Enhancing Search Precision With Embeddings-Based Semantics

Adarsh Hiremath
NEUROBIO 240: Biological and Artificial Intelligence
ahiremath@college.harvard.edu

May 5, 2023

1. Introduction

Embeddings are numerical representations of words, phrases, or documents that capture their semantic meaning in a vector space. They are commonly used in natural language processing applications, including search, recommendation systems, and sentiment analysis. In search, embeddings can efficiently and accurately retrieve relevant documents or web pages based on the similarity of their semantic content to the user's query.

In December 2022, OpenAI released its newest embedding model, `text-embedding-ada-002`. Open AI claims that `text-embedding-ada-002` is far more powerful similar models like BERT. However, little work has been done to quantify the differences between `text-embedding-ada-002` and more traditional approaches to search. In this paper, I intend to better quantify these differences by doing the following:

- Implementing search with `text-embedding-ada-002`, BERT, and fuzzy matching.
- Creating a custom dataset of my.Harvard classes and their corresponding data.
- Applying those searches to the my.Harvard dataset.
- Quantifying the differences between each search method.
- Building a web application to search Harvard classes and releasing it to the Harvard community.

2. Literature Review

Literature in the field of text embeddings for information retrieval is largely isolated to individual methods, with little work done to compare the effectiveness of different approaches. My paper aims to address this gap by evaluating the performance of three different search methods: `text-embedding-ada-002`, BERT, and fuzzy matching. From my literature review, I drew on the following papers the most:

- *Text and Code Embeddings by Contrastive Pre-Training* (Neelakantan et al. 2022): an evaluation of unsupervised text embedding models on large-scale semantic search. These researchers measured an improvement of 23.4, 14.7, and and 10.6 percent over MSMARCO, Natural Questions and TriviaQA benchmarks.
- *Neural Text Embeddings for Information Retrieval* (Mitra and Craswell 2017): an overview of text embeddings pre-`davinci` for natural language tasks and information retrieval.
- *Semantic Search With Sentence-BERT for Design Information Retrieval* (Walsh and Andrade 2022): an overview of sentence-BERT, a modification to BERT that uses siamese networks for sentence embeddings.

The Neelakantan paper forms the basis for my `text-embedding-ada002` implementation of semantic search. This paper asserts that it is possible to use a single embedding model to do large-scale semantic search. In my experiments, I use `text-embedding-ada002` as the single embedding model for semantic search and evaluate it on the my.Harvard classes dataset.

The Mitra and Craswell paper provided important context about embeddings in general and was helpful for my general understanding of embeddings.

Finally, the Walsh and Andrade paper was greatly helpful for my initial work on the implementation of the BERT-based search. The insighit of passing in the text through a siamese network to determine similarity was particularly helpful.

3. Dataset

Several datasets exist for text classification and aggregation. Initially, I considered testing my semantic search functionality on the following:

- *MS MARCO*: a text dataset with over $1,000,000$ queries and their corresponding saerch results from the Bing search engine. It is commonly used for passage retrieval.

- *The Reuters Corpus*: a text dataset with over 10,000 articles from Reuters, labeled with categoreis such as "earnings" and "acquisitions."
- *CORD-19*: a dataset with over 200,000 scholarly articles related to COVID-19 and other coronaviruses. It is commonly used for information retrieval.

However, none of these datasets had queries prelabeled with "ground truth" about how relevant search responses were. Instead of manually labeling the relevance of search queries for datasets I was unfamiliar with, I opted to use a custom dataset of my.Harvard classes. This dataset contains over 8000 my.Harvard classes organized in the following JSON format:

```
{
    "Id": "GOEnBh8",
    "Subject": "AFRAMER",
    "Number": "11",
    "ClassNumber": "13878",
    "CourseId": "123591",
    "Name": "Introduction to African
    Studies",
    "Desc": "<p>This course introduces
    students to the rich diversity and
    complexity of Africa, including
    its historical dynamics,
    economic developments, ...",
    "Profs": "Daniel Agbiboa",
    "F22": true,
    "Days": [
    "Th"
    ],
    "M": "N",
    "T": "N",
    "W": "N",
    "R": "Y",
    "F": "N",
    "StartTime": "9:45am",
    "EndTime": "11:45am",
    "HasSection": true,
    "Prereq": "Required of
    concentrators in African
    Studies track.",
    "SOC": true,
    "Exam": "12/09/2022 2:00 PM",
    "Location": "Cambridge",
    "URL": "https://locator.
    tlt.harvard.edu/course/
    colgsas-123591/2022/fall/13878",
    "AcadOrg": [
    "AAAS",
    "HIST",
    "EMR"
    ]
}
```

