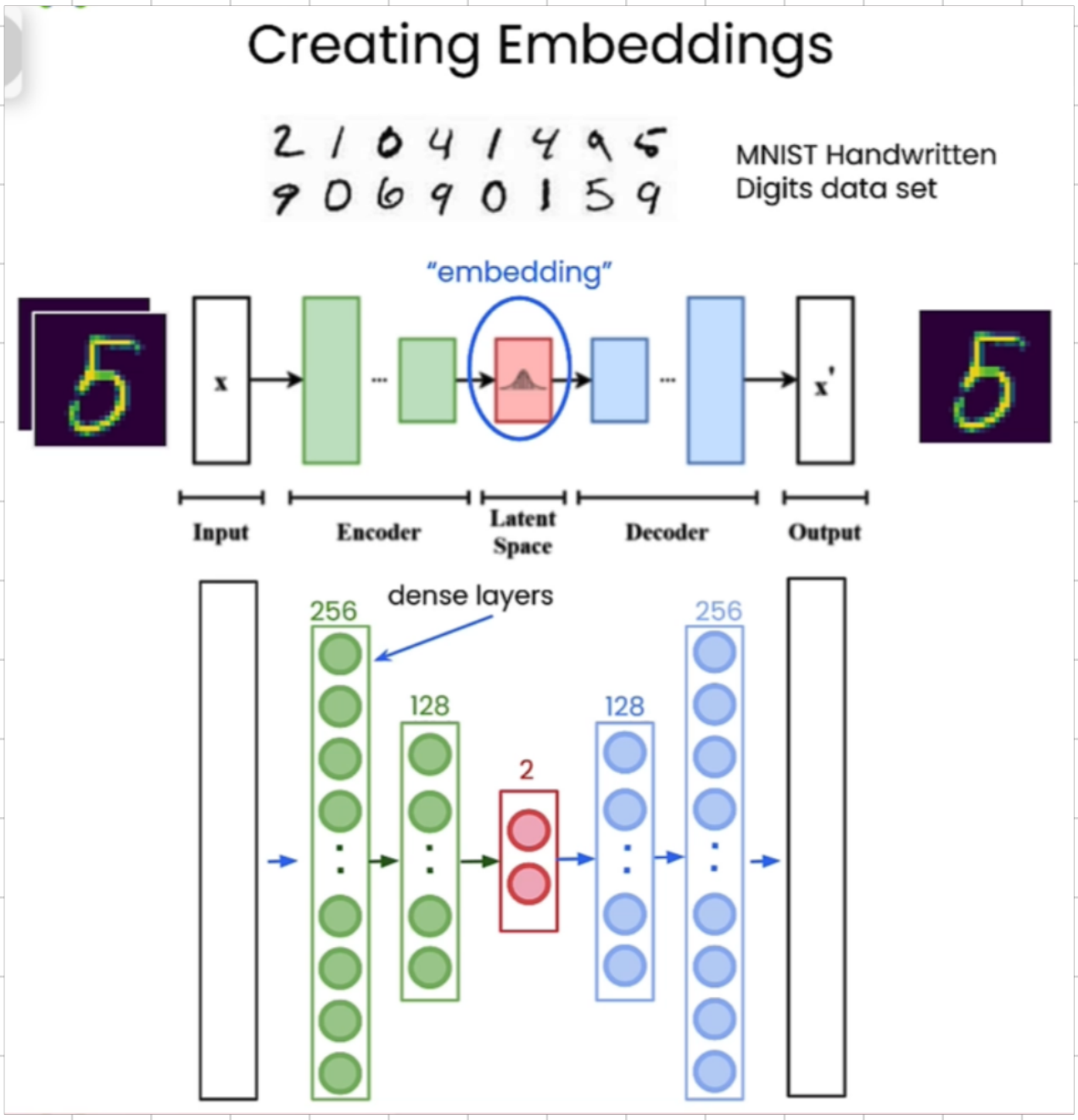
How to obtain Vector Representations of Data?



* The basic scheme of variational autoencoder. The model receives x as input. The encoder compresses it into latent space. The decoder receives as input the information sampled from latent space and produces x' as similar as possible to x.

