Real Analysis Project

Importance of Metric Spaces in Internet Search Engine

2024-11-11

Roll numbers 19, 20, 21, 24, 25, 35

by

Sahasri Gundapuneedi (SE23UCAM019)

Sanket Motagi (SE23UCAM020)

Seerat Chatha (SE23UCAM021)

Tanishka Wagh (SE23UCAM024)

Koushik Reddy (SE23UCAM025) and

Kolli Yasaswini (SE23UCAM035)

Contents

§1. Foundations of Metric Spaces and Search Engines

This presentation explores the foundations of metric spaces and their applications in the field of search engines. We'll delve into the core concepts of metric spaces, discuss their importance in indexing and retrieval techniques, and examine how they contribute to efficient and effective search results.

§1.1. Introduction to Metric Spaces

Formal Definition: A metric space is a set equipped with a distance function (metric) that satisfies certain properties.

Key Properties: The distance function is non negative , symmetric and satisfies the triangle inequality.

Applications: Metric spaces find applications in various fields, including geometry, topology and computer science.

Examples: Common examples include Euclidean space, discrete metric space and the space of continuous functions.

§1.2. Properties of Metric Spaces

Open Sets: A set is open if it contains a neighborhood around each of its points

Closed Sets: A set is closed if its complement is open

§1.3. Convergence and Completeness in Metric Spaces

Convergent Sequence: A sequence in a metric space converges to a point if the distance between the terms of the sequence and the point approaches zero.

Cauchy Sequence: A sequence is Cauchy if the distance between its terms becomes arbitrarily small as the terms get further apart.

Completeness: A metric space is complete if every Cauchy sequence converges to a point in the space.

§1.4. Continuous Functions and Homeomorphism

Continuous Function: A function between metric spaces is continuous if it preserves closeness: points that are close in the domain remain close in the range.

Homeomorphism: A continuous function is a homeomorphism if it is a bijection and its inverse is also continuous.

Topological Equivalence: Homeomorphisms preserve the topological structure of spaces, meaning that two spaces are considered topologically equivalent if there exists a homeomorphism between them.

§1.5. Applications of Metric Spaces

Metric spaces have a lot of applications such as document similarity, relevance ranking , document custering which is further expalined in detail in Section 5.

§1.6. Conclusion

Metric spaces provide a powerful mathematical framework for understanding and applying distance concepts in search engines. Future directions include exploring novel metrics, improving indexing techniques and developing personalized search experiences.

In a discrete metric space, the distance between any two distinct points is defined as 1, while the distance between a point and itself is 0.

§2. Similarity Queries and Search Algorithms

To search is human

— Unknown

§2.1. Introduction

Let's begin with the reason we search. A query reflects a desire to find information. The primary task of a search engine is to interpret the searcher's intent based on their query — this is the essence of query understanding! However, each of us has a distinct way of expressing ourselves, and this individuality in communication is a core aspect of our humanity.

Unfortunately, machines don't appreciate our individuality. Indexing and retrieving content would be much simpler if everyone agreed on a controlled vocabulary[1], as well as a rigid schema to organize each document into fields.

Similarity query is a mechanism applied on a very large scale data. What exactly is it, you ask?

Definition 2.1.1 (Similarity Query)**.**
A variety of methods share the common approach of searching through (often vast) collections of objects, where the only way to compare them is by assessing the similarity between any two objects.

Let's understand this by taking an example

Example: Imagine a teenager managing a large collection of music tracks on their phone. They wanted to find songs that sounded similar to one they really liked. This was where similarity queries came into play.

- **Collection of Data:** The music library was treated as a set of data. Each song represented an object with different characteristics—genre, tempo, mood, and duration. These characteristics were like the "features" of the song.
- **Comparing Objects:** To find songs similar to the one they liked, they needed to compare that song (the query) with the rest of the tracks. The comparison was based on features such as tempo, genre, or mood, allowing them to measure how similar the songs were.
- **Similarity Query:** When they issued a similarity query, they essentially asked the system, "Find the songs in my library that are most similar to this one." The

[1] https://en.wikipedia.org/wiki/Controlled_vocabulary - not very important in this context

system compared the query song with the rest, determining which tracks were close in terms of features like tempo, genre, and mood.

Note: Many upcoming concepts shall be explained using this example itself among a few more examples here and there.

§2.2. Types of Queries

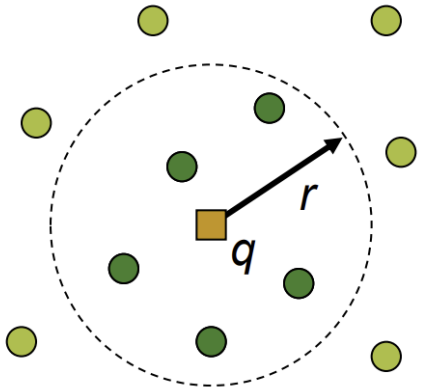
Now, there are quite a few types of Similarity Queries.

1. **Range Query:**

A **Range Query** is when you specify a certain distance r and ask for all objects within that distance from a query object q. Mathematically, it looks like this:

$$\mathcal{R}(q, r) := \{x \in X \mid d(q, x) \leq r\}$$

Figure 1 — Range Query



Continuing the previous example, say the teenager is interested in finding songs that are of same genre. The request may be, "Show me all songs that are within a certain range of similarity" (similarity being 'genre' in this case).

Another example might be, if someone wanted to find all museums within 2 km of their hotel, they would run a range query with q being the hotel and r being 2 km.

2. **Nearest Neighbor Query:**

In a **Nearest Neighbor Query**, the goal is to find the object closest to a given query object q. The result is NN(q) = x such that:

$$d(q, x) \leq d(q, y) \text{ for all } y \in X \text{ and } x \in X$$

It is like asking, "What is the most similar song to the one I'm listening to?" Instead of specifying a fixed range, the system looks for the one track in your collection that's closest to the query track based on some similarity measure (like genre, tempo, or mood).

- **k-Nearest Neighbor Query:** This is a variation where you want to find the top 'k' closest items. So instead of just the nearest neighbor, you get a set of the 'k' closest objects.

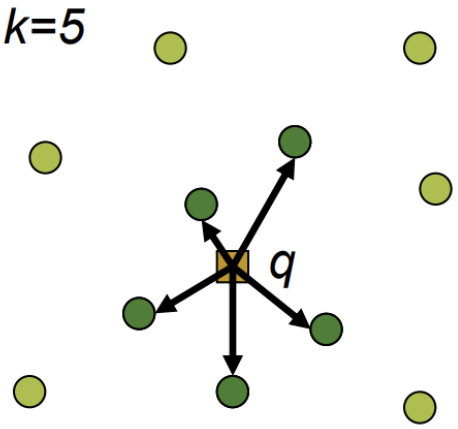
The result is k-NN(q, k) = A

where $A \subseteq X, |A| = k$, for all $x \in X, y \in X - A$ such that

$$d(q, x) \leq d(q, y)$$

For example, asking for the five closest museums to a hotel would return the five nearest options based on the distance.