Given that my.Harvard's native search only works with ultra-precise search queries for course titles and course names, applying semantic search to my.Harvard classes appeared to be a far more useful and relevant use case for the Harvard community.

my.Harvard has no consistency for the JSON representing each class, making the scraping operation extremely difficult. For help with generating the dataset, I reached out to the creator of Deez Classes.

4. Search Implementations

Each search technique varies in implementation. `text-embedding-ada-002`, a transformer-based deep learning architecture is utilized along with cosine similarity to measure query and text similarity. The BERT search is implemented using a custom tokenizer, while fuzzy matching uses string-matching algorithms to return results according to a given threshold.

4.1 text-embedding-ada-002

OpenAI's embedding model, `text-embedding-ada-002`, utilizes a transformer-based deep learning architecture to learn embeddings. `text-embedding-ada-002` outperforms the other top models in three standard benchmarks and is useful for working with natural language and code. Semantically similar text is also numerically similar, and the new embeddings endpoint in the OpenAI API provides easy access to the model used to generate embeddings. Although OpenAI offers three families of embedding models - text embeddings text search / similarity, and code search - I only used the base text embedding model for my project.

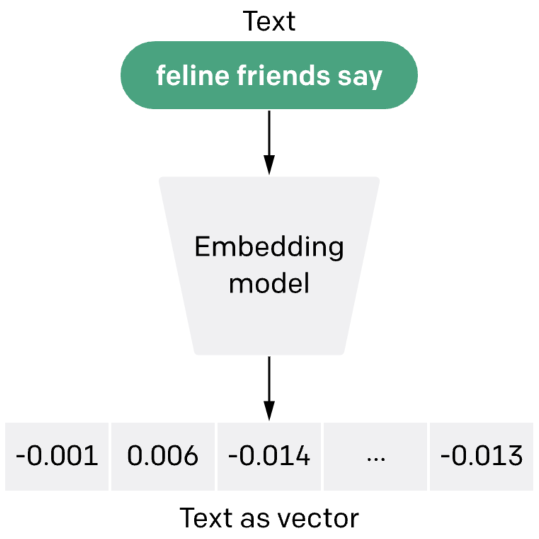

Figure 1: *An example of text embeddings in a vector space.*

I wanted my search to correctly identify classes based on their name, description, professors, start time, and end time. So, in a Pandas dataframe, I combined all of this information into a column called "combined." From there, generating the embedding was straightforward and required making a call to OpenAI's `text-embedding-ada-002` endpoint as follows:

```
course["combined"] = "Title: " +
course["Name"] + "; Description: " +
course["Desc"] + "; Professors: " + \
course["Profs"] + "; StartTime: " + \
course["StartTime"] + "; EndTime: " +
course["EndTime"]

course["embedding"] =
openai.Embedding.create(
input=course["combined"],
engine="text-embedding-ada-002")
["data"][0]["embedding"]
```

Initially, I considered combining the individual embeddings for the course name, description, professors, start time, and end time to increase the accuracy of my search with a custom formula. However, I decided against this for a couple reasons:

- Dimensionality: it was challenging to combine embeddings because their high dimensionality added an additional alignment step before making meaningful comparisons.
- Computational expenses: combining embeddings required calling the Open AI API and doing cosine similarity more times, increasing the cost and time of the algorithm.
- Context: several papers indicated that embeddings can be sensitive to context, leading to variability in their quality and usefulness for different search tasks. Thus, combining embeddings from different sources or models can result in inconsistent or noisy results.

