Fine-tuning a multi-language model for information retrieval using the Solr Manual

By Elpidio Gonzalez

Summary

We fine-tuned a pre-trained multilanguage SBERT model on the Solr Manual to test the feasibility of having a semantic search engine capable of retrieving documents written in a particular language, in this case, English, using non-english queries.

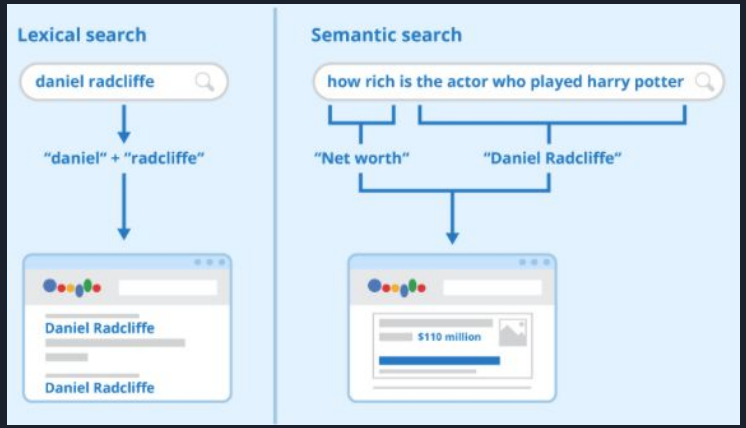
Agenda

- Semantic Search Overview
- Pretrained Models
- Solr Manual as a Dataset
- Fine-Tuning a Model
- Model Evaluation
- Solr Schema and Indexing
- Improvements
- Demo

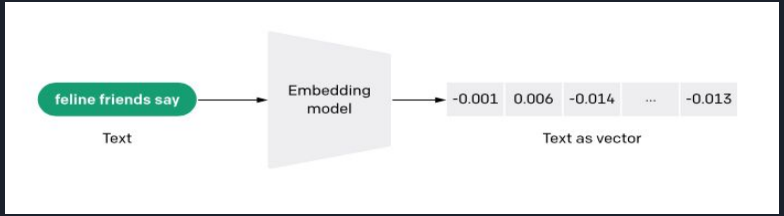
Semantic Search

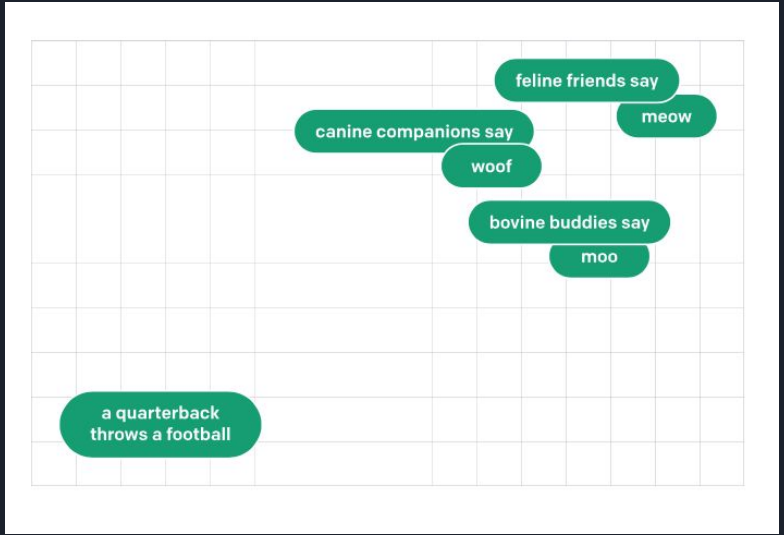
Semantic search is a data searching technique that aims to understand the meaning and context of words and phrases in a search query, as opposed to just matching keywords.

This means that it can provide more relevant and accurate results to a user's query, even if the search terms used aren't an exact match to the content being searched.



