FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Felipe Vianna

Expert Classification and Retrieval

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: doc. RNDr. Pavel Pecina, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2020

I would like to thank Pavel Pecina, for his encouragement and valuable consultations, making all this work possible. I am also grateful for all professors and colleagues at MFF I had the honor to meet during two intense years, for their immense contribution to my learning preceding this thesis. Finally, I would like to thank my family, especially my brother and best friend Leandro, for their unconditional love and support.

Title: Expert Classification and Retrieval

Author: Felipe Vianna

Institute: Institute of Formal and Applied Linguistics

Supervisor: doc. RNDr. Pavel Pecina, Ph.D., UFAL

Abstract: Searching for experts is a common demand, especially within organizations. A general task called expertise retrieval relates people to topics and, therefore, can be used for expert finding and/or profiling experts. Currently, most approaches used to solve this task are based on traditional document retrieval methods and do not consider prior profiling information available. In this thesis, it is proposed to map people to topics by training a multi-class classifier using available profile data. The inputs (documents associated to candidates by authorship or other relations) and target data (profile information) were prepared by unsupervised document classification methods and were used to train a neural network. The effects of feature ordering and a convolutional layer are also evaluated. The experiments show that profiling the experts is not only suitable for a recommender system, but also an effective way for expert finding, achieving a performance comparable to state of the art in benchmark tasks such as TREC Enterprise.

Keywords: expert profiling, expertise retrieval, document classification, neural networks

Contents

Preface

The major contribution of this work is its exploration of many NLP and Machine Learning concepts, gathering them together to solve an expertise retrieval task in a corporate data. Three major steps are followed: document classification, data preprocessing with feature ordering, and the multi label classifier. The effects of a convolutional layer is evaluated as well as the effect of ordering input features when they are composed by topics that carry relation to each other. In addition, different experimental setups are used for Document Classification, a prior step to the Expert Classification, being the latest also tested for several different parameters.

The structure of this work is as follows.

Chapter 1 contextualizes and describes the problem being solved. Also, the evaluation metrics to verify the performance on the task is explained.

In Chapter 2 previous solutions for the expertise retrieval task are explored, as well as general NLP and Machine Learning concepts relevant for understanding the proposed methodology.

Chapters 3 and 4 expose theoretical and practical details of the proposed method to classify experts with regards to topic queries.

Chapter 5 presents the results of experimental setups and, finally, Chapter 6 concludes the thesis with final remarks.

1. Introduction

"I don't need to know everything. I just need to know where to find it when I need it." - Unknown

Traditionally, Information Retrieval systems had been used to find the content needed. However, in many situations, it is desired to retrieve a knowledgeable person regarding a topic. In many cases, the information is not freely accessible, or requires a deep background to be interpreted. That would turn the quote above into *"I don't need to know everything. I just need to find who knows it".*

Manning et al. [2008] defines Information Retrieval as *"finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)".* This definition also applies if the searched material consists in experts instead of documents.

Demands for experts had existed since beginning of civilization, and as many new fields of expertise arise and people become more specialized, finding the right person for a specific task is even harder. Despite being an old issue, the research on how to find experts effectively is relatively new. Early researches in the 60's show that sharing the knowledge of experts in an organization leads to material gains (Davenport and Prusak [1998]).

Some well known circumstances when experts search is needed in modern days are, for example, recruiters searching for people whose profiles match a job description; an organization looking for an employee to be assigned to specific tasks; search for proper expert, with required competences, to review papers submitted for a conference (Charlin and Zemel [2013]).

Formalizing expertise is hard due to the tacit knowledge it is related to. People acquire knowledge through years of working experience rather than only formally studying, and all their *know-how* is not explicitly registered in documents. However, the only way to search for an expert in organizations is through the "explicit knowledge" presented in documents. Thus, we will notice that the challenges faced in document retrieval are also part of the ones we need to overcome in expert retrieval.

In document retrieval, each document is a retrievable unit to be ranked during a search. Yet, in expertise retrieval, the candidates are represented as a composition of multiple documents they are associated to, *e.g.* by authorship, mentions or citations. Moreover, different document sources might be weighted differently while building the candidate's representation, as technical manuals and email threads, for example, might indicate different levels of expertise in the underlying topic they refer.

Expertise Retrieval is the general area interested in creating a mapping between people and expertise topics. It covers two search interests: finding the expert on topic 'X', and find the topics in which a person is an expert (Balog et al. [2012]). The later is called *expert profiling* or *expert classification.* Both *Expert finding* and *Expert profiling* are directly related, as both aim to estimate associations between topics and people.

In this work, the links between topics and people are constructed from the perspective of *expert profiling* and it is ultimately treated as a machine learning classification task.

1.1 Problem Description

Expertise Retrieval is often reduced to the scope of specific organization and is usually part of an enterprise search system (Balog et al. [2012]). Most of the large organizations have internal systems and databases to store their proprietary knowledge. In the case of companies that provide specialized services to clients, it is also important to keep track of employees' profiles. This work is performed in such environment, in the context of an organization that manages their internal knowledge not only as documents, but also mapping its employees capabilities.

The aim of this work is to obtain the associations between experts and topics, as in example Table 1.1, based on a collection of documents belonging to an private organization. The values can be seem as the probability of person e being an expert in a topic query q. Specifically, the table is built from *Expert Profiling* perspective and serves as input to a Recommender System.

	topic1	topic2	topic3	topic4	...
person1	0	0	0.3	0.9	...
person2	0.7	0	0.3	0	...
person3	1	0	0	0.5	...
person4	0	0.8	0.1	0.2	...
⋮	⋮	⋮	⋮	⋮	...

Table 1.1: Association between people and topics.

According to Prem Melville and Vikas Sindhwani, from IBM Watson Research Center:

> *"The goal of a Recommender System is to generate meaningful recommendations to a collection of users for items or products that might interest them."*

Here, the items of interest are topics. Not only for the users themselves, but also for the retrieval system using this data to build the expert candidates representation, which is out of the scope of this work.

Feeding a recommender system with the results obtained by *expert profiling* provides a way to collect valuable data for evaluation. The user feedback (accept or reject recommendation) and current profile topics (when available) can be taken as true information, despite self assessment of expertise being still subjective. Figure 1.1 demonstrates the task and the focus of this work.

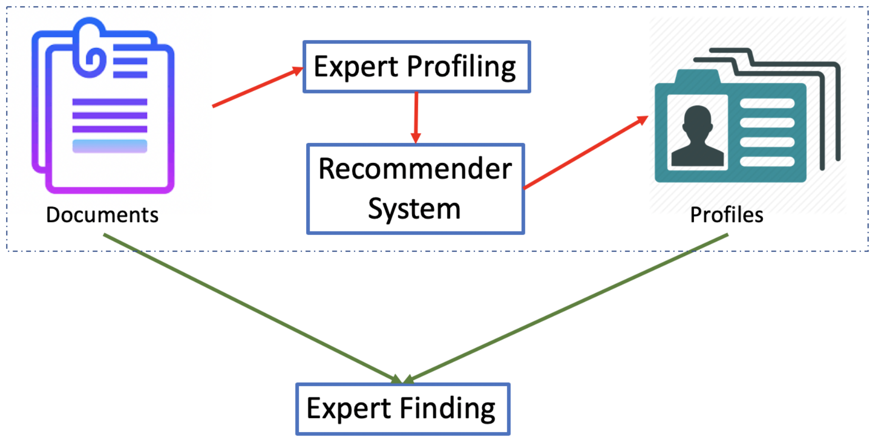


Figure 1.1: Architecture for expertise profiling and finding proposed. The focus of this work lies on the components inside the dashed box.

The assignment of topics known by an individual into his/her profile adds another source of data for further retrieval - *expert finding*. *Tags* assigned to specific field, e.g. *'skills'* in someone's profile are approved or curated by the individuals themselves, becoming trusted data for a retrieval system. Being that said, the focus of this work is to build associations between topics and people from input data, enriching profiles information and enabling their usage in further expert finding tasks.

Language modeling techniques from Information Retrieval had been used to compute the probability of a person being an expert given a query $P(e|q)$ (Balog et al. [2006]). However, in this work the task is solved as a multi label classification problem, computing the probabilities based on pre processed documents and current data available in people's profiles. The proposed classifier is a neural network with a convolutional layer and will be presented in details in Chapter 3.

1.2 Evaluation

The task consists in assigning scores associating each user (or expert candidate) to each topic generating a table resembling Table 1.1. Therefore, *expert profiling* and *expert finding* can be evaluated from the resulting table. For *profiling*, the top highest scored topics per person are retrieved. For *expert finding*, the highest scored people per topic are retrieved.

In Information Retrieval, metrics such as *Precision* and *Average Precision* (AP) are widely used. Following, the evaluation details based on these metrics are presented.

1.2.1 Expert Profiling Evaluation

The proposed system is built from *expert profiling* perspective and the top 5 topics per user are taken to feed the recommender system. Two metrics are used to evaluate profiling: AP over topics, and AP_5 over users.

Equations 1.1 and 1.2 shows how AP is obtained. First computing the Precision for each topic query:

$$P(q) = \frac{R_q}{N_q} \tag{1.1}$$

where $P(q)$ is the precision for all recommendations of topic q. R_q is the number of relevant recommendations of topic q, and N_q is the number of recommendations for topic q. A relevant recommendation is considered the one accepted by the user when recommended, or if the system recommends something that is already present in his/her profile.

AP, then, is computed as the average precision over all topic queries evaluated, according to Equation 1.2

$$AP = \frac{1}{|Q|} \times \sum_{q \epsilon Q} P(q) \tag{1.2}$$

where $|Q|$ is the number of topic queries, and $\sum_{q \epsilon Q} P(q)$ is the sum of all topics individual precision.

AP described above measures how well the system performs across the topic queries covered. However, it is AP_5 that measures how the system perform across the users. AP_5 can be computed as:

$$AP_5 = \frac{1}{|U|} \times \sum_{u \epsilon U} \frac{R_u}{5} \tag{1.3}$$

where $|U|$ is the number of users in the evaluation set U, R_u is the number of relevant recommendations for user u.

The above metrics are computed in the following sets of users:

- S_1: All users that processed the recommendations, including those present in the training set. Size: 6525 users.
- S_2: Only users present in training that processed recommendations. Size: 2362 users
- S_3: Only users not present in training and that processed the recommendations ($S_1 - S_2$). Size: 4163 users
- S_4: 100 users not present in training set and that had at least 5 topics already filled in their profiles.

1.2.2 Expert Finding Evaluation

Expert profiling and *expert finding* are sides of the same coin, and both can be done ranking the scores obtained by user-topic associations. Although the methodology discussed in this work assign the scores from *profiling* perspective, *expert finding* is also evaluated through AP_{10} and AP_{100}. Equation 1.4 shows how the values are calculated:

$$AP_k = \frac{1}{|U|} \times \sum_{q \epsilon Q} \frac{R_{kq}}{k} \tag{1.4}$$

where $|U|$ is the number of users in the evaluation set U, R_{kq} is the number of relevant users among top k users retrieved for topic query q. A relevant user for the topic is considered as the one who has the topic assigned to his/her profile.

AP_{10} is computed for all user sets $S1$, $S2$, $S3$ and $S4$ described above. AP_{100} is not computed for $S4$ as it has only 100 users, therefore all of them would be retrieved for any topic and the scores computed would become irrelevant.

2. Releated Work

In this chapter, popular methods for expert profiling and expert finding are introduced. Also, relevant concepts for the approach proposed in Chapter 3 are reviewed.

2.1 Previous Work in Expertise Search

Finding experts, before automated methods, relied on building a database to store the information regarding knowledge and skills of a person. These databases were manually structured and updated, resulting in high maintenance costs. Early automated approaches emerged, mainly focused on expert finding in specific domains, building candidate representations from homogeneous document sources. Finally, due to the limitations of these systems, the interest of industry and academia raised in this area making the systems evolve to perform the task over heterogeneous sources of data, as those found in document collections within corporate databases (Balog et al. [2006]).

One main source of studies and information regarding expertise retrieval is the outcome of work made for the Text REtrieval Conference (TREC) on their Enterprise test collections from 2005 to 2008. Some benchmark models and current state of the art methods were obtained during this effort. More details about the tasks, data and results can be found in Bailey et al. [2007], the overview of 2007 work.

Expertise retrieval systems aim to establish a relationship between experts and topics, through the collection of documents available. People are not directly retrievable, so they need to be represented based on the documents. Two main approaches can be used for that: *Profile-based* and *document-based* methods (Balog et al. [2012]). The first one starts from creating a textual representation of each person based on the documents they are associated to. These representations are built prior to any query, so they are *query-independent*. Once a query is performed over the collection of pseudo-documents that represent the people, they can be ranked using standard document retrieval techniques. The second approach happens the other way around. First ranking documents according to the query, and then ranking the experts based on the relevance score of the documents retrieved. As the representation of the experts changes according to topic queries, this approach is *query-dependent*.

Both methods above mentioned need to associate documents to people at some point. A document content can provide evidence of expertise for its author. However, when a document has multiple authors it becomes hard to define which section of the document represents the expertise of each of the authors. Moreover, some people might be mentioned in a document content, and the proximity between the mention and the topic could be an indication of expertise.

Now, some of the methods to model associations between the query topic and expert candidates are described . These methods were applied in related work and were proposed in Balog et al. [2012] and Balog et al. [2006].

2.1.1 Generative Probabilistic Models

In this family of models, the target is to compute $P(e|q)$, the likelihood of person e is an expert in topic query q, and then rank them by the highest probabilities. This likelihood can be computed directly in *Candidate Generation Models* as shown later, or by applying Bayes's Theorem, in *Topic Generation Models*:

$$P(e|q) = \frac{P(q|e)P(e)}{P(q)} \tag{2.1}$$

where $P(e)$ is the probability of a candidate and $P(q)$ the probability of a query. $P(q)$ is constant for each query, therefore the probability of a candidate being an expert is given by $P(q|e)$ weighted by prior probability $P(e)$. Both ways above mentioned are based on language modeling techniques, in which the basic idea is to compute the probability of observing a query in given document:

$$P(q|d) = \prod_{t \epsilon q} P(t|d)^{n(t,q)} \tag{2.2}$$

where $n(q,t)$ is the number of times term t is present in query q, and $P(t)$, the document language model, is the probability term t is in document d.

Candidate Generation Models:

The candidate generation models estimate the $P(e|q)$ from a document-centric approach, computing:

$$P(e|q) = \sum_{d} P(e,d|q) = \sum_{d} P(e|d,q)P(d|q) \tag{2.3}$$

where $P(d|q)$ denotes the probability document d is relevant to query q. To apply Equation 2.2, $P(d|q)$ can be rearranged applying Bayes's rule:

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)} \tag{2.4}$$

where $P(q)$ is the probability of a query and $P(d)$ the probability of a document. The co-occurrence model $P(e|d,q)$ represents how much associated expert e is to document d given query q. The actual computation of the co-occurrence model assumes e and q are conditionally independent given document d, so that $P(e|d,q) = P(e|d)$.

Topic Generation Models:

In this method, again, the target is to estimate $P(e|q)$, but since the queries consist of topics composed by few terms, a more accurate estimation is obtained applying Bayes' Rule as in Equation 2.1. Note $P(e)$, the candidate prior, is a term in this equation. It is usually assumed to be uniform, however non-uniform priors can lead to improvements (Fang and Zhai [2007]). For now, let us assume $P(e)$ to be uniform and it should not affect the ranking. The most important term to be estimated is $P(q|e)$, the probability query-topic q is generated by expert candidate e. In Balog et al. [2006] there are two different propositions to calculate this probability:

Candidate Model: In this model, an expert candidate language model is built from the collection of documents each person is associated to, i.e, an individual is represented as a composition of documents prior to any query. Then, $P(q|e)$ is estimated from this representation. Each candidate e is represented by candidate model θ_e, so that $P(t|\theta_e)$ is the probability of a term t given candidate model θ_e. Thus, the likelihood a candidate would produce a query can be predicted by:

$$P(q|\theta_e) = \prod_{t\epsilon q} P(t|\theta_e)^{n(t,q)} \tag{2.5}$$

If term $t\epsilon q$ is not in candidate model, $P(q|\theta_e)$ goes to zero. To address this issue, standard smoothing is employed as:

$$P(t|\theta_e) = (1-\lambda)P(t|e) + \lambda P(t) \tag{2.6}$$

where $P(t|e)$ is the probability of term t given candidate e, $P(t)$ the probability of term t in whole collection of documents, and λ is the smoothing parameter in the interval $[0, 1]$. Now, the terms are associated to candidates through the documents. Thus, $P(t|e)$ can be approximated by:

$$P(t|e) = \sum_d P(t|d, e)P(d|e) \tag{2.7}$$

$P(d|e)$ is weight factor as it represents the strength of association between document and candidate. $P(t|d, e)$ can be approximated by $P(t|d)$ assuming terms and candidates are conditionally independent given a document.

Combining the three equations above,:

$$P(q|\theta_e) = \prod_{t \epsilon q} \{(1 - \lambda)(\sum_d P(t|d)P(d|e)) + \lambda P(t)\}^{n(t,q)} \tag{2.8}$$

Document Model: Here, first the documents that best describe the topic query are retrieved. Then considers candidates associated to these documents as possible experts.

$$P(q|e) = \sum_d P(q|d, e)P(d|e) \tag{2.9}$$

where $P(d|e)$ works as a weight to indicate how well document d describe candidate e and $P(q|d, e)$ measures how document d is good indication candidate e is an expert in q. The probability of query given document and candidate is:

$$P(q|d, e) = \prod_{t \epsilon q} P(t|d, e)^{n(t,q)} \tag{2.10}$$

Assuming the candidates are independent given a document, $P(t|d, e) = P(t|\theta_d)$. The resulting equation combining the previous is:

$$P(q|e) = \sum_d \{\prod_{t \epsilon q} P(t|d)^{n(t,q)}\}P(d|e) \tag{2.11}$$

Document models have an advantage for being built on top of standard document index already available for document search. The major work become to implement candidate-document associations, whereas in candidate models a complete new index is required (Balog et al. [2006]).

2.1.2 Discriminative Probabilistic Models

Another set of models for information retrieval applications is the discriminative models. While Generative models rely on assumptions such as the independence of term distributions, discriminative models learn from the data itself. Also, document-candidate association in generative models are heavily based in heuristics, while in discriminative models more features could be incorporated, when available. Following, two of these models explored in Balog et al. [2012] are briefly presented.

