Vector Similarity Search with annoy

Challenge

Implement similarity search for a given set of tables. The data is stored in a file which should be read using duckdb.

The schema contains two tables: 'users' and 'movies'. In this demo, we will focus on the 'movies' endpoint, which will return the most similar entries when compared to a query.

Main questions:

What does similar means?

How can we measure similarity?

Tools?

Tooling and framework

- fastAPI
- annoy
- duckdb
- scikit-learn

We will implement an endpoint that receives a movie entity, the number of neighbors I want to get, and returns a list of these.

To measure the similarity between two entities, we will use vectors, as they provide concepts we can use to operate with them: add, subtract, and multiply.

Vectorization

How can we transform entities to vectors?

Using some standard rules to transform those.

Some example algorithms:

- Bag of words
- Word2vec
- TF-IDF (Term Frequency - Inverse Document Frequency)

Specific tool:

- FeatureHasher

Brute-force approach (effectiveness) - KNN algorithm

- Measure the distance between the query and each of the entries that are already in the dataset, repeat this until all the dataset is exhausted.
- We will reach the right answer, but not good with many entries, time complexity O(n).

Approximated approach (performance) - ANN

Annoy is an implementation of ANN. There are variants, but this is something easier to understand.

Explanation: https://sds-aau.github.io/M3Port19/portfolio/ann/

Recursively split a set of points into two parts until we reach a point where all the sets contain at most K points. We can keep track of the points and the splits using a binary tree. Once we have this binary tree, we can locate the leaf that contains the points closest to the queried one and return them.

Time complexity: O(log(n))

DEMO

GitHub Repository

Next Challenges

- Implement similarity search for users
- Index building could be time consuming, depending the amount of data, how can we improve this?
- DuckDB is an in memory DB for olap, how can we handle changes in the data, considering that it supports CRUD operations?
- Tools like pinecone and redis already provide searching functionalities, can we leverage those?

References

Similarity search:

https://www.piecone.io/learn/what-is-similarity-search/

https://mlops.community/vector-similarity-search-from-basics-to-production/

Vectorization:

https://neptune.ai/blog/vectorization-techniques-in-nlp-guide

ANN:

https://sds-aau.github.io/M3Port19/portfolio/ann/

https://erikbern.com/2015/10/01/nearest-neighbors-and-vector-models-part-2-how-to-search-in-high-dimensional-spaces.html

https://www.youtube.com/watch?v=DRbjpuqOsjk