Chapter 3
Semantic Vectorization: Text- and Graph-Based Models

Shalisha Witherspoon, Dean Steuer, and Nirmit Desai

Abstract Semantic vector embedding techniques have proven useful in developing mathematical relationships of non-numeric data such as text. A key application enabled by such techniques is the ability to measure semantic similarity between given data samples and find similar data points via encoding comparison. State-of-the-art embedding approaches assume all data are available at a centralized location. However, in many scenarios, data are distributed across multiple edge locations and cannot be aggregated due to a variety of constraints. Hence, the applicability of state-of-the-art embedding approaches is limited to freely shared datasets, leaving out applications with sensitive or mission-critical data.

In this chapter, we address this gap by reviewing novel unsupervised algorithms for learning and applying semantic vector embeddings in a variety of distributed settings. Specifically, for scenarios where multiple edge locations can engage in joint learning, we adapt the proposed federated learning techniques for semantic vector embedding. Where joint learning is not possible, we propose novel semantic vector translation algorithms to enable semantic query across multiple edge locations, each with its own semantic vector space. Experimental results on natural language as well as graph datasets show that this may be a promising new direction.

3.1 Introduction

Exponential growth of IoT devices and the need to analyze the vast amounts of data they generate closer to its origin have led to an emergence of the *edge computing* paradigm [15]. The factors driving such a paradigm shift are fundamental: (a) costs involved in transporting large amounts of data to Cloud, (b) regulatory constraints in moving data across sites, and (c) latency in placing all data analytics in Cloud.

S. Witherspoon (✉) · D. Steuer · N. Desai
IBM Research – Yorktown Heights, Yorktown Heights, NY, USA
e-mail: shalisha.witherspoon@ibm.com; dean.steuer@ibm.com; nirmit.desai@us.ibm.com

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

H. Ludwig, N. Baracaldo (eds.), *Federated Learning*,
https://doi.org/10.1007/978-3-030-96896-0_3

Further, deployments of applications enabled by 5G network architecture rely on edge computing for meeting the low-latency requirements [4].

A critical application of edge computing is the extraction of insights from the edge data by running machine learning computations at the edge agent, without needing to export the data to a central location such as the Cloud [18]. However, most of the recent advances in machine learning have focused on performance improvements while assuming all data are aggregated in a central location with massive computational capacity. Recently proposed federated learning techniques have charted a new direction by enabling model training from data residing locally across many edge locations [8, 17].

However, previous work on federated learning has not been primarily focused on machine learning tasks beyond classification and prediction. Specifically, representation learning and semantic vector embedding techniques have proven effective across a variety of machine learning tasks across multiple domains. For text data, sentence and paragraph embedding techniques such as doc2vec [7], GloVe [11], and BERT [1] have led to highly accurate language models for a variety of Natural Language Processing tasks. Similar results have been achieved in graph learning tasks [3, 16] and image recognition tasks [2, 10]. Key reasons behind the effectiveness of semantic embedding techniques include their ability to numerically represent rich features in low-dimensional vectors and their ability to preserve semantic similarity among such rich features. Further, little or no labeled data is needed in learning the semantic vector embedding models. Clearly, semantic vector embedding will remain a fundamental tool in addressing many machine learning problems in the future.

This chapter addresses the challenge of representation learning when data cannot reside in a centralized location. Two new research problems are introduced that generalize federated learning. First, we introduce the problem of learning semantic vector embedding wherein each edge site with data participates in an iterative joint-learning process. However, unlike the previous work on federated learning, the edge sites must agree on the vector-space encoding. Second, we address a different setting where the separate parties are unable to participate in an iterative joint-learning process. Instead, each edge site maintains a semantic vector embedding model of its own. Such scenarios are quite common where edge sites may not have continuous connectivity and may join and leave dynamically.

It is important to note that while the edge scenario motivated the study and development of the aforementioned research problems, they are not limited to edge scenarios and can be applied in many settings where data cannot be aggregated centrally, such as in mobile or enterprise computing use cases. In the edge environment, an edge device, be it a mobile phone, computer, sensor, etc., can be treated as a party in the traditional federated learning scenario and thus can be utilized in any setting federated learning is carried out.

3.2 Background

Before discussing the topic of semantic vector federation, it is necessary to define several terms and techniques used in the approach. The first is a brief overview of Natural Language Processing and natural language embedding. It is also necessary to define the algorithms that utilize components of natural language embedding that allow for the federated semantic search that follows.

3.2.1 Natural Language Processing

Natural Language Processing (NLP) is a broad field covering computer interpretation of human speech and text. NLP has a long history of study within computer science, with the first explorations going back to the 1950s. During that time, work focused on breaking speech and text into its formative components and interpreting language as ontologies from which computers could more easily reason [13, 14].

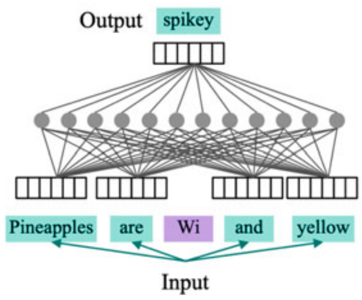
In the 1990s, compute power and new algorithms in the field had advanced sufficiently for research to move away from complex rules and toward utilizing machine learning algorithms [5, 6, 12] to identify patterns. Researchers moved to focus on unsupervised algorithms, as the abundance of information was difficult or impossible to classify. The proliferation of more complex algorithms such as neural networks in recent years has served as the backbone for continued research into natural language understanding.

