EPISODE 1236

[INTRODUCTION]

[0:00:00.3] JM: Using artificial intelligence and machine learning in a product or a database is traditionally difficult, because it involves a lot of manual setup, specialized training and a clear understanding of the various ML models and algorithms. You need to develop the right ML model for your data, train the model, evaluate it, optimize it, analyze it for outliers and anomalies, assemble confidence ranges for the predictions and feature importance and eventually, deploy it to make predictions. An emerging field in AI, called automated machine learning, or auto ML lowers these barriers to entry by using AI to automate much of this process.

One of the market leaders in auto ML is MindsDB. Their service lets business users and developers make predictions on top of the data at its source. Rather than make expensive copies of databases, MindsDB trains and deploys models right inside the database. The results of their ML models can be queried with standard SQL statements and integrated into other applications as easily as querying any other database. In this episode, we learn about the progress been made in auto ML to simplify incorporating machine learning throughout the organization.

[INTERVIEW]

[00:01:10] JM: Jorge, welcome to the show.

[00:01:12] JT: Hi, Jeffrey. Thank you for having me.

[00:01:15] JM: You work on MindsDB. What problem is MindsDB trying to solve?

[00:01:21] JT: We're trying to solve the problem of making it very easy for you to apply machine learning, if your data is in a database.

[00:01:30] JM: Give me a little bit more context. Why is that a problem that is worth solving?

[00:01:35] JT: Yeah. The problem that we see that people have when they have a prediction that they want to make, is that they tend to reinvent the wheel over and over. Maybe the best way to cover this as an example. Imagine that you have a database with inventory information. You want to forecast how your inventory looks in the next week, or the next day, next month. Traditionally, how you go about this problem is that you have data scientists, or a machine learning engineer, that goes into your database, extracts, say, inventory for your iPhone inventory, and then loads this into a data frame, Panda's data frame. They usually work on Python notebook.

They go and build a model, spend some time, weeks maybe, days if they're really good. Build a model. Then once they have this model, they even in one that was pretty good, they run into a wall when they want to move this model into production. What I mean by this is, usually in that same database, you not just have an iPhone. For many of the stores, you may have thousands of products, tens of thousands of products, or even, you may have tens of thousands of products, as well as many stores that you may have.

What that means is that you're going to have to train tens of thousands of models for solving that particular problem of predicting inventory, which is not viable. There's no way that you're going to train a model for each of the products that you have, in each of the stores that you have. Given that your data is already in the database, the workflow that MindsDB enabled for people is to say, “Well, actually, I want to predict this column from this table, or from this query and figure out the rest.”

What MindsDB does is that allows you to do two things. Tell the database itself with a simple command like, “I want to make this prediction.” You usually do this with an insert statement in the MindsDB world, and you do it straight in the database. Then, what MindsDB does behind the scenes is that it figures out what's the best way to train a model that is a time series model for this particular task. Then it will publish this model as a table that also lives in the database. Then you can query this predictor the same way that you query a table, so you can do okay, now from my select from my predictive model, where the product is MacBook and the date is tomorrow. It will tell you the inventory value from this.

Making this abstraction allows users to see machine learning models, not just the stables, but also to not have to deal with all of the complexities that usually, even if you have a machine learning engineer, they have to go to build a sophisticated model for making this type of inferences.

[00:04:22] JM: Tell me about some of the biggest time constraints that a developer deals with in traditionally constructing a machine learning model, things that they might encounter if they are not using a machine learning database.

[00:04:38] JT: The first thing that they encounter is a whole bunch of ETL'ing, or data extraction and transformation. They have to go into the database, pull this data and then massage this data, so that fits whatever their idea, or ideal input is for the model that they have in mind. Again, this ETL'ing, usually when you're doing in the Python world is at some time, is reinventing a lot of data transformations that are available to you at the database layer.

At some point later down the road, you will have to make so many transformations, that even though you're not thinking at the time of building the model data, you're going to have to now translate all of those transformations into something that you can scale when you want to move into production.

The second part is okay, the model training, which again, that's what they're experts at, and clearly, they will end up developing a model that may be a really good model, then you will have to build a whole bunch of infrastructure to consume this model in a production setting. Usually, what people end up doing is that they end up building a web service that then can expose this model that you can then go into some RESTful API to go and get predictions from.

