Section 4: Vector Embeddings And Vector Databases
Course: Ultimate RAG Bootcamp Using LangChain, LangGraph and LangSmith
Section Number: 4
Total Videos: 5
Date Created: 2024

Video 1: Introduction To Embeddings And Vector Databases
Video Order: 1/5
Topics: Embedding fundamentals, Vector databases vs traditional databases, Similarity search, Cosine similarity, Feature representations, Model choices (OpenAI & Hugging Face)
Difficulty: Beginner → Intermediate

— Overview —
Hello guys.

So we are going to continue the discussion with respect to Rag.

— Recap: Ingestion & Splitting —
In our previous video we have seen various data ingestion and parsing techniques. Uh, and the last one was like if our data is available in a specific databases for those kind of data, which is in the form of structured format, how do we read it? How do we convert that into a document we have already seen?

— Where We Are in the RAG Architecture —
Now if I go back again to the architecture, right rag architecture, we have completed all these things. We have seen how to do the document splitter and how with the help of text splitter, we are actually converting a document into chunks right now. The next step is that how do we go ahead and use some kind of embedding models and convert that into some embeddings. Right. So this is what we are going to now focus on. Our main aim is that once we get the data in the form of chunks, which all different types of embedding models, we can go ahead and apply. And the idea behind embedding model is very simple whatever documents or data is available, how do we go ahead and convert that into vectors?

— What & Why: Embeddings at a Glance —
Now in this video I'm just going to go ahead and give you a brief introduction about embeddings and why embeddings can be really, really important and useful and how it is different when we compare it to the traditional databases. So all those things we are going to go ahead and discuss. Okay.

— Traditional Databases (Exact Match Retrieval) —
Now let's say, uh, if I just take an example of a traditional database. So let's say if I have a traditional database. So this is my traditional database. Now for this traditional database let's consider this. Let's say if this is my traditional database. And since we are going to go ahead and discuss about embeddings here, uh, now what we are specifically going to use is vector databases. And I'll talk about it. Why vector databases will also be important vector database.

Now in a traditional database. Let's say that my search term, let's say I want to search something related to Cat. Or let's consider that in this particular database I have some of the items. The items can be let's say I have cat, I have dog, I have kitten, right. And let's say I have puppy. Now these are the information that is available in this specific database. Now whenever I go ahead and search anything related to Cat, then what this will do based on a specific query, we will go ahead and pick up the exact match that is available inside this database. So my output result. When I go ahead and search for Cat, I am going to go ahead and get Cat right. It is very simple because the matching term is nothing but cat. So this is what usually happens in a traditional database.

So if you probably go ahead and consider a SQL database, right. MySQL or SQL server. Let's say I go ahead and write a query, select star from employee table where the name is equal to crush. So I'm searching for a record wherein I'm putting a condition saying that the name should be equal to crush, then that specific record will be displayed, right? Similarly, if you have multiple tables in a traditional database, you may go ahead and write a complex query, and based on that particular query, you will be getting a exact match. So this is really important. The purpose of a traditional database like SQL, MySQL, NoSQL. Here you are going to get only exact matches exact matches.

— Vector Databases (Semantic Similarity Retrieval) —
Right now. What is the importance of a vector databases? We'll try to understand it. And for a vector databases, since we are specifically going to use embeddings. Now what is embeddings? Embeddings is nothing. But here embeddings is a technique wherein if I have any kind of text data, if I apply embeddings on top of it, I will be able to get some vector format or vector representation for this particular data, which is in the form of text.

Let's say for this particular text, let's say the text is like Apple okay. So if the text is Apple, and if I apply some embedding techniques in this, I will be getting a vector representation which looks like this. Let's say I'm just considering this. It looks something like this okay. Now this vector representation again for this particular course you should have a basic understanding what is embeddings. Because these all are the part of NLP techniques. But here I will just give you a brief idea about embeddings also.

Now whenever I have this kind of representation. So these are nothing but numerical representation. I cannot just go ahead and directly use a word and give it to my model. My model will not be able to understand this. Instead, it will be able to understand some kind of numerical representation. And this numerical representation is nothing but vector representation.

Now for storing this vector representation we can use something called as a vector database. So let's say if Apple is something like this, we can store this information over here in the form of a record or a text ID or document ID or something. Right. So we can store this entire vector representation over here.

— Similar Meaning vs Exact Match (Cat → Cat & Kitten) —
Now let's consider that I have the same thing. So let's say if I have the same data over here. So I'm I'm just going to go ahead and consider, let's say this is my same data that I have. Okay. Let's say if this is oops this is the same data that I have. Okay I'm going to paste it over here. Now what is the main purpose of the vector database is that if I go ahead and search for Cat again Okay, now here it is not going to do the exact match, but instead it will go ahead and find the similar meanings word similar meanings word right. Or it will go ahead and find with respect to cat which are the similar words over here. So over here, if you go ahead and see the result right here with respect to the result, two things you will be able to get. See, cat is obviously cat only kitten. Also we can basically talk about cats right. So here what we are going to get. The result you are going to get is cat and kitten.

So a vector represent now why you are getting this two words over here as in the form of results. Because if you consider at the end of the day in a vector databases, we will be storing the information of all the text which is in the form of vector representation. So this cat and kitten will have a similar kind of vector representation because these words are almost similar. So all the similar words. So usually what happens is that in deep learning in NLP we have good embedding models, word embedding models, right. Which are trained with huge amount of data. These are trained with huge amount of data, huge amount of data. And this embedding models main aim is whatever word it has, whatever word it takes, it will be able to provide a vector representation for that word. Vector representation for that word vector representation will be in such a way that similar kind of words, let's say if I take an example of cat and kitten, since both the words are same, the vector representation may look like this .2.3.4, let's say, and kitten may look something like this. .2.4.3.

— Cosine Similarity (How We Compare Vectors) —
So here you can see these are my vectors right. These are the vector representation for this word cat. And this is for kitten right. You can see that both the vector representation are very much near right. So what we do is that in order to compare whether these two are

same or not, we apply something called as cosine similarity. This is just one of the metric. And based on this cosine similarity it actually says that how similar this both words are. I will show you this all in the terms of coding right. But I hope you got a idea about what is vector representation. And with respect to any kind of vector databases where you are specifically applying embedding techniques and storing those vector representation inside the vector database, you will be getting a result based on finding similar meanings or similar match. That is how it is completely different from a traditional database.

— Embedding Models (OpenAI & Hugging Face) —
Now this vector database is or vector embeddings, right? Vector database is basically means we will be using this databases to store this embeddings. Right. This vector representation. And to get this vector representation we use embedding models. Now there are different different embedding models. We have models from OpenAI. We have open source models like hugging face right hugging face and all.

— Intuition via Feature Representation (Movie Analogy) —
But now the question arises, Krish, why you are representing in this format, let's say, why you have written point two, .3.4, what this entire words also represent that also will go ahead and discuss. So let's, let's take one, uh, you know, example. I hope everybody has seen the movie of marvels. Right. And many people marvel, right. And, you know, Marvel is having the biggest franchisee in the world, right? So let's consider a movie from Marvel, let's say Marvel. There is a movie of Iron Man, right? Iron man.

