Fast Nearest Neighbour Classification

Seminar
Intelligent Software Systems
Summer semester 2015

Lecturer: Stefan Fricke

June 2016
Supervisor: Stephan Spiegel

Gordon Lesti
Course of studies: Bachelor of Computer Science
gordon.lesti@campus.tu-berlin.de

Technische Universität Berlin
Fakultät IV Elektrotechnik und Informatik
Fachgebiet AOT
Prof. Dr. Sahin Albayrak
http://www.aot.tu-berlin.de/

Abstract. This term paper is about nearest neighbour classifiers which work with the triangle inequality of metric spaces. The Orchards Algorithm, the Annulus Method and the AESA will be explained and compared to the Full Search. Benchmarks will show the advantages and disadvantages of the three algorithms. Counting the calls of the distance function is the focus of the benchmarks.

Keywords: Nearest neighbour classifiers, Triangle inequality, Metric space

1 Introduction

Kenneth L. Clarkson wrote the paper *Nearest-Neighbor Searching and Metric Space Dimensions* [1] which deals amongst other approaches with the topic nearest neighbour search algorithms that use the triangle inequality. This paper focuses on three of them.

1.1 Notation

Table 1 explains the basic variables that are used in this paper.

Symbol	Description
$\mathbb{U}$	a set containing all items of the domain
S	a set with $S \subset \mathbb{U}$
n	the size of set S
d	a distance function on $\mathbb{U}$ with $d : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$
q	a query with $q \in \mathbb{U}$
x	a temporary variable with $x \in S$
a	nearest neighbour to q with $a \in S$ and $d(a, q) \leq d(x, q)\ \forall x \in S$
r	the minimal distance $d(a, q)$ with $r \in \mathbb{R}$

Table 1. A table of basic variables that are used in this paper.

1.2 Description of the problem

S is a subset of $\mathbb{U}$. The set S of size n and a distance function d on $\mathbb{U}$ is given. A nearest neighbour search algorithm has the mission to find the nearest neighbour $a \in S$ of a given query $q \in \mathbb{U}$, so that $d(a, q) \leq d(x, q)\ \forall x \in S$.

Example A simple real world example would be a company with many stores which are distributed over a country. This company has a small online tool to find the next store to a given location of a customer. The set $\mathbb{U}$ would be a infinite set of all locations over the whole country. The distance function d calculates the distance between two locations. The set S would contain the locations of the stores. The customer enters his current location that is represented by the query q. The tool should return the location of store a that has the smallest distance to the customers location q.

2 Algorithms

Every nearest neighbour search algorithm can be divided into two steps. The preprocessing and the query processing.

Preprocessing During the preprocessing the algorithm prepares the given set S for the nedds of the query processing. The set S can be transformed into different data structures. Those data structures are the basis for a fast calculation of the nearest neighbour during the query processing. The data structures are independent from the query q.

Query Processing The query processing is the main nearest neighbour search, the algorithm takes the query q as input and works on the preprocessed data to find the nearest neighbour a.

The following section will explain the preprocessing and query processing of the Full Search, the Orchards Algorithm, the Annulus Method and the AESA.

2.1 Full Search

The Full Search is the upper bound for the following algorithms. This algorithm has no preprocessing. The query processing will calculate the distance $d(x, q)$ for every $x \in S$. Nearest Neighbour is the the item from the set S with the smallest distance to query q.

Exactly n calls of the distance function are required to find the nearest neighbour in a given S of size n. The advantage compared to the following algorithms is, that the Full Search works for none metric spaces.

2.2 Orchards Algorithm

The Orchards Algorithm was invented by Michael T. Orchard [2]. Table 2 explains additional variables that are used in the Orchards Algorithm.

Symbol	Description
p	a temporary variable with $p \in S$
c	is a candidate for the nearest neighbour with $c \in S$
s	a temporary variable with $s \in S$

Table 2. A table of additional variables that are used in the Orchards Algorithm.