Figure 2 — k-NN depiction (k being 5 in this case)



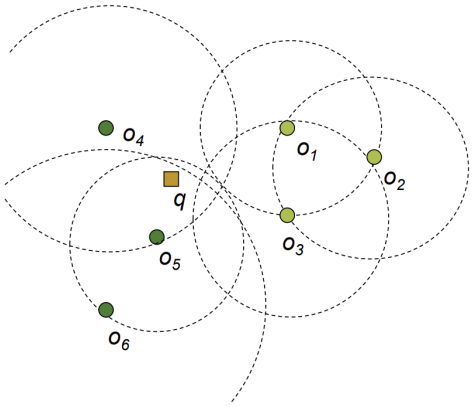
3. Reverse Nearest Neighbor Query:

The **Reverse Nearest Neighbor Query** flips the concept. Instead of finding the nearest object to a query, it finds all objects for which the query is the nearest neighbor. Mathematically:

$$\text{kRNN(q)} = \{R \subseteq X, \text{for all } x \in R : q \in \text{kNN(x)} \wedge \text{for all } x \in X - R : q \in \text{kNN(x)}\}$$

For instance, a reverse nearest neighbor query could be used to find which hotels have a specific museum as their nearest tourist attraction.

Figure 3 — 2RNN - Objects O_4, O_5, O_6 have q between their 2 nearest neighbor.

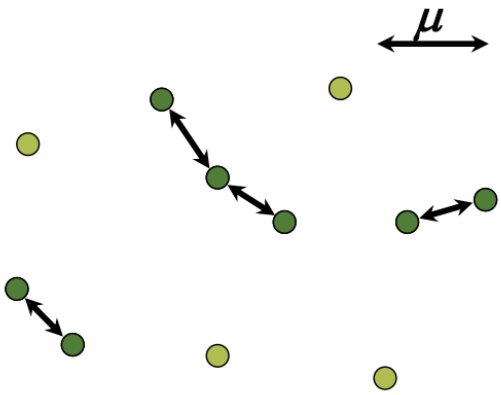


4. **Similarity Join:**

A **Similarity Join** compares two sets of objects, X and Y, and pairs items from both sets that are within a certain distance of each other (or within a similar range). This is useful in finding relationship between two datasets.

$$\text{J(X, Y, r)} = \{(x, y) \in X * Y \mid d(x, y) \leq r\}$$

For example, this could be used to find pairs of hotels and restaurants that are within a 10-minute walk from each other.



Imagine you have two playlists—one full of pop songs and another full of rock songs. A similarity join would compare each pop song to each rock song and return pairs of tracks that are within a certain similarity threshold (maybe based on mood or energy).

5. **Combined Queries:**

These queries combine multiple types of similarity queries. For instance:

- **Range + Nearest Neighbor Query:** Finding objects within a certain range that are also nearest neighbors. This can be depicted mathematically by:

$$\text{kNN(q,r)} = \{R \subseteq X, |R| \leq K \wedge \forall x \; \varepsilon \; \text{R, y} \; \varepsilon \; \text{X-R: d(q,x)} \leq \text{d(q,y)} \wedge \text{d(q,x)} \leq \text{r}\}$$

6. **Complex Queries:**

These involve more advanced criteria, such as finding the best match based on multiple attributes. For example,

- Find the best matches of *circular* shape objects with red **color**.
- The best match for circular shape or red color need not be the best match combined!

§2.3. The A_0 Algorithm

The A_0 Algorithm is designed to handle similarity queries where multiple predicates (criteria) are used to match objects, and each object is evaluated based on these criteria to find the most similar items.

- For each predicate i
  - objects delivered in decreasing similarity

- incrementally build sets X_i with best matches till

$$\forall i \mid \cap_i X_i| = k$$

- For all $o \; \varepsilon \; \cup_i X_i$
  - consider all query predicates
  - establish the final rank (fuzzy algebra, weighted sets, etc.)

§3. Metric Index Structures

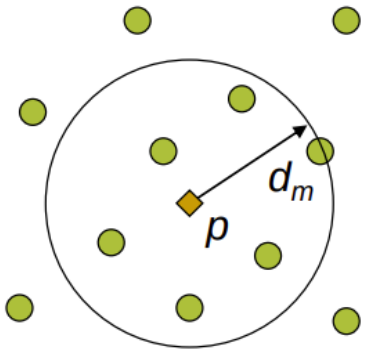
Cover the different indexing structures used in metric spaces to enhance search efficiency, like ball-partitioning and hyperplane partitioning. Explain how these structures reduce the computational cost of searches.

Given a set $X \subseteq D$ in $M = (D, d)$ three basic partitioning principles have been defined:

- Ball partitioning
- Multi-way ball partitioning
- Generalized hyper-plane partitioning
- Excluded middle partitioning

§3.1. Ball partitioning

- Inner set $\{x \in X \mid d(p, x) \leq d_m\}$
- Outer set $\{x \in X \mid d(p, x) > d_m\}$



By categorizing points into inner and outer regions, searches can be limited to relevant partitions, avoiding distance checks for points in the irrelevant partition.

Example Imagine a set of cities with a central "hub" city, say Paris. Let's use Paris as the pivot point p, with $d_m = 500km$

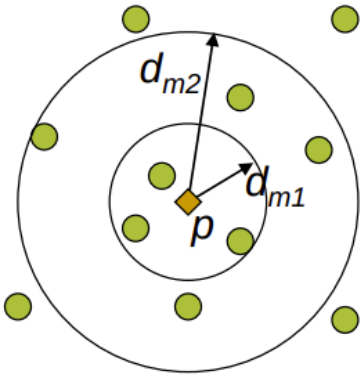
- Inner set: Cities within 500 km of Paris (e.g., Brussels, Luxembourg, Zurich).
- Outer set: Cities farther than 500 km from Paris (e.g., Madrid, Rome, Berlin).

In a search for cities near Paris, we can restrict attention to the inner set, saving time by ignoring distant cities.

§3.2. Multi-way ball partitioning

- Inner set $\{x \in X \mid d(p, x) \leq d_{m1}\}$

- Middle set $\{x \in X \mid d(p,x) > d_{m1} \wedge \leq d_{m2}\}$
- Outer set $\{x \in X \mid d(p,x) > d_{m2}\}$



Enables more granular divisions, especially useful for dense datasets by focusing search efforts on the most relevant partition.

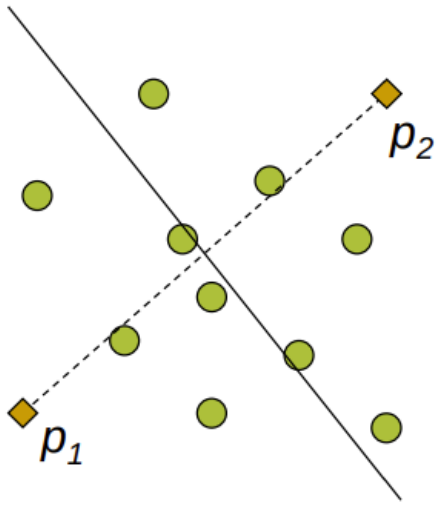
Example: Using Paris again as the pivot point but with two distances $d_{m1} = 300km$ and $d_{m2} = 800km$

- Inner Set: Cities within 300 km of Paris (e.g., Brussels, Luxembourg).
- Middle Set: Cities between 300 km and 800 km (e.g., Zurich, Munich).
- Outer Set: Cities more than 800 km away (e.g., Rome, Madrid).

