Conversational Task Agent

Diogo Rodrigues 56153, João Vargues 55185, and José Murta 55226

NOVA School of Science and Technology, Portugal

Abstract. This report has an introduction on the work done, an explanation about the chosen algorithms and implementation and an evaluation about the dataset description of the first phase for the Web Search and Data Mining course project.

Phase 1: Answering natural questions

1 Introduction

Conversational Task Agent have influenced the way many people engage with everyday activities and it is expected that in upcoming years, the use of the conversational paradigm will substitute the traditional search-and-browse paradigm[8].

In this report it will be discussed the implementation of a conversational assistant that helps the user to execute a task and provide the user with proper guidance throughout the task.

This first phase, the main focus was on creating a searchable index of a recipe knowledge base that allowed the system to be able to solve a Question-Answer retrieval task.

2 Algorithms and Implementation

As explained in the introduction, the goal of the first phase of this project was to create a searchable index of recipe knowledge base implementing a Question-Answer search framework to answer Natural Questions. For that purpose we used the OpenSearch framework. Opensearch is a distributed search and analytics engine base on Apache Lucene, which organizes data into indices. Searches on the added data can be made using several types of queries with different features to be able to retrieve the desired data[2].

To index documents using the OpenSearch framework specific settings and mappings have to be defined. In this project the documents provided are JSON files containing information about recipes. On our implementation we used 4 different types of data properties: to map the recipe_id and the ingredients we used the keyword type, since these index mappings are a string sequence of structured characters; to map the title and description of each recipe the text type was the property chosen, considering that they are string sequences of characters that represent full-text values; to map the title_embedding and the

description_embedding we used the knn_vector type; and finally, to map the number of servings that each recipe suggests we use the Integer data property.

To improve the index mappings on the properties of the text type we included two other parameters: analyzer and similarity. The analyzer refers to the text processing method that we desire to use when running queries on that property. OpenSearch provides several built-in analyzers with each one of these using different character filters, tokenizers and token filters, and in our implementation, we decided to use the standard analyzer. The similarity, the second parameter, refers to the similarity module to be used. Similarity is the process of computing how similar two words, phrases or sentences are, which can be very helpful to tasks like question answering[1]. The default similarity module is the BM25. This module is an improvement over the TF_IDF algorithm, a widely used algorithm in natural language processing, because BM25 takes into account the field length and applies term frequency saturation functions, making the result more relevant to the user's query. Because of these reasons, we decided to go with this module.

As described above the embeddings of the title and the description use the type knn_vector so we are able to perform Approximate k-nn searches . Knn means K-nearest neighbor and Approximate means that for a given search, the neighbors returned are an estimate of the true k-nearest neighbors [4]. To index mappings of knn_vector type we need to detail other required properties, such as the dimension and the method. The dimension property indicates the dimensionality of the indexed vectors and the method property needs to be filled with other specific properties to the indexing method. In our case, the indexing method chosen was the Hierarichical Navigable Small World, or hnsw. This method is an Hierarchical proximity graph approach that does not require training to approximate k-NN searches. The properties specific to the indexing method are: name, space_type, engine and parameters. The name is hnsw as described above. The space_type indicates the similarity function containing the vector space used to calculate the distance between vectors, and we opt with the innerproduct one. The engine property indicates the approximate k-NN library to use for indexing and search and we chose the engine faiss. Finally the parameters property, is divided in two other: ef_construction, which is the size of the dynamic list used during k-NN graph creation and the greater the value we choose here, the more accurate the graph and slower the indexing speed; and 'm' indicates the number of bidirectional links created for each new element, and the increasing of this value can have a large impact on memory[3]. For a balanced result between accuracy and speed we decided to go with the value of 256 on ef_construction property and the value of 48 on the 'm' property.

The majority of the choices we made for the values of the properties described above were based on the examples and documentation provided by the tutor on the workshops.