Encoder: compress picture

Decoder: decompress picture

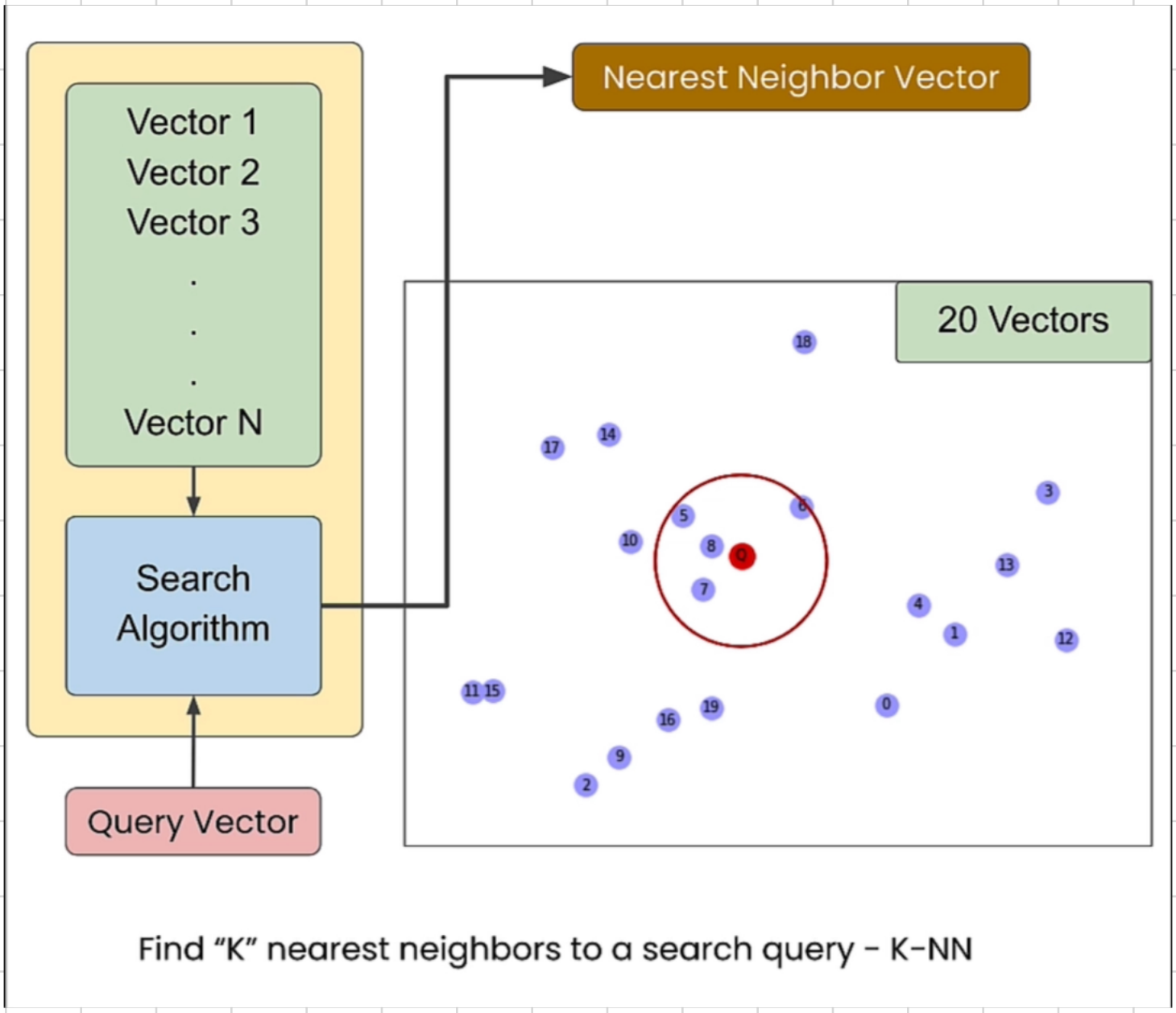
* The output image is generated using only the vector in the middle so that vector contains the meaning of that image and we can call that embedding.

* Vector embedding captures the meaning of underlying data

* You can think of vector embedding as machine-understandable format of the data.

