Software Impacts 8 (2021) 100069

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

An open-source framework for ExpFinder integrating N-gram vector space model and μCO-HITS

Hung Du [a], Yong-Bin Kang [b,*]

[a] Department of Computer Science and Software Engineering, Swinburne University of Technology, Australia
[b] Department of Media and Communication, Swinburne University of Technology, Australia

ARTICLE INFO

Keywords:
ExpFinder
Expert finding
N-gram vector space model
μCO-HITS
Expert collaboration graph

ABSTRACT

Finding experts drives successful collaborations and high-quality product development in academic and research domains. To contribute to the expert finding research community, we have developed ExpFinder which is a novel ensemble model for expert finding by integrating an N-gram vector space model (nVSM) and a graph-based model (μCO-HITS). This paper provides descriptions of ExpFinder's architecture, key components, functionalities, and illustrative examples. ExpFinder is an effective and competitive model for expert finding, significantly outperforming a number of expert finding models as presented in Kang et al. (2021).

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2021-18
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/9243982/tree/v1
Legal Code License	MIT License (MIT)
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Python environment version 3.6 or above, pandas, networkx, NumPy, scikit-learn, nltk, SciPy, Torch, Transformers, SciBERT
Link to developer documentation/manual	https://github.com/Yongbinkang/ExpFinder/blob/main/README.md
Support email for questions	ykang@swin.edu.au, hungdu@swin.edu.au

1. Introduction

Identifying experts given a query topic, known as *expert finding*, is a crucial task that accelerates rapid team formation for research innovations or business growth. Existing expert finding models can be classified into three categories such as *vector space models* (VSM) [1,2], *document language models* (DLM) [3–5], or *graph-based models* (GM) [6–8]. ExpFinder [9] is an ensemble model for expert finding which integrates a novel N-gram VSM (nVSM) with a GM (μCO-HITS)-a variant of the generalized CO-HITS algorithm [6].

As seen in Fig. 1, ExpFinder has nVSM, a vector space model, as a key component that estimates the weight of an expert and a document given a topic by leveraging the Inverse Document Frequency (IDF) weighting [10] for N-gram words (simply N-grams). Another key component in ExpFinder is μCO-HITS that is used to reinforce the weights of experts and documents given a topic in nVSM using an Expert Collaboration Graph (ECG) that is a certain form of an expert social network. The output of ExpFinder is the reinforced weights of experts given topics.

ExpFinder is designed and developed to improve the performance for expert finding. In this paper, we highlight two main contributions to the expert finding community. First, we provide a comprehensive implementation detail of all steps taken in ExpFinder. It could also be used as an implementation guideline for developing various DLM-, VSM- and GM-based expert finding approaches. Second, we illustrate how ExpFinder works with a simple example, thus researchers and practitioners can easily understand ExpFinder's design and implementation.

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

* Corresponding author.
E-mail addresses: hungdu@swin.edu.au (H. Du), ykang@swin.edu.au (Y.-B. Kang).

https://doi.org/10.1016/j.simpa.2021.100069
Received 1 March 2021; Received in revised form 14 March 2021; Accepted 19 March 2021

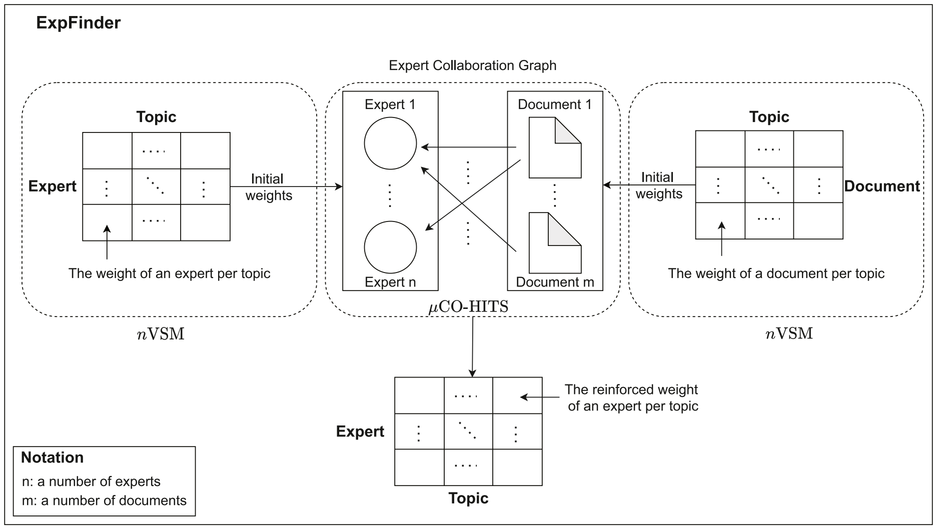


Fig. 1. The overview of ExpFinder.