Learning to Rank:

This method constructs a ranking model automatically, using training data. The documents are represented by feature vectors on the input side, and the output is composed by the relevance of each document. In expert search, previous work treated the problem as a binary classification - a person either is an expert or not - and trained a multi-layer perceptrons and logistic regression as a classifier. When the features are well designed, it is shown to outperform generative language models.

Learning Models for Ranking Aggregates:

Based on ensemble methods in supervised machine learning, the idea is to combine many weak learners to produce a strong model. Each weak learner is composed by document weighting models (tf-idf, BM25), ranking cutoffs and voting techniques. The strong ranking model is learned by weighting the combination of the weak models obtained before.

2.2 NLP and Machine Learning

This section explores some common NLP and machine learning algorithms that might be combined in the development of this work. Most of them are widely used in other NLP or retrieval tasks, and can also be combined for expertise classification and retrieval.

2.2.1 TF-IDF

Term Frequency-Inverse of Document Frequency is a popular statistical method to measure the importance of a word to a document in a collection. The scores given by tf-idf compose the vector representation of the documents and are widely used search engines to score the relevance of a document to a user query (Ramos [2003]).

To capture the importance of a word within a document it is assumed that high term frequency indicates the term is important, as the document refers to it often. However, from a document collection perspective, if a term is present in many documents, the term might not be so informative. It is wanted to set higher weights for rare terms. Therefore, the document frequency is inverted, and it is trivial to see that in Equation 2.13 common words like "the", "is" and other stop words, would automatically be assigned lower IDF weights.

The tf-idf is obtained by multiplying the Term Frequency by the Inverse of Document Frequency. A term with high tf-idf weight is considered to be highly discriminatory, meaning in a retrieval scenario, this document would be very likely relevant to a query containing this term.

Term Frequency:

The simplest way to compute term frequency is to count the number of times a term appears within a document $tf(t,d) = f_{t,d}$ (Ramos [2003]). However, some other variants might be used. One option is to scale *tf* by length of the documents, so that long ones don't produce too high tf scores:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \epsilon d} f_{t',d}} \tag{2.12}$$

Another common option is to have *zero* for $f_{t,d} = 0$ and $1 + log(f_{t,d})$ otherwise.

Document Frequency:

The $df(t,D)$ is computed by taking the number of documents in which term t is present throughout the whole collection D. The inverse of document frequency is, then, given by $\frac{N}{df(t,D)}$, where N is the total number of documents in the collection. As collections can have large amounts of documents, the *idf* is logarithmically scaled as

$$idf = log \frac{N}{df(t,D)} \tag{2.13}$$

The keyword extraction could be done by simply taking the terms with higher tf-idf scores. Some usual additional steps, though, are performed, such as removing stop words in advance and also generating *n-grams* tokens, so that expressions can be extracted as well. The final score is computed combining Equations 2.12 and 2.13:

$$TFIDF(t,d) = tf(t,d)log\frac{N}{df(t,D)} \tag{2.14}$$

2.2.2 BM25

BM25 is a Bag of Words retrieval function that assigns scores to documents given a query. It computes the score based on TF and IDF weighting. The base formula to compute a document score given a query is shown in Equation 2.15:

$$score(d,q) = \sum_{t \epsilon q} log\frac{N}{df(t,D)} \times \frac{(k_1+1)tf_{td}}{k_1(1-b+b(\frac{L_d}{L_{avg}}))+tf_{td}} \tag{2.15}$$

where $log\frac{N}{df(t,D)}$ is the idf, tf_{td} is frequency of term t in document d, L_d is the document length, L_{avg} is the average document length in full collection D. k_1 is a positive tuning parameter to scale the term frequency. Note that $k_1 = 0$ makes the score depend only on idf. b is another tuning parameter in interval $[0,1]$ which determines the scaling by document length. $b = 1$ makes fully scaling by document length whereas $b = 0$ results in no length normalization.

Long documents are penalized by BM25, clearly observed on Equation 2.15, as the denominator of the second term increases proportionally to document length. In BM25+, an improved version of BM25, the contribution of a term occurrence is lower bounded by a parameter δ. Its score is computed in Equation 2.16.

$$score(d,q) = \sum_{t \epsilon q} log\frac{N+1}{df(t,D)} \times (\frac{(k_1+1)tf_{td}}{k_1((1-b)+b(\frac{L_d}{L_{avg}})+tf_{td})}+\delta) \tag{2.16}$$

Lv and Zhai [2011] suggest that $\delta = 1$ is effective across collections.

2.2.3 Word Embeddings

In most machine learning models and Natural Language Processing tasks, it is needed to represent the strings as vectors. For some features with small vocabulary, it can be simply handled using one-hot encoding. When the vocabulary increases, this approach turns to be unfeasible as the vector representation needs the same dimension as the vocabulary size. Also, one-hot encoding assumes each word is independent, not capturing any semantic similarity between them.

Research has been made towards representing words as vectors with reasonable dimension size (much smaller than vocabulary size) and also embedding the semantics in these representations.

Context-free embeddings - Word2vec

Mikolov et al. [2013a] presents two methods for generating word embeddings from large corpus (billions of words) having a vocabulary in the order of millions of words. Both approaches shown are context-free since each word is represented by a single vector, regardless the multiple meanings it might have due to context.

The two methods are CBOW - Continuous Bag of Words - and Continuous Skip-gram, both based on Neural Network Language Model.

Continuous Bag of Words: in CBOW the basic idea is to train a neural network, with single hidden layer, to predict a target word given the inputs are its surrounding words (this surrounding is defined by parameter *window size*). Despite making use of a supervised method, the model is unsupervised, based only on the corpus. The embedding of each word is obtained from the weights connecting the hidden neurons to each of the words in the target vector. The figure 2.1 illustrates the architecture proposed.

Continuous Skip-gram: Skip-gram has an architecture very similar to CBOW, but the target is to predict the surrounding words given a word. Figure 2.2 shows an example for training word embeddings with vocabulary of size 10000 and embedding with 300 dimension, which means a hidden layer size of 300 neurons. Each time a word is seeing on the corpus, the input vector for it is a one-hot vector of size of vocabulary with a '1' on the position representing the word. The only weights to be used from the hidden layer will be the ones connected to the '1' in this vector, as the other weights would be multiplied by 0. The target vector is again a vocabulary-size one-hot representation with 1's in the positions of words in the context (window) of the input word. Therefore, a classifier is trained to predict words that are in the context of each word. After many iterations, the weight vector connecting the hidden layer to each of the words in the input vector is taken as the word embedding.

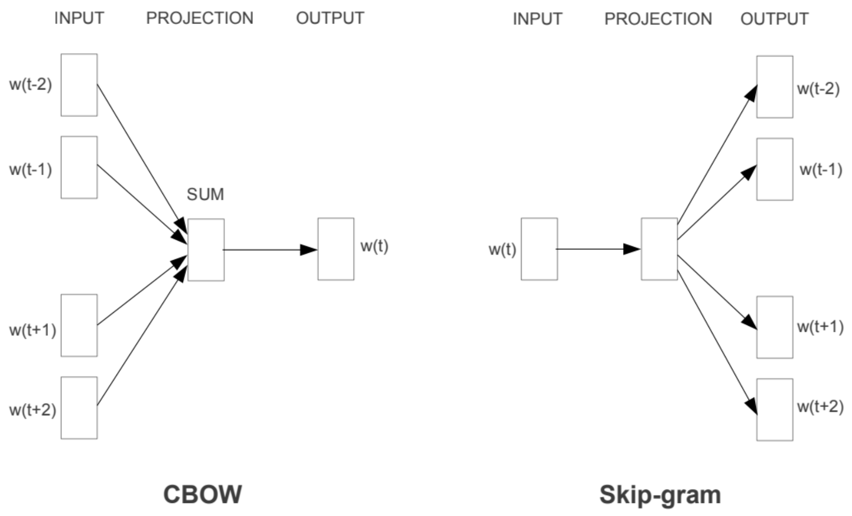


Figure 2.1: CBOW vs Skip-gram architectures. *Mikolov et al. [2013a]*

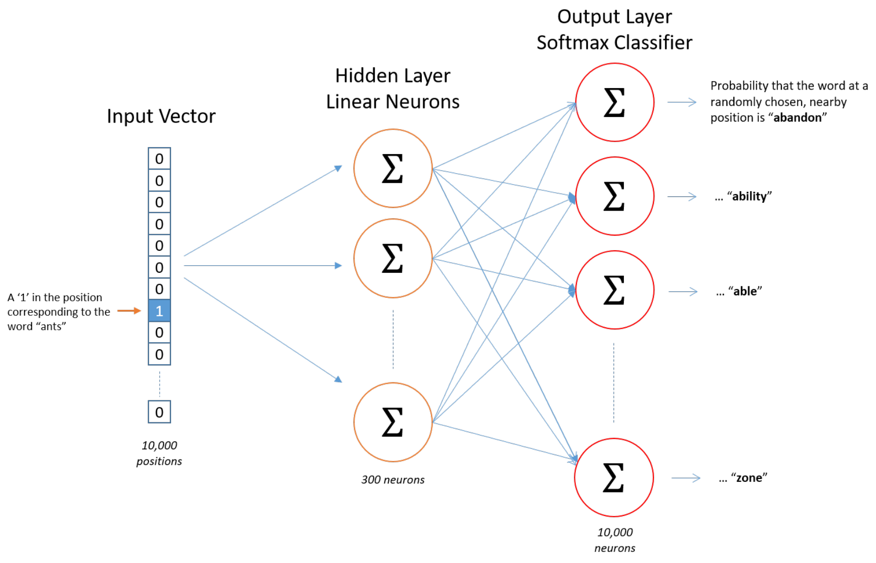


Figure 2.2: Skip-gram neural network. *http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model*

Contextualized Word Embeddings

Very frequently, words can have multiple meanings, depending on their context. Word embeddings obtained by word2vec methods described previously despite capturing latent syntactic and semantic similarities, do not handle the polysemous context-dependent nature of words.

One of the methods to obtain such contextual embeddings is through the implementation of character level language models. In Akbik et al. [2018] they implemented a framework called "Flair", consisting of a Neural character-level language model using Recurrent Neural Networks to learn how to predict the

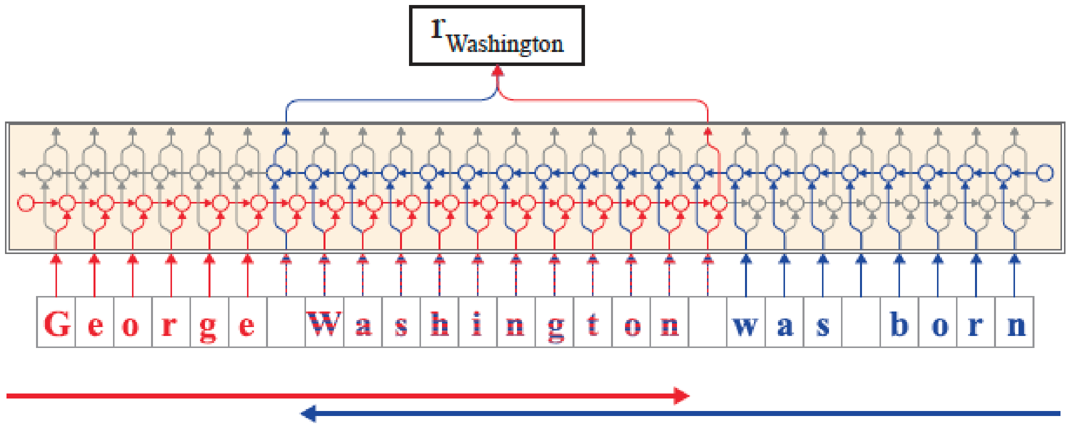


Figure 2.3: Contextual embeddings implemented in Flair: Forward LM embed context before the word while backward LM embed the context after the word. The two are concatenated together to represent the word in the specific context. (Akbik et al. [2018])

next character given sequence of characters. They show that selecting appropriate hidden states from the language model leads to word-level embeddings that are highly effective in downstream tasks.

The architecture in Flair is based on Recurrent Neural Network (Bi-LSTM to capture the context from forward and backward directions), and allows to pre-train in large unlabeled corpora. Modeling words and sentences as sequence of characters also helps handling better even misspelled words (Akbik et al. [2018]). To capture the context, the Bi-LSTM produces a forward and a backward language model. The embedding of a word is extracted from the output hidden state after the last character of a word (on forward LM) and before the first character (on backward LM). Figure 2.3 illustrates the method. An interesting observation is that it is possible to obtain the embeddings for *n-grams*, for example *"George Washington"* by simply taking the forward LM embedding of the word *"Washington"* and the backward LM embedding of the word *"George"*. It is also trivial to observe that in a sentence such as *"George Washington visited the city of Washington"*, it would be expected to obtain an embedding for *Washington* representing a person's name and another that would represent a city name.

2.2.4 Sentence Encoders

Deep Learning tasks require large data sets and, when it comes to NLP, they are still limited compared to image data sets. Annotated data is expensive to obtain and scarce. Many models, thus, perform transfer learning through the use

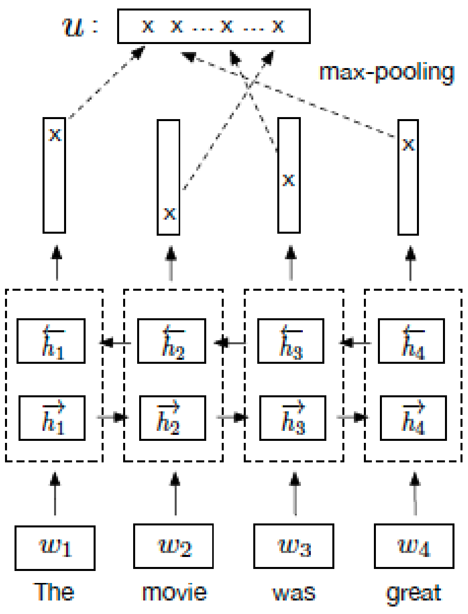


Figure 2.4: Sentence encoder proposed by Facebook for Infersent. Each LSTM cell provides its output state, and the max pooling is taken over all output states to form the sentence embedding. Conneau et al. [2017]

of word embeddings trained in large data sets. Recent studies show that sentence level embeddings improves results in transfer tasks (Conneau et al. [2017]), and therefore pre training sentence embeddings in publicly available annotated data might be worthy the effort.

Facebook Infersent

In Conneau et al. [2017] they show that an encoder composed by Bi-LSTM initialized with pre-trained word embeddings, *e.g* word2vec, further trained in Natural Language Inference downstream task, results in a sentence embedding that performs better in transfer tasks. The encoder is showed in Figure 2.6 and the NLI training architecture is illustrated in 2.6. After the whole training in Stanford NLI task, they provide the model to get the encoding *"u"* in Figure 2.6 given a input sentence.

Alternatively, Hierarchical Convolution Network can be used to encode the sentence. This last approach, illustrated in Figure 2.5, concatenates the different levels of abstraction from the input sentence and output a fixed size representation.

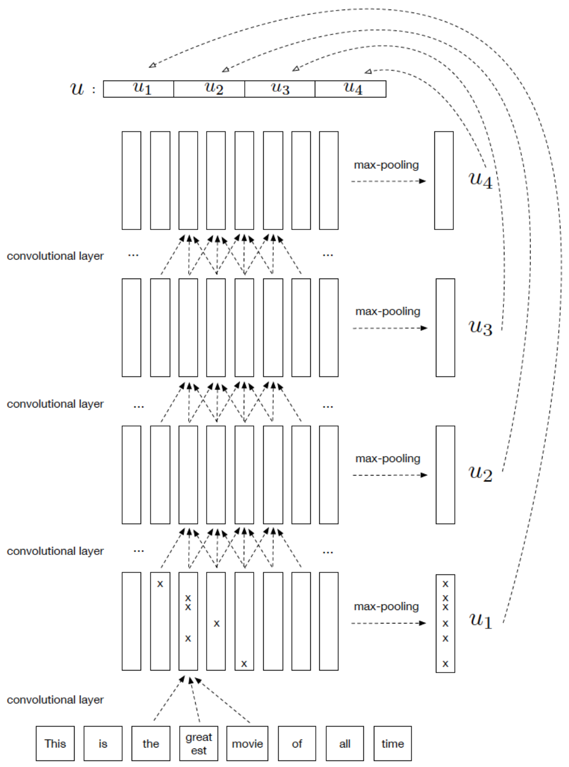


Figure 2.5: Sentence encoder proposed by Facebook for Infersent. Each convolutional layer provides a different abstraction representation of the input sentence. Conneau et al. [2017]

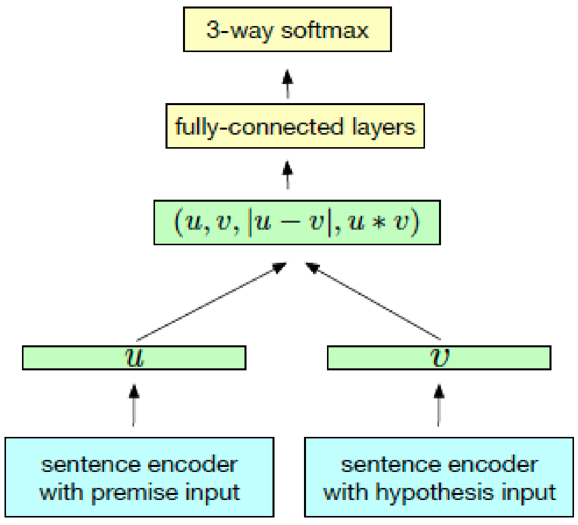

3-way softmax
fully-connected layers
$(u, v, |u - v|, u * v)$
u
v
sentence encoder with premise input
sentence encoder with hypothesis input

Figure 2.6: Sentence encoder trained over Stanford NLI data. Vectors u and v are obtained from encoder in . A bigger representation of both vectors is given by the concatenation of u, v, absolute element-wise difference, —u-v— and element-wise product u * v. Conneau et al. [2017]

2.2.5 Embedding Similarity

The main objective of embedding words, *n-grams* or sentences, regardless of contextual or context-free methods, is to obtain a vector with predefined dimension that encodes the semantics of such piece of text. To compare if two words are similar, for example, their embedding vectors generated by same encoder can be taken and compared with regard to some similarity metric. The most common metric for this case is *cosine similarity*, which computes the cosine of the angle θ between two vectors u, v as in Equation 2.17:

$$similarity(u, v) = cos(\theta) = \frac{u \cdot v}{\|u\|\|v\|} \tag{2.17}$$

2.2.6 String Similarity

Many times in text processing the difference between strings should be computed, for example in spell checking. Damerau-Levenshtein Distance is an improved version of Levenshtein-Distance (Setiadi [2013]) that computes the distance between two strings based on four operations: insertion, deletion, substitution and transposition, being the last the improvement proposed by Frederick Damerau. Below, some examples of distances between pairs of strings computed by DL distance:

- rock $\leftrightarrow$ sock: 1 (substitution (r,s))
- rock $\leftrightarrow$ rocky: 1 (insertion (y))
- rock $\leftrightarrow$ sick: 2 (substitution (r,s) and (o,i))
- rock $\leftrightarrow$ clock: 2 (insertion (c) and substitution (r,l))
- rock $\leftrightarrow$ rcok: 1 (transposition (o,c))

This method, differently than embedding similarity, is agnostic to any meaning of the words and it is only based on positions of characters in the words being compared.

