Machine Learning with Applications 7 (2022) 100237

Contents lists available at ScienceDirect

Machine Learning with Applications

journal homepage: www.elsevier.com/locate/mlwa

# Predicting help desk ticket reassignments with graph convolutional networks

Jörg Schad, Rajiv Sambasivan *, Christopher Woodward

*ArangoDB, CA, USA*

ARTICLE INFO

*Keywords:*
Machine learning
Graph convolutional network
Graph representation

ABSTRACT

Efficient triaging of incident tickets is a critical task in *Information Technology Service Management*. Introducing interventional measures on tickets that are difficult to resolve can help improve the triaging of complex tickets. This work reports a method to predict the resolution complexity of a reported incident. The number of times a ticket is reassigned is a measure of difficulty in resolving the incident. Ticket resolution is associated with a variable workflow. A graph representation of ticket resolution offers advantages from the standpoint of running ad hoc queries. Predicting ticket reassignments requires the application of machine learning to this graph. A *Relational Graph Convolutional Network* is used for this purpose. The developed model provides benefits beyond predicting ticket reassignments accurately. It provides embeddings that can be used to derive insights about the operation of the help desk organization and the users of the help desk.

## 1. Introduction

Efficient resolution of incidents related to issues with software malfunction has been of interest to both research (Gupta, 2020) and the practitioner community (Gartner, 2019). The ticket resolution process in this setting consists of the following three steps (Gupta, 2020):

1. *Ticket Understanding*: This step is concerned with the review of information on the ticket.
2. *Ticket Assignment*: This step is concerned with the assignment of the ticket to a developer or analyst.
3. *Ticket Fixing*: This step is concerned with the details of the technical solution to resolve the reported incident.

The ticket assignment step is a known bottleneck in the ticket resolution process. *Ticket reassignment is the event where a ticket is assigned to more than one developer in the course of its resolution*. Multiple assignments are often the result of errors in reporting or assignment of the ticket, for example, Jeong, Kim, and Zimmermann (2009) reports that between 37% – 44% percent of developer assignments in an open-source project were such errors. Errors in ticket assignment result in increases in costs and resolution time for the ticket. The perspective in this work towards this problem is that identifying tickets that are associated with resolution complexity can help integrate error-proofing measures and additional checks in triaging these tickets. As an illustration, consider the following example. Consider the use of a complex IT application, such as a system that processes requests for access to internal applications used by a company. The workflow for processing submitted requests is complex, requiring multiple approvals and verification steps. As a consequence, triaging the tickets associated with the application requires both experience and training. An inexperienced analyst can make mistakes in ticket assignment because of incorrectly interpreting the ticket or making decisions with insufficient information. By identifying incidents with known resolution complexity, an intervention such as a checklist that provides a list of required input verification steps can improve ticket triaging. Some commercial tools apply machine learning to deliver the intervention using tools like chatbots (Goasduff, 2019). The work involved in resolving a ticket is variable. Some tickets are resolved easily while others may involve multiple steps and involve support from software vendors. A plot of the variable that captures the number of times a ticket is modified is shown in Fig. 1. A review of the figure shows that there is a wide range in the number of updates that a ticket may encounter in the course of its resolution. In a help desk setting, the queries that need to be performed on the data may not be known *apriori*. Help desk technicians need to query ticket data in *ad hoc* manner to resolve tickets. Querying relational data models efficiently requires the use of indexes. If we do not know the queries apriori, in other words, if our queries are *ad hoc*, then we cannot build indexes for them. Without an index and supporting modeling simplifications, such as using views, ad hoc queries on such data would require multiple joins if modeled with a relational approach. A ticket associated with a complex resolution may require multiple joins. This can be computationally

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
*E-mail addresses:* joerg@arangodb.com (J. Schad), rajiv@arangodb.com (R. Sambasivan), christopher@arangodb.com (C. Woodward).

https://doi.org/10.1016/j.mlwa.2021.100237
Received 11 August 2020; Received in revised form 9 December 2021; Accepted 10 December 2021
Available online 16 December 2021

