TigerGraph

Intech Solutions

In-Database Machine Learning for Big Data Entity Resolution

By Dr. Changran Liu, Solution Architect, TigerGraph

DATA SCIENCE MINI-GUIDE

TigerGraph

Intech Solutions

Machine Learning and Entity Resolution

The world is gathering data faster than it can make sense of it. According to market intelligence company IDC, the "Global Datasphere" in 2018 reached 18 Zettabytes. One zettabyte is approximately equal to one billion terabytes and more data is being generated at a faster pace every day.

The data being collected by organizations can, however, give a misleading or fragmented view of the real world. A person, for example, could appear multiple times (or have multiple digital entities) within the same database, due to typos, name changes, aggregation of different systems and so on. If we try to merge two databases, how do we match entities, when the ID systems might be different or contain errors?

Entity resolution helps organizations get to the truth. Entity resolution, which is the disambiguation of real-world entities in a database, is an essential data quality tool for businesses wanting to see the full customer journey, and to make the right recommendations, or for law enforcement agencies trying to uncover criminal activity and the perpetrators.

Entity resolution helps get to the truth. Entity resolution, which is the disambiguation of real-world entities in a database, is an essential data quality tool.

Graph databases can help you disambiguate

The key task of entity resolution is to find and make linkages between digital entities which refer to the same real-world entities. Graph is the most intuitive, and as we will also show later, the most efficient data structure used for connecting dots. Using graph, each digital entity or attribute can be represented as a vertex, and the entity-has-attribute relationships can be represented as edges. In such an entity-attribute graph, when multiple entities share the same attributes, they will be linked together by the graph (see Figure 1). By representing the entity-attribute relationship as a graph, a variety of graph algorithms (e.g. cosine similarity, Jaccard similarity, connected component, label propagation, Louvain) can be performed for the task of entity resolution.

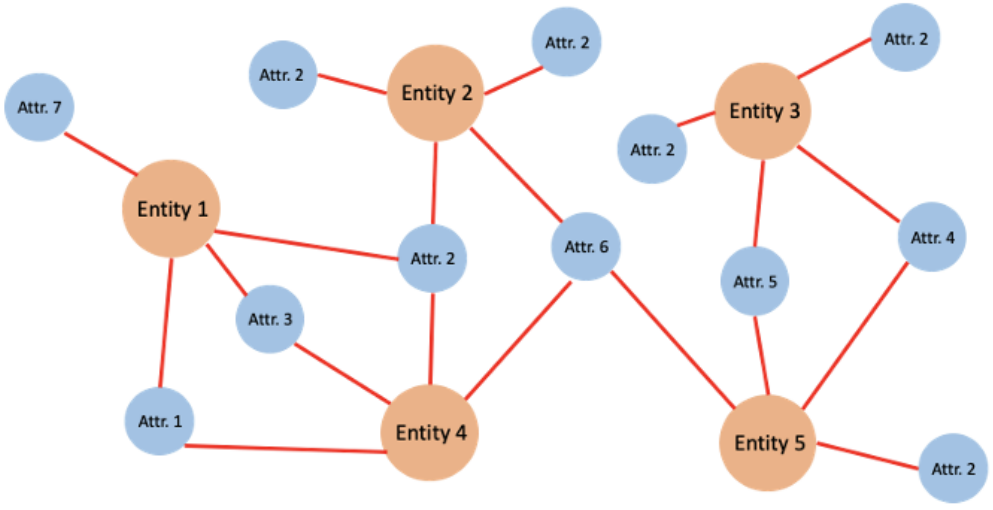
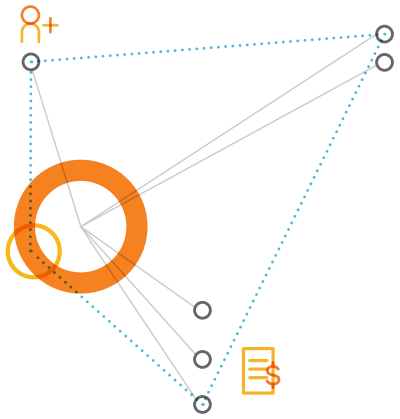


Figure 1: Each vertex in the graph represents either an entity or an attribute. The edges represent the entity-has-attribute relationship. The graph links different entities together when they share common attributes. For example, Entity 3 and Entity 5 are linked by Attribute 4 and Attribute 5.