This paper is organized as follows. Section 2 describes ExpFinder's architecture and functionalities. Section 3 demonstrates the procedural steps in ExpFinder. Section 4 provides the impact and conclusion of ExpFinder.

2. Functionality

ExpFinder is implemented in Python (version $\geq$ 3.6) with open-source libraries such as `pandas`, `NumPy`, `scikit-learn`, `SciPy`, `nltk`, and `networkx`. In this section, we present its architecture, key components, and their functionalities. The architecture of ExpFinder is presented in Fig. 2 that consists of four key steps with the corresponding functions and their functional dependencies:

1. **Step 1 - Extract tokens and topics**: Given an expertise source $\mathcal{D}$ (e.g., scientific publications) of experts $\mathcal{X}$, we extract expertise topics by using `tokenise_doc()` in **extractor.py**. We assume that expertise topics are represented in the forms of noun phrases. For each document $d \in \mathcal{D}$, the function splits it into sentences. Then, for each sentence, the function removes stopwords, assigns a part of speech (POS) to each word, merges the inflected forms of a word (i.e., the lemmatisation process, for example, 'patients' is lemmatized to 'patient'), and extracts single-word terms (called *tokens*) and topics with a given linguistic pattern. In addition, we use a regular expression (`regex`) in Python to construct a linguistic pattern based on POS that is further used for extracting four different types of topics as shown in Fig. 3. Note that we use `nltk` for performing this process. The output of this step is the list of the tokens and the list of topics for each document $d \in \mathcal{D}$. The set of the all tokens is denoted as $\mathcal{W}$, and the set of the all topics is denoted as $\mathcal{T}$.
2. **Step 2 - Estimate the weights of experts and documents given topics in nVSM**: The process includes four main steps with the corresponding functions in **generator.py**:
   - 2.1 We use `generate_tf()` to estimate the term frequencies (TFs) of $\mathcal{W}$ in each document $d \in \mathcal{D}$. For this estimation, we use `CountVectoerizer` in `scikit-learn`. The output of this function is the $|\mathcal{D}| \times |\mathcal{W}|$ Document-Token matrix (**DTM**) where each entry contains the TF of $w \in \mathcal{W}$ in d.
   - 2.2 We use `generate_dp_matrix()` to estimate the weights of documents given $\mathcal{T}$ in nVSM [9]. The function estimates nTFIDF of each topic $t \in \mathcal{T}$ by integrating the nTF weighting and the nIDF weighting. Intuitively, nTF estimates the frequency of t by averaging TFs of tokens in t where TF of each token is stored in **DTM**. In addition, nIDF [10] is the N-gram IDF weighting method that estimates the log-IDF, $\log \frac{|\mathcal{D}| \times df(t)+1}{df(w_1 \wedge w_2 \wedge \ldots \wedge w_n)^2+1} + 1$, of t where $w_1, \ldots, w_n$ are n-constituent terms in t. The output of this step is the $|\mathcal{D}| \times |\mathcal{T}|$ Document-Phrase matrix (**DPM**) where each entry contains the nTFIDF weight of t in d.
   - 2.3 Given $\mathcal{D}$, we use `generate_ed_matrix()` to generate the $|\mathcal{X}| \times |\mathcal{D}|$ Expert-Document matrix (**EDM**) where each entry shows a binary relationship between $x \in \mathcal{X}$ and d (e.g., 1 indicates that x has the authorship on d, and 0 otherwise).
   - 2.4 We use `generate_pr_matrix()` to estimate the weights of experts $\mathcal{X}$ and documents $\mathcal{D}$ given each topic $t \in \mathcal{T}$ in nVSM [9]. The weights of $\mathcal{X}$ are estimated by calculating matrix multiplication of $\mathbf{EDM}^{|\mathcal{X}| \times |\mathcal{D}|}$ and $\mathbf{DPM}^{|\mathcal{D}| \times |\mathcal{T}|}$ (e.g., **ETopM** = `numpy.matmul(`**EDM**, **DPM**`)` in Python). The output is the $|\mathcal{X}| \times |\mathcal{T}|$ Expert-Topic matrix (**ETopM**) where each entry contains the topic-sensitive weight of x given t. The weights of $\mathcal{D}$ are represented by **DPM**. Now, we denote **DPM** as the $|\mathcal{D}| \times |\mathcal{T}|$ Document-Topic matrix (**DTopM**) where each entry shows the topic-sensitive weight of d given t. It is worth noting that **DPM** can be integrated with another factor (e.g., the average document frequencies of $\mathcal{T}$) to obtain different weights for **DTopM**. However, in our approach, we set **DTopM** = **DPM**.
3. **Step 3 - Construct ECG**. We use `generate_ecg()` in **generator.py** to handle this step. The function receives $\mathcal{D}$ and builds an ECG using `DiGraph` in `networkx` to present a *directed, weighted bipartite graph* that has expert nodes V_x and document nodes V_d. The set of nodes in the graph is denoted as V such that $V = V_x \cup V_d$. A directed edge points from a document node $v_d \in V_d$ to an expert node $v_x \in V_x$ if x has published d. In this step, we also use `generate_ed_vector()` in **generator.py** to generate a $|V| \times 1$ Expert-Count vector ($\boldsymbol{c_x}$) and a $|V| \times 1$ Document-Count ($\boldsymbol{c_d}$) vector based on ECG. These vectors are used for the estimation of μCO-HITS in **Step 4**.
4. **Step 4 - Reinforce expert weights using μCO-HITS**. We use `run_expfinder()` in **trainer.py** to handle this step. The function receives **ETopM**, **DTopM**, ECG, $\boldsymbol{c_x}$ and $\boldsymbol{c_d}$, generated in **(Steps 2 and 3)** as parameters, and reinforces the estimation of expert weights given topics by integrating nVSM and μCO-HITS [9]. For each $t \in \mathcal{T}$, we perform the three steps:

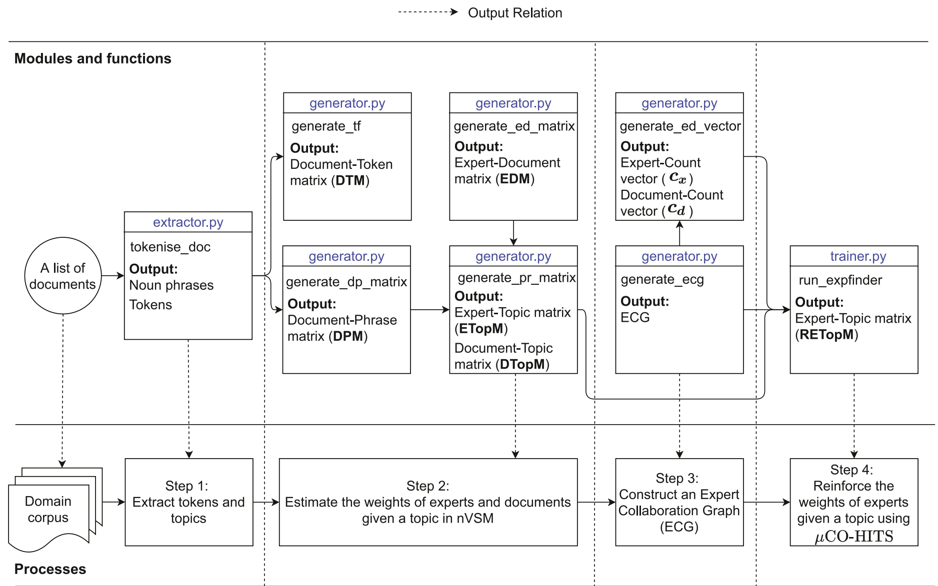


Fig. 2. The architecture and functional workflow of ExpFinder: blue labels indicate module names of ExpFinder, and 'Output Relation' maps the functional component to the corresponding processing step. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.1 **Generate the adjacency matrix of nodes and its transpose** - Given the ECG, we use `to_matrix()` in `networkx` to generate the $|V| \times |V|$ adjacency matrix of the graph $\mathbf{M}$, and also construct its transpose matrix $\mathbf{M}^\top$. These matrices are required in the initialization for running the μCO-HITS algorithm.

4.2 **Normalize the weights of experts and documents given a topic** - We get topic-sensitive weights of $\mathcal{X}$ and $\mathcal{D}$ given t from **ETopM** and **DTopM**, respectively. The output of this includes the $|\mathcal{X}| \times 1$ Expert-Topic ($\boldsymbol{\alpha_x}$) and $|\mathcal{D}| \times 1$ Document-Topic ($\boldsymbol{\alpha_d}$) vectors where each entry shows the topic-sensitive weight of an expert and a document given t, respectively. Then, we normalize these vectors using L2 normalization to scale their squares sum to 1 as the initialization for running the μCO-HITS algorithm [11].

