is no training phase, all computation is done during prediction phase.

*Time complexity will vary depending on sorting algorithm used

Time Complexity using k-d Tree

If **n** is equal to the number of points in the data set, **k** is equal to the number of neighbors we consider and **d** is equal to the number of dimensions:

Build the k-d Tree (Training Phase): Insert elements for every point **n** with **d** dimensions.

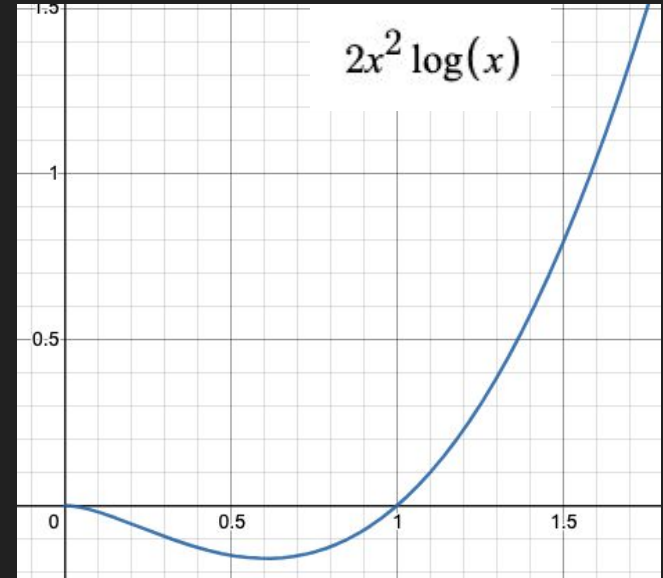
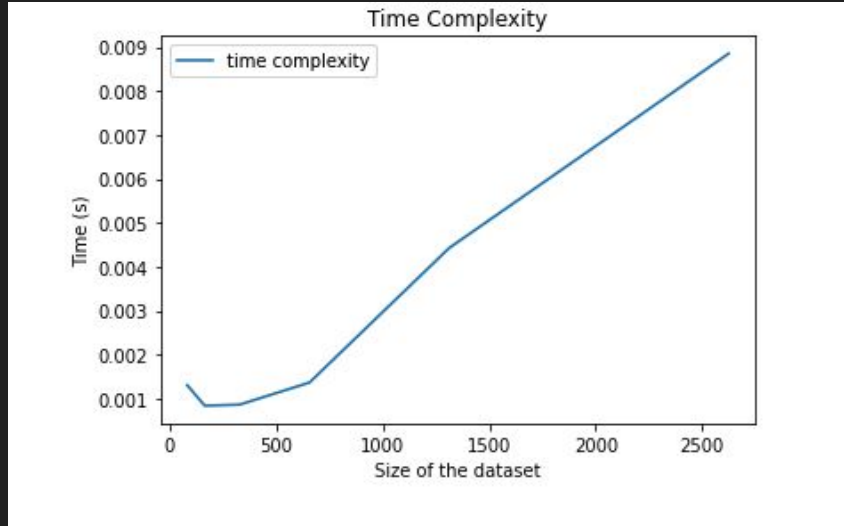
$$\Theta(dn \log n)$$

Search the Tree: Takes $(\log n)$ per data point of query, so total time takes $(n \log n)$ for **n** data query points.

$$\Theta(n \log n)$$

Group Findings

The graph on the left is the graph we obtained when we timed the code as the size of the training dataset increased (using $k=5$ and for 6 iterations). The graph on the right is the graph of the time complexity $2n^2 \log(n)$



Algorithm Pros and Cons

Pros

- It is simple, and powerful!
- No training step!
- Useful for many things such as regression and classification.
- Highly customizable! ex: Instead of looking at all the points distances, local linear approximation could be applied to minimize the number of calculations.