Now, let's say this Iron Man is there. Now, I may go ahead and create an embedding model, or I may use some kind of embedding model. And this embedding model, whenever it takes this Iron man as a text, it may convert this into a vector. The vector may look something like this. Right. And for this I will just assume some of the features. So these are nothing. But this vector representation is also called as feature representation. Why we say feature representation I'll just talk about it. Feature representation.

Let's say my first feature is nothing but action okay. The second feature is nothing but comedy. The third feature is nothing but suspense. Okay, let's consider this three feature. Now you know that Iron Man, whenever we talk about it, is a hardcore action movie, right? So let's say I go ahead and write this vector as 0.95. .95. Okay, so this iron man with respect to action is represented by a vector .95. Iron man in comedy. So there is not much comedy in this specific movie. So I'm just going to give one vector representation as point two. See, this is not how a model is trained, but I'm just trying to give you an idea. What does this feature representation of vector representation basically mean? Then Iron man two suspense, right? Suspense. There are also some kind of suspense, so I'll keep it as point six.

Now similarly, let's say there is another movie like Hulk and you know that is also a marvel movie, right? So Marvel, whenever we talk about Hulk in action, it is .96 comedy. Yes, it will be having point four and suspense. Yes, there is a kind of suspense point seven. But let's say that if I take some another movie, another movie basically means Sherlock Holmes. Sherlock Holmes, Sherlock Holmes. Action. There is not much kind of actions. So let's say, yeah, there they may be action, but let's say that I'm giving it as 0.6, right? Comedy. Sherlock Holmes has some kind of comedy suspense. There is a lot of suspense in this, so I'll put it as point nine right now here you can see all this numbers that you are able to see. So the Iron Man is basically represented by this vector Hulk is represented by this vector. And Sherlock Holmes is represented by this. And each of this vector represents this features representation. Right. Like it is a correlation of Iron Man to action movie, iron Man to comedy, Iron Man to suspense.

— 2-D Plot Thought Experiment —
Right now, let's say if I am seeing a Netflix movie and in the Netflix movie, I go and see an Iron Man movie. Now, after completing Iron Man, which movie do you think Netflix will recommend me? This is just a simple example for my side, so obviously you will be able to see from this vector representation and this vector representation if we go ahead and find the cosine similarity, right. The cosine similarity formula is very simple. It is nothing but Iron man. So let's consider that this is vector A. This is vector B, this is vector C. It is nothing but a dot product of B multiplied by normalization of A. That basically means magnitude of a and magnitude of b. So if we go ahead and apply this specific formula just by seeing this vector representation, don't you think that Iron Man and Hulk are almost similar? So what will Netflix recommend? Netflix will definitely go ahead and recommend Hulk, because if you go ahead and calculate the cosine similarity and if you just go ahead and plot it, plot it with respect to similarity.

So let's say I'm plotting it in this two dimension. Okay, I'm obviously there is three dimension, but I'll just try to plot it at two dimension by taking action and comedy. So let's say if this is action, this is comedy, right? So the first plot, you will be able to see Iron Man 0.9 5.2. Right. So 0.950.2. Let's say this is one, two, something like this, right. Or this is this is 101. And this is zero one. This is 0.5. This is 0.5. Right. So 0.92 is nothing but somewhere over here right. So this is Iron Man. Hulk will be able to see 0.96.4. So it will be somewhere over here. So this is Hulk. This is Iron Man. So this is Hulk. This is Iron Man. So just see this how near this is, right? So that basically means this two movies are almost similar. If I consider Sherlock Holmes, it is 0.6, 0.7, 5.6 basically means over here, 0.75 basically means over here. So this is where my Sherlock Holmes comes, Sherlock Holmes come. So the distance is very far When you consider this to distance, it is very near. So if Netflix is

showing you Iron Man, then the next movie that is recommended will be Hulk because this is very near to them, right.

— What We'll Do Next —
So this is the purpose of using embeddings, right. And that is how when we do a similar meaning search. Right. We get similar things over here based on this vector representation which is near to each other. Right. And this is the entire idea behind embeddings okay.

Now in our next video we will try to see again, I don't want to go much into theoretical things because these are some prerequisites that you should already know in NLP. Now in the next video, what I am actually going to do is that we are going to learn about how we can perform different, how we can use different embedding models and convert some data into vectors. Right. And later on we'll also talk about different kind of vector databases we can basically use. Okay.

So we will focus on two. type one is open. I will also go ahead and focus on hugging face. Right. So both of them, because you should also know how to use embedding model models which are open source and how to also use paid. Right. So open I will have paid hugging face will be having some of the open source models, but the main idea is that we will take our text data or document and then we will go ahead and use some embedding models. So here I will be having my embedding models. And with the help of this embedding models we will go ahead and convert that into vectors. We will go ahead and convert that into vectors. That is what we are basically going to do. And this I will show you with the help of practical examples as we go ahead. Right.

— Wrap-up —
But I hope you got a idea about embeddings. What is the difference between traditional databases and vector databases? Uh, yeah, this was it for my side. I will see you all in the next video. Thank you. Take care.

Video 2: Visualization Of Embedding And Cosine Similarity
Video Order: 2/5
Topics: Visualizing embeddings, 2D plots, Cosine similarity, Interpreting similarity scores, High-dimensional embeddings
Difficulty: Beginner → Intermediate

— Overview —
Hello guys.

So we are going to continue the discussion with respect to embeddings.

Now, uh, already we have understood some of the theoretical explanation about what exactly embeddings is, what exactly embedding models is. And, uh, you know, we also got a brief idea about vector databases, right.

— Project Setup —
So now, uh, what we are going to do is that I'll go ahead and create my second folder. So let's say that I name this folder as vector embedding vector embeddings and databases okay. So we are going to specifically work on two different things now vector embeddings and databases. Um that includes how we go ahead and like like what? Let's say that once we go ahead and read all the documents, you know, then we go ahead and apply some document splitter, we convert that into chunks. And then how with the help of embedding models, we go ahead and convert that into vectors. So over here I will quickly go ahead and create my files. Embedding dot Ipynb. Okay. I will go ahead and select a kernel over here. And then we can go ahead and start.

— Quick Definition —
So first of all I would like to provide a brief definition about what exactly are embeddings. Okay. So think of embeddings. Um it is a way of translating word into a language that computer understand that is nothing but numbers. So these all numbers are nothing. But we basically say it as vectors okay. And this is how we specifically go ahead and use it.

— Plan for This Video —
Now over here in our previous example, I had drawn some kind of visualization. Right. Like this. You could see that I have written Iron Man Hulk over here, Sherlock over here. So first of all, what we'll do is that in this specific video, we will go ahead and see that, let's say if I have some of the words right. And for all these specific words, how do we go ahead and just plot it and see that whether it is very near to each other or not.