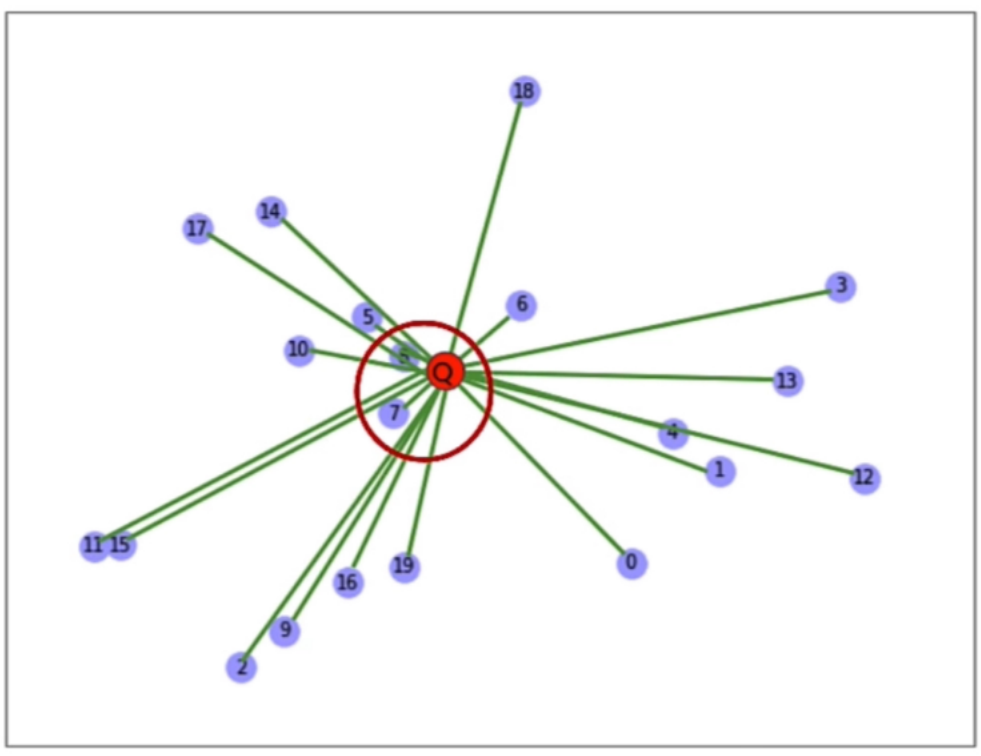
Searching for Similar Vectors :

Semantic (Vector Search) : Vectors capture the meaning behind our data and thus in order to find data points similar in meaning to our query we can search and retrieve the closest objects in vector space. By Semantic Search one means search that utilizes the meaning of words or images in question.



* "brute force" search algorithm:

'brute force' search algorithm



1. Measure the L2 distance between the Query and **each** vector
2. Sort **all** those distances
3. Return the top k matches. These are the most semantically similar points.

