Proximity Computation on High Dimensional Data
Curse of Dimensionality

Lecture Notes for Big Data Analytics

Nimrah Mustafa

March 2019

Contents

1 High Dimensional Data in Applications

High dimensionality of vectors is very common in modern datasets for a variety of application.

1.1 Text and Sequence Data - Vector Space Modeling

A datasets of books, articles, or other text documents considered as a set of words, bag or words, the TF-IDF vectors or other vector-space models (feature vector representation of texts) have dimensionality (number of coordinates) equal to the number of dictionary words/features. 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. Sequential data such as biological sequences (DNA or proteins), discretized audio signals [8, 4] and feature vectors extracted from EEG or EMG signals are often transformed into very high dimensional vectors. Such datasets are used in many applications, such as text classification, author identification, topic modeling, paraphrase identification, sentiment analysis, and many other tasks in NLP, signal analysis and bioinformatics. Past Electricity consumption data of customers can be considered as a sequence of readings (time-series data) and hence can be thought of as very high dimensional vectors.

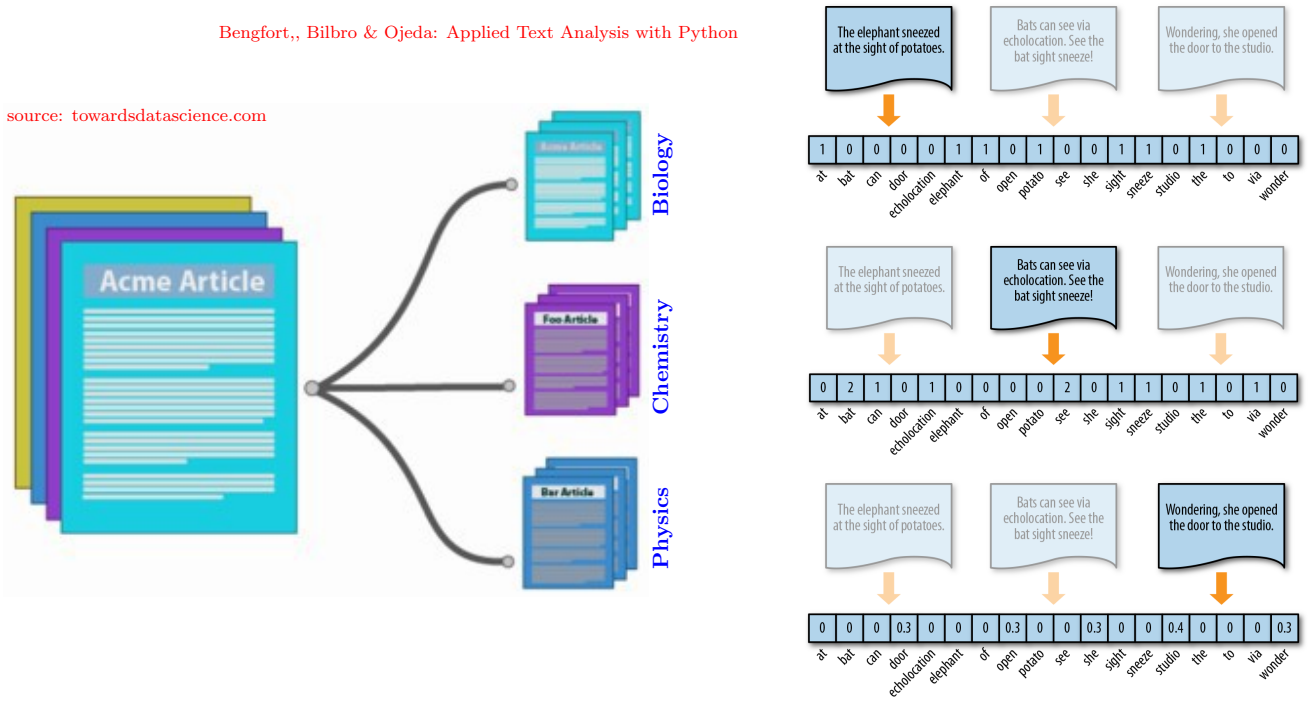


Figure 1 Vector space modeling of text documents using Set-of-Words, Bag-of-Words, and TF-IDF models for text analysis tasks

1.2 Multimedia data

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. Usually raw images are transformed into feature vectors extracted from data and these vectors are used for various image analysis tasks [2].

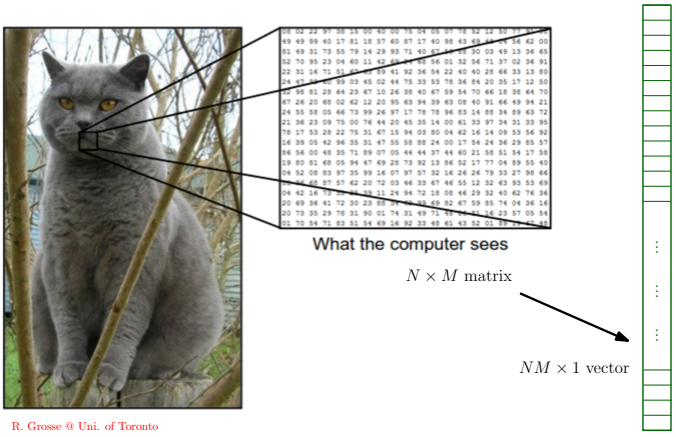


Figure 2 An RGB image converted to 1-d vector

1.3 Rating Matrices

On e-commerce platforms 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 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

Figure 3 Rating Matrix

1.4 Network Data

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. Another interesting application area requires only nodes or attributes of nodes to be represented as vectors for the tasks of node classification, graph classification or node attribute prediction [5, 9, 6, 1].

Figure 4 A social network with millions of users (vertices) and their relationship (edges)

2 Distance Matrix Computation and Applications

We assume our dataset X consists of n vectors $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, where each $\mathbf{x}_i \in \mathbb{R}^m$, 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 $D(i, j) = d(\mathbf{x}_i, \mathbf{x}_j)$. 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.

Figure 5 Fake reviews

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 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

2.3 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. The articles on other websites could be trimmed to adjust spacing, added advertisements and there could be some differences in metadata, but the article nonthenless is the same article.

Figure 6 Aggregation requires near duplicates news articles

3 Nearest Neighbor Search and Applications

Given a set X of m-dim vectors in $\mathbb{R}^m$, 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 a pre-defined proximity measure d, where $d : X \times X \to \mathbb{R}$. This problem is general referred to as the kNN search problems. 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 $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ 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 $q \in \mathbb{R}^m$ and $r \in \mathbb{R}$, the set $\{\mathbf{x}_i : d(\mathbf{x}_i, q) \leq r\}$ can be computed in time poly$(n, \log m)$. (The output set 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.

