Final Report

COMPUTATIONAL GEOMETRY

CS6319.001

Authors:

Nicholas Zolton (ngz200000)

April 24, 2024

Contents

Introduction

1 Reason for Study

Throughout this semester I've been working on an indepedent study project that has used AI embeddings to create a tailoring system. This caused me to quickly realize the main problem with the current state of AI vector databases: they, despite focusing on nearest neighbor queries, primarily run in linear time. This quickly becomes an issue at scale, as the number of vectors necessary for many projects can grow quite quickly.

Luckily, however, during class while discussing the Voronoi diagram I started to consider the potential of using it to solve the nearest neighbor problem for embeddings. This led me to the idea of using the Voronoi diagram to create a database that could be used to solve the nearest neighbor problem in logarithmic time. This project is the result of that idea.

2 Initial Ideas

The initial idea for a solution to this problem that I had would be to solve the following steps:

1. Embed text into 2D space using a pre-trained model.
2. Visualize the text embeddings to make sure they are in a usable format in low dimensions.
3. Create a Voronoi diagram of the text embeddings.
4. Create a function that would allow for the querying of the nearest neighbor to a given point.
5. Generalize the solution to work in higher dimensions.

So, with this in mind, I set out to follow these steps and create the Voronoi diagram database.

3 Embedding Text into 2D Space

For starters, I needed to embed the text into 2D space. Normally, I would love to create a custom trained model from scratch, but due to the lack of resources at my disposal (on a fairly low-end laptop) I decided to instead use a pre-trained model to handle the embedding process. I do not believe this is a major issue, as the goal of this project is the computational geometry aspect, not the AI aspect. With this in mind, I decided to use OpenAI's "text-embedding-3-small" model. This model is known for being fairly accurate and is also quite small so it can be run quite cheaply (in total I only spent around 5 cents on this project).

There is one small issue with using this model, however. The model is trained to embed text into 1536 dimensions, which is far too high for the purposes of this project, since the space complexity of the Voronoi diagram is $O(n^{d/2})$. This would, in theory, cause my laptop to run out of memory when trying to create even a small Voronoi diagram. So, I set out to reduce the dimensionality of the embeddings to 2D.

3.1 First Attempt at Dimensionality Reduction

Before I began to attempt to reduce the dimensionality of the embeddings after they were created, I decided to try and reduce the dimensionality of the embeddings at their time of creation. While this would not be possible just one year ago with OpenAI's text embedding models, they have since added a feature that allows for the embeddings to be created in a given dimensionality. So, I decided to try and utilize the OpenAI library to create the embeddings in 2D space from the start. This did indeed work in that it returned embeddings in 2D space, but the embeddings were a bit strange. Below is a figure of the embeddings of all the named colors (according to Wikipedia) in 2D space:

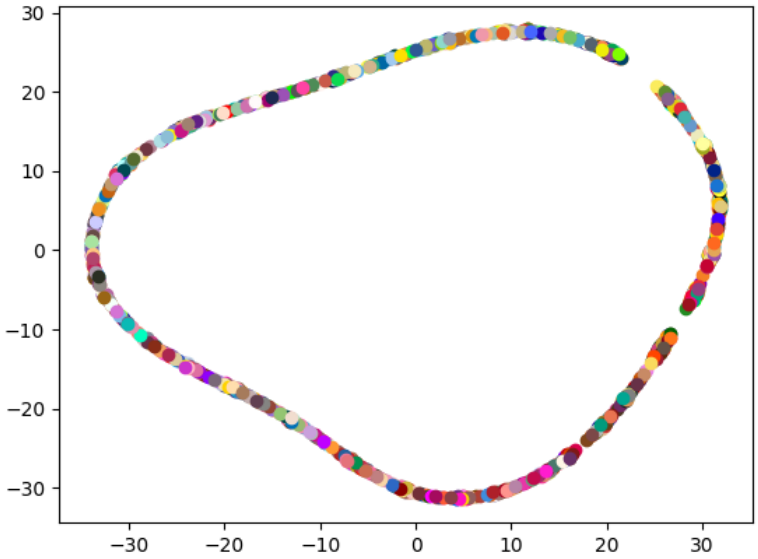

Figure 1: OpenAI's Embeddings of named colors in 2D space. The color of each point corresponds to the color it represents.

Notice how the embeddings are in a very specific, clean polygonal shape. This is not what we would expect from embeddings of colors, even though they are all colors. Instead, we would expect groupings of similar colors. While there is a little bit of grouping, it is not nearly as strong as we would expect. For this reason, it would seem that we cannot rely on OpenAI's text embedding model to create embeddings in 2D space.

