Understanding the approximate nearest neighbor (ANN) algorithm

By Elastic Platform Team

17 April 2024

If you grew up in a time before the internet made its debut, you'll remember it wasn't always easy to find new things to like. We discovered new bands when we happened to hear them on the radio, we'd see a new TV show by accident because we forgot to change the channel, and we'd find a new favorite video game based almost entirely on the picture on the cover.

Nowadays, things are *very* different. Spotify will point me to artists that match my tastes, Netflix will highlight movies and TV shows it knows we'll enjoy, and Xbox knows what we'll probably want to play next. These recommendation systems make it so much easier for us to find the things we're actually looking for, and they're powered by nearest neighbor (NN) algorithms. NN looks at the extensive sea of information it has available and identifies the closest thing to something you like, or something you're searching for.

But NN algorithms have an inherent flaw. If the amount of data they're analyzing gets too big, crawling through every option takes forever. This is a problem, especially as these data sources get bigger and bigger every year. This is where approximate nearest neighbor (ANN) grabs the baton from NN and changes the game.

In this document, we'll cover the following key topics about ANN:

- ANN definition
- How ANN works
- When to use ANN search
- ANN importance in vector search
- Various types of ANN algorithms

Approximate nearest neighbor explained

Approximate nearest neighbor (ANN) is an algorithm that finds a data point in a data set that's very close to the given query point, but *not* necessarily the absolute closest one. An NN algorithm searches exhaustively through all the data to find the perfect match, whereas an ANN algorithm will settle for a match that's *close enough*.

This might sound like a worse solution, but it's actually the key to nailing fast similarity search. ANN uses intelligent shortcuts and data structures to efficiently navigate the search space. So instead of taking up huge amounts of time and resources, it can

identify data points with much less effort that are close enough to be useful in most practical scenarios.

Essentially, it's a trade-off. If you absolutely need to find the one best match, you can do that at the expense of speed and performance with NN. But if you can tolerate a tiny drop in accuracy, ANN is almost always a better solution.

How approximate nearest neighbor algorithms work

The first part of how ANN works is **dimensionality reduction**, where the goal is to turn a higher-dimensional data set into a lower-dimensional one. The aim is to make the predictive model task less complicated and more efficient than having to analyze all the data.

These algorithms rest on the mathematical concept of metric spaces — where data points reside and distances between them are defined. These distances must adhere to specific rules (non-negativity, identity, symmetry, triangle inequality), and common functions like Euclidean distance or cosine similarity are used to calculate them.

To better understand this, imagine you're on holiday searching for the villa you've rented. Instead of checking every single building one-by-one (higher-dimensional), you'd use a map, which reduces the problem into two dimensions (lower-dimensional). (This is a deliberately simplistic example. Dimensionality reduction is not the sole method employed by ANN algorithms to improve efficiency.)

ANN algorithms also leverage clever data structures called **indexes** to improve efficiency. By pre-processing the data into these indexes, ANN can navigate the search space much quicker. Think of these as street signs, helping you find where you are on the map to reach your holiday villa quicker.

When to use approximate nearest neighbor search

In the fast-paced world of data science, efficiency reigns supreme. While finding the true closest neighbor (exact nearest neighbor search) holds value, it often comes at a computational cost, as we've already talked about. This is where ANN search shines, offering a compelling trade-off: lightning speed with high, but not absolute, accuracy.

But when exactly should you choose ANN over other search methods?

Exact nearest neighbor might be slow, but it's the best option when accuracy is your priority or you're using small data sets. **k-nearest neighbors (kNN)** sits between NN and ANN by giving you faster results while maintaining high accuracy. But it can be hard to get right when deciding the value of k, and it also struggles with high-dimensional data.

ANN's speed and efficiency combined with its high (but not absolute) accuracy makes it perfect in a number of situations:

- **Large data sets:** When dealing with millions or even billions of data points, the exhaustive nature of exact NN becomes sluggish. ANN excels in navigating vast data landscapes, delivering results swiftly.
- **High-dimensional data:** As dimensions climb, exact NN computations explode. ANNs dimensionality reduction techniques effectively shrink the search space and boost efficiency in complex data like images or text.
- **Real-time applications:** Need results instantly? Recommendation systems, fraud detection, and anomaly detection rely on real-time insights. ANN's speed makes it ideal for these scenarios.
- **Acceptable approximation:** If your application can tolerate slight inaccuracies in results, ANN's speed becomes invaluable. For example, in image search, finding visually similar images — instead of the absolute closest one — might be sufficient.

Importance of ANN in vector search

Vector search deals with data encoded as dense vectors, capturing complex relationships and embedded meanings. This makes it ideal for searching content like images, text, and user preferences, where traditional keyword-based search often falls short. But the curse of dimensionality applies here, too. Because as the number of dimensions representing these vectors increases, traditional search methods struggle, becoming slow and inefficient.

