Algorithms CS 5312

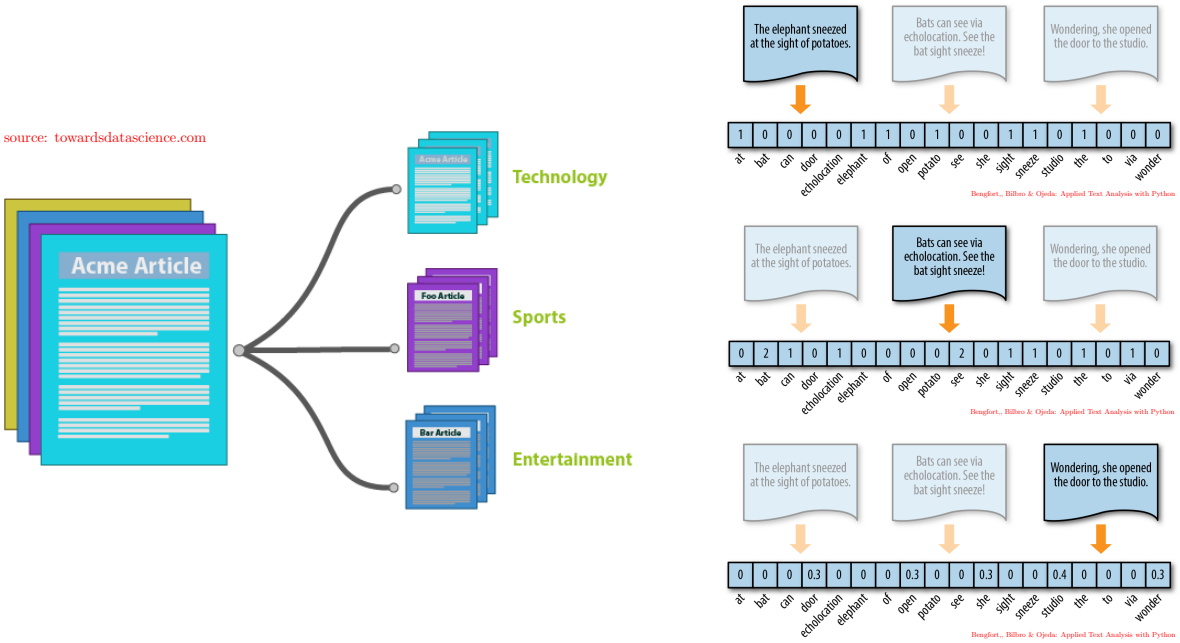
Proximity Computation on High Dimensional Data

Imdadullah Khan

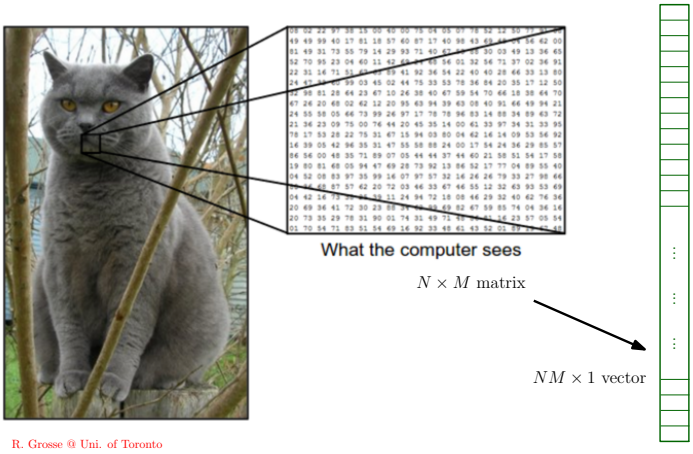
Contents

1 High Dimensional Data in Applications

High dimensionality of vectors is very common in modern datasets for a variety of application. A datasets of books, articles, or other text documents considered as a set of words, bag or words, or the TF-IDF vectors (vector-space models) have dimensionality (number of coordinates) equal to the number of dictionary words. The number of words even in very restricted settings are in thousands (unigrams). However, if we consider bigrams, then there are in almost all cases millions of them. Such a dataset is used in many applications, such as text classification, author identification, topic modeling.



Another source of high dimensional dataset is multimedia data. For instance an image is usually considered as a vector with at least one coordinate per pixel (that records pixel intensity). It's dimensionality is equal to image's resolutions. For RGB this would be 3 coordinates per pixel. The datasets of images and videos from multi-mega pixels digital cameras is a truly highly dimensional data and has various applications in image classification and clustering etc.