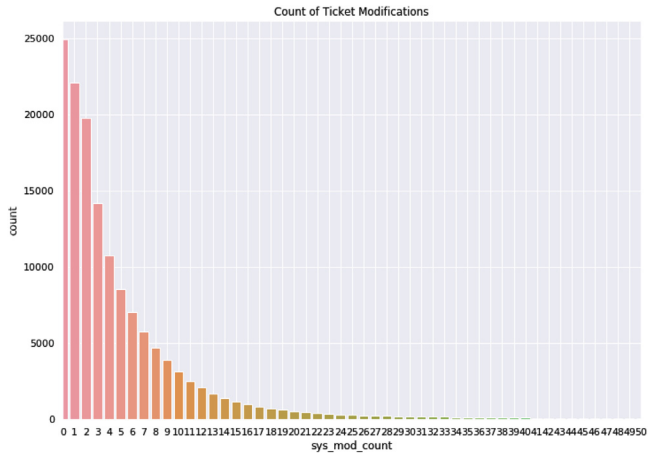


**Fig. 1.** Count plot of ticket modifications.

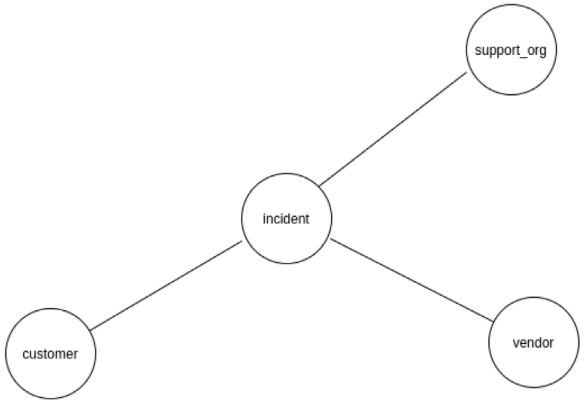


**Fig. 2.** Graph representation of ticket resolution.

expensive. Graph data models greatly simplify ad hoc querying of such data. Graph traversal involves following edges of the graph in contrast to performing joins. Traversals are cheaper computationally. Many graphs, that include knowledge bases such as the data used in this work, have the so-called *six degrees of separation* property (Easley, Kleinberg, et al., 2010). This implies that most nodes are only a few edges away from any given node. Therefore, a graph data model is a good choice for this application. The graph used to model ticket resolution is shown in Fig. 2. The entities involved in the ticket resolution process are the *customer* who reports the problem, the *incident* that captures the details of the problem, and, the *support organization* that resolves the incident. Some incidents require additional support from other software *vendors*. Each of these entities is of a different type. Such data are called *relational data* in *Statistical Relational Learning (SRL)* (Getoor & Taskar, 2007; Khosravi & Bina, 2010). Predicting the type of an entity is a known task in SRL, called *entity classification*. In this work, the ticket resolution complexity is modeled as a property of the *incident* node and is predicted using a *Graph Convolutional Network*. The *Graph Convolutional Network* provides more than an accurate predictive model for ticket resolution complexity. It yields Euclidean representations of the graph elements in Fig. 2. The analysis of these representations provides insights into the ticket resolution process. Additionally, they can also be used to develop features that facilitate the ticket resolution process. For example, they can be used to develop a similarity search feature (Johnson, Douze, & Jégou, 2021) for the incident resolution repository.

The organization of the rest of this paper is as follows. In Section 2, work related to this work is described. In Section 3, a detailed description of the data and its salient characteristics are presented. In Section 4 the machine learning methodology for the application is presented. In Section 5, the results of applying the developed machine learning model are presented. In Section 6, the salient facts from the results observed in this work are summarized.

## 2. Related work

A comprehensive description of the ticket resolution process is available in Gupta (2020). It provides a detailed description of the ticket resolution process and the various approaches to improving ticket resolution. Improving the ticket triaging process by introducing interventions on tickets has been applied with success on incident ticket resolution in software development. Gupta (2020) reports a study where requests for user input contributed to inefficiencies in the ticket resolution process. Machine learning was used to improve the ticket resolution process by identifying tickets that were likely to need further input during its resolution. A rule-based model was then used to identify likely input for these tickets. The approach proposed in this work is similar. In this work, tickets that are likely to have a complex resolution are identified using machine learning. The variable nature of the ticket resolution process motivated the exploration of a graph-based approach to this problem. Results reported later in this work suggest that this approach is indeed effective. Many process attributes, such as textual descriptions of activities, are not available in this dataset for privacy reasons. Application of this approach to event log data with a richer feature set is an area of future work.[1] Other approaches to improving the ticket resolution process have been reported in the literature. For example Anvik, Hiew, and Murphy (2006) identifies an appropriate developer to work on the ticket. Jeong et al. (2009) identifies team structures for ticket resolution. The dataset used in this work has been used by Amaral, Fantinato, Reijers, and Peres (2018) to estimate the resolution time for tickets. The time of opening the ticket and the time of closing the ticket is used to obtain the time to resolution. The machine learning task reported in this work is an entity classification task in a graph. Hamilton, Ying, and Leskovec (2017) provides an overview of machine learning tasks on graphs. For a comprehensive treatment of graph machine learning, see Chami, Abu-El-Haija, Perozzi, Ré, and Murphy (2020) and Hamilton (2020).

## 3. Data characterization

### 3.1. Data description

The dataset for this work is available in the UCI machine learning repository (Dua & Graff, 2017). A description of the dataset is available at Amaral, Fantinato, and Marques (2021). The dataset captures ticket resolution activity at an information technology help desk between March 2016 and February 2017. The original dataset contains a total of 91 attributes. Attributes that provide textual descriptions are not available. The attributes available with the dataset are shown in Table 1.

The modification activity on a ticket is variable (Amaral et al., 2018). The median number of updates on a ticket during its resolution is 6. There are some outliers, for example, there are tickets that have received as many as 58 updates in the course of its resolution. The dataset is largely categorical data with counts being used to capture various aspects of ticket modifications. The two numeric attributes, `sys_mod_count` and `D_reopen_count`, were discretized by quantile binning. A ticket is considered reassigned if the reassignment count associated with it is greater than 0. The attribute we want to predict is binary-valued. The number of unique values for the attributes[2] is shown in Table 2. A review of Table 2 shows that the cardinality of attribute values for many attributes in the dataset is large.

The graph data model for the application is shown in Fig. 2. The attributes of the nodes in the data model are provided in Table 3.

[1] The datasets discussed in Gupta (2020) are candidates for investigation in this regard.

[2] Attributes that are identifiers are excluded.

**Table 1**
ITSM attributes.

| Attribute | Description |
|---|---|
| number | Identifies the incident |
| incident_state | State of the ticket |
| active | Ticket active or closed |
| reassignment_count | **Count of reassignments** |
| reopen_count | Count of ticket reopenings |
| sys_mod_count | Number of ticket updates |
| made_sla | Target SLA exceeded? |
| caller_id | Identifier of the user affected |
| opened_by | User reporting the ticket |
| opened_at | Incident reporting time |
| sys_created_by | Technician creating the ticket |
| sys_created_at | Ticket creation time |
| sys_updated_by | Technician updating the ticket |
| sys_updated_at | Time of ticket update |
| contact_type | Incident reporting mode |
| location | Location of incident |
| category | Description of affected service |
| subcategory | Additional desc. of service |
| u_symptom | User desc of symptom |
| cmdb_ci | CI/CD code of service |
| impact | Impact of incident |
| urgency | Urgency associated with incident |
| priority | Priority associated with incident |
| assignment_group | Support team assigned |
| assigned_to | Support tech assigned |
| knowledge | Documented resolution? |
| u_priority_confirmation | User confirmation of priority |
| notify | Notify user of resolution |
| problem_id | Problem id for ticket |
| rfc | rfc for ticket |
| vendor | Vendor for ticket |
| caused_by | rfc id of cause |
| closed_code | Closure status |
| resolved_by | Tech resolving the issue |
| resolved_at | Time of resolution |
| closed_at | Time of ticket closing |

