How to Prepare Data to Build a Text Embedding Model

Danmei Xu · Follow
Published in Snowflake · 7 min read · 7 hours ago

Coauthors: Gaurav Nuti, Luke Merrick, Daniel Campos

Embedding models are one of the crucial building blocks of modern AI — powering everything from Internet search to RAG applications — but how are they made? Our previous article described how to *train* a state-of-the-art embedding model but left out an equally important secret: **how do you get the data used to train your state-of-the-art model?** Although data quality is one of the most important factors impacting model quality, few embedding models leading the MTEB leaderboard have released their dataset recipes, so much remains to be learned from firsthand experience. Luckily, we have that experience at Snowflake and can share some tricks here — not only do we have infrastructure to handle 4 billion queries daily [15] but also have rich crawled data through our acquisition of Neeva! By leveraging the above, we have built state-of-the-art (SoTA) retrieval, ranking, and embedding models and deployed them for customer usage!

As described in our previous article, training an embedding model requires two basic stages. In the first stage, we train using large batches (over 16,000 samples per batch) of weakly supervised pairs. Here, the key data challenge is to achieve scale and diversity. Given the size of batches and the broad understanding you want to imbue the model with, you should look to have at least a few hundred million pairs. Moreover, these examples should include general domain topics such as factoid queries," e.g., "How tall was Abraham Lincoln?" and domain-specific queries in areas like science or finance, "Do preoperative statins reduce atrial fibrillation after coronary artery bypass grafting?"

In the second stage, we train with a small, focused corpus with mined hard negatives presenting different data challenges. You cannot afford to just have hundreds of millions of samples, as such high quality data usually requires some amount of human input. Plus, it will make training prohibitively long.

Woven throughout these challenges is a deeper, systems-level challenge: the sheer quantity of data we must process. Here, we turn to Snowflake to leverage the easy-to-understand and trivial-to-scale infrastructure to power all of our extraction and processing of candidate datasets.

Problem #1: How to obtain massive query-document pairs?

During the first portion of training, the goal is to obtain as many high-quality positive pairs (text pairs close in semantic meaning) as possible across several important domains (web search, science, etc). While there are already many curated sources of such data, we find that a few extra steps of data processing can greatly benefit downstream model quality (i.e. >10% score improvement from a combination of these tricks!)

Trick #1: Sprinkle in Web Search Data

While one can focus on finding and processing existing domain-specific data, the licenses on these datasets, their provenance, and their scale can make unification and usage difficult. As a simple alternative, we find that data from web searches contains incredible domain-specific content without manually sourcing or filtering. In our experiments, this addition served as a coveted spice in cooking. A little goes a long way and provides one of the biggest lifts in model performance. When we add in web search data, we actually find that the common practice of making pseudo query document pairs by using title and page content is suboptimal. Instead, we find using actual queries matched with the page content to be much better. In our experiments, we mined such query document pairs from web crawl data by looking at the crawled pages associated with results that users clicked after entering specific queries. Once you do a primary extraction from your web search corpus you should be left with a few billion pairs which are like the samples below. Some of these samples are going to be bad, as is the nature of any large scale data sampling regime.

```
# This is a good example of positive pair.
Query: randy's fishmarket
Document: In the Know: Randy's Fishmarket closes; new seafood restaurant to laun
Its full name was Randy's Fishmarket Restaurant & Tap Room, but folks just knew
# This is a bad example of positive pair (from failed crawl) that needs to be pr
Query: stardust chronicle spark dragon rarity
Document: We're sorry but TCGplayer doesn't work properly without JavaScript ena
```

Massive Quality Filter

We cannot go ahead and box up all web search queries and call it a day. Most large-scale data is noisy, and web data is no exception. With the advent of

LLM data pipelines such as RedPajama-Data-v2, we have witnessed the power of quality filters and their downstream effect on model quality (the adage of "garbage in, garbage out"). For positive data pair cleaning, we need to ensure: a) each text in the pair is of good quality and b) text pairs (query, document) are similar in meaning. For a), we leverage existing knowledge on quality signal annotation as detailed in the ReadPajama-Data quality annotation step. For b), we apply a low-fidelity, high-throughput pair-similarity filter — sentence similarity using fasttext word embeddings. You read right, you can use other embedding models to help you filter your data to train your embedding model! We are not looking to use existing embeddings to have a strong label, but instead, we annotate as a weak proxy, filtering out things that are very far apart. Once you have these annotation signals, your entire quality filter step can be one simple SQL in Snowflake-like (read: running under minutes using a large warehouse cleaning a few billion rows of data)!

```
CREATE TABLE your_filtered_table AS
SELECT ...
FROM ...
WHERE language = $target_language
  -- consistency filter
  AND quality_signals:query_similarity::ARRAY[0][2] > $q2d_threshold
  -- number of normalized words in doc
  AND quality_signals:rps_doc_word_count::ARRAY[0][2] > $doc_word_count_low
  AND quality_signals:rps_doc_word_count::ARRAY[0][2] < $doc_word_count_high
  -- mean length of words after normalization
  AND quality_signals:rps_doc_mean_word_length::ARRAY[0][2] > $doc_word_len_low
  AND quality_signals:rps_doc_mean_word_length::ARRAY[0][2] < $doc_word_len_high
  -- fraction of lines that end with an ellipsis
  AND quality_signals:rps_doc_frac_lines_end_with_ellipsis::ARRAY[0][2] < $ellip
  -- fraction of words that contain no alphabetical character.
  AND quality_signals:rps_doc_frac_no_alph_words::ARRAY[0][2] < $alph_threshold
  -- perplexity score
  AND quality_signals:kenlm_perplexity::FLOAT < $perplexity_threshold
  -- density of line start with bulletpoint
  AND bulletpoint_sum.density < $bullet_threshold
```

```
    -- many other related quality filters
  ...
```