This involves that you're going to now invent a whole bunch of ETL'ing to make assumptions into the model and to bring those predictions to where you need them. Say, for instance, you want to use this BI tool, like you want to visualize this in Tableau, or whatever, now you're going to have to go make extractions from the database, run this into the model, and then get the predictions, feed them back into some tables, then you can actually use them in Tableau. That is the set of wheel reinvention that we enable avoiding people, or enable people to avoid.

[00:06:25] JM: Give me a sense of the onboarding of MindsDB. What does a developer do on day one?

[00:06:30] JT: Yeah. It's very simple. You just have to install MindsDB server. We MindsDB for most operating systems out there, as well as most container infrastructures, like Docker. Once you install a MindsDB server, the next thing that you have to do is plug it into your database. The databases that we support are MySQL, MariaDB, PostgresSQL, ClickHouse, Microsoft SQL Server. We're now working on integrations for Redis, as well as MongoDB.

It's just as simple as just telling what the database is and the credentials. Once you have that, then you can either train models directly from the database, or through a graphical user interface that we provide for you. To summarize, it is just a three-step process. Install, connect, and then you can do the training and predictions straight from your database.

[00:07:25] JM: How does MindsDB know how to query and make predictions to each of these databases?

[00:07:34] JT: Yeah, that's the essence that we've understood recently, and is that really the experts for any type of predictive problem are the people that know the data. Again, we developed this with the people that know how to query a database in mind. Ideally, what do you have to know is from what query you would like to learn what to predict, and you just have to tell them MindsDB it is a single statement for you. Going, okay, insert into predictors, predicts this column from this actual select statements, and that's what it is. We delegate the –

[00:08:11] JM: Swing —

[00:08:12] JT: Yeah. Essentially, you have the ability to tell MindsDB from what are you want to learn and then, therefore, you're delegating this to the user. You're delegating the responsibility to the user, to understand from what transformation in your database you want to learn. It’s not that MindsDB will figure out what predictions to get. You have to tell it what you want to predict and from what query you will like to learn.

[00:08:41] JM: Can you give a typical use case for MindsDB? What is a typical application that a user would want to build?

[00:08:51] JT: Yeah. Predicting inventory, anomaly detection on streams of data, predicting it, pricing, clustering of information based on some rows that you may have in your database, predicting sentiment based on documents or Mongo. Those are some of these cases. Might be as an open source project. Therefore, we don't have full visibility of all the things that people do with MindsDB, but we are certainly a big open source community.

[00:09:26] JM: If you give that example of inventory prediction, can you just walk through step by step how MindsDB would help with inventory prediction?

[00:09:37] JT: Sure. Imagine you have a table that has your inventory in say, MySQL. In this table, you have a column that has the inventory at a given time. Each row is an update to inventory. Essentially, let's say that you have iPhones, you have MacBooks, you have roller skates, you have a whole bunch of products. As your inventory gets updated, you have a new row that says, “Okay, I'm updating the inventory on iPhones.” I added 10 on my current inventory, now is 20. That in itself is a common architecture for how a inventory store will look like. Again, this information may be in different tables, but at the end, you can join those tables to make it look like this.

Assuming that the column that you want to predict is inventory, what do you do is you tell MindsDB, “Okay, from this query, select from the table inventory. I want to predict the column inventory count.” Then MindsDB goes and figures out, “Well, actually, this is a problem that has a group by, which is by grouping by product ID and say, store ID, and has some time element to this, because you may have a column for time.” It understands that it's a time series problem.

What MindsDB, also those behind the scenes is it understands that you may want to take into consideration the different columns that you have in this database, in this specific table. It builds a way to generate embeddings from each of those columns. Say for instance, you may have a description of the products. It realizes that you're going to have to use some way to get an embedding from the text in that column. For the time series one, it may use an RNN to get the latest state of the RNN as the embedding, or the actual historical information of the inventory.