**Table 2**
Unique values for attributes.

| Attribute | Unique values |
|---|---|
| assignment_group | 79 |
| vendor | 5 |
| caused_by | 4 |
| location | 225 |
| opened_by | 208 |
| knowledge | 2 |
| reassignment_count | 28 |
| active | 2 |
| cmdb_ci | 51 |
| impact | 3 |
| contact_type | 5 |
| category | 59 |
| closed_code | 18 |
| u_priority_confirmation | 2 |
| resolved_by | 217 |
| u_symptom | 526 |
| subcategory | 255 |
| priority | 4 |
| assigned_to | 235 |
| incident_state | 9 |
| urgency | 3 |
| rfc | 182 |
| D_sys_mod_count | 5 |
| D_reopen_count | 5 |

### 3.2. Exploratory data analysis

The dataset captures activities performed on a ticket as it progresses to resolution. Each modification activity related to the ticket is called an event. The dataset has 141,712 events that correspond to 24,918 incidents. There are 79 teams (called assignment groups) and 235 technicians that are assigned to work on these tickets. The help desk has 208 customers. The reassignment count attribute in the dataset reflects the number of times a ticket is reassigned.[3] This attribute is a reflection of the operational efficiency of the help desk organization. A count plot of this attribute is shown in Fig. 3. The ticket state indicates the stage in the resolution path of the ticket. When a ticket is opened, it is in a *new* state, as analysts work on the ticket, this state is updated until it reaches its final *closed* state. A count plot of the reassignment count attribute that includes information about the state of the ticket is shown in Fig. 4. A review of Fig. 4 shows that reassignment activity is not limited to a particular ticket state. Reassignments are done to tickets in various stages of resolution.

A review of Fig. 3 shows that with most incidents there is no need to reassign the ticket after the initial assignment. However, there are some cases where tickets are reassigned. In some cases, there are many reassignments.

If we define ticket reassignment as the event where a ticket is reassigned one or more times, then we have 13986 tickets that are never reassigned and 10932 tickets that are reassigned. A count plot is shown in Fig. 5. This reassignment event is what will be predicted by the machine learning model developed in this work.

**Table 3**
Node features for ITSM graph.

| Incident | Support organization | Customer | Vendor |
|---|---|---|---|
| D_sys_mod_count | assigned_to | node_id | node_id |
| D_reopen_count | assignment_group | opened_by | vendor |
| urgency | node_id | | |
| incident_state | | | |
| u_symptom | | | |
| impact | | | |
| contact_type | | | |
| u_priority_confirmation | | | |
| cmdb_ci | | | |
| rfc | | | |
| problem_id | | | |
| caused_by | | | |
| location | | | |
| knowledge | | | |
| resolved_by | | | |
| subcategory | | | |
| active | | | |
| category | | | |
| priority | | | |
| **reassigned**[a] | | | |
| node_id | | | |

[a]Predicted attribute.

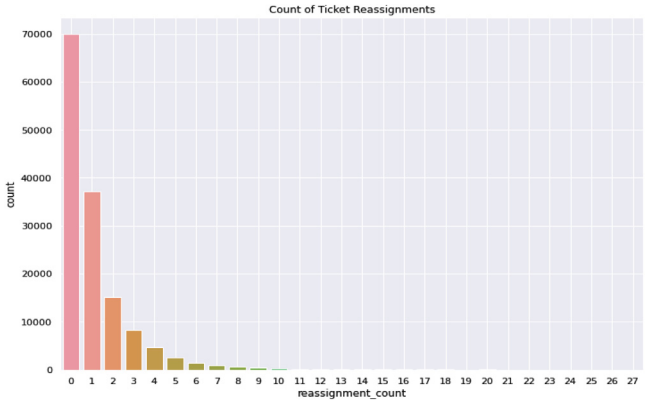


**Fig. 3.** Count plot of ticket reassignments.

[3] The provided description is: "the number of times the incident has the group or the support analysts changed".

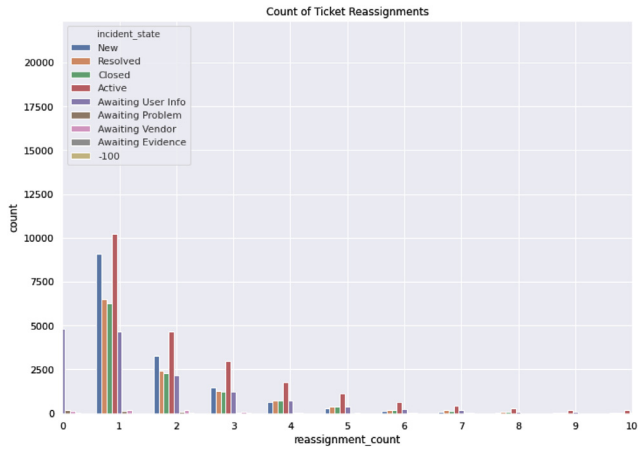


**Fig. 4.** Count plot of ticket reassignments — ticket state included.

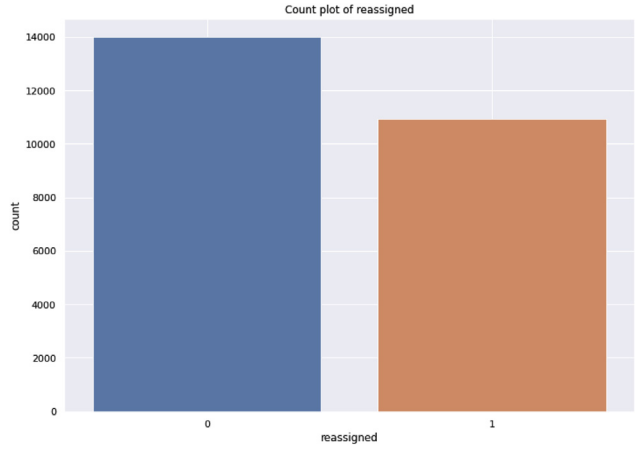


**Fig. 5.** Count plot of the `reassigned` attribute.

### 3.3. Data preprocessing

Node attributes in the graph representation take discrete or categorical values. A technique called embedding is used to obtain a numerical representation of these attributes in machine learning models. This process is used in text and natural language processing applications to obtain a numerical representation of words in a document. Please see (Pytorch, Facebook Research) for an illustrative example of word embeddings. The categorical values are encoded as integers in the input to the embedding layer of the neural network. Accordingly, we need to encode the categorical data in the raw dataset to integers. A preprocessing script was used to accomplish this.

## 4. Methods

The details of the method applied to develop a graph neural network for predicting ticket resolution complexity are provided in this section.