— Imports & Installation —
Okay, so first of all, what I'm actually going to do quickly, I will go ahead and import numpy as NP. I'm going to use numpy, and I'm going to also use matplotlib dot pyplot dot pyplot as

plt. Okay. And then I'm going to specifically use this two libraries. Let's say if this library is not there you can see matplot is not there. So what I will do I will quickly go ahead and update my requirement dot txt file. So let's say over here I will go ahead and write matplotlib okay. And why I am doing this because I want to visualize the embeddings right. Like how we had seen in this particular diagram. Can we go ahead and visualize? I'll take some 3 to 4 different kind of vector representation of a words. And then we will try to just plot it in this way so that you'll also get an idea about embeddings. You know how exactly it looks in terms of visualization.

Now I will go to my command prompt and it is already there. My my environment is already activated. So you also need to make sure that. And I'll write you've add requirement dot txt. So here you can see that already resolved packages. And here matplotlib has got installed right. So matplotlib by parsing all these things have got installed. Now I'm going to go back again to my embeddings and then execute this. Now it should work absolutely fine. Okay.

— 2D Example Setup —
Now what I'm actually going to do is that I will just take a simplified 2D examples. One very important point is that guys over here for this vector representation I use three feature representation. Right. So if you go ahead and see different different embedding models, they will be having different dimensions with respect to this feature representation. Let's say if I go ahead and use OpenAI some of the embedding model right. This may have 1500 dimensions okay. Hugging face it may have 384 dimensions. So different, different models have different capabilities. And based on that, whenever they take a word, it can convert that word into different dimensions of feature representation. Right. And that is how it is specifically trained. Every model has different capability. Right. It may not be two features or three features. Right. They may have different different features. Right.

So uh, in this particular example, I am considering some of the words where I've just given two features. Let's say .8.6. I'm not naming the features itself because we also don't know, like how open I may have trained the embedding models or how hugging face may have trained the embedding models. Right. So I'm considering some of the very common words like cat, kitten. So here you can see .8.6 is the kitten is .75.65. So it is very near to each other. Dog is .7.3, puppy is .65.35 car and truck. It is no way related to this. So what is my plan is that I will try to plot all this words, and it will then give you an idea about like how this word looks like, right?

— Plotting the Embeddings (2D) —
So for this again it is basic Python. We're going to use plt dot subplots figure size of eight comma six I am reading every items inside this word embedding. So for word comma coordinates right in embedding dot items. And I am taking the coordinates of zero in my x

axis. And this one on my y axis with a solid color of 100. Okay. Then uh I'm annotating with this specific information. So this is basic matplotlib visualization. So now I will just go ahead and show you. So here you can see this is my how my simplified word embedding in 2D space looks like here kitten and cat are is almost similar. Puppy and dog is almost similar. Car and truck is almost similar. Okay, so this is an example of vector representation. But in this particular example I've just taken two vectors itself. Right. But as I said with respect to different Embedding models. We definitely get different kind of, um, you know, we basically get different dimensions with respect to the vector representation.

— Measuring Similarity (Cosine Similarity) —
Now the next step is that, uh, if you go ahead and see this right, I told you, right. Netflix. If iron movie is seen, how does it go ahead and probably suggest like which is should be the next movie, right. And for this we apply cosine similarity. And the basic cosine similarity formula is something like this. It is nothing but vector a dot product of vector B divided by magnitude of A and magnitude of B, right? So we will also go ahead and apply cosine similarity between two vectors. So let's do that. And uh here I'll just go ahead and write something like measuring similarity. Okay. Measuring similarity. And for this we will try to use cosine similarity. If I want. Okay.

— Implementing Cosine Similarity —
So now quickly I will go ahead and define cosine similarity. And remember one thing about cosine similarity is that if we are using cosine similarity. These are how the outcome looks like. If the result is close to zero or close to one, it is very similar. If the result is close to zero, not related. If the result is close to minus one, it is opposite. Okay, so that is what the cosine similarity gives you an output. As I said, the formula of cosine similarity is a dot b that is nothing but a dot product divided by magnitude of A and magnitude of B, is there? Right? So first of all, what we need to do we will consider two vectors right. So vector one and vector two. And first of all what we are basically going to do. We are going to do a dot product of this. So here I will just go ahead and write. This is my dot product in order to calculate it. And here I will go ahead and write NP dot dot. And let's go ahead and write vector one comma vector two. So this is what we basically do with the dot product. Right.

And for doing the dot product we use numpy NP dot dot okay. Then what we do in order to calculate our magnitude of a I will just go ahead and write norm underscore a is equal to in order to calculate it. I also have an inbuilt function in numpy, which is nothing but a dot norm. So if you go ahead and use this particular function, you'll you should be able to get the magnitude of vector A okay. So here we are basically going to get the vector a. Similarly the same formula will go ahead and apply for the vector two okay. So this will basically be my b. So now here I've got my magnitude of a magnitude of B. And finally I need to just go

ahead and return dot product dot product divided by divided by norm. Underscore a. With a multiplication of norm underscore B. Remember this dot product is something different with respect to multiplication okay. So in a dot product vector or vector multiplication will happen. Right? For every vector representation or every dimension of the vector, they'll be getting multiplied with this other vector itself, right? So this is what is my entire function of cosine similarity.

— Examples (Cat–Kitten vs Cat–Car) —
Now we can go ahead and apply this. So let's say that I have some of the example of the vectors over here. The cat vector looks like this. The kitten vector looks like this. The vector looks like this okay. Now if I just go ahead and apply the similarity for cosine similarity, let's say for cosine similarity I go ahead and apply for cat. Let's say cat and kitten kitten kitten similarity I'm going to go ahead and apply okay. So for this I'm going to go ahead and apply my cosine similarity. And I'll give my two vectors okay. So here I'm going to go ahead and give my two vectors. One is cat underscore vector comma. And the next one is nothing but kitten underscore vector okay. Now if I just go ahead and print the cat kitten similarity you should be able to get the output okay. So let's execute this. So here you can see that my output is nothing but 0.996. So already I've told you that if it is close to one, it is very similar. That basically means it's cat and kitten vector are almost similar.

Similarly, I can go ahead and apply the different similarity for this. So here I'll go ahead and apply for cosine similarity. And here let's say if I go ahead and take car cat and car vector, then here you'll be able to see that I'm getting point -0.4367. That basically means it is completely like opposite. It has opposite meaning. So obviously car and cat are nowhere related at all. But here we have given some kind of vector representation that matches with this, right? So that is the reason we are able to get this.

— Notes on Vector DBs & High Dimensions —
So this is how uh you go ahead and apply a cosine similarity. Uh, or you just can define a function with respect to cosine similarity and do this, um, similar like internally in the vector databases. Also this all functionalities will be used that we will go ahead and see it as we go ahead. But yes, this was just an idea about visualization of embeddings. And then we have also spoken about how do you find the similarity between two different vectors. When we talk about vector databases, we will still talk about different, um, you know, different, uh, similarity algorithms that we are going to implement. Uh, there is also something called as similarity search internally. You know, this cosine similarity may get applied. But yes, I will just go ahead and talk about it as we go ahead.