4.3 **Reinforce expert weights given a topic** - We integrate nVSM and μCO-HITS through k iterations to reinforce expert weights given t. μCO-HITS is the extension of the CO-HITS algorithm [6] which contains two main properties such as *average authorities* $\boldsymbol{a}$ and *average hubs* $\boldsymbol{h}$ which show importance of $\mathcal{X}$ and $\mathcal{D}$, respectively, based on the ECG. In addition, these properties can be defined as [9]:

$$\boldsymbol{a}(\mathcal{X};t)^k = (1-\lambda_x)\boldsymbol{a}(\mathcal{X};t)^{k-1} + \lambda_x \left(\frac{\mathbf{M}^\top \cdot \boldsymbol{h}(\mathcal{D};t)^{k-1}}{\boldsymbol{c_d}}\right) \quad (1)$$

$$\boldsymbol{h}(\mathcal{D};t)^k = (1-\lambda_d)\boldsymbol{h}(\mathcal{D};t)^{k-1} + \lambda_d \left(\frac{\mathbf{M} \cdot \boldsymbol{a}(\mathcal{X};t)^{k}}{\boldsymbol{c_x}}\right) \quad (2)$$

where

- $\boldsymbol{a}(\mathcal{X};t)^k$ and $\boldsymbol{h}(\mathcal{D};t)^k$ are $|V|\times 1$ vectors which contain the reinforced expert weights and reinforced document weights, respectively, given t at kth iteration. As the initial weights of these vectors, we use the topic-sensitive weights of experts and documents estimated in nVSM. Thus, $\boldsymbol{a}(\mathcal{X};t)^0 = \boldsymbol{\alpha_x}$ and $\boldsymbol{h}(\mathcal{D};t)^0 = \boldsymbol{\alpha_d}$. By doing so, we integrate nVSM with μCO-HITS. Note that $\boldsymbol{a}(\mathcal{X};t)^0$ is a $|\mathcal{X}|\times 1$ vector, and $\boldsymbol{h}(\mathcal{D};t)^0$ is a $|\mathcal{D}|\times 1$ vector. However, for easily implementing the HITS algorithm, we have transformed the dimension of these vectors into $|V|\times 1$ vectors where additional entries hold the value of 0.

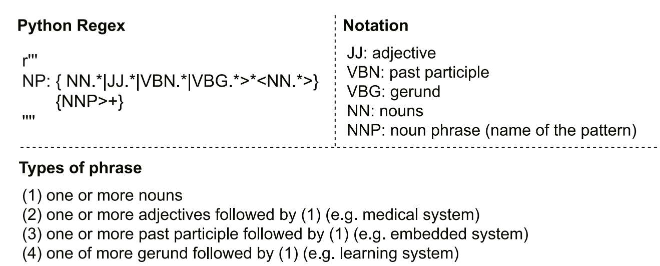


Fig. 3. The Python regular expression of a linguistic pattern for extracting topics in a single document.

- $\lambda_x \in [0,1]$ and $\lambda_d \in [0,1]$ are *parameters* for expert and document, respectively. These are used to control the impact of topic-sensitive weights on $\boldsymbol{a}$ and $\boldsymbol{h}$, respectively. Assigning lower values indicates the higher impact of topic-sensitive weights on $\boldsymbol{a}$ and $\boldsymbol{h}$.
- $\left(\frac{\mathbf{M}^\top \cdot \boldsymbol{h}(\mathcal{D};t)^{k-1}}{\boldsymbol{c_d}}\right)$ is the calculation for the *average authorities*. The numerator performs matrix multiplication between the $|V|\times|V|$ adjacency matrix $\mathbf{M}^\top$ and the $|V| \times 1\boldsymbol{h}$. The denominator is a $|V| \times 1$ counted vector $\boldsymbol{c_d}$ generated in **Step 3**. To calculate this in Python, we simply apply `numpy.matmul(`$\mathbf{M}^\top$, $\boldsymbol{h}(\mathcal{D};t)^{k-1}$`)/`$\boldsymbol{c_d}$.
- $\left(\frac{\mathbf{M} \cdot \boldsymbol{a}(\mathcal{X};t)^{k}}{\boldsymbol{c_x}}\right)$ is the calculation for the *average hubs*. The numerator performs matrix multiplication between the $|V| \times |V|$ adjacency matrix $\mathbf{M}$ and the $|V|\times 1\boldsymbol{a}$. The denominator is a $|V|\times 1$ counted vector $\boldsymbol{c_x}$ generated in **Step 3**. To calculate this in Python, we simply apply `numpy.matmul(`$\mathbf{M}$, $\boldsymbol{a}(\mathcal{X};t)^k$`)/`$\boldsymbol{c_x}$.

After computing $\boldsymbol{a}$ and $\boldsymbol{h}$ at kth iteration, we apply L2 normalization to both $\boldsymbol{a}$ and $\boldsymbol{h}$. We use the obtained $\boldsymbol{a}(\mathcal{X};t)$ after the final iteration to construct the $|\mathcal{X}| \times |\mathcal{T}|$ Expert-Topic matrix (**RETopM**) where each entry contains the reinforced weight of x given t.

Table 1
The document dataset $\mathcal{D}$ used in the example: extracted phrases are highlighted in yellow, and extracted tokens are in bold. (For reference to colour, the reader is referred to the web version of this article.)