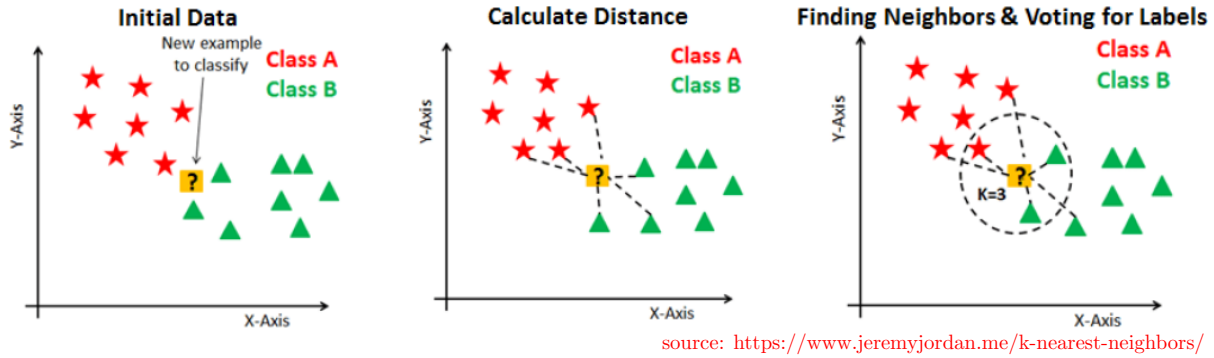
Suppose we have an algorithm A to solve the k-NN problem, we will use A to solve the fixed radius version. For $k = 1$ to n call A to get kNN of q and stop when the first time an element is returned that has distance more than r from q. We can also binary search to make fewer calls to A. i.e. Start from $k = 1$ 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 test instance to the 'class of nearest neighbors' of the instance in the training set. The class of nearest neighbors set may be the majority or the most frequent (mode) class.



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

Figure 7 Finding nearest neighbors for k-NN classification

3.2 k-NN Regression

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

The average can be weighted by the similarity of $\mathbf{x}$ 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 $(\mathbf{x}, ?)$, the value of target variable $y(\mathbf{x})$ is set as

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

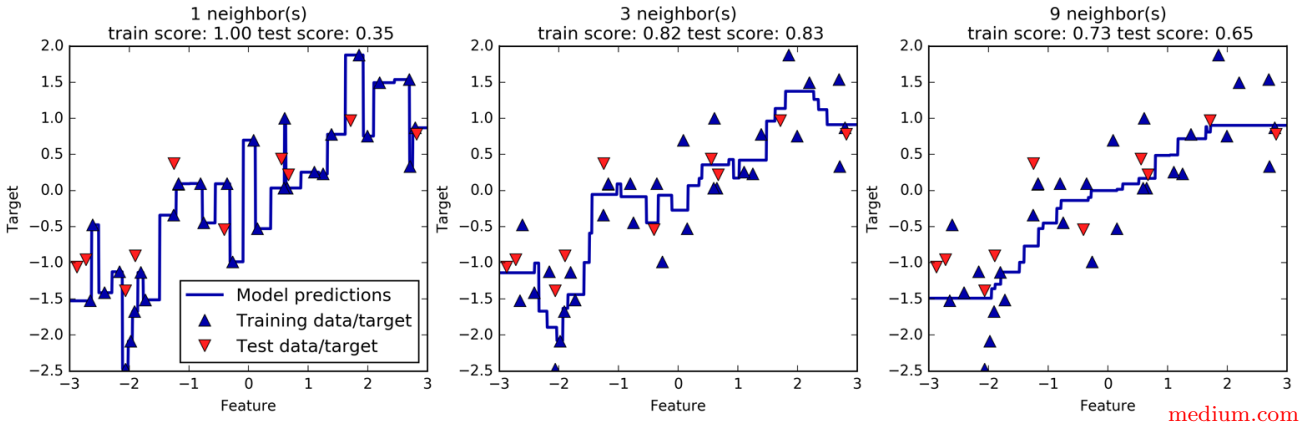


Figure 8 An example of k-NN regression

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 for Recommendation System and the method of Collaborative Filtering

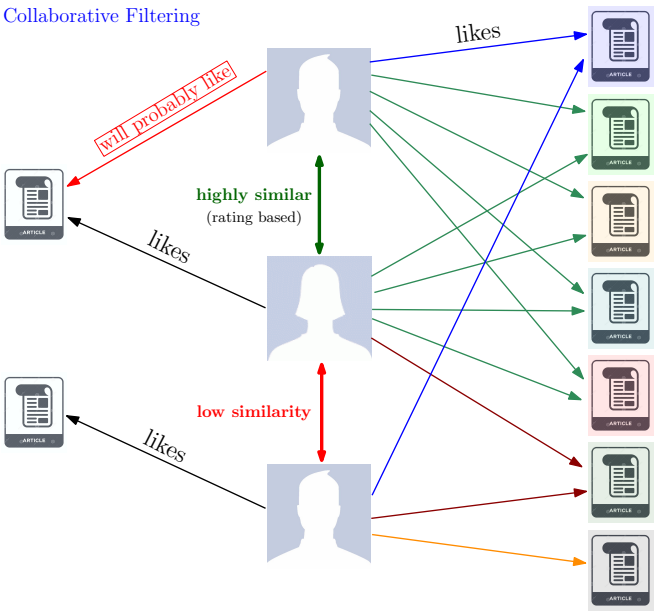


Figure 9 User-user collaborative filtering for recommendation systems

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

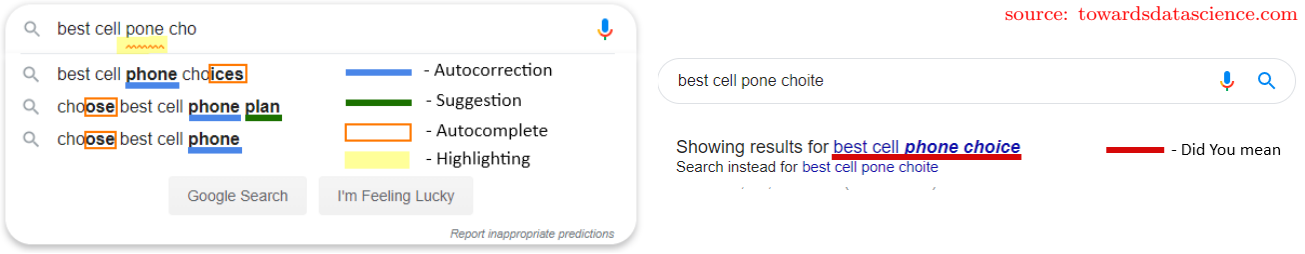


Figure 10 Auto correction, auto completion, highlighting and suggestions by search engine (Google Chrome)

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).