Solving the entity resolution problem with graph can be categorized into two main steps, namely linking and grouping. In the linking stage, graph algorithms, such as cosine similarity, Jaccard similarity, can be applied for pairwise matching. Once two entities are matched, an edge can be drawn between them to represent the same-as relationship. The same-as relationship can be deterministic or probabilistic. In the case of probabilistic relationships, a weighted edge can be used to represent the likelihood of one entity being the same as the other entity. In the grouping stage, the inserted same-as edges can greatly accelerate the computation in which graph community detection algorithms, such as connected component, label propagation, and Louvain, can be used to group the the digital entities by their real-world entities.

Graph provides an efficient approach for the entity resolution problem. A native graph database with massive parallel computing capability is the best tool to implement the approach. Solving entity resolution problems by running queries in the database not only saves the time for exporting the data to the other computing platform, but also addresses the challenge of ever-growing data size and data update. Additionally, because of the transitivity of the same-as relationship (i.e. if entity A is the same as entity B, and entity B is the same as entity C, then entity A is also the same as entity C), entity resolution requires multiple-hop queries. Such queries require multiple join operations which are computationally expensive for relational database. A graph database, on the other hand, is the ideal solution for the big data entity resolution task. In the next section, we will show how to implement a TigerGraph solution for entity resolution.



Graph provides an efficient approach for the entity resolution problem. A native graph database with massive parallel computing capability is the best tool to implement the approach.

TigerGraph is especially adept at uncovering the truth

Let's examine a real-world example of how TigerGraph disambiguates the users of movie streaming websites through some login information. When people are logging into their account of a streaming website to watch a movie, the website may collect some login information such as the user account, user email used for account registration, IP address, physical address (device ID), and login time. A movie streaming company can have millions of user accounts and billions of login records. One person may register multiple accounts and one account can be logged in with different IP addresses, and devices. In this scenario, each user account is a digital entity with multiple attributes. The entity resolution task is to link the accounts to the real-world entity, a human user. Without loss of generality, in this simplified example, we assume each login record contains the user account, user email, IP address, device ID, and phone number (see Table 1). The five accounts in this toy example were actually registered by the same person. Now let's build a graph in TigerGraph to find this person.

user account	user email	IP address	device ID	phone
1	jojo@adv.com	18.207.48.62	2E-1D-ED-EC-D9-ED	181-481-4409
2	jj@adv.com	18.207.48.62	2E-1D-ED-EC-D9-ED	415-509-4062
3	jjoe@adv.com	18.207.48.62	2E-1D-ED-EC-D9-ED	505-316-1184
4	jjoe@adv.com	18.207.48.92	E3-5B-BC-03-0D-AC	312-921-5548
5	jj@adv.com	18.207.48.92	E3-5B-BC-03-0D-AC	312-921-5548

Table 1: Sample entities for Joseph Joestar

Defining the graph schema

We need six types of vertices in this graph. The first type of vertex is for the user accounts. Similarly, each of the four attributes, user email, phone, IP address and device ID, can also be represented by a type of vertex. The last type of vertex is the user. The grouping of the entities will be done by connecting all the matching account vertices to the same user vertices. For the edges, we have "has" edges between accounts and the attributes, "Has Account" edges between users and their accounts, and "same as" edge between users.