To add the recipes on the index that we created, we iterated the json document and for each recipe we extracted the title, description ingredients and servings. To be able to create an embedding for the title and description, we used the encode function using a Transformer encoder trained with the MS-

MARCO dataset. After using the encode function on the title and description to create the embbedings, we added them on the index as well.

To search and test our index we implemented multiple based searches: a Text-based Search, a Boolean Text-based Search, a Boolean Search using a filter and an Embedding-based Search. We created a new Python document to create all the search based functions. For the Text-based Search we search a result providing a query and the fields we want in the response. We then created two boolean search, one where we search with a query and given ingredients where the result must have the ingredients provided and should have the query. The other boolean we get a result using a filter for the ingridients and as before, we also should have the given query in the response. Finally we implemented a search using the embedding that we add to the index earlier. We can search using the title embedding or the description embedding.

3 Evaluation

3.1 Dataset Description

Open Search is an engine for analysis and search of data. The data is stored in indices that are collections of JSON documents. In order to initialize it, the indices contains *settings* and *mappings*. The settings give information about the index name, creation date, number of shards and replicas and the mappings describe the collections of fields for the JSON documents that will be indexed [2].

Our index settings has four shards, zero replicas of each shard and a refresh interval of minus one (meaning that the refresh is disabled).

Our choice for the set of index mappings to index the recipes' data are:

- recipe_id *keyword*
- title *text*
- title_embedding *knn_vector*
- description *text*
- description_embedding *knn_vector*
- ingredients *keyword*
- servings *integer*

Recipe_id and ingredients are of type keyword because they are structured sequences of characters, the first is an unique id for each document indexed and the other is a list with all of the ingredients for that receipt. The title and description are both of type text because they represent full-text values that describe in the most generic and simple way the content of the receipt For this type text we used the standard analyzer and the similarity module BM25 that their goal are explained in section 2. The servings is of type integer because it is a 32-bit number that describes the number of people who can enjoy that receipt. The last two, description_embedding and title_embedding, are of type knn_vector (a custom type of data to allow knn-search on OpenSearch, better explained in section

2) and they represent the embeddings of both title and description calculated with Dual-Encoders.

This mappings' choice is directly connected to the natural questions that are going to be asked to OpenSearch. This is, we chose the mappings fields accordingly to the most important information that will most probably be asked in the questions. For example, in the query "How to cook chicken" is of our main interest to search on the title, descriptions and ingredients of the receipt and all of this attributes are indexed on the mappings of the documents. Servings and ingredients are a good addition to use in boolean queries to the data where it can be specified an ingredient or the amount of servings that are needed.On the other side the steps is not a information we need to keep indexed on Open Search because there is no interest in search on that data.

References

1. Author, Dan Jurafsky., Author, James H. Martin.: Speech and Language Processing. 3rd ed.draft. Chapter 6,
2. OpenSearch Documentation - Introduction to OpenSearch, https://opensearch.org/docs/latest/opensearch/index/. Last accessed 7 April 2022
3. OpenSearch Documentation - k-NN Index, https://opensearch.org/docs/latest/search-plugins/knn/knn-index/. Last accessed 8 April 2022
4. OpenSearch Documentation - Approximate k-NN search, https://opensearch.org/docs/latest/search-plugins/knn/approximate-knn/. Last accessed 9 April 2022
5. OpenSearch Documentation - Full-text queries, https://opensearch.org/docs/latest/opensearch/query-dsl/full-text/. Last accessed 8 April 2022
6. OpenSearch Documentation - Boolean queries, https://opensearch.org/docs/latest/opensearch/query-dsl/bool/. Last accessed 9 April 2022
7. OpenSearch Documentation - Term-level queries, https://opensearch.org/docs/latest/opensearch/query-dsl/term/. Last accessed 9 April 2022
8. Chatbots: History, technology, and applications, https://www.sciencedirect.com/science/article/pii/S2666827020300062/. Last accessed 10 April 2022