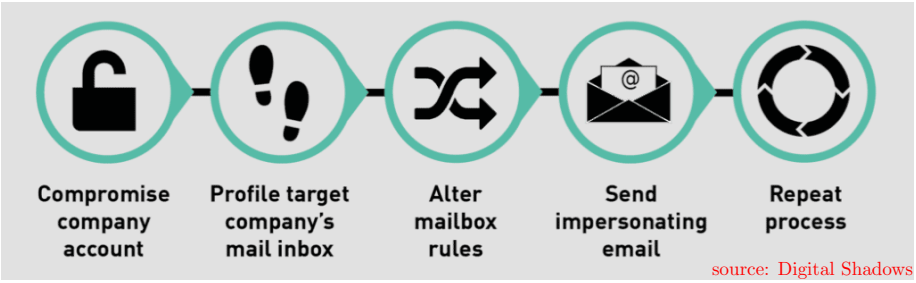


Figure 11 Dissimilarity of recipient list can figure out the phishing emails

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 [7].

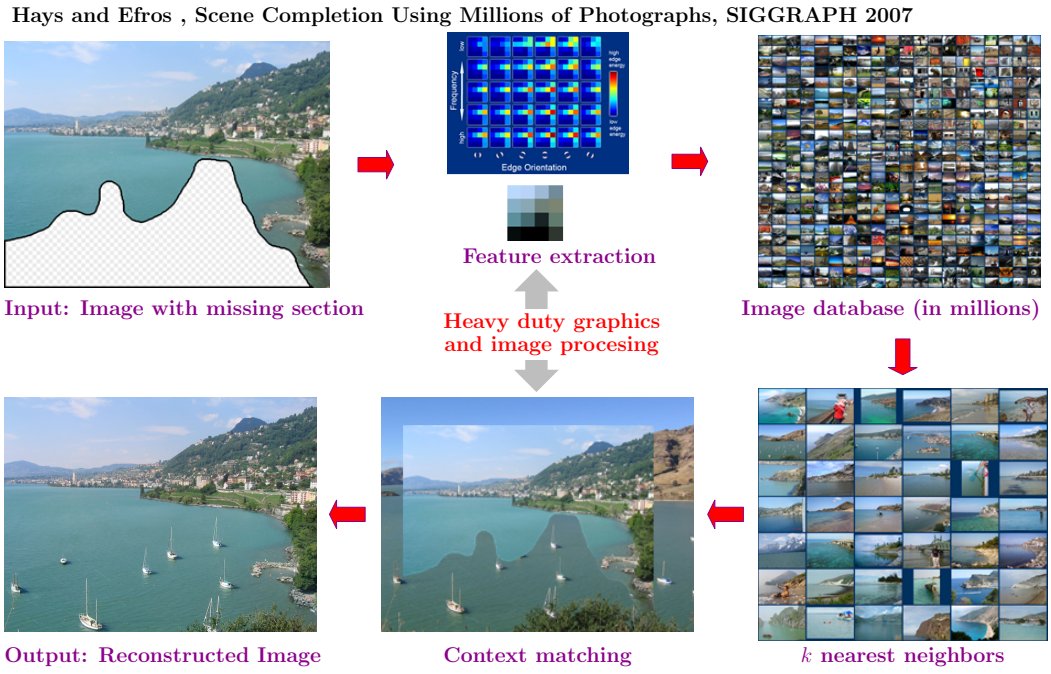


Figure 12 Completion of missing part of an image using section in near duplicate images

3.7 Complexities of brute-force algorithms

Given a set X of m-dim vectors, with $|X| = n$. Almost all $d(\mathbf{x}, \mathbf{y})$ measures require traversal of all coordinates of $\mathbf{x}$ and $\mathbf{y}$. 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

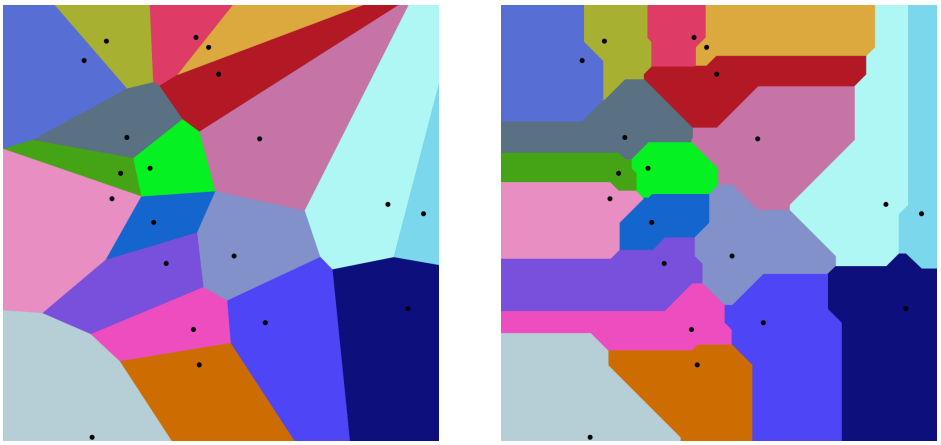
In case of $m = 1$, X can be stored in a *sorted array*, which is the best data structure for 1-dimensional k-NN problem, as in this case, using binary search reduces the number of distance computations to $O(\log n)$. Thus, this approach has $O(n)$ space complexity and $O(\log n)$ time complexity. However, 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.

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



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

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 13.

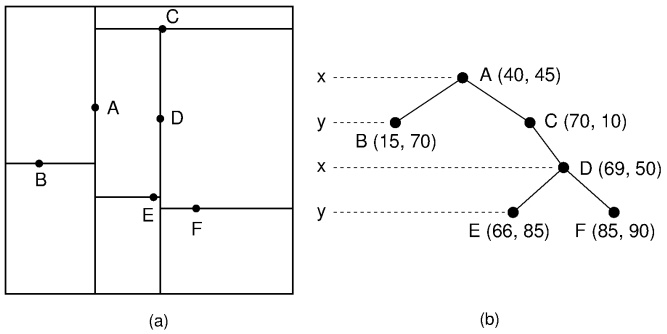


Figure 13 The kd-tree data structure partitions the space into non-uniform cells.

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 14 shows an example for 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.