Document Representation





Semantic Search in Solr

In the context of Solr, Semantic Search if referred to as Neural Search.

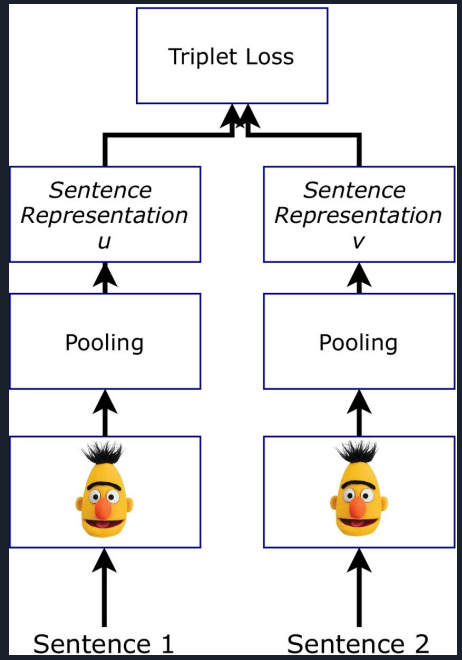
The first implementation was contributed to the Solr open source community by Sease in January 2022 thanks to the work of Alessandro Benedetti and Elia Porciani.

This implementation relies on the Apache Lucene implementation for K-nearest neighbor search to provide:

- A Dense Vector Field type : To store text embeddings
- A KNN Query Parser to run knn search on a target vector field : To compare query - document similarity.

Pretrained Models

Pretrained text models are deep learning models that have been pre-trained on large amounts of text data to learn the underlying patterns and relationships between words and phrases. These models are trained using techniques such as neural networks, and are designed to capture the meaning and context of text data. As such, they are the perfect tool to calculate a document's text embeddings. An example of such models is SBERT.



Pre-trained Model Advantages

- Time and Resource Savings
- High Performance
- Transfer Learning
- Accessible and Open Source

Pre-trained multi-lingual model

For our particular task we used the `sentence-transformers/paraphrase-multilingual-mpnet-base-v2` model available on Hugging Face.

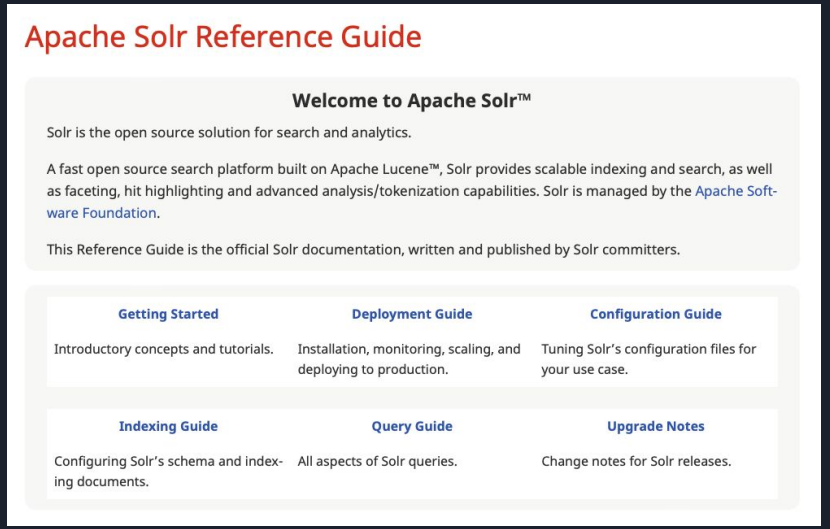
This particular model was trained using over a dozen datasets of parallel data (multiple languages), notably WikiMatrix, a dataset of 135M Parallel Sentences in 1,620 Language Pairs from Wikipedia.

This model maps sentences & paragraphs to a 768 dimensional dense vector space and can be used for tasks like clustering or semantic search.