Preprocessing The preprocessing of the Orchards Algorithm has a high complexity. For every item $p \in S$, the algorithm will generate a list that contains all $x \in S \setminus \{p\}$ with their distance $d(p, x)$. Those lists will be ordered ascending to the distance. For a set S with size n, the preprocessing will call the distance function $\frac{n(n-1)}{2}$ times and will sort n lists of size $n - 1$.

Query Processing When calling the query processing with query q, the algorithm will randomly select one item $c \in S$ as initial candidate and calculate $d(c, q)$. Afterwards the algorithm walks over the ordered list of c and with every picked s from the list the algorithm calculates $d(s, q)$. Item s is the new candidate c if $d(s, q)$ is smaller than $d(c, q)$. Thereafter the algorithm walks over the list of s. The algorithm interupts if it reaches the end of a list or if $d(c, s) > 2d(c, q)$. The current candidate c is the nearest neighbour.

Orchards Algorithm with marked bits There is a improved version of the Orchards Algorithm to ensure that no distance between an item $x \in S$ and q is calculated twice. This can be achieved by boolean flags for every item during query processing of the Orchards Algorithm.

2.3 Annulus Method

The Annulus Method is also described in [1]. Table 3 explains additional variables that are used in the Annulus Method.

Symbol	Description
p^*	a pivot item with $p^* \in S$
c	is a candidate for the nearest neighbour with $c \in S$
s	a temporary variable with $s \in S$

Table 3. A table of additional variables that are used in the Annulus Method.

Preprocessing Instead of generating a list for all $x \in S$, the Annulus Method generates just one list for a random $p^* \in S$. During preprocessing the algorithm generates one list which contains all $x \in S$ with their distance $d(p^*, x)$. This list will be ordered ascending to the distance. For a set S with size n, the preprocessing will call the distance function n times and will sort one list of size n.

Query Processing The query processing of the Annulus Method starts by picking a random candidate $c \in S$ from the ordered list of p^*. Afterwards the algorithm walks alternating away from p^* and back to it in the list. If the current item s has $d(s, q) < d(c, q)$, the algorithm sets s as the new candidate c. If the current item s is under c in the list and $d(p^*, s) < d(p^*, q) - d(c, q)$, all items under s in the list will be ignored. If the current item s is above c in the list and $d(p^*, s) > d(p^*, q) + d(c, q)$, the algorithm will ignore all items above s in the list. The item c is the nearest neighbour, if the entire list is traversed.

2.4 AESA

The Approximating and Eliminating Search Algorithm, or AESA was invented by Michael T. Orchard [3]. Table 4 explains additional variables that are used in the AESA.

Symbol	Description
d_P	a distance function that works as lower bound for the real distance function d. With $d_{P\cup\{y\}}(x,q) = \max\{d_P(x,q), \lvert d(y,q) - d(x,y)\rvert\}$ and $d_\emptyset(x,q) = -\infty$
y	a temporary variable with $y \in S$
P	a set of items with $P \subset S$

Table 4. A table of additional variables that are used in the AESA.

Preprocessing The preprocessing for the AESA is starting the same way as the preprocessing of the Orchards Algorithm. The algorithm calculates the distance $d(x, y)$ of every $x \in S$ to every $y \in S \setminus \{x\}$. But instead of writing those distances into lists and sorting them, the algorithm writes those distances into a matrix to access them quickly. This matrix would be symmetric. The algorithm can use also a hash table with an unsorted pair of items as key instead of an symmetric matrix. This data structure would bisect the memory usage. For a set S with size n, the preprocessing will call the distance function $\frac{n(n-1)}{2}$ times.

Query Processing When starting with the query processing, the algorithm generates a list for all items $x \in S$ with a bounded distance $d_P(x, q) = -\infty$ and initialize $r = \infty$ as smallest distance to q. Afterwards the algorithm iterates the following steps until the list is empty.

