Personalised Semantic Search Competition

NI-ADM

Antonín Kříž

2024-05-31

1 Introduction

The goal of this competition is to build a personalised semantic search recommendation system. Based on the training dataset with $7,761,370$ entries of `UserID (int), Query (text), ItemID (int)` triplets of $100,000$ unique users, $39,976$ unique items and $59,999$ unique queries, the system should recommend top 100 items given a `UserID (int)` + `Query (text)` pair. The result is evaluated using Mean Reciprocal Rank[1] (MRR) of the correct items that should've been predicted the first.

2 Approach

The solution I came up with was in Python, combining the power of Sentence Transformers[1][2, 3] and ELSA[4] - Scalable Linear Shallow Autoencoder for Collaborative Filtering. I used Sentence Transformers to compute the Semantic Textual Similarity (STS) of the input query and the queries in the dataset, extracted the top-K most similar queries and the items linked to these queries. Then ELSA was used to compute the item embeddings to calculate the recommendation score of each of these items, which decided, together with the weighted similarity of the query linked to the item, the final order of the recommended list of items. The approach is summarised in the pseudo code bellow.

```
A = ELSA.get_embeddings()
X = DATASET.load_interaction_matrix()

for user, query in test_dataset:
    encoded_query = STS_TRANSFORMER.encode(query)
    distances, items = INDEX.find_top_k(k, encoded_query)
    items = items[:limit]
    distances = distances[:limit]
    w = [distances[item] · weight for item in items]
    x = X_{user,:}
    order_score = ((xAA^T) − x)_{items}
    results.append(
        items.sort_by_score(w + order_score)[:100]
    )
```

Listing 1: Pseudocode of the approach

This model has multiple hyper-parameters, which need to be optimised. The optimisation procedure will be discussed in the next chapter.

[1] `https://www.sbert.net/`

3 Challenges

The approach above comes with multiple challenges. First challenge comes with the interaction matrix. Since the matrix has a dimension of $100,000 \times 40,000$, the memory requirements to just store it are over 14.9 GB (implying the use of float32). Although this is still manageable on my setup[2] when using CPU for computations, it isn't possible to fit in GPU VRAM. This leads to the usage of sparse matrices, since the interaction matrix is mostly empty ($\tilde{9}9.81$ %), this also comes with the benefit of faster computation. Sparse matrices come with another set of problems, mainly with far-from-perfect support in PyTorch[5]. For example, PyTorch does not support indexing of sparse matrices with arrays, which leads to the need of building sparse selection matrices (an identity matrix, which contains only rows of requested columns), and arithmetic operations, which requires careful conversion between sparse and dense tensors.

Next challenge is the hyper-parameter optimisation. The model uses following hyper-parameters with given values:

- ELSA
  - `n_dims` Number of factors of the latent-space
    $[64, 128, 192, 256, 512, 1024]$
  - `batch_size` Number of items processed at once
    $[64, 128, 256, 512]$
  - `epochs` Number of epochs
    $[3, 4, 5, 6, 7, 9, 11, 15]$
- STS Encoder
  - `model` What model should be used
    - `sentence-transformers/all-mpnet-base-v2`[3]
    - `mixedbread-ai/mxbai-embed-large-v1`[6, 7]
    - `nomic-ai/nomic-embed-text-v1.5`[8]
    - `nomic-ai/nomic-embed-text-v1`[8]
    - `Alibaba-NLP/gte-large-en-v1.5`[9]
- Prediction
  - `k` Number of most similar sentences to fetch from the index
    $[165, 200]$
  - `weight` Weight to scale the sentence distance with
    from 0 to 200 with the step size of 5
  - `limit` Cutoff limit of the number of items to process
    $[150, 350]$

To find the optimal values, I searched the space of possible combinations of each parameter with incrementally focusing on more promising examples based on the MRR score on 0.9 to 0.1 train-test split to reduce the total number of possible combinations from millions to just over $1,300$. Searching through this number of combinations would not be possible without optimisations mentioned in the beginning of this sections, which allowed me to finish a single run of the model in less than one and a half minute.

The final challenge was the limit on the number of allowed submissions per day in the system, so had to selected just a few results which seemed as the most promising ones. The hyper-parameter tuning process was verified using the competition system on the 2nd batch. These "validation" submissions were done to prevent over-fitting on the training dataset. Because of large number of common queries in test and train split, the hyper-parameter tuning preferred models with large `weight` parameter, but the 2nd batch preferred lower weights in the range of 30 to 70.

[2] Intel i7-11700, 80 GB RAM, NVIDIA RTX 3070

4 Results

The hyper-parameter tuning led to the selection of my final model:

ELSA `n_dims`	128
ELSA `batch_size`	64
ELSA `epochs`	5
STS `model`	`nomic-ai/nomic-embed-text-v1`
Prediction `k`	165
Prediction `weight`	65
Prediction `limit`	150

Table 1: Final model hyper-parameters

This model's MRR score was 0.3487, 0.3444 and 0.3458 on the 2nd batch dataset, leading to an average of 0.3463 and over 0.8043 on the test split.

The final code of this model, including it's hyper-parameter optimisation results, is available at `https://github.com/antoninkriz/ADMv2`.

References

[1] Wikipedia. *Mean reciprocal rank — Wikipedia, The Free Encyclopedia*. `http://en.wikipedia.org/w/index.php?title=Mean%20reciprocal%20rank&oldid=1218582630`. [Online; accessed 31-May-2024]. 2024.

[2] Thomas Wolf et al. *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. 2020. arXiv: `1910.03771 [cs.CL]`.

[3] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019. URL: `https://arxiv.org/abs/1908.10084`.

[4] Vojtěch Vančura et al. "Scalable Linear Shallow Autoencoder for Collaborative Filtering". In: *Proceedings of the 16th ACM Conference on Recommender Systems*. RecSys '22. ¡conf-loc¿, ¡city¿Seattle¡/city¿, ¡state¿WA¡/state¿, ¡country¿USA¡/country¿, ¡/conf-loc¿: Association for Computing Machinery, 2022, pp. 604–609. ISBN: 9781450392785. DOI: `10.1145/3523227.3551482`. URL: `https://doi.org/10.1145/3523227.3551482`.

[5] PyTorch. *Torch.sparse*. URL: `https://pytorch.org/docs/stable/sparse.html`.

[6] Sean Lee et al. *Open Source Strikes Bread - New Fluffy Embeddings Model*. 2024. URL: `https://www.mixedbread.ai/blog/mxbai-embed-large-v1`.

[7] Xianming Li and Jing Li. "AnglE-optimized Text Embeddings". In: *arXiv:2309.12871* (2023).

[8] Zach Nussbaum et al. *Nomic Embed: Training a Reproducible Long Context Text Embedder*. 2024. arXiv: `2402.01613 [cs.CL]`.

[9] Zehan Li et al. "Towards general text embeddings with multi-stage contrastive learning". In: *arXiv:2308.03281* (2023).