Zero-shot Fine-grained Entity Typing in Information Security based on Ontology

Han Zhang[1], Jiaxian Zhu[1], Jicheng Chen[2], Junxiu Liu[3], and Lixia Ji[1,*]

[1]Zheng Zhou University, No.100 Science Avenue, Zhengzhou 450001, China.

[2] Information Engineering University, No.62 Science Avenue, Zhengzhou 450000, China.

[3]Intelligent Systems Research Centre, School of Computing, Engineering & Intelligent Systems, Ulster University, Magee campus, Northern Ireland, BT487JL, United Kingdom.

[*]Corresponding author: Lixia Ji.   Email: jilixia@zzu.edu.cn

Abstract： *The field of information security suffers from the lack of labelled entities. This study proposes a zero-shot hybrid approach, combining a clustering algorithm with a method for representing category labels, to classify fine-grained entity typing based on unified cybersecurity ontology (UCO) to address this issue. However, certain category labels in UCO do not have distinct domain features, while certain abbreviations cannot be obtained directly from word embedding using Word2vec. Thus, we propose a new method, referred to as mixed entities and hierarchy of UCO (MEHC), to represent the category labels. Moreover, to further improve the performance of fine-grained entity typing we propose the triClustering algorithm to re-cluster coarse-grained classification results or determine corresponding types for new entities, based on the theorem that the sum of two sides of a triangle is greater than the third. The experimental results prove that our triClustering algorithm can effectively shorten the computation time and that the proposed hybrid method is superior to other baselines for information security applications.*

Keywords: *fine-grained entity typing; clustering algorithm; representation method for categories; information security; unified cybersecurity ontology*

1. Introduction

Fine-grained entity typing is beneficial to many natural language processing tasks, such as entity linking [1,2], relation extraction [3,4], and knowledge base completion [5,6]. In this study, we aim to utilise existing unified cybersecurity ontology (UCO) [7] to construct a formal knowledge graph, and thereby fill a gap in the information security domain. UCO was employed because it provides a common understanding of the cybersecurity domain and has been extended with numerous relevant cybersecurity standards, vocabularies, and ontologies. It contains 106 classes and 633 axioms. Moreover, compared with other ontologies in the information security domain, it features a more detailed classification system and axioms [7]. In our previous study [8], we proposed a model that identified named entities from crowdsourced annotations (coarse-grained typing). Herein, to obtain a simple knowledge graph of information security, we populated it with the extracted entities using fine-grained typing.

Generally, such research focuses on the general field [9-20], and to the best of our knowledge, no such studies exist in the information security domain. The primary reasons are as follows. (i) Although the general domain possesses certain mature knowledge bases that can be employed for fine-grained entity typing, such as Wikipedia, WordNet, and DBpedia, but they are not professional enough for the field of information security. Thus, studies such as [9-15], are not suitable for application in this domain. (ii) Furthermore, there is the absence of scaled, labelled data in the information security domain for use as a training corpus for classification models. The schemes that were proposed in [16-20] require a considerable amount of labelled data as a training corpus. Moreover, although certain studies have been conducted on zero- or few-shot entity classification for general applications [21-23], the features of the information security domain have been neglected.

For example, in the information security field, the feature vectors of various categories obtained by Word2vec lack the domain features of the field. For instance, 'Consequence' and 'Means' classes are common in the general domain; however, if Word2vec is used to obtain the representations of these two categories directly, the lack of domain features in the vectors results in bias in subsequent work. Further, certain abbreviations exist in the UCO categories; for example, TTP denotes 'tactics, techniques, and procedures', and cannot be mapped as a single word. However, this implies that TTP cannot be used to obtain representations directly from Word2vec. The scheme in [23] proposed that category labels be denoted via representative entities and category hierarchies. However, when these entities and categories are phrases, the use of only the average of the sum of word vectors, constituting the phrases, cannot accurately provide the domain features of the phrase. For example, in the general field, the three words in the entity 'steal login credentials' may be assumed to equally contribute to the representation of the phrase; however, in the information security domain, the word 'login' deserves greater domain features than the other two words. Furthermore, the representation of these entities (i.e., the parent of a category and the category itself) should be proportioned considering their importance in the final composition of the feature representation of the category. In addition, from the related literature on fine-grained entity typing [9-25], we determined that few researchers have considered the problem of misclassification in coarse-grained entity typing tasks. However, the inaccuracy in the results of coarse-grained typing certainly results in performance degradation in fine-grained entity typing tasks.

Therefore, we proposed a hybrid method that combines a clustering algorithm with a method for representing category labels. The contributions of this study are as follows.

a) We proposed a new method for fine-grained entity typing in the information security field based on UCO, eliminating the use of a training corpus.

b) We adopted a novel method referred to as mixed entity and hierarchy (MEHC) for representing category labels. This method utilises category hierarchy and representative entities to solve the problem of UCO category representation and introduces a feed-forward neural network, as well as a pooling mechanism, to learn the domain features that represent the categories and entities. The feed-forward neural network primarily aids in modelling the relationship between the words in phrases, whereas the pooling mechanism is used to model the contributions of the phrase and compositional word embeddings to final phrase representations. Our experimental results show that the MEHC method is superior to the other baselines considered in this study.

c) To further improve the performance of fine-grained entity typing, we suggested that the coarse-grained classification results should be clustered again. To reduce the computation time, we proposed the triClustering algorithm to process the results of coarse-grained typing. Through experiments, we demonstrated that this algorithm can effectively shorten the computation time and that re-clustering the results from coarse-grained classification can effectively improve the subsequent fine-grained classification.

The remainder of this paper is organised as follows. Section 2 discusses related works on fine-grained entity typing, and briefly introduces entity typing. Section 3 examines (i) the representation of category labels, (ii) reprocessing of coarse-grained typing, and (iii) fine-grained typing without a training corpus. Section 4 presents our experimental results; furthermore, based on the results of our previous work on coarse-grained typing, we discuss these results and compare the approach proposed in this study with other zero-shot and few-shot fine-grained entity typing methods. Finally, in Section 5, we present our conclusions and directions for future work.

2. Related Works

Entity typing is a task that infers the types of entities mentioned in any given text. Although it is similar to named entity recognition (NER), they differ in certain respects. NER extracts entities from unstructured text and classifies them, typically using coarse-grained typing, whereas entity typing primarily refers to fine-grained typing. For example, consider the sentence ‘Sleazy Android app developers continue to sneak their fake apps by the Google Play gatekeepers’. In this sentence, the word ‘Android’ is labelled the ‘software’ class by NER. However, if we desire to locate it in the ontology, it needs to be placed under the ‘operating system’ category under ‘software’, which is a fine-grained entity typing task. Extensive research has been conducted on NER [26,27], which will not be repeated in this paper.

Several studies on entity typing have been conducted in the past [9-25]. In general, these studies can be divided into unsupervised and semi-supervised groups depending on the approach used.

Unsupervised approaches focus on distant supervision via a knowledge base. Ling et al. [15] used distant supervision and features, such as word and Part-Of-Speech features, to tag entities with multiple labels. A linear classifier perceptron was used for multi-label classification, and the authors assumed that the entities existed in Wikipedia. However, this study failed to suggest a solution for entities that do not exist in Wikipedia. Consequently, Zhou et al. [20] considered the possibility that entities may not exist in Wikipedia and assumed that each entity corresponded to certain type-compatible entities in Wikipedia. Their work aimed to ground a given mention to a set of type-compatible Wikipedia entries and thereafter infer the types of target mention using an inference algorithm that utilises the types of these entries. However, these studies were limited to certain domains, genres, and languages; thus, Huang et al. [21], aided by a knowledge base, proposed a novel unsupervised entity typing framework, which combined symbolic and distributional semantics, to solve this problem. First, they learned a general embedding for each mention of an entity, composed an embedding for specific contexts using linguistic structures, linked each mention to the knowledge bases, and then learned its related knowledge representations. Subsequently, they used a hierarchical clustering and linking algorithm to type all mentions based on the new representations. Although these studies have achieved good results, their reliance on the assistance of an external knowledge base in methods renders them unsuitable for application to the field owing to the lack of large-scale and extensive coverage of knowledge base, such as information security.

