Nearest neighbor search

From Wikipedia, the free encyclopedia

Nearest neighbor search (**NNS**), also known as **proximity search**, **similarity search** or **closest point search**, is an optimization problem for finding closest (or most similar) points. Closeness is typically expressed in terms of a dissimilarity function: the less similar the objects, the larger the function values. Formally, the nearest-neighbor (NN) search problem is defined as follows: given a set S of points in a space M and a query point $q \in M$, find the closest point in S to q. Donald Knuth in vol. 3 of *The Art of Computer Programming* (1973) called it the **post-office problem**, referring to an application of assigning to a residence the nearest post office. A direct generalization of this problem is a k-NN search, where we need to find the k closest points.

Most commonly M is a metric space and dissimilarity is expressed as a distance metric, which is symmetric and satisfies the triangle inequality. Even more common, M is taken to be the d-dimensional vector space where dissimilarity is measured using the Euclidean distance, Manhattan distance or other distance metric. However, the dissimilarity function can be arbitrary. One example are asymmetric Bregman divergences, for which the triangle inequality does not hold.[1]

Contents

Applications

The nearest neighbor search problem arises in numerous fields of application, including:

- Pattern recognition - in particular for optical character recognition
- Statistical classification- see k-nearest neighbor algorithm
- Computer vision
- Computational Geometry - see Closest pair of points problem
- Databases - e.g. content-based image retrieval
- Coding theory - see maximum likelihood decoding
- Data compression - see MPEG-2 standard
- Robotic sensing[2]
- Recommendation systems, e.g. see Collaborative filtering
- Internet marketing - see contextual advertising and behavioral targeting
- DNA sequencing
- Spell checking - suggesting correct spelling
- Plagiarism detection
- Contact searching algorithms in FEA
- Similarity scores for predicting career paths of professional athletes.
- Cluster analysis - assignment of a set of observations into subsets (called clusters) so that observations in the same cluster are similar in some sense, usually based on Euclidean distance
- Chemical similarity
- Sampling-Based Motion Planning

Methods

Various solutions to the NNS problem have been proposed. The quality and usefulness of the algorithms are determined by the time complexity of queries as well as the space complexity of any search data structures that must be maintained. The informal observation usually referred to as the curse of dimensionality states that there is no general-purpose exact solution for NNS in high-dimensional Euclidean space using polynomial preprocessing and polylogarithmic search time.

Linear search

The simplest solution to the NNS problem is to compute the distance from the query point to every other point in the database, keeping track of the "best so far". This algorithm, sometimes referred to as the naive approach, has a running time of $O(dN)$ where N is the cardinality of S and d is the dimensionality of M. There are no search data structures to maintain, so linear search has no space complexity beyond the storage of the database. Naive search can, on average, outperform space partitioning approaches on higher dimensional spaces.[3]

Space partitioning

Since the 1970s, branch and bound methodology has been applied to the problem. In the case of Euclidean space this approach is known as spatial index or spatial access methods. Several space-partitioning methods have been developed for solving the NNS problem. Perhaps the simplest is the k-d tree, which iteratively bisects the search space into two regions containing half of the points of the parent region. Queries are performed via traversal of the tree from the root to a leaf by evaluating the query point at each split. Depending on the distance specified in the query, neighboring branches that might contain hits may also need to be evaluated. For constant dimension query time, average complexity is $O(\log N)$ [4] in the case of randomly distributed points, worst case complexity analyses have been performed.[5] Alternatively the R-tree data structure was designed to support nearest neighbor search in dynamic context, as it has efficient algorithms for insertions and deletions such as the R* tree. [6] R-trees can yield nearest neighbors not only for Euclidean distance, but can also be used with other distances.

In case of general metric space branch and bound approach is known under the name of metric trees. Particular examples include vp-tree and BK-tree.

Using a set of points taken from a 3-dimensional space and put into a BSP tree, and given a query point taken from the same space, a possible solution to the problem of finding the nearest point-cloud point to the query point is given in the following description of an algorithm. (Strictly speaking, no such point may exist, because it may not be unique. But in practice, usually we only care about finding any one of the subset of all point-cloud points that exist at the shortest distance to a given query point.) The idea is, for each branching of the tree, guess that the closest point in the cloud resides in the half-space containing the query point. This may not be the case, but it is a good heuristic. After having recursively gone through all the trouble of solving the problem for the guessed half-space, now compare the distance returned by this result with the shortest distance from the query point to the partitioning plane. This latter distance is that between the query point and the closest possible point that could exist in the half-space not searched. If this distance is greater than that returned in the earlier result, then clearly there is no need to search the other half-space. If there is such a need, then you must go through the trouble of solving the problem for the other half space, and then compare its result to the former result, and then return the proper result. The performance of this algorithm is nearer to logarithmic time than linear time when the query point is near the cloud, because as the distance between the query point and the closest point-cloud point nears zero, the algorithm needs only perform a look-up using the query point as a key to get the correct result.