Solr Manual

The Solr manual is a comprehensive guide that provides detailed information on how to install, configure, use, and maintain the Solr search platform.

It is the official Solr documentation, written and published by Solr committers, presented in adoc format.



Adoc Format

The AsciiDoc (adoc) format is a lightweight markup language that is used to write technical documentation.

Some of the features of the AsciiDoc format include:

- Headings and subheadings
- Lists and tables
- Links and cross-references
- Images and diagrams
- Source code blocks with syntax highlighting

Cleaning the Documents

- Remove Code Snippets, Hyperlinks and Images
  - Code snippets ([source, ...])
  - Links (xref)
  - Images (image::)
- Remove License Notice
- Leave Tables
- Use heading and subheading syntax to split documents.
  - Heading (=Heading)
  - Sub-heading 1 (==Heading)

Splitting the Manual

The root document 'Field Type Definitions and Properties would be split into:

- Field Type Definitions and Properties / Field Type Definitions in the Schema
- Field Type Definitions and Properties / Field Type Properties / General Properties

...

- Field Type Definitions and Properties / Choosing Appropriate Numeric Types

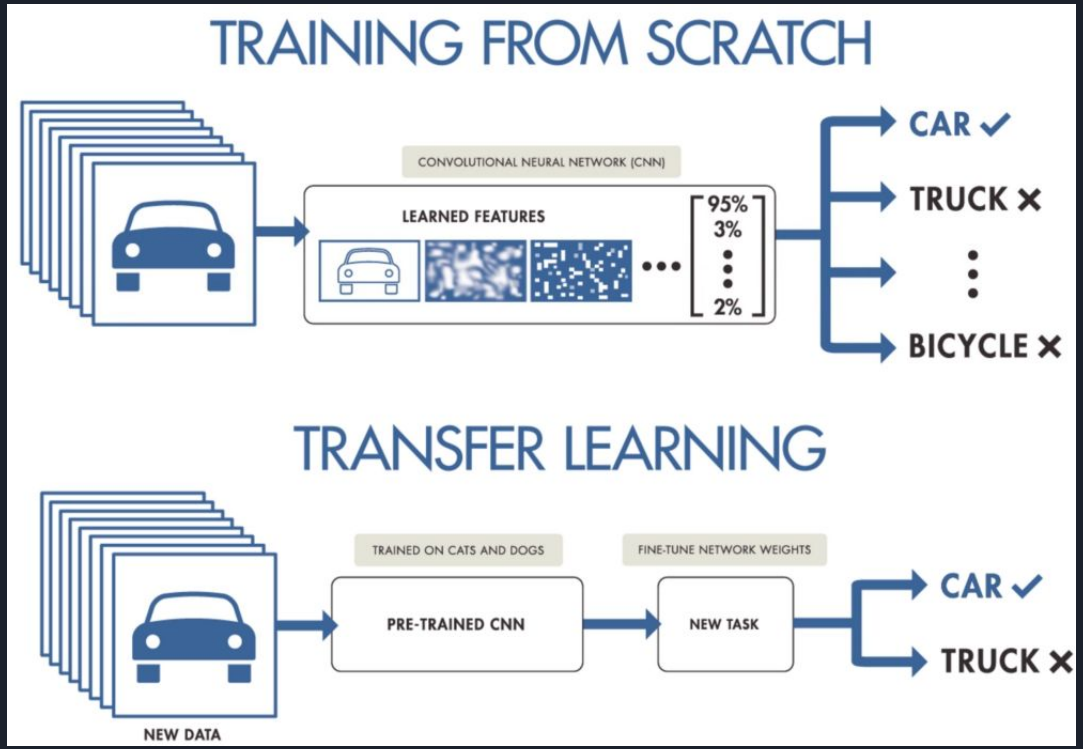
...

This data processing resulted in a total of 2,972 documents.

