Google Cloud

Blog

Developers & Practitioners

Find anything blazingly fast with Google's vector search technology

December 13, 2021

Kaz Sato
Developer Advocate, Google Cloud

Tomoyuki Chikanaga
Chief Architect, Groovenauts

Recently, Google Cloud partner Groovenauts, Inc. published a live demo of MatchIt Fast. As the demo shows, you can find images and text similar to a selected sample from a collection of millions in a matter of milliseconds:

Google Cloud

Blog

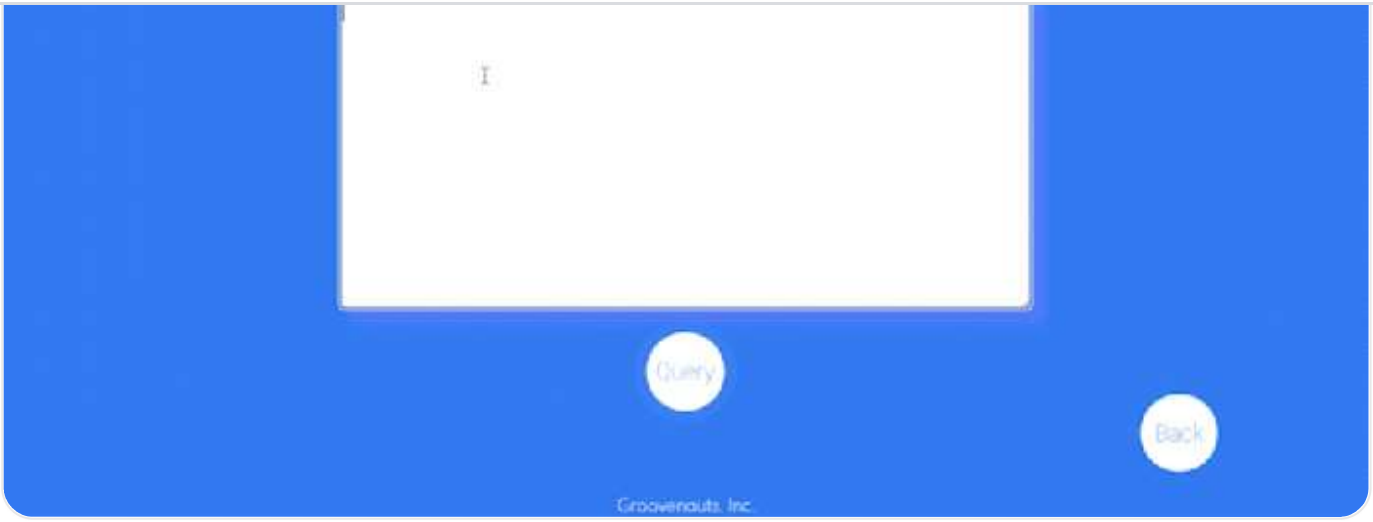
Image similarity search with MatchIt Fast

Give it a try — and either select a preset image or upload one of your own. Once you make your choice, you will get the top 25 similar images from two million images on Wikimedia images in an instant, as you can see in the video above. No caching involved.

The demo also lets you perform the similarity search with news articles. Just copy and paste some paragraphs from any news article, and get similar articles from 2.7 million articles on the GDELT project within a second.

Google Cloud

Blog



Text similarity search with MatchIt Fast

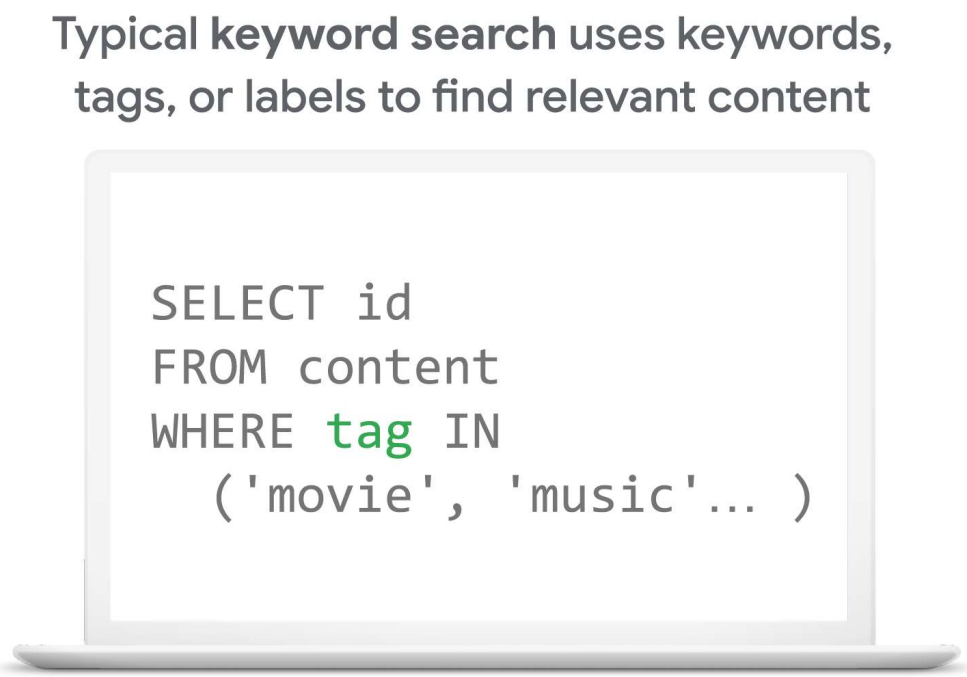
Vector Search: the technology behind Google Search, YouTube, Play, and more

How can it find matches that fast? The trick is that the MatchIt Fast demo uses the vector similarity search (or nearest neighbor search or simply vector search) capabilities of the Vertex AI Matching Engine, which shares the same backend as Google Image Search, YouTube, Google Play, and more, for billions of recommendations and information retrievals for Google users worldwide. The technology is one of the most important components of Google's core services, and not just for Google: it is becoming a vital component of many popular web services that rely on content search and information retrieval accelerated by the power of deep neural networks.