By contrast, the semi-supervised approach focuses on learning the representation of categories or entities. Dai et al. [24] proposed context-aware clause representations to predict the situation entity types of the clauses. In recent years, language pretrained models have been developed rapidly. Onoe et al. [25] input each mention and its context into a BERT-based model to embed a particular mention in box space. This model leverages typological clues present in the surface text to hypothesize a type representation for the mention. However, these approaches are dependent on the context of the entity in the sentence and thus are not suitable for the case of no context. To address this, Xu et al. [19] proposed an end-to-end solution with a neural network model using a variant of the cross-entropy loss function and a hierarchical loss normalization to address the problems involving out-of-context and overly specific labels, respectively. However, a significant amount of annotated data was required to train the model. Further, Ma et al. [23] eliminated the need for huge training data. They presented a label embedding method that incorporated prototypical and hierarchical information to learn pre-trained label embedding, focussing on the presentation of category labels. However, when learning category representation, this method considers the mean of all representative entities to represent the category, that is, the mean of all word representations of all constituent entities to represent the category, without considering the weight of word features in special fields.

Unlike these studies, we focused on the fine-grained entity typing problem in the information security field, which eliminates the need for annotation data and modelling knowledge bases. Therefore, we propose a zero-shot method for fine-grained entity typing, without using distant supervision. In addition, we consider the problem of accuracy of coarse-grained classification results and the representation of certain category labels, which lack domain features, or for which the embedding cannot be directly obtained from Word2vec.

3. Methods

Based on a previous study, we formulated fine-grained entity typing as a multiclass classification problem. We considered a set of classes $\{c_1, c_2, \cdots, c_n\}$, where c_i represents a category in the coarse-grained classification and a top-level category in UCO. Two sets exist within c_i. One set E_i, which contains all the entities classified under category c_i in the coarse-grained classification. The other is a hierarchy of c_i in UCO, which is represented by $\{l_1c_1, l_1c_2, \ldots, l_1c_n, l_2c_1, \ldots, l_2c_n, \ldots, l_mc_1, \ldots, l_mc_n\}$, where l_m represents the deepest layers under category c_i. We need to propose a method that predicts a hierarchy label l_ic_i for an entity e_j, $e_j \in E_i \cup E_{other}$ (E_{other} *means other entities in the information security field*).

As illustrated in Fig. 1, we proposed a hybrid fine-grained entity typing method that combines a novel clustering algorithm (triClustering) with a representation method for category labels (MEHC).

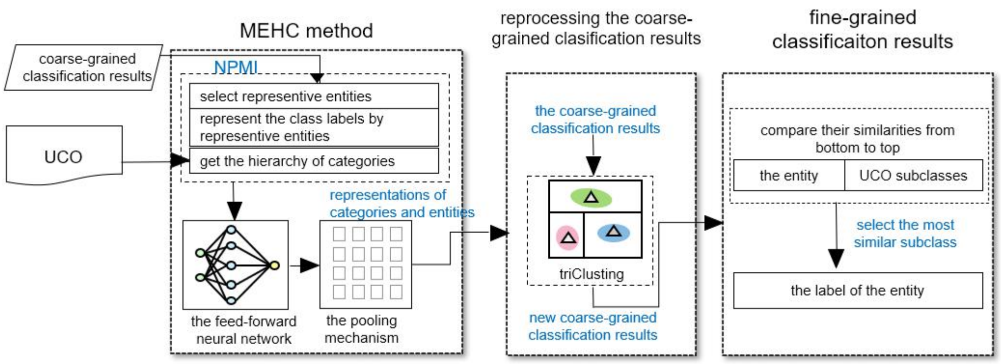

MEHC method
coarse-grained classification results
UCO
NPMI
select representive entities
represent the class labels by representive entities
get the hierarchy of categories
representations of categories and entities
the feed-forward neural network
the pooling mechanism
reprocessing the coarse-grained clasification results
the coarse-grained classification results
triClusting
new coarse-grained classification results
fine-grained classificaiton results
compare their similarities from bottom to top
the entity
UCO subclasses
select the most similar subclass
the label of the entity

Figure 1: Flowchart of zero-shot fine-grained entity classification, where NPMI denotes normalised point-wise mutual information. UCO is an information security ontology that is saved as a file with the suffix, .owl

First, we proposed the MEHC method to represent the category labels. In this approach, two different ways to represent the category labels were utilised according to the location of the categories. For a category in the coarse-grained classification results (one of the top-level categories in UCO), we selected the representative entities under the category as the representation of that class. The categories at other levels in UCO were represented by their parent categories and by themselves. A feed-forward neural network and two max-pooling layers were employed to obtain the representations of the categories. This results in the representation of the category labels being more domain-specific, and thereby solving the problem of using abbreviations as category labels in UCO, where their representations cannot be directly obtained from Word2vec. In addition, to obtain better representations, the feed-forward neural network in the MEHC was used to learn the structure and semantic features of the categories. Subsequently, the previous coarse-grained classification results were re-clustered using the triClustering algorithm to improve the performance of fine-grained entity typing. The triClustering algorithm is an improvement on the *k*-means clustering algorithm, based on the theorem that the sum of two sides of a triangle is greater than the third. It reduces the number of comparisons, and consequently, the computation time. Finally, we determined the most similar subcategories in UCO for each entity through similarity comparisons.

3.1 MEHC Method

The MEHC method was employed to obtain the category representation and it comprises two different representation methods. One is used for learning the representation of c_i, while the other is used for learning the representation of $l_i c_i$, which is the hierarchy of c_i in UCO.

First, we introduce the representations of c_i. Similar to [23], all entities under category c_i were not used to represent it as it may result in certain important features being ignored and secondary features being retained. Instead, we choose a subset of entities that are more representative of the category from

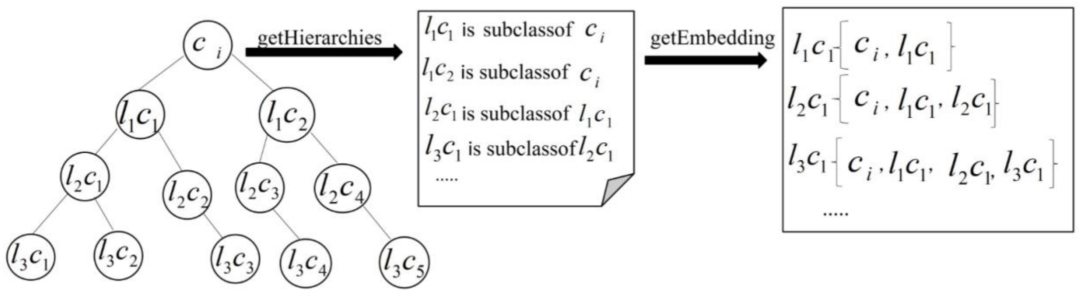
$E_i^{representi\ ve} = \{e_1, e_2, ..., e_n\}, E_i^{representi\ ve} \in E_i$ as the representation of c_i. Although the efficiency of manually selecting representative entities under category c_i is very low, the representative entities can be selected by calculating the Normalized Pointwise Mutual Information (NPMI) [28] between the entities and categories. NPMI is often used in data mining to measure the correlation between two things (such as two words). The NPMI between entity e_i and category c_i is calculated as follows

$$NPMI(e_i, c_i) = \frac{PMI(e_i, c_i)}{-\ln(e_i, c_i)} \tag{1}$$

$$PMI(e_i, c_i) = \log \frac{p(e_i, c_i)}{p(e_i)p(c_i)} \tag{2}$$

where $p(e_i)$, $p(c_i)$, and $p(e_i, c_i)$ represent the probabilities of e_i, c_i, and combined e_i and c_i, respectively. For each category, the NPMI is computed for all entities, and only the top k is selected as representative entities.

Next, we introduce a representation of $l_i c_i$. Fig. 2 illustrates the main workflow for constructing the vector representations of classes in ontology.