But this is just an example of how you can go ahead and visualize some vectors. Obviously it is not. It is difficult to visualize vectors that are greater than three. Right? I cannot like in

some of the algorithms, it will give you 1500 dimensions of vector representation that we cannot go ahead and plot it. But if you just apply some principal component analysis where you do dimensionality reduction, convert it into two vectors, then all the words will look something like this.

— Wrap-up —
So two different thing. Amazing things we learned in this particular video. One is how do you visualize it and how do you go ahead and measure the similarity. So yes, this was it from my side. I'll see you in the next video. Thank you.

Video 3: Creating Your First Embeddings With HuggingFace Embedding Models
Video Order: 3/5
Topics: Hugging Face overview, Sentence Transformers, LangChain integrations, Embed query vs. embed documents, Model choices & dimensions
Difficulty: Beginner → Intermediate

— Overview —
Hello guys.

So we are going to continue the discussion with the topic that is embeddings. And in this specific video we are going to create our first embeddings. Now I hope everybody has got an idea what exactly embeddings is. You know, at the end of the day we will be taking some text data and we will be converting that into some kind of vector representation. And for this we will be using various embedding models. I will be showcasing you two different types of embedding models that is OpenAI and Hugging Face. Hugging face has some cool open source models which you can directly use it. And for OpenAI you need to have OpenAI API key. So to use the embedding models.

— Exploring Hugging Face —
So let's go ahead and before I go ahead, first of all, let me just go ahead and show you about hugging face. So here what I will do I will go ahead and search for hugging face. So once you go in and right hugging face.co right. And let's say that I will just go ahead and log in over here. So once I login after login here, you can actually see that this is how a hugging face looks like, right? Hugging face website. Now, if you don't know about hugging face, what exactly it has, it has huge different kind of models that is available for different different tasks, right? So let's say that if you want to perform text generation, you want to generate image text to text. So for all this kind of tasks, you know you have different different models over here, right. Not only that, let's say I will just reset all this task. Let's say you want to go ahead and use different libraries, different languages. You know, let's say I want to go ahead and use transformers. I want to use TensorFlow. Along with that, if you want to perform any kind of task, let's say the task is like audio, text to text, computer vision, you know, natural language processing like text classification and all. So this hugging face has huge number of open source libraries which are hosted in that specific platform, right? So what you can actually do is that all those libraries also you can directly use it right now in our case we are going to focus on embedding models.

— Finding Embedding Models —
Now with respect to embedding models, if you probably go ahead and see over here right. So here you can see that with respect to natural language processing, there are so many different categories of tasks that you can actually do. You have feature extraction. You have

text generation, you have text classification, token classification. And all these are the models which you can actually use it. So what I am actually going to do, I'll go ahead and search for some of the embedding models. Right. If you want to go ahead and search there's something called as sentence transformers okay. So if I go ahead and search for sentence transformer here you can see different kind of models are over here. Like all mini lm, l6, v2. So this is the kind of embedding models. Then here you have paraphrase multilingual model. Then you have labs. So out of this you know you can use any of those embedding models. So if you also go ahead and search for embedding models in hugging face. Right? In hugging face. So here you'll be able to see various models itself, right? So if you just go ahead and click this getting started with embeddings. Here you have a lot of blogs that is available. You can directly open the code in Google Colab and you can directly use it. So if I'm just searching for this kind of models here, you can see there's so many different models that are already available which you can actually use for embedding type. So here you can see we have selected embeddings automatically. You can use all these specific models. But I want to go ahead and focus more on the open source models smaller models so that you can directly use this. And uh again we are using long chain right at the end of the day and we will try to convert our text into vectors. So for that we are going to use this all mini l6 v2. You can also go ahead and use different models based on your requirement. You can search from here and you can actually do it right. So I'm going to use this.

— Installing Required Libraries —
Now let's start with our first model itself. So what I'm actually going to do over here for this, we will be having some requirement with respect to the library. So I'm just going to make a simple category saying as embeddings. And we will be importing two different kind of. So we will be installing two different kind of libraries. One is lang chain hugging face and the other one is uh, let's say that after lang chain hugging face, since we also want to use the sentence transformer. So after writing lang chain hugging face, this is one of the library that we will be requiring. The other libraries that we will be using is nothing but sentence transformer, right? So sentence transformer will be my another library. Okay. Transformers okay. So these two libraries will be using and I already have Lang chain OpenAI. So with the help of this we will be able to import the OpenAI embedding models. But right now in this particular video we'll be focusing more on hugging face embedding models. Okay. So now it's time that we go ahead and install this library. So I will just go ahead and open my command prompt and quickly go ahead and write UV add minus our requirement dot txt. Okay. So here you can see that sentence transformer okay the spelling is wrong because I have to go ahead and write Transformers. Okay. Now I will just go ahead and do the installation. Now with respect to the installation here you can see that this installation has

been done. I think sentence transformer has already been installed in this. Okay. So here is my sentence transformer. I think I had already done it so we don't require it. Again that basically means the installation has been completed.

— Initializing a Hugging Face Embedding Model —
Now what we are going to do over here is that we are going to go ahead and write our embedding code. Okay. So in the embedding code I'm going to go ahead and use my Lang chain hugging face. And we will try to use a library. Right. Uh, the same sentence transformer A model for doing the text embeddings right? So first of all, what I will do, I will go ahead and import from Lang. And there is something called as underscore hugging face. I'm going to go ahead and import hugging face embedding. Okay. So hugging face embeddings. Now this hugging face embedding library will actually help you to call any kind of models that is available in the hugging face itself. Okay. So here we will go ahead and initialize initialize a simple embedding model. So here obviously no API key is needed. Okay. So this is the most important thing since we are using the hugging face embedding library before when we are using Lang. Right. And when we used to use separately hugging face, at that point of time we required API key. But because of this library now it has become very much easy in order to call any kind of hugging face models, right? So now what we are going to do I'm going to go ahead and create my embeddings. So embeddings is equal to we will be using this hugging face embedding. And then I will just go ahead and give my model name. So model underscore name is equal to. Now remember how do I give my model name. That is really important. So I will go back to my hugging face since I'm going to use this right. So I will copy this particular model. Right sentence transformer all mini LM. Let's see. This is a sentence transformer model. It maps sentences and paragraph to a 384 dimension dense vector space and can be used for tasks like clustering or semantic search. Now see. At the end of the day, whenever we are working with vector databases, we are working with embeddings. We are going to perform this kind of task that is semantic search, right? Later on all these vectors can will be also stored in some kind of vector database. So for this we are going to use this again. It is a hugging face and at the end of the day it takes up any word or sentence. It will try to convert that into a 384 dimensional dense vector space. Uh, and then it can be used for task like this. Right. So we are going to use this. Right. And this is some of the example that you can see. Let's say if I have this specific sentence you can go ahead and execute it. Now I will show you step by step. So the model name I will just go ahead and copy it over here. Sentence transformer slash all mini lm v6 two. Right now if I just go ahead and execute this and show, let's let's see what embeddings will be able to give you. Now it is going to go ahead and initialize this for the first time. It is going to take some amount of time. But after this whenever you try to execute this it will be very much faster. Okay, so here you can see I progress not found.