2.2.7 Deep Learning

In Machine Learning, the effort lies in creating computer programs that are able to improve its performance in some task automatically, through experience (Mitchell [1997]). The task can be in multiple domains and the experience comes from the data. A variety of paradigms and algorithms compose this multi-disciplinary field of studies, being Deep Learning one of them.

In some learning tasks, it is difficult for the computer to understand the raw input data, e.g the meaning of the pixel values in an image recognition task. Therefore, mapping from pixels to object classification in a single function is very complicated (Goodfellow et al. [2016]). Deep Learning addresses this issue by having multiple layers applying mathematical functions that create new representation of the input. Each layer extracts features and build a representation simpler to be understood by the next layer. In the image recognition example, one can think as the first layers extracting the edges of an image, followed by corners and contours, and lately object parts. From the representation of object parts, a classification layer is able to learn how to map the inputs to the labels.

As the model parameters grow with depth, Deep Learning models usually require larger datasets for the learning process. With the increase of hardware capabilities, Deep Learning models became more feasible and its performance is the State of the Art specially in NLP and Computer Vision tasks. Adding layers to Multi-layered Perceptron (MLP) is the first intuition expecting a model to learn from very low level features, but in practice does not perform well. Therefore, some modern approaches were developed, such as Convolutional Neural Networs, and Recurrent Neural Networks.

Convolutional Neural Networks- CNN

Three important characteristics serve as motivation for the development and use of Convolutional Neural Networks (CNN): sparse interactions, parameter sharing, and equivariant representations (Goodfellow et al. [2016]). First, let us investigate the convolution operation. Largely used in signal processing for Linear Time Invariant (LTI) systems, this operation shows that a signal can be represented by the weighted sum of impulses shifted in time (Haykin and Van Veen [1999]).

$$x[t] = \sum_{k=-\infty}^{\infty} x[k]\delta[t-k] \tag{2.18}$$

being $x[t]$ a discrete signal, $x[k]$ the value of the signal in instant k and $\delta[t-k]$ the impulse in instant t. The convolution operation above can be also represented by the asterisk symbol as $(x * \delta)(t)$. Switching from signal processing in engineering to Machine Learning nomenclatures, let us refer to x as **Input** and δ as the **Kernel**. In ML applications, such as image processing, the Inputs can be multiple dimensional arrays, and therefore the Kernel should also be multi dimensional. In the case of a 2-dimension input, e.g pictures, the convolution operation would be like:

$$x[i,j] = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.19}$$

being a commutative operation, it can be rewritten in a form more suitable for machine learning implementation:

$$x[i,j] = (I * K)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,j) \tag{2.20}$$

Note that when the index in the input increases, the index in the Kernel decreases. This is necessary to keep the commutative property. However, this property is not important in neural networks (Goodfellow et al. [2016]) and therefore implemented without flipping the Kernel as:

$$x[i,j] = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,j) \tag{2.21}$$

Figure 2.8 shows an example of convolution where the kernel traverses the input only through valid position, i.e. within the input. Imagine the kernel *'sliding'* over the input, and the output takes the summation of the element-wise multiplication between Kernel and the Input values it *'touches'*. Note that traversing through only valid positions the output has smaller dimension. If $I_{i,j}$, $K_{m,n}$, output $O_{i-(m-1),j-(n-1)}$.

Traditional MLP have a matrix of parameters describing interaction from each input unit to each output unit, i.e connections between the layers. In CNN, the interaction sparsity happens as the Kernel has a size smaller than the input. Therefore, each input unit is connected to at most the total number of parameters in Kernel.

Not only the Kernel promotes sparse interactions due to its size being smaller than input size, the same Kernel values are used all over the input. Instead of learning a full set of parameters connecting a input unit to all output units as in MLP, in CNN layers the weights in the Kernel are applied in multiple input signals, so the kernel parameters are shared by virtually all the inputs (with exception to the boundaries). However, although the direct connections are sparse, the deeper the CNN, the more input signals would reach each output unit.

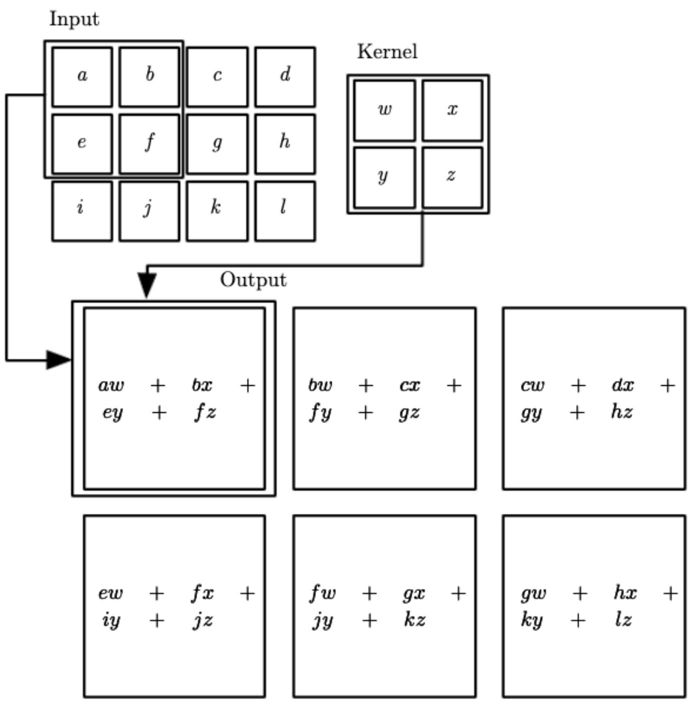


Figure 2.7: 2-D convolution without Kernel Flipping. Goodfellow et al. [2016]

The equivariant representation idea, indicates that for some changes in the input, the output changes similarly, e.g, if an object in a input image is shifted somewhere else, the output of the convolution operation would be the same as applying the convolution and later the shift (Goodfellow et al. [2016]). This is very useful as for identifying patterns that might be appearing in different locations in an image, or events in a time series.

A typical convolutional layer performs the convolution operation, which can consist in multiple kernels at the same time in parallel. Then applies the non-linear activation function, usually ReLU, sigmoid or tanh, in a stage known as ”Detector Layer”, and lately, a pooling operation (Goodfellow et al. [2016]).

Pooling

A pooling operation summarizes the convolution output. This is obtained by replacing the values in a certain location by the statistical representation of nearby outputs (Goodfellow et al. [2016]). Two examples of pooling operations are *max pooling* and *average pooling*. The pooling operation has also some kind of *”rectangular window”* that traverses the values outputted after activation function. The operation, then, reports the maximum output seen in the window (for max pooling) or the average of the outputs seen in the window (for average pooling).

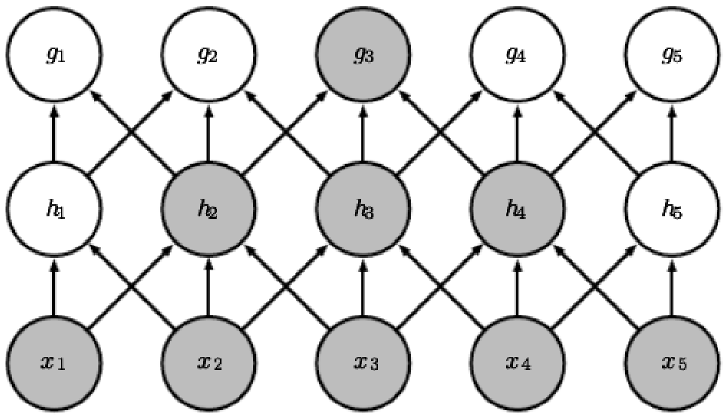


Figure 2.8: When multiple CNN layers are stacked, the receptive field in last layers receive signal (indirectly) from wide range of units from input. (Goodfellow et al. [2016])

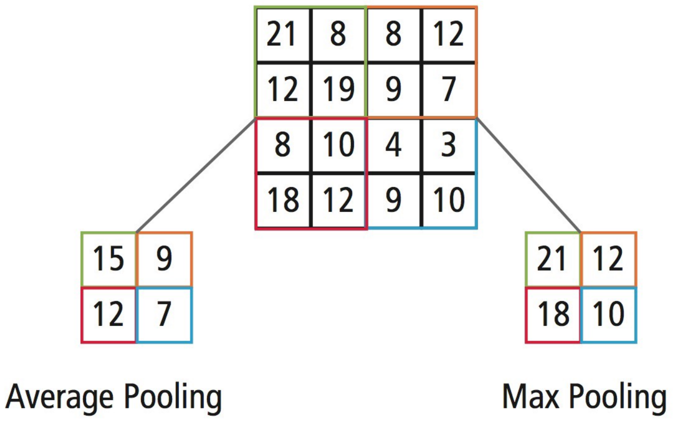


Figure 2.9: Taking an output layer after convolution and activation function applied, the average pooling and max pooling operations are displayed for pool size 2x2 and stride 2. (Source: *https://medium.com/@Aj.Cheng/convolutional-neural-network-d9f69e473feb*)

Regardless of the pooling method mentioned above, this operations results in a vector representation 'invariant' to small translations on input. Given the input is shifted in one direction, the output of pooling would have smaller changes. The larger the pooling window, the highest the invariance aspect.

The relative invariance to translations is a very important feature of CNNs. In many cases, it is more important if a feature is present than its exact location. In a sentence, for example, words can change the order and still keep same meaning. This is one of the motivations when using convolutions for sentence encoders as in 2.5. In images, the exact location of an eye is not determinant to classify if the picture contains a face.

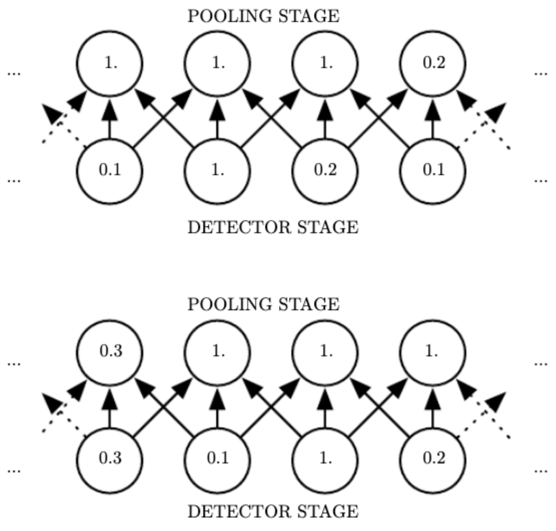


Figure 2.10: On the top, pooling output for pool size = 3, stride = 1 on first row, based on values of activation function in second row. On the bottom, the input is shifted to the right, changing the values of activations in all positions, but still on pooling output just half of values changes. (Goodfellow et al. [2016])

Recurrent Neural Networks - RNN

Recurrent Neural Networks are a type of neural networks specialized in processing sequence of values (Goodfellow et al. [2016]). Convolutional networks are able to scale up to large inputs in a grid representation. Similarly, Recurrent networks can process sequences of variable lengths, being these sequences much longer than any other type of network would be able to handle.

One key point to make recurrent networks feasible is parameter sharing across different parts of the model, so that the trained model can be used for sequences of different lengths and generalize across them. An example enabled by this structure is the identification of pieces of information in sequences of different lengths, or different ordering:

Michael Jordan was the best basketball player of Chicago Bulls.

The best basketball player of Chicago bulls was Michael Jordan.

Supposing both sentences above would be processed in a Multi Layer Perceptron handling fixed sequence lengths (using padding in case of short sentences, for example), each word would have an independent set of parameters according to its position in the sentence. Therefore, *"Michael Jordan"* in the first sentence would be multiplied by the same weights as the words *"The best"* in second sentence. Consequently, the network would need to update all the parameters to learn all language rules in all possible positions in a sentence. The RNN, instead, share the same weights across several time steps.

When describing the convolutional operations, it was mentioned that CNNs also have parameter sharing. However, the output of CNN is a function of small neighboring members of the input, whereas in RNN the output is a function of the previous outputs and the new input, computed using the shared parameters:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \tag{2.22}$$

$h^{(t)}$ is the current state, $h^{(t-1)}$ is the previous state, $x^{(t)}$ is the current input,and θ is the set of parameters shared throughout the computations. This recursive implementation allows for much deeper network. For a defined sequence length, the *"unfolded"* recurrent representation can be obtained, as in figure 2.11.

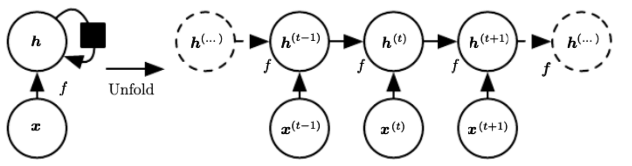


Figure 2.11: RNN cell unfolded as a graphical representation of Equation 2.22. (Goodfellow et al. [2016])

This kind of neural networks can be used to train how to predict the future state based on the past states, taking $h^{(t)}$ as a representation of the whole sequence. One example would be to train a language model to predict the next word given a sequence of words. The state $h^{(t)}$ is a lossy representation of the full sequence (Goodfellow et al. [2016]) but some mechanisms can be implemented to selectively keep some aspects of the past sequence. Gated RNNs, such as LSTM and GRU, are specialized architecture to control which information is allowed to pass through the states.

LSTM - Long Short-Term Memory

Long term dependencies are problematic in Recurrent Neural Networks. Gradients propagated over many stages during the training either vanish or explode. LSTM cells introduce self-loops to allow the gradient to flow for long duration (Goodfellow et al. [2016]). This architecture also innovates by implementing a cell state c_t that accumulates state information (Shi et al. [2015]). Self-parameterized gates control how new information is accumulated to the cell and how the previous are forgotten. The structure is shown in Figure 2.12, and the computations happening within a LSTM cell are presented in Equation 2.23.

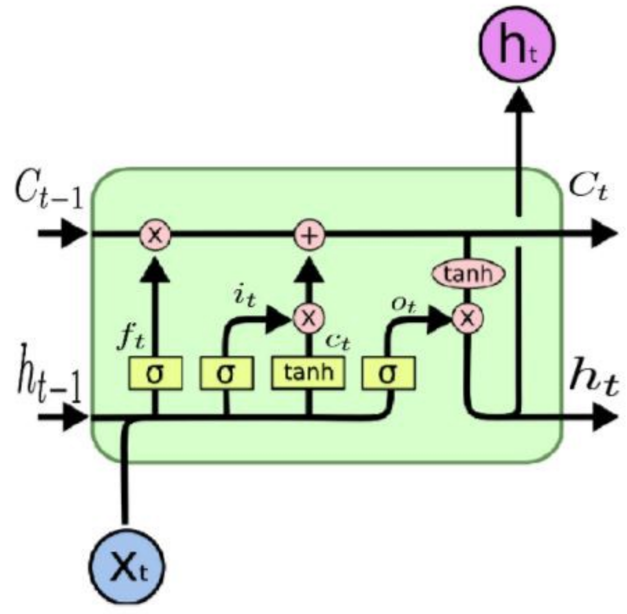


Figure 2.12: LSTM cell and all its gates. Source: *http://colah.github.io/posts/2015-08-Understanding-LSTMs*

$$
\begin{aligned}
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
c_t &= f_t \circ c_{t-1} + i_t \circ tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
h_t &= o_t \circ tanh(c_t)
\end{aligned}
\tag{2.23}
$$

Where $\circ$ denotes element-wise multiplication. f_t stands for the *forget gate*. This gate is responsible to decide which information on the cell state should go through the next cell state. It looks at h_{t-1} and x_t, multiplies by the weights and applies sigmoid function. The output of sigmoid is a number in the interval [0,1] that multiplies c_{t-1}. If 0, all previous information is forgotten'; if 1, all information passes through the next cell state. i_t is the input gate. Here the LSTM cell decides the new information to be added to the cell state. Similarly to forget gate, a sigmoid is applied after the linear transformation of new input and previous hidden state, to decide how much of the new input should be added. The candidate values to be added is given by $tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$. At this point, c_t updates are known, consisting in what to forget from previous cell state, and what to add to new cell state, resulting in the c_t value described in Equation

2.23. o_t is the output gate and, similarly to all other gates, outputs a number on interval [0,1] that defines how much of the updated cell state c_t goes through as the cell output h_t. The cell state c_t is also transformed to the interval [-1,1] by the application of *tanh* non linearity before the multiplication by o_t.

The LSTM enabled many tasks to be successful. Some examples are speech recognition, handwriting generation, machine translation, and image caption (Goodfellow et al. [2016]).

3. Methodology

In this chapter the datasets and the components of the expert classification tool developed to feed the recommender system are described.

3.1 Objective

In people's profile data, most of the fields, including the one about a person expertise, are filled as free text and are also optional, resulting in incomplete profiles. However, its content is validated by users themselves and the data presented there should be reliable.

Combined with other document collections, the profile data can be used for expert finding, and its completeness might increase the performance of such retrieval task. The target of the proposed system is to train a classifier that estimates the probability a user is an expert in a given topic. The input samples for the classifier are built from the document collection, and the available profile data is used to construct the target values. The predictions of this classifier shall feed a *Topic Recommendation System* that sends suggestions of topics someone might know about. The recommendations are expected to be validated by the users, and the ones accepted are automatically added to their profiles. Therefore, the users' feedback (accept or reject topic) can be used to evaluate the system and, additionally, more data is collected to retrain the model iteratively.