Natural language embedding is a technique by which human speech and text are converted into numeric vectors on which a computer can make calculations. This conversion of words into a numeric representation is referred to as vectorization and enables tasks such as finding semantically similar words, clustering documents, classifying text, extracting text features, etc. Techniques such as stemming, lemmatization, and stopword removal can also be used to reduce the size of the corpus into a smaller, but more valuable set of data by removing text that has little information. Once the text has been converted into vectors, similarity functions can be applied. One such example is cosine similarity, which works by projecting two vectors into a two-dimensional space. The cosine angle between these two vectors is then determined where the smaller the angle, the higher the similarity between two vectors. This process is done for the entire corpus of text to generate cosine similarity vector scores for all words.

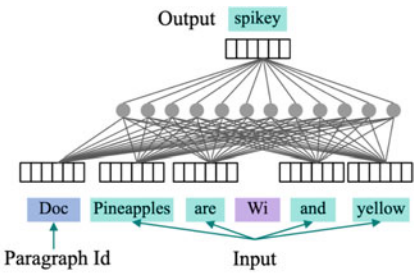
Representation learning and semantic vector embedding techniques have proven effective across a variety of machine learning tasks across multiple domains. For text data, sentence and paragraph embedding techniques such as Doc2Vec, GloVe, and BERT have led to highly accurate language models for a variety of NLP tasks. Key reasons behind the effectiveness of semantic embedding techniques include their ability to numerically represent rich features in low-dimensional vectors and their ability to preserve semantic similarity among such rich features. Furthermore, little or no labeled data is needed in training semantic vector embedding models as they rely on unsupervised learning.

3.2.2 Text Vectorizers

Text vectorizers are a series of algorithms used to embed text data. These algorithms attempt to identify and categorize text into more machine interpretable forms. One of the prominent algorithms to be developed in this area is Word2Vec. Word2Vec was first developed by Tomas Mikolov and team in 2013 [9]. This algorithm was able to solve the challenge of maintaining semantic meaning in text space and allow for words in similar contexts to be correlated. Typically, the corpus is many thousands, or millions of words. Underlying the Word2Vec model is a neural network that takes as input the vectorized words and creates a mapping of the vast input data. Two addition techniques are a part of the Word2Vec algorithm. These are the continuous bag-of-words (CBOW) model and the Skip-Gram model. CBOW works by creating a vector projection of the words that surround a word w in order to predict the word. The number of words to include is defined by a "window." A window describes the number of words before and after the query word in a sentence to include in the vector projection. In a sample sentence such as "the dog jumped over the lazy fox," suppose we wanted to find the vector space of the word "over," the algorithm would look at the context surrounding the word. If we additionally provide a window size of two, our vector projection would include the words "dog" (w-2), "jumped" (w-1), "the" (w+1), and "lazy" (w+2). Skip-Gram works in a reverse fashion where we attempt to predict the words surrounding some particular word. Using the sample sentence again, and using the same word, "over," the approach would attempt to learn that the word "jumped" and "the" are contextually close to "over." See Fig. 3.1 to compare the approaches.



(a)



(b)

Fig. 3.1 Text vectorizers. (**a**) Word2vec. (**b**) Doc2vec, based on the CBOW algorithm (continuous bag-of-words)

In Word2Vec, vectors of fixed dimensions representing each word in the vocabulary are initialized randomly. The learning task is defined as predicting a given word based on the preceding *N* words and following *N* words. The loss function is defined as the error in predicting the given word. By iterating through many sentences during training, the word vectors are optimized using gradient descent and updated to minimize the loss and accurately represent the semantic concept. Interestingly, such semantic vectors also exhibit algebraic properties, e.g., vector representing "Queen" is similar to the one corresponding to subtracting "Man" from "King" and adding "Woman." Doc2Vec is a simple yet clever tweak of Word2Vec where a vector representing an entire document, e.g., a paragraph, is learned along with the words in it.

3.2.3 Graph Vectorizers

Another area of valuable semantic meaning is that of graphs. Graphs are generally structured as a series of nodes linked by edges. A neighborhood defines a portion of the nodes in the overall graph that are connected together. These graphs can be complex or small depending on the datasets. Examples of graph datasets include social network graphs where individuals are nodes and friendships are edges; author collaboration networks where authors are nodes and co-authorships are edges; and road networks where cities are nodes and roads are edges. Given these scenarios, it becomes valuable to find patterns in these potentially massive graphs. Node2Vec is an algorithm for representation learning on graph data that was first proposed by Grover and Leskovec in 2016 [3].

The paper's effort is two-fold; first, by using graphs and node neighborhoods, the algorithm can generate nodes of similar semantic meaning; second, by using graphs where some subsets of the links are missing in an attempt to predict where links should exist. Our work focuses on the first technique of identifying semantically similar nodes. Semantically similar nodes can be described in two ways: homophily and structural equivalence. Homophily describes a scenario where nodes are highly interconnected and, therefore, similar to each other. Structural equivalence describes a scenario where nodes that are similarly connected or fulfill a similar role within the graph are similar to each other. These nodes need not be highly connected or even connected to each other.