So what's the difference between traditional keyword-based search and vector similarity search? For many years, relational databases and full-text search engines have been the foundation of information retrieval in modern IT systems. For example, you would add tags or category keywords such as "movie", "music", or

Google Cloud

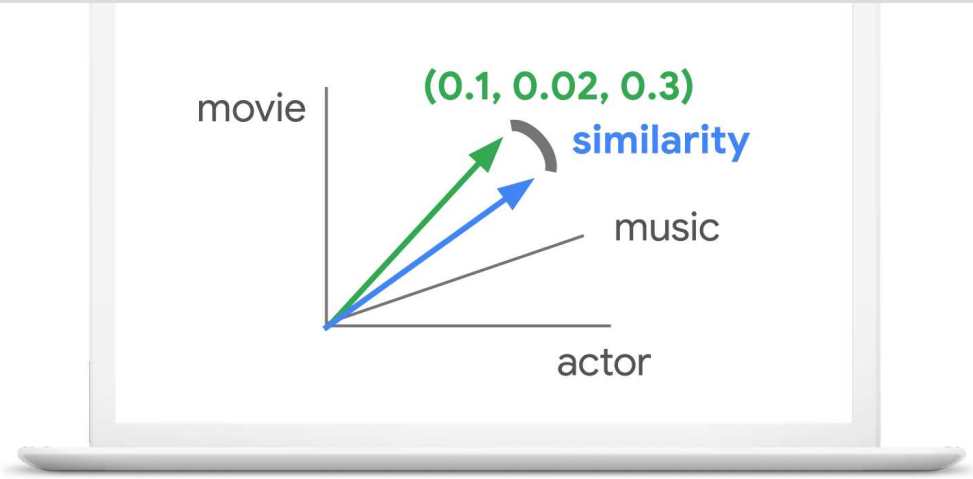
Blog



In contrast, vector search uses vectors (where each vector is a list of numbers) for representing and searching content. The combination of the numbers defines similarity to specific topics. For example, if an image (or any content) includes 10% of “movie”, 2% of “music”, and 30% of “actor”-related content, then you could define a vector [0.1, 0.02, 0.3] to represent it. (Note: this is an overly simplified explanation of the concept; the actual vectors have much more complex vector spaces). You can find similar content by comparing the distances and similarities between vectors. This is how Google services find valuable content for a wide variety of users worldwide in milliseconds.

Google Cloud

Blog

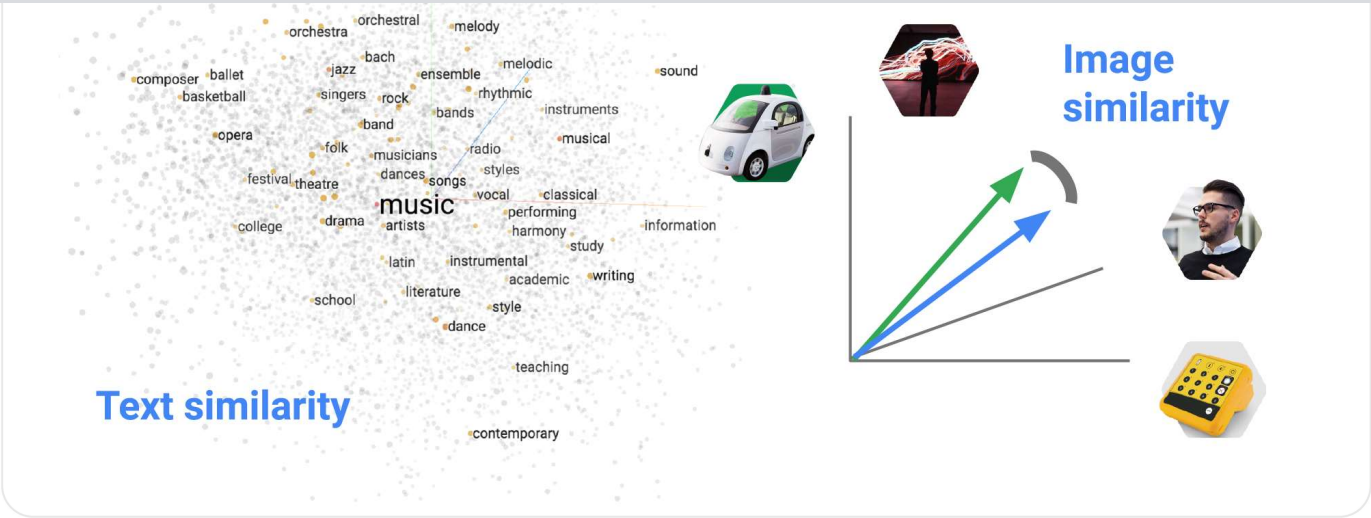


With keyword search, you can only specify a binary choice as an attribute of each piece of content; it's either about a movie or not, either music or not, and so on. Also, you cannot express the actual "meaning" of the content to search. If you specify a keyword "films", for example, you would not see any content related to "movies" unless there was a synonyms dictionary that explicitly linked these two terms in the database or search engine.

Vector search provides a much more refined way to find content, with subtle nuances and meanings. Vectors can represent a subset of content that contains "much about actors, some about movies, and a little about music". Vectors can represent the meaning of content where “films”, “movies”, and “cinema” are all collected together. Also, vectors have the flexibility to represent categories previously unknown to or undefined by service providers. For example, emerging categories of content primarily attractive to kids, such as ASMR or slime, are really hard for adults or marketing professionals to predict beforehand, and going back through vast databases to manually update content with these new labels would be all but impossible to do quickly. But vectors can capture and represent never-before-seen categories instantly.