3.2 Data Collection

The data used in this work has been provided by a private company and is confidential. The main data sources are, thus, described in terms of their characteristics, but content will remain undisclosed.

3.2.1 Internal Publications

The Internal Publications is a collection of documents that covers a diverse number of topics. It comprises in reports of outcomes resulting from multidisciplinary studies worldwide. Experts publish their work on intranet, resulting in a knowledge repository, composed mainly by PowerPoint and PDF files. By the time of this study, the collection had 36486 documents and the following fields:

- ID: Unique identifier of the document: 36486 documents.
- Authors: Each document might have one or more authors. The total number of authors is 28272, identified by distinct user IDs.
- Title: The title of the document
- Abstract: Each document contains a short description with average 90 words, explaining the content of the document
- Primary Content: This field holds the parsed bodies from PowerPoint or PDF files. Contains average 5840 words.
- Authored Date: Date of publication of the document.

3.2.2 Previous Work Assignments

Another source of information is the data related to previous work assignments. Each project executed by group of experts has a list of related topics and list of workers in the team assigned for the task. This collection contains data for 37621 projects, and 26509 unique employees participating in them. The fields of interest in this data are:

- Project ID: Unique identifier of the document: 37621 in total.
- Team Person IDs: Each document explicitly identifies the workers assigned to the project described in the document. There are 26509 unique person IDs.
- Team Incurred Hours: Number of hours each person dedicated to the project.
- Team Roles: Role each person played in the project.
- Primary Terms: The expert managers assign topics which the project is related to. On average, 7 topics are assigned per document.

3.2.3 "Topics I Know" in Profiles

One important section in people's profiles is called *"Topics I Know"*, and is filled as free text by the users themselves, listing their skills. Currently, there are 119449 entries in profiles, being 27447 unique *"topics"* (they can not be assured to be relevant topics, as it is a free text field) for a total of 38013 active users. However only 10487 of the active users have at least 1 topic filled in their profiles. For the purpose of this study, the focus is in a subset of 17753 users that belong to a specific category of employees. From this subset, 7539 have at least one topic filled in the section *Topics I Know* in their profiles.

3.2.4 Taxonomy

The taxonomy is structured as a graph that is supposed to hold the topics covered by all the projects executed by the organization. It is built by experts and currently 4600 topics compose it, starting from the most abstract terms and going deeper to specific detailed topics. The maximum depth of the taxonomy is 10. An example of small section of the taxonomy is shown in Figure 3.1.

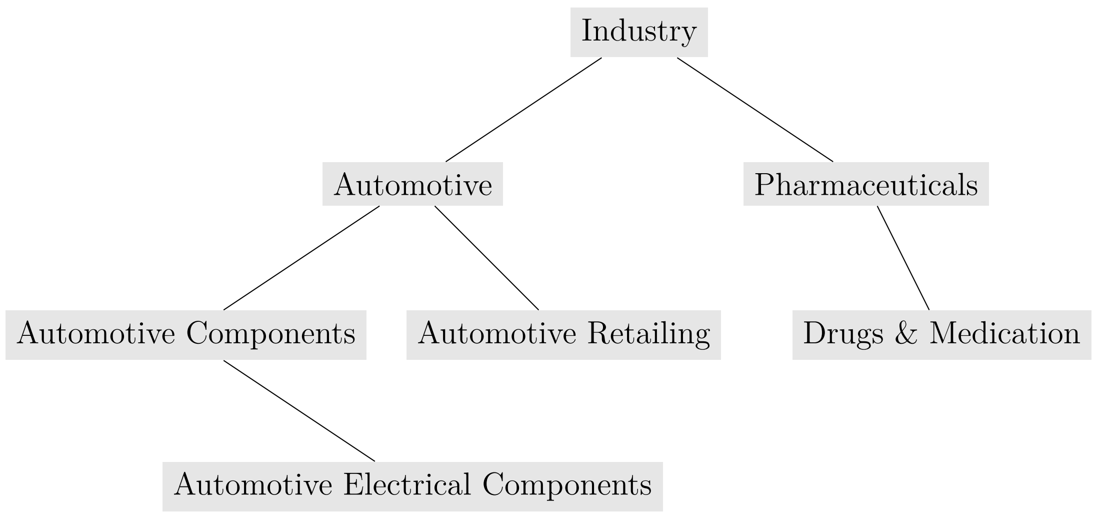


Figure 3.1: Example of how the taxonomy graph is structured.

All topics for expert profiling or query topics for expert finding discussed further on should be taken from this taxonomy.

3.3 Expert Classification Tool

Classifying the users with respect to their expertise consists ultimately in building a matrix associating the person to the topics assigning a score for each pair person-topic. Table 1.1 shows an example of such representation. The scores represent the degrees of association between topics and people, or the probability a person *p* is an expert in topic *t*. Now, the proposed steps to get this table completed are described.

3.3.1 Classification of Documents

Considering the topic queries are fixed in advance by the taxonomy set, it is proposed to classify the documents as a initial step. They can be associated to the employees later (by authorship, for example), and the authors can be represented by a concatenation of their documents representations. Therefore, all documents are classified assigning labels taken from the taxonomy. Optionally, knowing it has a graph structure, a subset of these terms can be selected to perform the classification, for example setting minimum depth in the graph and taking only terms deeper than it, controlling how granular the topics should be.

Internal Publications:

Internal Publications, described in 3.2.1, compose the biggest set of documents. Its classification consists of associating each document to Taxonomy terms. Later on, the frequency of each term assigned to the document is taken as a measurement of how strong a document is associated to the term. To achieve these representations for all documents, as in table 3.1, the following steps are performed:

	topic1	topic2	topic3	topic4	...
document1	3	1	0	9	...
document2	2	0	1	4	...
document3	5	0	0	2	...
document4	0	0	2	1	...
⋮	⋮	⋮	⋮	⋮	...

Table 3.1: Association between documents and topics.

Keyword Extraction: The first step for labeling the documents is to extract the keywords, more specifically *n-grams*, from the texts. The *n-grams* can be generated in two ways: all possible *n-grams* in some range ([1,3] for example), or using word collocation ("*phrases*") as proposed in Mikolov et al. [2013b]. After the vocabulary is built as the *n-grams*, the TF-IDF score can be calculated for each of them.

The TF-IDF score ranks the importance of the words for each document. A fixed number of keywords per document can be extracted as the most important words, or an amount proportionally to document length can be taken. *E.g.*, extracting the top 5% *n-grams* from a document of 1000 *n-grams* would return 50 *n-grams*. Note that, if the vocabulary is composed by all possible *n-grams*, a document with 100 words would ultimately generates 100 unigrams, 99 bigrams, 98 trigrams, and so on. Using word collocation to build the vocabulary, its size cannot be known in advance. At the end of this step, for each document there would be a list of keywords (*n-grams*) and their respective TF-IDF scores. A minimum score is set as a threshold for a keyword to be considered.

Word2vec CBOW: Word2vec models were trained in the Internal Publications using traditional CBOW method. As the semantics of *n-grams* are not simply given by the composition of each word individual meanings, the method to generate *phrases* proposed in Mikolov et al. [2013b] can also be applied. This reduces the size of vocabulary and speed up the training, while still captures important *n-grams*. It could also be trained in all *n-grams*, but it would require much heavier computation.

More details on parameters and implementation will be presented later in Chapter 4.

Keywords Augmentation: To allow for semantic matches, similar words, e.g. words/phrases that have high cosine similarity score (measured between the vectors from embeddings), are added in a list for each of the keywords extracted. An example to clarify the idea: A document D_a containing a *n-gram artificial intelligence* would probably have a similar term *machine learning* from document D_b matching by embeddings similarity because both terms would appear in similar context. This helps in a sense that keywords from a document are expanded using words that are not present in the document itself, but are observed in similar contexts on other documents within the collection. Summarizing, it is an attempt to add words with similar meanings, or synonyms related to the set of keywords.

The result of this step is the augmentation of the list of keywords obtained in Keyword Extraction step.

String Matching with Taxonomy Topics: Having the list of Keywords already augmented, one last step is needed to assign labels to the documents. For each document keyword, a string match with topics from the Taxonomy is tried. To add flexibility in this matching, considering some words can have typos or small differences, e.g. American and British English particularities, some tolerance is allowed based on Damerau-Levenstein Distance. For example, the DL-distance between two words can be set to be at most 15% of string length. Consider that *"analyze"* has length 7 and 15% would allow for distance=1 in this metric. Therefore it would match with *"analyse"* (DL distance is 1 in this case by substitution operation) and this keyword match would not be lost.

The output of the steps above is a set of topics per document, with the count of how many times they had matched with keywords previously extracted, generating a table similar to example Table 3.1. Multiple matches to the same topic can happen due to Augmentation step. For example, let us assume "Milk" has a word embedding very similar to "Yogurt". In case "Milk" belongs to Taxonomy and the document contains both words, it would count one match between "Milk" keyword and "Milk" label, and also a match between "Yogurt" keyword and "Milk" label. Regardless, the count gives an idea of how much the document is related to each topic. It is expected that higher counts on specific topic means the document is more related to it.

Previous Work Assignments:

Similarly to what is done for the Internal Publications, the Previous Work Assignments should also be classified by assigning taxonomy terms as labels. The word2vec model trained on Internal Publications is used to perform the Keywords Augmentation taking the *Primary Terms* directly as the keywords, considering they are already given. Next, the *String Matching* is done exactly the same way as mentioned previously for Internal Publications. Let us call this method *"standard"*.

Additionally, an alternative way for classifying these documents is proposed. This collection has human-assigned labels (*Primary Terms*). Knowing all the terms in this field are assigned by experts and very likely describe accurately the document, a direct semantic match seems a suitable option in addition to the string matching presented previously. To do so, the following steps are performed:

Encode All Terms as Vectors: Semantic matching requires a vector representation to encode meaning of the terms and lately their similarities can be computed. Word2vec model trained on Internal Publications provides possible embeddings for the terms. However, the vocabulary obtained is limited to the words seen in those documents. Therefore, another powerful and flexible embedding can be utilized.

Flair Embeddings: Flair embeddings (Akbik et al. [2018]) is a character level language model. Therefore it can encode any sequence of characters. The model pre-trained on Web, Wikipedia and Subtitles (mix-forward and mix-backward in Flair Embeddings [1]) is taken as a starting point, and transfer learning is done, continuing the training on proprietary collection. The assumption on transfer learning for language models is that general English is already learned from the large corpus, and the model specializes in vocabulary, jargons and context of the smaller proprietary collection.

After training, all topics (*n-grams*) can be encoded using the same technique used for sentence encoding. Flair offers *pool* or *RNN* options. In *pool* each word is encoded separately, and the topic is represented by the vector obtained by averaging the vectors that represent each of word composing the topic name. For *RNN*, the embedding for each word is used as input to an RNN layer, and the hidden state vector in output cell is taken as the sentence embedding.

Topics Filtering: An optional step is to filter out some topics. When the embedding vectors are calculated for all topics, it is expected that the encoder performs well enough to produce vectors that are close in the vector space only when they are semantically similar. Therefore, if many topics have high cosine similarity to each other, very likely it is a result of bad performance of the encoder. This issue might happen to terms not seen frequently in training data, for example, meaning their final vector representation are closer to random initialization than to the place around the words from similar context. A possible solution for this issue is to get list of similar topics for each topic, based on a similarity metric such as cosine distance. Setting a high similarity threshold and counting how many words are in each list, the cleaning step consists in removing the topics with long lists.

[1] https://github.com/flairNLP/flair/blob/master/resources/docs/embeddings/FLAIR_EMBEDDINGS.md

The assumption is that long lists are results of encoding issues rather than a topic in taxonomy would really be similar to many others. The list length to be considered *"long"* is defined by a parameter.

Semantic Matching: To start this process, the following content are already encoded as vectors (embeddings):

- All taxonomy topics.
- The *Primary Terms* of Work Assignments.

All pairwise combinations of taxonomy topics and Primary Terms can be evaluated with respect to their embeddings similarities. The metric used is, again, cosine similarity. A high threshold is set so that all matches above it are taken A low threshold is defined so that, if no word matches above high, the best match within the range [low, high] is taken. These thresholds are in the parameter settings of the model.

Summarizing, two methods described above are used independently to assign a set of topics to each document:

- *Primary Terms* as keywords, Augmentation by word2vec trained on Internal Publications followed by String Matching with Damerau-Levenstein tolerance.
- Flair embeddings: Semantic matching comparing *taxonomy topics* to *Primary Terms*.

Now all the documents are classified/labeled with the taxonomy topics that have matched in one of the methods selected. Despite all above mentioned approaches being implemented, their results can be selected independently. Some comparison of the results of each method will be showed later on chapter 4.

"Topics I Know":

Similarly to *Primary Terms*, the topics written in profiles are human curated and should be contain meaningful words. Therefore, the same previous methods are performed to classify them with closest labels from taxonomy topics:

String Matching with Taxonomy Topics: Augmentation of each *"Topic I Know"* by word2vec similar words and direct match with DL distance as tolerance.

Ecoding using Flair: Compute the embeddings for each *"Topic I Know"* and *Taxonomy Topics* using Flair and follow the steps:

- Topics Filtering on the labels set.
- Semantic Matching calculated as cosine similarity between *"Topics I Know"* and filtered taxonomy topics.

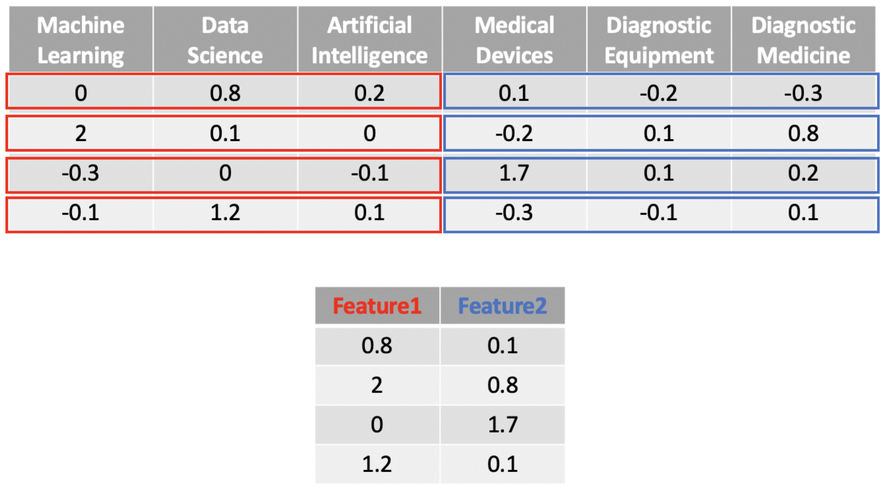
Finally, a list of topics per expert is produced, based on the matching of taxonomy with the skills found in their profiles section *"Topics I Know"*. One assumption is that a topic present in the profile is a true indication that the user knows it, but the absence of a topic not necessarily represents he/she does not know about it. Also, uncertainties are added since unsupervised approaches are used for matching profile topics with taxonomy topics (keyword augmentation and direct semantic matches). Nevertheless, all matched topics are later assumed to be relevant when training the classifier.

3.3.2 Expert Classification

In the previous sections the intent was to build the input and target data to train a classifier in supervised fashion. The aim now is to develop a multi-label classifier capable of learning how to estimate the probability a person knows about a topic. Many classifiers can be devised for this task, but neural networks with a convolutional layer can address one characteristic of the input data: the features are not fully independent from each other. The input features are topics from taxonomy, that is already constructed in a hierarchical way so that terms under same lineage are closer related to each other than to terms in other branches. Not all classifiers could take advantage of this particularity, thus this choice.

Another step proposed is to order the input features so that related topics are placed consecutively. More details on ordering implementation is given in Chapter 4. Having the input ordered, a convolutional layer composed by convolution and max pooling operations is expected to extract the most important feature for the next layer. In this particular case, differently from image recognition, it is not expected that the weights on kernel filters learn patterns that can be in different positions of the input vector (e.g. A wheel in bottom left corner or in bottom right corner is identified as a wheel). A pattern in a region of the input with topics related to Data Science definitely have different meaning compared to the same pattern in a region with topics related to Medical Devices, if these patterns would eventually occur. Therefore, the important feature of the convolution layer is more related to the max pooling operation, extracting the highest valued topic for the next layer, although some input patterns can be learned and extracted to next layers in the network.

Figure 3.2 shows the intuition of what is expected when applying a max pooling with 1x3 pool size and stride=3 in an artificial example considering values after the convolution operation. However, this is an extreme best case scenario example, as the number of related topics in same groups vary and there is no single pooling window that could fit all cases, making the task much more complex to be learned by the network.



Machine Learning	Data Science	Artificial Intelligence	Medical Devices	Diagnostic Equipment	Diagnostic Medicine
0	0.8	0.2	0.1	-0.2	-0.3
2	0.1	0	-0.2	0.1	0.8
-0.3	0	-0.1	1.7	0.1	0.2
-0.1	1.2	0.1	-0.3	-0.1	0.1

Feature1	Feature2
0.8	0.1
2	0.8
0	1.7
1.2	0.1

Figure 3.2: Example of feature extraction in max pooling layer

Its interesting to observe in this example, that someone who knows one of the related topics would produce similar values for next layer compared to another user knowing a topic in the same group. The feature extraction is learned during the training of the neural network, where the weights of the kernel filters are updated.

Ensemble generally produce more accurate results than individual classifiers, and theoretical and empirical research demonstrated that a good ensemble is the one where the errors of individual classifiers lies in different parts of input space (Maclin and Opitz [2011]). Given the input and target data are generated from unsupervised approaches, and also the previously mentioned incompleteness of the target data due to missing data on profiles, it is proposed in this work to use a bag of neural networks. The assumption is that the classifiers in the ensemble could learn how to fill the gaps in different profiles learning from randomly sampled inputs (bootstrapping). Neural Networks are also unstable due to random initialization of weights, making the individual classifiers unique and suitable for ensemble methods.

Summarizing the proposed classifier:

Input Ordering: Organize input features in a sequence that groups related topics.

Target Data: Some topics in target data have too little input samples, meaning a model can not learn a pattern to classify the user with respect to these topics. Therefore, there might be needed a reduction to fewer topics. The implementation of this step is discussed in Chapter 4.