As an example, consider a grade school population that consists of all students and staff. Suppose a node represents a single individual and an edge represents individuals attending the same class. A cohort of students of a particular grade are likely to appear in several classes together and act as a neighborhood. A teacher may teach this cohort but may also teach other classes of entirely different students at different times. Students who appear in the same classes would be homophilic and considered a highly interconnected group of nodes. The teacher is structurally equivalent to other teachers who are a point of a single connection to other large

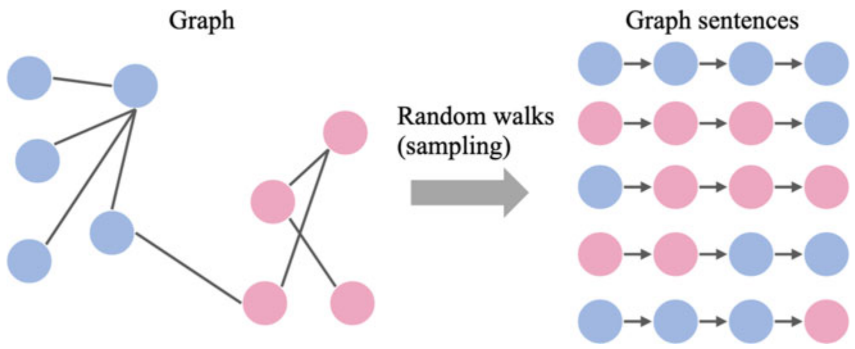


Fig. 3.2 Node2vec: random walks on graphs

groups of students. Structurally equivalent nodes need not be transition nodes into larger neighborhoods.

Nodes that exist on the periphery of a graph with single connections or no connections at all may also be treated as structurally equivalent. The preference to identify nodes via homophily or structural equivalence is treated as a parameter during Node2Vec model training. Given a graph, Node2Vec can learn vector representations for the nodes, which can then be used for node comparison and link prediction. Unlike text sentences where each word is preceded or followed by at most one word, graphs have a complex structure. It is necessary to convert this potentially complex and interconnected graph in a sequence of elements much like a sentence. One of the key innovations behind Node2Vec is mapping a graph to node sequences, a.k.a. graph sentences, by generating random walks and then using Word2Vec to learn the vector representation of the nodes in these sequences. Hyperparameters control the number of walks to generate, walk length, as well as the preference of the walk to keep close to its starting node and potentially revisit nodes it has already seen, or explore further out away from the starting node. Once the walk sequences have been generated, the walks are provided as sentences to a text vectorizer as described above (Fig. 3.2).

3.3 Problem Formulation

With the background on semantic embedding covered, we are ready to formally define the problem of semantic vector federation for edge environments.

In the introduction, we presented two problem scenarios for semantic vector federation in edge environments: the first being when iterative joint learning is possible and the second when edge sites were unable to participate in joint learning. Joint learning is the process by which multiple parties collaborate and share some form of information. In cases where the data is not sensitive or all parties are controlled by a single organization, the raw information could be shared. However, in many scenarios, it may be necessary to minimize what data is being shared with others. The latter scenario was used to inform and design new algorithms for the joint-learning process.

To address the conflicting challenges, we developed novel algorithms for each scenario. In the case of joint learning, prior to beginning the iterative distributed gradient descent, edge sites collaborate to compute an aggregate feature set so that the semantic vector spaces across edge sites are aligned. In the case where joint learning is not possible, edge sites learn their own semantic vector embedding models from local data. As a result, the semantic vector spaces across edge sites are not aligned, and semantic similarity across edge sites is not preserved. To address this problem, we propose a novel approach for learning a mapping function between the vector spaces such that vectors of one edge site can be mapped to semantically similar vectors on another edge site.

3.3.1 Joint Learning

Joint learning can be described as scenarios where synchronous learning is able to transpire, i.e., all parties are able to participate in federated learning at the same time to train a global model. The global model could then be used in performing semantic similarity searches across all edges sites for new data.

The joint-learning algorithm adapts the federated averaging algorithm, which achieves model fusion by averaging the learned weights during iterative rounds of training, to a semantic vector embedding setting. The main challenge in applying federated learning in semantic embedding models is in ensuring that concepts across edge sites are aligned. For example, in the case of text data, if the vocabulary of words is different across edge sites, federated averaging cannot be readily applied because the learned weights of the embedding model are what eventually ends up as the word embeddings, and if they are not aligned, the correct embeddings would not be updated properly during averaging. Hence, a key innovation in the joint-learning adaption is to align the vocabulary of concepts as a prerequisite step in the iterative synchronous training process to ensure consistent embedding across sites.

Figure 3.3 depicts an illustration wherein EDGE1 wants to perform a global search for top-3 experts most similar to person X. Assuming that a Doc2Vec model m_1 has been distributed to all edge sites via joint learning, EDGE1 uses m_1 to vectorize person X's document as vector v_1 and sends v_1 to other sites. Other sites apply a similarity metric, e.g., cosine similarity, to find top-3 nearest-neighbor vectors to v_1 and return the corresponding person identities and cosine similarity score back to EDGE1. After receiving the results from all edge sites, EDGE1 can select the top-3 results having the highest cosine similarity.

