Vector Databases

How are they used?

- LLMs' "knowledge" is limited to the data they were trained on - current data cannot be used
- LLMs therefore rely on vector databases when required to use recent data to complete actions/prompts
- More often now, vector databases are used by private companies to pass private company info in vector databases to LLMs for proprietary use on company/corporation specifics

How do they work?

1. First step - embedding model
    - Many pretrained models exist from OpenAI, Meta, Google, open source libraries, etc.
    - Learn from huge collections of text and the contexts they are used in
    - Able to map words into vector spaces with multiple dimensions - can imagine each dimension corresponds to some feature/criteria/parsing the model uses for each word, which leads to long vector coordinates for words
2. Next step - comparison
    - Now that words have been mapped as points in a space (Vector Database), the distance can be mathematically measured to determine some degree of commonality - however, not as easy for a model to see “groups” or nearby points as easily as humans, especially with many dimensions involved

Our example

- Convert 10 sentences into vector embeddings with 384 dimensions
- Compute the distances between these vectors to identify potential similarities
- For human visualization, compress embeddings into two-dimensional graph using principal component analysis
    - Like introducing a bias for certain features/dimensions the embedding model uses on each sentence, then squeezing all the data in these multiple dimensions down into a flat plane with preference for the important features/dimensions
    - Could easily be more than 2 important features (2 dimensions), could easily result in loss of big data, but practical for ability to see some results as an image

First Step - Embeddings

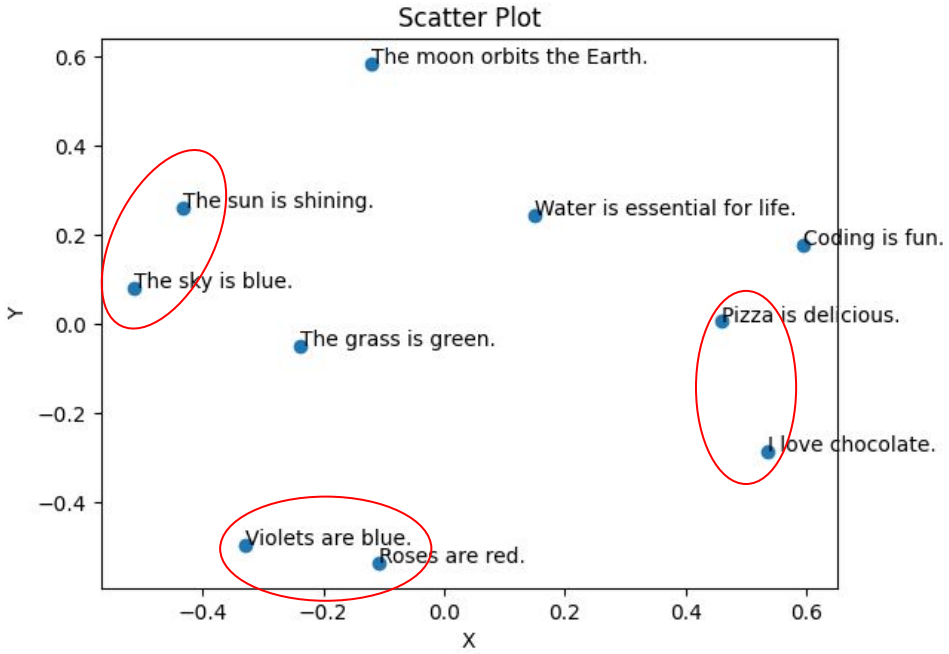
1. Create a HuggingFace token so that the program has access to the Hugging Face API and the sentence transformer model all-MiniLM-L6-v2
    - all-MiniLM-L6-v2 is an example of an open-source embedding model
2. Now we run the first code segment, which houses our 10 sample sentences (can change to literally anything)
3. Function `_get_embeddings` calls the sentence transformer model and passes our “text chunks” through it, then returns the embeddings given through the function
4. Function `from_text_to_embeddings` uses `get_embeddings` to simply create a data frame with the embeddings and export it to a .csv file which we can find in our Data folder

Next Step - Comparison

The next two programs are all about comparison - the first for human visual understanding, and the second for a more accurate but lengthy comparison