Google Cloud

Blog



Vector search changes business

Vector search is not only applicable to image and text content. It can also be used for information retrieval for anything you have in your business when you can define a vector to represent each thing. Here are a few examples:

- Finding similar users: If you define a vector to represent each user in your business by combining the user's activities, past purchase history, and other user attributes, then you can find all users similar to a specified user. You can then see, for example, users who are purchasing similar products, users that are likely bots, or users who are potential premium customers and who should be targeted with digital marketing.
- Finding similar products or items: With a vector generated from product features such as description, price, sales location, and so on, you can find similar products to answer any number of questions; for example, "What other products do we have that are similar to this one and may work for the same use case?" or "What products sold in the last 24 hours in this area?" (based on time and proximity)

ads for viewers in milliseconds at high throughput.

- Finding security threats: You can identify security threats by vectorizing the signatures of computer virus binaries or malicious attack behaviors against web services or network equipment.
- ...and many more: Thousands of different applications of vector search in all industries will likely emerge in the next few years, making the technology as important as relational databases.

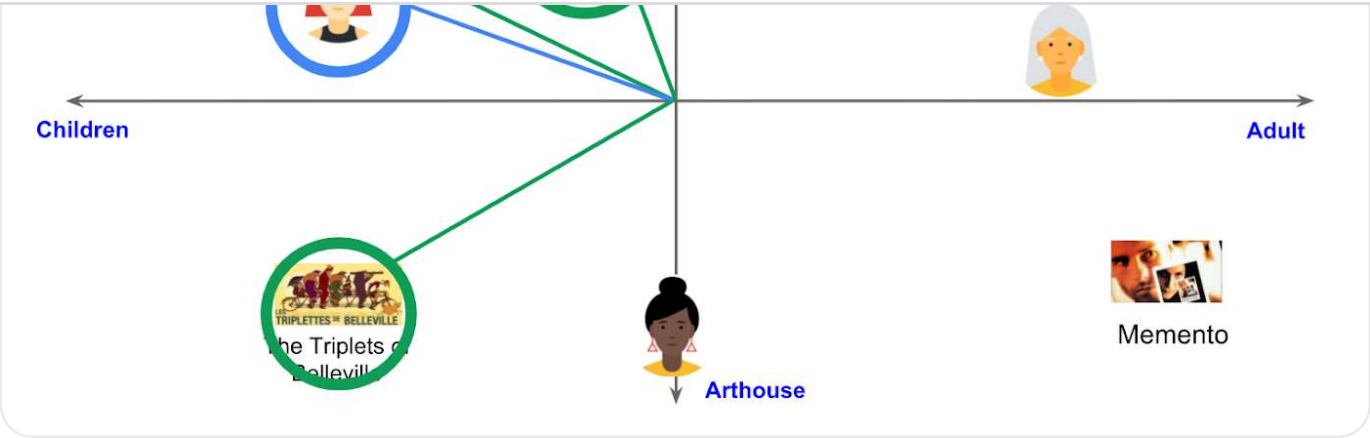
OK, vector search sounds cool. But what are the major challenges to applying the technology to real business use cases? Actually there are two:

- Creating vectors that are meaningful for business use cases
- Building a fast and scalable vector search service

Embeddings: meaningful vectors for business use cases

The first challenge is creating vectors for representing various entities that are meaningful and useful for business use cases. This is where deep learning technology can really shine. In the case of the MatchIt Fast demo, the application simply uses a pre-trained MobileNet v2 model for extracting vectors from images, and the Universal Sentence Encoder (USE) for text. By applying such models to raw data, you can extract "embeddings" - vectors that map each row of data in a space of their "meanings". MobileNet puts images that have similar patterns and textures closer to one another in the embedding space, and USE puts texts that have similar topics closer.

For example, a carefully designed and trained machine learning model could map movies into an embedding space like the following:



An example of a 2D embedding space for movie recommendation

(from Recommendation Systems, Google MLCC)

With the embedding space shown here, users could find recommended movies based on the two dimensions: is the movie for children or adults, and is it a blockbuster or arthouse movie? This is a very simple example, of course, but with an embedding space like this that fits your business requirements, you can deliver a better user experience on recommendation and information retrieval services with insights extracted from the model.

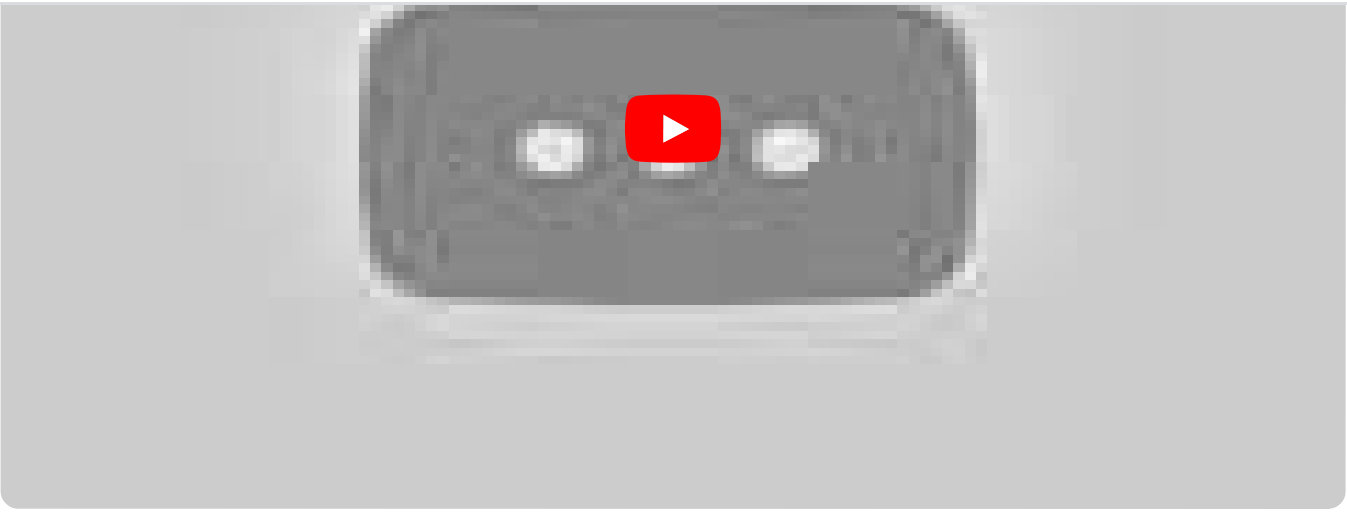
For more about creating embeddings, the Machine Learning Crash Course on Recommendation Systems is a great way to get started. We will also discuss how to extract better embeddings from business data later in this post.