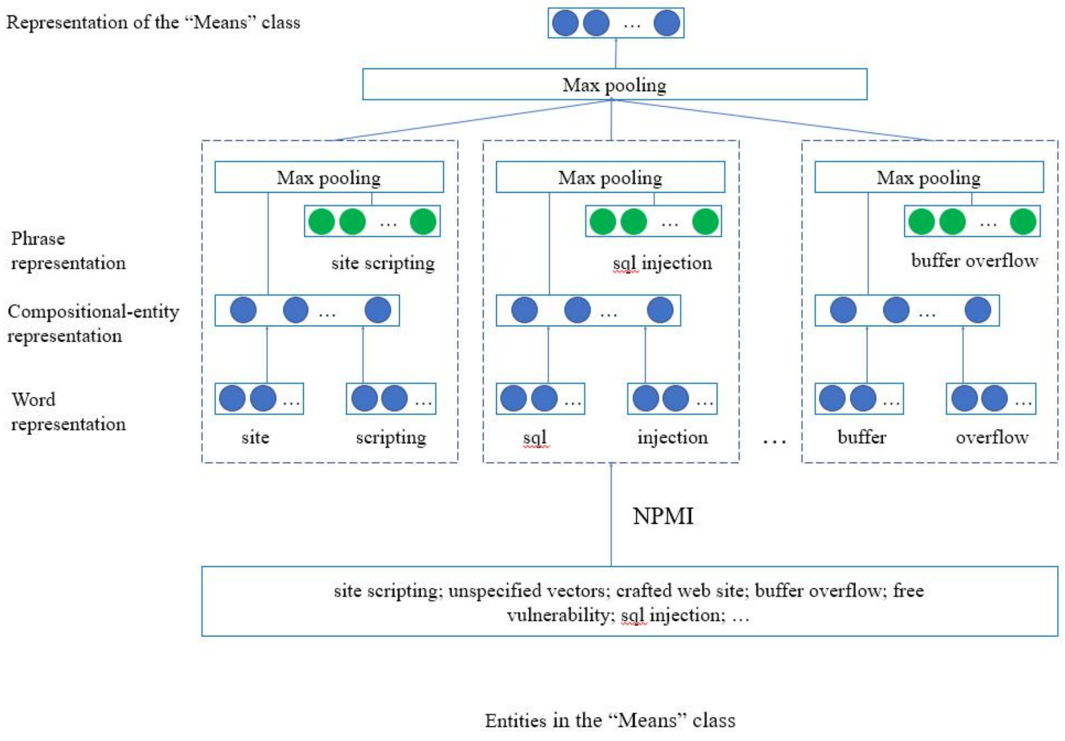


c_i
getHierarchies
l_1c_1
l_1c_2
l_2c_1
l_2c_2
l_2c_3
l_2c_4
l_3c_1
l_3c_2
l_3c_3
l_3c_4
l_3c_5
l_1c_1 is subclassof c_i
l_1c_2 is subclassof c_i
l_2c_1 is subclassof l_1c_1
l_3c_1 is subclassof l_2c_1
.....
getEmbedding
$l_1c_1\{c_i, l_1c_1\}$
$l_2c_1\{c_i, l_1c_1, l_2c_1\}$
$l_3c_1\{c_i, l_1c_1, l_2c_1, l_3c_1\}$
.....

Figure 2: Representation of a category in UCO

We represent each subcategory of c_i as its parent categories. Here, *getHierarchies* is code that was written in Groovy to obtain the hierarchy of categories directly from UCO.

In contrast to [23], which used the mean value of representative entities to represent a category, in practice, the entities of category play a different role when we obtain the representation of the category. Moreover, the representations of these representative entities (or the parent of the category and the category itself) should have different weights when representing the category. We used a feed-forward neural network and max-pooling layer to capture the important features between the word and phrase features and thereafter employed these important features to represent the entity. Further, another max-pooling layer is then used to obtain the important features of these representative entities to create a representation of the category. The architecture of the MEHC is shown in Fig. 3, using the 'Means' class as an example.



Representation of the "Means" class
Max pooling
Max pooling
Max pooling
Max pooling
Phrase representation
Compositional-entity representation
Word representation
site scripting
sql injection
buffer overflow
site
scripting
sql
injection
buffer
overflow
NPMI
site scripting; unspecified vectors; crafted web site; buffer overflow; free vulnerability; sql injection; …
Entities in the "Means" class

Figure 3: Architecture of MEHC. Here, the representative entities in the category that are selected by NPMI can also be replaced by the parent class of the category and the category itself

As shown in Fig. 3, the representation of an entity is a fusion of the phrase and compositional entity representations. The phrase representation e_i^{phrase} is obtained from Word2phrase, which is a function in Word2vec; meanwhile, the compositional entity representation $e_i^{compositional}$ is computed using the feed-forward neural network.

$$e_i^{compositional} = \sum_{j=1}^{N} v_{ij} \otimes w_{ij} \tag{3}$$

$$v_{ij} = \tanh(W[w_{ij}; e_i]) \tag{4}$$

$$e_i = \frac{1}{N}\sum_{j}^{N} w_{ij} \tag{5}$$

where w_{ij} denotes the representation of the j^{th} word in the entity e_i, v_{ij} controls the contribution of the j^{th} word in entity e_i, W is a trainable parameter, and $[w_{ij}; e_i]$ denotes the concatenation of word w_{ij} and the vector mean of the words that make up entity e_i.

The phrase presentation e_i^{phrase} and compositional entity representation $e_i^{compositional}$ were then mixed using a max pooling approach as follows

$$e_i^{final} = \max_{k=1}^{d}(e_{ik}^{phrase}, e_{ik}^{compositional}) \tag{6}$$

where d is the dimension of the vector, e_{ik}^{phrase} and $e_{ik}^{compositional}$ represnt the *kth*-dimension in e_i^{phrase} and $e_i^{compositional}$.

Finally, we acquired the representation of category c_i using another max-pooling layer.

$$c_i = \max_{k=1}^{d}(e_{1k}^{final}, e_{2k}^{final}, \ldots, e_{nk}^{final}) \tag{7}$$

We can train the model parameters with reference to the contrastive loss of the Siames network [29]. We consider a case where a pair of phrases, p_i and p_j. s_i and s_j are respectively sets of phrases that are semantically similar to p_i and p_j; L_+ represents a positive case($y_{ij} = 1$); and, L_- represents a negative case ($y_{ij} = 0$). Then the loss function is obtained as follows

$$Loss = y_{ij}L_+(p_i, p_j) + (1 - y_{ij})L_+(p_i, p_j) \tag{8}$$

$$L_+(p_i, p_j) = \frac{1}{2}(1 - D_w)^2$$
$$L_-(p_i, p_j) = \begin{cases} D_w^2 & if\ D_w > m\arg in \\ 0 & otherwise \end{cases} \tag{9}$$

$$D_w = \frac{p_i * p_j}{\sqrt{(p_i)^2} \times \sqrt{(p_j)^2}} \tag{10}$$

where p_i and p_j are equivalent to any two categories c_i and c_j, respectively, whose representation can be calculated using Eq. (7), and s_i and s_j are equivalent to $E_i^{representi\ ve}$ and $E_j^{representive}$, respectively, which are collections of representative entities for these two categories and can be used as inputs of the two MEHC methods in the Siames network. Further, the *margin* is a hyperparameter. During training, the model parameter *W* was updated using the Adam method.

3.2 Reprocessing the coarse-grained classification results

After learning the representation of the category, we reprocessed the coarse-grained classification results using the triClustering algorithm. The triClustering algorithm is similar to *k*-means clustering [30] but with a few differences. It performs fewer comparisons and requires less computation time. In our problem, *k* and the cluster centres are fixed, *k* is set to the number of coarse-grained categories, and each cluster centre is set as the representation of the corresponding coarse-grained category. The remaining problem involves calculating the distances between the entities and each cluster centre. However, if the number of entities is large, running the algorithm is time-consuming. Furthermore, because the sum of the two sides of a triangle is greater than the third, e_j, c_i, and c_m can be considered to form a triangle in a multidimensional space, providing us the following heuristic

Heuristic: If $2D(e_j,c_i) < D(c_i,c_m)$ *then* $D(e_j,c_i) < D(e_j,c_m)$

where $D(e_j, c_i)$ and $D(c_i, c_m)$ represent the distance from entity e_j to cluster centre c_i and the distance from c_i to c_m, respectively.

Considering this heuristic, the following conclusion can be drawn

Conclusion: If $2D(e_j,c_i) < D(c_i,c_m)$ ***, where*** c_m ***is the category point closest to*** c_i***, then***

label $(e_j) = c_i$ ***that is,*** e_j ***is in the same cluster as*** c_i.

Proof.

$$\forall e_j \in E, C = \{c_1, c_2, c_3, \ldots, c_n\}, D(c_i, c_m) = \min_{c \in C} D(c_i, c),$$
$$2D(e_j, c_i) < D(c_i, c_m) \Rightarrow 2D(e_j, c_i) < D(c_i, c_k)(c_k \in C, k = 1,2,3,\ldots,n, k \neq m)$$
$$\Rightarrow D(e_j, c_i) < D(e_j, c_k) \Rightarrow D(e_j, c_i) = \min_{c \in C} D(e_j, c)$$

c_i ***is the closest category point to*** e_j***, thus*** *label* $(e_j) = c_i$.

Our method can be described as follows.

Step 1: Calculate the distance between each cluster centre c_i and the other cluster centres, and then arrange them in order from smallest to largest, that is, $[c_m, c_{m+1}, ... c_n]$.