### 4.1. Graph neural networks

A schematic illustrating the computational aspects of a *Graph Neural Network* to perform entity classification on an input graph is shown in Fig. 6 (see, Chapter 5 of Hamilton, 2020). Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the input graph. Some nodes of this graph are labeled $\mathcal{V}_L$, and, some nodes are unlabeled $\mathcal{V}_U$, with $\mathcal{V} = \left(\mathcal{V}_L \bigcup \mathcal{V}_U\right)$. We wish to learn a graph neural network $\hat{f}$, with parameters $\mathbf{W}$, that predicts a label $\mathbf{z}_u$ for a node that has a label $\mathbf{y}_u$, such that the discrepancy between the prediction and the actual label is as small as possible. The discrepancy is determined by a loss function $\mathcal{L}$ that penalizes disagreement between the predicted and the actual label. The loss function used in this work is discussed subsequently. The learning task can be described as (see Section 3.2, Chami et al., 2020):

$$\underset{\mathbf{W}}{\text{minimize}} \sum_{u \in \mathcal{V}_L} \mathcal{L}(\mathbf{z}_u, \mathbf{y}_u) \tag{1}$$

The details of learning the function $\hat{f}$ are as follows. The main idea in a graph neural network is that a Euclidean representation of the graph is used to perform the machine learning task on the graph. This representation is developed at every graph node by setting up a computational mechanism similar to the one shown in Fig. 6 at every node. This figure illustrates the neural computational model to predict node **A** in the graph. With this setup, the representation of every node is determined by its neighborhood and attributes. The representation of the nodes is learned iteratively by exchanging messages with neighboring nodes. The following notation is used to describe the computation at each iteration.

The graph neural network determines an embedding $\mathbf{z}_u$, $\forall u \in \mathcal{V}_L$. At each iteration, a *hidden embedding*, $\mathbf{h}_u$ is determined for each node $u \in \mathcal{V}_L$ using information aggregated from the neighborhood of $u$, $\mathcal{N}(u)$. The following equations summarize the computation of the hidden embedding for node $u$ at iteration $k+1$:

$$\mathbf{h}_u^{k+1} = \text{update}^k\left(\mathbf{h}_u^k, \text{aggregate}^k\left(\{\mathbf{h}_v^k, \forall v \in \mathcal{N}(v)\}\right)\right) \tag{2}$$

$$= \text{update}^k\left(\mathbf{h}_u^k, \mathbf{m}_{\mathcal{N}(u)}^k\right) \tag{3}$$

The computation to determine the embedding, $\mathbf{h}_u^{k+1}$, at iteration $k+1$, consists of two steps. The first step is to aggregate information from the neighborhood of $u$. The embeddings of the neighborhood of $u$ arrive at $u$ during this step. After the information from the neighborhood of $u$ has been collected, the second step is to update the embedding of $u$ with the information collected during the aggregation step, $\mathbf{m}_{\mathcal{N}(u)}^k$. In an entity classification task, the embeddings $\mathbf{h}_u^k$ are used in a classifier to predict a label at the node. For an entity classification task, the update equation, Eq. (3), can be written as:

$$\mathbf{m}_{\mathcal{N}(u)}^k = \sum_{v \in \mathcal{N}(v)} \mathbf{h}_v^k \tag{4}$$

$$\text{update}^k(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) = \sigma\left(\mathbf{W}_{\text{self}}\mathbf{h}_u^k + \mathbf{W}_{\text{neigh}}\mathbf{m}_{\mathcal{N}(u)}^k\right) \tag{5}$$

where:

- $\mathbf{W}_{\text{self}}$ and $\mathbf{W}_{\text{neigh}} \in \mathbb{R}^{d \times d}$ are trainable parameters associated with the node. The size of the embedding, $d$, is a hyper-parameter.
- $\sigma$ is a non-linear activation function used for the classification task, for example, the softmax.

The model's performance can be evaluated using a loss function such as the *cross-entropy* loss:

$$\mathcal{L} = \sum_{u \in \mathcal{V}_L} -\log(\text{softmax}(\mathbf{y}_u, \mathbf{z}_u)) \tag{6}$$

$$\text{softmax}(\mathbf{y}_u, \mathbf{z}_u) = \sum_{i=1}^{i=c} \mathbf{y}_u[i] \frac{e^{\mathbf{z}_u^T \mathbf{w}_i}}{\sum_{j=1}^{j=c} e^{\mathbf{z}_u^T \mathbf{w}_j}} \tag{7}$$

where:

- $\mathbf{z}_u$ is the prediction from model for node $u$
- $\mathbf{y}_u$ is the ground truth for node $u$
- $c$ represents the number of categories for the classification task. In a binary classification task, as in this work, $c = 2$

The loss is evaluated on the training dataset. Eq. (7) can be used to determine the model's estimate of the probabilities for each of these categories. These are used to evaluate the loss function associated with the model's prediction using Eq. (6). Back propagation can then be applied and the gradient of this loss function is determined with respect

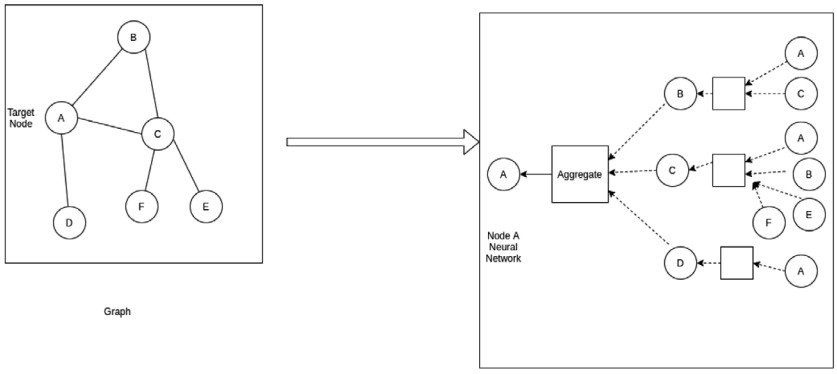


**Fig. 6.** Graph neural network.

to the parameters, ($\mathbf{W}_{\text{self}}$ and $\mathbf{W}_{\text{neigh}}$) and the parameters of the model are updated. Computation can then proceed to the next iteration of the computation where the process described above is repeated. The computational model described above used a single hidden layer for illustration of the computational model. The computational model with additional hidden layers is similar.