Classifier: Neural Networks with Convolutional Layer to extract features to next dense layers where the classification is learned, optimizing the $F_{0.5}$ score.

Ensemble: Build an ensemble of 250 classifiers based on bootstrapping and bagging, training individual neural networks. Take the averaged predictions as the final prediction of the model.

Predictions: As the models are trained in sampled inputs, each model ends up seeing different samples during training. Moreover, the target data is incomplete, meaning a 0 in target data might indicate a missing topic in Profile. Therefore, the averaged predictions is taken from all data used for training and validation (users that are authors in Internal Publications, participated in Previous Work Assignments, and have at least one topic in their Profiles), assuming the ensemble compensates for the missing topics in profiles. Also, averaged predictions for test set (users for which there are input data, but there were no topics in their profiles) is computed for the recommender system and further evaluation based on users feedback.

4. Expert Classifier Implementation

In this Chapter, the implementation details of the proposed approach in Chapter 3 are discussed. The codes are fully available in GitHub [1].

4.1 Baseline model

The baseline line model is implemented as a document-based method, relying on the retrieval system BM25+, an improved BM25 version where the score is given by Equation 2.16. This model was selected for its ease of implementation and for having results comparable to the ones obtained by generative models (Balog et al. [2012]) described in Chapter 2. In this baseline the parameter values are $k_1 = 1.5$, $b = 0.75$, and $\delta = 1$.

To build a table as Table 1.1, firstly one similar to Table 4.1 for documents is built. The association between topics and documents (the *Internal Publications* in this case) is given by the BM25+ scores for each document, given each topic query taken from taxonomy.

	topic1	topic2	topic3	topic4	...
doc1	3	0.1	0	6.2	...
doc2	2.7	1	0	0	...
doc3	0	0	0	6	...
doc4	0	0	0	0	...
⋮	⋮	⋮	⋮	⋮	...

Table 4.1: BM25 scores for each document from Internal Publications, for each taxonomy topic.

Having this table and the authorship of the documents, the table for the authors can be constructed, having as topic score:

$$score(q,a) = \sum_{d\epsilon D_a} \frac{S(q,d)}{max(S(q))} \times e^{\frac{-t_d}{T}} \tag{4.1}$$

where q is the query topic, a is the author, D_a is the collection of documents author a participated in, $S(q,d)$ is the BM25+ score for document d given topic query q, $max(S(q))$ is the maximum score of a document given query q, this term

[1]https://github.com/felipenv/expertise-retrieval/tree/master/src

is responsible for scaling all scores to the [0,1] interval. Scaling makes difference as in BM25 queries with different lengths usually produces scores in different ranges, and scaling make the scores comparable for all topics. t_d is the time in years since document d was written, and T is the information retention constant. This is a parameter setting, and can be seen as the amount of years to have knowledge decaying by the factor of $e^{-1} \approx 37\%$.

$e^{\frac{-t_d}{T}}$ approximates the forgetting factor, and obsolescence of content. It is inspired by the psychology studies on knowledge retention (Rubin et al. [1999]) and also by the half life of content (Arbesman [2012]). Therefore, even the document being related to some topic, the author might not know it anymore due to forgetting it, or obsolescence of the topic.

Equation 5.1.1 is defined inspired by Document Model Balog et al. [2012] and the additional *forgetting factor* privileges people currently applying knowledge in some topic. The result with and without this factor is presented in Chapter 5.

4.2 Document Classification

The document classification is performed following the steps described in Chapter 3. Following, some implementation details are discussed.

Keyword Extraction

All Internal Publications Abstract and Primary content where used to train the TF-IDF model. First, they were cleaned by removing English stop words[2]. After some experimentation with different parameters, and manual evaluation of keywords extracted for some documents, the model used for the downstream tasks was the one trained with the following parameters:

- minimum document frequency: 5
- maximum document frequency: 0.5 * $|D|$
- n-gram range: [1,3]
- n-gram method: 'All'

The minimum document frequency of 5 seems to exclude partially the noise due to parsing from PPT or PDF (e.g. object names, characters representing

[2]https://gist.github.com/sebleier/554280

bullet points, text hidden by figures), but the frequent noisy terms are still kept. Increasing the value of this parameter would start to ignore rare, but important words. Using all possible *n-grams* makes it heavier for the computation. However, as later on they are filtered by string matching with human curated terms, in this way we do not miss *n-grams* that would not be in the vocabulary in a more strict method like *Phrases*. The extraction parameters are ratio of total words in document vocabulary (including all possible *n-grams* generated) and minimum TF-IDF score. The parameter values in this case were 0.05 for ratio, and 0.01 for the score. This step was implemented utilizing mainly *TfidfVectorizer*[3] from *scikit-learn*.

Word2vec CBOW

In parallel with the Keyword Extraction, the word2vec model is trained on the exactly same set of documents. Here, *n-grams* vocabulary is built using *"Phrases"* method, as the computation demanded for word2vec training is much higher than for TF-IDF. The parameters used were:

- minimum document frequency: 5
- size: 300 dimensions
- window: 10
- *n-gram* method: 'Phrases' for bigrams and trigrams.

The implementation is built over *Gensim Word2Vec models*, and the parameters not mentioned here are kept as default values from Gensim package [4].

4.3 Internal Publications Classification

Having the keywords extracted and the word2vec model trained, the document labeling can be performed. The first step is to expand the keywords by semantic similarities, computed as cosine distance between word vector pairs. The parameter and their default values [5] used for this expansion and their descriptions are as following:

[3]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction

[4]https://radimrehurek.com/gensim/models/word2vec.html

[5]Values selected after experimenting with several different settings and manual evaluation of sampled documents

- w2v similarity: 0.75. Minimum cosine similarity to accept a word as similar.
- max similarities: 5. It was empirically verified that when a word has too many similar words by word vector similarity, the word2vec encoding is generally not accurate. Most likely due to the lack of input data covering that word. If a word has the count of similar words higher than this parameter value, they are all removed assuming their encoding is most likely inaccurate.

Having the keywords expanded, they can be matched with the topics. The taxonomy structure, is in fact an Directed Acyclic Graph, as a term might have multiple paths that leads to it. The higher in the tree, the more generic is the term. The amount of terms at each level are:

	0	1	2	3	4	5	6	7	8	9
n Topics	3	30	325	1489	1911	665	260	65	18	7

Table 4.2: Number of taxonomy topics at each level. Level 0 is the most generic, level 9 the most specific

Therefore, it might be useful to set a minimum depth on this DAG, or a maximum height if looked bottom-up, and take terms that are granular enough for the document classification. The parameters in this step and their default values are [6]:

- Taxonomy Depth: 2. The level on the taxonomy DAG to be minimum depth or maximum height. When minimum depth, all topics from this level down are taken. When maximum height, all topics from the leaves till this level are taken.
- Depth type: minimum depth. Set if it's minimum depth or maximum height.
- Damerau Levenstein tolerance: 15%. The number of DL operations allowed as a percentage of string length.

The effects of the parameters are investigated through plots of the distribution of amount of keywords per document, before and after the expansion, for both *n-grams* methods. Figures 4.1, 4.2 and 4.3 show the results of keyword extraction

[6]Comparison of multiple values are displayed in graphs throughout this Chapter

for some parameters settings. The chart titles indicate the parameter values, being *mindf* the minimum document frequency in absolute number, and *maxdf* the maximum document frequency as a proportion of number of documents in the collection.

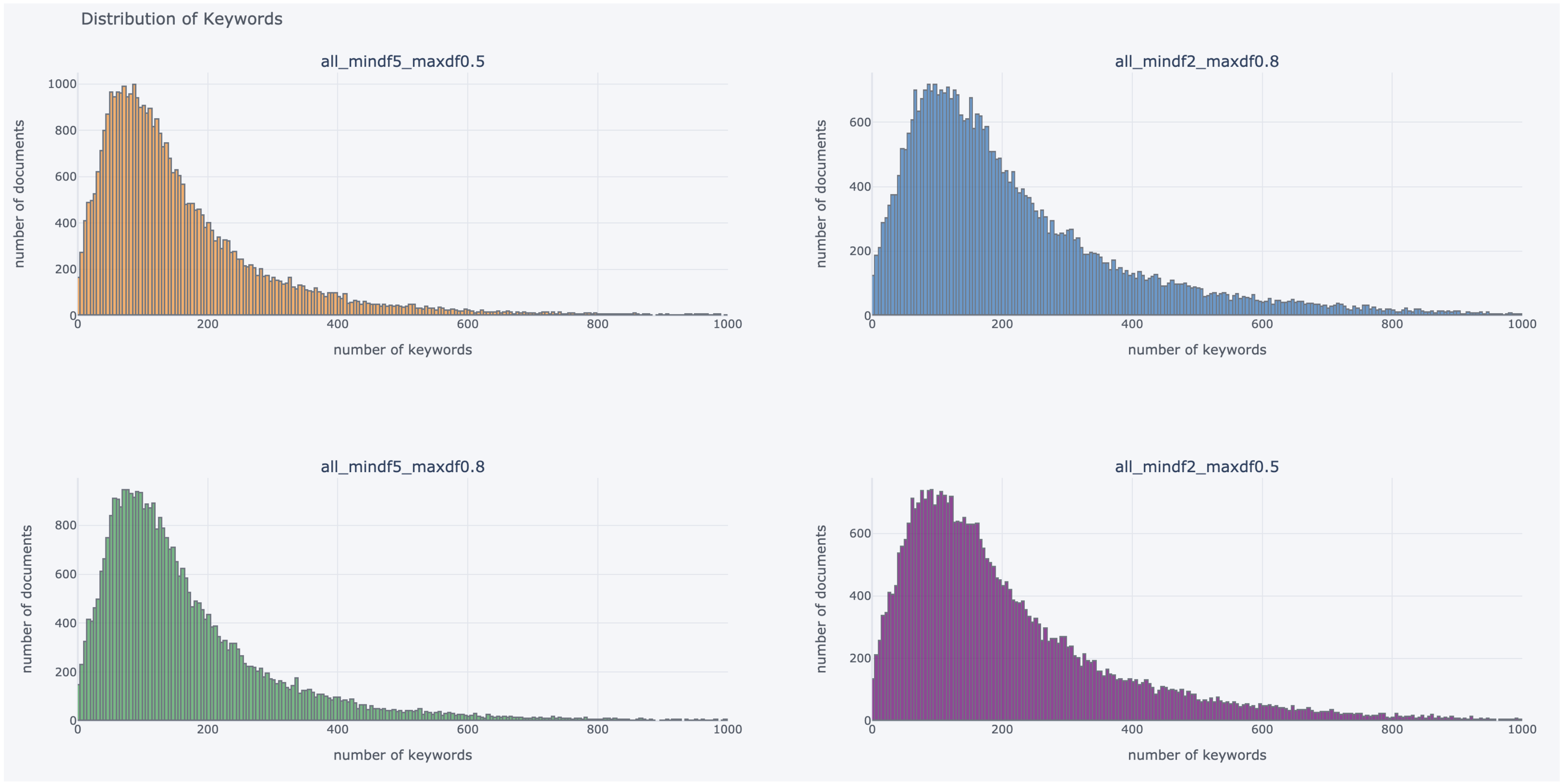


Figure 4.1: Comparison of distribution of keywords extracted using *n-gram* method ”all”. Bin size = 5. It can be observed that the min_{df} parameter affects the distribution, making a reduction of overall number of keywords per document. max_{df} controls the impact of very common words. The higher the max_{df}, the higher is overall number of keywords per document. To make the analysis easier, the number of keywords was clipped at 1000, therefore the last bar represents the number of documents with keywords ≥ 1000

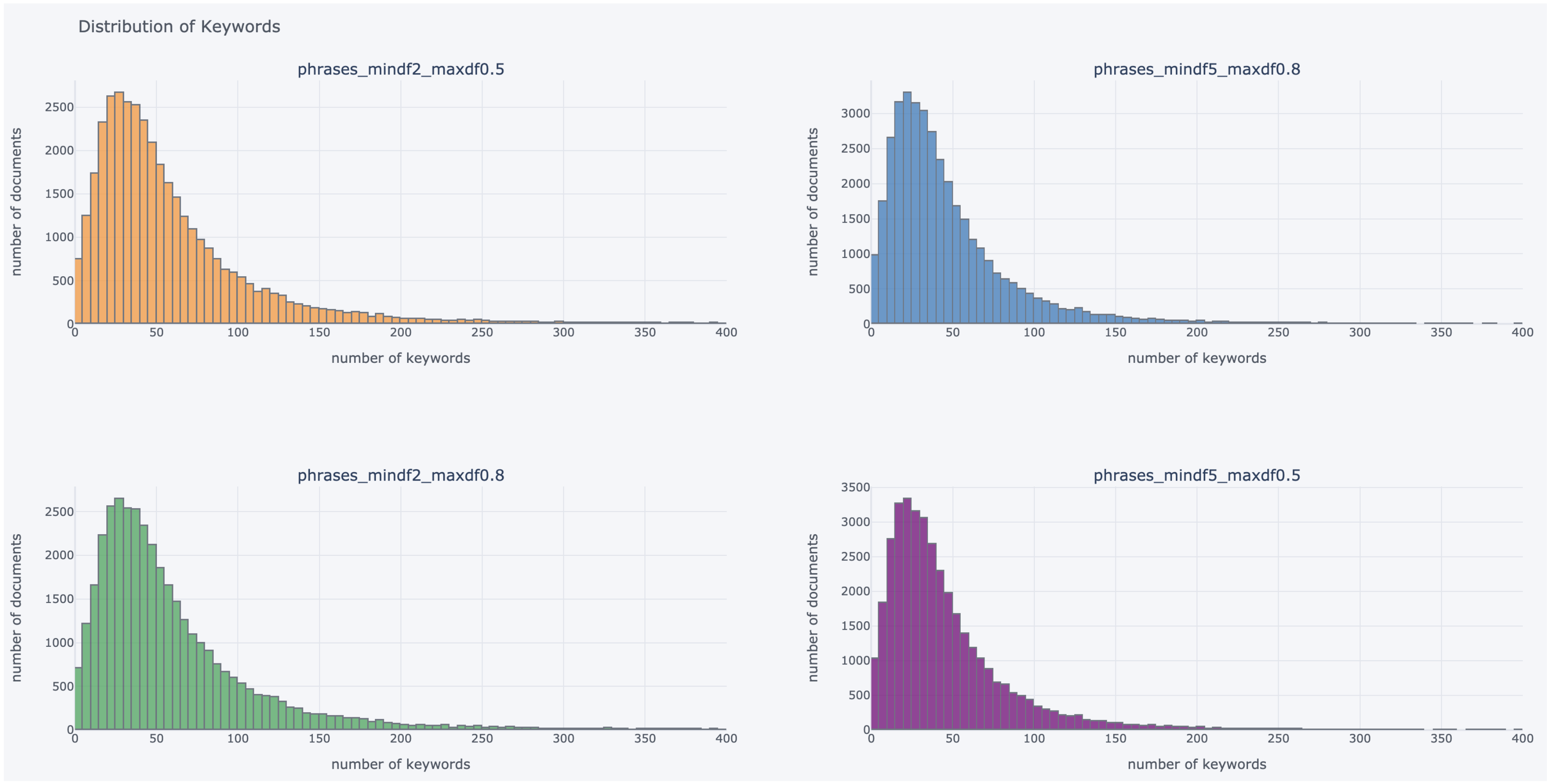


Figure 4.2: Comparison of distribution of keywords extracted using n-gram method ”phrases”. Bin size = 5. The effects are same observed for *n-grams* ”all”. The phrases method itself produces a massive reduction on number of keywords per document, as expected and observed by the distribution skewed to the left.. To make the analysis easier, the number of keywords was clipped at 400, therefore the last bar represents the number of documents with keywords ≥ 400

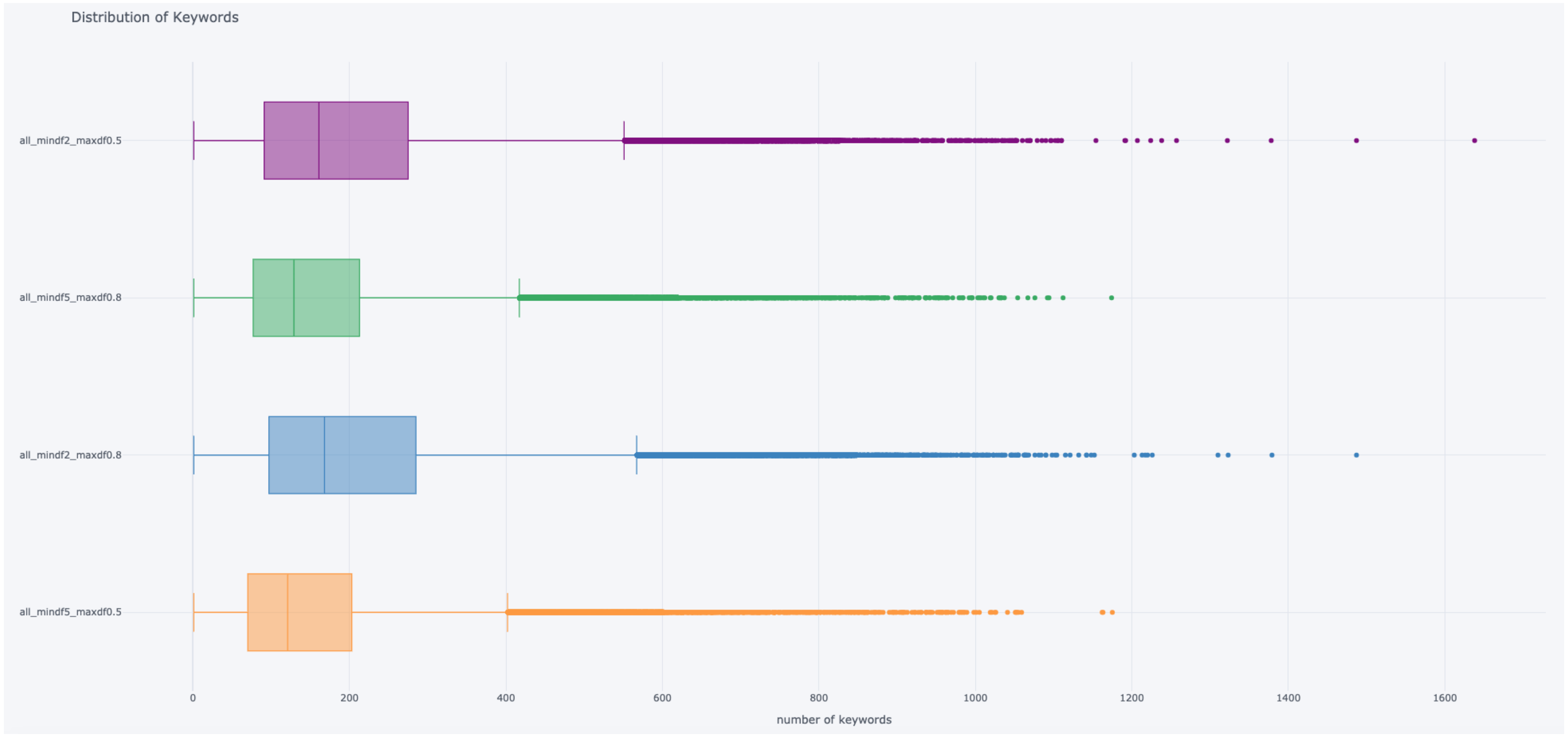


Figure 4.3: Comparison of distribution of keywords extracted using n-gram method ”all”. The observations made looking at the histogram are clearly confirmed. Increase on min_{df} skew the distribution, shifting the box to the left (less keywords per document) and making it narrower. max_{df} moves the box to the right (more keywords per document)

