Nearest Neighbor

George Stamatelis, Odysseas Stefas

University of Athens

sdi1800185@di.uoa.gr

June 29, 2021

Overview

Introduction

Problem Statement

Nearest Neighbor search is a form of proximity search used extensively by scientists in the fields of data mining and machine learning. Given a new data point x , we try to find the nearest point n in the training dataset . x is assigned the same label as point n.

While that is a simple and effective classification algorithm ,when the training data (and test data) is very large , it becomes way to slow for practical use.It has time complexity of:
O(#of Train Examples $*$ #*of* test Examples $*$# of dimensions)

Complexity explained

- We want to classify each point of the test set
- the euclidean distance of points a and b is given by

$$(x_{a1} - x_{b1})^2 + ... + (x_{ad} - x_{bd})^2$$

  Hence calculating it takes O(#dimensions)
- FOR EACH point of the test set , we calculate it's distance with EACH point of the train set
- Therefore the complexity is :

complexity

O(#of Train Examples $*$ #*of* test Examples $*$# of dimensions)

Solution

KD-Trees

Kd-Tree is a data structure that (amongst other things) can speed up nearest neighbor algorithm.We have developed this data structure in c++ and we have done some experiments.

about

It resembles a typical balanced binary search tree,but it is suitable for partitioning high dimensional data . We have talked about it in the lectures so I won't bore you with details. Let's get to the experiments.

Experiments

- First we have a small main program that inserts a couple of points (train set) and for some other points (test set), it searches their neighbors. This was done to make sure our code works correctly.
- Then we created a large number of synthetic 4d data and tested how long it takes to construct the tree and to classify the test set. We also measured how long it took to search for the nearest neighbor of each test point by brute force.
- We also did the same for 2d 8d ,16d and 32d data .
- For each case the data was generated such that the two classes are far away from each other. The goal is not to see how well NN works compared to another classifier such as SVM since this is not a machine learning course. We want too see how much we can speed up NN!

Experiments

- After we got NN search to work correctly, we wanted to see how fast we can make it using multiple threads.
- Testing was done with an Intel 10600K with all cores locked at 4.8 GHz, and dual-rank 4266 MHz RAM.
- We have 6 hyperthreaded cores, so scaling with 12 threads will not be as good as with 6. If we had 12 physical cores without hyperthreading, it would scale better. Plus, when using all the threads, the search has to compete with the stuff running in the background.
- Time format is minutes:seconds.fractions

4d data

Train Size	Test Size	Building tree	Brute Force NN	NN Search	6 threads	12 threads
10k	10k	0.004	1.31	0.017	0.003	0.003
20k	20k	0.010	5.92	0.038	0.007	0.005
10k	50k	0.004	6.6	0.086	0.016	0.012
100k	100k	0.053	3:04	0.24	0.06	0.043
200k	200k	0.110	13:54	0.85	0.19	0.14
100k	500k	0.053	15:18	1.26	0.28	0.18
1m	1m	0.700	-	21.4	4.20	2.53
1m	5m	0.700	-	1:48	21.2	12.7
10m	10m	11.0	-	23:54	4:18	2:33
10m	100m	11.0	-	-	-	25:33

8d data

Train Size	Test Size	Building tree	Brute Force NN	NN Search	6 threads	12 threads
10k	10k	0.004	2.26	1.53	0.34	0.24
10k	20k	0.004	4.6	3.1	0.65	0.47
20k	20k	0.01	10.2	5.77	1.17	0.83
10k	50k	0.004	11.5	7.67	1.55	1.16
20k	50k	0.01	25.6	14.5	2.98	2.13
20k	100k	0.01	51.4	29.5	6.1	4.3
100k	100k	0.056	5:07	1:19	16.5	11.4

16d data

Train Size	Test Size	Building tree	Brute Force NN	NN Search	6 threads	12 threads
10k	10k	0.004	4.12	3.36	0.68	0.56
10k	20k	0.004	8.29	6.65	1.43	1.04
20k	20k	0.01	17.3	12.3	2.5	1.81
10k	50k	0.004	20.6	16.6	3.4	2.56
20k	50k	0.01	44.2	30.5	6.4	4.58
20k	100k	0.01	1:29	1:01	12.2	9.1
100k	100k	0.06	9:16	5:13	1:05	48.9

32d data

Train Size	Test Size	Building tree	Brute Force NN	NN Search	6 threads	12 threads
10k	10k	0.004	8.0	9.02	1.82	1.43
10k	20k	0.004	16.0	18.06	3.52	2.85
20k	20k	0.010	33.4	37.76	7.5	5.9
10k	50k	0.004	40.4	45.6	9.1	7.15
20k	50k	0.010	1:22	1:34	18.8	14.24

Results

With 4 and 8 dimensions we achieve a massive speedup with a proper search algorithm, but as the number of dimensions increases, the proper search algorithm slows way down, and ends up as bad as brute force.Still ,even for 32 dimensional data we get a pretty good speed up using multithreading

How do you split P

Question

As we are building the tree, at each level we split according to a different coordinate . P1 takes all the points with split-coordinate smaller than the mean and P2 with split-coordinate larger then the mean. How do we find P1 and P2 as fast as possible ?

How do you split P ?

- At first, we tried brute force approach but it took way too long. Building the tree took longer than the simple brute force nearest neighbor algorithm.
- You can sort the array according to the desired coordinate in O(nlogn) and then split it in half.
- That being said, we don't need P1 and P2 to be sorted, since on the next level we will split by a different coordinate.
- We can use the function std::nth_element(), and it's faster than sorting. It makes sure that the point in position n should be at that position if the array was sorted.

n-th element

- Expected complexity O(n)
- 4 arguments: first element in the array P, n-th, last, and our own compare function
- comparePoints(), compares two points according to the "splitting" coordinate of each level
- All the points in positions [0,npoints/2] of P now belong in P1 and the rest in P2

Resources

- Lecture slides
- std::n-th_element(click)
- Introduction to Data Mining (Second Edition)(click)