4.2 BERT

Just like for the `text-embedding-ada-002` approach, implementing the BERT search required a similar set of steps: make a combined column, encode the text, and return results by cosine similarity. However, unlike `text-embedding-ada-002`, BERT required some additional modifications to work properly.

BERT expects input in a specific format, where each token is mapped to a unique integer ID, and special tokens are added to denote the start and end of the input sequence. Therefore, before encoding the text using BERT, I need to tokenize the text and convert the tokens to their corresponding integer IDs.

My bert_encode function uses a custom tokenizer that is specifically designed for the course description data used in this project. The tokenizer is created by combining several standard tokenization techniques, such as lowercasing, removing stop words, and splitting on punctuation marks. Additionally, I include some domain-specific tokenization techniques to handle course codes and titles, such as splitting on parentheses, forward slashes, and colons. This helped to improve the accuracy of the BERT embeddings and, therefore, the quality of the search results.

4.3 Fuzzy Matching

To compare my transformer-based search approaches to a more traditional search algorithm, I also implemented a fuzzy matching search algorithm using Levenshtein distance. Levenshtein distance is a string metric used to measure the difference between two sequences and is calculated as the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into the other.

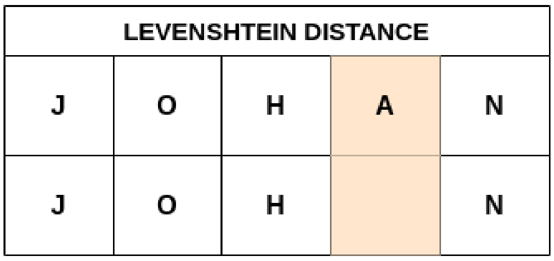

Figure 2: *Fuzzy matching with Levenshtein Distance.*

Fuzzy matching with Levenshtein distance involves comparing a search query to a set of strings and returning the closest matches based on their Levenshtein distance. The algorithm works by calculating the Levenshtein distance between the search query and each string in the set. The strings with the smallest Levenshtein distance are considered the closest matches and are returned as search results.

In my search algorithm, I used the following Python implementation of fuzzy matching with structural modifications to calculate the Levenshtein distances between the search query and the class data:

```
import numpy as np
from Levenshtein import distance

def fuzzy_match(query, strings, threshold):
  distances = np.array([distance(query, s)
   for s in strings])
  matches = [s for i, s in
  enumerate(strings) if
  distances[i] <= threshold]
  return matches
```

In the following example, the fuzzy_match function returns a list of strings from `strings` that are closest to query based on their Levenshtein distance:

```
query = "Harvard"
strings = ["Harvrd", "Havard",
"Harvad", "Havrd", "Harvardd", "Hrvard"]
threshold = 2

matches = [s for s in strings
```

```
    if distance(query, s) < threshold]
print(matches)
```

The algorithm will only return "Harvrd," "Havard," and "Harvad" because they are all distance 1 away from the original query. Conversely, "Havrd" and "Hrvard" will not be returned because they are distance 2 away from the original query.

While Levenshtein distance is a great matching technique for smaller queries and search text, the distance metric fails for larger search text (like Harvard class data), which will be shown later on in this paper.

5. Result Sampling

Before quantifying the differences in search techniques numerically, I wanted to document notable search-response results for each search technique to confirm the notion that an embeddings-based search would be better than naive approaches.

5.1 text-embedding-ada-002

Unlike my.Harvard's native search, `text-embedding-ada-002` is exceptional at dealing with search queries that don't directly correspond with course names or contain spelling errors. To illustrate the power of embeddings-based semantics, we examine a couple examples:

```
Query:
I want to take classes about African Studies
with an emphasis on Christianity or culture.

Output:
```

- `Christianity, Identity and Civil Society in African`
- `Introduction to African Studies`
- `Introduction to African American Studies`

Figure 3: *An example query to the `text-embedding -ada-002` search engine. Although class descriptions are used for the search responses, they are omitted for length. Additionally, the results are ranked in descending order based on relevance. Only 3 responses are returned per query.*

The OpenAI-enabled search engine returned three relevant search results, sorted from most to least relevant. The class descriptions for Christianity, Identity and Civil Society in African matched the search query most heavily, while Introduction to African Studies had a class description that aligned more similarly with the search query than Introduction to African American Studies.

