Semantic Search

Note: The code implementation of the Semantic Search pipeline can be found in Van Gogh repository.

Introduction

Normal search systems only provide you results based on the **content** of the information which has been indexed. What they are missing is the **meaning**. Without understanding the semantics of the information the end-user won't be able to retrieve data based on their meaning.

In the context of semantic search, a user enters a short free-text query, and documents are ranked based on their semantic similarity to the query. Text similarity can be useful in a variety of use cases including, **Question-answering**, **Article search**, and **Multimedia search**.

To be able to index data based on their semantics, we use vector representations which are designed to capture the linguistic content of the text and can be used to assess the similarity between a query and a document.

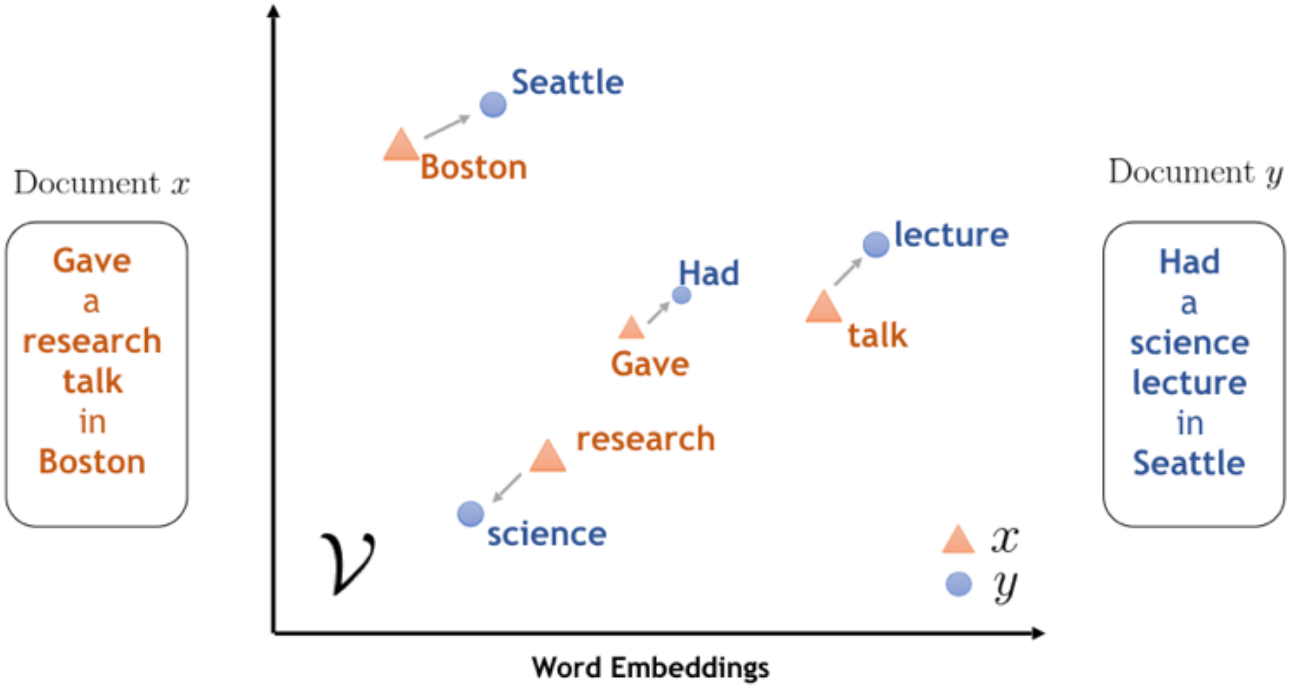
The vector representation of the text is based on a natural language processing (NLP) technique called Word Embedding.

Word embeddings

A **word embedding** model represents a word as a dense numeric vector. These vectors aim to capture semantic properties of the word — words whose vectors are close together should be similar in terms of semantic meaning. In a good embedding, directions in the vector space are tied to different aspects of the word's meaning. As an example, the vector for "Canada" might be close to "France" in one direction, and close to "Toronto" in another.

Most current approaches are based on complex neural network architectures, and sometimes incorporate labeled data during training to aid in capturing semantic information.

Once trained, the models are able to take a sentence and produce a vector for each word in context, as well as a vector for the entire sentence. Pre-trained versions of many models are available, allowing users to skip the expensive training process and are typically fast enough to be used as part of real-time applications. Some common sentence embedding techniques include **InferSent**, **Universal Sentence Encoder**, **ELMo**, and **BERT**.



Indexing Semantic Data

In our pipeline, we first split the text into snippets. Snippets could be either overlapping or nonoverlapping texts and can comprise of a few sentences or paragraphs.

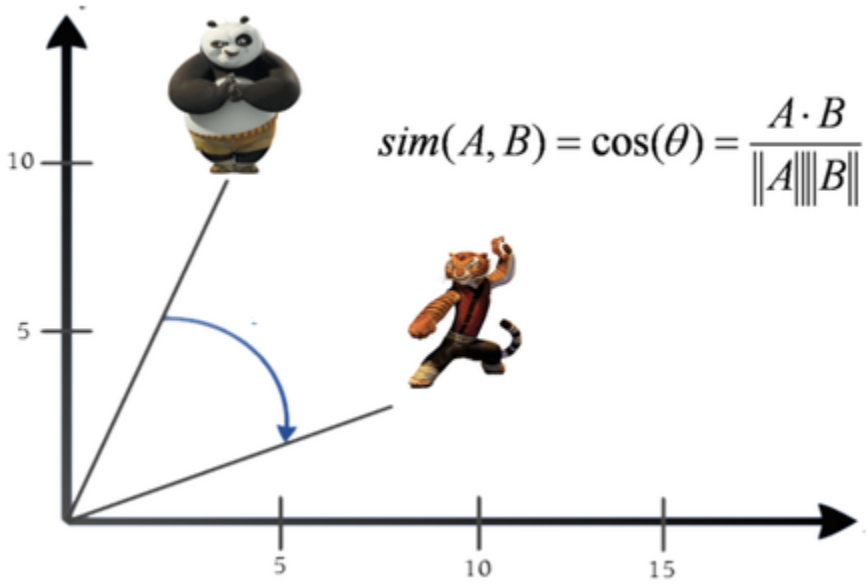
Then we use and embedding model to map each snippet to a corresponding vector representation. For the model either we can use off the shelf models such as **Universal Sentence Encoder** as-is or, fine-tune those models.

Then, we create the Elasticsearch index, which includes those snippets, vector representations, and corresponding metadata such as document titles, tags, and etc.

In the mapping for the embedding vector, the dimensions of the vectors should be fixed across all documents.

Retrieving Semantic Data

When a user enters a query, the text is first run through the same embedding model and stored in the parameter `query_vector`. Elasticsearch provides a `cosineSimilarity` function in its native scripting language. So to rank questions based on their similarity to the user's query.



The following script can retrieve documents based on their semantic similarity to the query:

```
{
  "script_score": {
    "query": {"match_all": {}},
    "script": {
      "source": "cosineSimilarity(params.query_vector, 'title_vector')
+ 1.0",
      "params": {"query_vector": query_vector}
    }
  }
}
```

Conclusions

Embedding techniques provide a powerful way to capture the semantic of each document. By indexing embeddings and scoring based on vector distance, we can compare documents using a notion of similarity that goes beyond their word-level overlap.

Note: The code implementation of the Semantic Search pipeline can be found in Van Gogh repository.