Please update. Okay this is fine. This is a kind of warnings. And now, uh, just in some time. In some seconds this should get executed. Um, with respect to this. Okay. So, uh, this is how you go ahead and import this now, whatever models that is available in the hugging face with respect to embeddings, you can just go ahead and write the model name over here and automatically that embedding model will be loaded. Okay.

— Creating Your First Embedding —

Now till this is getting executed, let me go ahead and start writing my next text. Because at the end of the day, we want to see a example of how to give a text and get an embedding vector of that right or vector representation of that. So here I'm going to go ahead and create your first embeddings okay. And let's say that I'm going to give the text which looks something like this. Hello I am learning about embedding. Now understand one thing guys okay so here we are saying this is a part of a sentence Transformers. Right. So what this model does is that it does not just take a single word word and convert that into a vector. It takes the entire sentence and convert it into a vector. Okay, this is really important. And since I'm using this particular model it will convert into a 384 dimensions. Okay, just to explain again here, we are not going to convert a word into a vector since we are using something called as a sentence transformer. So this is nothing, but we are using sentence transformer and this specific model that we have used. Right. It is going to convert a sentence into vector representation. Vector representation. Yeah. There are different models which will be able to convert a word into a vector representation. But in this particular scenario we are going to take this sentence because most of our data will be in the form of sentences chunks. And. All right. So here you can see this has got executed. Now let's say this is my text. I'm learning about embeddings okay embeddings. And I want to go ahead and see that if I give this text to my embedding model, what is the output that I'm actually going to get. So I'm going to go ahead and execute embeddings Dot embed query. See, whenever we are giving a single sentence at that point of time, we use this function which is called as embed query. If you are giving a list of sentences, then you can go ahead and give this embed document function. Right. So here we are going to go ahead and give my embed query. And here we are going to go ahead and give the text. Finally I will be able to get my embedding embedding that is my vector embeddings. Now I will go ahead and write print. Let's say initially this is what my text look like right. So text colon. Let's say this is what's my text. Right. Now we are going to go ahead and print two things. Okay. First let's say we are going to print the embedding length. Okay. So with respect to the embedding length we will be using length function. And we will give this embedding over here. Okay. So this will basically give me what is the length of the vector representation for this particular sentence. And you know the answer. It is 384 right. We saw in the documentation. Let's say I want to go ahead and print my entire embedding. Right. So what

I will do, first of all, I will just let's say I want to go ahead and print it okay. So. So now I will go ahead and print my embedding. So let's go ahead and print this. Okay. Now let's execute this okay. So here you can see this is my text. Hello I am learning about embeddings. You can see this embedding length is 384. And this is how my embedding looks like right. So all these values and if you just go ahead and see this length it is nothing but 384. So that basically means this sentence has got converted in this to this particular vector representation. Right. Not a word, but the entire sentence has converted. So congratulations. This was quite amazing because at the end of the day here you are able to probably go ahead and use hugging face model, specifically embedding model. The embedding model name was all mini LM, L6, v2 and you are able to convert this into a vectors.

— Embedding Multiple Sentences —
Okay. Now similarly there you can also even try different different models. It is up to you. Okay, no one is stopping you that you only have to use this. You can use any of the models that is specifically available, right. But we will also go ahead and see some differences between this particular model. Like what are the basic difference? What are different kind of hugging face models are actually there, you know, so that things we will try to see it as we go ahead. And the next thing is that, you know, you can also give multiple text and probably convert that into a sentence. So let's say that I have I have some sentences okay. So let's say I have a sentence like this. The cat sat on the mat, a feline rested on the rug. The dog played in the yard. I love programming in Python. Python is my favorite programming language. Okay, now what I'm actually going to do. I will just go ahead and paste this same thing. Instead of using embed query, I'm going to go ahead and write embed documents. And instead of giving text I'm going to go ahead and give my sentences. Okay. So here I will go ahead and write embedding sentences. So it should be a list of vectors right. It should be a list of vectors. So now if I just go ahead and print this. Now here you should be able to get an output. See. So list of list right. So let's say if this is my first sentence what is my first sentence. This is how the vector looks like. If I were to go ahead and print my embedding of the second sentence that I've given in that list, I will just go ahead and print like this. Right? So similarly, you can go ahead and print the other sentences. That basically means a cat sat on the mat is nothing, but it is represented over here. If you want to see that, whether I'm going to get the same thing or not. Right. Let's see. Okay. So I will just copy the same line. I'll paste it over here to just show you. If I give the same sentence I'm going to get the same vector. Also see over here. Same vector. Same vector. Right. So now I hope you have understood what is the differences between embed query and embed documents. Right.

— Quick Model Guide (HF) —
So this was with respect to the hugging face where we have basically used this. What are

the advantages. Definitely this is open. Like you can use it without any keys okay. And there are also different different models which I would definitely like to talk about it, you know, and I will probably give the comparison also for you okay. So let me paste some of the information over here. So see I've given all the information when you should use it. How you should use it in this entire code okay. So let's let's execute this okay. So here you can see if you are planning to use all mini lm, l6 v2. The embedding size is 384 dimension when you should use it. It should. Description is fast and efficient. Good quality use cases, general purpose and real time application. There is also one more example all MP. Net base v2. So here the embedding size is 768 dimension description best quality but it is slower than mini LM. Right. Mini LM which is their use case when quality matters more than speed. So if you have that kind of scenario where quality is important, then speed, then you can go ahead and use this. And as you know you are going to get more dimensions with respect to the vector representation. Then there is also like all mini L to L v2. So here also you are getting 384 dimensions. Description. You can see what it does slightly better than L6 bit slower okay. So if you compare it to L6 it is bit slower. Good balance of speed and quality right. So what you can actually do if you want to use this, just go ahead and change the model name. And automatically you will be able to Change the model name in hugging face embeddings automatically. You'll be able to do this right. Then you have the sentence transformer. This model also you can use it usually for Q&A systems. Semantic search. We basically use this optimized for question answering. Then you have one more model which is called as paraphrase multilingual mini lm l2 v2. Again 384 dimensions supports 50 plus languages. Let's say if you have a scenario where you have multiple languages, let's say if the text is in multiple languages, you can use this. And again multilingual application. So you can use any of them. Okay. Try to probably make a list of statement sentences with different different languages. Since it supports 50 plus languages. I think this will also be able to perform the embeddings.

— Wrap-up —
So this was about hugging face embeddings. I hope you like this particular video. Um, this was it. In the next video, we will also be talking about OpenAI and we will be comparing again different embedding models even in OpenAI and all. Okay. So yeah this was it. Thank you. Take care. Have a great day.