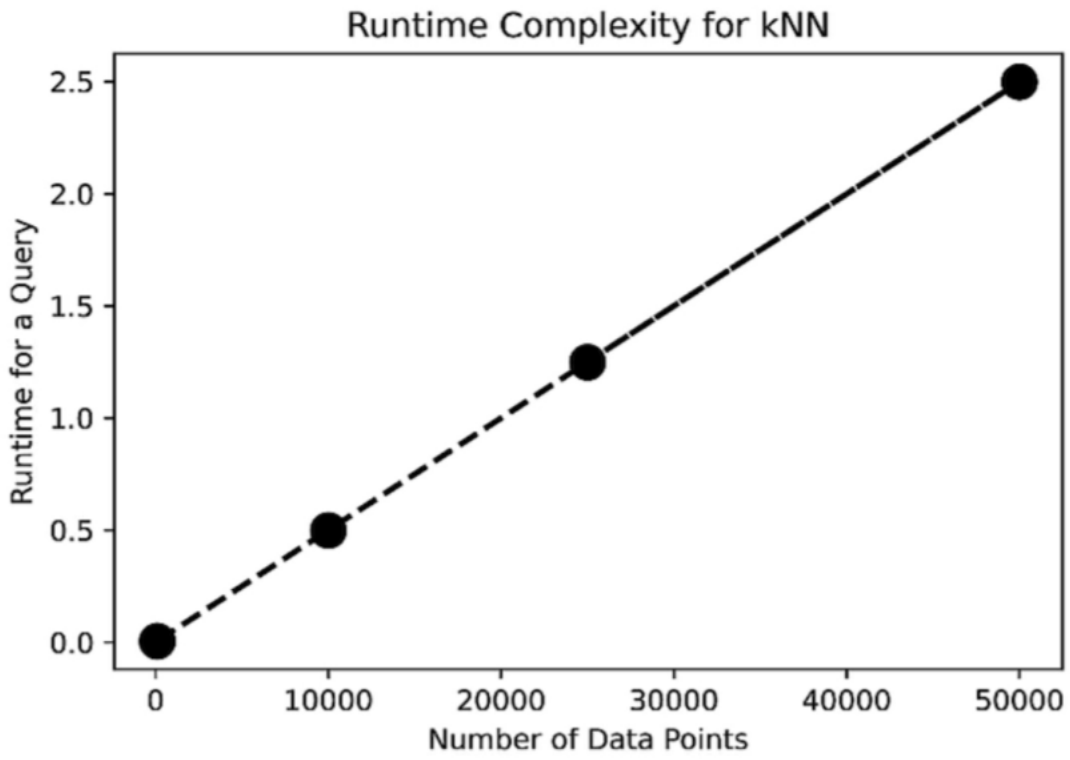
* brute force searches is similar to k nearest neighbor algorithm in classical ML.

* brute force searches are with a large computational cost.

* in brute force searches overal query time linearly grows with the amount of objects that we have in our star.

Runtime Complexity of kNN

- ***O(dN)* runtime complexity for search**

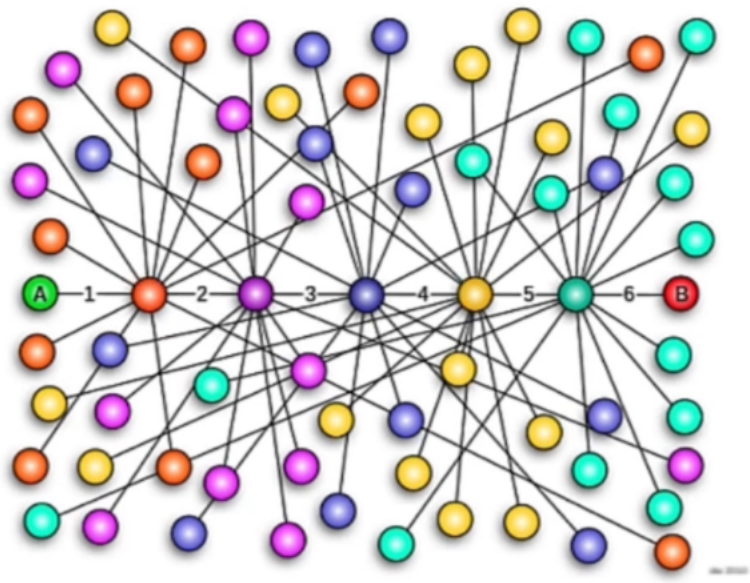


Approximate Nearest Neighbours:

* In an small vector db, searching for the nearest vector is not necessarility a big issue. However, as soon as we get to thousands or millions of vectors, this is becoming a way bigger problem and definitely a big no-go. There are many algorithms that allow us to actually find the nearest vectors in a more efficient way.

* one of the algorithms that solve this problem is HSNW, which is based on small world phenomenon in human social networks

small world



- Observed phenomenon in human social networks where everyone is closely connected.
- **Six Degrees of Separation:** Popularized concept that any two individuals are on average separated by six acquaintance links.
- **High Clustering:** Nodes form tightly-knit groups, with members of groups connecting to one another.
- **Short Path Lengths:** Despite high clustering, only a few steps are needed to connect any two nodes.