Like that, it does it for all the columns. Once it has way to generate embeddings from each of the columns, it finds a way to mix all of this information for the targets. That's what is known as machinery mixers. Then, it compounds all of that into what is called the MindsDB model. Now, this model then gets published to the database as an abstraction of a table. What that means is that the table really doesn't exist anywhere, but just being an external table to the database. Again, it leverages on the capacity of some of these databases to have external tables. What I mean by this is MySQL, PostgresSQL, all of those have a concept of bringing tables that don't exist in the database, and querying them. Then when you query, essentially, what the engine does is that it for was a query to the external database engine.

MindsDB behaves like a database server. When it publishes, the model is just telling, “Okay, MySQL Server, I'm going to have a table called inventory prediction. Whenever you get queries to that, just give me those queries.” Essentially, what you're going to have is that when those queries come in, MindsDB knows that whatever is on the worst statements is what inputs to the model. Whatever output you get, form that sets a table and then returns that information.

Now, for the client though, even though this is what happens behind the scenes, really what they see is a way to create a model by a simple statement. Okay, I want to predict, learn how to predict from this column from these select statements. They do that by inserting into a table called predictors. Then the other thing that they see now is another table that appears on the database call inventory prediction. Then they can query this table. Select from my inventory prediction table, where product is iPhone and the date equals tomorrow.

Then MindsDB does, that was mechanics that I describe to get the prediction, returns a prediction to you and you're ready to consume that prediction that way. Even though behind the scenes, MindsDB does some complicated stuff. For the user, they are just either inserting into a table to generate models, or querying that machine learning models to table to get predictions. You can do crazy things like, okay, let's say you want to predict all your inventory tomorrow, then you can join your table of inventory with the model, then MindsDB will just take each one of those as an individual prediction, return that as a table, and then you just get the predictions for all your inventory items for the day tomorrow. Does that make sense?

[00:14:10] JM: Yeah. Is this saving – This is typically saving time for an engineer who would typically be building the solution outside of the database layer? Or is it just providing a additive functionality that they simply wouldn't have otherwise?

[00:14:30] JT: Yeah. On the one hand, it saves a hell of a lot of time for someone that will have to do this manually. On the other hand, even if you know how to do machine learning models, say for the case of inventory, that example, it is a complicated thing to do machine learning models when you have high-cardinality problems, and especially time series ones. When I mean high-cardinality is again, in the case of inventory, you can see the information contained in the inventory table as many, many times there's problems. Again, as many times series as products, stores, pairs you have.

Usually, for a machine learning engineer, they will be thinking this off as a single time series problem, but they want to move this into production, really what they realize is that they have to train tens of thousands of models. That is viable, and many of those projects stay there. Even though there is an abstraction that makes it easy for them to get started, it is also solving problems that are complex, even for very experienced machine learning engineers. That is essentially the value that we provide. We provided a way for them to do it in a very fast way, even if they want to have it as a baseline for benchmarking if they are experiencing machine learning. Also, to solve problems that are hard.

Again, time series is one of them. Imagine that you also want to do this for a stream of data. Let's imagine that you have a stream of inventory and you want to get anomalies being flagged. Now, doing machine learning on streams of data involves that you're not only very good at doing machine learning, but now that you have to be really good at doing machine learning on time series data, but also in time series data with high degrees of cardinality, as well as low latency predictions. Then, that you have to plug this into some streaming architecture.

MindsDB abstracts that part as well, but by plugging into some of those streaming databases, such as Redis. We're doing also for Kafka. Essentially, as a user, you continue to work with the database that your company uses. It’s just that now, you have some extra superpowers, which are machine learning at the data layer.

[00:16:46] JM: Let's get into the architecture of MindsDB. Can you just tell me a little bit about how it fits together and programming language choices and just the overall architecture?

[00:16:58] JT: Yeah. MindsDB at the core uses PyTorch for its computational graph. That's the main architectural choice that we made. We picked PyTorch, because of its dynamic graph capabilities. Then on top of that, we build pretty much everything ourselves. Even though the computing graph is PyToch. On top of this, we had to develop a way to do the auto ML capabilities on top of PyTorch. We have that as a separate project called lightwood, and that anyone can use if they are just into building auto ML models on top of PyTorch.