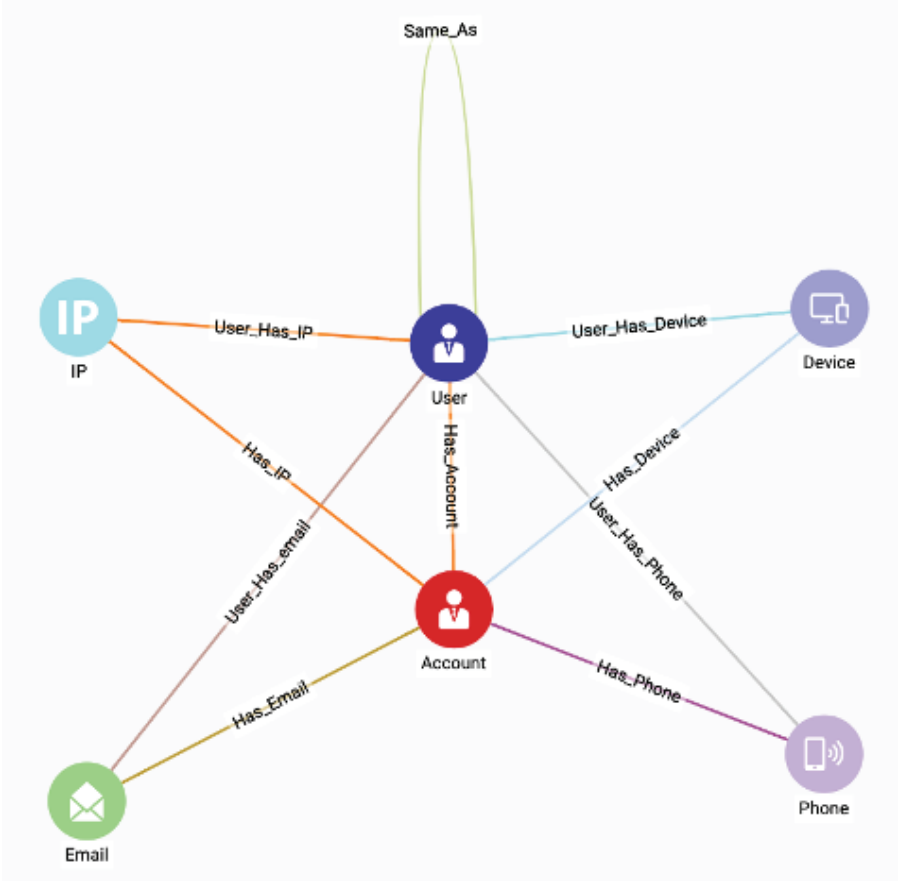


Figure 2: TigerGraph Cloud Schema for the [Starter Kit - In-Database Machine Learning for Big Data Entity Resolution]

Loading Data

With the graph schema above, the data can be loaded to create the entity-attribute graph. For each record, we will use the user account as well as the attributes to create vertices, and connect the account vertex with its attribute vertices. Figure 3 shows the graph created by the sample data in table 1.

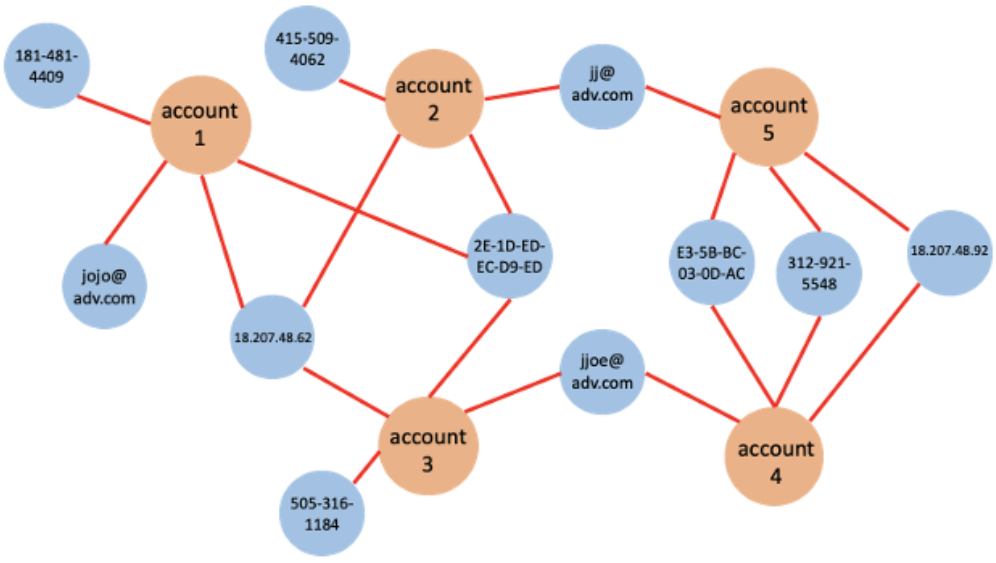


Figure 3. The graph created by loading the data in Table 1

Running queries

We implement entity resolution as three queries: Initialization, linking and grouping.

1) Initialization

The initialization query assumes each account is registered by a different user and let the user inherit all the attributes of his account. For simplicity purposes, figure 4 only shows the user-attribute relationships and does not show the account vertices in the graph.