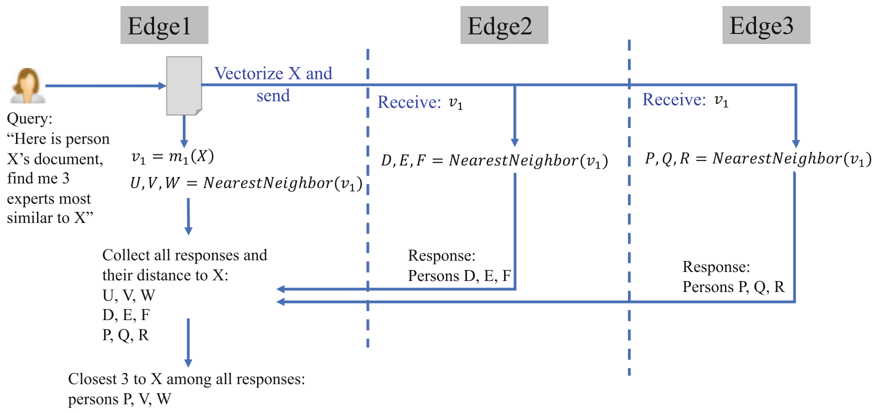

Edge1
Edge2
Edge3
Query: "Here is person X's document, find me 3 experts most similar to X"
Vectorize X and send
$v_1 = m_1(X)$
$U, V, W = NearestNeighbor(v_1)$
Receive: v_1
$D, E, F = NearestNeighbor(v_1)$
Receive: v_1
$P, Q, R = NearestNeighbor(v_1)$
Collect all responses and their distance to X: U, V, W D, E, F P, Q, R
Response: Persons D, E, F
Response: Persons P, Q, R
Closest 3 to X among all responses: persons P, V, W

Fig. 3.3 Semantic search for the motivating example: joint learning

3.3.2 *Vector-Space Mapping*

The problem of *vector-space mapping* of semantic vector embedding is defined as having N edge sites, each edge site i with local dataset D_i and a pre-trained semantic vector embedding model m_i trained on D_i. Each of the edge sites wants to collaborate in performing global similarity search for a new example d across all edge sites but do not want to share their data with each other and are not able to participate in jointly training a common model.

A key property of semantic embedding models is that each has what is known as its own vector space. This means that the real-valued vectors produced for semantic representations are initialized randomly. As an example, because of this property, even if two word embedding models were trained on the exact same corpus, their semantic vectors would be different, and the semantic meaning would not be preserved across the other embedding models. This introduces a concept known as *vector-space mapping*, which is the ability to translate the vector space of one embedding model into another independently trained embedding model's vector space in order to retain the semantic meaning learned across both models and enable queries of similarity among their vectors.

One of the main challenges is to identify a training set of semantically similar words in the different vector spaces and use the corresponding vectors as reference vectors that can be used to generate a function that is capable of transforming any vector from one vector space to the other. Given this, our algorithm makes use of the properties of multi-layer perceptron (MLP) neural networks to potentially learn universal functions and, therefore, the possibility to train such a network to learn the mapping. However, training a MLP model requires a training set that is commonly available to all edge sites. Availability of such training data is highly constrained, especially given that the sites do not wish to share their proprietary datasets with each other.

Hence, another key innovation of our vector-space mapping algorithm is the idea of leveraging any publicly available corpus, regardless of its domain, as a training dataset generator for the mapper MLP model. The formal algorithm definition can be defined as outlined in Algorithm 3.1 and illustrated in Fig. 3.4.

Algorithm 3.1 Vector-Space Mapping Algorithm

```
Input: Local Dataset $D_i$, Public dataset $D_p$, Loss Function $F_i$, Epochs $T$, learning rate $\eta$
Function $Main(D_i, D_p, F_i, \eta)$:
    $m_i \leftarrow TrainDoc2Vec(D_i, D_p, F_i, \eta)$
    store $m_i$
Function $Map_j(m_j, D_p, F_i)$ :
    $W_{i\to j} \leftarrow RandomNN()$
    for all $b \in D_p$ do
        $v_i \leftarrow predict(m_i, b)$
        $v_j \leftarrow predict(m_j, b)$
        $L \leftarrow F_i(v_i, v_j)$
        $\nabla \leftarrow Gradient(L, F_i, W_{i\to j})$
        $W_{i\to j} \leftarrow W_{i\to j} - \eta\nabla L$
    end for
    $m_{i\to j} \leftarrow Model(W_{i\to j})$
    store $m_{i\to j}$
Function $GlobalSearch(d)$ :
    for all $Edge_j \in Edges$ do
        $v_i \leftarrow predict(m_i, d)$
        $v_j \leftarrow m_{i\to j}(v_i)$
        Send query $v_j$ to $Edge_j$
        $V_{sim} \leftarrow$ Receive result vectors from $Edge_j$
    end for
    return $V_{sim}$
```

As shown in Fig. 3.4, consider that a semantic vector embedding model m_1 is trained from local data on EDGE1 and another semantic vector embedding model m_2 is trained from local data on EDGE2. The objective is to train a mapper MLP model that can map vectors produced by vector space of m_1 to the vector space of m_2. An auxiliary dataset D_p that is accessible to both edge sites can serve as the training samples generator and facilitate the training of MLP mapper model. Input to the MLP model are the vectors produced by m_1 on samples of D_p, and the ground-truth labels are the vectors produced by m_2 on the same samples of D_p. Since the input and the output of the MLP mapper model can have a different dimensionality, this approach works even when EDGE1 and EDGE2 choose a different dimensionality for their semantic vectors.