Video 4: Getting Started With OPENAI Embeddings
Video Order: 4/5
Topics: OpenAI embedding models, API key setup, pricing & quotas, model selection, dimensions, single vs. batch embeddings, environment variables, LangChain integration
Difficulty: Intermediate

Introduction

Hello guys.

So we are going to continue the discussion with respect to embeddings. Already in our previous video we created our first embedding using Hugging Face (the **all-MiniLM-L6-v2** sentence transformer). We walked through multiple examples and compared a few Hugging Face embedding models like **all-mpnet-base-v2** and others.

Now it's time to show you how to perform embeddings with OpenAI's models—which are strong, accurate, and production-friendly.

Exploring OpenAI Models

- I'll first search for **OpenAI API key** and open the OpenAI platform.
- On the platform, you can find **Models**: GPT-4 families (text/audio/real-time), image generation (GPT-image-1, DALL·E 3/2), TTS, STT, and **Embeddings**.
- Each model family serves a specific purpose. For RAG, we'll focus on **Embedding models** (convert text → vectors), e.g.:
    - **text-embedding-3-small**
    - **text-embedding-3-large**
    - (Legacy) **text-embedding-ada-002**

When you click into a model you'll see performance, speed, and **pricing** (typically listed per **1M tokens**). You can also compare **3-large** vs **3-small** (quality vs. cost). We'll use the embedding endpoint from code shortly.

Creating Your OpenAI API Key

1. Go to **Create a new secret key** on the OpenAI dashboard.

2. Give it a name and generate—your key starts with sk-….
3. You need **credits** in your account for usage. Under **Settings → Billing**, ensure you have positive balance (e.g., a few dollars is plenty to practice). In my account example, I show a balance of **$0.62**—I'll top up when it hits zero. If you have zero, add at least **$5**.

Picking an Embedding Model

For embeddings we'll choose one of:

- **text-embedding-3-small** — balanced quality/cost, fast.
- **text-embedding-3-large** — highest quality, more expensive.
- **text-embedding-ada-002** — previous-gen (legacy). Generally avoid for new builds unless you must match older systems.

We'll start with **3-small** for demos.

Project Setup & Environment Variables

We'll work inside a notebook/file (e.g., openai_embeddings.ipynb).

Steps:

1. Create a .env file and add:
    - OPENAI_API_KEY="sk-..."
2. Load environment variables in Python using python-dotenv:
    - from dotenv import load_dotenv
    - load_dotenv() → returns True if loaded
3. Verify key with os.getenv("OPENAI_API_KEY") (don't print secrets in real projects).
4. Set process env to ensure libraries see it:
    - os.environ["OPENAI_API_KEY"] = os.getenv("OPENAI_API_KEY")

If you miss setting the env var, you'll see errors like "API client option must be set…". Once set, you're good.

Initializing OpenAI Embeddings in LangChain

- Import: from langchain_openai import OpenAIEmbeddings
- Init the model:
    - embeddings = OpenAIEmbeddings(model="text-embedding-3-small")

If the key is loaded properly, the instance initializes without error and you can call embedding methods.

Single-Text Embedding (embed_query)

We'll embed a single sentence:

Text

"LangChain and RAG are amazing framework and projects to work on."

Call

- vector = embeddings.embed_query(text)

Result

- **Dimension**: 1536 for text-embedding-3-small.
- The returned vector is a list of 1536 floats. For display, I print the first 5 values and the total length.

Output pattern

- *Original Input*: (the sentence above)
- *Embedding Size*: 1536
- *Sample Embedding Head*: [... first 5 floats ...]

Batch Embeddings (embed_documents)

We can embed multiple texts at once:

Texts

1. "Python is a programming language."

2. "LangChain is a framework for LLM applications."
3. "Embeddings convert text to numbers."
4. "Vectors can be compared for similarity."

Call

- vectors = embeddings.embed_documents(texts)

Result

- **Number of texts**: 4
- **Number of embeddings**: 4
- **Each embedding size**: 1536
- You can inspect any item in vectors[i] and confirm the dimension.

Model Cheat-Sheet & When To Use What

text-embedding-3-small

- **Dimension**: 1536
- **Cost**: ~$0.02 per 1M tokens (approx; check dashboard)
- **Use**: General purpose, cost-effective, good performance

text-embedding-3-large

- **Dimension**: 3072
- **Cost**: higher (~$0.13 per 1M tokens; check dashboard)
- **Use**: Highest quality; choose when accuracy is critical

text-embedding-ada-002 (legacy)

- **Use**: Support older/legacy setups; not recommended for new builds

---

Wrap-Up

That's all about getting started with **OpenAI Embeddings**:

- Create & load your **API key**

- Initialize **OpenAIEmbeddings** with your chosen model
- Use embed_query for single inputs and embed_documents for batches
- Understand model trade-offs (dimension, quality, cost)

In the next steps of the course, we'll compare these embeddings, then store them in a **vector database**, and use them in the RAG retrieval pipeline.

Video 4: Semantic & Similarity Search Using OpenAI Embedding Models — Study Guide
Video Order: 5/5
Topics: OpenAI embedding models, Semantic & Similarity Search,

Difficulty: Intermediate

Overview

Turn raw sentences into embeddings with OpenAI Embeddings and use cosine similarity to (1) compare pairs of sentences and (2) run a simple semantic search that returns the most relevant lines.

What you'll learn

- Compute embeddings with text-embedding-3-small using LangChain's OpenAIEmbeddings.
- Implement cosine similarity from scratch.
- Compare every pair of sentences and read the similarity scores.
- Build a lightweight semantic search: embed a query, rank documents by similarity, return the top-K.

Key concepts

- **Embedding**: numeric vector representation of text (here, 1,536 dimensions).
- **Cosine similarity**: dot(a, b) divided by (‖a‖ × ‖b‖). Values near 1 → very similar; near 0 → unrelated; near −1 → opposite.
- **Semantic search**: retrieve items by meaning (vector proximity), not exact keywords.

Quick setup (LangChain + OpenAI)

```
from langchain_openai import OpenAIEmbeddings

embeddings = OpenAIEmbeddings(model="text-embedding-3-small")

# single sentence

v = embeddings.embed_query("Your sentence here")  # -> list[float] length 1536

# batch

V = embeddings.embed_documents(["s1", "s2", "s3"])  # -> list[list[float]]
```

Cosine similarity helper

```
import numpy as np


def cosine_similarity(a, b):
    a, b = np.array(a), np.array(b)
    num = np.dot(a, b)
    denom = np.linalg.norm(a) * np.linalg.norm(b)
    return num / denom
```

Pairwise similarity (example)

```
sentences = [
    "The cat sat on the mat.",
    "A feline rested on the rug.",
    "The dog played in the yard.",
    "I love programming in Python.",
    "Python is my favorite programming language.",
]
embs = embeddings.embed_documents(sentences)
for i in range(len(sentences)):
    for j in range(i+1, len(sentences)):
        sim = cosine_similarity(embs[i], embs[j])
        print(f"{sentences[i]} vs {sentences[j]} -> {sim:.3f}")
```

Expected pattern:

- Cat ↔ Feline ≈ ~0.65 (similar topic).
- Python-love ↔ Python-favorite ≈ ~0.70+ (very similar).
- Cat ↔ Python sentences ≈ low similarity.