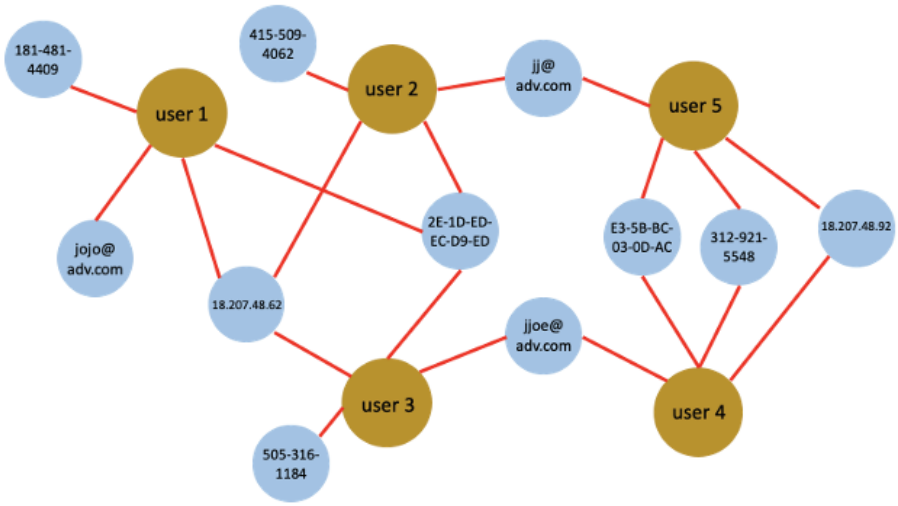


Figure 4: The initialization query creates users, one user per account

2) Linking

The linking query connects the entities that satisfy the matching condition. In this example, let's assume two accounts were registered by the same person when they share at least two attributes. For example, account 1 and 3 were logged in both using the same IP address and device, thus we consider they were registered by the same person (see figure 4). In this case, the linking query traverses all the user - attribute - user paths and counts how many attributes are shared for each pair of users. Lastly, the query will insert an edge between two user vertices if they are linked by at least two attributes.

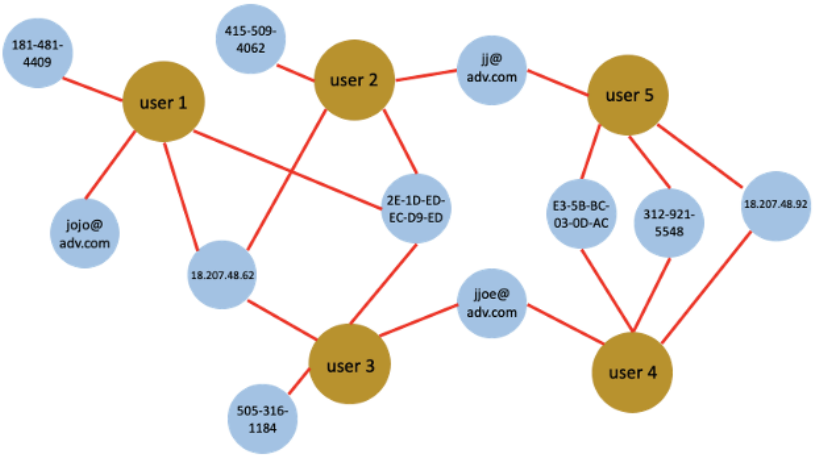


Figure 5 (a)

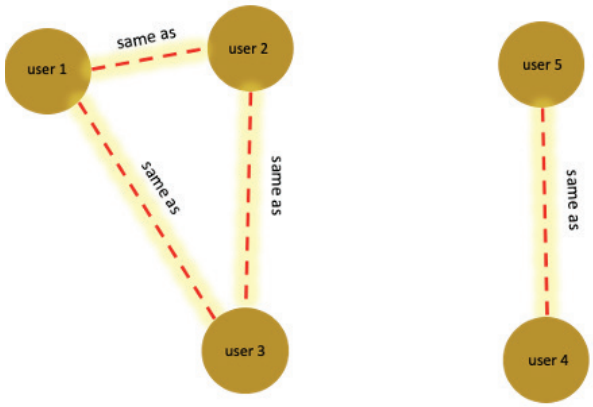


Figure 5 (b)

Figure 5: (a) In the above example, four same-as edges will be inserted between user (1, 2), (1, 3), (2, 3) and (4, 5) because each pair shares at least two common attributes. The connections between user 1 and 2 are highlighted. (b) The graph of user-user relationship after inserting the same-as edges in the linking query. For simplicity purposes, this figure does not show the account and attribute vertices in the graph.

3) Grouping

After the linking query finishes, all the matching users will be connected. The entity deduplication can be achieved by merging the connected user vertices together. This can be done by performing the connected component algorithm on the network formed by the user vertices and same-as edges. Briefly, the connected component algorithm will find the connected entities by assigning the same label to the entities in the same connected group.