In a *heterogeneous* graph, such as the one used in this application, each edge represents a relation. The aggregation step of the computation involves a summation of the relations in the graph. The aggregation function has the following form (see Section 5.4.1, Hamilton, 2020):

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{\tau \in \mathcal{R}} \sum_{v \in \mathcal{N}_\tau(u)} \frac{\mathbf{W}_\tau \mathbf{h}_v}{f_n(\mathcal{N}(u), \mathcal{N}(v))} \tag{8}$$

where:

- $\mathcal{R}$ is the set of relations in the graph
- $f_n$ is a normalization constant that depends on the neighborhood of $u$ and $v$ being aggregated over.

The specific form of the embedding update at each iteration for heterogeneous graphs is discussed next.

### 4.2. Relational graph convolutional networks

In a heterogeneous graph, the aggregation step takes the form given by Eq. (8). The representation of a node $i$ in the $(k+1)$th iteration, $h_i^{(k+1)}$, in a heterogeneous graph takes the form given by Eq. (9).

$$h_i^{(k+1)} = \sigma\left(\sum_{\mathcal{R} \in \mathbf{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{c_{ir}} W_r^l h_j^k + W_0^l h_i^k\right) \tag{9}$$

where:

- $W_r^k$ represents weights at the $k$th iteration with relation $r$
- $h_j^k$ is the representation at the $k$th iteration of node $j$
- $\mathcal{N}_i^r$ represents the neighborhood of the node $i$ with relation $r$
- $c_{ir}$ is a normalization constant such as $\left|\mathcal{N}_i^r\right|$
- $\sigma$ represents the activation function
- $\mathcal{R}$ represents the relations (edges) associated with the node

Training the neural network involves evaluating Eq. (9) for every node in the graph. As noted in Schlichtkrull et al. (2018), as the number of relations increases, the number of parameters in the network increases and this can cause overfitting in rare relations and models of very large size. A regularization approach to solving this problem is discussed in Schlichtkrull et al. (2018). In this work, we found that we could control overfitting with *early stopping*. Increasing the number of training epochs yields increases in both validation and training accuracy, however, the observed accuracy improvement on the validation set saturates after a certain number of training epochs. Increasing the number of training epochs beyond this threshold does not yield better validation set performance. Stopping training at the point where there is no observable increase in validation set performance is a simple but effective strategy to manage overfitting (see Chapter 7, Section 7.8 Goodfellow, Bengio, & Courville, 2016 for more details).

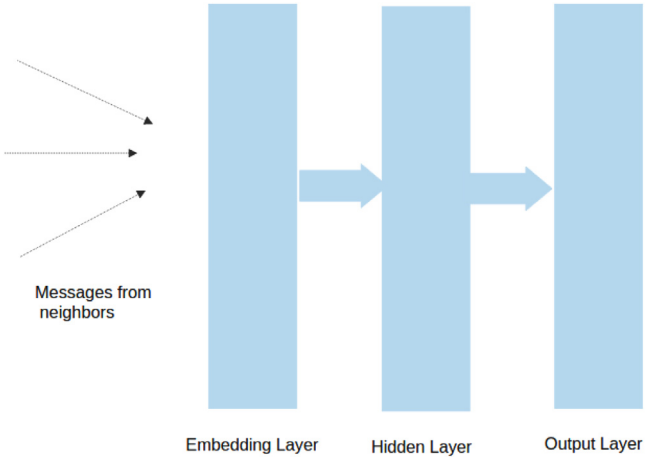


**Fig. 7.** Neural network processing at a graph node.

### 4.3. Network architecture

The neural network associated with a graph node has the architecture illustrated in Fig. 7. Each node receives messages from its neighbors. Since node attributes are categorical, an embedding layer is used to obtain a numeric representation of these attributes. The size of the embedding is a hyper-parameter. The node representation obtained from the embedding then propagates through a hidden layer and finally arrives at the output layer. The size of the hidden layer (number of units) is a hyper-parameter. The influence of these hyper-parameters is evaluated experimentally. We observed overfitting to occur with a single hidden layer and therefore additional hidden layers were not considered in the design.

## 5. Results and discussion

The dataset obtained from applying the preprocessing script is used to create the relational graph convolutional neural network model. A test-train split of 80–20 was used for this work.

### 5.1. Predicting reassignments

A cross-entropy loss and the Adam optimizer were used in this work. A training accuracy of 86.37% and test accuracy of 85.02% were achieved. There are hyper-parameters to be specified for the neural network model such as the number of neurons, parameters of the optimization algorithm, etc. An embedding size of 256 and a hidden layer size (number of neurons) of 256 was used. The optimal values for these parameters were determined using Bayesian Optimization (Snoek, Larochelle, & Adams, 2012). The Adam optimizer (Kingma & Ba, 2015) was used for this work because it is computationally efficient and well suited for problems with a large number of parameters (see Kingma & Ba, 2015 for a list of benefits). The effect of the hyper-parameters and the neural network architecture choices are discussed in the next section.

### 5.2. Influence of hyper-parameters

The hyper-parameters for the model are:

1. Embedding Size
2. Hidden Layer Size

To determine the optimal values for these parameters, embedding and hidden sizes of $\{16, 32, 64, 128, 256, 512\}$ were used in an `hyperopt` (Bergstra, Yamins, & Cox, 2013) experiment. The best results were obtained with a hidden size of 256 and an embedding size of 256. Increasing network capacity beyond these levels produced over-fitting with no gains in observed test set accuracy.

### 5.3. Software for implementation

The following software was used for this work.

1. ArangoDB (ArangoDB (a)) was used to store the incident with a graph representation.
2. The Deep Graph Library (DGL) (Wang et al., 2019) was used to implement the Graph Convolutional Network presented in this work. ArangoDB's Networkx-Adapter (ArangoDB (b)) was used to obtain transformations of the data between ArangoDB graphs and the graph input format (Networkx Hagberg, Schult, & Swart, 2008) used by DGL. An executable version of the Graph Convolutional Network (colab notebook) is available at (ArangoDB (b)).
3. The `hyperopt` (Bergstra et al., 2013) library was used for hyper-parameter optimization.