To verify whether the OpenAI-enabled approach truly has semantic understandings of search queries, we can even replace most words in the search query with a synonym and obtain similar performance:

```
Query:
I would be interested in taking courses
about African Learnings with an priority on
Catholicism or culture.

Output:
```

- `Introduction to African Languages and Cultures`
- `Christianity, Identity and Civil Society in African`
- `African Literature and Culture Since 1800`

Figure 4: *An example query to the `text-embedding -ada-002` search engine. Although class descriptions are used for the search responses, they are omitted for length. Additionally, the results are ranked in descending order based on relevance. Only 3 responses are returned per query.*

While slightly different, the search engine returns highly relevant results when given a search query modified with synonyms. However, unlike the embeddings-based approach, my.Harvard's native functionality fails with similar queries. This is easily verifiable by inputting the same search query into my.Harvard:

```
Query:
I want to take classes about African Studies
with an emphasis on Christianity or culture.

Output:
No results found.  Please refine your search
and try again.
```

Figure 5: *An example query to my.Harvard's native search engine.*

The `text-embedding-ada-002` approach is highly effective in understanding search semantics, even when there are spelling errors in the search query. In fact, modifying the previous search query with spelling errors results in a response that is nearly identical.

```
Query:
I wnt take clss abt African Stds with emphs
on Christianity or culture.

Output:
```

```
• Christianity, Identity and Civil Society
  in African
• Introduction to African Popular Culture
• Introduction to African American Studies
```

Figure 6: *An example query to the* `text-embedding -ada-002` *search engine. Although class descriptions are used for the search responses, they are omitted for length. Additionally, the results are ranked in descending order based on relevance. Only 3 responses are returned per query.*

The embeddings model overweights for words spelled correctly, which is why the previously second ranked response is removed in favor of "Introduction to African Popular Culture" when "culture" is one of the only words in the search query that is spelled correctly.

For my.Harvard's search, replacing even one character from the course title results in an invalid search query. Here, we search directly for the class name with but spell "Christianity" as "Christinity":

```
                  Query:
Christinity, Identity and Civil Society in
                 African.

                 Output:
No results found.  Please refine your search
              and try again.
```

Figure 7: *An example query to my.Harvard's native search engine.*

Ultimately, without even conducting any numerical tests, it is already clear that an embeddings-based approach is far more powerful than search protocols implemented in many contexts, including my.Harvard.

5.2 BERT

Like the `text-embedding-ada-002` approach, we also expect BERT to perform far better than my.Harvard's native search. However, qualitative results indicate that there is a marked difference between BERT and `text-embedding-ada-002` in terms of performance. First, repeating the previous specific search query about classes about African studies shows noisier results for the top 3 responses:

```
                  Query:
I want to take classes about African Studies
with an emphasis on Christianity or culture.

                 Output:

• Methods of Behavioral Research
• Methods of Behavioral Research
• Ancient Greek Political Thought
```

Figure 8: *An example query to the BERT search engine. Although class descriptions are used for the search responses, they are omitted for length. Additionally, the results are ranked in descending order based on relevance. Only 3 responses are returned per query.*

While the OpenAI-enabled search engine returned three relevant search results, sorted from most to least relevant, the BERT search approach did not return any relevant responses for the exact same search query about African Studies classes with an emphasis on Christianity or culture. This indicates that the BERT approach is less effective in understanding search semantics compared to the embeddings-based approach.

While unreliable, BERT search does handle some search queries exceptionally well. For instance, take the following example for a query for discrete mathematics classes:

```
                  Query:
Discrete mathematics for computer science
                 majors.

                 Output:

• Studies in Real and Complex analysis
• Discrete Mathematics for Computer
  Science
• Studies in Algebra and Group Theory
```

Figure 9: *An example query to the BERT search engine. Although class descriptions are used for the search responses, they are omitted for length. Additionally, the results are ranked in descending order based on relevance. Only 3 responses are returned per query.*

This example clearly illustrates the power of semantics. Although "discrete mathematics" is nowhere to be found within the course titles or descriptions for two of the responses, the embeddings model is able to associate "discrete mathematics" with semantically similar topics such as "abstract algebra."