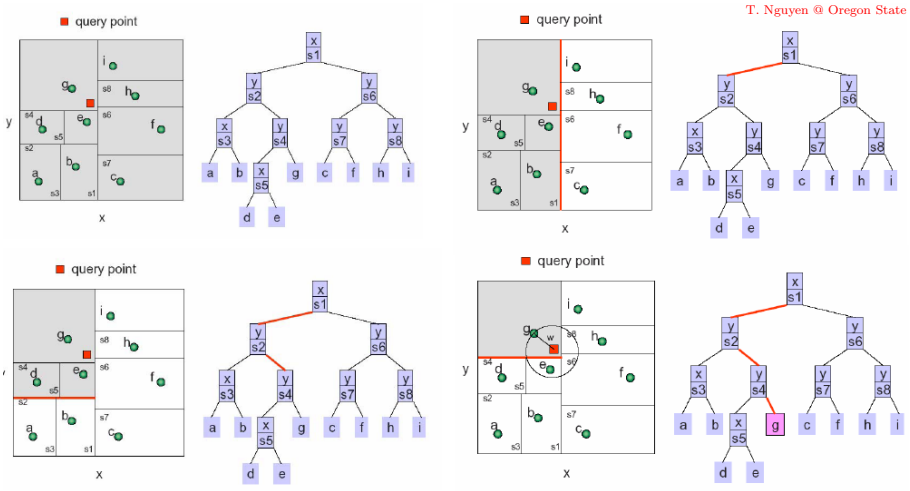


Figure 14 Searching for a nearest neighbor in *kd*-tree

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.

5 Problems with High Dimensional Data

High dimensional data refers to data with a large number of features, variables or dimensions often represented by the columns in a dataset, where each row is an instance or observation. Often, the number of features (columns) can exceed the number of instances (rows).

The term 'curse of dimensionality' coined by Richard Bellman refers to the difficulty of dynamic optimization with many variables. Broadly, the following issues are faced when working with high dimensional data:

1. Working with large dimensional data is computational challenging. Processing, Storing, Communication of high dimensional data require substantially more computational resources.

2. Large dimensional data is very hard to visualize and interpret

3. In addition, generally as number of features increases redundancy also increases - more noise is added to data than signal. This result in degradation of performance of all analytic methods. In these notes we focus on various manifestation of this issue.

The term is used to refer to different phenomena that arise in high dimensional data analytics. In particular we discuss the impact of high dimensionality of data on the two canonical proximity computation problems of distance matrix computation and nearest neighbor search. Recall we discussed that they are the building blocks of almost all data analytics.

We discuss the curse of high dimensionality in light of the distance matrix computation problem. The K-NN problem will be discussed in detail later.

We now discuss the issues that arise in the distance matrix computation with high dimensional data one by one.

6 Computational Complexity

Given a set X of m-dim vectors in $\mathbb{R}^m$, with $|X| = n$. A straight forward observation about running time of the brute force solution to compute the distance matrix D is $O(n^2 \times m)$, since for almost all distance measures computing $d(i, j)$ requires traversal of all coordinates of i and j. Note that this run-time grows linearly with dimensionality of data.

7 Data Sparsity

As the dimensionality of data increases, the relative input space covered by a fixed-size training set diminishes. Many methods require a sizeable number of examples/samples in every region of the space to support a hypothesis or train a generalizable model. Since the available data becomes very sparse because the volume of the space has increased so much, such methods that need statistical significance fail. For example, in machine learning a certain number of training

data items is required to give meaning to a model (e.g. a classifier), which is not there in such a sparse space. Similarly, in statistical analysis a certain sample size is required to support a hypothesis, which is not available.

In class we discussed that suppose we have data for 1000 students performance (discretized scores of $0, 25, 50, 75, 100)\%$ in 2 courses c_1 and c_2. Then in total there are $5 \times 5 = 25$ different grade combinations. If the 1000 students are randomly distributed among each grade combination, then on average there are 40 students with each possible grade combination, which is a good enough sample to draw conclusions such as if, for a student, $grade(c_1) \leq 50$ and $grade(c_2) \geq 75$, then that student is likely to be a Math major. Now suppose there are 4 courses, then the number of possible grades combination is $5^4 = 625$, and an average number of students per combination is 1.6. For 10 courses, this number reduces to 0.0001024. This means that almost all possible combinations are never observed.

8 Huge Search Space for Nearest Neighbor Search

A basic approach for nearest neighbor search is use a data structure that partition the input space into cells (grids or mesh). Having preprocessed the data (i.e. storing the dataset X in this grid structure). To find nearest neighbor(s) of a query point one locates the cell containing q. And return all points in X in that the cell and perhaps those in the 'neighboring' cells.

The search space for nearest neighbors in this data structure grow exponentially with the dimensionality of the data. i.e. The number of 'neighboring' cells to search for in 2-d is $3^2 = 9$, that in 3-d is 3^3, which in m-d the number of cells one need to examine is 3^m

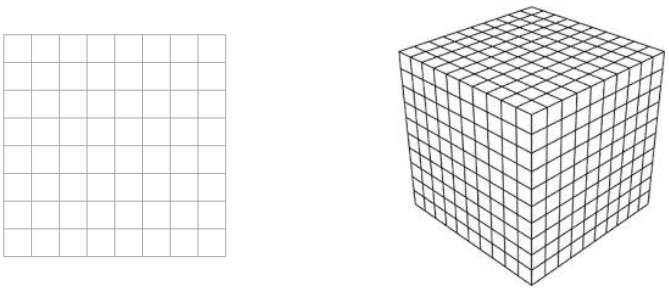


Figure 15 Partition the space into cells (grids or mesh) for large dimensions

Recall the approaches for nearest neighbor search. The grid (or lattice or mesh) could be non-uniform see for instance kd-tree, one of the most widely used data structures for nearest neighbor search, which works quite well when the dimension $n \leq 10$.

We demonstrate this phenomenon that higher dimensional neighborhood is very large and not local. i.e. the notion of nearest neighbor breaks down as follows.

Suppose n points in X are chosen uniformly at random from $[0, 1]^m$ (m-cube). For the query point q grow a hypercube around q to contain f fraction of points ($k = fn$) in X. We show that this cube (the search space for q) grows very large (covering almost the whole input space) in large dimension. The Expected length of the edge of the search cube $E_m(f) = (f^{1/m}$. i.e. in $10d$

to get 10% points around q need cube with edge length 0.8 (which is 80% of the whole cube, the input space). Similarly, to get only 1% points one needs to extend the search cube by 0.63 units along each dimension

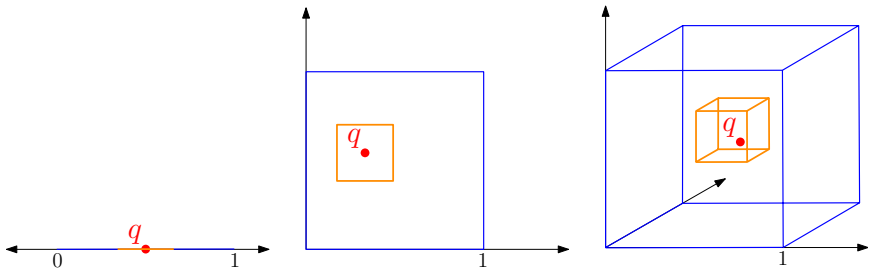


Figure 16 1-d (left), 2-d (middle) and 3-d (right) search spaces example

We give a concrete example to demonstrate this phenomenon of non-locality of higher dimensional neighborhoods