All software and code in this work are open-source and available at the indicated locations.

### 5.4. Using graph embeddings

The model developed to predict ticket reassignments can provide other benefits. In particular, the embeddings of the vertices of the graphs are useful. These embeddings can be used to triage incidents. The embedded representation of an incident can be used to check if it is similar to an incident that has been resolved. Checking the resolution details of incidents similar to the one that is being analyzed can assist help desk personnel in determining the most effective assignment of personnel for the ticket. The embeddings can be used for enhancing the search of incidents. Rather than simply using keyword search, the embeddings can be used to determine the similarity of a search query to historical resolution activity (Johnson et al., 2021). The embeddings can also be used for personalization.[4]

As a result of model development, we get embeddings for the customer interaction for each incident and the incident. Note that we get the embedding for a customer interaction related to an incident and not the customer itself. This is because the interaction related to the incident is the graph for which the representation is learned. These embeddings can be clustered. This can tell us if there exist cohesive groups of customer interactions and incidents in the data. By identifying the cohesive customer[5] and incident groups, we can obtain insights about each group. For example, customer cluster profiles yield insights about the applications and issues for each customer cluster. For this work, the hdbscan algorithm by McInnes and Healy (2017) was used. Using hdbscan to cluster the data for exploratory insights is attractive for the following reasons. It uses a density-based approach and does not require us to make assumptions about clusters being Gaussian balls. So there is no assumption about the shape of the clusters. The algorithm can also work with noise in the data. The algorithm can tag points as being either noise or belonging to a cluster. Results from clustering the embeddings indicate that these features of hdbscan are indeed useful for this data. Most clusters observed with this dataset are irregularly shaped and the data has significant background noise. Being able to discriminate the clusters from the noise helps us identify customer and incident activities that are noise from those that represent a usage pattern. The minimum size of the clusters (`min_cluster_size`) can be specified to the hdbscan algorithm. In this work, the default value of 5 was used. The objective here is to illustrate that clear clustering tendencies exist in the data and these can be exploited.[6] For a specified value of `min_pts`, hdbscan uses the notion of a *persistence score* to determine the number of clusters (see McInnes & Healy, 2017). This criterion determines an optimal level at which the hierarchical clustering solution must be *cut* to get a set of disjoint clusters. McInnes and Healy (2017) also discusses techniques to improve the performance of the hdbscan algorithm.

### 5.5. Clustering customer interactions

The results of applying hdbscan to cluster the customer interactions are as follows. The algorithm determines a large noise cluster (about 17 K of the nearly 25 K tickets). This implies that these customer interactions are unlike the customer interactions that show clustering behavior. About 8K customer interactions show clear clustering behavior. The clustering solution produced 576 clusters of various sizes. A count plot showing the sizes of the 30 largest clusters is shown in Fig. 8. A tSNE (t Stochastic Neighborhood Embedding) (Maaten & Hinton, 2008) plot can be used to visualize the embeddings of the customers obtained from the trained model. A tSNE plot showing the 10 largest clusters is shown in Fig. 9. A review of the figure shows that the clusters are irregular. Therefore clustering algorithms that assume spherical clusters, like K Means are indeed not optimal choices for this data.

The embeddings can be used to obtain insights about the operational effectiveness of the help desk organization as well as the characteristics of a customer group. Fig. 10 shows the counts of the symptoms reported by the 10 largest customer interaction clusters. There are 230 symptoms.[7] Each cluster is associated with a group of symptoms. Similarly, questions about the number of service level agreement violations, the number of ticket reopenings associated with a cluster, etc., can be analyzed.

[4] For example, a customer may be interested in the incidents of customers like him or her.

[5] In this section, the terms customer and customer interactions are used interchangeably. When used in the context of embeddings, the term customer refers to the embedding associated with the customer interaction for a particular incident.

[6] If the optimal choice is desired, a reasonable approach could involve using a clustering index like the *Calinski–Harabasz* (Caliński & Harabasz, 1974) index to capture the clustering quality associated with a particular choice of `min_pts` and then trying a range of values for this parameter. The value that yields the best clustering quality can then be determined.

[7] These are the descriptions reported by the user when opening a ticket.