Similarly on an e-commerce platform such as Amazon users likes, rating or purchase history is a vector of dimensions equal to the number of items in Amazon's product space, which is in millions. Consider the rating matrix for recommendation system. Here rows typically represent users who provide rating for a few out of millions of products. Thus, each user is a data point of very high dimension. Similar each product such as movie may be rated by any of the millions of user, making each column also a very high dimensional vector.

The netflix prize training rating matrix: had $\sim 1M$ ratings of the form $\langle$ user, movie, date of grade, grade $\rangle$. The number of users (rows) were 480,189 users and number of movies (columns) were 17,770.

	p_1	p_2	p_3					p_j								p_m
u_1	1		2	1		4		2		3	2		5			2
u_2		1				2		1		2			1			3
u_3		1	1	2				1						1		2	
				3		2			5			2			3	4	
		1			2								5				
u_i			3	2	1		4	5	?	1		3	1		2		1
			4												4		
			5			1									5		
		1		4							1	3		5		1	2	
u_n			3		1	1		2	1					4			5

Network data is another potent source of high dimensional data. Consider the adjacency matrix of any modern scientific, social or communication network. Considering each vertex as a data point (a row in adjacency matrix) we get vectors with millions of dimension. Though network data is generally not processed as adjacency matrix, the dimensionality of the data nonetheless is very high. For example a row in adjacency matrix of the Facebook graph would have more than a billion coordinates [1]

2 Distance Matrix Computation and Applications

We assume our dataset X consists of n vectors $(\vec{x}_1, \ldots, \vec{x}_n)$, where each $\vec{x}_i \in \mathbb{R}^m$, i.e. each vector is a sequence of m real numbers. A distance measure d is defined over pairs of vectors, i.e. $d : X \times X \to \mathbb{R}$. This could be the ℓ_p, the cosine distance, the Jaccard distance, Hamming distance, or edit distance depending on the types of vectors and specific application.

Given the above dataset we want to compute $D = n \times n$ matrix such that $D(i, j) = d(\vec{x}_i, \vec{x}_j)$.

First we note that D matrix can be any of the above other mentioned distance metric, $d(\cdot, \cdot)$ does not have to be a distance metric. It could be a similarity measure too. This is generally referred to as the proximity matrix.

There are many applications where the proximity matrix is needed. Below we mention some applications.

2.1 Near Duplicate Detection

Near duplicates detection can be used for plagiarism detection and de-duplication. Vectors could be documents, books, papers, homework texts etc. Our goal could be to find "similar" (plagiarized) pairs of documents. One could return all pairs (i, j) of documents such that $D(i, j) < t$, for some fixed threshold t (say 10%). Another way to find documents of unusual similarity is to find all pairs of documents whose distance is say two standard-deviations below the mean distance. Again having this matrix will readily solve this problem.

The above is an abstraction of the problem in many applications. In many datasets where data is merged from different sources often there are (near) duplicate data items. These are data points who have very low distance between them. Such datasets need to be de-duplicated for efficiency and quality of analytics, as otherwise analytics could be unnecessarily biased. Examples include similar genes (with very few mutations), detecting mirror pages etc.

Detecting Fake reviews about a product on websites such as Amazon is an important problem. In many situation a single entity (usually the product marketeer or seller) write multiple reviews in order to rate up their product. In many cases these reviews are highly similar, the wordings of the texts are similar and in addition to high similarity in the metadata. Thus spotting near duplicate reviews will can identify review potentially written by the same user (hence likely fake).

2.2 News Aggregation

A near duplicate detection task is that of finding articles written by same writer on news aggregation site such as Google news. A story written by one journalist appears on many news websites with trimming to adjust spacing, added advertisements and some differences in metadata.

2.3 Input to many data analytics tasks

The pairwise proximity matrix is input in the following problems. We discussed most of these problems in this course.

- Agglomerative clustering
- Principal Component Analysis

- Spectral Clustering
- Multi-dimensional Scaling
- Kernel Method

3 Nearest Neighbor Search and Applications

Given the data set $X \subseteq \mathbb{R}^m$, with $|X| = n$, we are given a query point q in the same space as X, and we want to find the k closest points to q in X. Closeness is measured by the pre-defined proximity measure d. This problem is general referred to as the kNN search problem. The query point q may or may not be in X but it is in the same space as X, i.e. $q \in \mathbb{R}^m$.