To continue the analysis of the first steps in document classification, let us observe the distributions of keywords after the expansion by semantic similarities. The parameters that control the expansion are *max similarities* and *w2v similarity*. It is intuitive that lower numbers for *max similarities* produce less augmentation, as all expanded terms could be easily rejected (it can be observed comparing expanded_0.7_5_all_mindf5_maxdf0.5 with expanded_0.7_3_all_mindf5_maxdf0.5 in Figure 4.4). On the other hand, a small *w2v similarity* allows for more words to match in the expansion (compare expanded_0.7_5_all_mindf5_maxdf0.5 with expanded_0.75_5_all_mindf5_maxdf0.5 in Figure 4.4). Therefore, one parameter controls the other. If similarity threshold is too low, many words can match on cosine similarity, and thus max similarities would make all of them be rejected. It is important to find a threshold where the similar words found are really related, and only bad encoding would result in a number of matches above it. Table 4.3 shows an overview of main values observed in the charts. For keywords *n-grams = 'all'* each row is followed by the values of its expanded values

Parameters	**Mean**	**Median**	**std**	**max**	**min**
keywords all, mindf5, maxdf0.5	158	121	133	1175	1
expanded w2v sim. 0.70, max sim. 5	205	156	173	1735	0
expanded w2v sim. 0.75 max sim. 5	182	139	152	1479	0
expanded w2v sim. 0.70, max sim. 3	190	145	161	1561	0
expanded w2v sim. 0.75, max sim. 3	176	135	148	1406	0
keywords all, mindf5 maxdf0.8	166	129	135	1174	1
expanded w2v sim. 0.70, max sim. 5	216	166	177	1621	0
expanded w2v sim. 0.75, max sim. 5	191	148	154	1374	0
keywords all, mindf2 maxdf0.5	209	161	169	1638	1
expanded w2v sim. 0.70, max sim. 5	256	196	208	1949	0
expanded w2v sim. 0.75, max sim. 5	233	179	188	1790	0
keywords all, mindf2 maxdf0.8	217	168	171	1487	1
keywords phrases, mindf5, maxdf0.5	46	35	41	755	0
keywords phrases, mindf5, maxdf0.8	46	36	41	755	0
keywords phrases, mindf2, maxdf0.5	59	44	56	1702	1
keywords phrases, mindf2, maxdf0.8	60	45	56	1703	1

Table 4.3: Median and rounded mean and std, maximum and minimum values for number of keywords per document in Internal Publications, followed by their expanded sets.

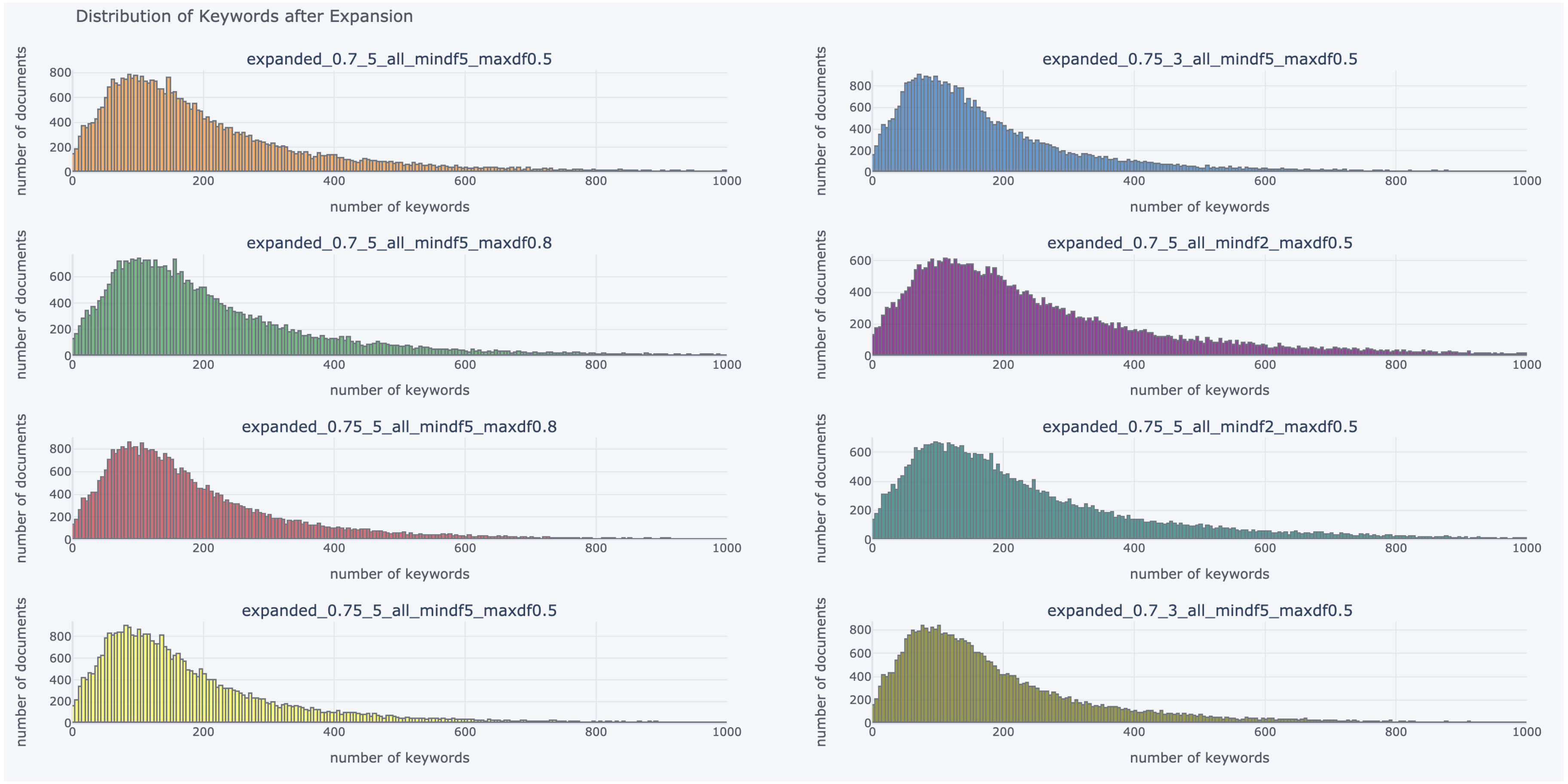


Figure 4.4: Comparison of distribution of keywords expanded for keywords extracted on *n-gram* method "all". Bin size=5. The number of keywords was clipped at 1000, therefore the last bar represents the number of documents with keywords ≥ 1000.

Finally, after having the keywords already expanded, the string match between them and the taxonomy topics is performed. For some different parameters settings the distribution of labels over the documents are displayed. On Table 4.4 one can notice that the parameters for keyword extraction and expansion other than *n-gram* method does not affect significantly the number of labels further assigned, specially for "phrases" method. For "all" method, it slightly changes the distribution.

Parameters	**Mean**	**Median**	**std**	**max**	**min**
w2vSim0.70, DL0.15, depth2, keys: all, mindf2, maxdf0.5	7	5	7	124	0
w2vSim0.70, DL0.15, depth2, keys: all, mindf2, maxdf0.8	7	5	7	105	0
w2vSim0.70, DL0.15, depth2, keys: all, mindf5, maxdf0.5	6	4	6	183	0
w2vSim0.70, DL0.15, depth2, keys: all, mindf5, maxdf0.8	7	5	6	177	0
w2vSim0.75, DL0.15, depth1, keys: all, mindf2, maxdf0.5	6	4	6	114	0
w2vSim0.75, DL0.15, depth1, keys: all, mindf2, maxdf0.8	6	5	6	100	0
w2vSim0.75, DL0.15, depth1, keys: all, mindf5, maxdf0.5	6	4	6	186	0
w2vSim0.75, DL0.15, depth1, keys: all, mindf5, maxdf0.8	6	5	6	160	0
w2vSim0.75, DL0.15, depth2, keys: all, mindf2, maxdf0.5	6	4	6	103	0
w2vSim0.75, DL0.15, depth2, keys: all, mindf2, maxdf0.8	6	4	6	87	0
w2vSim0.75, DL0.15, depth2, keys: all, mindf5, maxdf0.5	**5**	**4**	**5**	**162**	**0**
w2vSim0.75, DL0.15, depth2, keys: all, mindf5, maxdf0.8	5	4	6	145	0
w2vSim0.70, DL0.15, depth2, keys: phrases, mindf5, maxdf0.5	2	1	3	48	0
w2vSim0.70, DL0.15, depth2, keys: phrases, mindf5, maxdf0.8	2	1	3	46	0
w2vSim0.70, DL0.15, depth2, keys: phrases, mindf2, maxdf0.5	2	1	3	87	0
w2vSim0.70, DL0.15, depth2, keys: phrases, mindf2, maxdf0.8	2	1	3	85	0
w2vSim0.75, DL0.15, depth1, keys: phrases, mindf2, maxdf0.5	2	1	3	88	0
w2vSim0.75, DL0.15, depth1, keys: phrases, mindf2, maxdf0.8	2	1	3	86	0
w2vSim0.75, DL0.15, depth1, keys: phrases, mindf5, maxdf0.5	2	1	3	50	0
w2vSim0.75, DL0.15, depth1, keys: phrases, mindf5, maxdf0.8	2	1	3	49	0
w2vSim0.75, DL0.15, depth2, keys: phrases, mindf2, maxdf0.5	2	1	2	84	0
w2vSim0.75, DL0.15, depth2, keys: phrases, mindf2, maxdf0.8	2	1	2	82	0
w2vSim0.75, DL0.15, depth2, keys: phrases, mindf5, maxdf0.5	2	1	2	47	0
w2vSim0.75, DL0.15, depth2, keys: phrases, mindf5, maxdf0.8	2	1	2	47	0

Table 4.4: Median and rounded mean and std, maximum and minimum values for number of taxonomy labels assigned per document in Internal Publications. In Parameters column, the name indicates parameters: w2v similarity, Damerau Levenstein tolerance, taxonomy depth, followed by parameters used for keyword extraction - min document frequency and max document frequency proportion.

For a document level observation, 50 documents are sampled from the whole collection and plot the number of labels per document, for the same parameters. Figures 4.5 and 4.6 exhibit these charts. Roughly, with all *n-grams*, the amount of tags are twice as high compared to "phrases". Considering the tags are resulted from string matching with human-curated set of terms, it is worthy using "all" method.

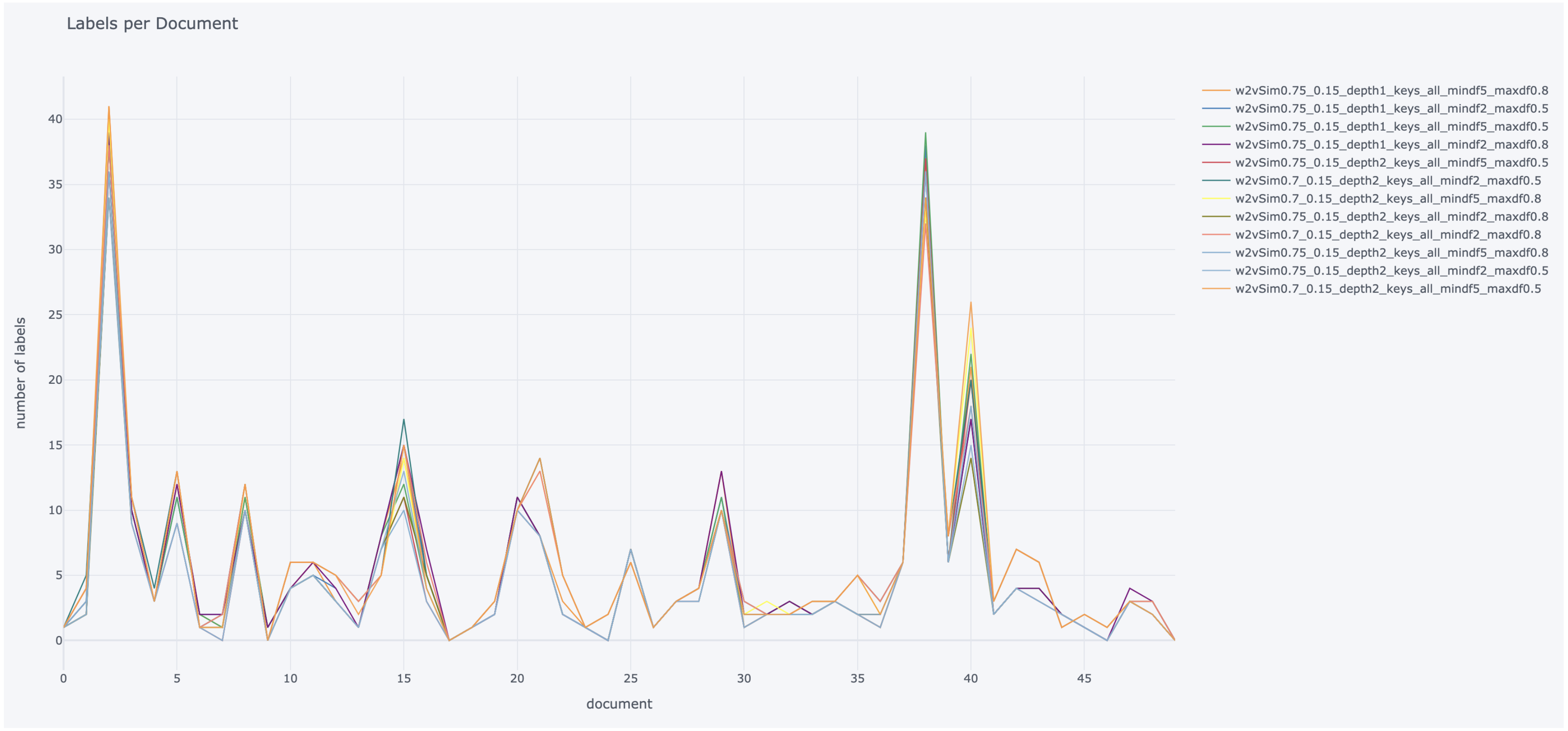


Figure 4.5: Comparison of labels assigned per documents on 50 samples. ”all” *n-grams* in original keywords.

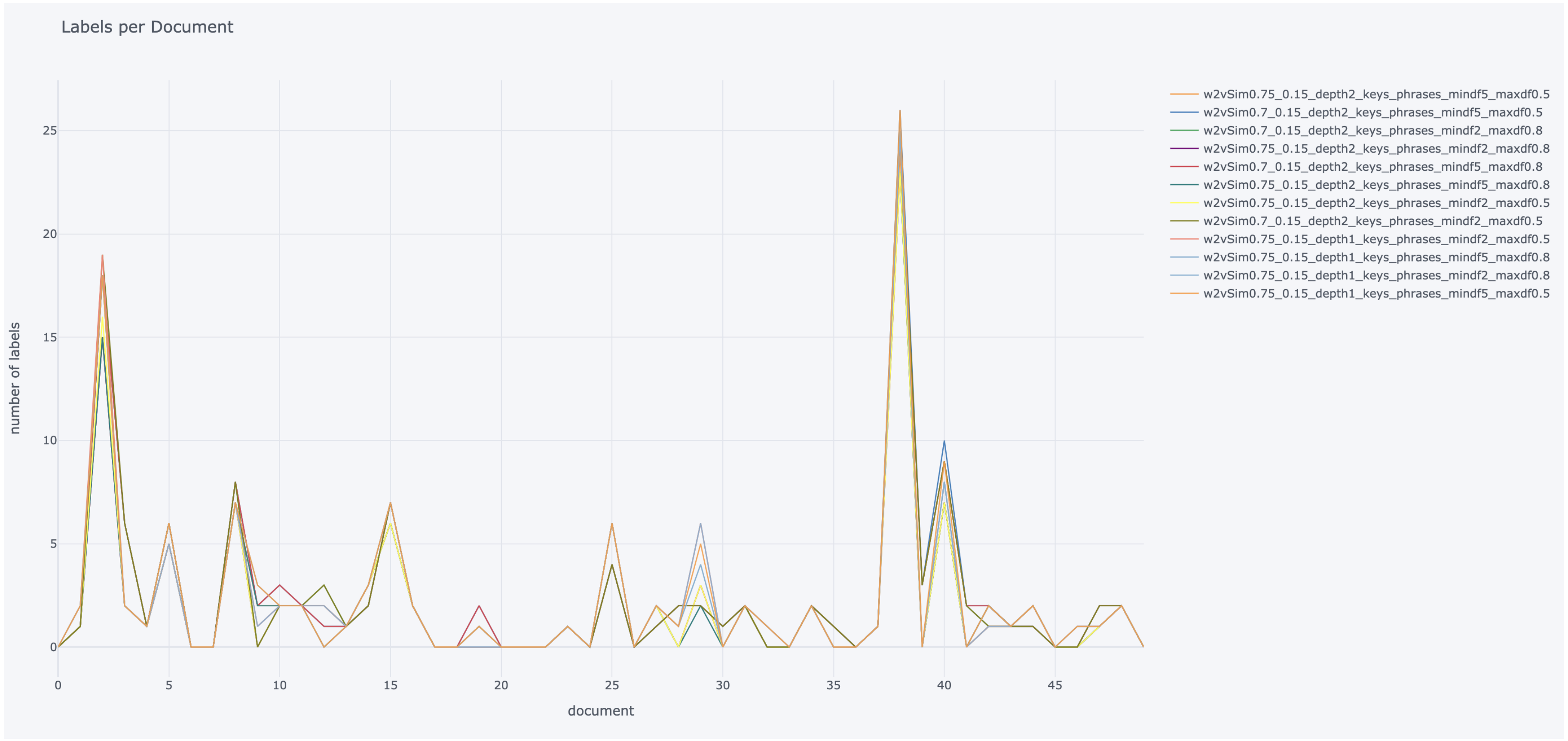


Figure 4.6: Comparison of labels assigned per documents documents on 50 samples. ”phrases” *n-grams* in original keywords.