ANN solves this problem by switching the focus from finding an exact match to "close enough" matches. This enables fast retrieval, where your vector search can find similar vectors in massive data sets lightning fast. It also gives you baked-in scalability, so you can grow your data set as much as you want without sacrificing speed.

These real-time responses combined with the improved relevance and efficiency often mean that ANN can play a critical role in unlocking the true potential of your vector search.

Types of approximate nearest neighbor algorithms

While the concept of ANN offers a compelling speed advantage in search, this term actually covers a diverse toolbox of algorithms. They all have their own strengths and trade-offs, and understanding these nuances is critical when choosing the right tool for your specific data and search needs.

KD-trees

KD-trees organize data points in a hierarchical tree structure, partitioning the space based on specific dimensions. This enables fast and efficient search in low-dimensional spaces and Euclidean distance-based queries.

But while KD-trees excel at finding nearest neighbors in low dimensions, they suffer from the “curse of dimensionality.” This is where, as the number of dimensions increases, the space between points explodes. In these high dimensions, KD-trees' strategy of splitting based on single axes becomes ineffective. This makes the search examine most of the data, losing the efficiency advantage and approaching the slowness of a simple linear scan through all points.

Locality-sensitive hashing (LSH)

LSH is a powerful ANN technique that works by "hashing" data points into lower-dimensional spaces in a way that cleverly preserves their similarity relationships. This clustering makes them easier to find, and it allows LSH to excel in searching massive, high-dimensional data sets like images or text with both speed and scalability. And it does all this while still returning "close enough" matches with good accuracy. But keep in mind that LSH might also occasionally produce false positives (finding non-similar points as similar), and its effectiveness can vary based on the distance metric and data type. There are various LSH families designed to work with different metrics (e.g., Euclidean distance, Jaccard similarity), which means LSH remains versatile.

Annoy

Annoy (Approximate Nearest Neighbors Oh Yeah) isn't a single algorithm, but an open-source C++ library that uses its own algorithms for building and querying trees, without directly implementing LSH or KD-trees. It's designed for memory-efficient and fast search in high-dimensional spaces, making it suitable for real-time queries. Essentially, it’s a user-friendly interface offering flexibility for different data types and search scenarios. Annoy's strength lies in leveraging multiple ANN approaches under one roof, allowing you to choose the best fit for your needs. While it simplifies the process, remember that picking the right internal algorithm within Annoy is crucial for optimal performance, and its effectiveness still depends on factors like your data and accuracy requirements.

Linear scan algorithm

Although not *typically* classified as an ANN technique, it’s worth mentioning linear scan because it’s a brute-force approach that gives you similar results to other ANN algorithms. It iterates through every data point sequentially, calculating the distances

between records and keeping track of the best matches. Because of the simplistic nature of the algorithm, it's easy to implement and great for small data sets. The downside of the more basic approach is that it's inefficient for large data sets, slow when used with high-dimensional data, and impractical for real-time applications.

Choosing the right ANN

Before you dive into picking an ANN, there are a few things you should consider before deciding:

- **Data set size and dimensionality:** Consider using locality-sensitive hashing for large and high-dimensional data and KD-trees for smaller and lower-dimensional data.
- **Desired accuracy level:** If absolute precision is vital, linear scan is likely the best option — otherwise, look at LSH or Annoy for good accuracy with speed.
- **Computational resources:** Annoy offers flexibility, but consider memory and processing limitations before choosing an algorithm within it.

Remember – there's no one-size-fits-all solution. Experiment with different ANN algorithms and evaluate their performance on your specific data to find the perfect match for your vector search needs. Beyond these options, the world of ANN algorithms is constantly evolving, so it's also worth keeping an ear to the ground so you don't miss something new that could improve your search.

ANN is the secret sauce for better search

The vast, complex world of data demands efficient tools to navigate its labyrinths. This is where ANN can be the secret sauce that takes your similarity search from good to great. It offers speed and scalability, albeit at the cost of a slight accuracy compromise. And there is ongoing research with developments being made *weekly*, which will all contribute to the dynamic nature of ANN space. For instance, advancements in quantum computing and machine learning could lead to new types of ANN algorithms that are even faster and more efficient.

We've explored different ANN algorithms, each with its unique strengths and weaknesses. But ultimately, the optimal choice depends on your specific needs. Consider factors like data size, dimensionality, accuracy requirements, and resources. Experiment, explore, and choose the right algorithm to get the most out of ANNs. From image search to fraud detection, these algorithms can make a huge difference, revealing hidden connections and empowering data-driven insights fast.

So, the next time you search for the next song, movie, or video game, remember the silent heroes behind the scenes — the ANN algorithms — joining the dots and making connections.