Step 2: Calculate the distance between each entity e_j and its original category centre c_i. If 2D(e_j, c_i) < D(c_i, c_m), e_j belongs to category c_i, where D(e_j, c_i) and D(e_i, c_m) represent the distances from entity e_j to cluster centre c_i and from c_i to c_m, respectively. Otherwise, calculate the distance between entities e_j and c_m, and then compare 2D(e_j, c_m) with the distance between c_m and the nearest cluster centre c_k till the cluster centre with the smallest distance from e_j is determined; this is the category of e_j.

Step 3: Repeat Step 2 until the categories for all entities are determined.

The triClustering algorithm is shown in Algorithm 1.

```
Algorithm 1 triClustering Algorithm in our work
Input: Entity set $e$ of category $c_i$,category label $c_i$
Output: the real category label of $e_m$ in set $e$
 1: entNum=getEntityNum($e$);
 2: for m=1:entNum do
 3:     ecDistance=getDistance($e_{(}m)$,$c_i$);
 4:     ccDistance=getSmallestDistance($c_i$)
 5:     while ecDistance*2 >ccDistance do
 6:         category=getNearestClass($c_i$)
 7:         ccDistance=getSmallestDistance(category)
 8:     end while
 9:     label=category
10:     i++;
11: end for
```

If an entity does not satisfy condition 2D(e_j, c_i) < D(e_i, c_m) after comparison with all categories, the closest category is chosen as its label. In addition, the Euclidean distance [31] was used for this calculation.

Consequently, we analysed the time complexity of the algorithm. In our problem, k and the class centres were determined; thus, only the time complexity of one iteration clustering was compared with other algorithms. Let n represent the number of entities, and k be the number of categories. The triClustering algorithm comprises two processes: determining the smallest distance of c_i with other categories and determining c_i closest to entity e_i. Thus, to search for the smallest distance of c_i with other categories, the first category c_1 should be compared with other $(k-1)$ categories (namely, $\{c_2, c_3, ..., c_n\}$). However, because the distance between c_1 and c_2 is the same as that between c_2 and c_1, c_2 only needs to be compared with categories $\{c_3, c_4, ..., c_n\}$; that is, in the first part, only the $(k-1)+(k-2)+(k-3)+....+1+0=\frac{k(k-1)}{2}$ distance needs to be calculated. In addition, the distance from each category to the other categories needs to be sorted in ascending order. Consequently, the implementation of a quick sorting algorithm adds time $o((k-1)^2 \log(k-1))$. In the second part, as shown Algorithm 1, in the worst case, the time cost is $o((k-1)n)$. In summary, we achieved a per-iteration total time complexity of $o(\frac{k(k-1)}{2} + k^2 \log(k-1) + (k-1)n)$. The per-iteration time complexities in the worst-case scenarios for the baseline algorithms (Lloyd [30], Hamerly [32], Dualtree_kd [33], Yinyang [34], Annulus [35], Exponion [36], and Ball k-means [37]) in comparison to our triClustering algorithm, is presented in Table 1. Similar to our algorithm, these algorithms need to pre-process data in advance; hence, this portion of time has been ignored in our study.

Table 1: Per-iteration time complexity of models ($1 \le m \le k;\ n' \le n$)

Model	Time costs (in the worst case)
Lloyd	$O(kn)$
Dualtree_kd	$O(k \log k)$
Annulus	$O(k^2 \log(k) + n \log \log(k) + k^2 + kn)$
Exponion	$O(k \log(k) + n \log(k) + k^2 + kn)$
Yinyang	$O(kn)$
Ball k-means	$O(k^2 + km \log m + mn' + n)$
triClustering	$o(\frac{k(k-1)}{2} + k^2 \log(k-1) + (k-1)n)$

Although the Annulus, Exponion, and Ball k-means algorithms are efficient, they have a high time complexity; By contrast, the Dualtree algorithm has a competitive time cost, but its time costs depend on certain assumptions regarding dataset-dependent constants, particularly in high-dimensional datasets; thus, the performance is not satisfactory [37].

3.3 Fine-grained Entity Typing

Following the reclassification of the coarse-grained classification results, the next step is to classify the entities in a fine-grained manner. At this stage, there are no entities in the ontology and no hierarchically labelled entities; therefore, machine learning and neural network methods cannot be applied. Only the hierarchical label of an entity can be determined by comparing its similarity to each subclass of the corresponding category. If the subcategory has maximum similarity with the entity, it is set as the category label of the entity.

In Section 3.1, we represented each subcategory of c_i as all of its parent categories and the category itself; this implies that each category contains more information than its parent category. Hence, when comparing similarities, we move from the lowest to the highest level. The process is as follows.

Step 1: Compare the similarity between the entity and all subclasses at the lowest level to determine the one with the highest similarity, which is set as s_1.

Step 2: Compare the similarity between the entity and parent of the subclass, which is set as s_2. Subsequently, compare s_1 and s_2: if s_1 is larger than s_2, the subclass is the category of the entity; otherwise, the value of s_2 is assigned to s_1. Thereafter, clear s_2, and repeat step 2.

Step 3: Repeat until all entities are matched with a corresponding subcategory label.

This process is shown in Algorithm 2.

Algorithm 2 Fine-grained Entity typing Method

```
Input: Entity $e_i$, category set $c$ of the deepest layer in UCO
Output: the real category label of $e_i$ in set
 1: cateNum=getCategoryNum($c$);
 2: for m=1;cateNum do
 3:     ecSimilarity=getSimilarity($e_{(i)}, c_m$);
 4:     ccSimilarity=getBiggetSimilarity($c_j$)
 5: end for
 6: category set $c$=getHierarchies($c_j$)
 7: cateNum2=getCategoryNum($c$);
 8: for m=1;cateNum2 do
 9:     ecSimilarity2=getSimilarity($e_{(i)}, c_m$);
10:     if ccSimilarity<ecSimilarity2 then
11:         label=$c_m$;
12:         break;
13:     else
14:         ccSimilarity=ecSimilarity2
15:     end if
16: end for
17: label=category
```

Here, the cosine similarity [38] was used to calculate the similarity.

4. Experiments and Results

We intend to prove the following four aspects in our experiments. First, the proposed hybrid method can perform better than other baseline models for zero-/few-shot fine-grained entity typing in the information security domain. Second, the representation method of MEHC is better than the general category representation method. Third, the triClustering algorithm is more efficient than the other *k*-means clustering methods. Finally, prepossessing coarse-grained classification results can improve the performance of fine-grained entity typing.

4.1 Data Sources and Experimental Settings

In our experiments, we used two datasets owing to their different purposes. One, mainly drawn from the information security field, was collected in our previous study and includes related blog posts (such as WeLiveSecurity and Threatpost), descriptions from common vulnerabilities and exposures, Microsoft security bulletins, and information security abstracts. In our previous study, we extracted approximately 5,000 entities from this corpus, which can be classified into eight categories and serve as our main experimental data for validating the triClustering algorithm and fine-grained entity typing tasks. Moreover, to train the MEHC method, another dataset called the paraphrase database [39] comprising tens of millions of automatically extracted paraphrase pairs (synonymous phrase pairs), including words and phrases was used. 2,000 paraphrase pairs were extracted with the highest accuracy. Using the provided search engine (http://paraphrase.org), a semantically similar phrase set corresponding to each phrase in the paraphrase pair was obtained; for instance, for a paraphrase pair delighted and pleased, their corresponding phrase sets are {absolutely delighted, so grateful, so thrilled,....} and {so pleased, so gratifying, so happy about,....}, respectively. These are the positive samples of the training MEHC method, while the negative samples are semantically dissimilar paraphrase pairs composed of any combination of phrases selected from the positive samples.

The word vectors used in the experiment were trained using Word2vec having dimension as 300. Following [40], out-of-vocabulary words are hashed to 1 of 128 random embeddings, which are initialised using a uniform distribution between [−0.05, 0.05].

Further, when we trained the MEHC model, the mini-batch size was set at 25, and a learning rate of 10^{-3} was used to update the model parameters. Moreover, the dropout technique was used to avoid overfitting, with a dropout value of 0.3.

Our method is implemented in Java, Python, and Groovy using Eclipse, PyCharm, and IntelliJ IDEA on a Surface Pro 6 with an i7 CPU and 16 GB memory. We ran all experiments five times and reported the mean values.

4.2 Baselines

In our experiments, we used four baselines groups. The first and second groups are compared with the zero-shot fine-grained named typing method proposed in this study, and the advantages of our method were verified by performing zero-/few-shot experiments. Further, the third group was used for comparison with the MEHC method, while the last was used for comparison with the triClustering algorithm. The specific details of each group are as follows.