Although BERT is a powerful model, it may not be the best choice for semantic search due to its inconsistency. OpenAI's embeddings approach achieves similar performance on the above search queries (including the discrete mathematics example) without the inconsistency of BERT. There could be several reasons why BERT may be less effective at semantic search than Open AI's embeddings model.

- BERT is a far more contextual model that

relies on the surrounding words to understand the meaning of a word, whereas `text-embedding-ada-002` does not require as much context to generate a fixed vector representation of words or phrases.
- BERT is trained on less text than text-embedding-ada-002, which can lead to BERT being less effective at identifying the most relevant information for a given search query.

5.3 Fuzzy Matching

While the Levenshtein fuzzy matching approach performs well on small bodies of text such as just the course name, it fails to produce relevant results when searching through the entire body of text (course name, course description, and professor name) or when there are spelling errors. This is in contrast to the OpenAI embeddings-based approach and BERT, which are highly effective in understanding search semantics, even when there are spelling errors in the search query or the body of text to be searched is extremely large.

```
Query:
I want to take classes about African Studies
with an emphasis on Christianity or culture.

Output:
• Directed Study in History of Art and
  Architecture
• Directed Study in History of Art and
  Architecture
• Independent Study in Genetics
```

Figure 10: *An example query to the Levenshtein fuzzy matching search engine when searching for through the entire body of text associated with each course (course name, course description, and professor name). Although class descriptions are used for the search responses, they are omitted for length. Additionally, the results are ranked in descending order based on relevance. Only 3 responses are returned per query.*

Clearly, these results are incoherent. Two of the search results are duplicates and none of them even remotely correspond to the initial search query. This is to be expected, however, since finding matching text by calculating single character differences is a lot more difficult for large bodies of text than small bodies of text.

Additionally, the fuzzy matching approach also fails when searching for the title of a specific course verbatim when the target text for each course includes the entire body of text associated with each course:

```
Query:
Introduction to African Popular Culture.

Output:
• Directed Study in History of Art and
  Architecture
• Directed Study in History of Art and
  Architecture
• Independent Study in Genetics
```

Figure 11: *An example query to the Levenshtein fuzzy matching search engine when searching for through the entire body of text associated with each course (course name, course description, and professor name). Although class descriptions are used for the search responses, they are omitted for length. Additionally, the results are ranked in descending order based on relevance. Only 3 responses are returned per query.*

However, searching for the exact course title when the target text only includes course titles yields better results:

```
Query:
Introduction to African Popular Culture.

Output:
• Introduction to African Popular Culture
• Introduction to African Languages and
  Cultures
• Introduction to Pre-Columbian America
```

Figure 12: *An example query to the Levenshtein fuzzy matching search engine when searching for through just the course titles. Additionally, the results are ranked in descending order based on relevance. Only 3 responses are returned per query.*

These results indicate that the Levenshtein fuzzy matching algorithm is far better when the text to be searched contains many individual strings of shorter length. In the above example, we see that the first result is indeed the course that we searched for. Additionally, the second best result is also the most similar match in terms of single character operations. However, none of these search results are accurate by semantic similarity because they don't search over the course description or look at words and phrases contextually.

6. Performance Metrics

While the above results provide a qualitative baseline for evaluating each of the search techniques, I also wanted to explore 3 more numerical statistics to evaluate each search implementation: speed, NDCG, and code reliability.

6.1 Speed

When it comes to search using text-embedding-ada-002 or BERT, the most expensive part is generating the vector embeddings to represent the search query and target text. On the other hand, the least expensive part is performing cosine similarity.

To ensure accurate semantic search, I only created embeddings for the classes that had data for both the course title and description while iterating through the data for the 8192 classes. If a course lacked a description, I skipped over it entirely. To evaluate the speed of generating the embeddings, I timed the time it took to generate the embeddings for all the classes for both `text-embedding-ada-002` and BERT.