In many setting another variant of the k-NN problem is used that is stated as follows.

Problem 1 (**Fixed radius near neighbors**). *Given a set of n points $\mathcal{X} = \{\vec{x}_1, \ldots, \vec{x}_n\}$ in $\mathbb{R}^m$ and distance measure $d : X \times X \to \mathbb{R}$. Pre process $\mathcal{X}$ into a data structure that is of size poly(n, m) such that for any query point $q \in \mathbb{R}^m$, the set $\{\vec{x}_i : d(\vec{x}_i, q) \leq r\}$ can be computed in time poly$(n, \log m)$. (The output set by the way is called the m dimensional ball with radius r centered at q).*

This variant is the same as the k-NN problem, in the sense that they are reducible to each other. We briefly sketch the reduction.

Suppose we have an algorithm A to solve the k-NN problem, we will use A to solve the fixed radius version. For $k = 1$ to n call A to get kNN of q and stop when the first time an element is returned that has distance more than r from q. We can also binary search to make fewer calls to A. i.e. Start from $k = 1$ and every time double k. The first k for which we have some neighbors that are more than r distance aways from q, we find the exact cutoff for k by doing another search between this k and $k/2$.

The other side of reduction is also very similar, in that we repeatedly call a solution to the fixed radius NN problem for increasing values of r until the algorithm returns exactly k NN in the m-d ball of radius r centered at q.

We describe some important and interesting application of nearest neighbor search problem, in order to motivate for the various approach we discuss for this problem.

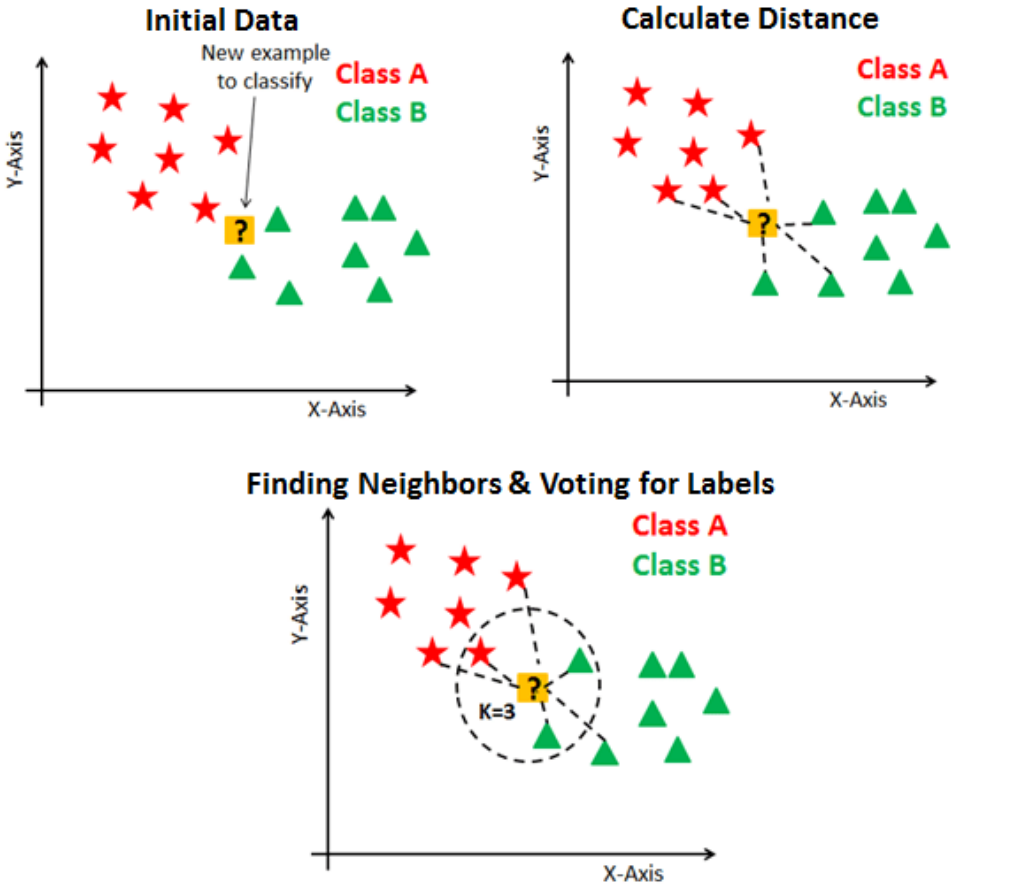
3.1 k-NN Classification

The k-nearest neighbor classifier is the simplest classifier, which classify a new instance (a data point in the testing set) to the 'class of nearest neighbors' of the instance in the training set

In other for a training instance $\vec{x}$ the set of its kNN are found and the class label of $\vec{x}$ is assigned that is the majority among the nearest neighbors (or the most frequent (mode) class).

