You might not know it yet, but vector embeddings are everywhere. They are the building blocks of many machine learning and deep learning algorithms used by applications ranging from search to AI assistants. If you're considering building your own application in this space, you will likely run into vector embeddings at some point. In this post, we'll try to get a basic intuition for what vector embeddings are and how they can be used.

What problem are we trying to solve?

When you build a traditional application, your data structures are represented as objects that probably come from a database. These objects have properties (or columns in a database) that are relevant to the application you're building.

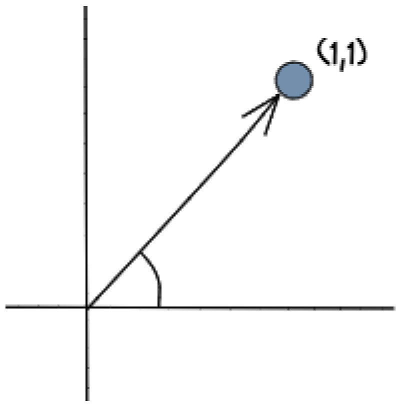
Over time, the number of properties of these objects grows — to the point where you may need to be more intentional about which properties you need to complete a given task. You may even end up creating specialized representations of these objects to solve particular tasks without paying the overhead of having to process very "fat" objects. This process is known as feature engineering — you optimize your application by picking only the essential features relevant to the task at hand.

When you deal with **unstructured** data, you will have to go through this same feature engineering process. However, unstructured data is likely to have many more pertinent features, and performing manual feature engineering is bound to be untenable.

In those cases, we can use **vector embeddings** as a form of automatic feature engineering. Instead of manually picking the required features from our data, we apply a pre-trained machine learning model that will produce a representation of this data that is more compact while preserving what's meaningful about the data.

What are vector embeddings?

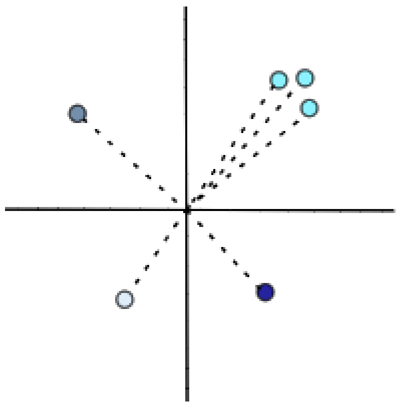
Before we delve into what vector embeddings are, let's talk about vectors. A **vector** is a mathematical structure with a size and a direction. For example, we can think of the vector as a point in space, with the "direction" being an arrow from (0,0,0) to that point in the **vector space.**



As developers, it might be easier to think of a vector as an array containing numerical values. For example:

```
vector = [0,-2,...4]
```

When we look at a bunch of vectors in one space, we can say that some are closer to one another, while others are far apart. Some vectors can seem to cluster together, while others could be sparsely distributed in the space.



We'll soon explore how these relationships between vectors can be useful.

Vectors are an ideal data structure for machine learning algorithms — modern CPUs and GPUs are optimized to perform the mathematical operations needed to process them. But our data is rarely represented as vectors. This is where vector embedding comes into play. It's a technique that allows us to take virtually any data type and represent it as vectors.