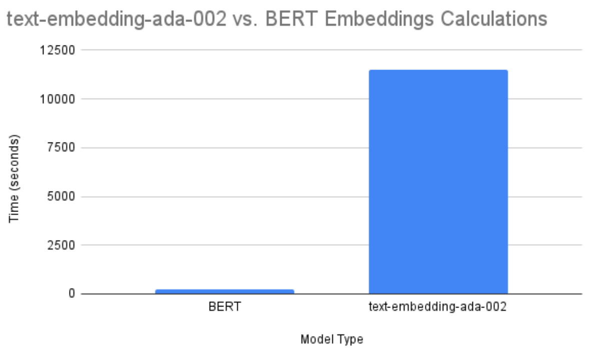

Figure 13: *Time to compute embeddings for the Harvard classes dataset.*

Clearly, BERT has a much smaller time cost to compute the embeddings for the 3000+ classes in the filtered down version of the Harvard classes dataset. It takes approximately 256 seconds to compute the embeddings for the BERT approach whereas it takes over 11503 seconds to compute the embeddings with the OpenAI approach This could be for a couple of reasons:

- Network bottlenecks: currently, generating the OpenAI embeddings is only possible with the API endpoint published by OpenAI. This adds a further network bottleneck to sending and receiving the embeddings data, making it unreliable.
- Custom tokenizers: as detailed in the above sections, the BERT tokenizer and encoder was optimized for the classes task specifically, making it far faster at encoding text.

Next, to evaluate the speed of each of the searches after completing the encodings, I ran an experiment for each of the search techniques with 1000 randomized search trials for each of the search techniques testing for a variety of different query lengths. After averaging all the results, I obtained the following data for each of the searches:

- `text-embedding-ada-002`: 0.4411180019378662 seconds.
- BERT: 0.37630796432495117 seconds.
- Fuzzy search: 0.17109990119934082 seconds.

The differences between search time for each of the approaches is shown in in the following graph:

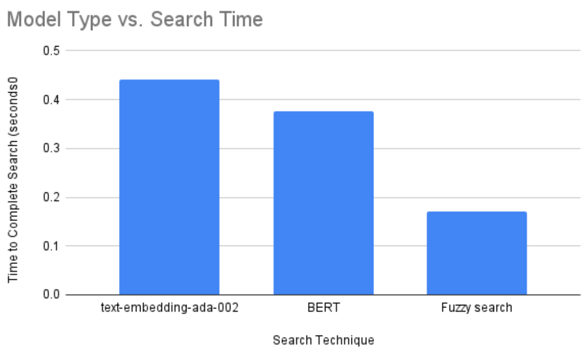

Figure 14: *Average time for search for each search technique (1000 trials).*

Thus, in both the embedding computation and search steps, `text-embedding-ada-002` was the worst search technique in terms of time. As expected, the fuzzy search was the fastest search technique because it takes linear time to compute text differences.

6.2 Normalized Discounted Cumulative Gain (NDCG)

Measuring the quality of a search result is incredibly difficult because the relative quality of search results is inherently arbitrary. However, after speaking to researchers on the Walmart search team, I chose to analyze search quality using the Normalized Discounted Cumulative Gain (NDCG) technique.

The NDCG approach is a method for measuring search quality. It relies on human judges to assess the relevance of search results and then computes a metric to compare different search algorithms. The process starts with crowdsourcing, where independent people evaluate search results on a four-point scale: irrelevant (0), partially relevant (1), relevant (2), and perfect (3).

To calculate NDCG, first, the Cumulative Gain (CG) is determined by summing up the relevance scores for each result. The Discounted Cumulative Gain (DCG) then adjusts the CG by dividing the score by the rank (or log of the rank) to account for the importance of higher-ranked results. The Ideal Discounted Cumulative Gain (iDCG) represents the best possible DCG score given the results. Finally, the NDCG is calculated by dividing the DCG by the iDCG, allowing for comparison of scores across different queries.

By averaging NDCG values across thousands of queries, the performance of different algorithms can be compared using statistical tests. The NDCG approach, along with other metrics, helps determine which search algorithm is superior.

To compare each of the search techniques, I set up my own NDCG experiment by administering a survey to 10 computer science about which search responses were most relevant to the following search query:

```
Data structures and algorithms classes for
        computer science students.
```

For each search approach, I administered a