In the first group, four baselines were used: 1) the method called Zoe, (source code provided in [21]). 2) The method in [23] is called ProtoLE, which uses representative entities and hierarchical information to learn pre-trained label embeddings. However, it captures the embedding of a label based on the average of the sum of each embedding of the words that constitute the label. 3) The method in [41], known as NZET, also starts by learning from the representation of entities and categories and then transfers the knowledge from seen entity types to zero-shot ones by modelling the relationship between entities and categories. However, it incorporates character-level, word-level, and contextual-level information to learn the entity mention representation, which was ignored in the experiment, as there is no contextual information in our dataset. 4) The work in [42], known as DZET, proposes a zero-shot entity typing approach that utilises the type description available from Wikipedia to build a distributed semantic representation of the types. Subsequently, modelling is performed by aligning the entities with known types, and a new type can be incorporated into the model, given its Wikipedia descriptions, to realise zero-shot classification.

In the second group, two baseline models are used: the first method is reported in [22] and the second in [23]. The authors in [22] proposed a framework, referred to as auto fine-grained entity typing (AFET), that projects labels in high-dimensional space, uses label correlations to better model their relationships, and then predicts the type path mentioned by the entity. The source code for this model has also been provided in [22].

In the third group, two methods were used as baselines: the first obtains the representation of the category directly from Word2vec, which is referred to here as a representation of Word2vec, whereas the second obtains a representation of the category as follows

$$e_i = \frac{1}{n}\sum_{i}^{n} w_i \qquad (10)$$

$$c_i = \frac{1}{k}\sum_{i}^{k} e_i \qquad (11)$$

This is similar to the method in [23]. For convenience, we refer to this method as the average number of words.

Finally, the last group comprises several clustering algorithms, including Lloyd [30], Hamerly [32], Dualtree_kd [33], Yinyang [34], Annulus [35], Exponion [36] and Ball k-means [37]. The details of these algorithms are discussed in the previous section and will not be repeated here.

4.3 Experimental Results and Analysis

We evaluated our method and baseline methods using precision (*P*), recall (*R*), and F-measure (*F*).

4.3.1 Comparison with fine-grained entity typing baselines on dataset of information security

In this section, we evaluate the proposed method of fine-grained entity typing from the perspective of both zero- and few-shot schemes. We conducted two experiments to demonstrate the superiority of the proposed method. In the first experiment, we used the results of the coarse-grained classification as the source data for zero-shot fine-grained entity typing. In this experiment, the ‘Consequence’ and ‘Means’ classes were chosen as our experimental data. There are 625 entities in the ‘Consequence’ class and 520 entities in the ‘Means’ class.

The coarse-grained entities serve as inputs to our method and the third group of baselines. Table 2 presents the results of the zero-shot fine-grained entity typing.

Table 2: Results of zero-shot fine-grained entity typing

Method	***P***	***R***	***F***
Zoe	70.1%	71.9%	70.9%
ProtoLE	69.1%	70%	69.5%
NZET	70.3%	69.2%	69.7%
DZET	72.1%	72.4%	72.2%
Our proposed method	**75.2%**	**76.7%**	**75.9%**

Table 2 shows that our method performed better than the baselines. The main difference between our approach and ProtoLE lies in the method of obtaining vectors of entities and labels. The proposed approach uses a feed-forward neural network to learn the features of the entities and labels, whereas ProtoLE obtains the average of the sum of word embeddings. The reason for the better performance by our method compared to ProtoLE is discussed in the subsequent experiment, where we verified the performance of the MEHC scheme. We focus on the proposed method and Zoe in the remainder of this section. An analysis of the experimental results provides two main reasons for the unsatisfactory performance of Zoe in this task. First, Zoe focuses on the representations of entity mentions rather than on the category representations, which represent the embedding of the entity vector. It first identifies the sentence wherein the entity appears via WikiLinks (a public labelled corpus), and then selects the context to represent the entity. However, in the information security field, many entities lack corresponding terms in Wikipedia, such as ‘net flood’ or ‘memory consumption’. Thus, the advantage of using Zoe is not clear. Furthermore, fine-grained classification was conducted in Zoe by calculating the probability of the entity appearing in the WikiLinks for a certain subclass. However, for UCO, all subclasses are not included in Wikipedia; for example, ‘LossOfConf’ cannot be split into individual words and is less likely to appear in Wikipedia. Therefore, Zoe is more suitable for tasks in the general domain than for those in information security. Both NZET and DZET predict the unknown category of an entity through the known category of the entity. The coarse-grained category of the entity is regarded as the known category, while the fine-grained category is regarded as the unknown category. Because the core idea of DZET involves learning the semantic representation of categories through Wikipedia, the reason for its poor performance is similar to that of Zoe; hence, it is not repeated here. However, the reasons for the unsatisfactory performance of NZET are as follows. 1) In this experiment, owing to a lack of context, the context representation part of the model was removed, which has been aggregated into the final representation of the mention to guide the classification. 2) The model constructs the association space between known and unknown categories by calculating the similarity between them and then realises the transformation from known knowledge to unknown knowledge through the association space. However, in our experiment, the known categories are coarse-grained, such as ‘Consequence’, while the unknown ones are lower-level categories, such as ‘Consequence/Remote Access’ and ‘Consequence/botnet Attack’. As their similarity to ‘Consequence’ is negligible, the parts that play an important role in the model almost lose their effectiveness.

In the second experiment, we randomly selected 100 entities from each category as seeds (training set) and mixed the remaining entities after removing labels (test set). The results of our method and the fourth group of baselines are listed in Table 3.

Table 3: Results for few-shot fine-grained entity typing

Method	***P***	***R***	***F***
AFET	69.7%	66.4%	68.0%
ProtoLE	67.9%	65.3%	66.6%
Proposed method	**72.2%**	**73.4%**	**72.8%**

As shown in Table 3, our model exhibits a slight advantage over AFET and ProtoLE in terms of performance, and the values for *P*, *R*, and *F* are higher than those for AFET by 2.5, 7, and 4.8 %, respectively. The reasons for this are similar to those described by Zoe. AFET also uses distant supervision to obtain candidate types for each mention, and these mentions are partitioned into a 'clean' set and a 'noisy' set, based on the given type hierarchy. Further, as discussed above, owing to the expertise required in the information security domain, many entities do not exist in the knowledge base that was used, which is also the reason why the performance of this method is not high in the field of information security.

4.3.2 Comparison of the MEHC method

In this section, we demonstrate the superiority of the proposed MEHC method over other baseline models in terms of re-clustering the coarse-grained classification results and populating UCO with the entities after re-clustering.

First, we selected the representative entities using NPMI to represent the corresponding category embedding. Table 4 lists the examples of certain categories and their corresponding representative entities from the results of coarse-grained classification.

Table 4: Examples of categories and their corresponding representative entities

Category	Representative entities
Consequence	denial of service; execute arbitrary code; inject arbitrary web script
Means	site scripting; unspecified vectors; crafted website; buffer overflow
Attack	memory corruption; code execution; multi-collision attack
Software	adobe reader x; oracle java se 7; windows; unix
Hardware	Cpu; nexus 7000; mac
Network	ssl; firewall; http; tls

File_Name	Php,cji;libavcodec;flash file
Modifier	before 83 HP2; through update 38; after SP3; and SP1

Next, we determined the top *k* entities. For clustering, 3,000 out of 5,000 entities were randomly selected. Figure 4 shows the performance of the triClustering algorithm for various sizes of *k*.

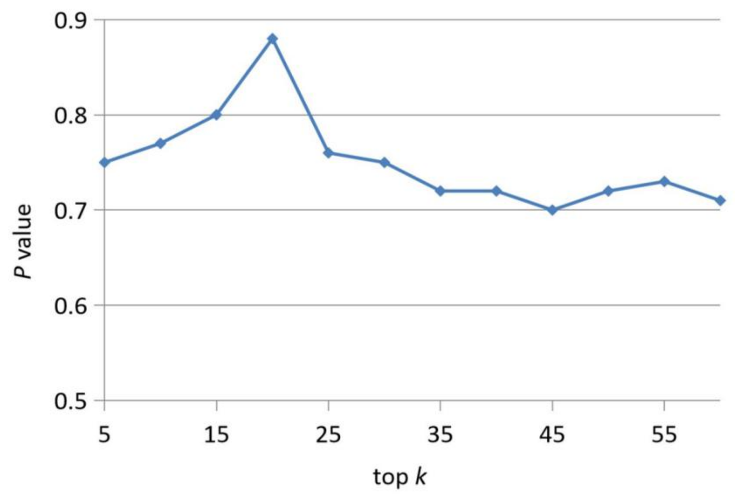


Figure 4: Performance for various sizes of *k*

Through experiments, it was found that when the top *k* was 20, the model performance was the best. To distinguish it from the latter experiment, we refer to it as the MEHC of representative entities. The results of MEHC representative entities compared with the third group of baselines are listed in Table 5.