Faceting


SOLR_CLASSES
Cluster Field Node XML Replica Shard Overseer SolrConfig Package RequestHandlers
Resource SearchComponent HighlightComponent Metrics SolrClient Suggester Type AnalyticsComponent
CloudSolrClient ClusteringComponent Command ConcurrentUpdateHttp2SolrClient ConcurrentUpdateSolrClient
CoreContainer DebugComponent EmbeddedSolrServer ExpandComponent FacetComponent FacetModule
Http2SolrClient HttpSolrClient LBHttp2SolrClient LBHttpSolrClient MoreLikeThisComponent
PhrasesIdentificationComponent QueryComponent QueryElevationComponent RealTimeGetComponent
ResponseLogComponent SpellCheckComponent StatsComponent SuggestComponent TermVectorComponent
TermsComponent Token Cache CloudHttp2SolrClient ClusterState CollectionsHandler ConfigSetService
CoreAdminHandler ExactStatsCache FileSystemConfigSetService HttpShardHandlerFactory IndexSchema InfoHandler
InitParams Metric MetricsMap NamedList QParser QParserPlugin QueryParser QueryResponse
SearchHandler SolrCore SolrGraphiteReporter SolrJmxReporter SolrMetricReporter SolrParams SolrQuery
SolrRequest SolrResponse SolrSlf4jReporter ZkConfigSetService

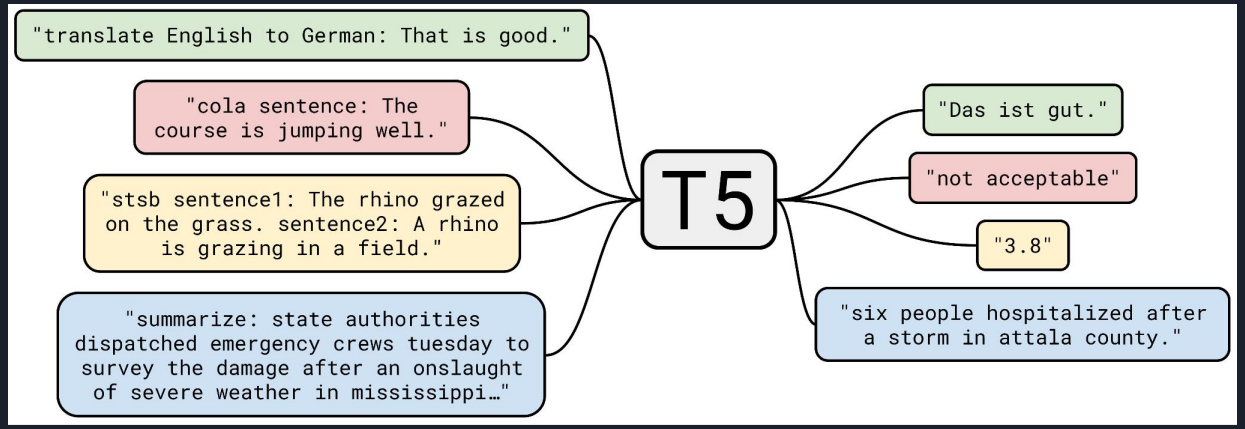
Fine-Tuning


TRAINING FROM SCRATCH
CONVOLUTIONAL NEURAL NETWORK (CNN)
LEARNED FEATURES
95%
3%
2%
CAR ✓
TRUCK ✕
BICYCLE ✕
TRANSFER LEARNING
TRAINED ON CATS AND DOGS
FINE-TUNE NETWORK WEIGHTS
PRE-TRAINED CNN
NEW TASK
CAR ✓
TRUCK ✕
NEW DATA