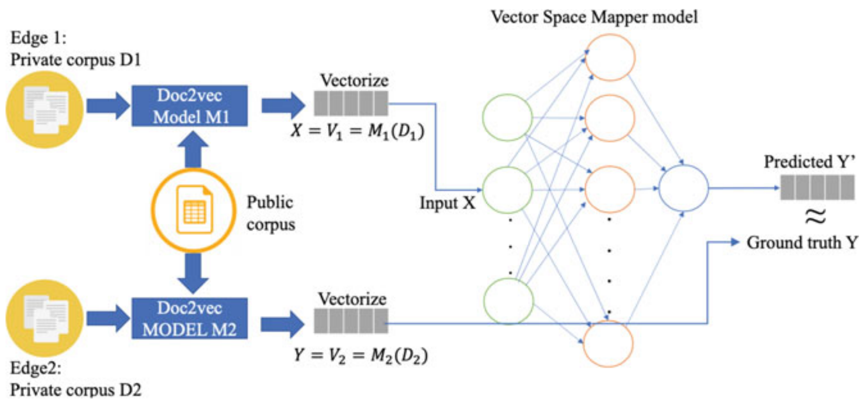


Fig. 3.4 Learning to map vector space of Edge1 to that of Edge2

3.4 Experimentation and Setup

We evaluate the two algorithms of joint learning and vector-space mapping via extensive experiments on two data modalities: natural language and graph. The experiments are anchored on the motivating example of performing a global semantic search for individuals with expertise. The evaluation metric is dependent on the algorithm. For joint learning, we perform an objective evaluation of how well the federated semantic vector embedding model performs relative to the baseline of a centralized model; comparing with a baseline is a standard practice for unsupervised algorithms since there is no ground truth on semantic similarity between samples. And for vector-space mapping, we perform an objective analysis comparing cosine similarity of reference vectors with and without our mapping algorithm being performed.

3.4.1 Datasets

For the natural language modality, we leverage three different datasets: (a) an internal dataset consisting of Slack collaboration conversations, (b) the 2017 Wikipedia dataset with 10K samples, and (c) the 20-newsgroup public datasets with 18,846 samples. For joint-learning experiments, (a) is used for both the centralized and federated experiments. For vector-space mapping experiments, (b) is used as the private datasets, with (c) acting as the public dataset accessible by all edge sites. For the graph modality, we leverage (a) above for the joint-learning experiments but instead of looking at the text content of the posts, we construct a collaboration graph between users.

The Slack dataset (a) consists of natural language conversations across 7367 Slack channels among 14,208 unique users. Of these, only 1576 users having sufficient activity (more than 100 posts) are used in the experiments. All Slack posts of a user are treated as a single document in training the Doc2vec models. For the centralized case, Slack posts of all users are used for training a single Doc2vec model, whereas for the federated case (joint learning), the users are uniformly distributed across two edge sites. No additional knowledge of the organization hierarchy, projects, or teams is included, leaving the models to rely solely on the content of the Slack posts as a basis of semantic vector embedding representing each user.

In constructing a graph from the Slack dataset, each user is treated as a node in the graph, and other users who participate in the same Slack channel as the user are treated as the edges. For avoiding noisy edges due to having channels with a large number of users, a pair of users participating together, i.e., co-occurring, in less than 10 channels do not have an edge between them. Another approach would have been to assign weights to edges; however, Node2vec does not take advantage of edge weight information. The entire graph is used for training the centralized Nodes2vec model. For the federated case, users are randomly assigned to one of the edge sites. When doing so, the cross-site edges are handled in two alternative ways: (1) the cross-site edges are not retained, so each edge site has edges only among the users assigned to the site, called *no retention*, and (2) the nodes involved on cross-site edges are retained on both sites, called *retention*.

3.4.2 Implementation

For the natural language dataset, we use the Doc2vec model architecture with the Skip-Gram PV-DM algorithm with 40 epochs and a learning rate of 0.025. Doc2vec semantic vectors are 50-dimensional real-valued vectors. For the graph dataset, we use the Node2vec architecture with 40 epochs and a learning rate of 0.025. Node2vec semantic vectors are 124-dimensional real-valued vectors. The hyperparameters of the Node2vec favor homophily approach where the return parameter p is favored over the in–out parameter q. We set $p = 0.6$ and $q = 0.1$. The walk length parameter, the number of hops to other nodes from the start node, is set to 20, and the number of walks, the number of iterations of node hopping to perform, is also set to 20.