See slides uploaded for classification and a brief description of k-NN classifier

source: https://www.jeremyjordan.me/k-nearest-neighbors/



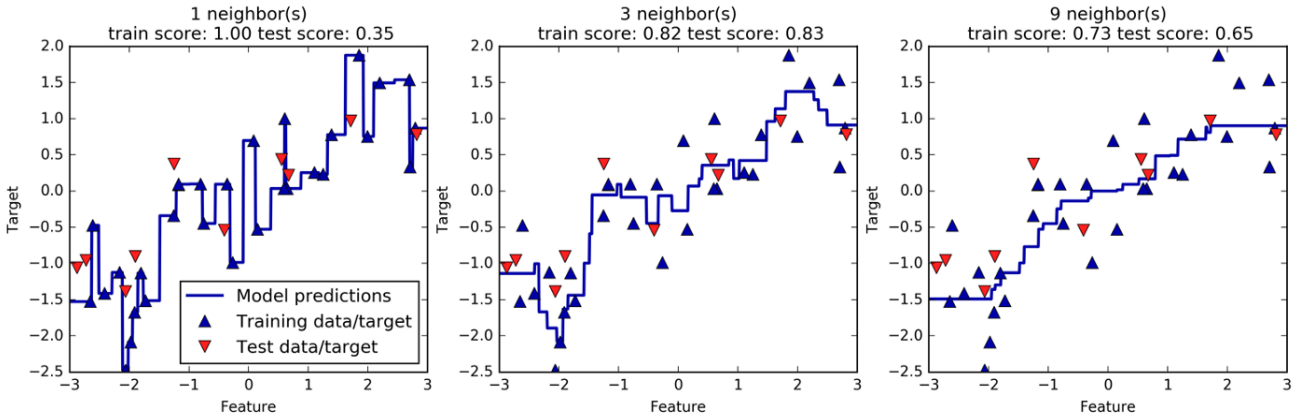
3.2 k-NN Regression

Given a dataset of n data points $(\vec{x}_i, y_i)$ for $i = 1$ to n, where $\vec{x}_i \in \mathbb{R}^m$ and $y_i \in \mathbb{R}$, i.e each vector has a value of the target variable y. In kNN regression, for a test vector $(\vec{x}, ?)$, the value of target variable $y(\vec{x})$ is predicted to be the 'average' of k-nearest neighbors of $\vec{x}$ in the train set.

The average can be weighted by the similarity of $\vec{x}$ with the nearest neighbors. Typically, in case of weighted average, we include all points (discarding k), as the farther points get a very low weight anyway.

In other words, for a test data point $(\vec{x}, ?)$, the value of target variable $y(\vec{x})$ is set as

$$y(\vec{x}) = \frac{\sum_{\vec{x}' \in X} sim(\vec{x}, \vec{x}')y(\vec{x}')}{\sum_{\vec{x}' \in X} sim(\vec{x}, \vec{x}')}$$



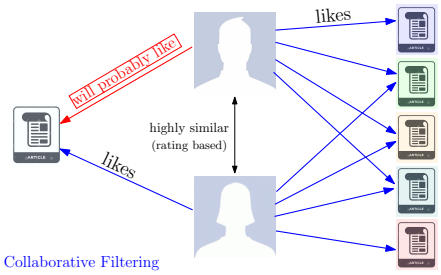
source: medium.com

3.3 Collaborative Filtering for Recommendation Systems

Recall collaborative filter for recommendation system. Where the rating of user i for an item j, $R(i, j)$ is predicted as follows.

Find the k most similar users as i (kNN of u_i) who have rated item j and output their average rating for item j. Usually some a weighted (by similarity with i) average is reported.