4.4 Previous Work Assignments Classification

To classify the documents related to Previous Work Assignments (also known as ”engagements”), the steps are similar as the ones aforementioned, except by the keyword extraction. Firstly, the Primary Terms are taken as the keywords, then the expansion with word2vec is performed, followed by the string match. Besides that, the semantic match between the Primary Terms and Taxonomy topics is computed, using Flair to embed the terms and the topics. Figures 4.7 and 4.8 show the distribution of keywords on this collection. Note that this is the distribution of topics taken straight from the raw data.

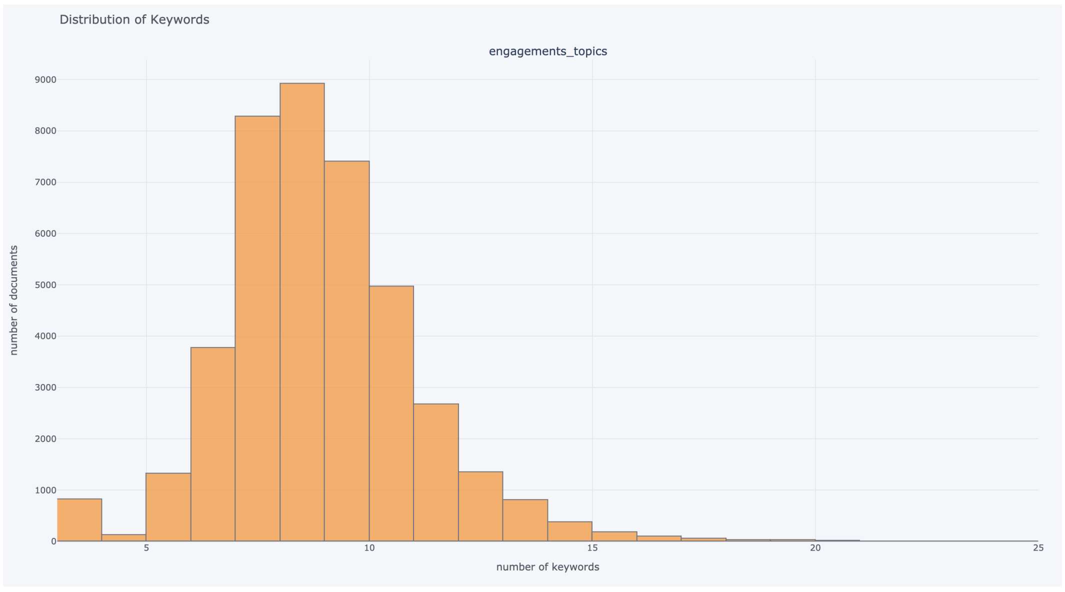


Figure 4.7: Distribution of keywords on Work Assignments collection. Bin size = 1. First bin = 3.

Now, let us observe the keywords expansion results for some parameters. This step takes the Primary Terms as keywords and use the vocabulary and vectors from word2vec trained in Internal Publications. Basically for each term, it checks if its embeddings exist in the vocabulary, and if yes, add the most similar words to the term, expanding the set for future string match. Figure 4.9 shows the distributions and Table 4.5 summarizes the original Primary Topics and the effects of expansion using different parameters.

Finally, let us observe the impact of the different parameters tested previously in the matches with Taxonomy terms. The string matching is done with allowing for Damerau-Levenstein distance of 0.15% of string length. Each match results in a final labels assigned to the document. Figures 4.10, 4.11 and 4.12 display the results for standard method used till this point. Table 4.6 summarizes the

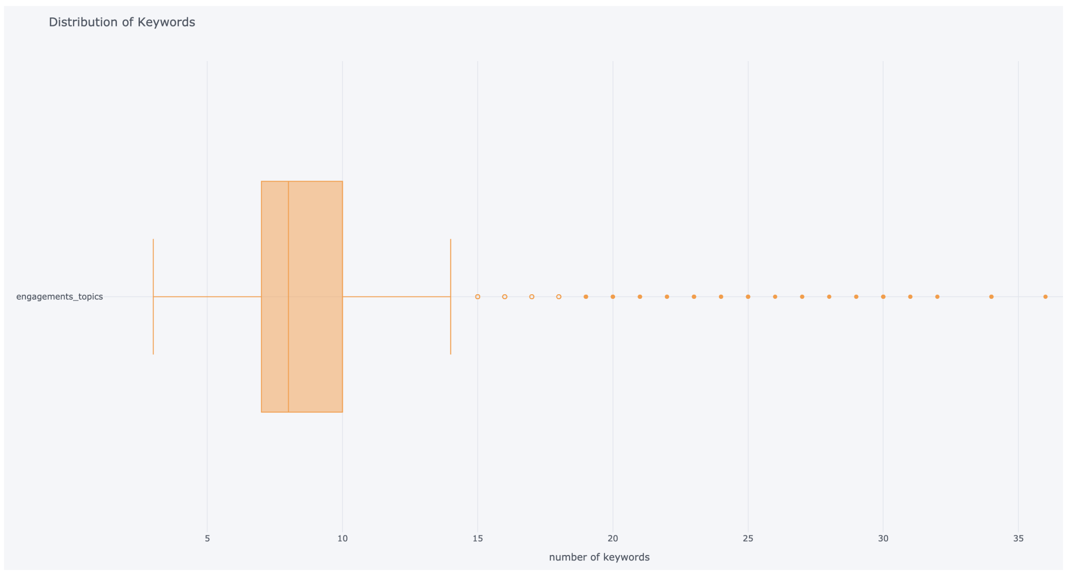


Figure 4.8: Distribution of keywords on Work Assignments collection.

Parameters	Mean	Median	std	max	min
Work Assignments Primary Topics	8	8	2	36	3
expanded w2v sim. 0.80, max sim. 5	8	8	3	39	2
expanded w2v sim. 0.75, max sim. 5	9	9	3	43	2
expanded w2v sim. 0.70, max sim. 5	10	10	4	59	2
expanded w2v sim. 0.65, max sim. 5	14	13	5	65	2

Table 4.5: Median and rounded mean and std, maximum and minimum values for number of keywords per document in Previous Work Assignments, followed by their expanded sets.

main values observed in the experiments. Later on, the quantitative analysis for semantic match with Flair embeddings will be also displayed.

Parameters	Mean	Median	std	max	min
w2vSim 0.80, DL 0.15, depth1	5	5	2	30	0
w2vSim 0.75, DL 0.15, depth1	5	5	2	32	0
w2vSim 0.70, DL 0.15, depth1	5	5	2	31	0
w2vSim 0.65, DL 0.15, depth1	6	6	2	35	0
w2vSim 0.80, DL 0.15, depth2	3	3	2	26	0
w2vSim 0.75, DL 0.15, depth2	**3**	**3**	**2**	**26**	**0**
w2vSim 0.70, DL 0.15, depth2	3	3	2	27	0
w2vSim 0.65, DL 0.15, depth2	4	3	2	30	0

Table 4.6: Median and rounded mean and std, maximum and minimum values for number of labels assigned per document in Previous Work Assignments for some parameter values.

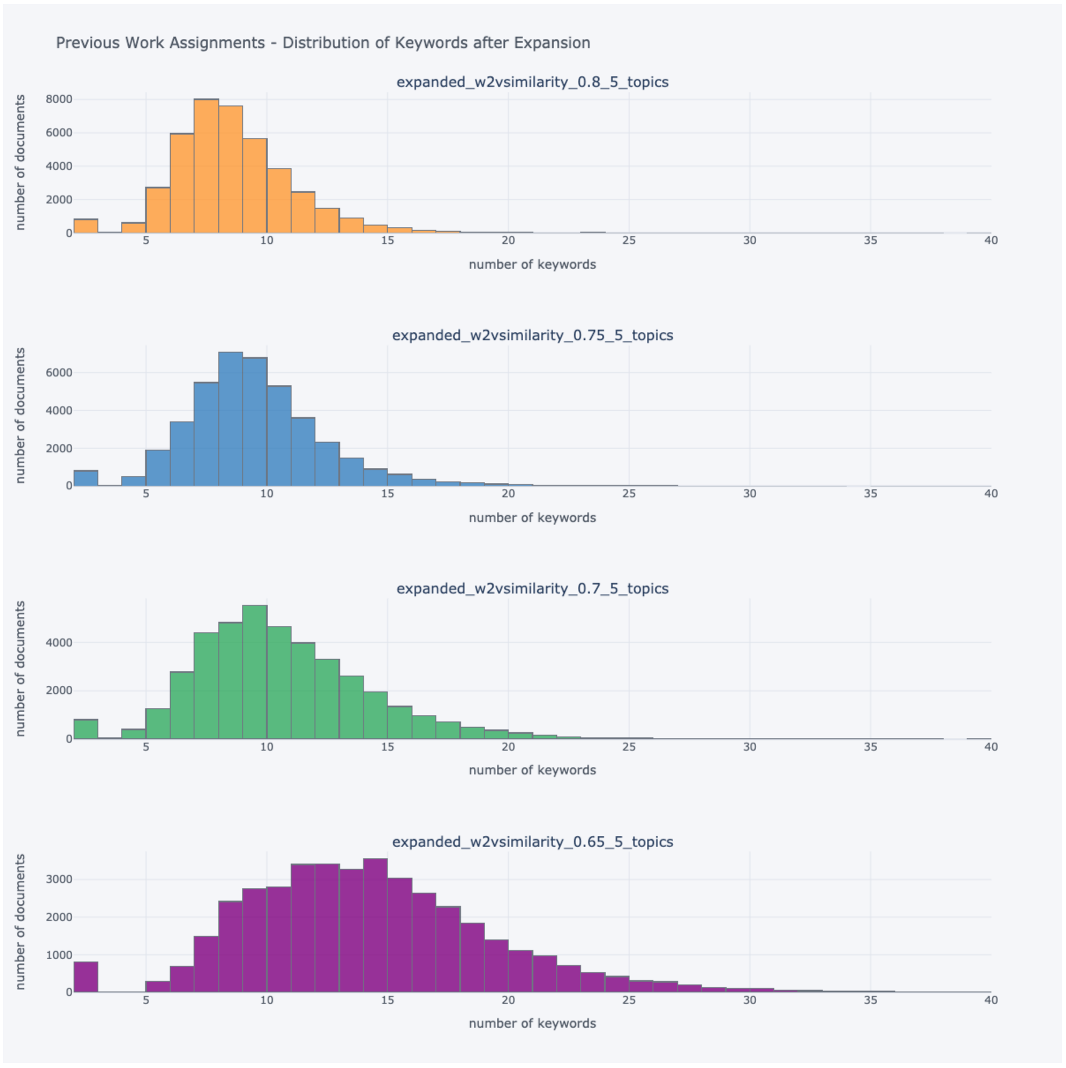


Figure 4.9: Distribution of keywords expanded on Work Assignments collection. Bin size = 1. First bin = 2. The reduction on minimum cosine similarity parameter allows more words to be aggregated, but they are less similar.

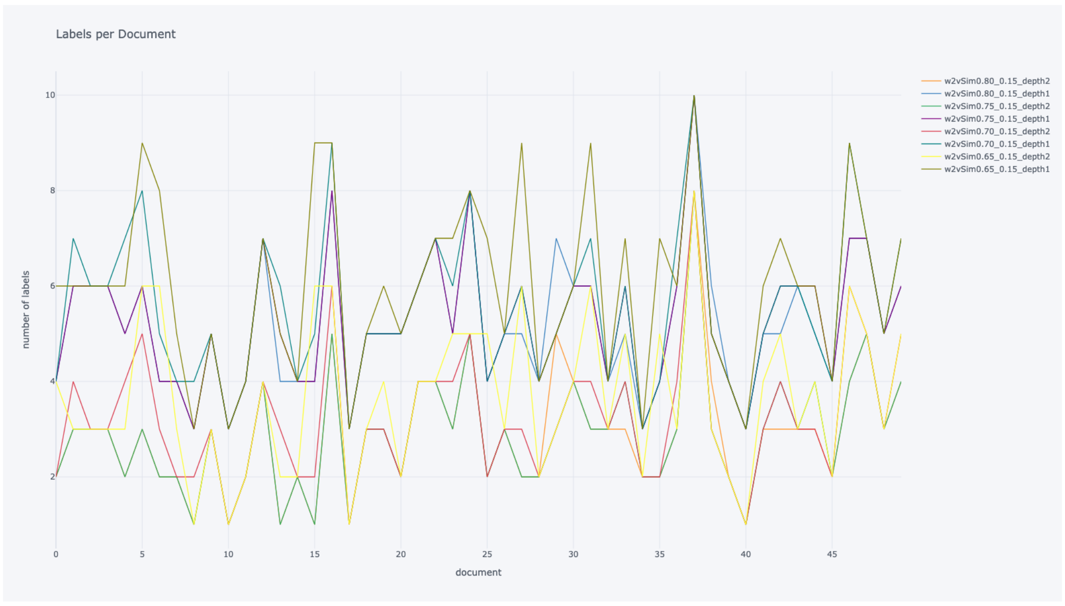


Figure 4.10: Comparison of labels assigned per documents on 50 samples. Labels assigned to keywords expanded varying w2v similarity and matching with Damerau-Levenstein tolerance of 15% of string length. Again, it is interesting to observe, that although a lower cosine similarity on w2v brings more similar words during the expansion, the $max_{similarities} = 5$ would remove them. This is clearly observed as the final number of labels assigned can be higher for some documents that had keywords expanded with higher cosine similarity value. The depth parameter changes the size of set of possible matches, therefore depth 1 always gives more labels than depth 2, for same similarity parameters

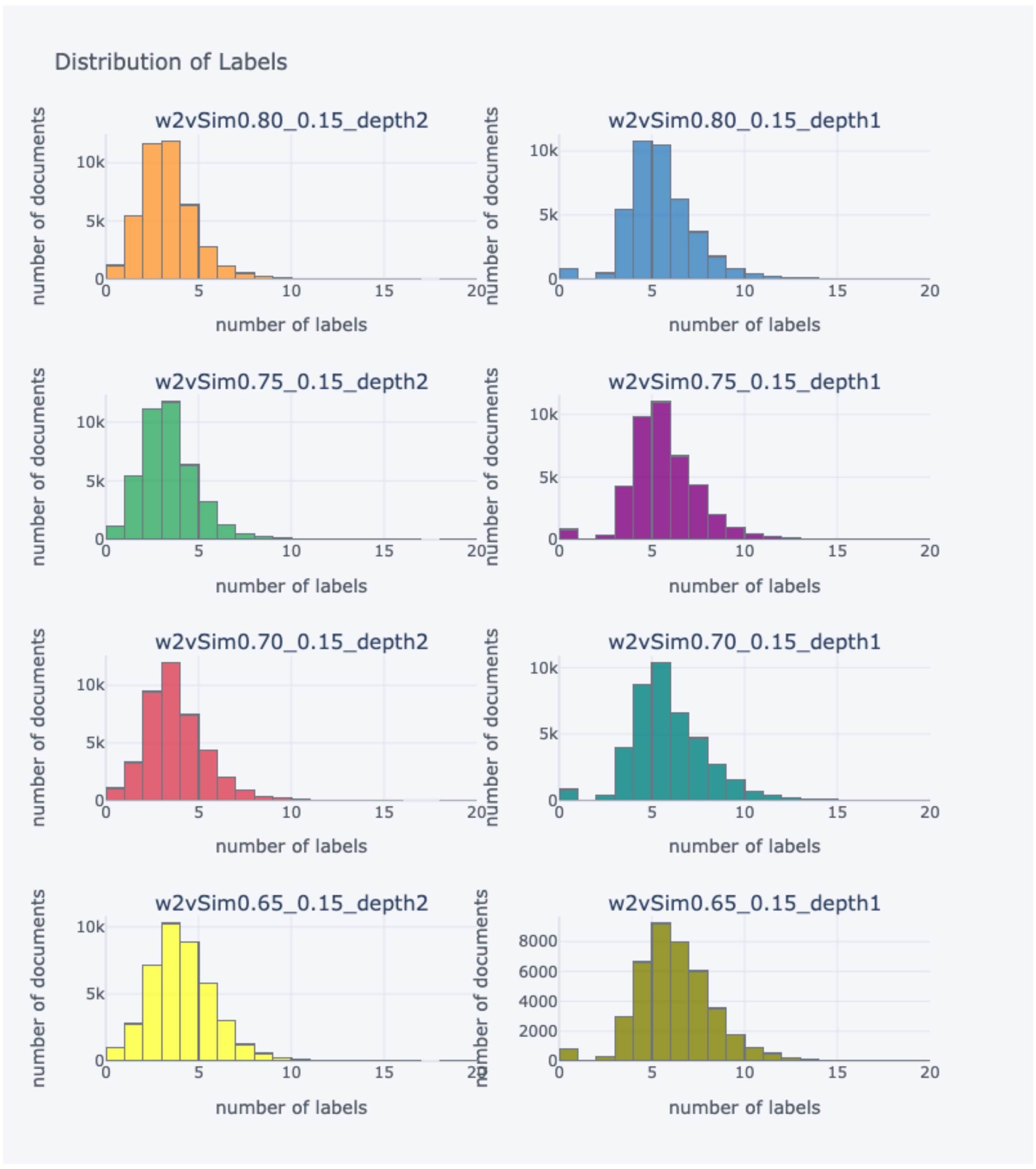


Figure 4.11: Comparison of labels assigned for documents, for the various w2v similarity parameters used in keyword expansion step. It is clear that for similarity in the range 0.7 to 0.8, the distribution does not change much, given same taxonomy depth.However, for values lower than 0.7 it can be expected that the model adds noisy labels.