Building a fast and scalable vector search service

Suppose that you have successfully extracted useful vectors (embeddings) from your business data. Now the only thing you have to do is search for similar vectors. That sounds simple, but in practice it is not. Let's see how the vector search works when you implement it with BigQuery in a naive way:

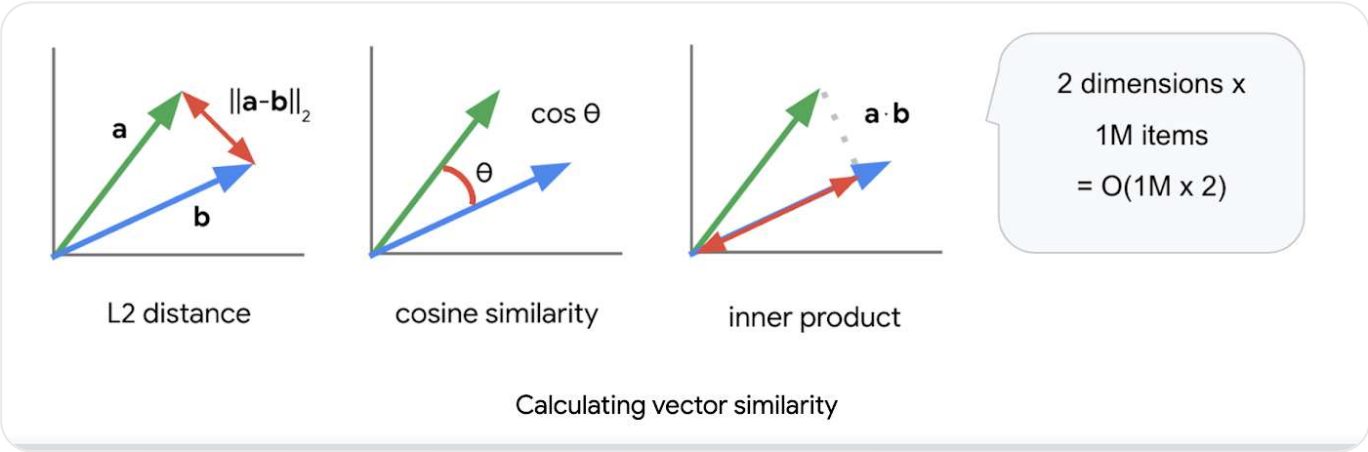
Google Cloud

Blog



It takes about 20 seconds to find similar items (fish images in this case) from a pool of one million items. That level of performance is not so impressive, especially when compared to the MatchIt Fast demo. BigQuery is one of the fastest data warehouse services in the industry, so why does the vector search take so long?