Table 5: Clustering results from different representation methods

Methods	***P***	***R***	***F***
MEHC of representative entities	**88.1%**	**90.1%**	**89.5%**
Average of words	83.2%	80.1%	81.6%
Representation of Word2vec	80.4%	77.3%	78.8%

As shown in Table 5, MEHC of representative entities was more effective than the other two methods. Compared with representation of Word2vec, which showed the worst performance, the values of *P*, *R*, and *F* of our method were significantly higher by 7.7, 12.8, and 10.7 %, respectively. One reason is that certain categories lack domain specialisation, such as the ‘Consequence’ and ‘Means’ classes. Moreover, in Word2vec, the word vector possesses no domain characteristics, indicating a possibility of mistakes occurring during clustering. After the analysis of misclassified entities, we found that these two categories indeed had the highest clustering errors. The second reason is that certain categories in UCO contain abbreviations, such as ‘LossOfConf’, which is a subclass of the ‘Consequence’ class. Further, ‘LossOfConf’ can be divided into three words: ‘loss’, ‘of’, and ‘conf’. However, ‘conf’ does not appear in the word list. Consequently, when Word2vec is used to obtain the category representation directly, it inevitably results in a loss of features, thus affecting the classification results.

In addition, the above-mentioned reasons also clarify why the average of words method is slightly better than the representation of the Word2vec method. Further, Table 5 also shows that it is feasible to obtain the maximum features of the representative entities using MEHC and combine them into the category representation.

In the MEHC method, a hierarchical category in UCO is represented by its parent categories as well as itself when fine-grained classification is applied. Furthermore, in UCO, certain categories have complex hierarchies, such as the ‘Consequence’ and ‘Means’ classes. Fig. 5 illustrates the hierarchies of the ‘Consequence’ class.

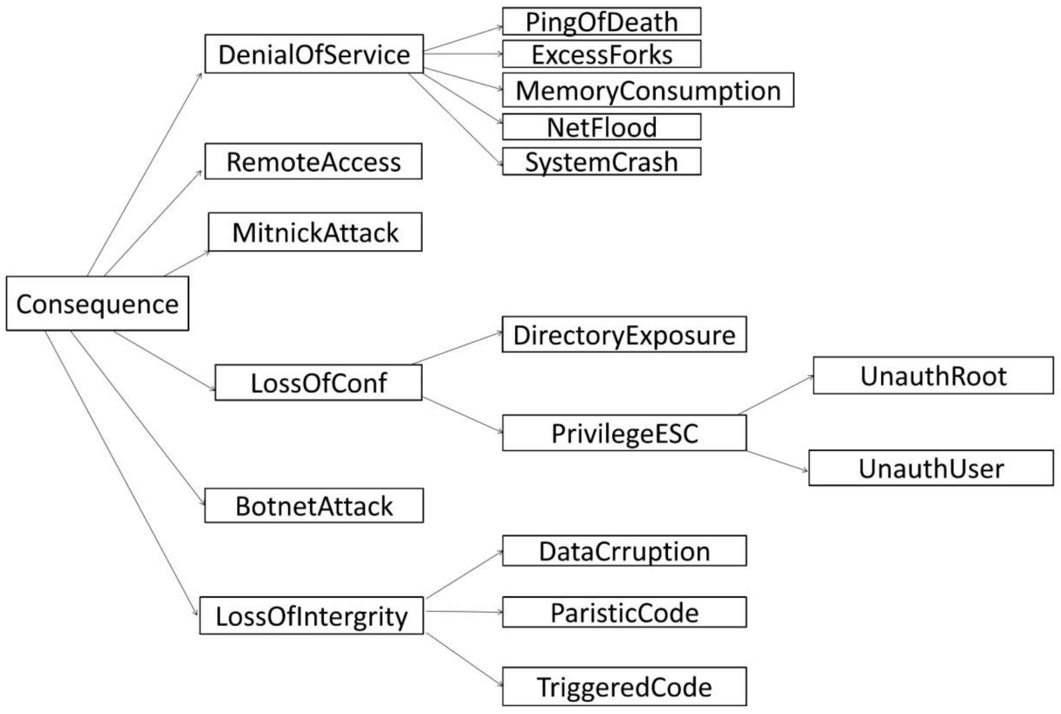


Figure 5: Hierarchies of the ‘Consequence’ class in UCO

As evident from Fig. 5, the subclass ‘UnauthUser’ has parent classes ‘PrivilegeESC’, ‘LossOfConf’, and ‘Consequence’. This provides us with the representation of the ‘UnauthUser’ class in MEHC. However, to distinguish it from the MEHC of representative entities in the above experiment, we refer to it as the MEHC of the hierarchy.

Further, we utilised the results of clustering 3,000 entities of the above experiment as the source data of this experiment and populated them into UCO using the same two baselines as mentioned above. The results are presented in Table 6.

Table 6: Comparison of results from different category representation methods

Methods	*P*	*R*	*F*
MEHC of hierarchy	**79.2%**	**78.9%**	**79.0%**
Average of words	71.3%	70.6%	70.9%
Representation of word2vec	65.5%	69.1%	67.2%

As evident from Table 6, the MEHC of the hierarchy was more effective than the other two methods. Compared with the representation of the Word2vec method, which showed the worst performance, the values of *P*, *R*, and *F* of our method were significantly higher by 13.7, 9.8, and 11.8 %, respectively. There are two reasons for this result. First, we used the results of the above quadratic clustering as the input for this experiment. In the previous experiment, the results from the MEHC of representative entities were superior to those of the other two baselines. Consequently, even in this experiment, the results from the MEHC of representative entities were better than the baseline. Second, certain categories in UCO contained abbreviations, such as 'LossOfConf', which is a subclass of the 'Consequence' class. 'LossOfConf' can be divided into three words: 'loss', 'of', and 'conf'. However, 'conf' does not appear in the word list. Consequently, when Word2vec is used to obtain the category representation directly, it results in a loss of features, thus affecting the classification results.

4.3.3 Comparison of clustering algorithms

In this experiment, we first selected 1000 entities from our dataset and used a dimension reduction technique called PCA to visualise the vector representation of categories and entities. The clustering results using the triClustering algorithm are shown in Fig. 6.

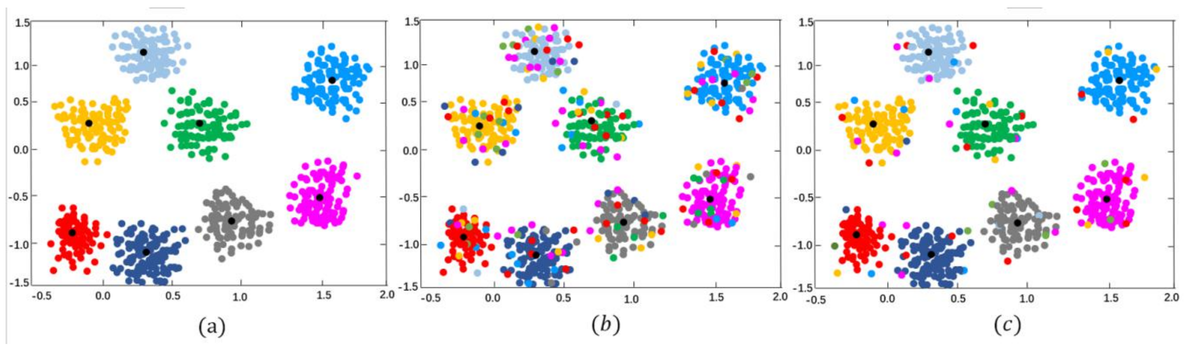


Fig. 6: Different clustering results. (a) ground_truth category distribution; (b) category distribution before triClustering algorithm;(c) category distribution after triClustering algorithm.

As shown in Fig. 6, although the clustering results with the triClustering algorithm are different from the ground truth category distribution (as shown in Fig. 6(a)), they have been improved when compared with the category distribution without the triClustering algorithm (as shown in Fig. 6(b)). Therefore, it can be proven that the proposed triClustering algorithm is effective and can have a positive impact on fine-grained entity typing task.