- pick and remove x from the list with the smallest $d_p(x, q)$ or a random one during first iteration and add x to the set P
- calculate $d(x, q)$
- set $r = d(x, q)$ if $r > d(x, q)$ and remember x as current nearest neighbour
- for all y from the list, update $d_P(y, q) = max(d_P(y, q), \lvert d(x, q) - d(x, y)\rvert)$ and remove y from the list if the new $d_P(y, q) > r$

3 Benchmarks

The benchmarks will compare the calls of the distance function that the algorithm needs for preprocessing and for finding a nearest neighbour. Those benchmarks are useful when the distance function has a heavy complexity. For example the string metric Levenshtein distance or in higher dimensional vector spaces.
In general a benchmark is affected by the implementation, by the programming language, by the executing machine, by the domain of the problem and the distribution of the example. As the following benchmarks only counting the calls of the distance function its are not affected by the programming language or the executing machine.

3.1 Implementation

All presented algorithms are implemented in Java. The implementation of the algorithms and the benchmarks are open source and available on GitHub[1].

Example data Every algorithm is working on the same test data as the other algorithms. The items for the benchmarks are double floating points in $\{(x, y) \in \mathbb{R}^2 | 0 \leq x < 1, 0 \leq y < 1\}$. One thousand test sets with randomly generated points are created for every size $n \in \{100, 200, \ldots, 1900, 2000\}$. The application has calculated the average calls of the distance function for every algorithm over the one thousand sets of every size.

3.2 Result

Preprocessing The results of the preprocessing are static. Table 5 summarizing again the complexity of the preprocessing for every algorithm depending on the size of set S.

	Full Search	Orchard	Orchard MarkBits	Annulus	AESA
n	0	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	n	$\frac{n(n-1)}{2}$

Table 5. The calls of the distance function d during preprocessing depending on the size n of the test data set S.

Query Processing Table 6 and figure 1 show the results of the query processing for all algorithms.

	Full Search	Orchard	Orchard MarkBits	Annulus	AESA
100	100 (±0)	29 (±13)	20 (±8)	46 (±23)	4 (±1)
200	200 (±0)	39 (±17)	26 (±11)	82 (±46)	4 (±1)
300	300 (±0)	46 (±21)	30 (±13)	117 (±69)	4 (±1)
400	400 (±0)	53 (±25)	35 (±15)	160 (±94)	4 (±1)
500	500 (±0)	57 (±27)	40 (±17)	189 (±119)	4 (±1)
600	600 (±0)	63 (±29)	42 (±19)	228 (±144)	4 (±1)
700	700 (±0)	68 (±32)	44 (±19)	263 (±162)	4 (±1)
800	800 (±0)	73 (±33)	46 (±21)	303 (±190)	4 (±1)
900	900 (±0)	78 (±36)	50 (±22)	341 (±213)	4 (±1)
1000	1000 (±0)	79 (±37)	51 (±23)	367 (±232)	4 (±1)
1100	1100 (±0)	84 (±38)	56 (±24)	409 (±264)	4 (±1)
1200	1200 (±0)	86 (±40)	56 (±26)	442 (±277)	4 (±1)
1300	1300 (±0)	89 (±43)	59 (±27)	466 (±301)	4 (±1)
1400	1400 (±0)	90 (±43)	60 (±27)	510 (±339)	4 (±1)
1500	1500 (±0)	97 (±47)	63 (±27)	541 (±350)	4 (±1)
1600	1600 (±0)	100 (±46)	66 (±29)	573 (±374)	4 (±1)
1700	1700 (±0)	102 (±48)	67 (±29)	624 (±401)	4 (±1)
1800	1800 (±0)	104 (±49)	69 (±31)	660 (±429)	4 (±1)
1900	1900 (±0)	109 (±52)	71 (±32)	664 (±442)	4 (±1)
2000	2000 (±0)	113 (±52)	73 (±33)	740 (±487)	4 (±1)

Table 6. The average calls of the distance function d during query processing depending on the size of the test data set S.