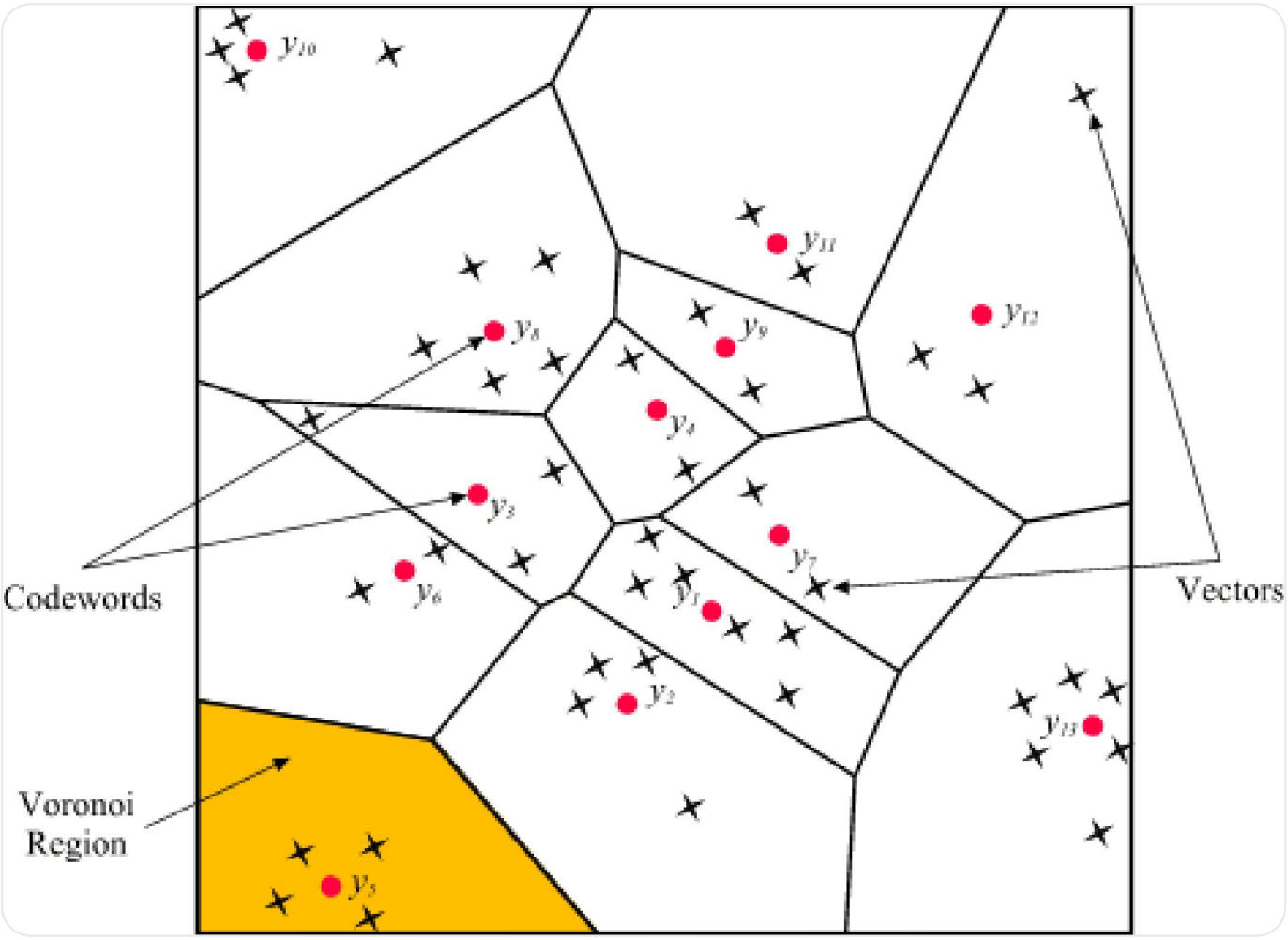
This illustrates the second challenge: building a fast and scalable vector search engine isn't an easy task. The most widely used metrics for calculating the similarity between vectors are L2 distance (Euclidean distance), cosine similarity, and inner product (dot product).



Calculating vector similarity

through all the entities for a single search, and the reason why the BigQuery demo above takes so long.

Instead of comparing vectors one by one, you could use the approximate nearest neighbor (ANN) approach to improve search times. Many ANN algorithms use vector quantization (VQ), in which you split the vector space into multiple groups, define "codewords" to represent each group, and search only for those codewords. This VQ technique dramatically enhances query speeds and is the essential part of many ANN algorithms, just like indexing is the essential part of relational databases and full-text search engines.

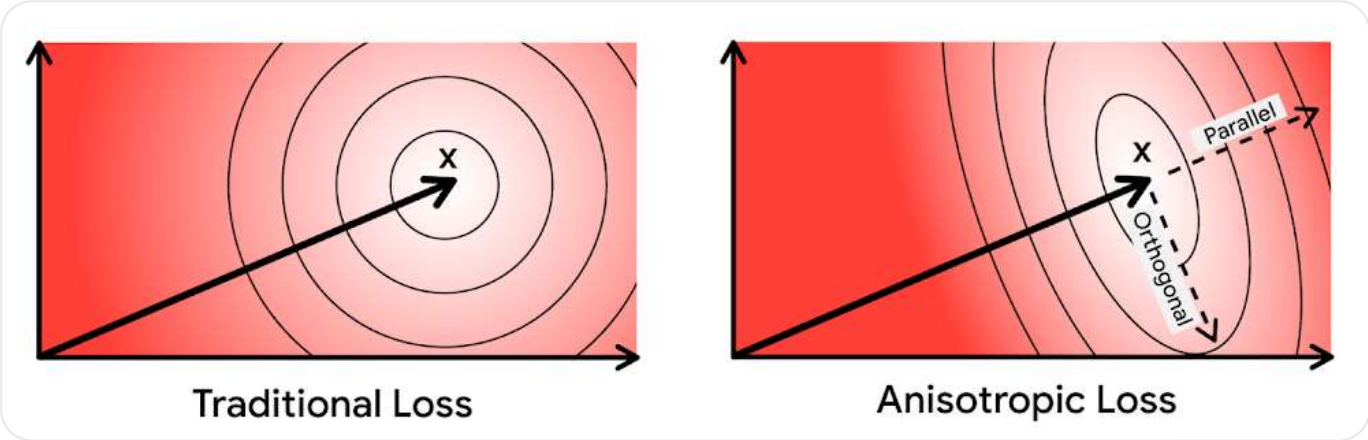


An example of vector quantization (from: Mohamed Qasem)