The basic flow of the connected components algorithm is as follows:

1. Assign a unique integer as an ID to each entity.
2. Propagate this ID to its connected neighbors, so that each vertex will receive all its neighbors' IDs.
3. Update the old ID to be the smallest ID among its neighbors' and its own current ID.
4. Repeat above steps 2 and 3 until no ID update happens.

When iteration stops, the vertices in each connected component will have the same ID. And this ID will be the smallest ID within the connected component. The algorithm above not only finds the connected entities, but also selects a "lead" entity within each connected group. The last step of the grouping query is to let the lead entities inherit all the attributes in the group and to remove the other entities. The graph of user-attribute relationships after the grouping query finishes is shown in Figure 6.

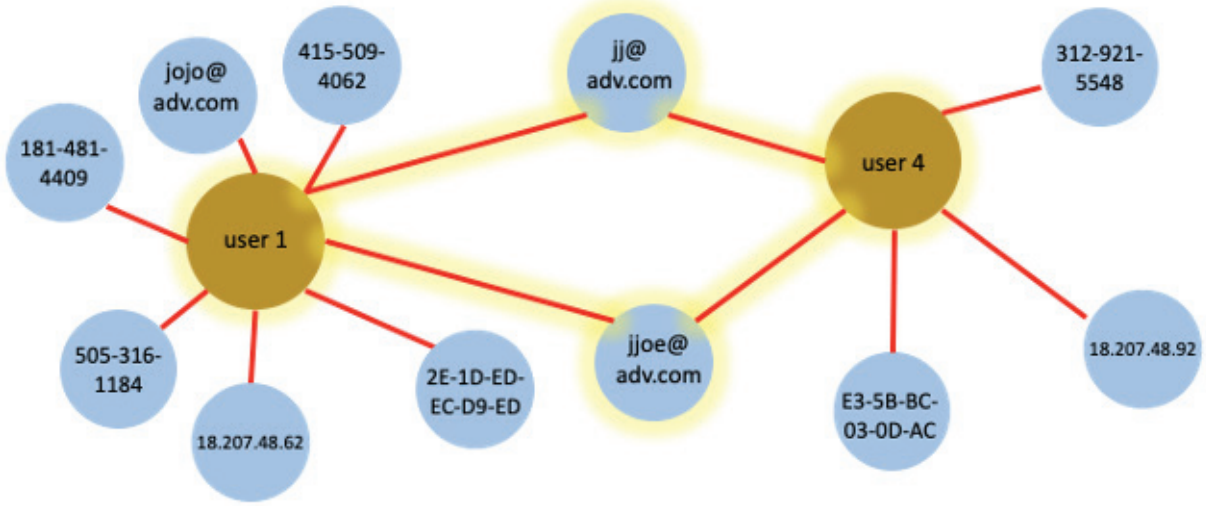


Figure 6: After the grouping query finishes, user 1, 2, 3 will be merged into user 1. user 4 and 5 will be merged into user 4.

In some use cases, multiple iterations are needed to repeat the linking - grouping steps. For example (see figure 6) after we merge users 1, 2, 3 into 1, user 1 will have all the attributes from 2, 3 and itself. So user 1 can connect to other entities that it did not in the last iteration, thus new groups can be formed. Therefore, the linking and grouping shall be repeated until no more same-as edge needs to be inserted.

Key benefits of the solution

Data lineage

One key feature of the above solution is that all the linking and grouping are operated on the user vertices (i.e. only the user vertices and the edges connected to them will be inserted or deleted). By constraining the data insertion/deletion within the user subgraph, the original account record (i.e. the account vertices and their relationships to the attributes) will be left untouched, thus the data lineage is well preserved.

Parallel computation

It is also worth mentioning that in all above steps the computation for each vertex or edge can be run in parallel. For example, in the second step of the connected component algorithm, one thread can handle one edge computation where the ID of the source vertex is aggregated on the target vertex. Similarly, in the third step, each vertex can have one thread to update its old ID. Such a computation model can naturally fit into a map-reduce framework.

Incremental update

In real practice, the batch process on all the data does not need to be performed whenever new records are loaded into the database. The above solution allows incremental updates where only new accounts (or old accounts with updates) and the minimum amount of relevant old accounts need to be processed. When having new data or streaming data loaded, the pairwise matching only needs to be performed on the newly loaded account vertices and the old account vertices that share the same attributes with the newly loaded vertices. Such an optimization significantly reduces the computational workload.