3.2 Second Attempt at Dimensionality Reduction

So, although the OpenAI dimensionality method was promising, it did not fill the requirements of the project. Instead, I decided to investigate other methods of dimensionality reduction. From this, I found the T-SNE algorithm. T-SNE is a fairly popular algorithm for dimensionality reduction and is known for being able to reduce the dimensionality of embeddings while still keeping the groupings of similar items. So, I decided to try and use T-SNE to reduce the dimensionality of the embeddings from 1536 to 2. This worked quite well, and the embeddings of the named colors in 2D space looked much more like what we would except:

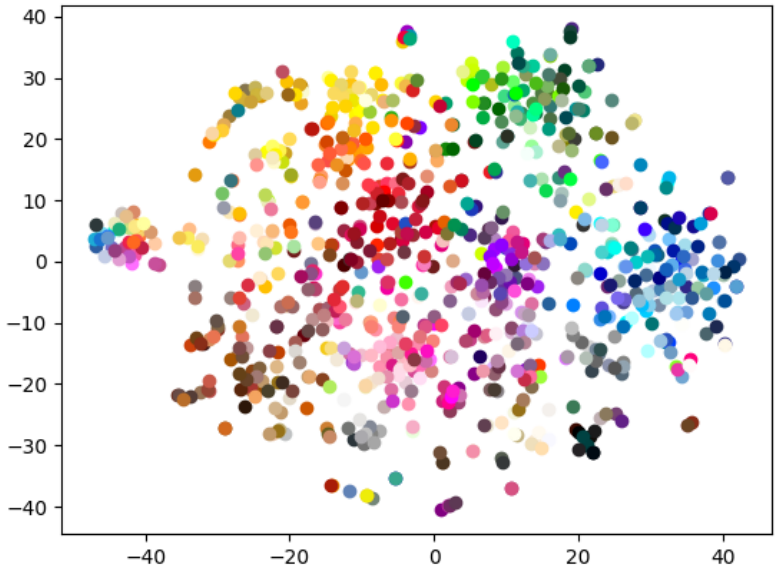

Figure 2: T-SNE's Embeddings of named colors in 2D space.

Here we can absolutely see more clear groupings of similar colors, and with a lot more variation in the x and y dimensions. This is perfect for our use case. So, with this in mind, I decided to use T-SNE to reduce the dimensionality of the embeddings to 2D space. There are, however, a few issues with this approach that we will need to address later on. The first is that T-SNE is a fairly slow algorithm, so it reduces the speed increase that we would hope for by using the Voronoi diagram, but only in the creation process of the diagram. Regardless, this would allow us to begin targeting the other steps of the project since we can focus on the overarching plan to generalize nearest neighbor queries in our data structure. Second, T-SNE is a stochastic algorithm, so it will not always produce the same results. This is not a major issue, but it is something to keep in mind when we are testing the system. Finally, T-SNE is not perfect, so we may lose some information in the dimensionality reduction process (this is not only expected, but necessary for low dimensional reduction so we do not need to worry about it). Luckily, all of these issues should not affect our generalized nearest neighbor query function, so we can move on to the next step of the project and simply consider the database to be "static" for now.

4 Creating the Voronoi Diagram

As seen both in class and throughout computational geometry literature, the Voronoi diagram is a powerful tool for solving nearest neighbor queries. Because of this, it's an extremely well-studied data structure and there are many algorithms for creating it. This poses an interesting question: which algorithm should we use to create the Voronoi diagram? Should we even program it ourselves? Part of the problem as well is that the future data structure we will be creating will hopefully be both generalized and incremental, so we need to make sure that the Voronoi diagram creation algorithm we choose will allow for this (although, in practice, we could theoretically create the Voronoi diagram using one algorithm and then modify it with another algorithm). For now, however, I decided to use Quick Hull to create the Voronoi diagram, since it is fairly simple, generalizable to higher dimensions, and is incremental. Running a very simple version of Quick Hull on the named colors embeddings, we get the following Voronoi diagram:

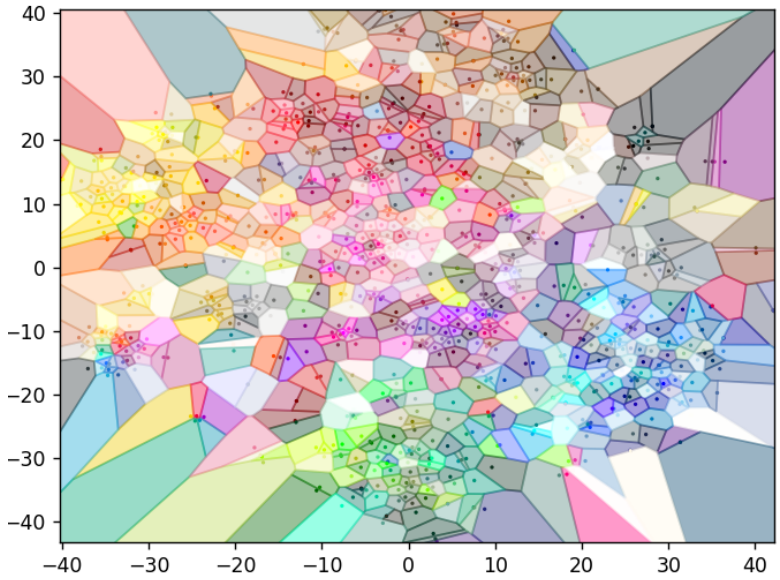

Figure 3: Voronoi Diagram of named colors embeddings.