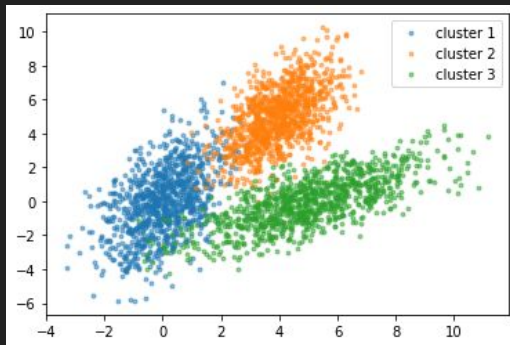
Cons

- For each classification the whole algorithm needs to start from scratch. As a result, it takes a lot of computational power.
- If k is chosen at random without data set in mind results could be wrong.
- If data set is not well prepped the classification accuracy drops.

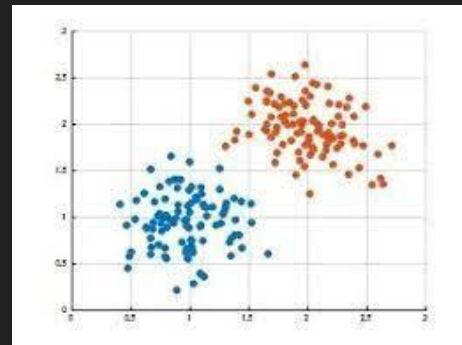
Challenges

- For this algorithm is important that the data set is distinct set of items. If the data set is not distinct the classification could be false.
- If the set of items being classified are not distinct enough:
 - Introduce more classification parameters to make items distinct. Ex: Mass, color etc..
 - Re-evaluate distances once the K closest items are selected to eliminate outliers.

Unapparent



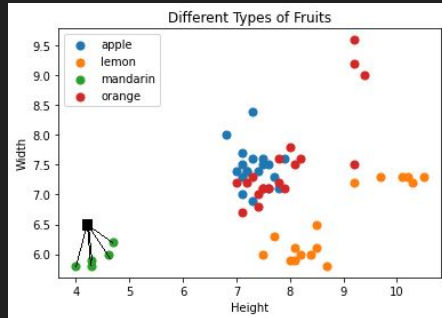
Distinct



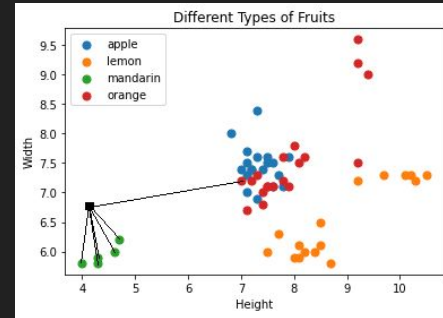
Suggestions

- When k is an even number and a tie occurs between the k closest classes, introduce a function to determine which class would be better fit by evaluating the closest distances.
- Using a data structure, like the KD tree or other tree structures, would optimize the algorithm's overall performance and time complexity.
- The value of k should be chosen with the data set in mind. If value of k is too large or too small depending on the data set could result in a false classification.

correct



incorrect



References

- Deng, Zhenyun, Zhu, Xiaoshu, Cheng, Debo, Zong, Ming, & Zhang, Shichao. (2016). Efficient kNN classification algorithm for big data. *Neurocomputing (Amsterdam)*, 195, 143–148. <https://doi.org/10.1016/j.neucom.2015.08.112>
- Pandey, A., & Jain, A. (2017). Comparative analysis of KNN algorithm using various normalization techniques. *International Journal of Computer Network and Information Security*, 10(11), 36. doi:<http://dx.doi.org.proxy.library.cpp.edu/10.5815/ijcnis.2017.11.04>
- Zhang, Shichao, Cheng, Debo, Deng, Zhenyun, Zong, Ming, & Deng, Xuelian. (2018). A novel kNN algorithm with data-driven k parameter computation. *Pattern Recognition Letters*, 109, 44–54. <https://doi.org/10.1016/j.patrec.2017.09.036>
- Zhang, Shichao, Li, Xuelong, Zong, Ming, Zhu, Xiaofeng, & Cheng, Debo. (2017). Learning k for kNN Classification. *ACM Transactions on Intelligent Systems and Technology*, 8(3), 1–19. <https://doi.org/10.1145/2990508>

Questions?