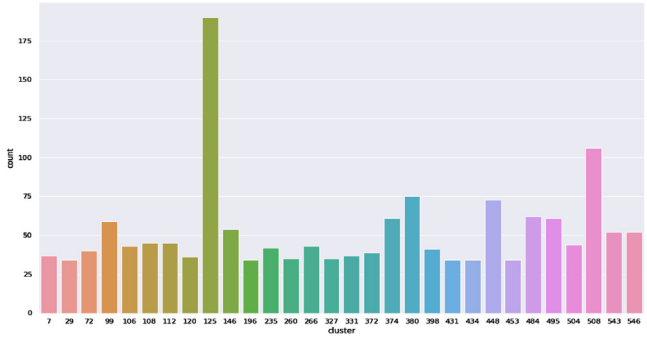


**Fig. 8.** Cluster counts for customers (top 30 by counts).

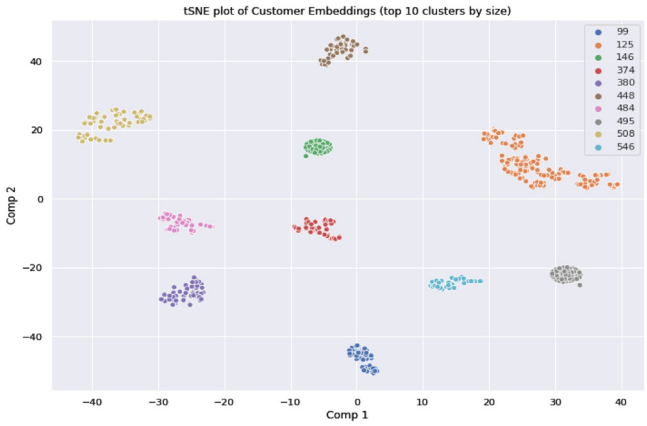


**Fig. 9.** Clustering of customer embeddings.

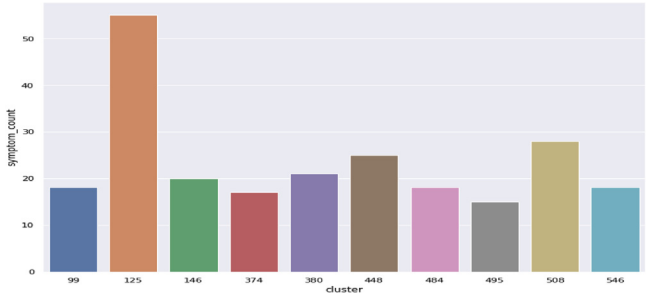


**Fig. 10.** Counts of incident symptoms in customer clusters (top 10 by size).

### 5.6. Clustering incidents

Clustering incidents are similar to clustering customer interactions. As with customer interactions, there is a large noise cluster, of about 11 500 incidents that are unlike incidents that show clear clustering tendencies. About 13 400 incidents show clear clustering tendencies. The clustering solution produced 230 clusters. A count plot showing the sizes of the 10 largest clusters is shown in Fig. 12. The cluster sizes for the incident clusters is shown in Fig. 11. Incident clusters are also irregular in shape. The tSNE plot of incident embeddings is shown in Fig. 12

As with the embeddings of the customer interactions, the clustering solution using the incident embeddings can be used to obtain insights about the operation of the help desk organization. For example, the number of technicians assigned to the 10 largest incident clusters is shown in Fig. 13.

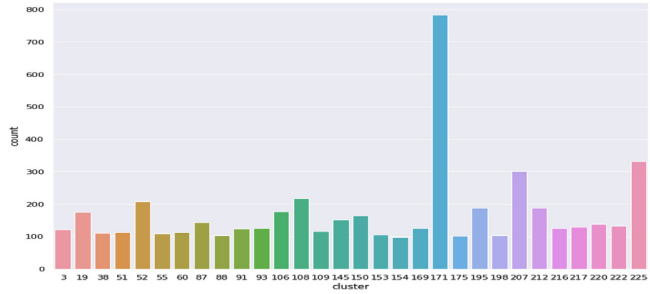


**Fig. 11.** Cluster counts for incidents (top 30 by counts).


**Fig. 12.** Clustering of incident embeddings.

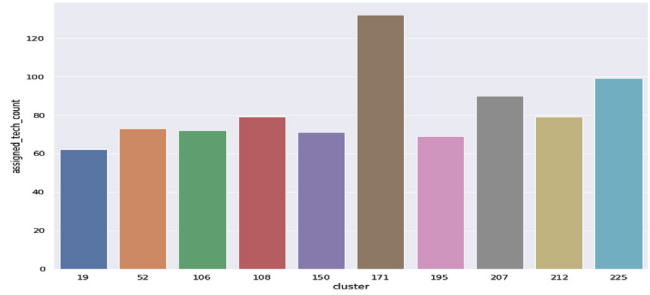


**Fig. 13.** Count of technicians assigned to incident clusters (top 10 by size).

### 5.7. Discussion

The task of predicting ticket resolution complexity can be modeled as an entity classification task. R-GCN's have been applied with success to perform entity classification on graphs with characteristics similar to the one used in this work. For example, Schlichtkrull et al. (2018) applies them to bio-informatics and research co-authorship networks. To the best of our knowledge, we have not found another study modeling ticket resolution complexity as reported in this work. As the results discussed in Section 5 indicate, the R-GCN is not only accurate in terms of predicting ticket resolution complexity, the representations of the entities, or the embeddings, obtained from modeling can be analyzed to extract insights about the ticket resolution process. A clustering solution using the developed embeddings was used to illustrate this. The clustering solution indicates that ticket resolution activity is noisy, but clear clustering tendencies exist. By examining service performance metrics using the clustering solution, a summary view of the operational effectiveness of the help desk organization can be obtained. The embeddings can also be useful for information retrieval tasks involving the embedded entity. The sample of data used in this study covers a large number (over 24 K) of tickets. As a result, the clustering solutions developed in this study yield a large number of clusters. Clustering customer interactions yielded 576 clusters. Clustering incident embeddings yielded 230 clusters. A review of Table 2

shows that tickets capture many attributes and some of these attributes can take a wide range of values. Neural network models are not interpretable. *Post-hoc* techniques are an approach to explaining the behavior and characteristics of neural network models. A consequence of the size of the data and the cardinality[8] of some attributes is that a systematic evaluation and analysis of the results from embedding would merit a separate study. A method based on a game-theoretic approach, called the *Shapley Value (SHAP)*, has been applied to explaining the behavior of neural networks (Molnar, 2020). Recent work applying this to graph neural networks has been reported (Duval & Malliaros, 2021). Developing a comprehensive explanation using such a technique for the model developed in this work is an area of future work.

Though R-GCN has been applied to graph data from a range of domains, they do have the following known limitations. Schlichtkrull et al. (2018) notes that R-GCN is vulnerable to overfitting and may require regularization methods to achieve generalization. Zügner, Akbarnejad, and Günnemann (2018) show that graph neural networks for entity classification tasks can be vulnerable to adversarial attacks. R-GCN does not account for edge features, only node features are used in the model. Kipf and Welling (2017) notes that optimization with full batch gradient descent can be memory intensive for large graphs. Mini-batch gradient descent can alleviate this problem to an extent. Only the immediate neighborhood of a node is considered for the machine learning task associated with the node. There are some tasks for which this is not a reasonable model, see Garg, Jegelka, and Jaakkola (2020) for a discussion of these tasks.

## 6. Conclusion

In a recent customer survey by Microsoft, 95% of the respondents indicated that their choice of brand and loyalty to the brand was determined by customer service. Microsoft (2018). For help desks supporting services that are not related to customers, for example, help desks supporting applications for access to corporate resources, efficient resolution of incident tickets translates to higher organizational productivity and lower costs. Therefore, measures to improve the ticket resolution process provide important benefits. The data and activities associated with ticket resolution are variable. A graph data model is a good fit for data with this characteristic. It facilitates efficient ad hoc querying of ticket data which helps help desks analysts analyze tickets faster. Graph machine learning methods can be leveraged to improve the ticket resolution process. A method to identify tickets that may be complex to resolve was reported in this work. Interventions on such tickets, either manually or by using tools such as chatbots can improve the ticket resolution process. Textual descriptions of ticket details were not available in the dataset used in this work for privacy reasons. Application of the method reported in this work in conjunction with an approach to explain neural network models, such as those discussed in Molnar (2020), to a dataset with textual descriptions of the ticket activities should yield further insights about the ticket resolution process. The results reported in this work encourage the evaluation of graph-based machine learning approaches to help desk applications.