This setup would allow a finer search within the middle range if we're interested in nearby regions but want to focus only on the closest or intermediate-distance cities.

§3.3. Generalized Hyper-plane Partitioning

- $\left\{x \in X \mid d(p_1,x) \leq d_{p_2,x}\right\}$
- $\left\{x \in X \mid d(p_1,x) > d_{p_2,x}\right\}$



Useful for cases where a binary partition around two points reduces search areas significantly, as objects closer to one pivot are in a separate set from those closer to the other.

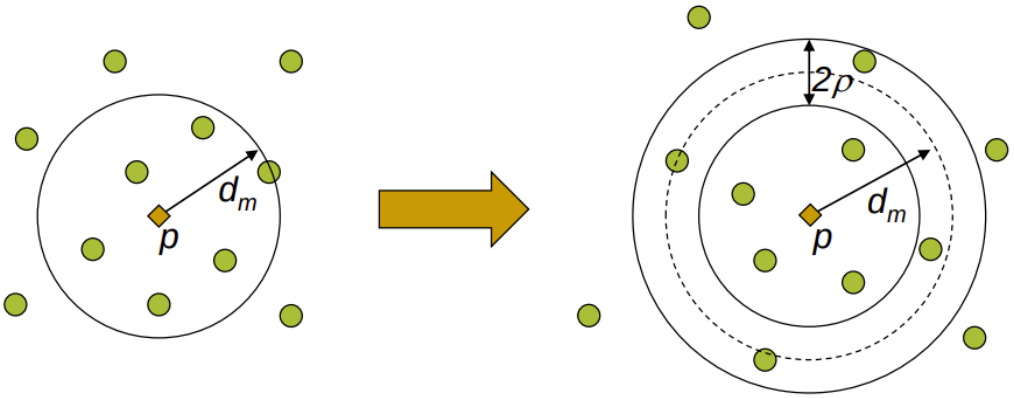
Example: Consider two distribution centers, one in New York (pivot p_1) and one in Chicago (pivot p_2). For each city x, we compare its distance to New York and Chicago.

- Left Set: Cities closer to New York than Chicago (e.g., Boston, Philadelphia).
- Right Set: Cities closer to Chicago than New York (e.g., Detroit, Minneapolis).

For a search prioritizing either distribution center, this structure lets us quickly determine which cities are closer to New York or Chicago, avoiding unnecessary comparisons.

§3.4. Excluded Middle Partitioning

- Inner set $\{x \in X \mid d(p, x) \leq d_m - \rho\}$
- Outer set $\{x \in X \mid d(p, x) > d_m + \rho\}$
- Excluded set: otherwise



The excluded region reduces ambiguity near boundaries, lowering the likelihood of unnecessary comparisons for points close to the partition edge.

Example: Using Paris again as the pivot city p, we set $d_m = 500km$ and introduce a "buffer" distance $\rho - 100km$

- Inner Set: Cities within 400 km of Paris (e.g., Brussels, Luxembourg).
- Outer Set: Cities more than 600 km from Paris (e.g., Rome, Madrid).
- Excluded Set: Cities between 400 and 600 km of Paris (e.g., Zurich, Munich).

The excluded region (400-600 km) avoids ambiguous cases for cities that are neither near nor far from Paris, speeding up the search by clearly defining what's "close enough" and what's not.

§4. Approximate Similarity Search

Approximate similarity search is a practical approach to finding things that are "close enough" to what you're looking for, without spending too much time or computational resources. It's all about balancing speed and accuracy, which is essential when you're dealing with big datasets or complex queries. Let's break down the core ideas behind this approach and how it works.[2]

§4.1. What is Approximate Similarity Search?

Imagine you're trying to find a specific image from a massive database. Exact search methods would compare each image pixel by pixel, which is computationally expensive and impractical. With approximate similarity search, we aim to retrieve images that are close enough to the target image by comparing key features instead of exact details.

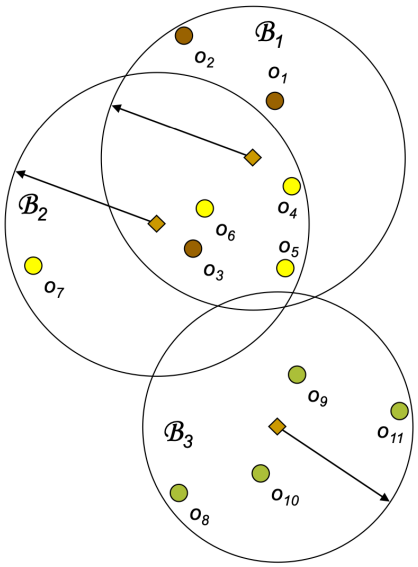
For example, if you upload an image of a golden retriever into a search engine, you don't need to find the exact same image but rather other images of golden retrievers. Approximate Similarity Search allows the system to quickly retrieve relevant results by focusing on core characteristics like color patterns, shapes, and other defining features.

Why "Approximate"?

The reason we settle for an "approximate" match instead of an exact one is that, in most real-world applications, exactness is not required for meaningful results. Approximate similarity search has the goal of reducing the cost of similarity queries by relaxing the correctness constraint, i.e., the k objects in the approximate result might not be the closest to the query object q. For instance, in recommendation engines, a user's preference for action movies doesn't require the algorithm to suggest movies with identical scenes or plot structures; instead, it can focus on approximate similarities such as genre, tone, or even the presence of similar actors.

Example :

[2] https://www.truefoundry.com/blog/similarity-search



This figure demonstrates a hypothetical index structure for approximate search in a metric space using three ball regions (B_1, B_2, B_3). Each ball region is defined by a center point and a certain radius, encompassing several objects within that range. This structure facilitates efficient approximate search by limiting the search space and focusing on relevant regions only.

Approximate Search : Range Query

Given a range query, the search algorithm accesses each ball region sequentially:

1. Access B_1 :

- Reports objects o_1.
- If early termination occurs, there is a risk of missing objects.

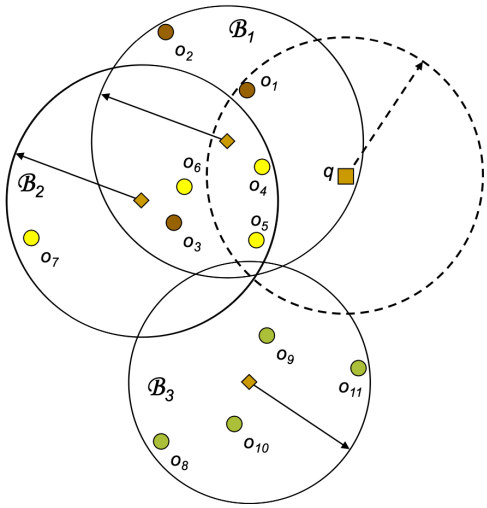
2. Access B_2 :

- Reports objects o_4 and o_5.
- No risk of missed objects if early termination occurs.

3. Access B_3 :

- No objects to report.
- A relaxed branching strategy can discard this region without missing objects.

Figure 4 — Approximate Search : Range Query



Approximate Search: 2-NN Query

For a 2-nearest neighbor (2-NN) query, the search proceeds as follows:

1. Access B_1 :

- Finds o_1 and o_3 as neighbors.
- If early termination happens, it may miss certain objects.