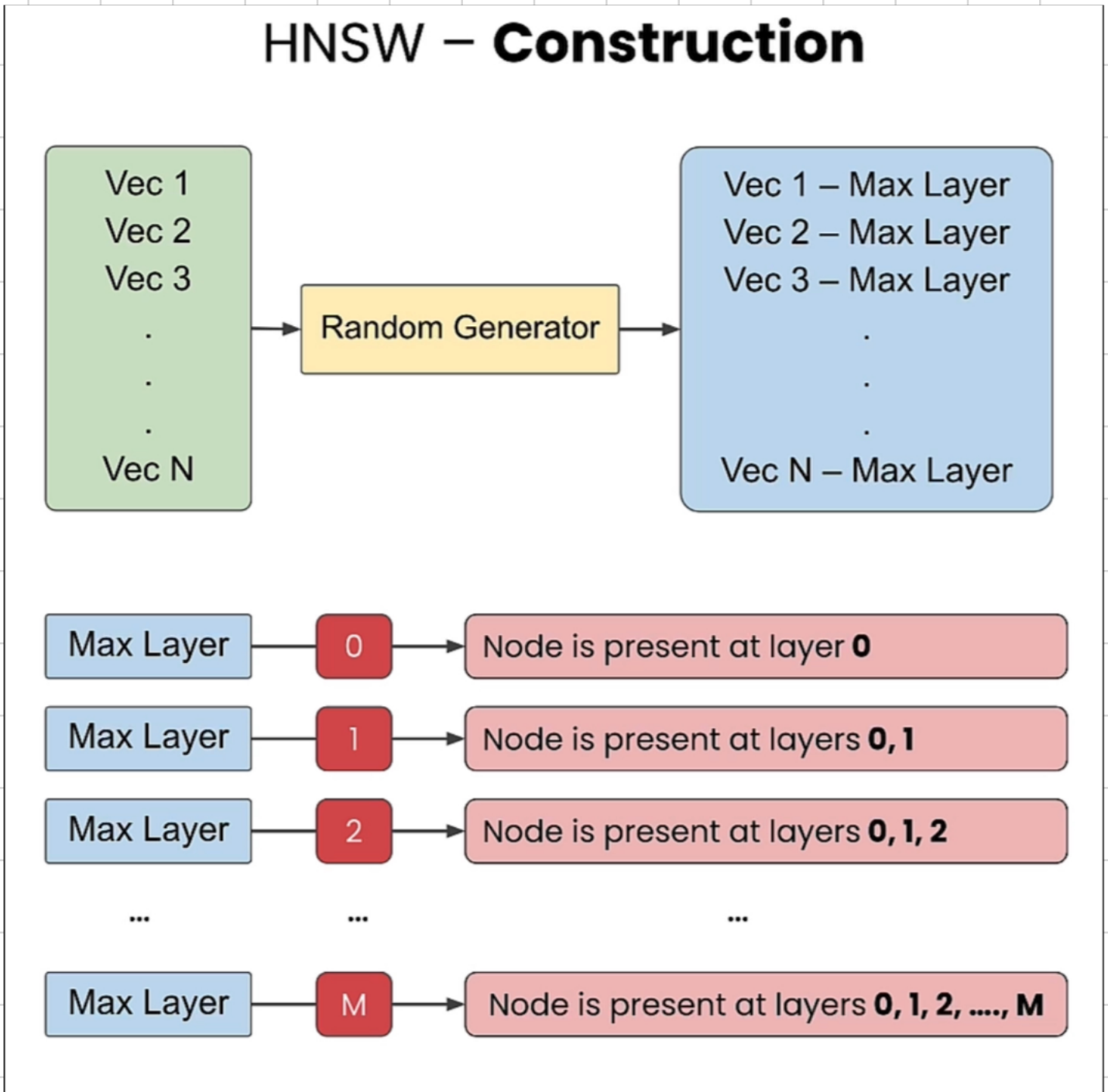
Navigable Small World (NSW): It is an algorithm that allow us to construct these connections between different nodes.

Hierachical Navigable Small World (HNSW): It puts several layers of navigable small worlds on top of each other. And the way, you can imagine it's a bit like if you were to travel to some place in the world. First, you probably would take a plane to the nearest airport to where you're trying to get to. Then, maybe you would go to and catch a train that would take you to the town where you wann get to. And then finally, once you are at the bottom layer, you would walk more take a taxi to the destination that you are going to.