Suppose 5000 points are randomly placed in $[0,1]^m$. Let $q = \mathbf{0}$

- In 1d we must go a distance of $5/5000 = 0.001$ on average to capture 5 NN (i.e to to capture 5 of those random input points for a typical q, we need to grow a search cube to capture only .1% of the input cube)
- In 2d, on average we must go a distance $\sqrt{5/5000} = 0.031$ units along both dimensions to get 5 nearest neighbors points (about 3% of the whole cube)
- In 3d, on average we must go $\sqrt[3]{0.001} = 0.1 = 10\%$ of the total (unit) length in each of the 3 dimensions
- In 4d, we must go $\sqrt[4]{0.001} = 0.177 = 17.7\%$ of unit length
- In 10d, we must go 50.1% of unit length along each dimension
- In md, we must go $(5/5000)^{1/m}$ along each dimension

To express this phenomenon the phrase "in high dimensional space nobody can hear you scream" is used.

9 Diminishing Volume of m-ball

A manifestation of this phenomenon that points in higher dimensions are isolated is the diminishing relative volume of the m-ball in m-cube

There is another way one can look at this issue with high dimensional space that is a big hurdle for the fixed radius nearest neighbors search problem. The problem is that in very large dimensions, this ball (the output of the problem) is essentially empty. This is sometimes referred

to as in high dimensions, every point is an outlier. This is also referred to as high dimensional space is lonely.

The m-ball (m-d hypersphere) of radius r centered at origin is defined as

$$B_{m,r} := \{\mathbf{x} \in \mathbb{R}^m : d(\mathbf{x}, \mathbf{0} \leq r) \implies \|\mathbf{x}\|_2 \leq r\}.$$

The m-dimensional volume of m-ball of radius r in m-dimensional Euclidean space is:

$$\text{Volume of } B_{m,r}: \qquad V_m(r) = \frac{\pi^{m/2}}{\Gamma(m/2+1)}\, r^m$$

$\Gamma(\cdot)$ essentially is factorial of fractional numbers

For our purposes

$$V_m(r) = \frac{\pi^{m/2}}{m/2!} r^m \qquad \text{For simplicity assume } m \text{ is even}$$

The m-cube (m-d hypercube) is the set $[-1,1]^m$ (note edge length is 2)

$$\text{Volume of } m\text{-cube:} \qquad 2^m$$

In m-d ratio of volume of unit m-Ball to that of m-cube (edge length 2) is

$$\frac{\pi^{m/2}/m/2!}{2^m} \text{ approaches 0 very fast}$$

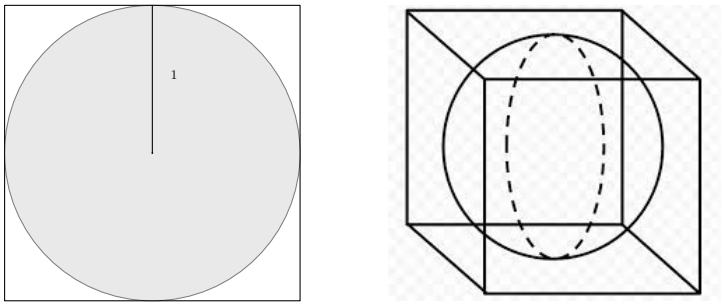


Figure 17 In 2-d (left) ratio of volume of unit m-ball is higher than that of 3-d (right) to that of unit m-cube, and approaches to 0 very fast in higher dimensions

Observe the ratio in the following table.

Ratio of volumes of unit m-Ball and $[-1,1]^m$

$$\frac{\pi^{m/2}/m/2!}{2^m}$$

dim m	volume of m-ball	volume of m-cube	ratio
2	π	2^2	~ 0.785
3	$4/3\pi$	2^3	~ 0.523
4	$\pi^2/2$	2^4	~ 0.308
6	$\pi^3/6$	2^6	~ 0.080
m	$\frac{\pi^{m/2}}{m/2!}$	2^m	$\to 0$

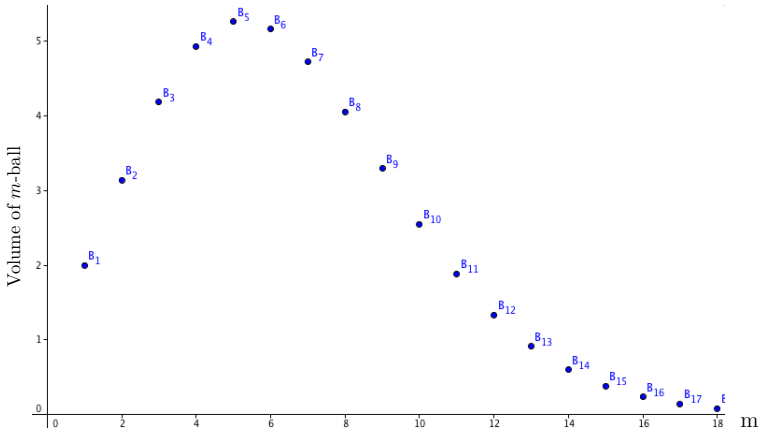


Figure 18 Volume of m-ball (denoted by B_m) increases from $m = 1$ to $m = 5$, afterwards it sharply declines and get close to 0 for $m > 20$.

Let say we select a point x at random in the $[-1, 1]^n$ (cube centered at the origin) cube, what is the probability that x is the unit n ball (inscribed). This probability is exactly equal to the ratio of the volume of the unit ball to the volume of the cube.

Hence the volume of the unit sphere compared to the cube is diminishing. This means that in very high dimensions, if we are doing a nearest neighbor search, (say the fixed radius version), it is empty, almost no point is within distance r. In other words for a fixed query point q, if we choose n points at random in $\mathbb{R}^m$ almost no point will be within r distance to q, equivalently almost all will be at distance more than r. So nearest neighbor (or distance for that matter) lose all its effectiveness.

- In higher dimensions all the volume is in 'corners'
- Points in high dimensional spaces are isolated (empty surrounding)
- The probability that a randomly generated point is within r radius of q approaches 0 as dimensionality increases
- The probability of a close nearest neighbor in a data set is very small

There is a caveat that we must keep in mind for this and the following issues that the real datasets are not random.

However if a dataset exhibit this phenomenon that the issue has be overcome by getting a larger training set (exponential in m). One way to look at this is as follows.