Docs	Experts	Text
d_1	x_1, x_2	A **prerequisite** for using **electronic health records** (**EHR**) **data** within **learning health-care system** is an **infrastructure** that **enables** **access** to **EHR** **data** **longitudinally** for **health-care analytics** and **real time** for **knowledge delivery**. Herein, we **share** our **institutional implementation** of a **big data-empowered clinical natural language processing** (**NLP**) **infrastructure**, which not only **enables** **healthcare analytics** but also has **real-time NLP processing capability**.
d_2	x_1, x_3	Word **embedding**, where **semantic** and **syntactic features** are **captured** from **unlabeled text data**, is a **basic procedure** in Natural Language Processing (**NLP**). In this **paper**, we first **introduce** the **motivation** and **background** of **word embedding** and its related **language models**.
d_3	x_2	Structural **health monitoring** at **local** and **global levels** using **computer vision technologies** has **gained** much **attention** in the **structural health monitoring community** in research and **practice**. Due to the **computer vision technology application advantages** such as **non-contact**, **long distance**, **rapid**, **low cost** and **labor**, and **low interference** to the **daily operation** of **structures**, it is **promising** to consider **computer vision structural health monitoring** as a **complement** to the **conventional structural health monitoring**. This **article** **presents** a **general overview** of the **concepts**, **approaches**, and **real-life practice** of **computer vision structural health monitoring** along with some **relevant literature** that is rapidly **accumulating**.

3. Illustrative examples

In this section, we illustrate how ExpFinder works. The input data[1] includes three experts (i.e., x_1, x_2 and x_3) and three documents (i.e., d_1, d_2 and d_3) as shown in Table 1. Fig. 4 presents the output examples of the steps in ExpFinder:

1. **Step 1 - Extract tokens and topics**: Given $\mathcal{D}$, we extract tokens $\mathcal{W}$ and topics $\mathcal{T}$. In this step, we set a maximum length of phrase to be 3 such that we only obtain phrases that have less than or equal to 3 tokens. Additionally, we use the linguistic pattern presented in Section 2. The output of this step contains the set of 50 unique topics $\mathcal{T}$ and the set of 85 unique tokens $\mathcal{W}$. For example, extracted topics in d_1 include some single-token topics (e.g., `prerequisite` and `capability`) and some multi-token topics (e.g., `real-time nlp processing` and `electronic health record`).
2. **Step 2 - Estimate the weights of experts and documents given topics** - Given $\mathcal{T}$ and $\mathcal{W}$, we generate three main matrices (i.e., **EDM**, **DTopM** and **ETopM**) that will also be used in **Step 4.** To do this, we perform the following:
   - Given $\mathcal{W}$, we generate $\mathbf{DTM}^{3\times85}$ where each entry shows the TF of a token $w \in \mathcal{W}$ in a document $d \in \mathcal{D}$. For example, the 3×1 vector of `healthcare`, $\mathbf{DTM}_{*,\texttt{healthcare}}$, is $(1,0,0)$ which shows it occurs only in d_1 (see also $\mathcal{D}$ in Table 1). As another example, we obtain $\mathbf{DTM}_{*,\texttt{analytics}} = (2,0,0)$ which denotes that `analytics` appears twice in d_1.
   - Given $\mathcal{T}$ and $\mathbf{DTM}^{3\times85}$, we generate $\mathbf{DPM}^{3\times50}$ where each entry contains the weight of a phrase $t \in \mathcal{T}$ for a document calculated in nVSM. For example, suppose that `health analytics` is denoted as t_1, we then calculate nTF of t_1 in d_1 as:

$$n\mathrm{TF}(t_1,d_1) = \frac{\mathbf{DTM}_{1,\texttt{healthcare}} + \mathbf{DTM}_{1,\texttt{analytics}}}{|t_1|} = \frac{(1+2)}{2} = 1.5$$

     where $|t_1|$ is a number of tokens in t_1. Then, we calculate the N-gram IDF of t_1 as:

$$\begin{aligned} n\mathrm{IDF}(t_1) &= \log \frac{|D| \cdot df(t_1)+1}{df(\mathbf{DTM}_{*,\texttt{healthcare}} \wedge \mathbf{DTM}_{*,\texttt{analytics}})^2+1} + 1 \\ &= \log \frac{3\times1+1}{df((1,0,0)\wedge(2,0,0))^2+1} + 1 \\ &= \log \frac{4}{1^2+1} + 1 = 1.693. \end{aligned}$$