In the case of vector-space mapping, the mapper model is an MLP model with a single hidden layer with 1200 neurons and a dropout ratio of 0.2. We use the cosine embedding loss in training the MLP as the semantic similarity is based on cosine similarity. ADAM optimizer with a learning rate of 0.00001 and 20 epochs of training with the batch size of 64 was applied.

It is worth emphasizing that these details are provided for completeness and these parameters are quite commonly used in the literature. The objective here is not to produce the best-performing semantic vector embedding models. Instead, we are primarily interested in evaluating the *relative* performance of the federated algorithms compared to the traditional centralized ones. Hence, all of the above parameters are kept the same for the centralized and federated cases.

3.5 Results: Joint Learning

3.5.1 Metrics

For objectively measuring how well the federated algorithms perform relative to the centralized case, the degree of overlap is computed as follows. For a given document d in the dataset, the centralized model is used to vectorize the document, and the set of top-k most similar documents from the dataset is found based on cosine similarity, called d_c^k. Then, using the respective federated algorithm, the set of top-k most similar documents is found for the same document d, called d_f^k. The degree of overlap sim_k then is the ratio of cardinality of the intersecting set and k, denoted as $sim_k = \frac{|d_c^k \cap d_f^k|}{k}$. For multiple documents in the dataset, a simple mean of sim_k is computed over all documents. When $sim_k = 1$, the centralized and federated models produce identical results on semantic search. The idea behind the measure is simple: the higher the sim_k, the closer the federated case performance is to the centralized case. In evaluating the federated algorithms relative to the centralized case, we set $k = 10$.

3.5.1.1 Natural Language

Figure 3.5 shows the distribution of the number of overlaps between the centralized case and the joint-learning case ($sim_{10} \times 10$) when the joint-learning algorithm is applied to the Slack dataset. As indicated by the sim_k of 0.609, for a majority of the users, the joint-learning model found about 6 of the same users found by the centralized model. It is important to note that an average degree of 6 out of

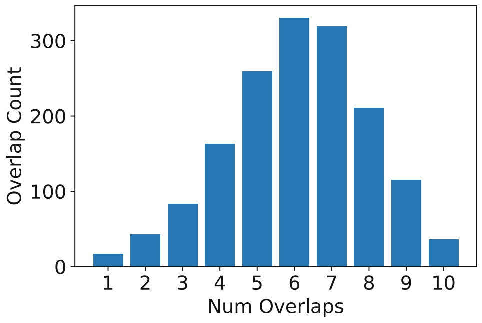


Fig. 3.5 Performance of Doc2vec joint learning relative to centralized learning, $sim_k = 0.609$

10 overlaps is an adequate result, because we found that when retrieving the top 10 results from cosine similarity, the bottom half results are usually inconsistent between experiments, even in the centralized case, due to the fact that the latter results are lower and closer in score, often only separated by trailing decimal digits. Thus, the fact that the federated model was able to overlap with more than half of the results produced from the centralized model demonstrates similar performance from the models.

Based on the above, we can conclude that there is not a significant loss in performance introduced by the joint-learning algorithm when compared to the centralized model, making the joint-learning algorithm a viable alternative to the centralized case.

3.5.1.2 Graph

Figure 3.6 shows the distribution of the number of overlaps between the centralized case and the joint-learning case ($sim_{10} \times 10$) when the joint-learning algorithm is applied to the graph dataset with no retention of cross-site collaborators. As seen in the distribution as well as indicated by the sim_k of 0.138, for a majority of the users, the joint-learning model found almost no users returned by the centralized model. This is not an encouraging result by itself. However, since the cross-site edges are dropped from the graph corresponding to the joint-learning case, valuable information about those users' collaboration behavior is lost compared to the centralized case having the entire graph. Although this explanation is intuitive to validate it, we need to examine the result when the cross-site collaborators are retained and discussed next.

Figure 3.7 shows the distribution of the number of overlaps between the centralized case and the joint-learning case ($sim_{10} \times 10$) when the joint-learning algorithm is applied to the graph dataset with all cross-site collaborators retained across both sites. As seen in the distribution as well as indicated by the sim_k of 0.253, for a majority of the users, the joint-learning model found more than 2 of the same users returned by the centralized model. Compared to the no retention

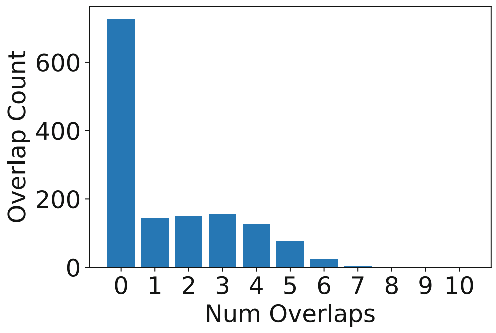


Fig. 3.6 Performance of Node2vec joint learning relative to centralized learning, no retention, $sim_k = 0.138$

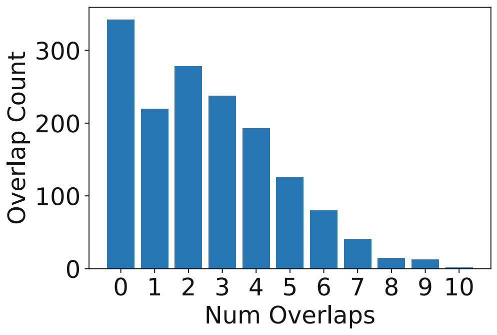


Fig. 3.7 Performance of Node2vec joint learning with collaborator retention relative to centralized learning, $sim_k = 0.253$

result, this is a significantly better result. Thus, the explanation above is validated as retaining the cross-site collaborators clearly helps the joint-learning model achieve more accurate user embedding.