Further, we compared the triClustering algorithm with the second group of baselines in terms of running time. For consistency, these algorithms were run on the same dataset and all used the category label as the initial cluster centre. Subsequently, we divided the 5000 entities into five batches, observed the running time of the algorithms as the number of entities was increased, and measured the running time using *currentTimeMillis*() in Java. The running time is devoid of the time required to train the feed-forward neural network for MEHC or to obtain the representations of the entities and categories from the model. In addition, we compared the elapsed time for only one iteration. Fig. 7 shows the change in the running time with an increase in the dataset.

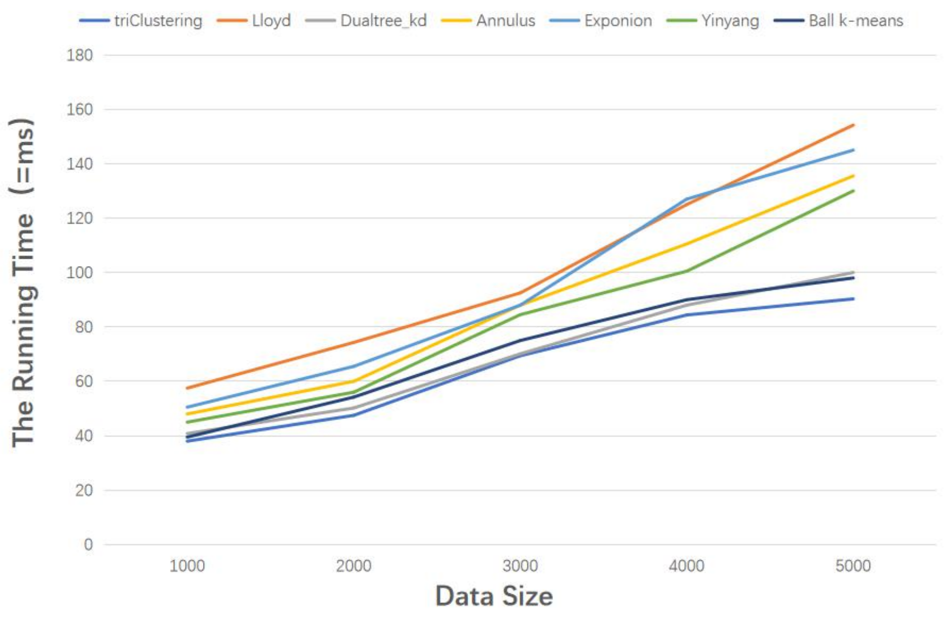


Figure 7: Running times of the proposed algorithm and the k-means algorithm

As evident from Fig. 7, both algorithms have running times that vary linearly with an increase in the dataset, with the proposed algorithm yielding the best performance. This is because, in Hamerly, Yinyang, Annulus, and Exponion algorithms, the upper and lower boundaries need to be calculated. Moreover, although there is no need for ball k-means to calculate these boundaries, each cluster must be divided into stable and active regions by calculating the distance from the centres. Therefore, in addition to the direct distance comparison, it is necessary to compare the point with the radius of each ball to determine the area wherein the point will fall. Furthermore, although Dualtree_kd performs well on low-dimensional data, it performs poorly on discretely distributed high-dimensional data.

4.3.4 Role of pre-processing of the coarse-grained classification

As shown in Table 5, the MEHC method can optimise the results of the coarse-grained classification.

Consequently, an increase in the accuracy of coarse-grained classification improves the accuracy of fine-grained named entity typing and further improves the performance of the entities that populate the UCO. Three thousand entities of the above experiment were utilised as the source data of this experiment, and Fig. 8 shows the results for these entities populating UCO in both cases, that is, with and without pre-processing of coarse-grained classification.

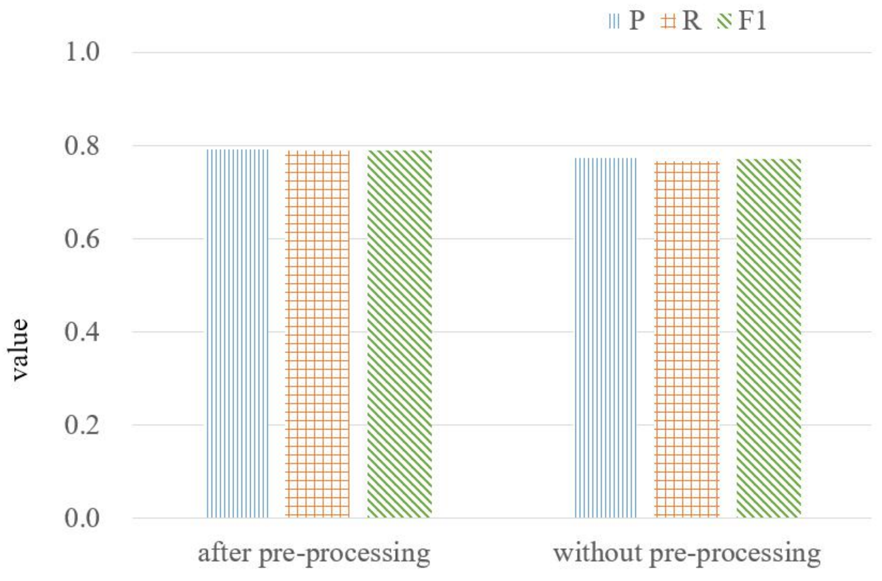


Figure 8: Results for *P*, *R*, and *F* values when populating the entities into UCO with and without preprocessing.

As evident from Fig. 8, the results obtained after pre-processing are better than those without pre-processing, where the values of *P*, *R*, and *F* after pre-processing are 79.2, 78.9, and 79%, respectively, and those without pre-processing are 77.3, 76.7, and 77.0 %, respectively. However, the differences are not significant and are 1.9, 2.2, and 2 %, respectively. Although the gap between the two cases is smaller, it indicates that the performance of the method can be further improved after pre-processing. In addition, as shown in Fig. 1, the core of the pre-processing is the triClustering algorithm. As shown in Fig. 7, the triClustering algorithm requires only 90 ms to cluster 5000 entities, demonstrating its very low time consumption. Therefore, it is necessary to perform pre-processing.

4.4 Discussion

In our experiments, first, we compared our proposed method with Zoe, ProtoLE, NZET, DZET, and AFET, which were used to perform zero- and few-shot fine-grained entity typing. ProtoLE focuses on learning the embeddings of the labels and ignores the structural features of phrases that represent entities and category labels. By contrast, the other four models are designed for general applications and do not consider certain special cases in the field of information security and UCO, and thus provide unsatisfactory performance.

Second, we compared our MEHC model with two baselines. The first was the typical method of obtaining the category representation directly from Word2vec, whereas the second method involved obtaining the category representation by averaging the words that comprise the representative entities. From our results for fine-grained classification, it can be observed that the *P* and *R* values of the proposed MEHC model are higher than those of the other two baselines.

Third, we re-clustered the coarse-grained classification results to improve the accuracy of the coarse-grained classification results by proposing the triClustering algorithm, which is based on the tripartite theorem of the triangle. The proposed method was compared with other *k*-means clustering algorithms in an experiment and it was found that the triClustering algorithm is superior to other clustering algorithms in terms of running time (Fig. 7). When the data sizes are 1k, 2k, 3k, 4k, and 5k, the running times of these methods increase linearly; however, the running time for the triClustering algorithm is always lower than that of the other algorithms. In certain clustering algorithms, such as those proposed in [33,43], when the vector dimension is small, the performance takes precedence over k-means clustering. However, if the dimension is larger than 100, the performance declines significantly. In our experiment, although the dimension of the word vector was as large as 300, the results did not decline, proving that our hypothesis is still valid for high dimensions. In addition, the accuracy of the fine-grained classification results after re-clustering was better (Fig. 8) than that of direct fine-grained classification results.

5. Conclusions

This study proposed a hybrid computational approach that combined a clustering algorithm with a category representation for zero-shot fine-grained entity typing in information security. Based on the particularities of UCO and the information security field, we proposed a new category representation method called MEHC, which represented the category by selecting the representative entities under it. Moreover, the parent categories were used to represent each subcategory by employing the hierarchy of UCO.

In addition, to improve the accuracy of fine-grained classification results, we proposed the triClustering algorithm, which re-clusters the coarse-grained classification results using the tripartite theorem of a triangle in which the sum of the two sides of a triangle is greater than the third. The results showed that the triClustering algorithm could effectively shorten the running time and that re-clustering the results of coarse-grained classification effectively improved the results of subsequent fine-grained classification. Finally, we verified our method using Zoe, ProtoLE NZET, DZET, and AFET on datasets in the information security field. The experimental results showed that the proposed hybrid method can achieve satisfactory results in terms of the filling of UCO with entities.