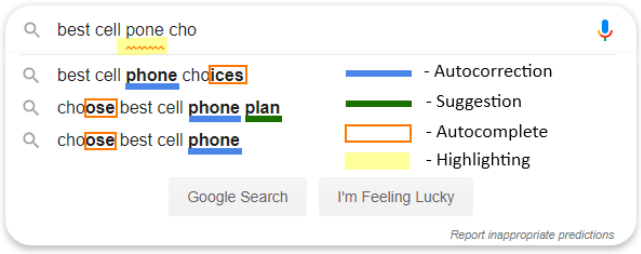
This is the user-user collaborative filtering, one can also use the so-called item-item collaborative filter, where $R(i, j)$ is predicted as the average of u_i's rating for the kNN of product j that user i has rated. See slides uploaded for Recommendation System and the method of Collaborative Filtering



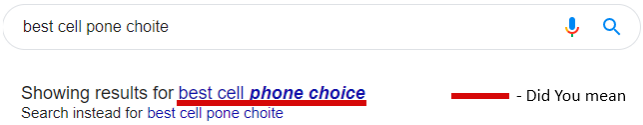
3.4 Search Engines' Autocorrect utility

Another application of kNN is the search engine Auto correct and auto complete utility (it can be and has been used in various other setting too other then search engine)

To autocorrect a user typed query one can keep a list L of commonly used query terms/phrases. When a user types a query phrase q, one can find the k most similar query phrases in L to q and suggest to the user to click on them (or if $k = 1$ automatically replace q). This has to be done in near real-time as linear searching through L may take way too much time

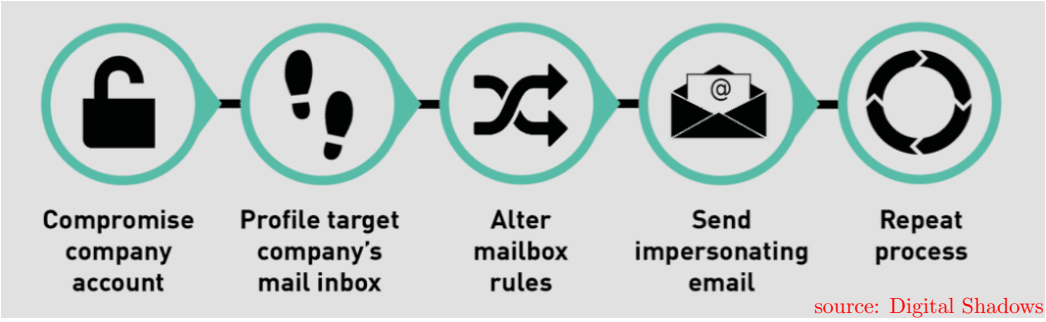


source: towardsdatascience.com



3.5 Lateral Phishing Emails

In this type of attack phishing emails are sent from a legitimate but compromised email address within an organization. Usual spam filter may fail to catch such emails as the sender domain matches that of the receiver. One can spot such email by checking if the recipient list is *very dissimilar* from usual recipients (in other emails).



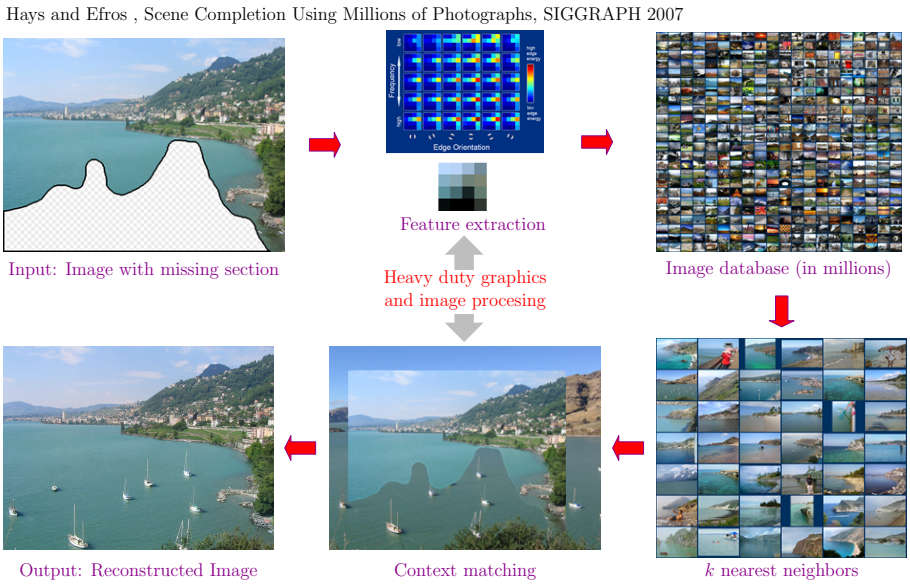
source: Digital Shadows

3.6 Image Completion, Scene completion, image or art restoration

Here a missing section of a piece of art (or image) is substituted for with a near duplicate section in some other image. See Hays and Efros, Scene Completion Using Millions of Photographs, ACM SIGGRAPH 2007 [2]