To cover $[-1,1]^m$ with $B_{m,1}$'s, the number of balls n must be

$$n \geq \frac{2^m}{V_m(1)} = \frac{2^m}{\pi^{m/2}/m/2!} = \frac{m/2!\, 2^m}{\pi^{m/2}} \overset{m\to\infty}{\sim} \sqrt{m\pi}\left(\frac{m2^{m/2}}{2\pi e}\right)^{m/2}$$

For $m = 16$ (a very small number) this n is substantially larger than 2^{58}

10 Instability of Nearest Neighbor

A qualitative problem in higher dimensional space resulting from the phenomenon of empty neighborhood is that the notion of nearest neighbor breaks down. This means that there is substantial difference or contrast between nearest and farthest neighbors of a point q.

A **nearest neighbor query is ϵ-unstable** ($\epsilon > 0$), if the distance from q and most other points are at most $(1+\epsilon)$ times the distance from q to its 1NN.

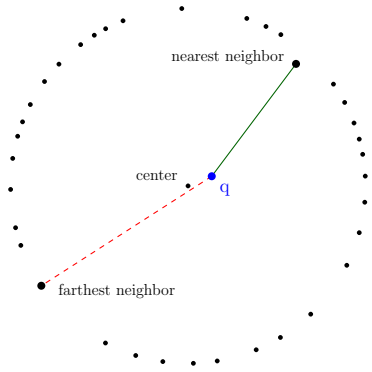


Figure 19 No difference (contrast) between nearest and farthest neighbors in higher dimensions

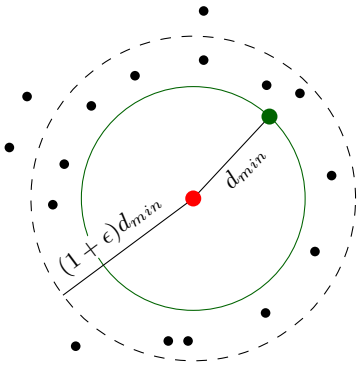


Figure 20 Higher dimension causes ϵ-unstable ($\epsilon > 0$) nearest neighbor query

We show that as dimensionality increases the probability of all nearest neighbors queries becoming unstable increase (by showing the distance concentration phenomenon)

11 Distance Concentration

Another facet of the curse of dimensionality is the phenomenon of distance concentration.

Assume points in $\mathbb{R}^m$ and ℓ_2 distance measure. As m increases, almost all pairs of points have their ℓ_2 distances (i) similar to distance of other pairs and (ii) very high. Consequently, the distance measure loses its meaning, and since proximity measures is the building block of data analytics, as we discussed earlier, when it becomes meaningless the building collapses. For

example, in the K-NN problem, if all distances are similar and high, the nearest neighbor is as good as the farthest neighbor and it becomes difficult to build clusters since there is no justification to group a pair of points and not another.

It can be said that the normalized distance is close to 1, so both factors that distances are high and similar are encompassed. We demonstrate it by observing distribution of pairwise distances for n points in $\mathbb{R}^m$.

11.1 Analytical Bounds

Suppose we generate a set $\mathcal{X}$ of random vectors in m-dimensional unit cube with $|\mathcal{X}| = n$, i.e. n points in $[0,1]^m$. Let $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_m)$ be two such vectors. The Euclidean distance ℓ_2 between x and y is given by

$$d(x,y) = \|x - y\|_2 = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}.$$

The maximum possible distance b/w a pair $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ is $d(\mathbf{x}, \mathbf{y}) \leq \sqrt{m}$. For simplicity we consider the squared distance (to get rid of square root) i.e.

$$d^2(x,y) = \|x - y\|_2^2 = \sum_{i=1}^{m} (x_i - y_i)^2.$$

Thus,

$$d^2(\mathbf{x}, \mathbf{y}) := \|\mathbf{x} - \mathbf{y}\|^2 \leq m$$

For a fixed coordinate $i < m$, We have that

$$Pr(|x_i - y_i| \geq 1/4) > 1/2. \tag{1}$$

To see this, note that $Pr(|x_i - y_i| \geq 1/4) = Pr(\{x_i - y_i \geq 1/4\} \cup \{v_i - u_i \geq 1/4\})$. Since the joint distribution of x_i and y_i is uniform on $[0,1]^2$, this event correspond to area of the lower right triangle and upper left triangle (below the line $y_i = x_i - 1/4$ and above the line $y_i = x_i + 1/4$, the shaded regions in Figure 21). These two triangles stacked together, is a square of length $3/4$, and its area (and hence $Pr(|u_i - v_i| \geq 1/4)$) is $9/16$ which is greater then $1/2$.

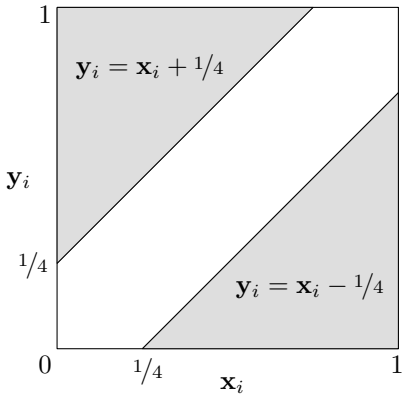


Figure 21

Using this we will argue that when dimensions are large then almost all distances are large.

The following is a very useful and elegant trick, we will use it perhaps many timed. It is imperative

that you master it, so think about it and keep thinking until it is clear (without worrying about technicalities like exact calculations etc.).

For fixed $x, y \in \mathbb{R}^m$ chosen as above, let V_i be the indicator random variable for the event that $|x_i - y_i| \geq 1/4$. i.e. if coordinate difference is big.

$$V_i = \begin{cases} 1 & \text{if } |x_i - y_i| \geq 1/4 \\ 0 & \text{otherwise} \end{cases}$$

We know that $E[V_i] = Pr(V_i = 1) > 1/2$ by Equation (1). Let $V = \sum_{i=1}^{m} V_i = \left|\left\{i : |\mathbf{x}_i - \mathbf{y}_i| \geq 1/4\right\}\right|$, i.e. V is the counter to see how many coordinates have big difference. By linearity of expectation, we know that $E(V) = m/2$, which means that on average $m/2$ summands of the d^2 sum will be at least $1/16$.

In order to show concentration around this mean, which implies that with very high probability the distance is large for this pair, we will use Chernoff bound.