     Here, $\wedge$ is implemented in `NumPy`. Finally, we multiply $n\mathrm{TF}(t_1,d_1)$ with $n\mathrm{IDF}(t_1)$ to obtain nTFIDF of d_1 given t_1 as: $\mathbf{DPM}_{1,t_1} = n\mathrm{TFIDF}(t_1,d_1) = n\mathrm{TF}(t_1,d_1) \times n\mathrm{IDF}(t_1) = 1.5 \times 1.693 = 2.540$.
   - Given $\mathcal{D}$, we generate $\mathbf{EDM}^{3\times3}$ where each entry shows the authorship of an expert on a document. For example, x_1 is an author of d_1, and hence, the entry between x_1 and d_1 ($\mathbf{EDM}_{1,1}$) equals 1. Also, $\mathbf{EDM}_{1,3} = 0$ shows that x_1 is not an author of d_3 (See Table 1).
   - Given $\mathbf{EDM}^{3\times3}$ and $\mathbf{DPM}^{3\times50}$, we generate $\mathbf{ETopM}^{3\times50}$ where each entry contains nTFIDF weight of an expert given a topic. As we explained in Section 2, we assume that $\mathbf{DTopM} = \mathbf{DPM}$. Now, we demonstrate the calculation for the weights of experts $\mathcal{X}$ given t_1 ($\mathbf{ETopM}_{*,t_1}$) in nVSM as:

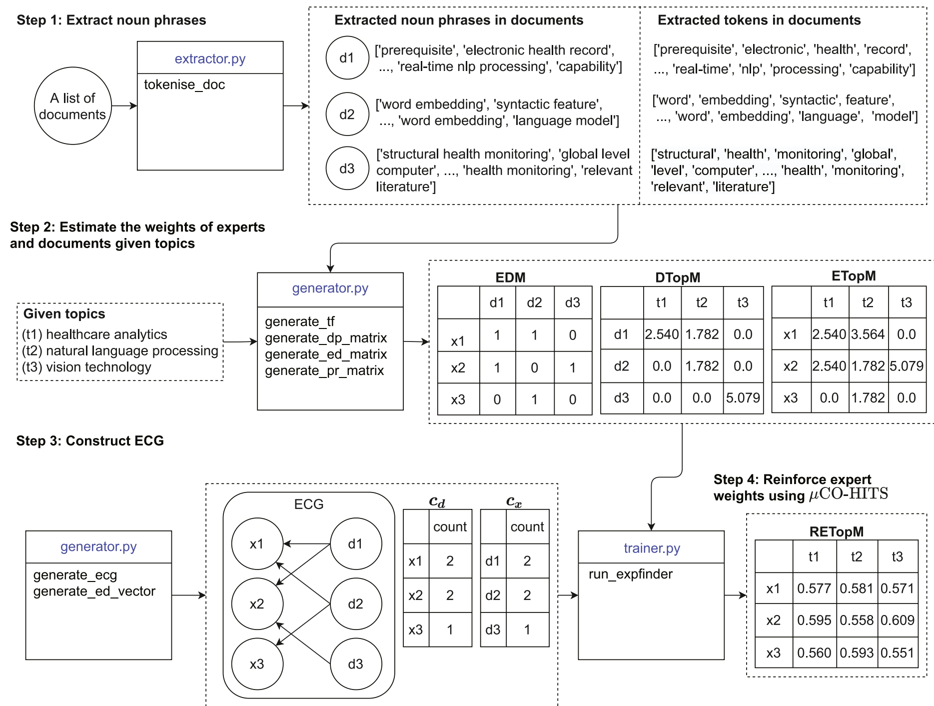
$$\begin{aligned} \mathbf{ETopM}_{*,t_1} &= \mathbf{EDM}^{3\times3} \cdot \mathbf{DTopM}_{*,t_1} \\ &= \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \cdot (2.540, 0, 0) = (2.540, 2.540, 0) \end{aligned}$$

   Note that $\mathbf{DTopM}^{3\times3}$ and $\mathbf{ETopM}^{3\times3}$ are only used for the visualization purpose. We use $\mathbf{DTopM}^{3\times50}$ and $\mathbf{ETopM}^{3\times50}$ for the estimation in **Step 4**.
3. **Step 3 - Construct ECG**: Given $\mathcal{D}$, we generate an ECG which has three expert nodes and three document nodes, as shown in Fig. 4. The graph is also used to generate 3×1 vectors (i.e., $\boldsymbol{c_x}$ and $\boldsymbol{c_d}$) that are used for the estimation of μCO-HITS in **Step 4**. For example, $\boldsymbol{c}_{\boldsymbol{d}_1} = 2$ indicates there are two documents (i.e., d_1 and d_2) pointing to x_1. Similarly, $\boldsymbol{c}_{\boldsymbol{x}_3} = 1$ indicates that there is one expert (i.e., x_2) who has authorship on d_3.
4. **Step 4 - Reinforce expert weights using μCO-HITS**: We use `run_expfinder()` in **trainer.py** to reinforce expert weights given topics $\mathcal{T}$. The function receives $\mathbf{DTopM}^{3\times50}$, $\mathbf{ETopM}^{3\times50}$, ECG, $\boldsymbol{c_x}$ and $\boldsymbol{c_d}$, generated in **(Steps 2 and 3)** as parameters, and generate the 3×50 Expert-Topic matrix where each entry shows the reinforced weight of an expert given a topic. Now, we illustrate the estimation for the reinforced weight of $\mathcal{X}$ given t_1 as:
   - Given 6 nodes in an ECG, we generate the adjacency matrix $\mathbf{M}^{6\times6}$ and its transpose matrix $\mathbf{M}^\top$ as:

$$\mathbf{M} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \mathbf{M}^\top = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

     where rows and columns are labeled with the sequence $\boldsymbol{s}$ (i.e., $\boldsymbol{s} = (d_1, x_1, x_2, d_2, x_3, d_3)$).
   - We apply L2 normalization for the 6×1 Expert-Topic ($\boldsymbol{\alpha_x}$) and the 6×1 Document-Topic ($\boldsymbol{\alpha_d}$) vectors. The output

[1] The example data are also provided in our Github repository.



EDM	d1	d2	d3
x1	1	1	0
x2	1	0	1
x3	0	1	0

DTopM	t1	t2	t3
d1	2.540	1.782	0.0
d2	0.0	1.782	0.0
d3	0.0	0.0	5.079

ETopM	t1	t2	t3
x1	2.540	3.564	0.0
x2	2.540	1.782	5.079
x3	0.0	1.782	0.0

c_d	count
x1	2
x2	2
x3	1

c_x	count
d1	2
d2	2
d3	1

RETopM	t1	t2	t3
x1	0.577	0.581	0.571
x2	0.595	0.558	0.609
x3	0.560	0.593	0.551

Fig. 4. Illustrative examples for ExpFinder: the blue labels indicate module names of ExpFinder. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

of each vector is as:

$$\boldsymbol{\alpha_x} = \mathrm{L2}-\mathtt{normalize}(\mathbf{ETopM}_{*,t_1}) = (0, 0.707, 0.707, 0, 0, 0)$$
$$\boldsymbol{\alpha_d} = \mathrm{L2}-\mathtt{normalize}(\mathbf{DTopM}_{*,t_1}) = (1, 0, 0, 0, 0, 0)$$

- We reinforce expert weights given t_1 in 5 iterations with $\lambda_x = 1$ and $\lambda_d = 0.7$. Here, we demonstrate the calculation of *average authorities* $\boldsymbol{a}$ and *average hubs* $\boldsymbol{h}$ at the first iteration ($k = 1$):

$$\begin{aligned}\boldsymbol{a}(\mathcal{X};t_1)^1 &= (1-\lambda_x)\boldsymbol{a}(\mathcal{X};t_1)^0 + \lambda_x\left(\frac{\mathbf{M}^\top \cdot \boldsymbol{h}(\mathcal{D};t_1)^0}{\boldsymbol{c_d}}\right)\\ &= 0\cdot(0,0.707,0.707,0,0,0) + 1.0\cdot\left(\frac{(0,2,2,0,1,0)}{(2,1,1,2,1,1)}\right)\\ &= (0,2,2,0,1,0)\end{aligned}$$

$$\begin{aligned}\boldsymbol{h}(\mathcal{D};t_1)^1 &= (1-\lambda_d)\boldsymbol{h}(\mathcal{D};t_1)^0 + \lambda_d\left(\frac{\mathbf{M}\cdot \boldsymbol{a}(\mathcal{X};t_1)^1}{\boldsymbol{c_x}}\right)\\ &= 0.3\cdot(1,0,0,0,0,0) + 0.7\cdot\left(\frac{(4,0,0,3,0,2)}{(1,2,2,1,1,1)}\right)\\ &= (3.1,0,0,2.1,0,1.4)\end{aligned}$$

where $\boldsymbol{a}(\mathcal{X};t_1)^1$ and $\boldsymbol{h}(\mathcal{D};t_1)^1$ are 6×1 vectors. At the end of the iteration, we normalize these vectors by applying the L2 normalization technique as:

$$\begin{aligned}\boldsymbol{a}(\mathcal{X};t_1)^1 &= \mathrm{L2}-\mathtt{normalize}(\boldsymbol{a}(\mathcal{X};t_1)^1)\\ &= (0,0.667,0.667,0,0.333,0)\\ \boldsymbol{h}(\mathcal{D};t_1)^1 &= \mathrm{L2}-\mathtt{normalize}(\boldsymbol{h}(\mathcal{D};t_1)^1)\\ &= (0.776,0,0,0.525,0,0.35)\end{aligned}$$

After 5 iterations, we obtain $\boldsymbol{a}(\mathcal{X};t_1)^5 = (0, 0.577, 0.595, 0, 0.56, 0)$ whose labels are presented by $\boldsymbol{s}$, and hence, we use x_1, x_2 and x_3 as indexes for obtaining a 3×1 vector (i.e., $\mathbf{RETopM}_{*,t_1} = (0.577, 0.595, 0.56)$).