Locality sensitive hashing

Locality sensitive hashing (LSH) is a technique for grouping points in space into 'buckets' based on some distance metric operating on the points. Points that are close to each other under the chosen metric are mapped to the same bucket with high probability.[7]

Nearest neighbor search in spaces with small intrinsic dimension

The cover tree has a theoretical bound that is based on the dataset's doubling constant. The bound on search time is $O(c^{12} \log n)$ where c is the expansion constant of the dataset.

Projected radial search

In the special case where the data is a dense 3D map of geometric points, the projection geometry of the sensing technique can be used to dramatically simplify the search problem. This approach requires that the 3D data is organized by a projection to a two dimensional grid and assumes that the data is spatially smooth across neighboring grid cells with the exception of object boundaries. These assumptions are valid when dealing with 3D sensor data in applications such as surveying, robotics and stereo vision but may not hold for unorganized data in general. In practice this technique has an average search time of *O* (*1*) or *O*(*K*) for the *k*-nearest neighbor problem when applied to real world stereo vision data. [2]

Vector approximation files

In high dimensional spaces, tree indexing structures become useless because an increasing percentage of the nodes need to be examined anyway. To speed up linear search, a compressed version of the feature vectors stored in RAM is used to prefilter the datasets in a first run. The final candidates are determined in a second stage using the uncompressed data from the disk for distance calculation.[8]

Compression/clustering based search

The VA-file approach is a special case of a compression based search, where each feature component is compressed uniformly and independently. The optimal compression technique in multidimensional spaces is Vector Quantization (VQ), implemented through clustering. The database is clustered and the most "promising" clusters are retrieved. Huge gains over VA-File, tree-based indexes and sequential scan have been observed.[9][10] Also note the parallels between clustering and LSH.

Greedy walk search in small-world graphs

One possible way to solve NNS is to construct a graph $G(V,E)$, where every point $x_i \in S$ is uniquely associated with vertex $v_i \in V$. The search of the point in the set *S* closest to the query *q* takes the form of the search of vertex in the graph $G(V,E)$. One of the basic vertex search algorithms in graphs with metric objects is the greedy search algorithm. It starts from the random vertex $v_i \in V$. The algorithm computes a distance value from the query q to each vertex from the neighborhood $\{v_j : (v_i, v_j) \in E\}$ of the current vertex v_i, and then selects a vertex with the minimal distance value. If the distance value between the query and the selected vertex is smaller than the one between the query and the current element, then the algorithm moves to the selected vertex, and it becomes new current vertex. The algorithm stops when it reaches a local minimum: a vertex whose neighborhood does not contain a vertex that is closer to the query than the vertex itself. This idea was exploited in VoroNet system [11] for the plane, in RayNet system [12] for the $\mathbb{E}^n$ and for the general metric space in Metrized Small World algorithm.[13] Java and C++ implementations of the algorithm together with sources are hosted on the GitHub (Non-Metric Space Library (https://github.com/searchivarius/NonMetricSpaceLib), Metrized Small World library (http://aponom84.github.io/MetrizedSmallWorld/)).

Variants

There are numerous variants of the NNS problem and the two most well-known are the *k*-nearest neighbor search and the ε-approximate nearest neighbor search.

k-nearest neighbor

k-nearest neighbor search identifies the top *k* nearest neighbors to the query. This technique is commonly used in predictive analytics to estimate or classify a point based on the consensus of its neighbors. *k*-nearest neighbor graphs are graphs in which every point is connected to its *k* nearest neighbors.

Approximate nearest neighbor

In some applications it may be acceptable to retrieve a "good guess" of the nearest neighbor. In those cases, we can use an algorithm which doesn't guarantee to return the actual nearest neighbor in every case, in return for improved speed or memory savings. Often such an algorithm will find the nearest neighbor in a majority of cases, but this depends strongly on the dataset being queried.

Algorithms that support the approximate nearest neighbor search include locality-sensitive hashing, best bin first and balanced box-decomposition tree based search.[14]

Nearest neighbor distance ratio

Nearest neighbor distance ratio do not apply the threshold on the direct distance from the original point to the challenger neighbor but on a ratio of it depending on the distance to the previous neighbor. It is used in CBIR to retrieve pictures through a "query by example" using the similarity between local features. More generally it is involved in several matching problems.

Fixed-radius near neighbors

Fixed-radius near neighbors is the problem where one wants to efficiently find all points given in Euclidean space within a given fixed distance from a specified point. The data structure should work on a distance which is fixed however the query point is arbitrary.

All nearest neighbors

For some applications (e.g. entropy estimation), we may have *N* data-points and wish to know which is the nearest neighbor *for every one of those N points*. This could of course be achieved by running a nearest-neighbor search once for every point, but an improved strategy would be an algorithm that exploits the information redundancy between these *N* queries to produce a more efficient search. As a simple example: when we find the distance from point *X* to point *Y*, that also tells us the distance from point *Y* to point *X*, so the same calculation can be reused in two different queries.

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.