Google Cloud

Blog

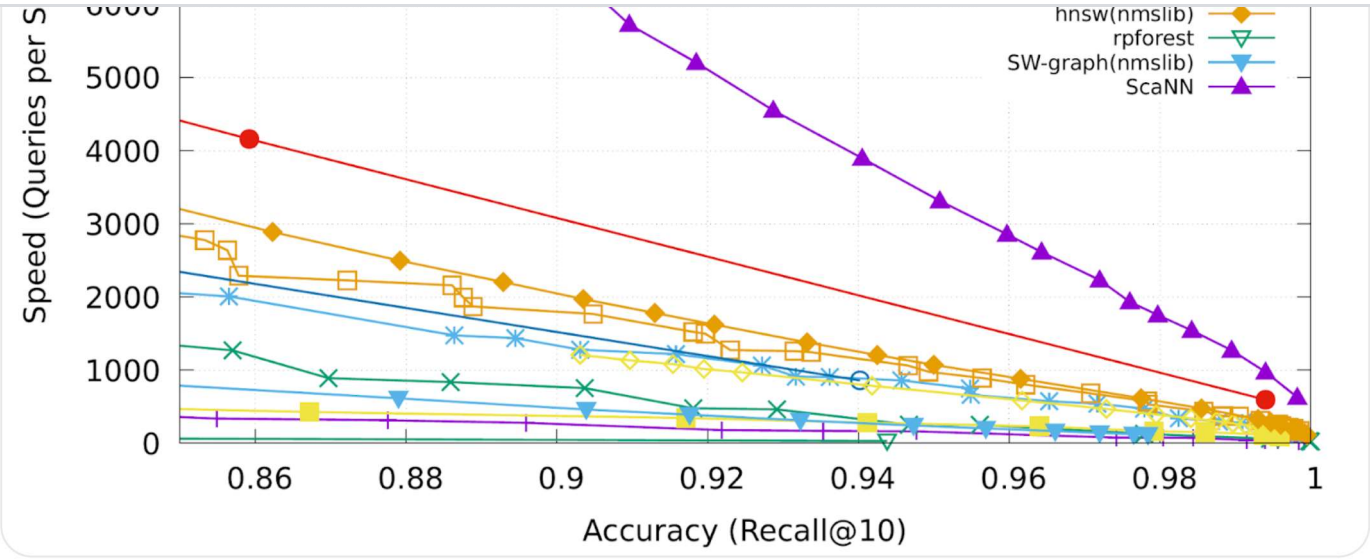
Last year, Google Research announced ScaNN, a new solution that provides state-of-the-art results for this challenge. With ScaNN, they introduced a new VQ algorithm called anisotropic vector quantization:



Anisotropic vector quantization uses a new loss function to train a model for VQ for an optimal grouping to capture farther data points (i.e. higher inner product) in a single group. With this idea, the new algorithm gives you higher accuracy at lower latency, as you can see in the benchmark result below (the violet line):

Google Cloud

Blog



ScaNN consistently outperforms other ANN algorithms in speed and accuracy benchmark tests

This is the magic ingredient in the user experience you feel when you are using Google Image Search, YouTube, Google Play, and many other services that rely on recommendations and search. In short, Google's ANN technology enables users to find valuable information in milliseconds, in the vast sea of web content.

How to use Vertex AI Matching Engine

Now you can use the same search technology that powers Google services with your own business data. Vertex AI Matching Engine is the product that shares the same ScaNN based backend with Google services for fast and scalable vector search, and recently it became GA and ready for production use. In addition to ScaNN, Matching Engine gives you additional features as a commercial product, including:

- Scalability and availability: The open source version of ScaNN is a good choice for evaluation purposes, but as with most new and advanced technologies, you can expect challenges when putting it into production on your own. For

with a recall rate of 95 - 98%.

- Fully managed: You don't have to worry about building and maintaining the search service. Just create or update an index with your vectors, and you will have a production-ready ANN service deployed. No need to think about rebuilding and optimizing indexes, or other maintenance tasks.
- Filtering: Matching Engine provides filtering functionality that enables you to filter search results based on tags you specify on each vector. For example, you can assign "country" and "stocked" tags to each fashion item vector, and specify filters like "(US OR Canada) AND stocked" or "not Japan AND stocked" on your searches.

Let's see how to use Matching Engine with code examples from the MatchIt Fast demo.

Generating embeddings

Before starting the search, you need to generate embeddings for each item like this one:

```
{
  "Id":"b5c65fea9b0b8a57bfa574ea",
  "Embedding": [
    0.16329009830951691,
    0.92436742782592773,
    0.00095699273515492678,
    0.011479727923870087,
    0.0089491046965122223,
    0.019959751516580582,
```

This is an embedding with 1280 dimensions for a single image, generated with a MobileNet v2 model. The MatchIt Fast demo generates embeddings for two million images with the following code:

```
class Vectorizer:
    def __init__(self):
        self._model =
tf.keras.Sequential([hub.KerasLayer("https://tfhub.dev/google/imag
enet/mobilenet_v2_100_224/feature_vector/5", trainable=False)])
        self._model.build([None, 224, 224, 3])  # Batch input
shape.

    def vectorize(self, jpeg_file):
        ...snip...
        embedding = self._model.predict({"inputs": input_tensor})
[0].tolist()
        return embedding
```

After you generate the embeddings, you store them in a Google Cloud Storage bucket.

Configuring an index

Then, define a JSON file for the index configuration:

```
    "approximateNeighborsCount": 150,
    "distanceMeasureType": "SQUARED_L2_DISTANCE",
    "algorithm_config": {
      "treeAhConfig": {
        "leafNodeEmbeddingCount": 1000,
        "leafNodesToSearchPercent": 5
      }
    }
  }
}
```

You can find a detailed description for each field in the documentation, but here are some important fields:

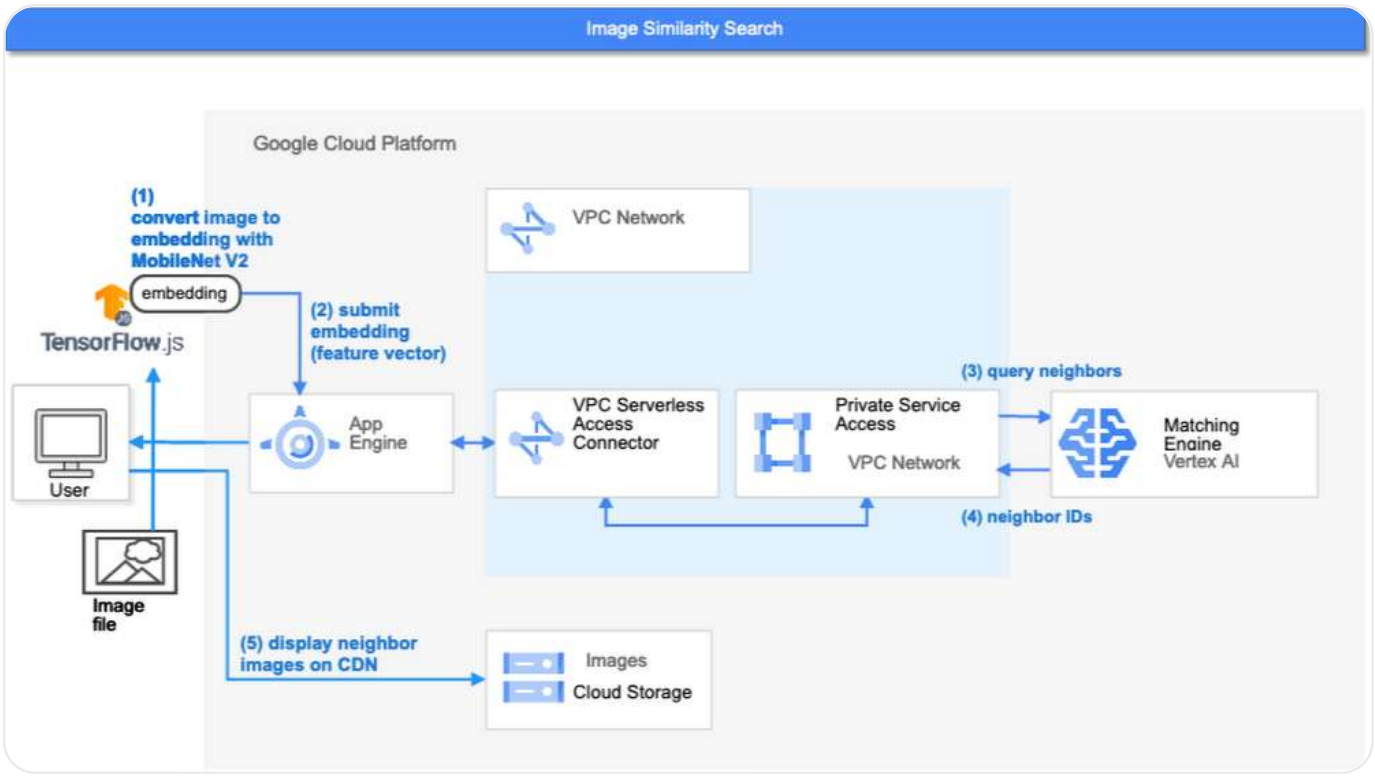
- `contentsDeltaUri` : the place where you have stored the embeddings
- `dimensions` : how many dimensions in the embeddings
- `approximateNeighborsCount` : the default number of neighbors to find via approximate search
- `distanceMeasureType` : how the similarity between embeddings should be measured, either L1, L2, cosine or dot product (this page explains which one to choose for different embeddings)

To create an index on the Matching Engine, run the following gcloud command where the `metadata-file` option takes the JSON file name defined above.

```
gcloud --project=gn-match-it-fast beta ai indexes create \
       --display-name=wikimedia-images \
```

Run the search

Now the Matching Engine is ready to run. The demo processes each search request in the following order:



The life of a query in the MatchIt Fast demo

1. First, the web UI takes an image (the one chosen or uploaded by the user) and encodes it into an embedding using the TensorFlow.js MobileNet v2 model running inside the browser. Note: this "client-side encoding" is an interesting option for reducing network traffic when you can run the encoding at the client. In many other cases, you would encode contents to embeddings with a server-side prediction service such as Vertex AI Prediction, or just retrieve pre-generated embeddings from a repository like Vertex AI Feature Store.

applications.

3. Matching Engine executes its search. The connection between App Engine and Matching Engine is provided via a VPC private network for optimal latency.
4. Matching Engine returns the IDs of similar vectors in its index.

Step 3 is implemented with the following code:

```
class MatchingQueryClient:
    ...snip...

    def query_embedding(self, embedding, num_neighbors=30):
        request = match_service_pb2.MatchRequest()
        request.deployed_index_id = self._deployed_index_id
        for v in embedding:
            request.float_val.append(v)
        request.num_neighbors = num_neighbors
        response = self._stub.Match(request)
        return response
```

The request to the Matching Engine is sent via gRPC as you can see in the code above. After it gets the request object, it specifies the index id, appends elements of the embedding, specifies the number of neighbors (similar embeddings) to retrieve, and calls the Match function to send the request. The response is received within milliseconds.

Next steps: Making changes for various use cases and better search quality

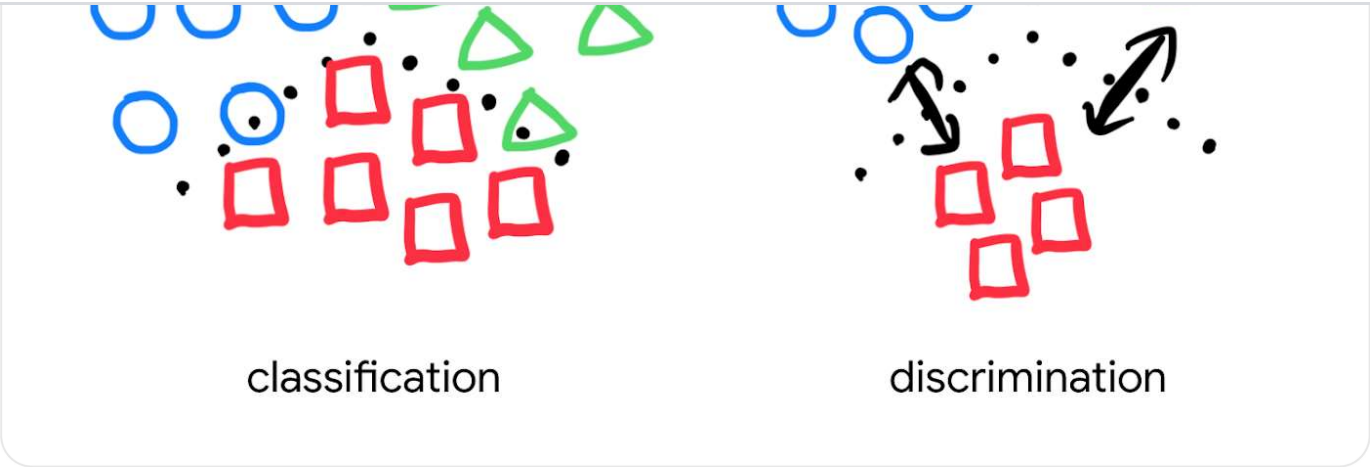
From the example above, you can see that Vertex AI Matching Engine solves the second challenge. What about the first one? Matching Engine is a vector search service; it doesn't include the creating vectors part.