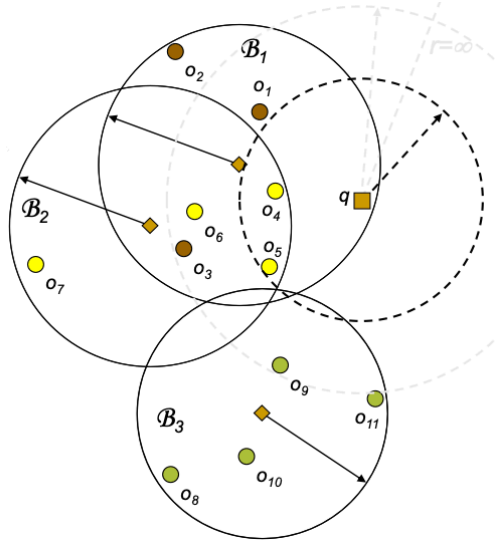
2. Access B_2 :

- Finds o_4 and o_5. as neighbors.
- No risk of missed objects if early termination occurs.

3. Access B_3 :

- Identifies o_4 and o_5 as neighbors, with no change in result.
- A relaxed branching strategy can safely discard this region.

Figure 5 — Approximate Search : 2-NN Query



This method provides a balance between efficiency and completeness. The early termination allows the algorithm to stop if the required information is found, saving computation time while ensuring essential objects are not overlooked.

§4.2. Key Strategies Used in Approximate Search

1. Dimensionality Reduction: When you're working with complex data, like high-resolution images or detailed text, the amount of information can be overwhelming. Dimensionality reduction is like shrinking down the data without losing the essential details. Techniques like PCA (Principal Component Analysis) or KLT (Karhunen-Loeve Transformation) are commonly used to reduce the complexity, making it faster to search. For example, instead of comparing every pixel in an image, you might only compare the most important features, like color or texture.

2. Early Termination: Think of this as knowing when to stop looking. In approximate search, the algorithm doesn't wait until it's checked everything; it stops as soon as it's confident it has a good enough answer. This is especially useful when the search is part of an interactive system—like when you're refining search results on an e-commerce site. The system can return a decent list of recommendations quickly, even if it hasn't scanned every possible item.

3. Relaxed Branching: This is a clever way of skipping over parts of the data that are unlikely to contain relevant results. Imagine you're searching for the nearest Starbucks in your city. If you know there are no Starbucks in the suburbs, you don't need to check those areas. Relaxed branching applies this same idea to data, skipping over regions that are less likely to contain good matches, which saves time.[3]

§4.3. Understanding Measures of Performance

When working with approximate similarity search, it's essential to evaluate how well the system performs—especially since it trades off some level of accuracy for speed and efficiency.

§4.3.1. Improvement in Efficiency (IE) :

One of the main reasons we use approximate similarity search is to make the search process faster. Improvement in Efficiency (IE) quantifies how much faster an approximate search is compared to an exact search.

What is IE? It's calculated as the ratio of the cost of exact search to the cost of approximate search. The "cost" here refers to computational resources, like the number of distance computations or disk accesses.

[3] https://en.wikipedia.org/wiki/Dimensionality_reduction

$$\text{IE} = \frac{\text{Cost(Q)}}{\text{Approximate Cost(Q)}}$$

- Q is a range or k-nearest neighbors query.

IE = 10 means that approximate execution is 10 times faster.

§4.3.2. Precision and Recall :

While efficiency is important, we also need to assess how accurate the search results are. Two widely-used metrics for this are Precision and Recall.

- **Precision** measures how many of the retrieved results are actually relevant. It's the ratio of relevant results retrieved by the search to the total number of results it returns.

$$\text{Precision} = \frac{\text{Number of relevant results retrieved}}{\text{Total number of results retrieved}}$$

A high precision means that most of the returned results are what the user wanted, but it doesn't tell us if we've found all the relevant results.

- **Recall**, on the other hand, measures how many of the total relevant results the search managed to find. It's the ratio of relevant results retrieved to the total number of relevant results in the database.

$$\text{Recall} = \frac{\text{Number of relevant results retrieved}}{\text{Total number of relevant results in the dataset}}$$

High recall means you've found most or all of the relevant items, but it doesn't say anything about how many irrelevant items were also retrieved.

Accuracy can be quantified with Precision (P) and Recall (R) :

$$\mathbf{P} = \frac{|S \cup S^A|}{|S^A|} \text{ , } \mathbf{R} = \frac{|S \cup S^A|}{|S^A|}$$

S – qualifying objects, i.e., objects retrieved by the precise algorithm

S^A – actually retrieved objects, i.e., objects retrieved by the approximate algorithm

The Trade-off Between Precision and Recall : You get faster results, but there's a chance that the most accurate matches might be missed or that irrelevant results might sneak in. The trick is to find the right balance between speed and accuracy for your specific use case.

For example, in applications like online shopping recommendations, where customers expect fast results, approximate search is perfect because it returns good enough suggestions quickly. On the other hand, in medical research, where precision is critical, you might need a more exact search even if it takes longer.

§4.3.3. Relative Error on Distances (ED)

In approximate similarity search, we're often interested in how far off the results are compared to what we would get with an exact search. Relative error on distances (ED)

helps to measure this by comparing the distances from the query object to the objects retrieved in both the approximate and exact results.

A low value of ED means that the approximate method is getting pretty close to the actual result, while a high value indicates that there's a significant error in the distances.

- Another possibility to assess the accuracy is the use of the relative error on distances (ED). It compares the distances from a query object to objects in the approximate and exact results.

$$\mathbf{ED} = \frac{|d(q, O_a), - d(q, O_n)|}{d(q, O_n)} = \frac{d(q, O_a)}{d(q, O_n)} - 1$$

where o_a and o_n are the approximate and the actual nearest neighbors, respectively.

- Generalisation to the case of the j-th NN:

$$\mathbf{ED_j} = \frac{d(q, O_a^j)}{d(q, O_n^j)} - 1$$

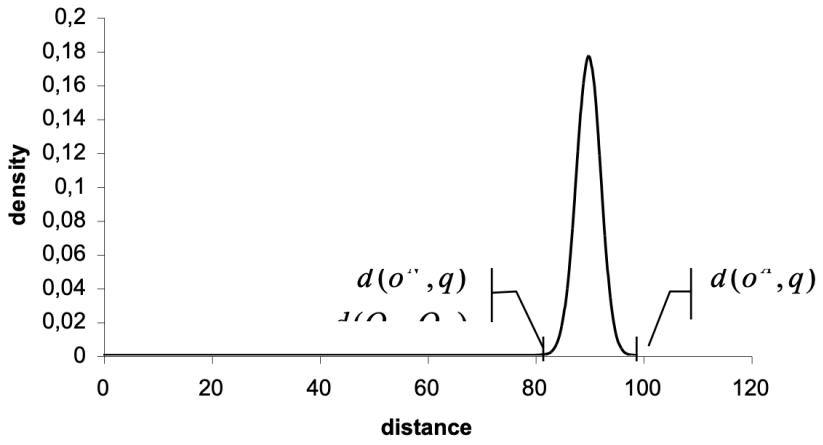
The concept of relative error on distances ED_j measures the deviation of approximate results from the exact distances. It has a drawback in that it does not account for the distribution of distances. Two examples illustrate this:

Example 1 : The difference in distance from the query object to o^N and o^A is large (compared to the range of distances).