The output is $\mathbf{RETopM}^{3\times 50}$. If we use t_1 and the other two topics (i.e., **natural language processing** and **vision technology**, denoted as t_2 and t_3, respectively), we can generate $\mathbf{RETopM}^{3\times 3}$ in Fig. 4. This matrix can be used for two major tasks (1) finding the most expertise query for each expert (also known as expert profiling); and (2) finding the best expert for a given query (also known as expert finding).

4. Impact and conclusion

With the growth of expertise digital sources, expert finding is a crucial task that has significantly helped people to seek the services and guidance of an expert [12]. ExpFinder is an ensemble model for expert finding that integrates nVSM with μCO-HITS to enhance the capability for expert finding over existing DLM, VSM and GM approaches. To our best knowledge, ExpFinder is the first attempt to provide the implementation of nVSM and μCO-HITS for expert finding.

The implementation of ExpFinder also provides functionalities that can be potentially useful for implementing other expert finding models. For example, our tokenisation module for extracting noun phrases using a linguistic pattern based on a part of speech (POS) can be easily customized based on researchers' purposes. The modules for building the presented Expert-Document matrix (**EDM**), Expert-Topic matrix (**ETopM**), and Document-Topic matrix (**DTopM**) can be usefully leveraged to represent relationships between experts and documents, experts and topics, and documents and topics. These relationships can be used to represent a collective information among experts, documents and topics and used to implement other graph-based expert finding models such as an author–document-topic (ADT) graph [7] and an expert–expert graph via topics.

We highlight that ExpFinder is a state-of-the-art model, substantially outperforming the following widely known and latest models for expert finding: *document language model* [3], *probabilistic-based expert finding model* [5], *graph-based models* [6–8]. Thus, the ones who want to extend ExpFinder can harness our implementation for further improvement of ExpFinder.

We presented the architecture and implementation detail of ExpFinder with an illustrative example. This would help researchers and practitioners to better understand how ExpFinder is designed and implemented with its core functionalities.

Declaration of competing interest

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

References

[1] F. Riahi, Z. Zolaktaf, M. Shafiei, E. Milios, Finding expert users in community question answering, in: Proceedings of the 21st International Conference on World Wide Web, 2012, pp. 791–798.
[2] C.T. Chuang, K.H. Yang, Y.L. Lin, J.H. Wang, Combining query terms extension and weight correlative for expert finding, in: 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), Vol. 1, IEEE, 2014, pp. 323–326.
[3] K. Balog, L. Azzopardi, M. de Rijke, A language modeling framework for expert finding, Inf. Process. Manage. 45 (1) (2009) 1–19.
[4] B. Wang, X. Chen, H. Mamitsuka, S. Zhu, BMExpert Mining MEDLINE for finding experts in biomedical domains based on language model, IEEE/ACM Trans. Comput. Biol. Bioinform. 12 (6) (2015) 1286–1294.
[5] P. Cifariello, P. Ferragina, M. Ponza, WISER: A semantic approach for expert finding in academia based on entity linking, Inf. Syst. 82 (2019) 1–16.
[6] H. Deng, M.R. Lyu, I. King, A generalized CO-HITS algorithm and its application to bipartite graphs, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2009, pp. 239–248.
[7] S.D. Gollapalli, P. Mitra, C.L. Giles, Ranking experts using author-document-topic graphs, in: Proceedings of the 13th ACM/IEEE-CS Joint Conference on Digital Libraries, JCDL '13, 2013, pp. 87–96.
[8] D. Schall, A Social Network-Based Recommender Systems, Springer, 2015.
[9] Y.-B. Kang, H. Du, A.R.M. Forkan, P.P. Jayaraman, A. Aryani, T. Sellis, ExpFinder: An ensemble expert finding model integrating N-gram vector space model and μCO-HITS, 2021, arXiv:2101.06821.
[10] M. Shirakawa, T. Hara, S. Nishio, IDF for word N-grams, ACM Trans. Inf. Syst. 36 (1) (2017).
[11] J.M. Kleinberg, Authoritative sources in a hyperlinked environment, J. ACM 46 (5) (1999) 604–632.
[12] R. Gonçalves, C.F. Dorneles, Automated expertise retrieval: A taxonomy-based survey and open issues, ACM Comput. Surv. 52 (5) (2019) 1–30.