3.7 Complexities of brute-force algorithms

Given a set X of m-dim vectors, with $|X| = n$. Almost all $d(\vec{x}, \vec{y})$ measures require traversal of all coordinates of $\vec{x}$ and $\vec{y}$. The following are straight forward observation about running time of the brute force solutions to the distance matrix computation and kNN search problems, that just use the problem definition.



1. Clearly the brute force algorithm to compute the proximity matrix D is $O(n^2 \times m)$ for almost all similarity and distance measures. There are $O(n^2)$ entries in D and each entry takes $O(m)$ arithmetic operations

2. The brute force algorithm for the nearest neighbor computations take $O(n \times m)$. We need to compute the distance of q to all points in X, which takes $O(nm)$ time.

Both runtimes grow linearly with dimensionality m. Below we discuss some classic solutions for the kNN problem.

4 Approaches for k-NN

4.1 Approach 1: No Preprocessing

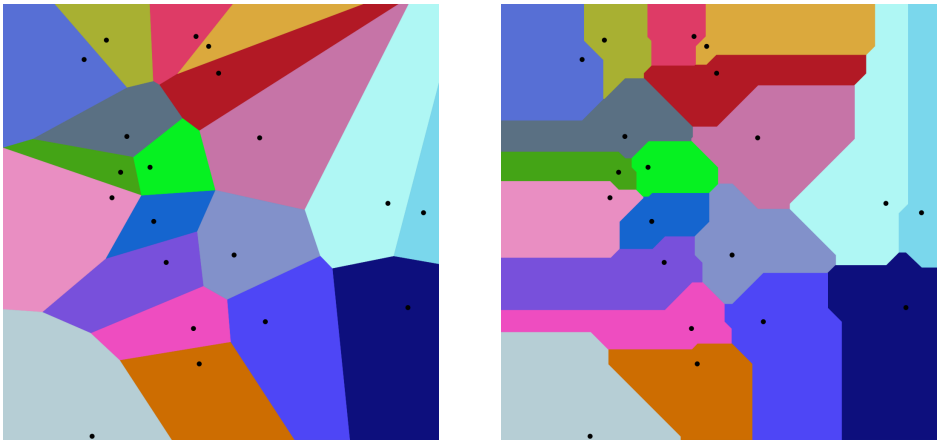
The simplest approach is to store X in a list without any preprocessing. On query. run a FINDMIN algorithm on distance to q. The runtime is $O(n)$ distance computations. Overall both the space and time complexity is $O(nm)$.

4.2 Approach 2: Sorted Array

In case of $m = 1$, X can be stored in a *sorted array*, which is the best data structure for 1-dimensional k-NN problem, as in this case, using binary search reduces the number of distance computations to $O(\log n)$. Thus, this approach has $O(n)$ space complexity and $O(\log n)$ time complexity. However, this doesn't generalize to higher dimensions.

4.3 Approach 3: Voronoi Diagram

If $m = 2$, the plane can be partitioned into regions based on which points are the nearest neighbor of a given point. Region R_i of a point $x_i \in X$ is the set of all points that are nearest neighbors of x_i, i.e. R_i is the intersection of perpendicular bisectors of x_i with all other points. For $m = 2$, the Fortune's algorithm constructs the Voronoi diagram for n pointa in $O(n \log n)$. However, this becomes extremely hard to even describe in higher dimensions.



Voronoi diagrams of 20 points under (left) Euclidean and (right) Manhattan distance. source: Wikipedia

4.4 Approach 4: *kd*-Tree

The approach using *kd*-tree data structure partitions the space into non-uniform cells. The *kd*-tree is a binary tree where each level compare 1 dimension (cutting dimension). This means that every leaf node is a k-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the space into two parts, known as half-spaces. Points to the left of this hyperplane are represented by the left subtree of that node and points to the right of the hyperplane are represented by the right subtree, as shown in Figure 1.