- If the algorithm misses o^N and takes o^A, ED is large even if just one object has been missed.

Example 2 : Almost all objects have the same (large) distance from the query object.

- Choosing the farthest rather than the nearest neighbor would produce a small ED, even if almost all objects have been missed.



These examples highlight the importance of considering distance distribution in designing approximate search methods.

§4.4. Applications of Approximate Similarity Search

1. Image and Video Search

Think of platforms like Google Images or Pinterest. Have you ever uploaded a picture and found similar images or videos instantly? That's approximate similarity search at work. Instead of trying to find an exact match, these platforms use algorithms or advanced deep learning techniques to look for images or videos that have similar patterns, shapes, or colors.

2. Recommendation Engines

Netflix and Amazon use approximate similarity search to recommend shows and products. Instead of finding users with the same tastes, they group people based on similar habits, allowing for personalized suggestions even if preferences don't match perfectly.

3. Natural Language Processing (NLP)

When you use search engines or chat with virtual assistants, a lot of the "smarts" come from NLP systems that understand the meaning behind words. Approximate similarity search allows these systems to find words, sentences, or even entire documents that are "close enough" in meaning to what you're looking for.

4. Biometrics and Security

Think of fingerprint scanners, iris recognition, or face unlock on your phone. These systems need to recognize you even if your scan isn't exactly the same every time (different lighting, angles, etc.). Approximate similarity search helps match fingerprints or facial features even when the conditions vary slightly, ensuring secure and reliable identification without needing an exact match.

§5. Applications of Metric Spaces

Metric spaces have a significant role in search engines, like Google, especially in structuring data retrieval tasks and recommendation systems. The key idea is that objects (e.g., documents, images, web pages) are represented as points in a high-dimensional space, where distances between points are used to evaluate their similarity. The closer the the two points are, the more similar the search results or recommendations

§5.1. Application of Metric Space in Other Fields:

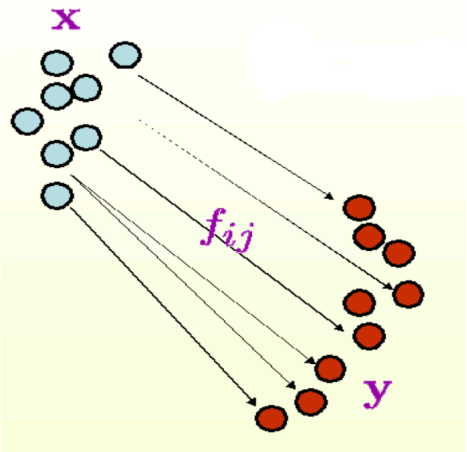
Before we dive into the applications of metric spaces in search engines, let's have a brief look at the uses of metric spaces in other fields.

§5.1.1. Earth Mover's Distance:

- In computer science, the earth mover's distance (EMD) is a measure of dissimilarity between two frequency distributions, densities, or measures, over a metric space D using color or texture-based image retrieval.
- Intuitively, given two distributions, one can be seen as a mass of earth properly spread in space, the other as a collection of holes in that same space. Then, the EMD measures the least amount of work needed to fill the holes with earth. Here, a unit of work corresponds to transporting a unit of earth by a unit of ground distance.[4]

$$\text{EMD(x, y)} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij}d_{ij}}{\sum_{i=1}^{m}\sum_{j=1}^{n} f_{ij}}$$

Figure 6 — Earth Mover's Distance



§5.1.2. The Hausdroff distance

[4] https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RUBNER/emd.htm

- In mathematics, the Hausdorff distance measures how far two subsets of a metric space are from each other.[5]
- The Hausdorff distance represents the farthest distance you might need to travel to go from one shape to the nearest point on the other shape.
- We are given two point sets $A = \{a_1, a_2, a_3....a_n\}$ and $B = \{b_1, b_2, b_3, ...b_m\}$ in E^2. The directed Hausdroff distance from A to B is defined as :

$$\tilde{\delta}_H(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\|$$

The bidirectional Hausdorff distance between A and B is then defined as:

$$\delta_H(A, B) = \max\Big(\big(\tilde{\delta}_H(A, B), \tilde{\delta}_H(B, A)\big)\Big)$$

- In computer vision, the Hausdorff distance helps find a specific template (like a shape or image) within another image. One of the main application of the Hausdorff distance is image matching, used for instance in image analysis, visual navigation of robots, computer-assisted surgery,
  - Basically, the Hausdorff metric will serve to check if a template image is present in a test image ; the lower the distance value, the best the match. Taking an example:

Figure 7 — Test and Template Image

[5] https://en.wikipedia.org/wiki/Hausdorff_distance

Figure 8 — Edge Extraction

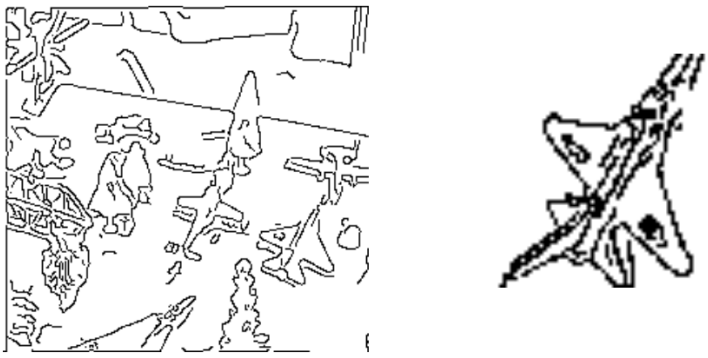
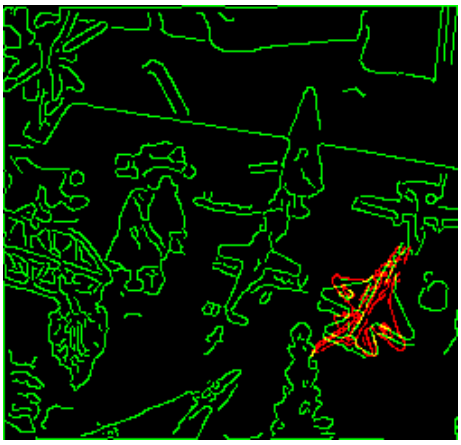


Figure 9 — Best Match



§5.1.3. Fréchet distance

- In mathematics, the Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Hausdroff distance has this downside as it may call things similar which dont seem alike.

Figure 10 — Problem with Hausdroff distance

  - As we can see that the two curves have a small Hausdorff distance, but do not look like each other at all.[6]

- Therefore, we may wish for the correspondence between the two shapes to reflect the underlying connectivity of their shapes. The Fréchet distance takes this into account.

[6] http://cgm.cs.mcgill.ca/~athens/cs507/Projects/2002/StephanePelletier/

- Let A and B be two given curves in a metric space S. Then, the Fréchet distance between A and B is defined as the infimum over all reparameterization α and β of [0,1] of the maximum over all $t \in [0, 1]$ of the distance S between $A(\alpha(t))$ and $B(\beta(t))$. In mathematical notation, the Fréchet distance F(A,B) is

$$F(A, B) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \{d(A(\alpha(t)), B(\beta(t)))\}$$

where d is the distance function in S.