Then we wrapped that capability of doing auto ML on a framework called MindsDB. MindsDB is a server that essentially, behaves like MySQL server. You can connect to MindsDB like you connect to any other MySQL server using the MySQL TCP/IP wire protocol. What this does is that you can, from any clients say, either a BI tool, or any clients for MySQL, you can just connect to MindsDB. The difference from a database is that instead of seeing tables, you actually see models. When you query these models, you would get the same behavior that you get when you will query a table.

What this does is that enables you to then plug MindsDB into databases that support federated storage, or external tables. That's essentially the main building blocks. To summarize it, we have an auto ML library that depends on PyTorch. On top of that auto ML library, we have server infrastructure that is essentially, providing the MySQL wire protocol capabilities for you to expose machine learning models, or stables. Then from there, everything is done by your database. Because then, we plug MindsDB by means of external table capabilities, then you can do this straight from the database.

We do have a graphical interface that ships in with the server. Essentially, this graphical interface allows you to do the same thing that you will do from database, but like in a drag and drop interface.

[00:19:10] JM: What have been the hardest engineering problems you've had to solve in building MindsDB?

[00:19:15] JT: Yeah. I think that that question changes every few months. I think that problems that seem very difficult at one point, now we see as trivial, even though they may be still difficult problems. The very first one that we have solved was the auto ML part. Auto ML can get very complicated if you don't define your constraints very well. The cool thing about doing at a database is that the constraints tend to be better defined.

For many databases, most of them are problems, they at the end get translated into, “Okay, let me understand what table you have, what different column data types you have.” The cool thing about the way that machine learning is developed in the world is that is more data type driven than problems driven. Most people are doing research on text and different tasks within text, regardless of the industry that that text may contain. Then, the same thing for dealing with numerical virals, or time series information.

We can leverage in that by just bringing the latest of understanding of dealing with different data types. What we really want to get there is an embedding. Then the way that you mix this can also be a science in itself. At the end, we realize that there are simple mechanics that you can assemble like Lego blocks. We ended up breaking that problem into different units of work that we can then assemble into a bigger model. That was the first challenge that we got to solve. That's actually what kicked off as our open source project that got a lot of attention and made us understand the second part of problems, which is okay, once you have a model, auto ML or not, how do you listen to production in a seamless way?

To understand that for many people, the data that they were training this models from came from databases. How do we expose this to databases became a second challenge. Luckily, we had other investors, the guys from MySQL and MariaDB, they have a fund called open ocean. We started to see the synergies between databases and machine learning and essentially, now we understand how to solve this at scale. It was a challenge at the time. Now, we know what the solution is.

I think that once we started doing this, we started to realize that there are problems that people have are very common on their databases, that can become very tricky, even if you know what you're doing in terms of machine learning. Is that I've been describing to you before of time series problems with high-cardinality. Their problems are very difficult to solve. That you cannot

solve with off-the-shelf libraries. Like say, for instance, profit, which is very good if you're only dealing with one single time series problem with only one single variable that you want to predict on one input.

That's not the case for many of these problems inside a database. Solving those did require quite a decent amount of engineering. Now we solved it and that is something that any of the users of MindsDB can leverage upon. NOP problems at scale in databases is another challenge that we're currently working on. Being able to do predictions with very low latency is another one. One thing is to train a model that can make a prediction that is accurate. The other one is to bring those very accurate models, so that they can give you predictions in the hundreds of milliseconds, or tens of milliseconds.

[00:22:48] JM: A lot of this work was done when you were at Berkeley, right?

[00:22:54] JM: The auto ML part, yes. That's when we started understanding that auto ML was going to be a problem that people could solve. We actually did. Actually, one of the interesting things that we also understood there is that being able to produce models automatically, also have to go hand in hand with being able to provide tools for the user, so that they can develop trust on the model.

Most people are not aware of explainability being an issue, until they have the model. That is essentially a byproduct of you having a successful auto ML, or successful machine learning engineer producing a model. Then you realized that you need to understand how to trust this model. That is a problem that we're still cracking, but we understand that trust can be developed over time, if you can, in the different stages of the model, be very transparent as to what is going on.

The approach that we took to explainability is more about quality over the different stages. You have the data quality. You want to inform the user about any potential quality issues that you see on the data, such as potential outliers, or even more importantly, are there biases on your data? We do that for the user.