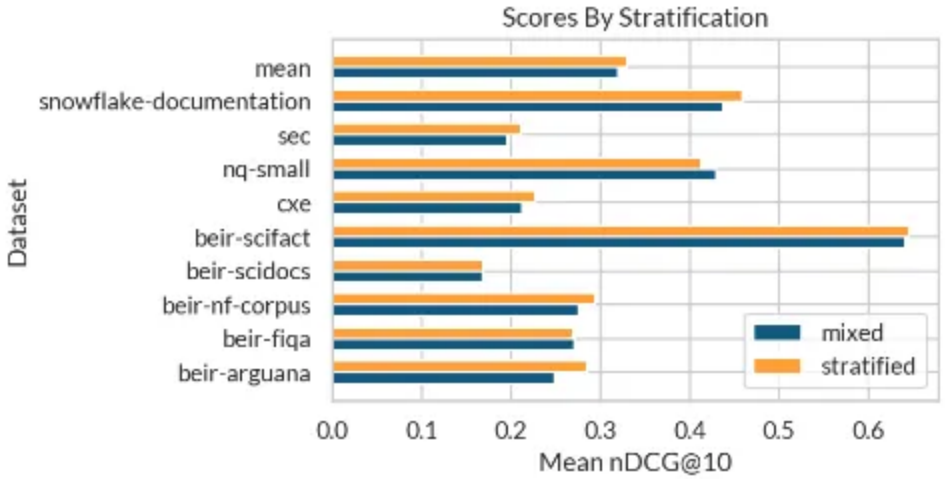
Problem #2: How to improve utility of large scale in batch negatives?

In our first step of training, we only leveraged in-batch negatives. This means that our batches contain only positive query-document pairs, and we assume that every other document in the batch can be a negative example for each query. Being this signal is a weak but large scale, it is critical to maximizing the usefulness of these negative examples.

Trick 2: Pre-shuffle your data by source.

With clever training code, we first scale the batch size as big as we can fit, often over 30,000 examples per batch, to maximize the number of in-batch negatives, hoping that more of these will teach the model about performing quality searches. While the age-old wisdom in training steers towards the widespread use of randomness, we found it quite important to modify this sampling approach such that each batch only has samples from the same data source. The instinct here is that irrelevant in-domain documents will be harder for the model to down-rank than irrelevant out-of-domain documents, e.g., it's easier to know that a social media post is irrelevant to a scientific domain query than an irrelevant scientific paper. In Snowflake, it's easy to pre-shuffle and pre-stratify your data so that within each batch, the source of positive pairs is the same dataset, making embedding models learn faster and better:

```
-- GID is precomputed batch id based on your batch size within each dataset.
-- Below SQL ensures that globally, contiguous batches come from randomized data
CREATE OR REPLACE TABLE ... AS (
  WITH BATCH_GROUP_ORDER AS (
    SELECT GID, ROW_NUMBER() OVER (ORDER BY RANDOM($rand_seed) DESC) AS batch_or
    FROM ...
    GROUP BY GID
  )
  select
    ...,
    s.gid AS GROUP_ID, b.batch_order AS BATCH_ORDER
  from ... s JOIN BATCH_GROUP_ORDER b on s.gid = b.gid
  ORDER BY b.batch_order
);
```