## CRediT authorship contribution statement

**Jörg Schad:** Conceptualization, Methodology, Validation, Writing – review & editing. **Rajiv Sambasivan:** Conceptualization, Methodology, Writing – original draft, Software. **Christopher Woodward:** Software, Validation, Data curation, Writing – review & editing.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

[8] Cardinality refers to the number of unique values an attribute can take.

## References


Amaral, C., Fantinato, M., & Marques, S. (2021). UCI machine learning repository, itsm dataset. URL: https://archive.ics.uci.edu/ml/datasets/Incident+management+process+enriched+event+log#.

Amaral, C. A., Fantinato, M., Reijers, H. A., & Peres, S. M. (2018). Enhancing completion time prediction through attribute selection. In *Information Technology for Management: Emerging Research and Applications* (pp. 3–23). Springer.

Anvik, J., Hiew, L., & Murphy, G. C. (2006). Who should fix this bug? In L. J. Osterweil, H. D. Rombach, & M. L. Soffa (Eds.), *28th International Conference on Software Engineering (ICSE 2006)* (pp. 20–28). Shanghai, China: ACM, http://dx.doi.org/10.1145/1134285.1134336.

ArangoDB (a) Arangodb. (0000). URL: https://github.com/arangodb/arangodb.

ArangoDB (b) Networkx-adapter. (0000). URL: https://github.com/arangoml/networkx-adapter.

Bergstra, J., Yamins, D., & Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning* (pp. 115–123).

Caliński, T., & Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics. Theory and Methods*, *3*, 1–27.

Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., & Murphy, K. (2020). Machine learning on graphs: A model and comprehensive taxonomy. CoRR, URL: http://arxiv.org/abs/2005.03675. arXiv:2005.03675.

Dua, D., & Graff, C. (2017). UCI machine learning repository. URL: http://archive.ics.uci.edu/ml.

Duval, A., & Malliaros, F. D. (2021). Graphsvx: Shapley value explanations for graph neural networks. In N. Oliver, F. Pérez-Cruz, S. Kramer, J. Read, & J. A. Lozano (Eds.), *Lecture Notes in Computer Science.: vol. 12976*, *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021* (pp. 302–318). Bilbao, Spain: Springer, http://dx.doi.org/10.1007/978-3-030-86520-7_19, Proceedings, Part II.

Easley, D., Kleinberg, J., et al. (2010). *Networks, Crowds, and Markets, Vol. 8*. Cambridge university press Cambridge.

Garg, V., Jegelka, S., & Jaakkola, T. (2020). Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning* (pp. 3419–3430). PMLR.

Gartner (2019). Gartner help desk press release. URL: https://www.gartner.com/en/newsroom/press-releases/2019-09-25-gartner-says-only-9--of-customers-report-solving-thei.

Getoor, L., & Taskar, B. (2007). *Statistical Relational Learning*. MIT press Cambridge.

Goasduff, Laurence (2019). Chatbots will appeal to modern workers. URL: https://www.gartner.com/smarterwithgartner/chatbots-will-appeal-to-modern-workers.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Gupta, M. (2020). Improving software maintenance ticket resolution using process mining (extended abstract). In W. M. P. van der Aalst, J. vom Brocke, M. Comuzzi, C. D. Ciccio, F. García, A. Kumar, J. Mendling, B. T. Pentland, L. Pufahl, M. Reichert, & M. Weske (Eds.), *Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Track at BPM 2020 Co-Located with the 18th International Conference on Business Process Management (BPM 2020)* (pp. 26–30). Sevilla, Spain: CEUR-WS.org volume 2673 of CEUR Workshop Proceedings. URL: http://ceur-ws.org/Vol-2673/paperDA6.pdf.

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), Proceedings of the 7th Python in Science Conference (pp. 11–15). Pasadena, CA USA.

Hamilton, W. L. (2020). *Graph Representation Learning*. Morgan & Claypool Publishers.

Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Representation learning on graphs: Methods and applications. *IEEE Database Engineering Bulletin*, *40*, 52–74, URL: http://sites.computer.org/debull/A17sept/p52.pdf.

Jeong, G., Kim, S., & Zimmermann, T. (2009). Improving bug triage with bug tossing graphs. In H. van Vliet, & V. Issarny (Eds.), *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 111–120). Amsterdam, The Netherlands: ACM., http://dx.doi.org/10.1145/1595696.1595715.

Johnson, J., Douze, M., & Jégou, H. (2021). Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, *7*, 535–547. http://dx.doi.org/10.1109/TBDATA.2019.2921572.

Khosravi, H., & Bina, B. (2010). A survey on statistical relational learning. In A. Farzindar, & V. Keselj (Eds.), *Lecture Notes in Computer Science: vol. 6085*, *Advances in Artificial Intelligence, 23rd Canadian Conference on Artificial Intelligence, Canadian, AI 2010, Ottawa, Canada, May 31 - June 2, 2010. Proceedings* (pp. 256–268). Springer, http://dx.doi.org/10.1007/978-3-642-13059-5_25.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

Maaten, L. v. d., & Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, *9*, 2579–2605.

McInnes, L., & Healy, J. (2017). Accelerated hierarchical density based clustering. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)* (pp. 33–42). IEEE.

Microsoft (2018). Gartner help desk press release. URL: https://Stateofglobalcustomerservicereport[Stateofglobalcustomerservicereport].

Molnar, C. (2020). Interpretable machine learning. Lulu. com.

Pytorch Facebook Research. (0000). Word embeddings with pytorch, URL: https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., & Welling, M. (2018). Modeling relational data with graph convolutional networks. In *European Semantic Web Conference* (pp. 593–607). Springer.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems* (pp. 2951–2959).

Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., et al. (2019). Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*. URL: https://arxiv.org/abs/1909.01315.

Zügner, D., Akbarnejad, A., & Günnemann, S. (2018). Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 2847–2856).