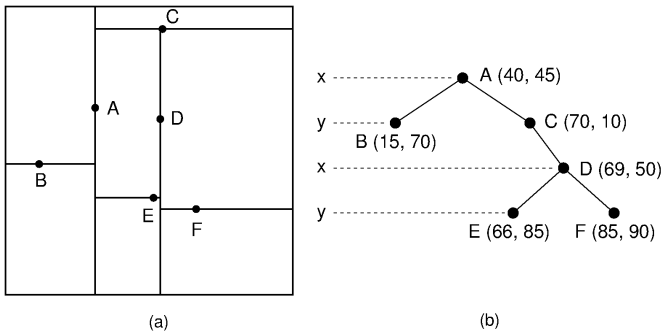


Figure 1 The *kd*-tree data structure partitions the space into non-uniform cells.

The hyperplane direction is chosen in the following way: every node in the tree is associated with one of the k dimensions, with the hyperplane perpendicular to that dimension's axis. So, for example, if for a particular split the 'd_i' axis is chosen, all points in the subtree with a smaller 'd_i' value than the node will appear in the left subtree and all points with larger 'd_i' value will be in the right subtree. In such a case, the hyperplane would be set by the 'd_i'-value of the point, and its normal would be the unit 'd_i'-axis.

The idea is to recursively construct kd-tree for the two halves, until one point remains, and this cycles through all dimensions. Figure 2 shows an example for searching for a nearest neighbor in kd-tree.

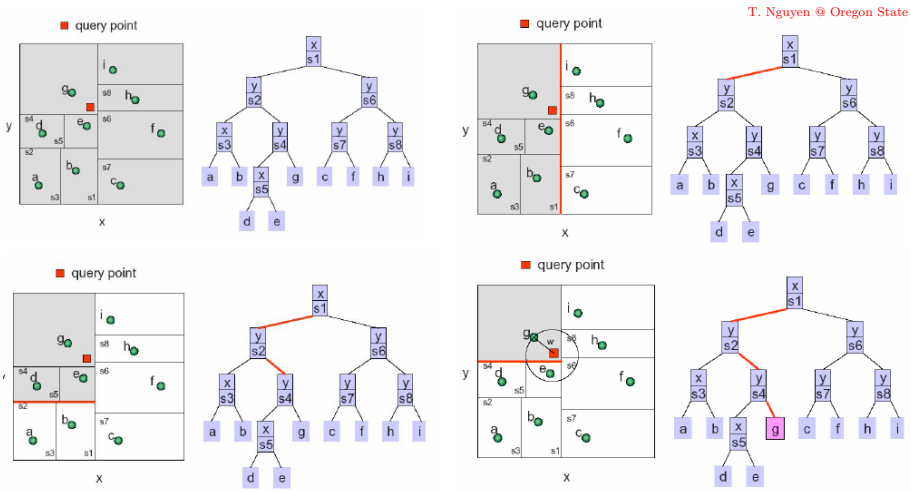


Figure 2 Searching for a nearest neighbor in kd-tree

This approach is very complicated for large m but works reasonably well for $m \leq 10$ or so.

4.5 Dimensionality Reduction and Locality Sensitive Hashing

There are two general approximation approach to solve the nearest neighbor problems. In dimensionality reduction we reduce dimensionality of the data to mitigate the impact of one factor in the complexity. Dimensionality reduction can be data dependent (Principal Component Analysis) or data oblivious (Johnson-Lindenstrauss transform). However, these dimensionality reduction methods only work for real vectors and Euclidean proximity measure.

Locality Sensitive Hashing used to tackle the second factor (number of datapoints) in the complexity of brute force nearest neighbor search. Locality Sensitive Hashing has been proposed for various data types and distance measures.

References

[1] Beg M.A., Ahmad M., Zaman A., Khan I. (2018) Scalable Approximation Algorithm for Graph Summarization. In: Advances in Knowledge Discovery and Data Min-

ing. PAKDD 2018. Lecture Notes in Computer Science, vol 10939. Springer, Cham. https://doi.org/10.1007/978-3-319-93040-4_40

[2] Hays J., and Efros A.,. 2007. Scene completion using millions of photographs. In ACM SIGGRAPH 2007 papers (SIGGRAPH '07). DOI:https://doi.org/10.1145/1275808.1276382 ...