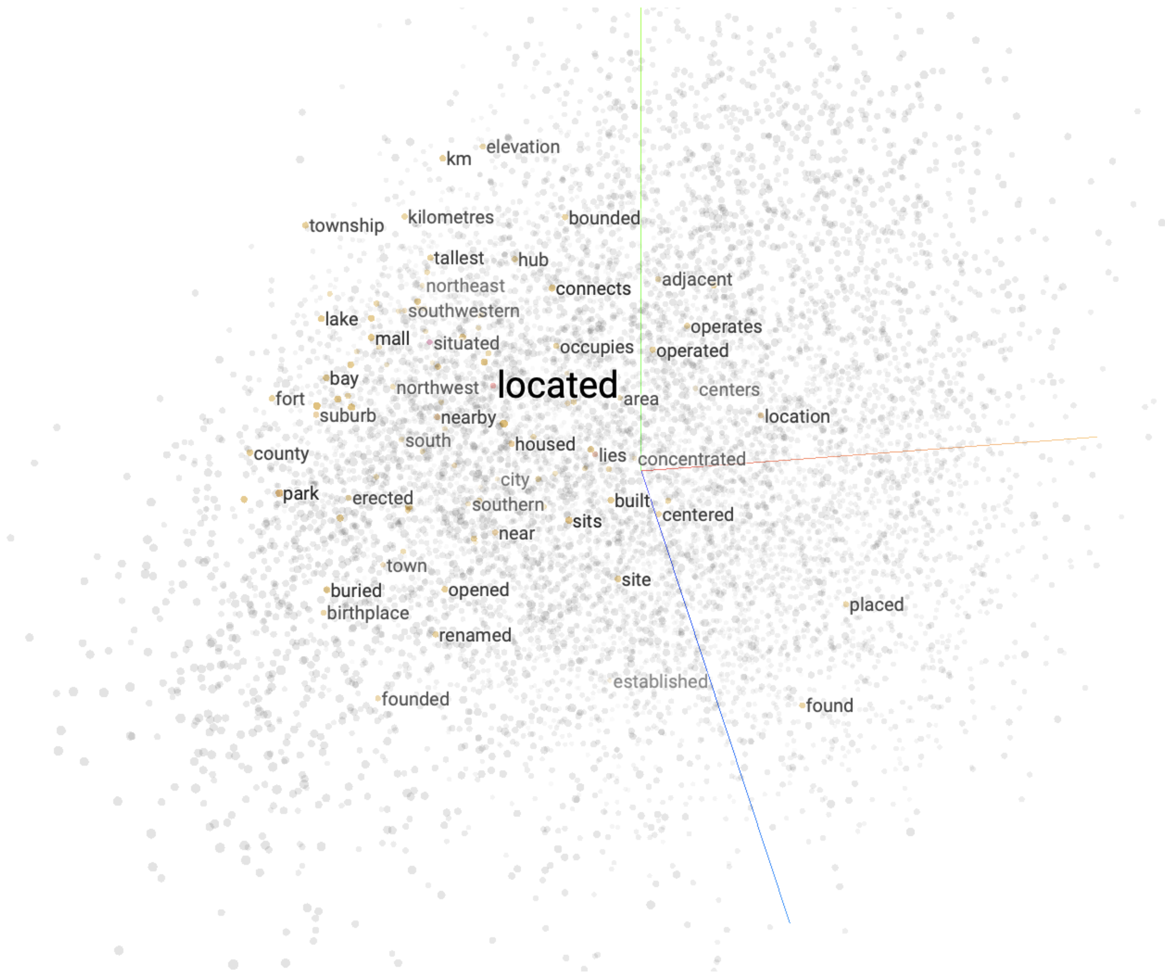
But it isn't as simple as just turning data into vectors. We want to ensure that we can perform tasks on this transformed data without losing the data's original meaning. For example, if we want to compare two sentences — we don't want just to compare the words they contain but rather whether or not they mean the same thing. To preserve the data's **meaning**, we need to understand how to produce vectors where **relationships** between the vectors **make sense.**

To do this, we need what's known as an **embedding model**. Many modern embedding models are built by passing a large amount of **labeled data** to a neural network. You might have heard of neural networks before — they are also a popular tool used to solve all sorts of complex problems. In very simple terms, neural networks are made of layers of nodes connected by functions. We then train these neural networks to perform all sorts of tasks.

We train neural networks by applying supervised learning — feeding the network a large set of training data made of pairs of inputs and labeled outputs. Alternatively, we can apply self-supervised or unsupervised learning either of which doesn't require labeled outputs. These values are transformed with each layer of network activations and operations. With every iteration of training, the neural network modifies the activations in each layer. Eventually, it can predict what an output label should be for a given input — even if it hasn't seen that particular input before.