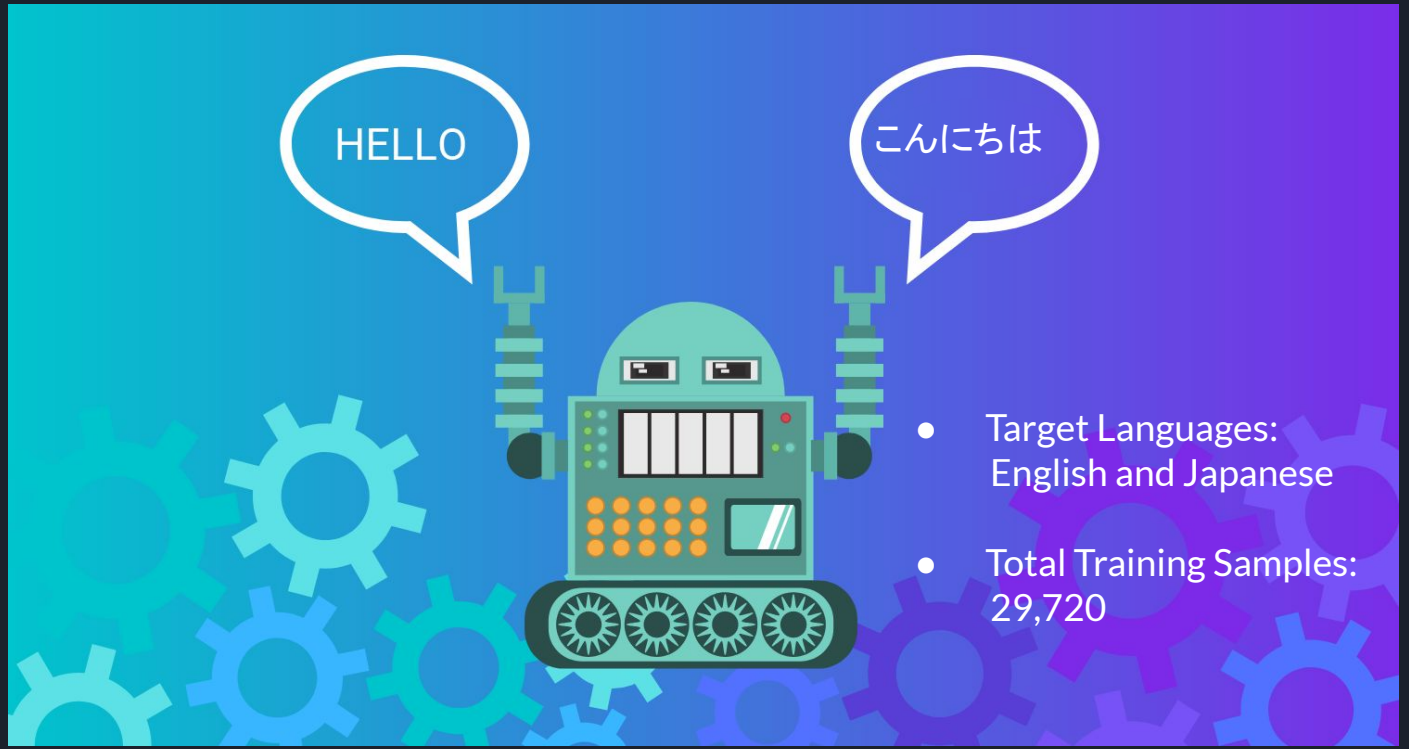
Query Generation

BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models proposed using synthetic queries for unsupervised domain-adaptation approach for dense retrieval models.

We generated 4 extra queries per document and used the original title as a fifth one.