* Construction of each layer of HNSW is very similar to how it's done in NSW.

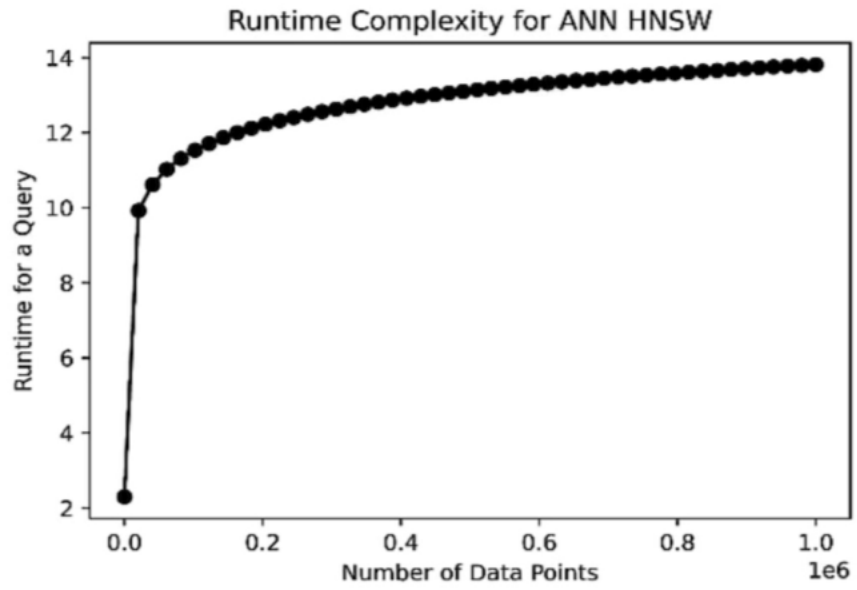
* The way the querying works with HNSW is that, we are starting with random node and we only can choose from those available at the highest level, and then we move to the nearest one within that layer. once, we are there, we can find the best possible match at that and the next layer, and eventually, once we are at the bottom layer, we can go to the object which is the closest to the query vector, which will help us to the last mile of search.

* the way nodes are assigned to each different layer. It's done by randomly generating a number which assigns that node to that layer and all the ones bellow and it's worth pointing that the chances of landing in a higher layer is logarithmically lower versus the one below. As a result we'll have a lot fewer nodes in the top layer versus the one towards the bottom.



HNSW Runtime

- **Low likelihood in Higher Levels**
  - The likelihood of a vector being found on the higher up levels of our HNSW graph goes down exponentially
- **Query time increases Logarithmically**
  - This means that as the number of datapoints increases the number of comparisons to perform vector search only goes up logarithmically.
- ***O(log(N))* runtime complexity**
  - As a results we have a *O(log(N))* runtime complexity for the HNSW algorithm



Vector databases:

Sparse, Dense, and Hybrid Search:

Sparse vs Dense Search

- **Dense Search (Semantic Search)**
  - Uses vector embedding representation of data to perform search.
    (as described in the previous lessons)
  - This types of search allows one to capture and return semantically similar objects.

💭 "Baby dogs"

🤖 "Here is content on puppies!"

* If the model that we are using was trained on a completly different domain, the accuracy of our queries would be rather poor. It's very much like if you want to go to doctor and asked them how to fix a car engine. well, the doctor probably wouldn't have a good answer for you

Dense Search Challenge

- **Out of domain data -** will provide poor accuracy
  - A Neural Network is as good as the data it was trained on.

💭 "Fixing a car engine"

👨‍⚕️ "Errrm... I am not a car doctor!"

* Another example is when we're dealing with stuff like serial numbers, like seemingly random string of text. In this case, there isn't a lot of meaning into codes like BB4330.

Dense Search Challenge

- **Product with a Serial Number**
  - Searching for seemingly random data (like serial numbers) will also yield poor results.

> 💭 "BB43300"
>
> 🤖 "Errrm... Bumble Bee?"