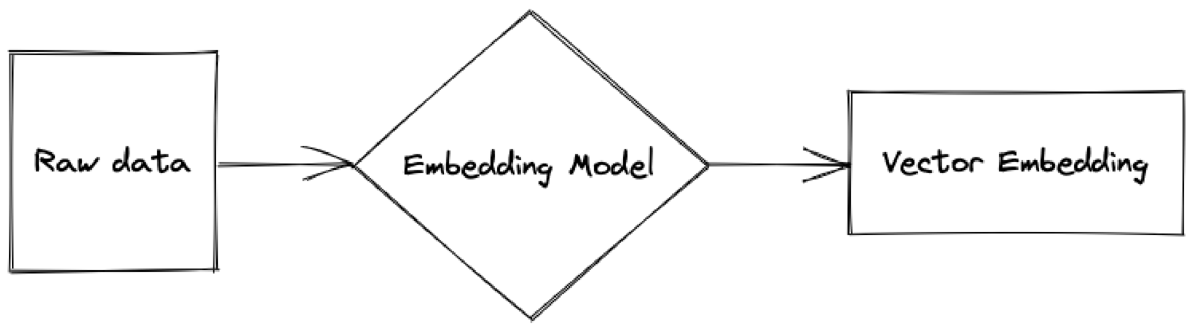
The embedding model is basically this neural network with the last layer removed. Instead of getting a specific labeled value for an input, we get a vector embedding.

A great example of an embedding model is the popular word2vec, which is regularly used for a wide variety of text-based tasks. Let's take a look at a visualization produced by TensorFlow's projector tool, which makes it easy to visualize embeddings.



While this visualization represents only three dimensions of the embeddings, it can help us understand how the embedding model works. There are multiple data points highlighted in the visualization, each representing a vector embedding for a word. As the name suggests, word2vec embeds words. Words that appear close to one another are semantically similar, while far-apart words have different semantic meanings.

Once trained, an embedding model can transform our raw data into vector embeddings. That means it knows where to place new data points in the vector space.



As we saw with word2vec, within the context of the model, vectors that are close together have a contextual similarity, whereas far-apart vectors are different from one another. That's what gives our vector **meaning** — its relationship with other vectors in the vector space depends on how the embedding model "understands" the domain it was trained on.

What can I do with vector embeddings?

Vector embeddings are an incredibly versatile tool and can be applied in many domains. Generally speaking, an application would use a vector embedding as its query and produce other vector embeddings which are similar to it, with their corresponding values. The difference between applications of each domain is the significance of this similarity.

Here are some examples:

- Semantic Search - search engines traditionally work by searching for overlaps of keywords. By leveraging vector embeddings, semantic search can go beyond keyword matching and deliver based on the query's semantic meaning.
- Question-answering applications - by training an embedding model with pairs of questions and corresponding answers, we can create an application that would answer questions that have not been seen before.
- Image search - vector embeddings are perfectly suited to serve as the basis for image retrieval tasks. There are multiple off-the-shelf models, such as CLIP, ResNet, and more. Different models handle different types of tasks like image similarity, object detection, and many more.
- Audio search - by converting the audio into a set of activations (an audio spectrogram), we produce vector embeddings that can be used for audio similarity search.
- Recommender Systems - we can create embeddings out of structured data that correlate to different entities such as products, articles, etc. In most cases, you'd have to create your own embedding model since it would be specific to your particular application. Sometimes this can be

combined with unstructured embedding methods when images or text descriptions are found.
- Anomaly detection - We can create embeddings for anomaly detection using large data sets of labeled sensor information that identify anomalous occurrences.

Vector embeddings are incredibly powerful, and this is by no means an exhaustive list — head to our example apps section to go deeper. You can also read more about the basics of vector search to see how Pinecone can help you wrangle vector embeddings.