Theorem 1 (Chernoff Bound (tail inequality))**.** *Let $X_1, X_2, \ldots, X_n$ be independent Bernoulli random variables. Let $S = X_1 + X_2 + \ldots + X_n$, and let $E(S) = \mu$. The loose Chernoff bounds are stated as follows:*

- $Pr(S \geq (1+\delta)\mu) \leq e^{\frac{-\delta^2\mu}{3}}$, *for* $0 < \delta < 1$
- $Pr(S \geq (1+\delta)\mu) \leq e^{\frac{-\delta\mu}{3}}$, *for* $\delta > 1$
- $Pr(S \leq (1-\delta)\mu) \leq e^{\frac{-\delta^2\mu}{2}}$, *for* $0 < \delta < 1$

Using the Chernoff Bound with $\delta = 1/2$:

$$Pr(V < m/4) = Pr(V < (1 - 1/2)m/2) < e^{-m/16}$$

So almost surely, at least $m/4$ coordinate differences are at least $1/4$, and hence squared distance is at least $m/64$. So, the probability that a given pair is far, i.e. its squared distance is more than $m/64$ is at least $(1 - e^{-m/16})$.

$$Pr\left[V \geq \frac{m}{4}\right] \geq 1 - e^{-\frac{m}{16}}$$

In other words, for fixed $\mathbf{x}, \mathbf{y}$: $Pr\left[\|\mathbf{x} - \mathbf{y}\|^2 \geq \frac{m}{64}\right] \geq 1 - e^{-\frac{m}{16}}$

Now we want to find what is the probability that all $\binom{n}{2}$ pairs are far. This probability is equal to 1 minus the probability that some pair is close (squared distance $m/64$). This latter probability is less than the sum of probabilities of individual pair closeness (union bound).

$$Pr\left(\text{all } \binom{n}{2} \text{ pairs are far}\right) \geq 1 - \sum_{\text{pairs}} e^{-m/16} = 1 - \binom{n}{2} e^{-m/16}$$

Solving the above inequality such that the probability is at least $1/2$, then as long as $m = \Omega(\log n)$, then with high probability for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$, we have $d^2(\mathbf{x}, \mathbf{y}) \geq \frac{m}{64}$. This means all pairs are far, i.e. $d(\mathbf{x}, \mathbf{y}) \geq \sqrt{m}/8$.

Here is the result of simulation of this distance concentration phenomenon. In your homework you were supposed to observe this for varying parameters.

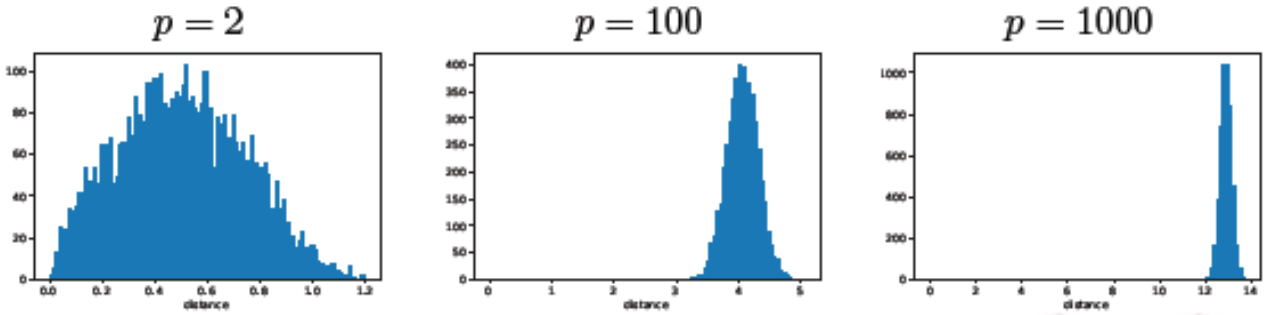


Figure: Histograms of pairwise-distances between $n = 100$ points sampled uniformly in the hypercube $[0, 1]^p$

12 Angle Concentration

In large dimensions (at least for random points) the distance measure (at least ℓ_2 distance) is more or less meaningless. Then, can we use cosine distance? We will show that the same concentration phenomenon is observed for pairwise angles.

A pair of vectors $\mathbf{x}, \mathbf{y}$ is orthogonal if $\mathbf{x} \cdot \mathbf{y} = 0$, $\theta_{x,y} = 90°$. The maximum number of such pairwise orthogonal vectors in $\mathbb{R}^2$ is 2 and in $\mathbb{R}^3$ is 3. In $\mathbb{R}^m$, the maximum number of pairwise almost orthogonal vectors ($\mathbf{x} \cdot \mathbf{y} \leq \epsilon$, $\theta_{x,y} = 90° \pm \epsilon$) is $e^{\Omega(m)}$.

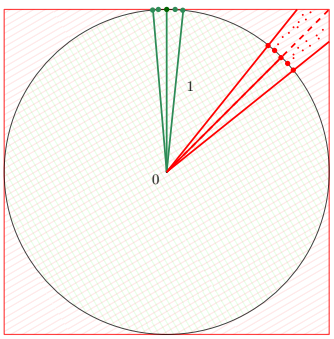
12.1 Generating Random Direction in $\mathbb{R}^m$

Choosing a random direction (equivalently, a random unit vector) is not a straight forward task. We think of this as choosing random unit vectors (normalized) in $\mathbb{R}^m$. W will also need this later for dimensionality reduction and also for Random Hyperplanes based LSH for Cosine distance and Random Projections based LSH for Euclidean distance.

An immediate way of picking a random vector in $\mathbb{R}^m$ is to choose a random point in $\mathbf{v} \in [-1, 1]^m$ and normalize it as $\hat{\mathbf{v}} = \mathbf{v}/\|\mathbf{v}\|$. For example, in $2D$ ($\mathbb{R}^2$), generate a point in the unit square and then normalize it. However, this doesn't produce all directions uniformly at random, i.e. let $v = (v_1, v_2)$ where v_1 and v_2 are randomly chosen from $[0, 1]$. This picks a random vector in the unit square (or cube as we saw earlier), we can normalize v as $\hat{v} = \frac{v}{||v||_2} = \frac{v}{\sqrt{v_1^2+v_2^2}}$.

As you can see in Figure 22, the distribution of randomly generated direction is skewed towards the diagonal directions.

A quick fix to this method is the algorithm of George Marsaglia and Arif Zaman, i.e. to generate a random



The red points have significantly high probability of begin chosen compared to the green points

vector in the unit cube, but if the vector is outside the unit ball, then discard it, (i.e. if $v_1^2+v_2^2+\ldots v_m^2 > 1$, then discard it), so we only consider points within the unit ball. Then, when normalized, each point on the surface of the unit ball will be equally likely. However, this can be computationally expensive. Recall the problem of the diminishing volume of the m-ball, which implies that all points points lie outside the unit m-ball and thus will have to be discarded.

In $2D$, an easier way is to get a random direction is to generate a random number between $[0, 2\pi]$ (randomly choose an angle) and use standard trigonometric calculations to create a unit vector. The former method is computationally more expensive while the latter only works for $2D$.

The correct way to generate random directions (or random unit vectors equivalent to generating random points on the surface of the unit m-ball) is to use the spherical symmetry of the standard normal distribution. We generate m dimensional vector $v = (v_1, \ldots, v_m)$ by picking each of v_i independently from the standard normal distribution $\mathcal{N}(0, 1)$ and then normalize v by $\|v\|_2$ as earlier. Such a random vector in $\mathbb{R}^m$ is known to be uniformly distributed over the surface of unit ball B_m, as can be seen in Figure 23.