The triClustering method proposed in this study fixes the centre and the value of *k*, which is not sufficiently flexible. We are currently in the process of improving this method to optimise the initial centre selection of the cluster; this method will be applied to public datasets. In addition, we expect to further expand the simple and limited knowledge base in the field of information security obtained through this study and apply it to other systems, such as question answering systems [44] and text classification [45].

Acknowledgement

This work was supported by Major Science and Technology Project in Henan Province (grant No.201300210500).

References

[1] Efrén Rama-Maneiro, Juan C. Vidal, Manuel Lama,Collective disambiguation in entity linking based on topic coherence in semantic graphs, Knowledge-Based Systems,Volume 199,2020,105967,ISSN 0950-7051.

[2] Onoe Y, Durrett G. Fine-grained entity typing for domain independent entity linking[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2020, 34(05): 8576-8583.

[3] Jiangying Zhang, Kuangrong Hao, Xue-song Tang, Xin Cai, Yan Xiao, Tong Wang,A multi-feature fusion model for Chinese relation extraction with entity sense,Knowledge-Based Systems,Volume 206,2020,106348, ISSN 0950-7051.

[4] Nayak T, Ng H T. Effective modeling of encoder-decoder architecture for joint entity and relation extraction[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2020, 34(05): 8528-8535.

[5] Malaviya C, Bhagavatula C, Bosselut A, et al. Commonsense knowledge base completion with structural and semantic context[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2020, 34(03): 2925-2933.

[6] Zhou J, Lv X, Yang C, et al. KACC: A Multi-task Benchmark for Knowledge Abstraction, Concretization and Completion[J]. arXiv preprint arXiv:2004.13631, 2020.

[7] Syed Z, Padia A, Finin T, et al. UCO: A unified cybersecurity ontology[C]//Workshop at the 30th AAAI Conference on Artificial Intelligence. 2016, pp. 195-202.

[8] Zhang H, Guo Y, Li T. Multifeature named entity recognition in information security based on adversarial learning[J]. Security and Communication Networks, 2019.

[9] Ali M A, Sun Y, Li B, et al. Fine-Grained Named Entity Typing over Distantly Supervised Data via Refinement in Hyperbolic Space[J]. arXiv preprint arXiv:2101.11212, 2021.

[10] Ali M A, Sun Y, Li B, et al. Fine-grained named entity typing over distantly supervised data based on refined representations[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2020, 34(05): 7391-7398..

[11] Nayak N V, Bach S H. Zero-Shot Learning with Common Sense Knowledge Graphs[J]. arXiv preprint arXiv:2006.10713, 2020.

[12] Zhang T, Xia C, Lu C T, et al. MZET: Memory Augmented Zero-Shot Fine-grained Named Entity Typing[J]. arXiv preprint arXiv:2004.01267, 2020.

[13] Ren X, El-Kishky A, Wang C, et al. Clustype: Effective entity recognition and typing by relation phrase-based

clustering[C]//Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015, pp. 995-1004.

[14] Gangemi A, Nuzzolese A G, Presutti V, et al. Automatic typing of DBpedia entities[C]//International Semantic Web Conference. Springer, Berlin, Heidelberg, 2012, pp. 65-81.

[15] Ling X, Weld D S. Fine-grained entity recognition[C]//26th AAAI Conference on Artificial Intelligence. 2012, pp. 94-100.

[16] Yuan Z, Downey D. OTyper: A neural architecture for open named entity typing. AAAI, 2018, pp. 6037-6044.

[17] Mengge X, Yu B, Zhang Z, et al. Coarse-to-Fine Pre-training for Named Entity Recognition[C]//Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2020: 6345-6354.

[18] Yaghoobzadeh Y, Schütze H. Corpus-level fine-grained entity typing using contextual information[J]. arXiv preprint arXiv:1606.07901, 2016.

[19] Xu P, Barbosa D. Neural fine-grained entity type classification with hierarchy-aware loss[J]. arXiv preprint arXiv:1803.03378, 2018.

[20] Zhou B, Khashabi D, Tsai C T, et al. Zero-shot open entity typing as type-compatible grounding[C]//Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. 2018, pp. 2065-2076.

[21] Huang L, May J, Pan X, et al. Building a fine-grained entity typing system overnight for a new x (x= language, domain, genre)[J]. arXiv preprint arXiv:1603.03112, 2016.

[22] Ren X, He W, Qu M, et al. Afet: Automatic fine-grained entity typing by hierarchical partial-label embedding[C]//Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. 2016, pp. 1369-1378.

[23] Ma Y, Cambria E, Gao S. Label embedding for zero-shot fine-grained named entity typing[C]//Proceedings of COLING 2016, 26th International Conference on Computational Linguistics: Technical Papers. 2016, pp. 171-180.

[24] Dai Z, Huang R. Building context-aware clause representations for situation entity type classification [J]. arXiv preprint arXiv:1809.07483, 2018.

[25] Onoe Y, Boratko M, Durrett G. Modeling Fine-Grained Entity Types with Box Embeddings[J]. arXiv preprint arXiv:2101.00345, 2021.

[26] Zhong X, Cambria E. Time expression recognition using a constituent-based tagging scheme[C]//Proceedings of the 2018 World Wide Web Conference. 2018: 983-992.

[27] Luo Y, Xiao F, Zhao H. Hierarchical contextualized representation for named entity recognition[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2020, 34(05): 8441-8448.

[28] Bouma G. Normalized (pointwise) mutual information in collocation extraction[J]. Proceedings of GSCL, 2009: 31-40.

[29] Wang J, Zhu Z, Li J, et al. Attention based siamese networks for few-shot learning[C]//2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS). IEEE, 2018: 551-554.

[30] S. P. Lloyd, Least Squares Quantization in PCM,º IEEE Trans. Information Theory, vol. 28, 129-137, 1982.

[31] Danielsson P E. Euclidean distance mapping[J]. Computer Graphics and Image Processing, 1980, 14(3): 227-248.

[32] Greg Hamerly. Making k-means even faster. In Proceedings of the 2010 SIAM international conference on data mining, pages 130–140. SIAM, 2010.

[33] Ryan R. Curtin. A dual-tree algorithm for fast k-means clustering with large k. In Proceedings of the 2017 SIAM International Conference on Data Mining, pages 300–308. SIAM, 2017.

[34] Yufei Ding, Yue Zhao, Xipeng Shen, Madanlal Musuvathi, and Todd Mytkowicz. Yinyang k-means: A drop-in replacement of the classic k-means with consistent speedup. In International Conference on Machine Learning, pages 579–587, 2015.

[35] Drake, Jonathan. Faster k-means clustering. , 2013.

[36] James Newling and Franc¸ois Fleuret. Fast k-means with accurate bounds. In International Conference on Machine Learning, pages 936– 944, 2016.

[37] Xia S, Peng D, Meng D, et al. Ball k-means[J]. arXiv preprint arXiv:2005.00784, 2020.

[38] Steinbach M, Karypis G, Kumar V. A comparison of document clustering techniques[C]// KDD workshop on text mining. 2000, 400(1): 525-526.

[39] Ganitkevitch J, Van Durme B, Callison-Burch C. PPDB: The paraphrase database[C]//Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2013, pp. 758-764.

[40] Parikh A P, Täckström O, Das D, et al. A decomposable attention model for natural language inference[J]. arXiv preprint arXiv:1606.01933, 2016.

[41] Zhang T , Xia C , Lu C T , et al. MZET: Memory Augmented Zero-Shot Fine-grained Named Entity Typing[J]. 2020.

[42] Rasha Obeidat, Xiaoli Fern, Hamed Shahbazi, and Prasad Tadepalli. 2019. Description-based zero-shot fine

grained entity typing. In NAACL, pages 807–814.

[43] Kanungo T, Mount D M, Netanyahu N S, et al. An efficient k-means clustering algorithm: Analysis and implementation[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2002 (7): 881-892.

[44] Zhao W, Peng H, Eger S, et al. Towards scalable and reliable capsule networks for challenging NLP applications[J]. arXiv preprint arXiv:1906.02829, 2019.

[45] Minaee S, Kalchbrenner N, Cambria E, et al. Deep Learning--based Text Classification: A Comprehensive Review[J]. ACM Computing Surveys (CSUR), 2021, 54(3): 1-40.