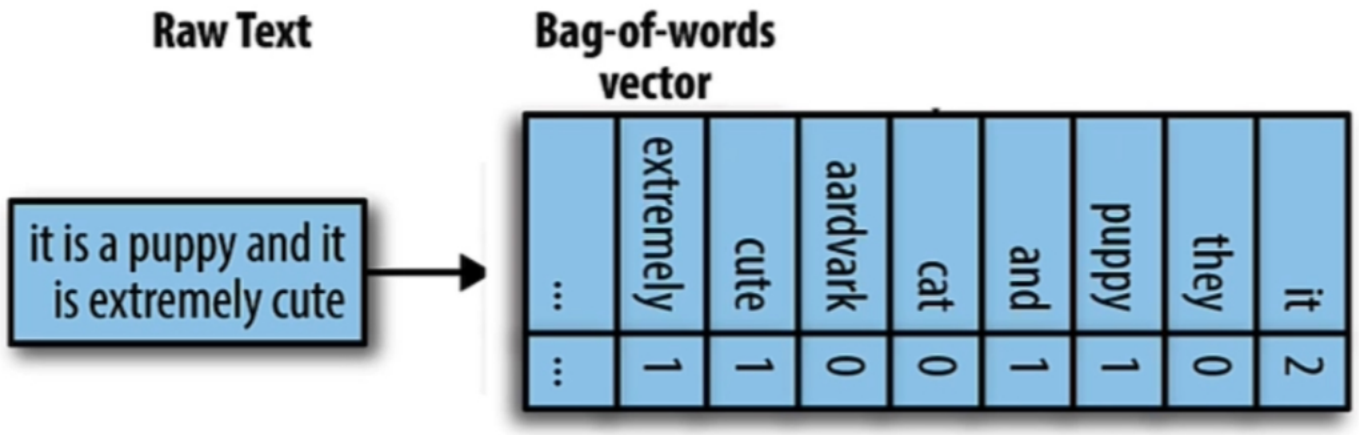
- **String or Word Matching**
  - Here we would be better off doing exact string or word matching to see where which product has the same serial number as the input serial number.
- This is known as **keyword search or sparse search!**

So, we need to go into a different direction for situations like this and try to go for keyword search, also known as sparse search.

Sparse Search: is a way that allows you to utilize the keyword matching across all of your content.

Sparse vs Dense Search

- **Bag of Words**
  - The easiest way to do keyword matching is using Bag of Words - to count how many times a word occurs in the query and the data vector and then return objects with the highest matching word frequencies.
- **This is known as Sparse Search**
  - because the text is embedded into vectors by counting how many times every unique word in your vocabulary occurs in the query and stored sentences.
- **Mostly Zeroes (Sparse Embedding)**
  - Since the likelihood of any given sentence containing every word in your vocabulary is quite low the embeddings is mostly zeroes and thus is known as a sparse embedding.



Source: https://uc-r.github.io/creating-text-features

* A good example of a keyword-based algorithms is Best Matching 25 (BM25). The idea behind it is that it counts the number of words within the phrase that you are passing in and then those that appear more often are weighted as like less important when the match occures but words that are rare if we match on that the score is a lot higher.

BM25

Best Matching 25 (BM25): In practice when performing keyword search we use a modification of simple word frequencies called *best matching 25*