The second part is when you have a model, you want to understand not only what makes this model tick and what are the things that are taking into account to make the predictions, but also, when can you trust a model? You shouldn't trust a model and be very transparent about this. Lastly, one of the things that we start to realize is that when people use a model, say for instance, when you're actually now making a prediction, say that going back to the inventory, you want to predict what the inventory is going to be tomorrow, conveying to users that this models, they are going to give you just an estimate of what that can be. You cannot take that answer to be deterministic.

Then, being able to portray the results that you get into a model into like, “Hey, look. What you're getting is a range of values and the probability of your answering landing there.” Being able to communicate this in a very simple way is what we've done. Also, to understand and make users understand that is not a single value what you get, but you get – that your value may fall within a certain range and there's a probability of it landing there.

[00:25:17] JM: You mentioned this issue of needing to know whether you can trust a model or not. Can you dive deeper into that?

[00:25:25] JT: Yeah, absolutely. I think that they're mission critical problems. That's up to the domain expertise that you have. It is not necessarily that someone is going to die or not, because of the prediction. It may be that you’re maybe losing money for, because of a prediction. At the end, it’s not that you're getting predictions out of the whim of wanting them, you are getting predictions because you're going to decide something based on that prediction. In the case of inventory, you're going to probably decide if you're going to buy more or less. Making sure that let's say for instance, if you're doing anomaly again, on inventory is so that you can act ahead of time, so that you don't get caught unprepared.

All of these predictions lead to some action. The last thing that you want to do is to act upon something that you don't trust. If you have a system that is breaking inventory and selling you, their inventory is going to drop. If you don't trust this, then you're never going to use these predictions actually. You're going to go for a less reliable system, perhaps, which is you your intuition. You'd be like, “Oh, I actually think it’s going to go up. I actually think it’s going to go down.”

The reason why you may prefer that is because you may have someone that is accountable for that decision. What we want to do is, rather than replacing the prediction, is augment the capability of that prediction, so that the decision-maker continues to be the decision-maker, but has some tools to understand whether to trust or not trust that prediction. In our opinion, they boil down to being able to communicate very clearly the quality of the different stages of machine learning, even if they're not machine learning engineers.

[00:27:09] JM: Do you have a selection of pre-trained models that you apply the data to?

[00:27:18] JT: We do transfer learning for some of the problems, where we understand that there is significant gain in time for the model training. The cool thing, though, is since we break the problem into smaller problems, then you can leverage on transfer learning for some cases. Say, for instance, you have, let's go back to the inventory problem, so we keep it within that scope. You may have a column that has the description of the products. Now, that column may give you or not some information about the inventory.

What you want to do is you, of course, want to extract some meaningful information to make this prediction. You can go two ways about this. The building block for us is to get an embedding from text. In this case, for that description column. You can transfer learn from models that have been really good at giving you an embedding from sentences. That's what we do. Now, we do this for most columns. Some of those columns, you can do transfer learning, some of those you can't. At the end, what you get is various different building blocks for each of the columns that you have in your problem.

Then from there on, each model is very different. Because unless you have two tables are identical, MindsDB will come up with a different way to mix this different embeddings to get to your predictive variable. Now, the mechanics for this, now we understand better how to do this. Initially, what MindsDB was to just bring whack-a-mole edit, it was trying very different models that will at the end, benchmark and will give you the best one. Now it has very serious access to okay, for this statistical information that I have from my input data, the best candidate models to mix it, embeddings to get to the target may be this one. Then again, tries them all and gives you the best one.

[00:29:19] JM: Are you tuning hyper parameters, like the number of layers and the layer sizes, etc.? Are you trying a variety of different models to present to the user?

[00:29:31] JT: Yeah, that's a great question. Initially, we were on the fly. Then, we realized that the time to walk through all of those hyper parameters was something that many users didn't have the patience for, or the computational resources. Now, we understand that also, there are places where hyper-tuning will give you a significant increase in accuracy. One order of magnitude difference in accuracy. There are some other hyper parameters where you're talking about your model being 98% accurate, versus 98.5% accurate. That difference is not a substantial to the user, so that they can now wait an extra five hours to go and fine tune those hyper parameters automatically.
We do fine tuning for those elements that we understand, have a significance output in terms of gains in accuracy. Then for the other ones, we have a set of rules that we understand will give you the optimal number. May not necessarily be the absolute best, but close to the best.