Minimal semantic search

```
def semantic_search(query, documents, embed_model, top_k=3):
    qv = embed_model.embed_query(query)
    Dv = embed_model.embed_documents(documents)
    scored = [
        (cosine_similarity(qv, dv), doc) for doc, dv in zip(documents, Dv)
    ]
    scored.sort(reverse=True, key=lambda x: x[0])
    return scored[:top_k]


docs = [
    "LangChain is a framework for developing applications powered by language models.",
    "Python is a high-level programming language.",
    "Machine learning is a subset of artificial intelligence.",
    "Embeddings convert text into numerical vectors.",
    "The weather today is sunny and warm.",
]
print(semantic_search("What is LangChain?", docs, embeddings))
print(semantic_search("What are embeddings?", docs, embeddings))
```

Expected top hits:

- Query “What is LangChain?” → the LangChain sentence first.
- Query “What are embeddings?” → the embeddings sentence first.

Notes & pitfalls

- Always normalize by vector norms when computing cosine similarity.
- Embedding sizes differ across models (OpenAI small = 1536; large = 3072). Don’t mix models in one index.
- For large corpora, use a vector DB (e.g., FAISS, Pinecone, Chroma) instead of brute-force Python loops.

Sec 5 · Video 5 — Semantic & Similarity Search Using OpenAI Embedding Models

Overview

Short demo showing how to compute cosine similarity scores between sentence embeddings produced by **OpenAI's text-embedding-3-small** and how to do a simple **semantic search** over a list of documents.

What you'll see

- Problem statement & example sentences
- Cosine similarity helper (NumPy)
- Creating embeddings with OpenAIEmbeddings
- Pairwise similarity comparisons (all-vs-all)
- Building a minimal semantic search: query → top-K matches
- Example outputs & interpretation (what high/low scores mean)

Key snippets (for orientation only)

- cosine_similarity(a, b) = (a · b) / (||a|| * ||b||)
- embeddings = OpenAIEmbeddings(model="text-embedding-3-small")
- embeddings.embed_documents(list_of_texts) / embeddings.embed_query(query)

Notes

- Cosine ≈ **1.0** → very similar; ≈ **0** → unrelated; **< 0** → opposite.
- Semantic search = rank documents by similarity to the **query embedding**.

Full transcript below

Sematic And Similarity Search Using Open AI Embedding Models:
Hello guys.

So we are going to continue our discussion with respect to vector embeddings.

Already in our previous video, we have seen that how we can use various embedding models and convert

your text into vectors.

Along with that, we also saw the comparison between three different embedding models that is provided

by OpenAI.

Now in this particular video, we are going to perform some cosine similarity with OpenAI embeddings.

Now what is the problem statement that we are trying to solve over here?

The problem statement is very simple here.

Let's consider that I have some sentences okay.

So these are my sentences that the cat sat on the mat, a feline rested on the rug a the dog played

in the yard.

I love programming in Python.

Python is my favorite programming language, so these are my sentences that are available now.

What I will do is that I will try to use OpenAI embeddings.

Along with that, I will try to go ahead and compare what is the similarity score, cosine similarity

score between this line to this line and this sentence to this sentence.

So all the comparison will be done 1 to 1.

Okay.

So this is the problem statement that I'm trying to solve over here.

We are just trying to see that one sentence comparison to the other sentence.

If we try to see with respect to vector representation how similar that is.

So that is the problem statement that we are trying to solve okay.

Now if you remember in our previous video we have already created a function which is called as cosine

similarity.

Right.

So this is my cosine similarity here.

Uh it is basically telling me to import numpy.

So let's go ahead and import numpy as np okay.

So here in the cosine similarity the function is very simple.

We're just going and applying the cosine similarity score formula.

That is the dot product divided by the magnitude of a multiplied by magnitude of p right.

And once we do this we will be able to get the cosine similarity itself.

Now let's go ahead and apply all this particular sentence that we have.

We will go ahead and apply the cosine similarity between each other.

Okay.

So quickly I will go ahead and write from line chain underscore OpenAI.

So first of all I have to go ahead and import OpenAI embeddings.

So I have already imported it.

But again in front of you I'm actually trying to do it again.

So embeddings is equal to we will go ahead and use the embeddings.

Oh sorry.

It should be OpenAI embeddings OpenAI embeddings.

And we will load the model.

Which model we will use the smaller one because it is quite cheap.

Right.

So the model that we're going to use is text embedding three small.

Okay.

And this is how my embeddings looks like okay.

Now what I am planning is that I will consider all the sentences and I'll try to find out the similar

sentence, or I'll try to compare the similarity score between each other.

Right.

This is all sentences.

Now what I'm actually going to do I will go ahead and first of all create my sentence Dense underscore

embeddings okay is equal to and I'm going to use embeddings inside this embed underscore documents.

We are going to give the value over here as sentences.

So once I go ahead and see this this will basically give me my entire sentence embeddings right.

So if you see over here we'll be getting a list of embeddings.

Right.

How many different sentences are there based on that we will be getting this.

Now my main over here aim is to calculate the calculate the similarity between all pairs okay.

So I'm going to go ahead and calculate this between all pairs.

So what I will do I will go ahead and run one loop for I in range off length of sentences.

Right.

So I'm going to go ahead and run this particular loop with respect to all the sentences.

And then I will take another variable for j in range of I plus one.

That is, I'm going to the next sentence and then I will be taking this tilde length of the sentences.

So I'm comparing one sentences to all the other sentences.

Yes.

And then we will go ahead and compute the similarity score.

So here I'm going to go ahead and write.

Similarity is equal to cosine similarity.

Cosine similarity.

And we will get the sentence embeddings of I.

That is the first sentence and j sentence is the nothing but the other sentences based on the loop.

Right.

So we here we are going to get the similarity.

Now finally we're going to go ahead and print f is equal to r with respect to the printing this I will

just go ahead and use the simple print f statement okay.

And I'll use some styling over here.

So let's print it like this sentence of I versus sentence of Jay and we will just get the similarity

score.

Okay.

So once I go ahead and execute this here, you can actually see all the information.

The cat sat on the mat versus the feline rested on the rug.

The similarity score is 0.65.

The cat sat on the mat.

The dog played in the yard.

Here you can see the similarity score is 0.32 for the cat sat on the mat versus I love programming in

Python.

The similarity is 0.089.

The cat sat on the mat.

Python is my favorite programming language, so here you can see the similarity will be very very less

because nothing is matching here.

Also it is nothing is matching.

So I'm getting 0.089 right.

So with respect to this you can see all the values are less.

But here you can see right I love programming in Python.

Python is my favorite programming language.

So here the similarity search score is also very very high.