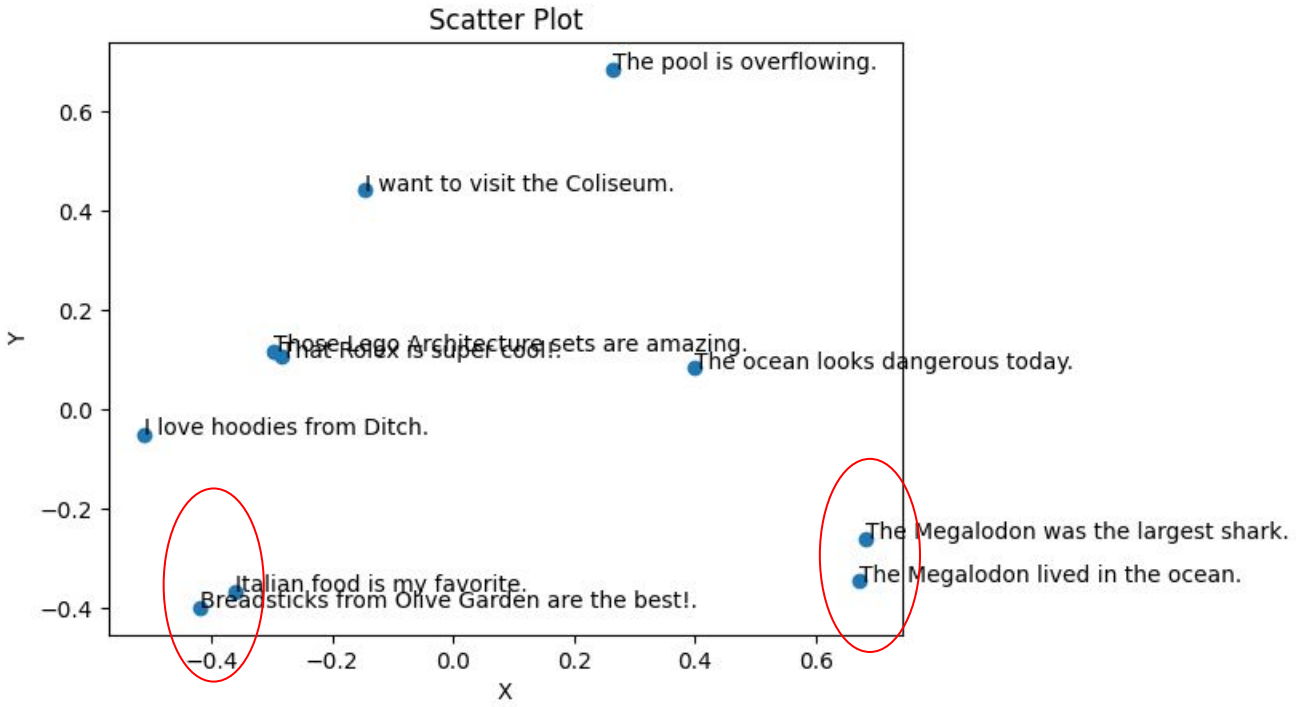
1. The function `create_pca_plot` performs a principal component analysis down to 2 features/dimensions from 384 and saves the data (now linear b/c 2 planes) to a data frame
    a. PCA identifies which features the data varies in the most and compresses it to most reflect those features/dimensions, allowing for summarization with the least possible info loss
2. `Create_scatter_plot` just takes that linear data and creates a scatter plot, then saves it as a png to our Data folder

The Results!



As humans, we can only assume what features were prioritized. 384 → 2 dimensions, even as efficient as the PCA can get, is a huge loss of information. However, we can still see some clusters in this image, like the sun and sky close to each other, pizza and chocolate close, and violets and roses in their own corner.

Custom Run



This is a run I created with sentences I made. Some clear clusters can be identified again, like Olive Garden and Italian food and the two Megalodon entries being close to each other.

But how can LLMs compare vectors?

The brain's ability to see and group so quickly is amazing, but we want our models to do the same, and in exponentially more complex situations because of the hundreds and sometimes even thousands of dimensions these vectors are in. So how do they do it?

Cosine Comparison

- In a vector space with so many directions, simple linear distance just won't do it - that takes into account only 2 planes maximum
- Many LLMs will use cosine similarity
  - The dot product of two vectors is divided by the dot product of their magnitudes
  - This will give a "cosine similarity" value
- Why is this so convenient?
  - The angle is very relevant to the direction of the vectors, so the score corresponds to their position in the space relative to each other
  - Cosine's range is only between -1 <= x <= 1, so a 1 means a high similarity score, 0 means no similarity, and -1 means a high dissimilarity or opposite
  - This corresponds to direction as follows: 1 means in the same direction/area, around a 0 degree angle; 0 means in perpendicular directions, around a 90 degree angle; -1 means in opposite directions, around a 180 degree angle

Third and Final Step - Cosine Comparison

This program calculates the similarities between vectors as a real LLM would, with all 384 features taken into account instead of being compressed to 2.

1. `Calculate_cosine_similarity` performs the cosine similarity calculations previously discussed on the embeddings we calculated from the initial program
2. These values are simply stored into another data frame, exported as a .csv file, and can be found in our Data folder