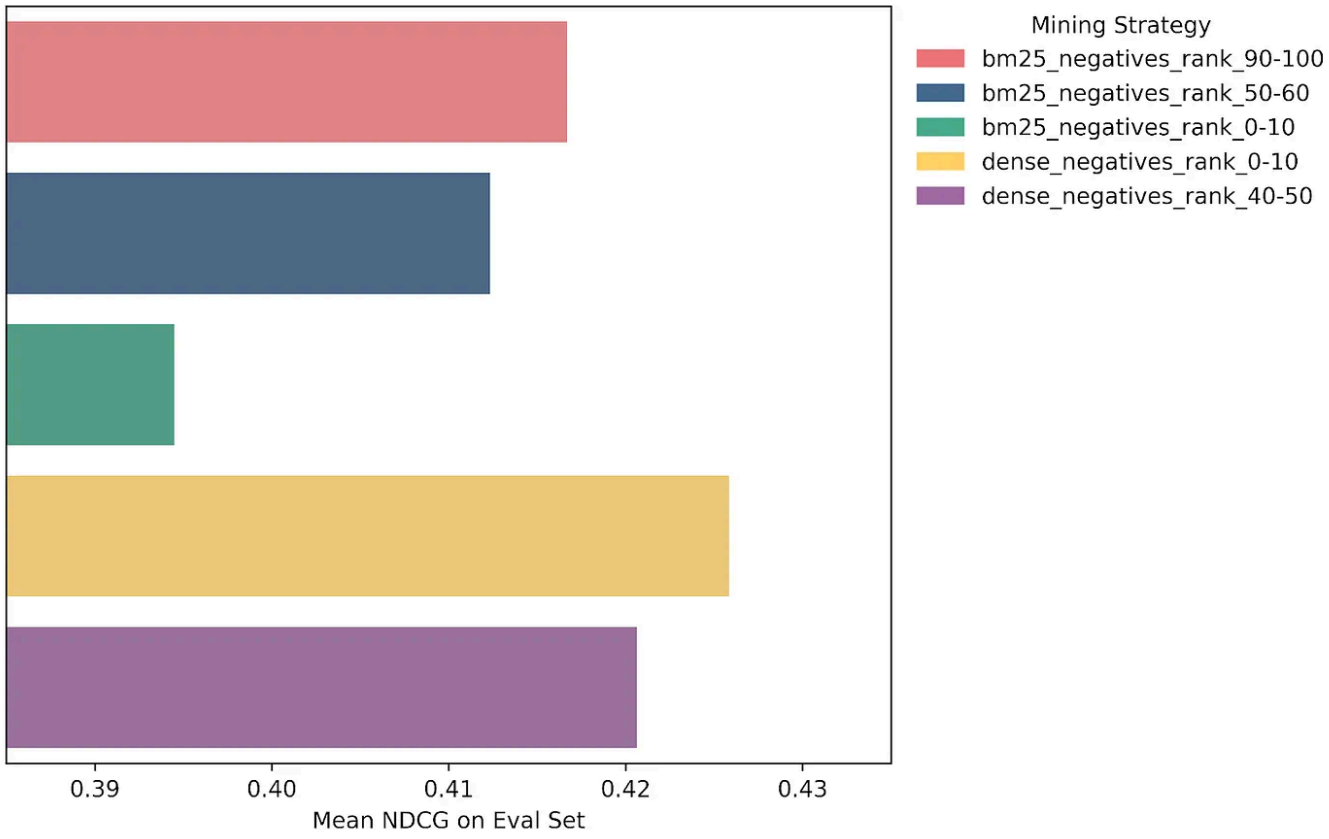
Problem #3: How to improve the quality of massive query-document pairs?

During hard negative-minded training, the goal is to obtain high-quality triplets (text, positive_texts, negative_texts) for contrastive learning. Unlike earlier stages of training, where we infer negatives in a batch, we must ***mine*** to have negative pairs with a strong signal. Many strategies exist for mining negatives from an existing corpus of texts, including easier strategies like BM25 and harder strategies like reranking or fine-tuned models as

instructors. The key question here is how to compare your mining strategies for better results.

Trick #3: Iterate on your negative mining strategy!

Instead of choosing an arbitrary mining approach that assumes a single miner or mining parameters, make sure to study your approach to validate your parameter choices. First, choose a small, semantically dense corpus that is representative of your workload but is relatively inexpensive to train and validate. Next, go ahead and create many variants of each dataset, varying the retriever, the depth of retrieval (Top 10, Top 10–20, etc.), and the number of negatives you use. As shown below, we can see that variations in sampling methodology, negatives used, and sampling model can substantially impact model quality.



Conclusion

While the loss function, model architecture, and all other parameters may be what gets headlines in AI press releases, how you source, sample, clean, and process your training data is incredibly important. At Snowflake, we have found that having quick, accurate, and effective ways of doing this is paramount to our march to create a world-class embedding model. We look forward to sharing more exciting news on our embedding model as part of Snowflake Cortex service in the weeks to come.

Come Define the future of AI with us

Snowflake's AI technology supports some of the world's most consequential enterprise AI workloads. Help us push the state of the art of enterprise AI at http://snowflake.com/careers

Acknowledgments

We thank Adrien Treuille for his helpful feedback in editing this article.

Text Embedding | Data Infrastructure | Snowflake