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

Learn more about BM25: https://en.wikipedia.org/wiki/Okapi_BM25

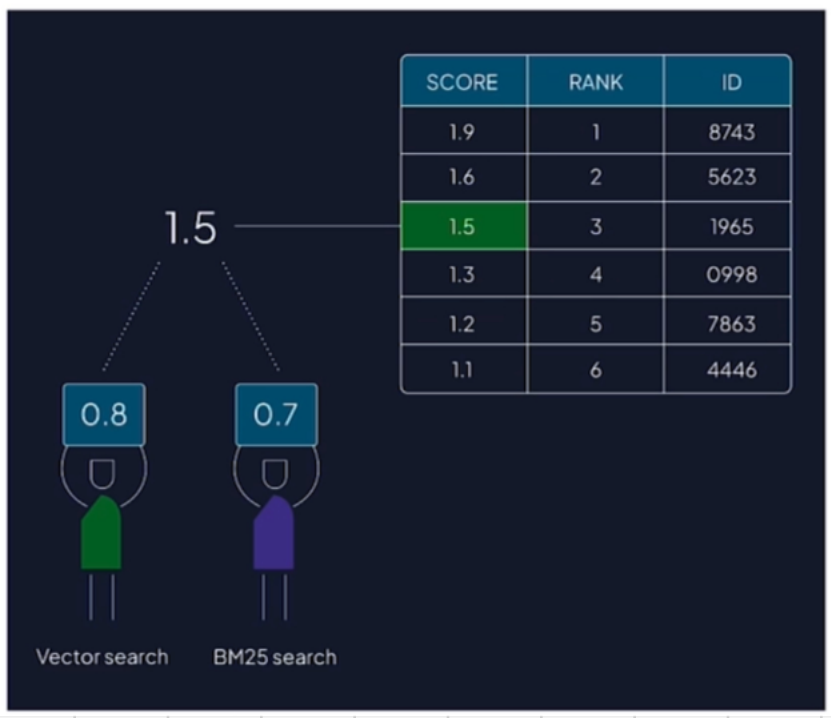
"If you can look into the seeds of time, And say which grain will grow and which will not."

[0.,0.,0.,0.33052881,0.,.,0.,0.20961694,0.,0.,.,0.,0.,0.,0.,.,0.,0.,0.20961694,0.20961694,.,0.20961694,0.20961694,0.,0.20961694,.20961694,0.20961694,0.,0.,0.,.,0.,0.,0.,0.20961694,0.,0.20961694,0.10938682,.,0.,0.20961694,0.41923388,0.41923388,0.,0.,0.20961694]

Source: https://en.wikipedia.org/wiki/Okapi_BM25

Hybrid Search

- **What is Hybrid Search?**
  - Hybrid search is the process of performing both vector/dense search and keyword/sparse search and then combining the results
- **Combination based on a Scoring System**
  - This combination can be done based on a scoring system that measures how well each objects matches the query using both dense and sparse searches.



Application, Multilingual Search:

Multilingual Search: It is very similar to how Semantic Search works where we can compare like you know dog and puppy. Then, still be able to find a very similar match. But in case of multilingual search you can have the same text but in different languages which will also generate very similar embeddings if not identical and through that we can use the same methods to search across content in any languages. We need.

Multilingual Search

- Because embedding produces vectors that convey meaning, vectors for the same phrase in different languages produces similar results.

"Vacation spots in California"

[-0.2479, -0.1360, -0.1075, 0.0973,
…
, -0.0055, -0.0283, -0.313, 0.1390]

"加利福尼亚州的度假胜地"

[-0.2479, -0.1360, -0.1075, 0.0973,
…
, -0.0055, -0.0283, -0.313, 0.1390]

Retrieval Augmented Generation (RAG)

- **Using a Vector Database as an External Knowledge Base**
  - Enable a large language model (LLM) to leverage a vector database as an external knowledge base of factual information
- **Retrieve Relevant Info and provide to LLM**
  - Improve a LLM by enabling it to retrieve relevant source material from the vector database and read it as part of the prompt prior to generating an answer to the prompt
- **Synergizes with a Vector Database**
  - vector databases can be queried for concepts using natural language
- **Better to do RAG**
  - Performing RAG is a lot more practical than having the LLM attend over its trained knowledge base
- **Example: Visiting a Library**
  - It's akin to a human visiting a library and checking out and reading source material and books prior to writing a well thought out response to a question.

Advantages of RAG

Here we can list out the advantages of RAG if we have time

- **Reduce hallucinations** - furthermore allow the user to identify hallucinations by comparing generated text to source text
- **Enable a LLM to cite sources** that it used to generate answers
- **Solve knowledge intensive tasks** and prompts more accurately

The RAG Workflow

- At its simplest the **RAG workflow consists of 4 steps:**
  - Step 1: **Query a vector database**
  - Step 2: **Obtain relevant source objects**
  - Step 3: **Stuff the objects into the prompt**
  - Step 4: **Send the modified prompt to the LLM to generate an answer**