That is near to 1.708.708 is near to one.

So this two sentences are almost similar.

So this is how we are specifically getting the similarity score.

Okay, now this is one example here with the help of similarity score.

What you can also do is that you can give or you can retrieve a specific similar results.

Right.

So for that, what I will show is that my second example will be on.

So if I go ahead and take this particular example it will be on semantic search okay.

So what does semantic search mean.

See my main aim is basically that whichever has the highest search try to give those similar kind of

sentence.

So what I will have let's say that I have a list of sentences okay.

And I go ahead and take one sentence and probably do a similarity search from those list of sentences.

I should be getting out the similar, similar sentence itself.

So now if I go ahead and do the semantic search in the semantic search, uh, first of all, let's say

that I have some documents.

Okay.

So this is my documents.

My documents has like this.

Right.

So here I'm having like chain is a framework for developing application powered by language.

Language models.

Python is a high level programming language.

Machine learning is a subset of artificial intelligence.

Embeddings convert text into numerical vectors.

The weather today is sunny and warm.

Now what is my task?

Let's say if I just go ahead and take a query like what is lang?

What is lang?

Now what is lang?

When I give this specific query from this documents, which is the similar sentence that should be retrieved,

right?

So with some specific score, just by directly seeing this, you know that this will be the first sentence

that will be similar to each other, right?

Similar to this sentence itself.

Similar to this query.

So this vector will match to this particular vector, you know, the vector representation.

And based on that the cosine similarity will be also high.

So we can retrieve this particular result directly from here.

So semantic search basically means we are going to find or retrieve retrieve the similar sentence.

That is what we are going to do over here.

Okay.

So let's go ahead and execute this.

This is my document.

This is my query.

Now I will go ahead and create a function which is called as semantic underscore search.

And with respect to semantic underscore search.

Here my first parameter that I'm actually going to take is query.

So this will basically be my query.

The second parameter will be nothing but documents.

The third parameter is nothing but my embedding model.

Let's say here I'm going to go ahead and give my embedding model.

And third I want to get how many similar sentence I have the top most three similar sentences.

By default I want to give it as three.

If you want four, you can go ahead and give it four also.

Okay.

So here this particular function is doing what.

So here I will go ahead and write the sentence simple semantic search implementation.

Right.

So this is what we are basically going to do.

And here we are going to write embed query and document.

So what is first of all what we are going to do as soon as we call this particular function.

Right.

So first of all we will go ahead and create a embedding only for the query.

So here I will go ahead and write embedding underscore models dot query or sorry dot embed query embed

underscore query.

So this will be the function that will convert the query into vectors okay.

Similarly for documents if I go ahead and use documents for that we will be using embed documents.

Right.

So here you'll be able to see that my embedding models dot embed documents.

We are going to basically do it.

And this documents will get converted into document embedding right.

Now inside this function we will also go ahead and calculate the similarity score.

Okay.

So first of all I will go ahead and make a list.

So let's say similarities is equal to this will be an empty list.

So I will iterate from I comma document embedding okay in enumerate.

So I'm going to go ahead and use an enumerate.

And I will go back to all the embeddings okay.

So step by step we will go right.

So inside this document embeddings it will be a list of embeddings.

Right.

So inside this embeddings when I'm going here I'm going to just go ahead and calculate my similarity

by using the same function that is called as cosine similarity.

Cosine similarity.

And inside this I'm going to first of all give my query embedding.

So this with respect to this particular query.

This is my embedding that I'm actually going to get.

And we are going to compare with the document embeddings right which we are iterating from here.

So every document embeddings we are going to compare.

So if I get the cosine similarity like some similarity score I will be able to get it over here.

Right.

So here I will go ahead and write.

This is my similarity score.

And this similarity I'm going to just go ahead and append inside my list.

So similarity is dot append.

So all the score we have first of all appending it right.

So here we are going to append it right.

So how do we append it.

First of all we'll go ahead and write the similarity score whatever similarity score we are having.

And then we are also going to go ahead and provide our documents information like whichever is the document.

With respect to that, we are going to put it over here right now.

Since this is a list all the information will get once it is comparing with each and every sentences,

right?

Then we will go ahead and sort by similarity so that we get the top three records right.

So here we will go ahead and write similarities dot sort.

And here, while we are using the sort function we will give reverse is equal to true because whichever

is the highest that needs to be shown in the top right.

And then we are going to return similarities with respect to the top underscore key result.

Right.

So how many top underscore key result is nothing but three.

So I need to only get the top three results right.

So this is my function.

And once we call this particular function you will be able to see that what we are going to get okay.

So now I will just go ahead and display the results.

So results I will call semantic search.

And inside this first parameter is nothing but query.

Query over here is nothing but what is launching.

The second is nothing but documents.

So here I'm going to go ahead and give my documents.

The third parameter is nothing but embeddings right?

So once I give all this three parameter the fourth parameter, I do not have to give it because top

underscore k default value is three.

Now I'll go ahead and display my results.

So once I display my results here you can see when I am searching for this, what is lying in the first

sentence that I got right is lying in is a framework for developing application powered by language

models.

And here you can see the match is somewhere around 0.67.

The similarity score is 0.67.

Then next document that we got based on the similarity score is 0.103130.

Python is a high level programming language out of all these documents.

First is language, then then you have Python C.

If you want to display this in a much more better way, I will print it in this way so that you will

be able to see it much more clearly.

Okay, so here you can see .676 language is the framework for developing application powered by language

model.

Then the second record that I got is this embedding converts text into vectors.

Okay.

Now let's say I go ahead and change my sentence.

My query this time should be what is embeddings?

Okay, so let's say I will just go ahead and search for what is embeddings.

Okay.

Now you know that out of all the specific records, this should be the first thing that should get displayed

right?

So I will use the same query and I will try to call this result.

Now see this.

I will try to call this result.

Now you see the magic.

The first sentence that you are going to get.

Embedding converts text into numerical vectors.

This is the maximum matching that is 0.662.62.

Then machine learning is getting matched.

Then long chain is a framework matched right?

So this way you are specifically applying a semantic search.

So based on a query based on a list of documents, what is the top match from that particular documents

and how you are retrieving this specific results.

This will be very, very handy because this all concepts will be reutilized in vector store and vector

database.

If you want to understand the differences between vector storm vector database, I will cover that in

the future videos.

But just understand inside our vector database or vector store, this kind of search will specifically

happen in order to provide you the similar meaning sentence, right?

And this is how the similar search will basically happen, right?

So I hope you like this particular video.

Uh, herein we discussed about two amazing things.

Right.

What all things we basically did over here.

First of all, we tried to find out the similar sentences based on cosine similarity score.

We find out we compared each and everything and we got the answer.

We just calculated the similarity.

This last one was with respect to the semantic search.

Here we are trying to retrieve the similar sentences.

Right.

So both the topics we discussed in a much more depth, uh, which you can still more practice as you

like.

Yes.

This was it from my side.

I hope you liked this particular video.

I will see you in the next video.

Thank you.

Take care.