Although the difference between the Node2vec joint-learning results above can be explained by the difference in retention policy, the inferior results of Node2vec when compared with Doc2vec require further investigation. One hypothesis is that the random assignment of users to edge sites can have an adverse effect on the joint-learning performance because such an assignment can have an uneven effect on the collaborative user clusters in the graph. For example, one edge site may end up having most of its collaborative clusters unaffected, whereas another may have its collaborative clusters split into two sites. Although the immediate collaborators may be preserved via cross-site retention, the higher-order collaborations are still affected.

3.6 Results: Vector-Space Mapping

To construct the required vector spaces, we used 10,000 randomly shuffled subsamples from the 2017 Wikipedia dataset as the private data on two edge sites to train two Doc2vec models using different initial random weights. For our public dataset used to generate input and ground-truth vectors for training the MLP mapper model, we leveraged the 20-newsgroup data consisting of 18,886 samples. Our experiments focused on mapping the vector space of EDGE1 to EDGE2.

3.6.1 Cosine Distance

To illustrate the impact of not having mapping across vector spaces, we measured the cosine similarity between vectors for the same documents in both vector spaces. Without mapping, the resulting cosine distance distribution was shown to have a similar distribution to orthogonal vectors, which is essentially akin to

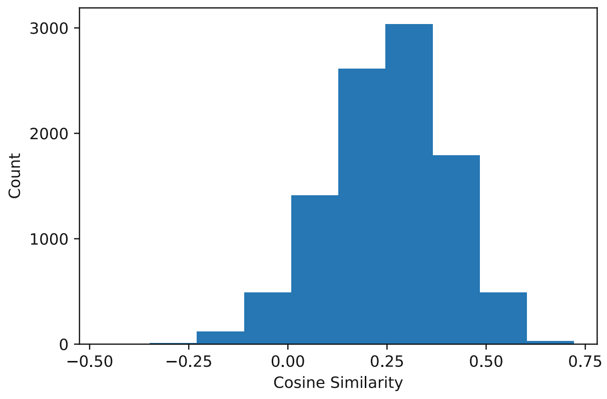


Fig. 3.8 Distribution of cosine distance, no mapping

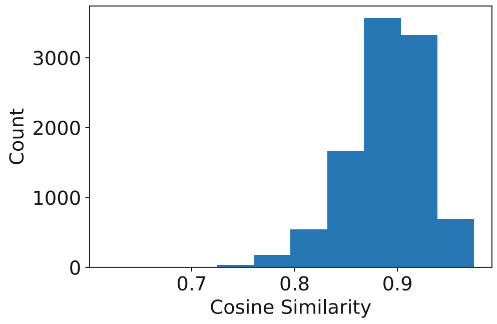


Fig. 3.9 Distribution of cosine distance, mapping performed

comparing random vectors, as shown in Fig. 3.8. For comparison, Fig. 3.9 shows the distribution of cosine distance after mapping the vector spaces, which shows a significant shift in the mean and variance of the distribution away from the random distribution and toward a similarity around 1.0.

3.6.2 Rank Similarity

To further determine the quality of the vector-space mapping, we measured the rank similarity of comparable vectors in both vector spaces. To do this, we vectorized documents in EDGE1's vector space and performed the mapping into EDGE2's vector space to find its 20 nearest matching vectors. If the nearest matching vector was the same as the test document, we gave it a rank of 0; otherwise, we assigned it the rank that it appeared in the similarity result. In cases where the test document was not returned in the similarity result, we assigned it a rank of 20. As shown in Fig. 3.10, we achieve a 0.95 percent accuracy of a perfect match between the

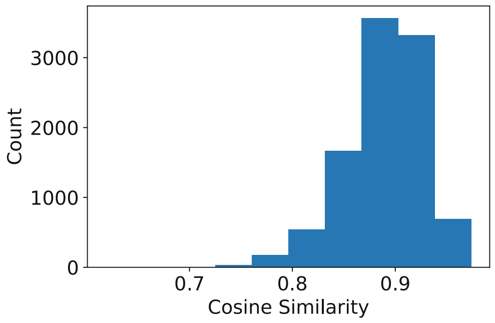


Fig. 3.10 Distribution of rank similarity, mapping performed

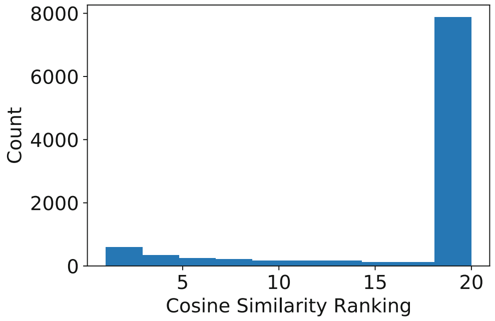


Fig. 3.11 Distribution of rank similarity, no mapping

document vectors after mapping. We also performed the rank similarity experiment without mapping, shown in Fig. 3.11, which resulted in a mere 0.03 percent accuracy for matching the appropriate vector, and the majority of the results gives a rank of 20. Thus both results illustrate the effectiveness of our vector-space mapping algorithm for semantic search across independently trained local models.