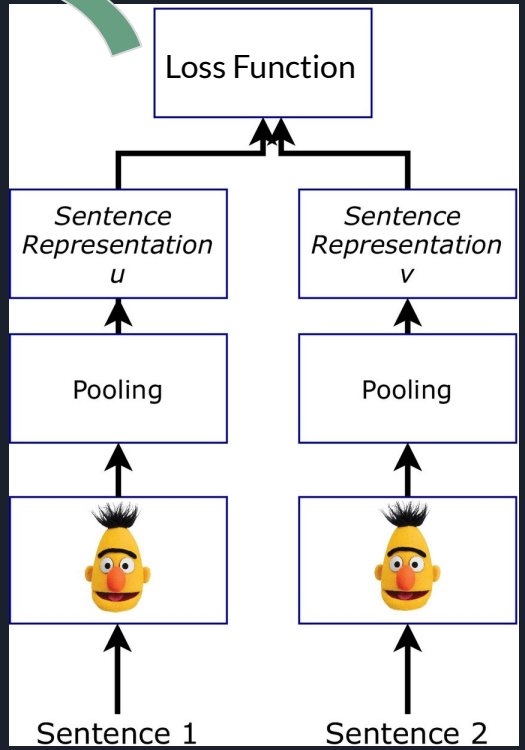
Translation



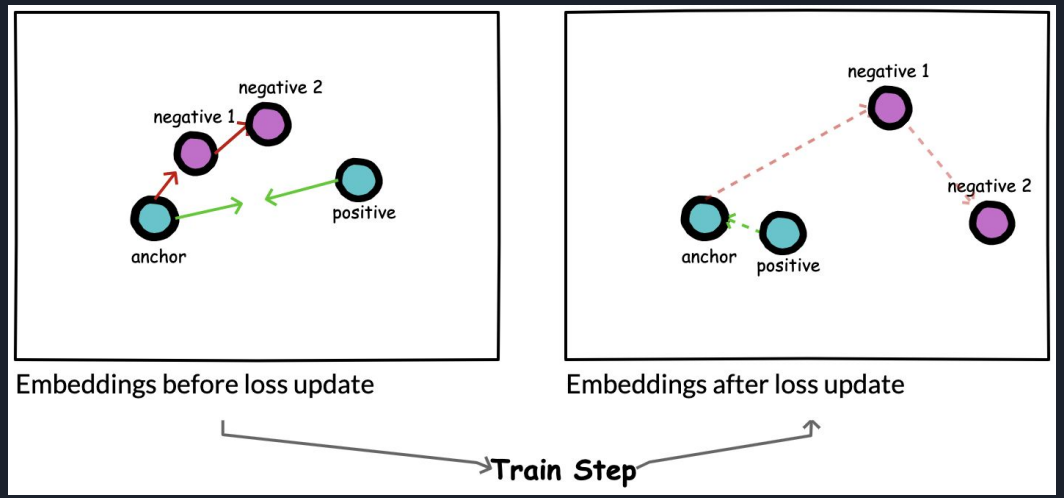
- Target Languages: English and Japanese
- Total Training Samples: 29,720

Fine-tuning a Bi-encoder model

CosineSimilarity?
MultipleNegativesRanking?
Triplet?



Multiple Negatives Ranking Loss

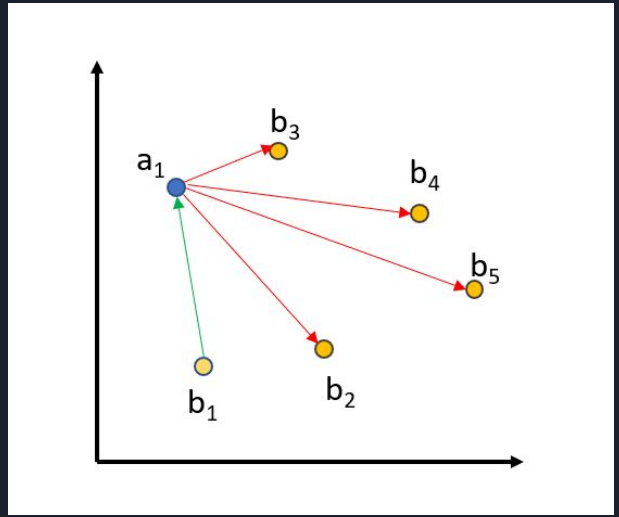


Multiple Negatives Ranking Loss

For each query (a_i) the MultipleNegativesRankingLoss class will create negative samples by randomly choosing non-positive documents associated with the other queries b_j (j != i). So , a batch of samples in a training step would look like:

$(a_1 - b_1)$
$(a_2 - b_2)$
$(a_3 - b_3)$
...
$(a_n - b_n)$

$(a_1 - b_1)$ $(a_2 - b_1)$ ... $(a_n - b_1)$
$(a_1 - b_2)$ $(a_2 - b_2)$ $(a_n - b_2)$
$(a_1 - b_3)$ $(a_2 - b_3)$ ... $(a_n - b_3)$
... ...
$(a_1 - b_n)$ $(a_2 - b_n)$ $(a_n - b_n)$



Hard negatives can be introduced by structuring the samples like this: $(a_1 - b_1 - n_1)$

Evaluation

To evaluate our model we use the InformationRetrievalEvaluator in the SentenceTransformers library. It provides a way to evaluate the performance of a model on an information retrieval task. It computes standard IR metrics such as recall, precision, and mean average precision (MAP).

- **Input Data** : A list of queries and a corresponding list of documents for each query.
- **Model Inference**: The model is used to encode the queries and documents into fixed-length vector representations.
- **Similarity Scoring**: A similarity function such as cosine similarity or dot product. The similarity scores are used to rank the documents.
- **Metrics Calculation**: The ranked list of documents for each query is compared to the ground truth relevance labels to calculate metrics such as recall, precision, and MAP.
- **Output**: A dictionary containing the computed metrics, including recall, precision, and MAP.

Results

Metric	Value	Metric	Value
Accuracy @ 1	0.890	Recall @ 1	0.890
Accuracy @ 3	0.993	Recall @ 3	0.993
Accuracy @ 5	0.999	Recall @ 5	0.999
Precision @ 1	0.890	NDCG @ 10	0.956
Precision @ 3	0.331	MRR @ 10	0.941
Precision @ 5	0.199	mAP@ 100	0.941

The small print about the results...

The results above look very good because the evaluation is run on the entire corpus. We didn't work with a train/test split because ...

1. *The manual will not change drastically over time.* The existing documents might change a bit, but we can consider them pretty much static.

   New documents will definitely be added over time, but...

2. *Re-training the model using new data is not a computationally expensive operation.* The corpus is very small. Also, we can re-use the generated queries and generate new ones only for any new additions to the corpus. Given that the training can be finished in under 5 hours (20 epochs @ Google Colab), we can update the model every time a new Solr version comes out.

Basically, we don't care about overfitting the model (high variance, low bias), because the model won't work with unseen data at all.

Improvements

1. Symmetric Vs. Asymmetric Search Models
2. Better Synthetic Queries
3. Real User Queries

Apache Solr Dense Vectors

The dense vector field gives the possibility of indexing and searching dense vectors of float elements. e.g. `[1.0, 2.5, 3.7, 4.1]`

Here's how `DenseVectorField` should be configured in the schema:

```
<fieldType name="knn vector"
           class="solr.DenseVectorField"
           vectorDimension="4"
           similarityFunction="cosine"/>
<field name="myVector" type="knn_vector" indexed="true"stored="true"/>
```

Apache Solr Dense Vectors

Parameter Name	Required	Default	Description	Accepted values
`vectorDimension`	True		The dimension of the dense vector to pass in.	Integer < = 1024
`similarityFunction`	False	`euclidean`	Vector similarity function; used in search to return top K most similar vectors to a target vector.	`euclidean`, `dot_product` or `cosine`.

knn Query Parser

Parameter Name	**Required**	**Default**	**Description**
`f`	True		The DenseVectorField to search in.
`topK`	False	10	How many k-nearest results to return.

```
&q={!knn f=myVector topK=10}[1.0, 2.0, 3.0, 4.0]
```

Demo

Thank you!