- The Fréchet distance can be illustrated as follows : suppose a man is walking his dog and that he is constrained to walk on a curve and his dog on another curve. Both the man and the dog are allowed to control their speed independently, but are not allowed to go backwards. Then, the Fréchet distance of the curves is the minimal length of a leash that is necessary.
    - The equation reads like: for every possible function α(t) and β(t), find the largest distance between the man and its dog as they walk along their respective path; finally, keep the smallest distance found among these maximum distances.

§5.2. Applications of Metric Spaces in Search Engines:

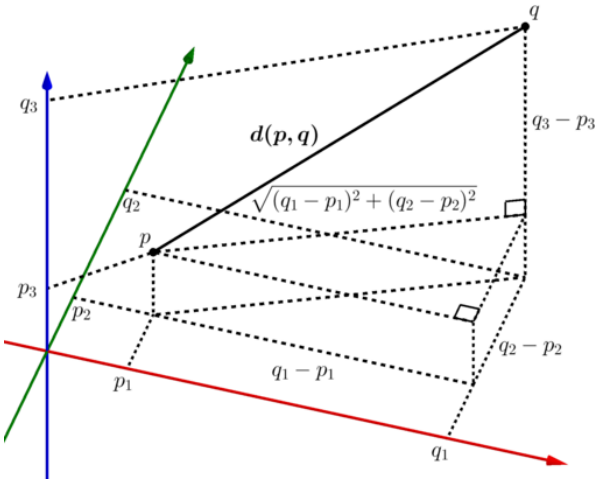
§5.2.1. Document Ranking and Retrieval

- One of the primary tasks of a search engine is to retrieve and rank documents based on their relevance to the user's query. Metric spaces enable search engines to quantify the "distance" between a query and the documents in the search index. By applying specific distance metrics—such as **Euclidean distance, Cosine similarity, or Jaccard similarity**- the search engine can rank documents in order of relevance. For example:
- **Euclidean distance** measures the straight-line distance between two points in a vector space, and can be used when queries and documents are represented as vectors.

$$d(p, q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2})$$

p,q = two points in Euclidean n-space
q_i, p_i = Euclidean vectors

Figure 11 — Deriving the n-dimensional Euclidean distance formula



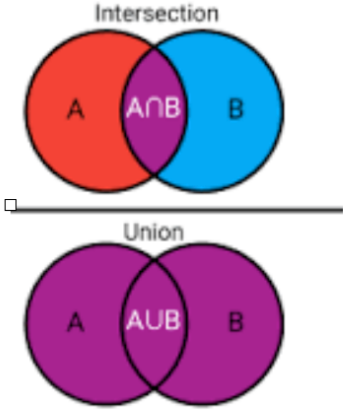
- **Cosine similarity** compares the angle between two vectors, focusing on the direction rather than the magnitude, making it ideal for comparing text-based content where the number of words (magnitude) may vary but the overall context (direction) remains similar.

$$\cos\theta = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

- **Jaccard similarity** measures the similarity between two sets by comparing the ratio of shared elements to the total number of unique elements. It is useful for determining overlap between user queries and document keywords.
  - In mathematical notation, it can be expressed as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Figure 12 — Jaccard similarity



These distance metrics are critical in ranking algorithms such as Google's PageRank, which determines the most relevant and authoritative documents for a given query.

§5.2.2. Query Optimization and Refinement

When users input vague or incomplete queries, search engines utilize distance metrics to suggest more refined or related queries. For instance, if a query yields few results, the search engine might suggest alternative terms with a closer semantic distance to the user's original input.

§5.2.3. Clustering Algorithms

Search engines often need to group similar documents together to provide more organized results. For example, documents that discuss the same topic or are highly related to one another are clustered based on their proximity in a metric space. Algorithms like K-means clustering use Euclidean distance or other metrics to create clusters of documents, improving the search engine's ability to categorize and display results based on topics or themes.

§5.2.4. Personalized Search Results

Search engines use distance metrics to compare user preferences and behavior patterns with potential search results. By constructing a user's profile based on their previous searches and interactions, the search engine can measure the distance between the user's profile and the available documents.

§5.2.5. Image and Multimedia Search

Metric spaces are not limited to textual data, they are equally important in multimedia search. In image retrieval systems, search engines use metric spaces to measure visual similarity between images. Features like color, texture, and shape are represented as vectors in a metric space, and distance metrics are applied to find visually similar images.

§6. Practical Challenges in Internet Search Engine

In the context of internet search engines, numerous practical challenges arise, among which the curse of dimensionality is particularly prominent.

Let us explore the Curse of Dimensionality in greater detail.

§6.1. Curse of dimensionality

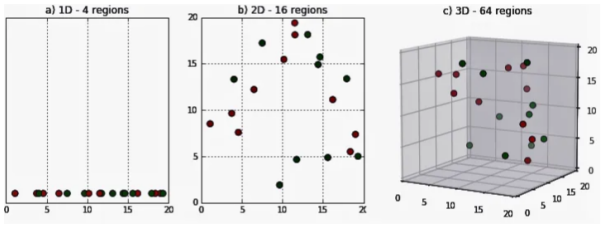
Definition 6.1.1 (curse of dimensionality)**.**
The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings

Let's understand this with an example:

Example: If you're measuring 100 different weather factors (like temperature, humidity, wind speed, etc.) in several cities, you're working in a 100-dimensional space because each factor is one dimension. Similarly, if you're analyzing black-and-white videos where each frame is 40x40 pixels, each frame has 1,600 dimensions (since 40 x 40 = 1,600 pixels). If the videos are in full color (RGB), each frame has 4,800 dimensions (because each pixel has three values: one for red, one for green, and one for blue, so 40 x 40 x 3 = 4,800).

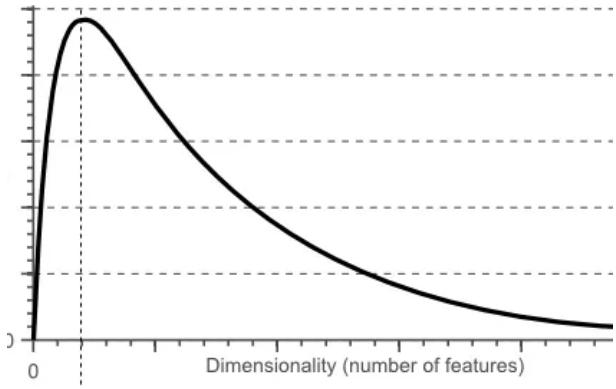
This means, as the dimensions increase we need to collect exponentially larger quantities of data (to be statistically significant). This exponential increase of data is the curse. It limits our ability to store, compute and make decisions quickly.

Figure 13 — "Illustration of the cod: (a) 1D space divided into 4 regions, (b) 2D space divided into 16 regions, and (c) 3D space divided into 64 regions."



The curse of curse of dimensionality in data science was first termed by Richard E. Bellman when considering problems in dynamic programming.

Hughes Phenomenon: This phenomenon means that, when you have a limited amount of training data, adding more features (or dimensions) at first improves the model's ability to make accurate predictions. However, after reaching a certain number of features, adding even more features starts to make predictions worse instead of better.