[00:30:47] JM: What do you do when the user tries to apply MindsDB to a use case that doesn't end up being a good fit? Is the database able to flag to the user that this data just can't really be trained around?

[00:31:05] JT: Yeah. That's the cool thing about the trust dimension that I was describing to you. You can train a model with MindsDB and then at the end, if your data is not enough, you're just going to get poor quality dimension that will be explained to you. Then you can decide if that's better than random guessing or not, then you're informed guests can be or not. We don't constrain users by not training the model. What we do is we train it, we try and then we give you some metrics around the quality of the model in itself. That will make you decide whether to use it or not.

[00:31:44] JM: Who is the user of MindsDB? Is it a data analyst?

[00:31:49] JT: Data analyst for sure. Account for them. Now we're making integrations to BI tools. Again, because since MindsDB behaves like a SQL, that MySQL database, and you can plug it to most BI tools out there. Not necessarily that we're making development for those

integrations, but we're building partnerships with database makers, with BI tool makers. We have one with Booker right now and developing one with Domo and various ones.

That's because we understand that, of course, the data analysis parts, many of those already understand and how to do SQL. Many of their BI tools are SQL-driven, so it works for them. It's also developers that touch the database and they need to implement a feature that has predictive capabilities. Developer in general accounts for a good number of the users as well. Not just data analysts.

[00:32:45] JM: The actual machine learning that's taking place in MindsDB, what machine learning framework are you using?

[00:32:54] JT: Yeah. MindsDB has its auto ML framework called lightwood. Then under the hood, lightwoord uses PyTorch.

[00:33:04] JM: Any color on PyTorch versus TensorFlow?

[00:33:08] JT: Yeah. I think that it definitely has color. It's a preference. We picked PyTorch, because they were pioneers on the dynamic graph. It just lends itself for the way that we were building auto ML. I think that now, TensorFlow has evolved in that direction as well, so everything that we do in PyTorch we could do in TensorFlow. We're certainly happy with the direction that we took at that time. We understand that there are things that one does better than the other.

I think that for the implementation that we had, PyTorch was a good solution at the time and continues to be. TensorFlow now is also a very good one, especially now with the abstractions that they made, so that people can also do them as if they were doing **[inaudible 00:33:57]**. That will certainly would have made it easier for us at the time. Yeah, we like them both, but we picked PyToch at that time, because it was the right solution for us. I really think that both of them are solid frameworks.

We do however, on top of our auto ML library, we have the MindsDB server, which is agnostic of what you used behind the scenes, so you can bring your own models, if you have installed the

libraries that you want to use in the server. Therefore, if your models are being developed on TensorFlow, then you may not have the auto ML capabilities, but then you have the ability to publish this models to databases and retrain them from the database and consume data from the database as well.

[00:34:41] JM: What are the biggest areas of improvement you'd like to add to MindsDB?

[00:34:46] JT: That's a million-dollar question. Everyone in MindsDB is obsessed with improving a given dimension of MindsDB. We certainly have various teams working on different elements. That target is always moving. Currently, I think that big efforts that we're doing are around NLP. Right now, MindsDB is good at it, but it can be stellar at NLP and that's one of the objectives that we have in the immediate term.

Just as a few months ago, our objective was to really hone down the very difficult time series problems, which we did work on streaming data; is certainly something that we're doing now. We understand that a lot of people have the need to get predictions from streaming data. It's just that it's very difficult. In many cases, it’s just theoretical what people have on doing machine learning on streams of data. Then providing those abstractions as we do it for other problems and stuff that what we want to do.

Essentially, with those two, likely the path is as soon as we get really good at NLP, as well as really good at streaming data, then we're going to see all the gaps that may be even more challenging problems. As of now, those are the two main areas of improvement.

[00:36:16] JM: Do you have any other ideas about how machine learning can be pushed into other areas of software development, or company building, such that the actual hard to implement parts of machine learning are abstracted away from the user?

