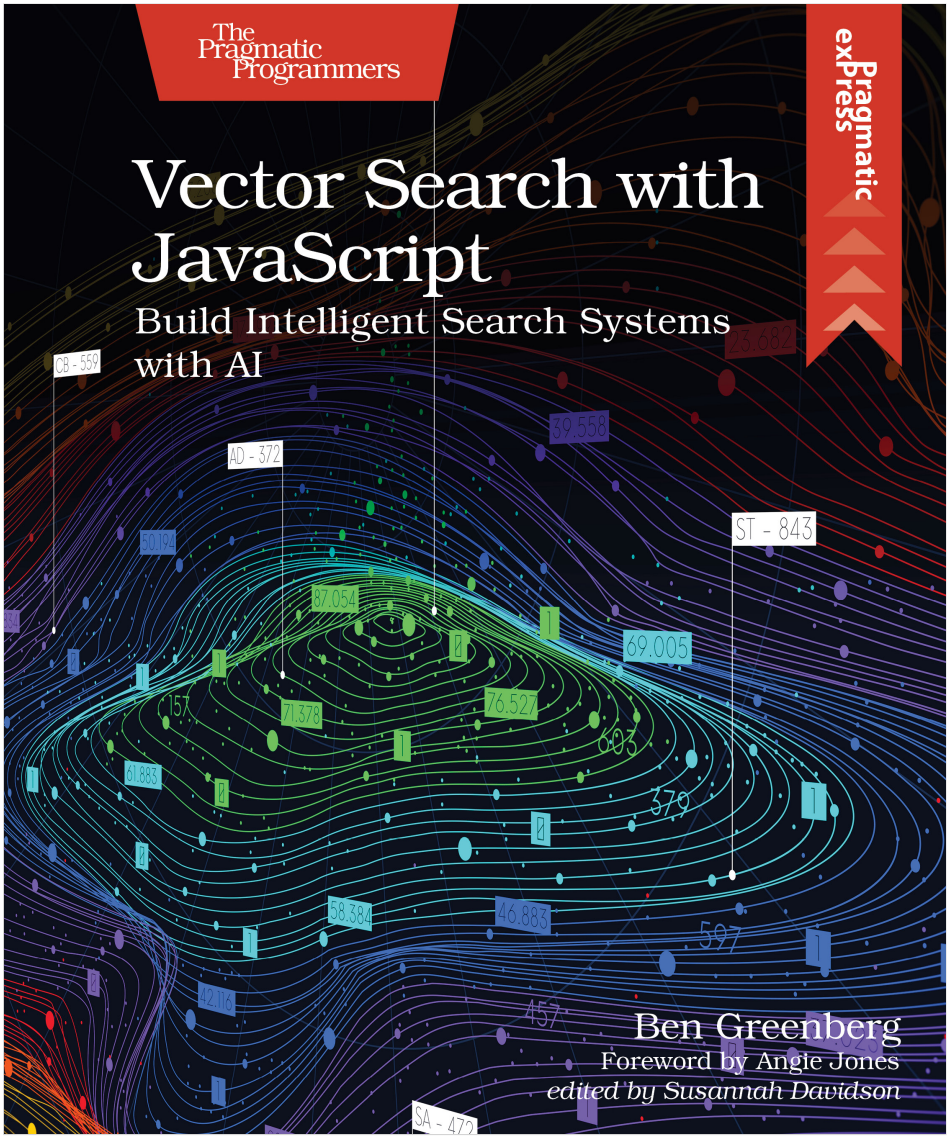

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Unpacking the Concept of Embeddings

Text is more complex than colors. You need a way to generate vectors for titles or paragraphs so that two texts with similar meanings produce points that are close together in vector space. One common technique is to create something called an embedding.

An embedding is just a long list of numbers. Each number represents a feature extracted by a machine learning model. If you use OpenAI's embedding model, each vector has 1,536 dimensions. You can think of these as axes in a massive coordinate space.

The goal is for texts that mean similar things to have similar embeddings. For example, the phrases "Learning JavaScript" and "Mastering JavaScript" should land near each other in vector space, since they share both subject matter and educational intent. A phrase like "Exploring Ruby" might be close as well, because it has a similar structure and purpose. But something like "Deploying Kubernetes at Scale" would be much farther away, even though it also describes a technical skill, because it belongs to a completely different domain.

You create an embedding by feeding your text into a pre-trained model. That model has already learned how language works by training on massive datasets. It compresses everything it knows into a set of internal weights. When you pass in new text, it produces a vector that captures key aspects of the input. This process may feel like a black box for now, but we will demystify it step by step.

Understanding Similarity

Once you have embeddings, the next question is how to compare them. In other words, how can you tell which vectors are close to each other?

To compare embeddings, we need a way to measure how similar two vectors are. The most common methods are cosine similarity, which measures the angle between vectors (regardless of their size), and dot product, which considers both their direction and magnitude. Cosine similarity answers "are these vectors pointing the same way?" while dot product adds the question, "and how strong is that signal?"

To make it more tangible, imagine comparing fruit. Let's say we use made-up 3D embeddings to represent "apple," "banana," and "orange." These numbers might reflect how sweet, fibrous, or acidic each fruit is. If the vectors are close, we assume the fruits are similar.

In fact, let's calculate it.

understanding_vector_search/similarity_applies_bananas_oranges.js

```
const vectors = {
  Apple: [0.9, 0.1, 0.0],
  Banana: [0.7, 0.3, 0.0],
  Orange: [0.8, 0.2, 0.1],
};

/**
 * Calculate the dot product of two vectors.
 * @param {Array<number>} vec1 - First vector.
 * @param {Array<number>} vec2 - Second vector.
 * @returns {number} - Dot product of the two vectors.
 */
const calculateDotProduct = (vec1, vec2) =>
  vec1.reduce((sum, value, index) => sum + value * vec2[index], 0);

/**
 * Calculate the magnitude of a vector.
 * @param {Array<number>} vec - The vector.
 * @returns {number} - Magnitude of the vector.
 */
const calculateMagnitude = (vec) =>
  Math.sqrt(vec.reduce((sum, value) => sum + value ** 2, 0));

/**
 * Calculate the cosine similarity between two vectors.
 * @param {Array<number>} vec1 - First vector.
 * @param {Array<number>} vec2 - Second vector.
 * @returns {number} - Cosine similarity.
 */
const calculateCosineSimilarity = (vec1, vec2) => {
  const dotProduct = calculateDotProduct(vec1, vec2);
  const magnitude1 = calculateMagnitude(vec1);
  const magnitude2 = calculateMagnitude(vec2);
  return dotProduct / (magnitude1 * magnitude2);
};

// Example: Calculate cosine similarity between "Apple" and "Banana"
const apple = vectors.Apple;
const banana = vectors.Banana;

const similarity = calculateCosineSimilarity(apple, banana);

console.log(`Apple/Banana similarity: ${similarity.toFixed(3)}`);
```

This code calculates the cosine similarity between "apple" and "banana." Even though the numbers are arbitrary, the math shows how closely aligned the vectors are. That alignment becomes the basis for comparison in vector search.

Key Takeaways

In this chapter, you learned that a vector is just a list of numbers and that similarity between vectors can be measured with cosine similarity or dot product. You saw how embeddings represent complex data as vectors and how systems use those vectors to find the most relevant matches.

These ideas might feel new now, but you've already used them. The title-matching script in Chapter 1 embedded and compared titles using these exact principles. With a better understanding of what was going on under the hood, you are now ready to go deeper.

In the next chapter, we'll explore how embeddings are actually generated, what makes a good embedding, and why these representations are so powerful for building intelligent search systems.