The MatchIt Fast demo uses a simple way of extracting embeddings from images and contents; specifically it uses an existing pre-trained model (either MobileNet v2 or Universal Sentence Encoder). While those are easy to get started with, you may want to explore other options to generate embeddings for other use cases and better search quality, based on your business and user experience requirements.

For example, how do you generate embeddings for product recommendations? The Recommendation Systems section of the Machine Learning Crash Course is a great resource for learning how to use collaborative filtering and DNN models (the two-tower model) to generate embeddings for recommendation. Also, TensorFlow Recommenders provides useful guides and tutorials for the topic, especially on the two-tower model and advanced topics. For integration with Matching Engine, you may also want to check out the Train embeddings by using the two-tower built-in algorithm page.

Another interesting solution is the Swivel model. Swivel is a method for generating item embeddings from an item co-occurrence matrix. For structured data, such as purchase orders, the co-occurrence matrix of items can be computed by counting the number of purchase orders that contain both product A and product B, for all products you want to generate embeddings for. To learn more, take a look at this tutorial on how to use the model with Matching Engine.

If you are looking for more ways to achieve better search quality, consider metric learning, which enables you to train a model for discrimination between entities in the embedding space, not only classification:

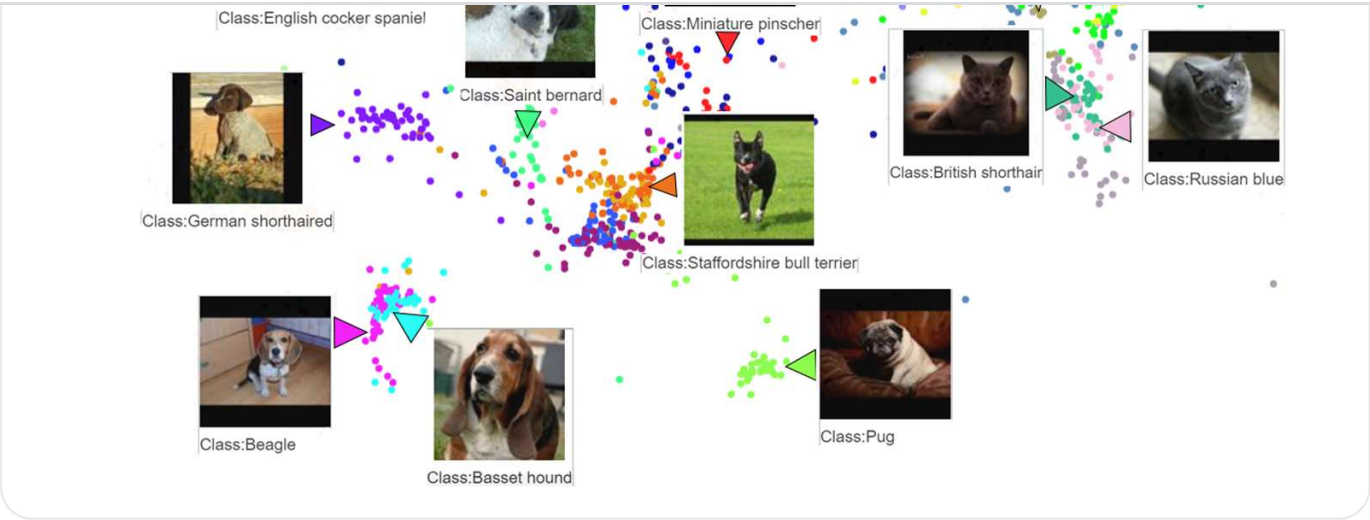


Metric learning trains models for discrimination with a distance metric

Popular pre-trained models such as the MobileNet v2 can classify each object in an image, but they are not explicitly trained to discriminate the objects from each other with a defined distance metric. With metric learning, you can expect better search quality by designing the embedding space optimized for various business use cases. TensorFlow Similarity could be an option for integrating metric learning with Matching Engine.

Google Cloud

Blog



Oxford-IIIT Pet dataset visualization using the Tensorflow Similarity projector

Interested? Today, we're just beginning the migration from traditional search technology to new vector search. Over the next 5 to 10 years, many more best practices and tools will be developed in the industry and community. These tools and best practices will help answer many questions, like... How do you design your own embedding space for a specific business use case? How do you measure search quality? How do you debug and troubleshoot the vector search? How do you build a hybrid setup with existing search engines for meeting sophisticated requirements? There are many new challenges and opportunities ahead for introducing the technology to production. Now's the time to get started delivering better user experiences and seizing new business opportunities with Matching Engine powered by vector search.

Acknowledgements

We would like to thank Anand Iyer, Phillip Sun, and Jeremy Wortz for their invaluable feedback to this post.