§6.1.1. Problems that arises due to curse of dimensionality

- **Data Sparsity:**
  - As the number of dimensions increases, data points become more spread out in the high-dimensional space, meaning fewer data points fall close to each other.
  - This sparsity makes it challenging to identify patterns, relationships, or clusters in the data because data points that might be close in a low-dimensional space are much farther apart in high-dimensional space.
- **Computational Complexity:**
  - The computational burden of algorithms increases exponentially.
  - The time and resources required for processing, storing, and analyzing high-dimensional data grow significantly, making it harder to run algorithms efficiently.
- **Over fitting:**
  - In high-dimensional spaces, models are more prone to memorize the data instead of learning the general patterns.
  - This can lead to poor generalization to new data because the model has essentially "over fitted" to the specific training data, capturing noise rather than signal.
- **Visualization Challenges:**
  - Visualizing data in high dimensions is challenging, as humans can only perceive up to three dimensions.
  - Without a way to effectively visualize the data, it's harder to understand patterns, relationships, and clusters.

§6.1.2. Techniques to Overcome Curse of Dimensionality:

1. **Dimensionality Reduction:**

Dimensionality reduction is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some

meaningful properties of the original data, ideally close to its natural dimension or in simple words,reducing the dimensions of our data set.

Dimensionality reduction decreases storage space and computation time, speeds up training, reduces multi collinearity, and can enhance model accuracy. It also makes data visualization easier, helping reveal meaningful patterns in lower dimensions (1D, 2D, or 3D).

Dimensionality Reduction can be divided into two types, **Feature Selection** and **Feature Extraction.**

- **Feature Selection**

  Selecting only the important features or eliminating the less useful features from our dataset. Instead of using all available features, algorithms like Recursive Feature Elimination (RFE) or using domain expertise can help in selecting only the most relevant features.

  In order to enhance our dataset, first delete the features, which have more than 50 - 60 % missing values since these features are not insightful enough for analysis. There can be a delimited specification when it comes to missing values, features going beyond this limit shall be dropped as well.

  Later, we will have to deal with the features that have a high correlation. If two variables are highly correlated, they contain almost the same information and it would be sensible to get rid of one of them without altering the result much. The correlation between the features can be established through the use of the corr() function in pandas.

  Feature Selection using **Random Forrest** — Random Forrest has an inbuilt feature of providing feature importance metric but before that, we need to convert our data into numerical form as Random Forrest takes only numeric inputs.

```
from sklearn.ensemble import RandomForestRegressor

fmodel = RandomForestRegressor(random_state=42,max_depth=10)

fmodel.fit(X_train,ytdf)

<ipython-input-52-5277a05809d0>:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cha
nge the shape of y to (n_samples,), for example using ravel().
  fmodel.fit(X_train,ytdf)

RandomForestRegressor(max_depth=10, random_state=1)

features = X_train.columns
importances = fmodel.feature_importances_
indices = np.argsort(importances)[-14:]
```

- **Feature Extraction**

  - Principal Component Analysis (PCA):

    PCA (Principal Component Analysis) is a technique of linear dimensionality reduction that takes a large number of variables and transforms them into a smaller number of variables known as the principal components. The first component is referred to as the 'first principle component', which captures most variance in the data set. There is a second component that captures variance uncorrelated with the first. Other additional components pursue the same goal, each of them explains the variance, which has not been explained by its predecessors. In other words, the goal of PCA is to find the set of new

variables that are linear combinations of the original variables and have the greatest variance in the new variables.

- t-Distributed Stochastic Neighbor Embedding:

  t-SNE is a non-linear dimensionality reduction algorithm designed for exploring high-dimensional data. It identifies patterns by assessing the similarity of data points based on their features, calculating the similarity as the conditional probability that point A would select point B as its neighbor.

  The algorithm aims to minimize the difference between these conditional probabilities in high-dimensional space and their representations in a lower-dimensional space, ensuring an accurate depiction of data points in a reduced dimensionality.

2. **Regularization Techniques**
   - L1/L2 Regularization (Ridge and Lasso Regression): Penalizing the complexity of a model by reducing the magnitude of coefficients can mitigate over fitting, especially in high-dimensional spaces.
   - Dropout: In neural networks, dropout randomly drops a fraction of the neurons during training to prevent the model from becoming overly reliant on certain features.
3. **Increase Training Data**
   - Data Augmentation: If collecting new data is impractical, techniques like data augmentation can synthetically increase the dataset size by introducing slight variations in the existing data points.
4. **Distance functions (especially Euclidean distance):**
   - A recommended solution is to use cosine similarity instead of Euclidean distance, as it remains less affected in high-dimensional spaces. This is particularly beneficial in text-related tasks (e.g., TF-IDF, word embeddings).

Note: Moreover, the effect of dimensionality varies based on point distribution:

1. Dense points: High dimensionality has a significant impact.
2. Sparse points: High dimensionality has a lesser impact.

§6.1.3. How does deep learning tackle the curse of dimensionality?

- **Identifying Key Features:**

Deep learning can separate the most important characteristics of the data through the exclusion of less relevant ones.

- **Constructing a Comprehensive View:**

Deep learning brings a hierarchical arrangement of structures in which a complex data is dissected into a set of simpler data and then composed together to realize the whole structure, the same way in which a complex structure is realized by assembling simpler structures like blocks.

- **Preventing Information Overload:**

Deep learning reduces complexity that comes with large datasets and uses approaches that give fewer errors when training our model.

- **Selective Utilization:**

In addition to that, deep learning sometimes only centers it's analysis on the most pertinent features and data while leaving out additional and irrelevant information.

- **Uncovering Latent Patterns:**

Deep learning can help to find patterns in the data even if data set is big, it's like seeing shapes in the cloud.

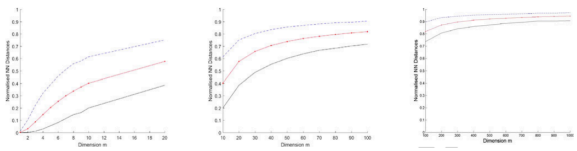
Another significant aspect of the DC that has recently emerged is the Concentration of Distances phenomenon.

§6.2. Concentration of Distances phenomenon.

Definition 6.2.1 (Concentration of Distances)**.**
(CoD), also referred to as Concentration of Norms in the literature, refers to the inability of distance functions to distinguish points well in high dimensions.

To measure the closeness between any two objects/points we need the concept of a distance or its dual concept of similarity. However, as the dimension increases all the points appear to be approximately at the same distance and the distance function seems to lose its discriminative power. This phenomenon is called the concentration of distances.

Figure 15 — "Concentration exhibited by the Euclidean distance when moving from low to high dimensions."



The concept of distance, or its dual notion of similarity, plays a central role in almost every algorithm or method in data analysis, from classification to clustering to similarity searches to pattern recognition.

§6.2.1. An Empirical Illustration of Nearest Neighbour Distances

This section examines how distances between data points change as dimensionality increases in a dataset.

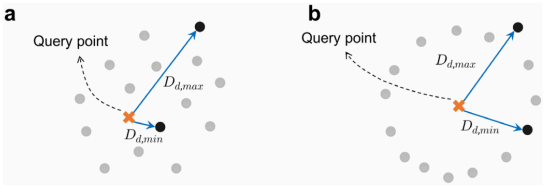
Given a dataset of 1,000 points, we focus on three types of Nearest Neighbour (NN) distances: The nearest neighbour distance for any given point is the distance to the closest other point in the dataset.