This, while being fairly simple and mostly just a visualization, shows that we can create the Voronoi diagram quite cleanly. Now, we will need to create a function that will allow us to query the nearest neighbor to a given point. This will be quite easy, as we can simply find the cell that the new point is in and then accept that cell's point as the nearest neighbor. However, it's also common to find the nearest n neighbors to a given point, so we will also need to explore nearby cells to find more neighbors. Regardless, this is a promising start to

the project and we will now move on to the next steps: creating the querying function and generalizing the function.

5 Querying the Voronoi Diagram

Now that we have the Voronoi Diagram, we need to create a function that will allow us to query the nearest neighbor to a given new point. The problem, however, is that up until now we have been using T-SNE to reduce the dimensionality of the embeddings to 2D space, but T-SNE is not able to add new points to the embeddings moving forward because it does not actually use parametric mapping from the original space to the new space. This makes it impossible to query the nearest neighbor to a truly new point. So, we will need to make a small change to how we are reducing dimensionality. Instead of simply using T-SNE to reduce the dimensionality of the embeddings, we will instead need to either create a multivariate regressor to predict the 2D space embeddings from the 1536D embeddings or use a regressor to minimize the loss between the 2D space embeddings and the 1536D embeddings directly, as was done in a paper by Laurens Matten[1].

Luckily for us, however, this was already implemented in openTSNE, a Python library that allows for the training of a regressor to predict the 2D space embeddings from the 1536D embeddings. This is perfect for our use case, as it enables us to create a query point inside of the 2D space without having to worry about the dimensionality reduction process. So, with this in mind, we can now move on to creating the querying function. This function will simply follow a few steps:

1. Take in a new input (string).
2. Embed the input into 1536D space.
3. Use the regressor to predict the 2D space embeddings.
4. Find the cell that the new point is in.
5. Return the point in that cell as the nearest neighbor.

6. If we want more than one nearest neighbor, explore nearby cells and return those as well.

This is a fairly simple process at a high level, but it will require a bit of work to implement. First, we will need to sanity-check the regressor to make sure it is working correctly.

5.1 Sanity-Checking the Regressor

To sanity-check the regressor, we will need to embed a new point into 1536D space, predict the 2D space embeddings, and then plot the new point into our Voronoi diagram to make sure it is in the correct cell. While I know this is not the most solid method of checking the validity of the method, it is a good first step for making sure that the queries make at least a little bit of sense. So, with this in mind, I decided to embed the word "lime". Below is the result of the sanity check test:

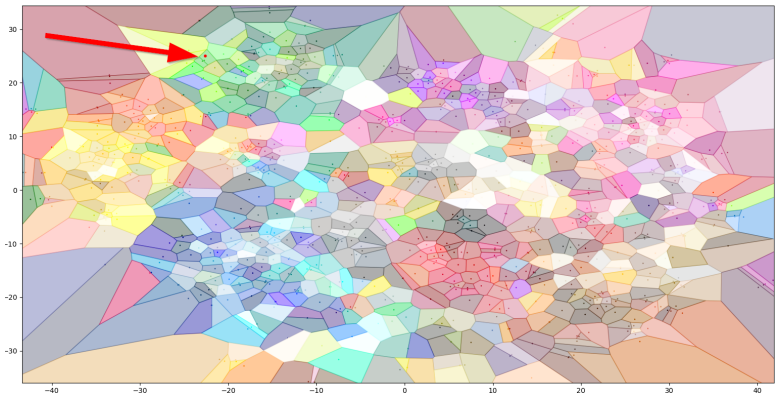

Figure 4: Sanity Check of the regressor. The word "lime" is embedded into 2D space and plotted into the Voronoi diagram.

While it is a bit difficult to quantify how well the overall process is working, it is clear that the regressor is at least somewhat working since we can see that the new point for "lime" is in the correct area, as it is grouped with the majority of the other "green" colors and is paired specifically with a bright green! Personally, I also tried this with a few other colors and received similarly accurate results, but due to brevity I will leave those out for now. So, now that we have proven that placing new points into the Voronoi diagram is viable, we can move on to the next step of doing planar point location to actually find what the nearest neighbor is for our new queried point.

5.2 Planar Point Location

References

[1] van der Maaten, L. (2009). Learning a parametric embedding by preserving local structure. https://lvdmaaten.github.io/publications/papers/AISTATS_2009.pdf.