HEIDI

John Abrahams

May 2024

“The truth... is much too complicated to allow anything but approximations”
- John Von Neumman

1 Introduction

Heidi is a misnomer for "Highlight Documents." The purpose of HEIDI is to highlight a specific section of a document (more specifically, a fixed number of sentences) that is most similar to a given query. The algorithm was first created in python for the Aequitas Capstone at Lehigh University. This paper aims to prove the algorithm's correctness. We will begin by introducing some theory:

2 HEIDI Algorithm

Assume our document D is comprised of n sentences, and correspondingly there exists a function f that is able to convert our document to a matrix of size nxE, where E is the dimensions of the dense embedding space we are converting to. Mathematically:

$$f(D) : Document \to \mathbf{R}^{nxE}$$

Moreover, we assume $f(D)$ is able to accomplish this perfectly with no errors (i.e., assume the semantics and sentences are captured perfectly. this is seldom true, but it is necessary for our proof of the algorithm below). We also assume that our mapping is 1-to-1. That is, each sentence maps exactly to a vector of size E. We will now introduce the algorithm here:

Algorithm 1: HEIDI

```
Input  : $D$: Document
         $\vec{q}$: Embedded query vector
Output: Most relevant document section
1  $D^* \leftarrow f(D)$
2  $\tilde{D} \leftarrow D^*[k-j:j]\ \forall j \in \{k, k+1, \dots, n-1, n\}$
3  $M \leftarrow -\infty$ // maximum value
4  $I \leftarrow$ NULL // index of the maximum value
5  for $D_i \in \tilde{D}$ do
6      $b \leftarrow D_i\vec{q}$
7      if $\sum_i b_i > M$ then
8          $M \leftarrow \sum_i b_i$
9          $I \leftarrow i$
10     end
11 end
12 return $D[I-k:I]$
```

Complexity: $O((n-k) \times E \times n)$

Unfortunately, I was unable to find a clever norm that is able to take into account the direction of our resultant vector. If we had $k = E$, it would have been suffice to introduce *Energy*, as defined in Gilbert Strang's *Learning from Data*, and maximize over this quantity:

$$b^T D_i b$$

This would produce the desired scalar, but it would not give us the freedom of choosing sentence lengths, which could be particularly restricting.

3 Proof

We will begin our proof with the following lemma:

Lemma 3.1. *Loop invariant: The sum of elements in b is maximized over all document sections $\tilde{D}$*

Proof. Base case: for the current trivial set of document sections $\emptyset$, the maximum value is $-\infty$ and maximum index NULL. Inductive step: if b_i is the maximum of $\{b_1, b_2, \ldots b_i - 1\}$, it will be chosen. Termination: When we are done, $i = n$, and we maximize over $\{b_1, b_2, \ldots b_n\}$

Thus, it is proven.

□

Theorem 3.2. *HEIDI finds the most relevant k sentences in a document.*

Assumptions: $f(D)$ captures the sentences in the document perfectly and without errors. Moreover, $f(D)$ captures the semantic meanings of the word perfectly. We will prove by contradiction:

Proof. Assume b^c is selected as the best choice, but b is more relevant. If b is more relevant, then

$$\sum_i b_i^c < \sum_i b_i$$

, which breaks the loop invariant proved in lemma 3.1. Hence, our assumption is incorrect, and the indices corresponding to b will be returned as the best choice. □

4 Implementation

In practice our assumptions do not hold: We use the nltk library for converting documents to lists of sentences, and the SentenceTransformer library for converting the sentences to their corresponding dense embedding space, where $E = 384$. These libraries are subject to error which will tend to throw off our results, but they provide the best results for our algorithm's implementation.

A class Highlighter was instantiated. Note that 'model.encode()' in SentenceTransformer takes quite a long time (in our cohort of documents, roughly 3 seconds per document). Therefore I found it was best to convert the cohort of documents into a sparse vector representation and save it using 'numpy.savetxt().' In practice we want to calculate HEIDI over multiple documents. Therefore we store our documents as a $MxNxE$ three-dimensional array, where M is the number of documents, N is the maximum number of sentences, and E is the embedding space. Note that E is uniform for all sentences, and N is chosen by selecting the maximum number of sentences in all documents, and filling up the documents with less than N sentences with $\vec{0}$ vectors. Then, we will call a helper function within the implementation to load in the three-dimensional array and loop through the documents, calling HEIDI and storing each result in a list.