[00:36:39] JT: Yeah. I think that eventually, the work that we are doing will be more natural, even at the database layer itself. We ourselves are working directly with some of the people that have been instrumental in the development of various database languages. Making some of these capabilities even more natural at the SQL state is something that we're doing right now. Similarly, we want to do it for non-SQL databases.

It all boils down into a pattern that we've seen before. It's like, say for instance, big data. A few years ago, you had to be a very skilled big data engineer, because you had to learn how to do parallel computing and a whole bunch of techniques to really do big data. They got abstracted and continued to be abstracted by solutions like Snowflake, now distributed data frames and the like. That you don't have to know any of that and you can still do big data, if you just know how to develop.

Now, similarly, we think that this is going to happen and it's happening in machine learning. Whether we understand exactly what those can be, it's up to the direction that users take implementation like MindsDB. What I'm trying to say here is MindsDB is already providing a great amount of abstraction. How these abstractions will evolve will really depend on how this users start applying those abstractions and what they start requesting from companies like MindsDB.
We are essentially trying to ensure that our development is driven by feedback from our community. That's probably the main reason why we weren't open source. By definition, I wouldn't be able to tell you what those abstractions would look like, but more into surely, users will have a lot of ideas in what they would like to do. Then, our task is to ensure that we can provide those abstractions for them.

[00:38:57] JM: If you are not building MindsDB today, what would you be working on?

[00:39:01] JT: That is a great question. I spend so much time thinking of MindsDB that right now, my obsession is really to make sure that MindsDB provides as much value as possible to its users. Before MindsDB really took off, as it's taking off right now, I felt that there were other important problems to solve, not necessarily on the horizontal approach that we've taken, but more in some domains.

I think that, for instance, whole genome sequencing data, the cost of it is dropping so much every year, that there's going to be large amounts of whole genome sequencing data available to people. There are very little solutions that can deal with that slice of datasets, like a whole genome sequence can be a few terabytes of data. Therefore, it's a challenging task to do machine learning on that in itself.

However, the applications are as far as your imagination can go, you can predict mental illnesses, you can predict a whole bunch of elements within your DNA that can lead you for better treatments. Again, that area is an open field right now, but the costs are what used to be the barrier of entry is dropping so much so, that the amount of data versus the capabilities to do machine learning on those continue to be a significant gap. Just to give you more into it, like a few years ago, it was on the millions of dollars for you to get a whole genome sequence. Now, it's in the hundreds, or less than a 100 for the general consumer. It's an incredible price drop. Therefore, it is bewildering to me that there are not many solutions out there for you to do machine learning at that scale.

[00:40:58] JM: Do you have any broader predictions about the ecosystem of machine learning and machine learning tools?

[00:41:07] JT: Yes, absolutely. I think that the approach that we've taken about machine learning at the data layer is going to be a trend. Now it’s starting to catch on. For instance, Neo4j, they started to realize that this is something that has to go this way. Neo4j is trying to support also native capabilities on their own at the data layer. For sure, this is one of the predictions that again, we are fully invested in, but we believe it to be true.

Then also, there will be a time, not far enough where quantum computing merges with machine learning. Even right now, there are very interesting companies, like **[inaudible 00:41:51 Company name]** and whatnot that are doing this machine learning with quantum computing for specific problems that are really hard to solve otherwise.

Now, the amount of qubits that these quantum computers are starting to be able to handle is growing every year. Now we see even more startups to jump into producing quantum computers. I do think that the merger of the two before was theoretical. Now, it will be a reality. That, you have also people like the guys at IBM producing high-level libraries for you to program quantum computers in a way that you don't have to go and do a PhD in quantum computing to go and do it.

Which is similar to what was happening before with programming in general. You have to learn how to do assembly code. Assembly code requires that you really understand how the computer works, and so you can move information around and make these computations in the way that the computer actually does it. Then you have high-level programming languages that abstracted this. Intel – sorry, IBM is going down the path of providing this tooling. Surely, that is going to be a game-changing industry, as well as not that far from what it was even last year. I do think that now, it's really going to happen very soon.

[00:43:16] JM: Well, that seems a good place to close off. Jorge, thanks for coming on the show. It's been a pleasure.

[00:43:21] JT: Likewise. Thank you so much, Jeff.

[END]