- Maximum NN Distance (k{max}): The largest distance to the nearest neighbour.
- Minimum NN Distance (k{min}): The smallest distance to the nearest neighbour.
- Average NN Distance (k{avg}): The mean of all nearest neighbour distances.

As dimensionality increases, these distances shift notably:

- **Low Dimensions:** NN distances are distinct, meaning points are spread out enough to be easily distinguishable.
- **Medium Dimensions:** NN distances start to converge, showing the onset of the "Curse of Dimensionality" (CoD), where points appear more uniformly spaced.
- **High Dimensions:** NN distances become nearly identical, making it challenging to distinguish between points based on distance alone due to the intensified CoD effect.

Figure 16 — "Distance concentration in NNS"



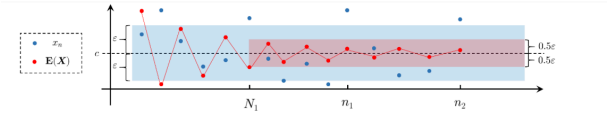
§6.2.2. Mathematics of Distance Concentration

Consider a sequence of data distributions $\{F_m\}$, each defined over increasingly high-dimensional spaces as m goes to infinity. For each m, let $\{x_1^{\{(m)\}}, l..., x_N^{\{(m)\}}\}$ be a sample of N random vectors. The distance function $|c \cdot|$ maps these vectors to positive real numbers, representing a measure of distance or magnitude.

Theorems on Distance Concentration

Theorem 2.1 (Beyer et al.): If the relative variance of distances (i.e., the variance divided by the square of the mean) converges to zero as dimensions increase, then the probability that the maximum distance is within a small factor $(1 + \varepsilon)$ of the minimum distance approaches 1. This implies that, in high dimensions, all distances between points become nearly equal.

Figure 17 — "Illustration for Theorem 1"



Theorem 2.2 (Durrant & Kabán): This theorem provides a converse to Theorem 2.1. If the probability of having the maximum distance close to the minimum distance approaches 1, then the relative variance must converge to zero. Thus, the concentration

of distances is both a consequence and a cause of the diminishing relative variance in high-dimensional spaces.

Collectively, these findings emphasize that with distance concentration, it becomes difficult to tell apart the points based on distance alone in high-dimensional spaces which undermines the usefulness of distance-based techniques.

§6.2.3. Specific challenges due to CoD in Internet Search Engines:

- **Reduced discrimination in high dimensions:** With CoD, common distance metrics produce identical results when comparing relevant and irrelevant factors. The quality of ranked search results is exposed especially in similarity/proximity based ranking tasks.
- **Increased computational load:**Traditional nearest-neighbour algorithms become computationally infeasible in high-dimensional contexts due to both the density of points and the complex calculations required to achieve meaningful distances .
- **Impact on Similarity Search:** As observed in the theoretical nearest-neighbour distances indeed lose contrast, such indefiniteness affects image and video retrievals as similarity is some times of the essence. CoD in that context might put the same query on several 'similar' search results which could lead to user frustration.
- **Decreased Recall:** In high-dimensional spaces, relevant documents may be placed far apart, leading to retrieval failures. This results in decreased recall, as users may not find all relevant information, undermining the comprehensiveness of search results.
- **Longer Query Response Times:** The complexity of processing high-dimensional data can slow down search engines. Increased computational demands and intricate indexing methods can lead to longer response times, frustrating users who expect quick results.

For Example:

For instance, in face recognition, one needs to search for a picture that is similar to the given query face in a database of images. A picture is made up of hundreds of thousands/ millions of pixels and hence is a high dimensional object. Similarity searching methods, typically employ some kind of a distance function to measure the closeness between two objects. However, as shown above, due to the high dimensionality of the data, all pairwise distances can converge and hence our search might return a lot of candidates similar to our query object. This clearly puts a question mark on the usefulness of distance functions in high dimensions

§6.2.4. Techniques to overcome concentration of Distances

- **Dispersion Function :** This function offers an empirical way to measure how points are distributed around any given point in high-dimensional space. It calcu-

lates the fraction of points within a certain range from each point, effectively capturing the spread of distances. Unlike other methods that require a theoretical understanding of the underlying data distribution, the Dispersion Function works directly with the dataset, making it applicable to real-world, complex data.

- **Empirical Calculation Advantage:** Unlike the Concentration Function , which is computationally intense and requires knowledge of the underlying distribution, the Dispersion Function uses only the internal characteristics of a dataset. This efficiency allows it to handle large datasets commonly seen in high-dimensional search applications without requiring a complex calculation of pairwise distances.
- **Comparison Capability:** The Dispersion Index associated with the Dispersion Function enables a direct comparison of how different distance metrics (such as Euclidean or fractional distances) perform in high dimensions. By providing an ordering of distance functions based on their concentration behavior, it helps identify the most suitable distance metric for a given high-dimensional dataset, significantly improving the effectiveness of similarity and nearest-neighbor searches in internet search engines.

§6.3. Conclusion:

Summarily, the challenges internet search engines face show problems that have to do with the curse of dimensionality and concentration of distances in high-dimensional space. As it gets increasingly sparse with data, similarity becomes difficult to measure and is more of a hindrance in delivering pertinent results from the search engines. Uniformity of distance can heavily damage the ability of the algorithm if it bases judgment on traditional metrics, which means their effectiveness becomes less distinguishable between nearly similar queries.

It will include dimensionality-reducing techniques and invent new metrics for distance. In this way, this approach will prove efficient with a solid background of robust mathematics at the core but more accurate and reliable in the core itself for search mechanisms. Interconnection of mathematics with life and technology on a day-to-day basis reveals that mathematics is a necessity which needs to be approached for retrieval and access problems for a modern information-based world.

§7. Bibliography

1. https://en.wikipedia.org/wiki/Controlled_vocabulary

2.https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RUBNER/emd.htm

3. https://en.wikipedia.org/wiki/Hausdorff_distance

4. http://cgm.cs.mcgill.ca/~athens/cs507/Projects/2002/StephanePelletier/

5. https://www.analyticsvidhya.com/blog/2021/04/the-curse-of-dimensionality-in-machine-learning/

6. https://www.xomnia.com/post/what-is-the-curse-of-dimensionality-and-why-does-deep-learning-overcome-it/

7. https://builtin.com/data-science/curse-dimensionality#:~:text=It%20increases%20exponentially%20from%2010%C2%B9,while%20also%20economizing%20on%20memory.

8. https://ieeexplore.ieee.org/document/4216305

9. http://www.nmis.isti.cnr.it/amato/similarity-search-book/slides/Similarity%20Search%20Part%20I%20Chapter%201.pdf

10. https://www.sciencedirect.com/science/article/pii/S1570866708000762#sec010

11. https://faculty.ucmerced.edu/mcarreira-perpinan/papers/phd-ch04.pdf

12. https://www.sciencedirect.com/topics/computer-science/relative-error

13. https://docs.oracle.com/en/database/oracle/oracle-database/23/vecse/perform-exact-similarity-search.html