410 Technology Review

Tianhao Luo
NetId: tluo3

Introduction

This is a summary of the 2020 paper, *Embedding-based Retrieval in Facebook Search.*

Search in social networks (Facebook, Twitter, TikTok etc.) has a unique challenge that classical web search does not have: it needs to take into account both the query text and searcher's context to provide relevant results. Users' social graph is a crucial part of this context and is a unique aspect of Facebook's search. Although embedding-based retrieval (EBR) had been widely used in web search engines for many years at the time (2020) the author published this paper, Facebook search was still mainly using Boolean matching model. In their paper, the author proposed a framework of how to unify the EBR framework to model semantic embeddings of personalized search, and the system is based on an inverted index in a typical search system. The authors evaluated EBR on verticals (search results based on result type, i.e. people, page, group etc.) and found significant metric gains observed in online A/B experiments.

Methodologies

Embedding, also known as representation learning, has been a successful technique contributing to the success in fields such as speech recognition, computer vision and natural language understanding. In short, embedding is a way to represent a sparse vector of ids as a dense feature vector. In essence, we try to learn the numerical representation of search queries (as a vector), and then convert the information retrieval problem into a nearest neighbor (NN) search problem in the embedding space.

In general, a search engine comprises two layers (stages). The first layer, *retrieval*, uses some simple models (logistic regression) to retrieve several thousand (possibly) relevant documents from a potential pool of millions of them. The objective of this step is to prepare a pool of documents for the next layer, which is called *ranking*. Because the system had millions of documents to start with at the beginning of retrieval and low latency is key, the results of this layer do not need to be very precise. Then at the ranking layer, we usually use more sophisticated algorithms such as deep learning to 'fine tune' the results. EBD can be used in both layers but more commonly used in the retrieval layer.

Model

The authors formulate search retrieval task as a recall optimization. They wanted to maximize recall by the top K results, i.e. recall@K.

The query and documents are encoded with a deep learning into dense vectors (embeddings), on which the authors use cosine similarity as a distance metric. They proposed to use triplet

loss (to be explained later) to approximate the recall objective to learn the neural network encoder, which is also known as embedding model.
The traditional semantic embedding which is widely used in search engines such as Google and Bing, is not sufficient in Facebook's use case. Suppose a user is search for "John Smith" in people search, the actual target person that the Facebook search engine should return should be the "John Smith" that is likely to be that person's friend or acquaintance. The authors proposed to use unified embedding which considers not only text but also user and context information in deriving embeddings.

The triple loss function mentioned earlier is defined as follows. For triplet (q(i), d(i)+,d(i)-) where q(i) is q query, and d(i)+ and d(i)- are the associated positive and negative documents, the triplet loss function is defined as

$$L = \sum_{i=1:N} max(0, D(q(i),d(i)+) - D(q(i),d(i)-)+m)$$

The intuition behind the triplet loss function is that it would be ideal to separate positive examples and negative examples by a distance margin. The distance margin, m, is this case is a tuning parameter.

To learn embeddings that are optimizing the triple loss, the authors model have three main components

1. A query encoder which produces a query embedding
2. A document encoder which produces document embedding
3. A similarity function between (query,document) pairs

A neural network that is used to transform an input into a lower-dimensional dense vector is also known as an embedding.

What the authors added on top of traditional embedding method is that they added location, user social connections on the user side, On the document side, they added information such as aggregated location and social clusters about a Facebook group.

Most of additional features are categorical features of high cardinality, which could be either one-hot or multi-hot vectors. For one-hot categorical variables, the authors created an embedding look-up layer. For multi-hot categorical variables, they used a weighted combination of multiple embeddings.

Some feature engineering techniques:

Text features: the authors used character n-gram to represent text embedding. Why not word n-gram?

1. Due to its very limited vocabulary size (for English, only 26. For other languages, a bit more but won't be huge either), the embedding lookup table has a small size
2. This representation is robust to out-of-vocabulary problem (e.g. spelling variations or errors)
3. The authors used both character n-gram vs word n-grams and found the former works better

Location features

The authors added location features into both query and document side features. For query side, they extracted searcher city, region, country and language. For document side, the added information such as group location tagged by admin.

Social embedding features

The authors trained a separate embedding model to embed users and entities based on social graph.

Conclusions

1. Information retrieval algorithms on social network is a difficult problem, since it needs to take user-specific and contextual information into consideration
2. It is also challenging because of modeling difficulty, system implementation and cross-stack optimization complexity
3. There will be lots of opportunities in this field

References

Huang, Jui-Ting, et al. "Embedding-based retrieval in facebook search." *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020.