[1] https://github.com/GordonLesti/FastNearestNeighbourClassification

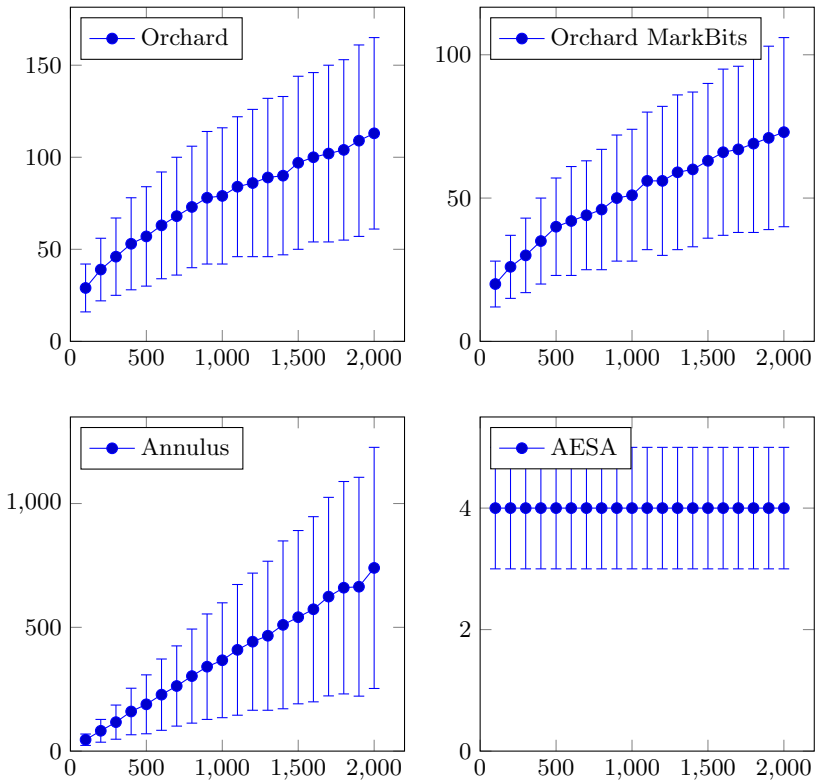

Fig. 1. The average amount of distance function calls depending on the size of set S for Orchards Algorithm with and without MarkBits, Annulus Method and AESA. The error bars are representing the standard deviation of the amount of distance function calls.

4 Conclusions

Full Search The results may leave the impression that the Full Search is not a good choice. But this algorithm remains useful if the set S changes completely with every new query or when $\mathbb{U}$ is no metric space.

Annulus Method The Annulus Method has a very complex query processing. But on a frequently changing set S a affordable preprocessing. Other algorithms with more complex preprocessing maybe hit its limits on huge sets that can be handled by the Annulus Method. The Annulus Method was the algorithm with the highest standard deviation during the benchmarks.

Orchards Algorithm with and without marking bits During query processing the Orchards Algorithm with marking bits consumed on average only 66% of the distance function calls that the algorithm needed without marking bits.

AESA Depending on the benchmarks above, the AESA should be preferred instead of the Orchards Algorithm. Both algorithms have a very complex preprocessing that may not work on very huge sets. But the AESA consumes more less calls of the distance function. The impact on real world applications should be tested. Beside the Full Search the AESA was the most stable algorithm.

5 Future work

The LAESA[4] algorithm and Metric Trees[1] are missing in the benchmarks above. These two algorithms are not parameter free and a optimal implementation for the given domain is not obvious. But both approaches are promising and their results compared to algorithms of this paper would be interesting.

References

1. Clarkson, Kenneth L. "Nearest-neighbor searching and metric space dimensions." Nearest-neighbor methods for learning and vision: theory and practice (2006): 15-59.
2. Orchard, Michael T. "A fast nearest-neighbor search algorithm." Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on. IEEE, 1991.
3. Ruiz, Enrique Vidal. "An algorithm for finding nearest neighbours in (approximately) constant average time." Pattern Recognition Letters 4.3 (1986): 145-157.
4. Mic, Mara Luisa, Jos Oncina, and Enrique Vidal. "A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements." Pattern Recognition Letters 15.1 (1994): 9-17.