Generic description and solution for the problem

Clustering is a classic unsupervised machine learning problem. The linking and grouping method described above converts the entity resolution problem to a graph clustering problem. Specifically, in the linking stage, an entity network is created. Then in the grouping stage, different graph clustering algorithms can be used to group the connected entities together. The connected component algorithm is a good choice when links are deterministic and certain, but other graph clustering algorithms such as label propagation and Louvain may be more suitable if the linkages are weighted according to confidence. Since graph is an abstraction of relationships, the graph solution in the above example can be easily adapted for other use cases. Here we will list four more examples.

1) Flight ticket booking

An airline company can have millions of ticket booking records in the database. Each booking record can include the ticket number, passenger first/last name, email address, phone number, departure, destination as well as other related information. One passenger may use different email addresses, phone numbers, or names to book flight tickets. In this scenario, each booking record is a digital entity with multiple attributes. The entity resolution task is to link the tickets to the real-world entity, passenger. Applying our approach above, we can create a graph with tickets connecting to their attributes. Then we can traverse all pairs of tickets that share some common attributes and insert an edge between them if they are considered to be booked by the same person. Lastly all the connected tickets will be grouped together.

2) Owner of the device

More and more devices (e.g. smartphone, television and even refrigerator) now have the capability to connect to the internet. For example, we can use the device network activity records to find the devices that belong to the same person. In each record, we have a timestamp, device id, IP address, URL, and cookie. Following our 3-step approach above, we will first create a graph using the device ID as the digital entity, and the IP address, URL and cookie as the attributes of the entity. In this case, the timestamp information can be stored on the edge between device and IP. In the initialization stage, we will assume each device is owned by a different owner, and connect the owner to the IP address of his device with the connection time range on the edge. In the linking stage, an edge will be inserted between each pair of owners if they were using the same IP during a period of time. Lastly in the grouping stage, the connected owners will be merged together and their devices will be assigned to the same owner.

3) Social media account

One person can have multiple social media accounts such as Instagram, Facebook, and TikTok. Different accounts can be registered using different user names, email addresses, and profile information. By adapting the 3-step approach above, we can also identify the real person behind different social media accounts. The initialization step will be the same where we assign a person to each account, and connect the account information to that person. In the linking stage, different weights can be given to different profile information entries (e.g. user name, email, age, gender). For each pair of persons that share a sufficient amount of information, an edge will be inserted between them with the weight sum of their common information entries. In the grouping stage, the connected component algorithm can be used to group the person vertices together if the total weight on their connections is above a certain threshold.

4) Vehicle identification

Electronic toll collection can significantly reduce traffic congestion. The signal from the IoT devices on the vehicle can be utilized to identify the vehicle. For each toll sensor read, the signal from a variety of IoT devices (GPS, RFID, WiFi, Bluetooth) on all the vehicles within a range can be obtained. To identify which group of IoT devices are on the same vehicle, when loading the toll sensor data, we can insert a vertex for each toll sensor read, and connect this vertex to all the IoT device vertices in this read so that the IoT devices on the same vehicle can be connected through multiple sensor read vertices. In this example, the count of sensor reads where two IoT devices are found can be used to draw linkage between the IoT devices.

Now is the time

We've shown that graph databases are a powerful tool for entity resolution especially on a large data scale. We also introduced a general approach for solving entity resolution tasks with graph algorithms, and discussed a representative use case (i.e. linking multiple login accounts for a streaming media service) in detail to show how the user entities can be resolved by running three queries in the graph database.

Now is an ideal time for you to get started with graph databases. One of the best ways to get started with graph analytics is with TigerGraph Cloud. It's free and you can create an account in a few minutes. Best of all it includes the starter kit called "In-database Machine Learning for Big Data Entity Resolution", which covers the three step approach outlined in this document.

About the Author

Dr. Liu received his B. S. degree from Tsinghua University, China, and his Ph.D. degree from Stanford University. His Ph.D. research topic is focused on applying numerical methods to simulate nanoscale physics and developing physics models based on statistical analysis. While completing his Ph.D. Dr. Liu published over 10 top journal papers including one paper published in the Proceeding of the National Academy of Sciences. He also earned a Ph.D. minor degree in philosophy focusing on the applications of mathematical logic in artificial intelligence. Shortly after graduation Dr. Liu joined TigerGraph, Inc. as a solution architect and currently works on implementing machine learning algorithms on native parallel graph databases.