3.7 Conclusions and Future Work

With the increasing regulation and the growth in data originating at the edge, edge computing is poised to be a critical area of research with significant impact on how IT systems are developed, deployed, and managed. This chapter introduced the novel research direction of federated semantic vector embedding, building on the unique combination of the well-known techniques of federated learning and semantic vector embedding. Specifically, two research problems were formulated to cater to two separate settings in which edge sites want to collaborate in performing global semantic search across sites without sharing any raw data.

The first setting, called joint learning, is when the edge sites have a tightly coupled collaboration to participate in a synchronous joint-learning process and have an agreement on the model architecture, training algorithm, vector dimensionality, and data format. A novel algorithm to address the joint-learning problem is presented with the novel idea of vocabulary aggregation before starting the iterative federated learning process.

The second setting, called vector-space mapping, is when the edge sites do not agree on the various parameters of joint learning or cannot participate in a synchronous process as they may need to join and leave dynamically. This is clearly a challenging setting and one of great significance in practice. Based on the novel idea of training another model to learn the mapping between vector spaces based on a public dataset from any domain, an algorithm for addressing the vector-space mapping problem was presented.

Experimental evaluation using multiple natural languages as well as graph datasets shows that these algorithms show promising results for both algorithms compared to the baseline centralized case where all data can be aggregated on one site. Several important research questions remain open. How do these algorithms scale in the number of edge sites, differences in data distributions, and the amount of data at edge site? How do we interpret such semantic vectors and explain the similarity results they produce? The work covered here is one of the first in the area of federated semantic vector embedding and has unlocked several key challenges for future research.

References

1. Devlin J, Chang M, Lee K, Toutanova K (2018) BERT: pre-training of deep bidirectional transformers for language understanding. CoRR abs/1810.04805, http://arxiv.org/abs/1810.04805,1810.04805
2. Frome A, Corrado GS, Shlens J, Bengio S, Dean J, Ranzato M, Mikolov T (2013) Devise: a deep visual-semantic embedding model. In: Advances in neural information processing systems, pp 2121–2129
3. Grover A, Leskovec J (2016) Node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, KDD'16. Association for Computing Machinery, New York, pp 855–864
4. Hu YC, Patel M, Sabella D, Sprecher N, Young V (2015) Mobile edge computing—a key technology towards 5G. ETSI White Pap 11(11):1–16
5. Kanerva P, Kristofersson J, Holst A (2000) Random indexing of text samples for latent semantic analysis. In: Proceedings of the 22nd annual conference of the cognitive science society, vol 1036. Erlbaum, New Jersey
6. Uesaka Y, Kanerva P, Asoh H, Karlgren J, Sahlgren M (2001) From words to understanding. In: Foundations of real-world intelligence. CSLI Publications, p 294). chapter 26
7. Le Q, Mikolov T (2014) Distributed representations of sentences and documents. In: International conference on machine learning, pp 1188–1196
8. McMahan HB, Moore E, Ramage D, y Arcas BA (2016) Federated learning of deep networks using model averaging. CoRR abs/1602.05629. http://arxiv.org/abs/1602.05629,1602.05629
9. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. 1301.3781
10. Norouzi M, Mikolov T, Bengio S, Singer Y, Shlens J, Frome A, Corrado G, Dean J (2014) Zero-shot learning by convex combination of semantic embeddings. In: Proceedings of 2nd international conference on learning representations

11. Pennington J, Socher R, Manning CD (2014) GloVe: global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp 1532–1543
12. Sahlgren M, Kanerva P (2008) Permutations as a means to encode order in word space. In: Cognitive science—COGSCI
13. Salton G (1962) Some experiments in the generation of word and document associations. In: Proceedings of the fall joint computer conference, AFIPS'62 (Fall), 4–6 Dec 1962. Association for Computing Machinery, New York, pp 234–250. https://doi.org/10.1145/1461518.1461544
14. Salton G, Wong A, Yang CS (1975) A vector space model for automatic indexing. Commun ACM 18(11):613–620
15. Satyanarayanan M (2017) The emergence of edge computing. Computer 50(1):30–39
16. Wang Q, Mao Z, Wang B, Guo L (2017) Knowledge graph embedding: a survey of approaches and applications. IEEE Trans Knowl Data Eng 29(12):2724–2743
17. Yang Q, Liu Y, Chen T, Tong Y (2019) Federated machine learning: concept and applications. ACM Trans Intell Syst Technol (TIST) 10(2):1–19
18. Zhou Z, Chen X, Li E, Zeng L, Luo K, Zhang J (2019) Edge intelligence: paving the last mile of artificial intelligence with edge computing. Proc IEEE 107(8):1738–1762