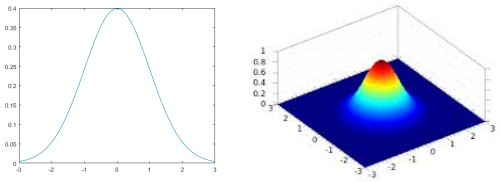


Figure 23 Generating random points on the surface of the unit m-ball

An approximate method to generate a random direction in $\mathbb{R}^m$ is to select the vector $v = (v_1, \ldots, v_m)$ by choosing $v_i \in \{-1, 1\}$ independently and equally likely. v is then normalized by $\|v\|_2 = \sqrt{m}$. This generate directions towards corners of the m-cubes $[-1, 1]^m$ as shown in Figure 24.

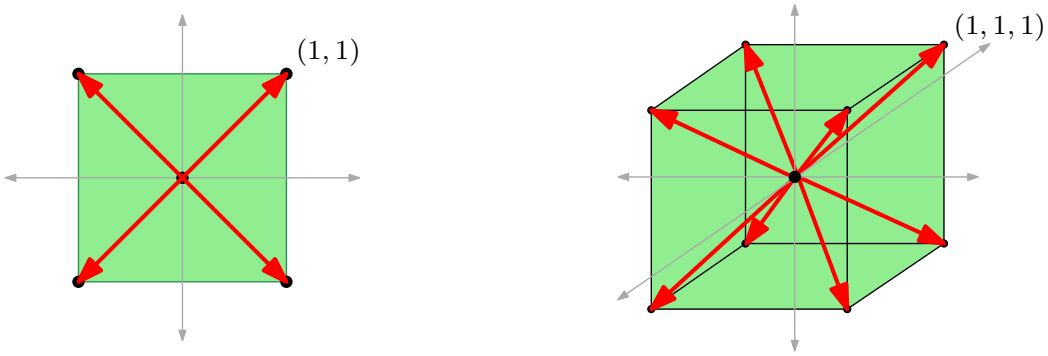


Figure 24 Generate directions towards corners of the m-cubes

The approximation is quite good in large dimensions and we will use it extensively in dimensionality reduction. In fact, we will discuss both of these techniques during the dimensionality reduction module. In order to see that almost all vectors are almost orthogonal in high dimen-

sions, we will use analytical bounds to show that in expectation, every pair of random high dimensional vectors is orthogonal [3].

12.2 Analytical Bounds

Similar to distance concentration analytical bounds, Suppose we generate a set $\mathcal{X}$ of random vectors in m-dimensional unit cube with $|\mathcal{X}| = n$, i.e. n points in $[0,1]^m$. Suppose these are random points on the surface of B_m, i.e. they are just directions chosen uniformly at randomly from $\{-1,1\}^m$ and normalized as discussed above.

Recall that unit vectors x and y are almost orthogonal mean that $\theta_{x,y}$ the angle between them is about 90°, or $\cos\theta_{x,y} = \langle x, y\rangle = x \cdot y = \sum_{i=1}^{m} \sim 0$ (as the denominator is 1).

For a fixed $\mathbf{x}$ and a random unit vector (chosen as in previous section) $\mathbf{y}$ in $\mathbb{R}^m$, let $V_i = \mathbf{x}_i\mathbf{y}_i$ and let $V = \sum_{i=1}^{m} V_i = \cos\theta_{\mathbf{x},\mathbf{y}}$

Lemma 2. *$E[V_i] = 0$ and $\dfrac{-\mathbf{x}_i}{m} \le V_i \le \dfrac{\mathbf{x}_i}{m}$.*

This implies that on average, $\mathbf{x}$ is orthogonal to any $\mathbf{y}$. To bound the probability of the deviation of V from its mean, we use the Hoefding's inequality.

Theorem 3 (Hoeffding's inequality)**.** *Let X_i be random variables bounded by the interval $[a_i, b_i]$ and let $S = \sum_{i=1}^{n} X_i$. Then*

$$Pr(|S - E[S] \ge t) \le 2\exp\left(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

For $S = V$:

$$Pr(V \ge \epsilon) \le 2\exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^{m}(2x_i/m)^2}\right) = 2e^{-\epsilon^2 m/2}$$

Thus, with very small probability, a pair of vectors deviates from orthogonality, i.e. the cosine of the angle between them is more than ϵ.

We can use union bound to compute the probability that no pair deviates from orthogonality which is equal to 1 minus the probability that some pair $\mathbf{x}, \mathbf{y}$ is not orthogonal, i.e. $\cos(\theta_{x,y}) \ge \epsilon$.

$$Pr(\forall\, x, y \in \mathcal{X} : \cos(\theta_{x,y}) < \epsilon) \ge 1 - \sum_{\text{pairs}} 2e^{-\epsilon^2 m/2} = 1 - \binom{n}{2} 2e^{-\epsilon^2 m/2}.$$

For $m = \frac{6}{\epsilon^2}(\ln n)$, we get that

$$Pr(\forall\, x, y \in \mathcal{X} : \cos(\theta_{x,y}) < \epsilon) \ge 1 - n^2 e^{-\frac{\epsilon^2 m}{2}} = 1 - \frac{1}{n}.$$

Thus, almost surely all vectors are mutually orthogonal (for $0 < \epsilon < 1$).

References

[1] S. Ali, MH. Shakeel, I. Khan, S. Faizullah, and MA. Khan. Predicting attributes of nodes using network structure. 12(2), 2021.

[2] L. Bo, X. Ren, and D. Fox. Kernel descriptors for visual recognition. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 244–252, 2010.

[3] Achlioptas D. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003.

[4] M. Farhan, J. Tariq, A. Zaman, M. Shabbir, and I. Khan. Efficient approximation algorithms for strings kernel based sequence classification. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6935–6945, 2017.

[5] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Intern. Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pages 855–864, 2016.

[6] Z.R Hassan, M. Shabbir, I. Khan, and W. Abbas. Estimating descriptors for large graphs. In *Advances in Knowledge Discovery and Data Mining (PAKDD)*, pages 779–791, 2020.

[7] Hays J. and Efros A. Scene completion using millions of photographs. In *ACM SIGGRAPH*.

[8] P. Kuksa, I. Khan, and V. Pavlovic. Generalized similarity kernels for efficient sequence classification. In *SIAM Intern. Conf. on Data Mining (SDM)*, pages 873–882, 2012.

[9] L. Yang, Y. Wang, J. Gu, C. Wang, X. Cao, and Y. Guo. Jane: Jointly adversarial network embedding. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1381–1387, 2020.