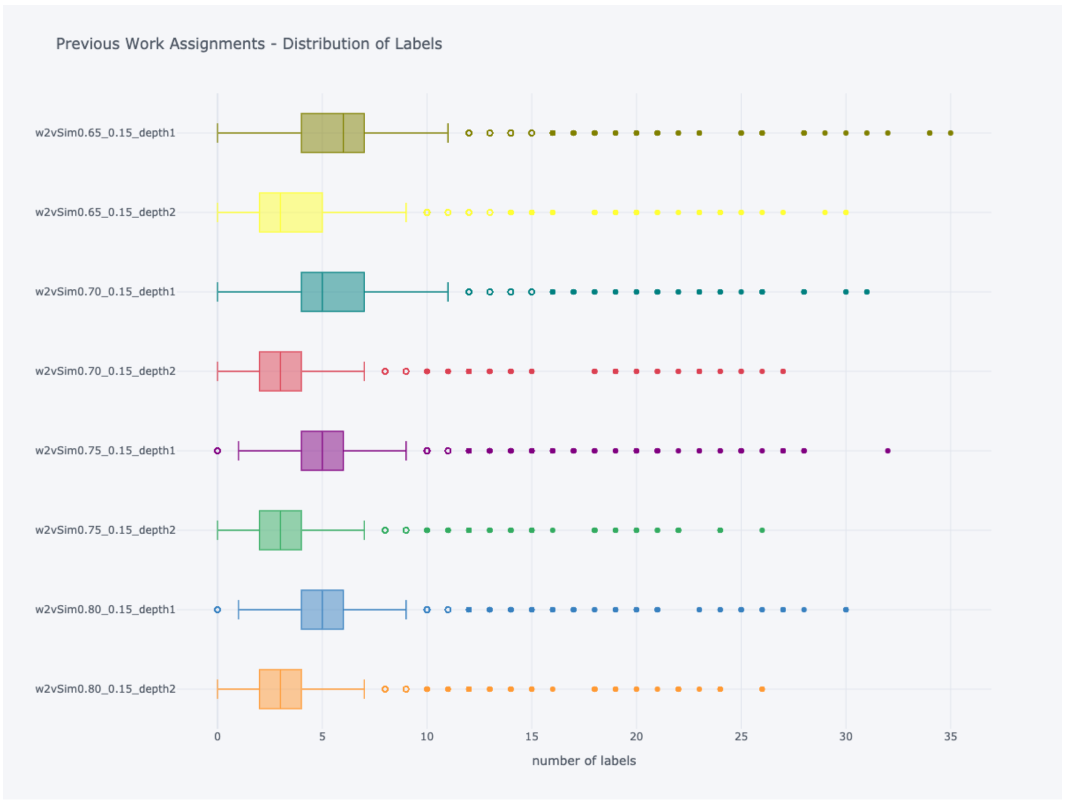


Figure 4.12: Distribution of labels assigned for documents, for the various w2v similarity parameters used in keyword expansion step.

Flair Embeddings

Flair embeddings, as mentioned in Chapter 3, is an optional encoder used to embed *n-grams*. Each language model, forward and backward, obtained after the transfer learning process, has a hidden state of 2048 dimensions. Therefore, the stacked embeddings used encodes the word in a 4096-dimension vector. For the embedding of *n-grams* or sentences, the resulting dimensions are:

- Pool: encode each of the words part of *n gram* or sentence, and take the average vector. Thus, the resulting vector has also 4096 dimension.
- RNN: encode each of the words and use the vectors in another recurrent neural network layer. Then take the hidden state of the last cell as the embedding of the *n gram* or sentence. The resulting vector dimension depends on the hidden state size, in this work it was set to 512. LSTM cells are chosen for the RNN method.

For assigning the labels using Flair embeddings the steps are:

- **Embed Primary Terms and Taxonomy Topics.**
- **Auto filtering of taxonomy topics**. This is made by checking how many taxonomy terms are similar to each other using cosine similarity. Words with too many similar words are removed, following the same logic as used on keyword expansion. This intends to avoid that n-grams with inaccurate encoding match with many others. A parameter $maxsimilar$ defines the ***limit*** of how many words are to be accepted.
- **Measure cosine similarity between all Primary Terms and all Taxonomy Topics and assign topics with similarity higher than threshold to the document.** There are two similarity thresholds: $max_{similarity}$ defines a high threshold which all terms above it are taken. $min_{similarity}$ defines the minimum similarity for a topic to be considered. If no topic has higher similarity than $max_{similarity}$ then the best match above $min_{similarity}$ is taken.

Figures 4.13, 4.15 and 4.14 show the effects of parameter choices during the assignment of topics using the method described above. Note how $max_{similarity}$ affects the variance of number of topics per document, as all topics matching higher than this number is added. The $min_{similarity}$ acts "shifting" the median, clearly observed in Figure 4.15. By observing these charts, and a careful inspection of the quality of topics added for some of the documents, values 0.90 and 0.80 for max and min similarities seemed to be a good compromise between number of matches and their quality. Table 4.7 summarizes the results using Flair embeddings approach.

Parameters	**Mean**	**Median**	**std**	**max**	**min**
flair: max sim. 0.95, min sim. 0.85, limit 5, pool	3	3	2	29	0
flair: max sim. 0.95, min sim. 0.80, limit 5, pool	**5**	**5**	**2**	**33**	**0**
flair: max sim. 0.90, min sim. 0.85, limit 5, pool	4	4	2	35	0
flair: max sim. 0.90, min sim. 0.80, limit 5, pool	6	6	2	39	0
flair: max sim. 0.90, min sim. 0.80, limit 10, pool	6	6	2	39	0
flair: max sim. 0.90, min sim. 0.75, limit 5, pool	7	7	2	40	0
flair: max sim. 0.90, min sim. 0.70, limit 5, pool	7	7	3	40	0
flair: max sim. 0.90, min sim. 0.65, limit 5, pool	8	8	3	41	0
flair: max sim. 0.90, min sim. 0.60, limit 5, pool	8	8	3	41	0
flair: max sim. 0.90, min sim. 0.75, limit 5, RNN	7	7	3	44	0
flair: max sim. 0.90, min sim. 0.70, limit 5, RNN	8	7	3	43	0
flair: max sim. 0.90, min sim. 0.65, limit 5, RNN	8	8	3	45	0
flair: max sim. 0.90, min sim. 0.60, limit 5, RNN	8	8	3	44	0

Table 4.7: Median and rounded mean and std, maximum and minimum values for number of labels assigned per document in Previous Work Assignments using different parameters for Flair method. Suggested settings is highlighted in the table.

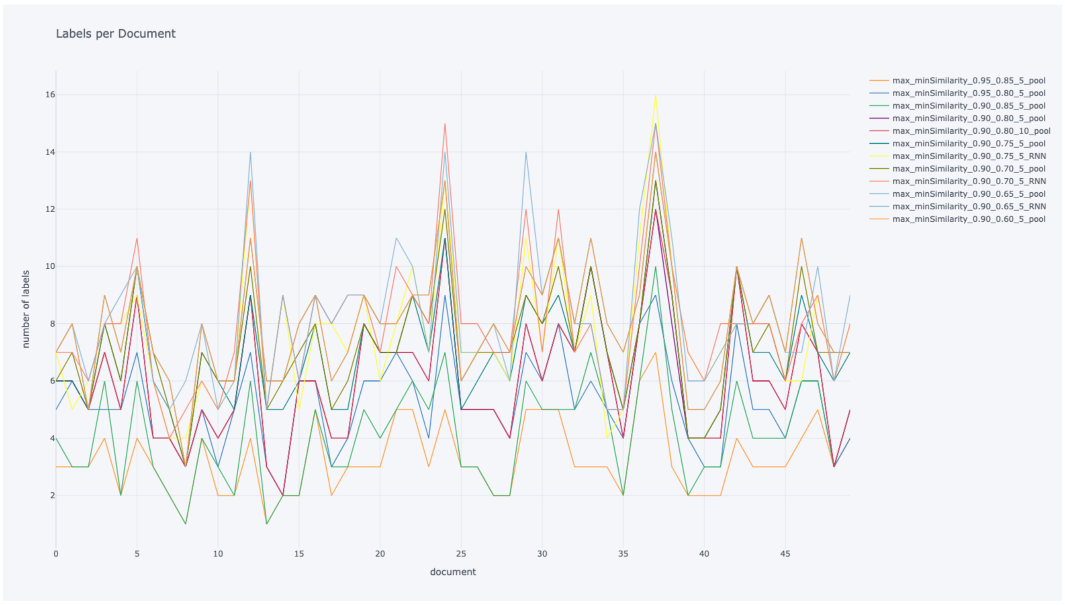


Figure 4.13: Comparison of labels assigned per documents on 50 samples. All plots used taxonomy topics at minimum depth 2. The chart presents different values for max and min similarities, RNN and Pool methods, and max similar words 5 and 10. RNN method produces more matches compared to Pool, but with more noise. When manually inspected, Pool method showed to be more reliable. As the method is total unsupervised, the best parameter value are chosen by expert inspection of sampled results.

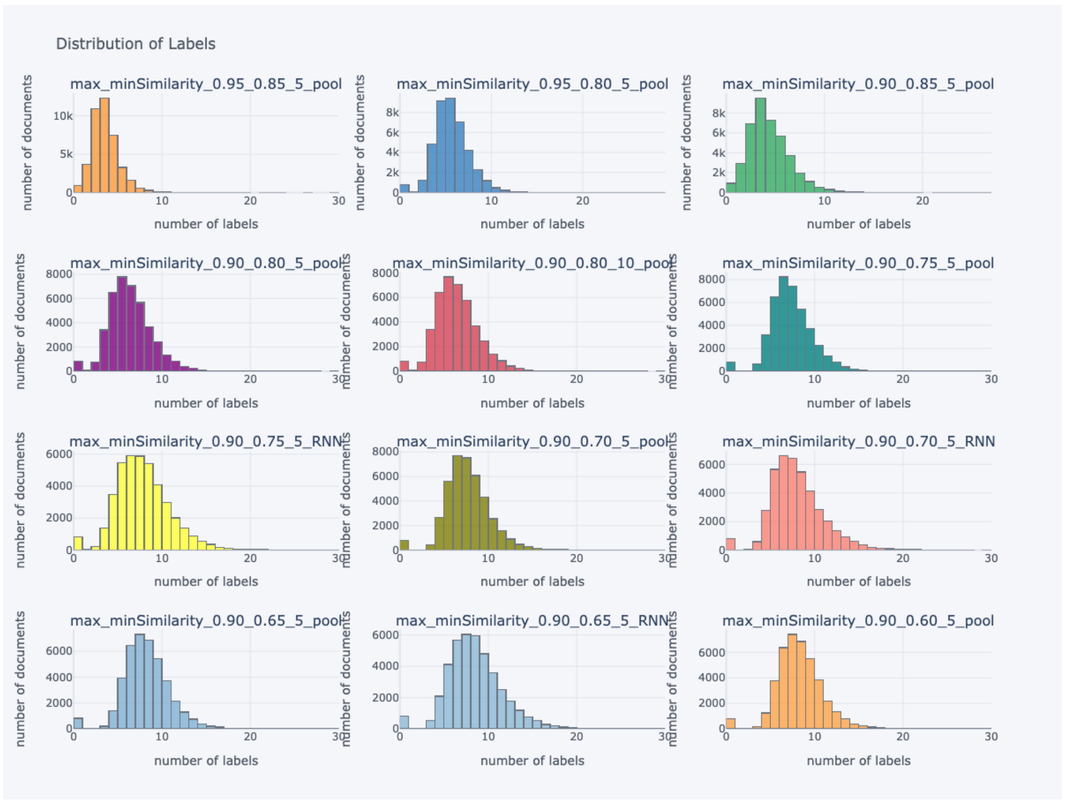


Figure 4.14: Histogram with distributions of number of labels assigned by semantic matching using flair embeddings, for different parameter settings.

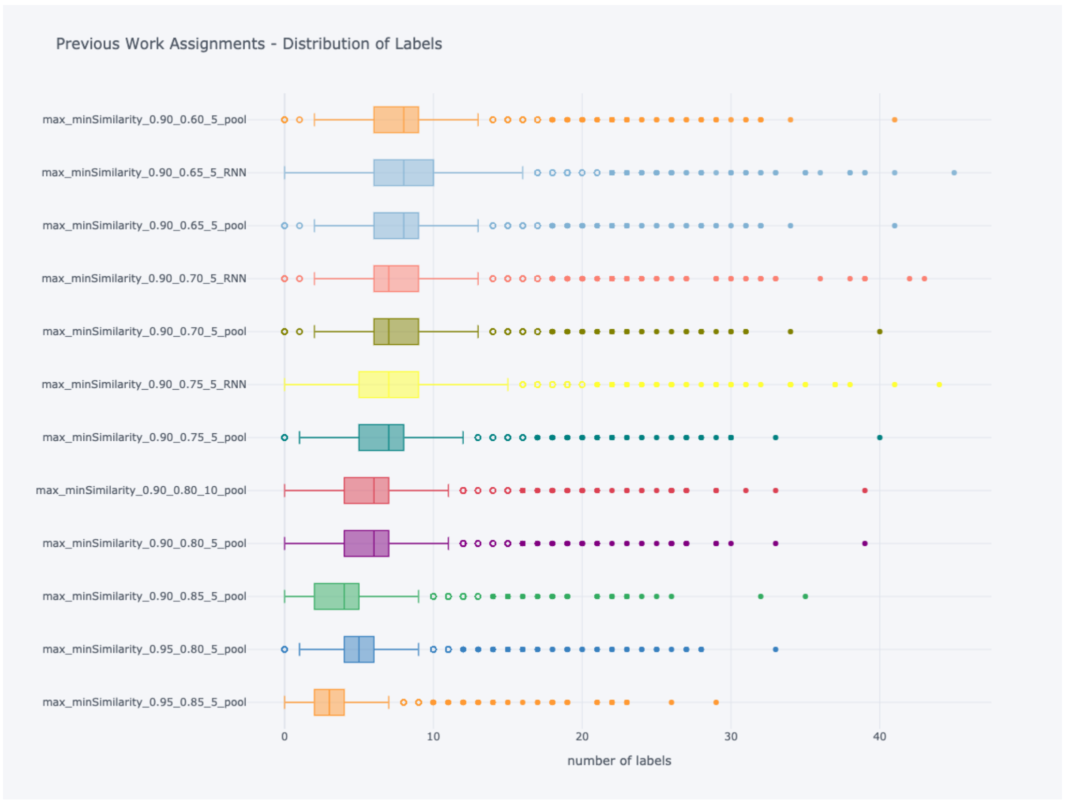


Figure 4.15: Distribution of number of labels assigned by semantic matching using flair embeddings, for different parameter settings.

4.5 Classification of ”Topics I Know” in Profiles

There are 119449 profile topics, being 27447 of them unique. The target is to match them with Taxonomy Topics and each match become a label for the user. The methods used here to assign the labels for people profiles are exactly the same explained in the previous session (Work Assignments Classification). Each topic in a person’s profile is taken as a ”keyword” and all the other steps remain the same, for both standard and Flair methods. The following figures show the overview of the data and the results at each step. For Flair method, all runs had taxonomy minimum depth = 2.

First, let us observe the original set and some results for expansion step. Note that users with no topics in their profile were removed. Figures 4.16, and 4.17 shows the results. Table 4.8 summarizes the the most important values.

Parameters	**Mean**	**Median**	**std**	**max**	**min**
”Topics I Know”	11	9	10	162	1
expanded w2v sim. 0.80, max sim. 5	12	10	12	174	1
expanded w2v sim. 0.75, max sim. 5	**14**	**11**	**13**	**196**	**1**
expanded w2v sim. 0.70, max sim. 5	16	12	15	214	1
expanded w2v sim. 0.65, max sim. 5	17	14	16	236	1

Table 4.8: Median and rounded mean and std, maximum and minimum values for number of topics per user in ”Topics I Know” profile section, followed by their expanded sets using some different parameter values.

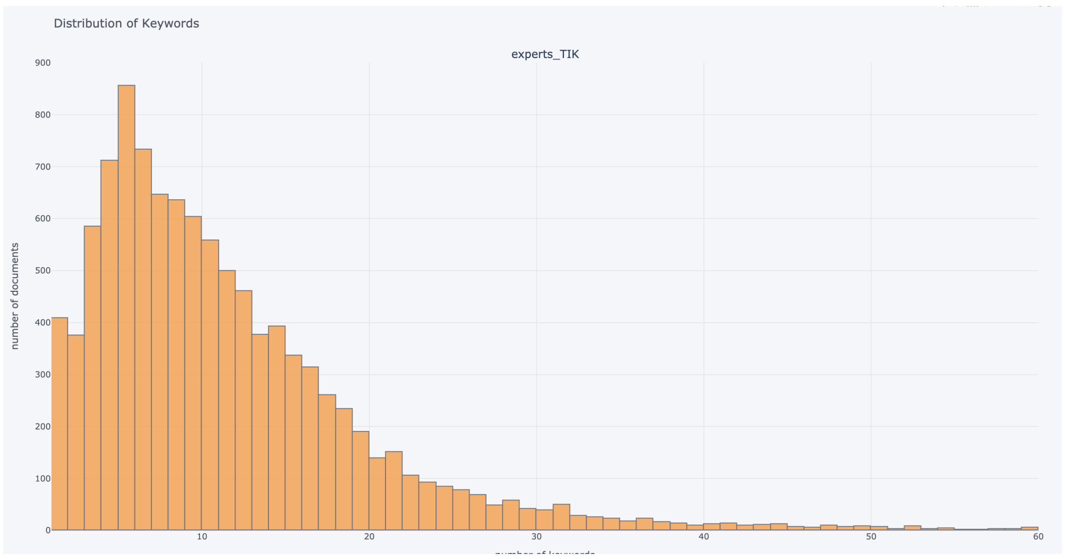


Figure 4.16: Distribution of original topics in people’s profiles. Bin size = 1. First bin = 1. Last bin for all profiles with more than 59 topics

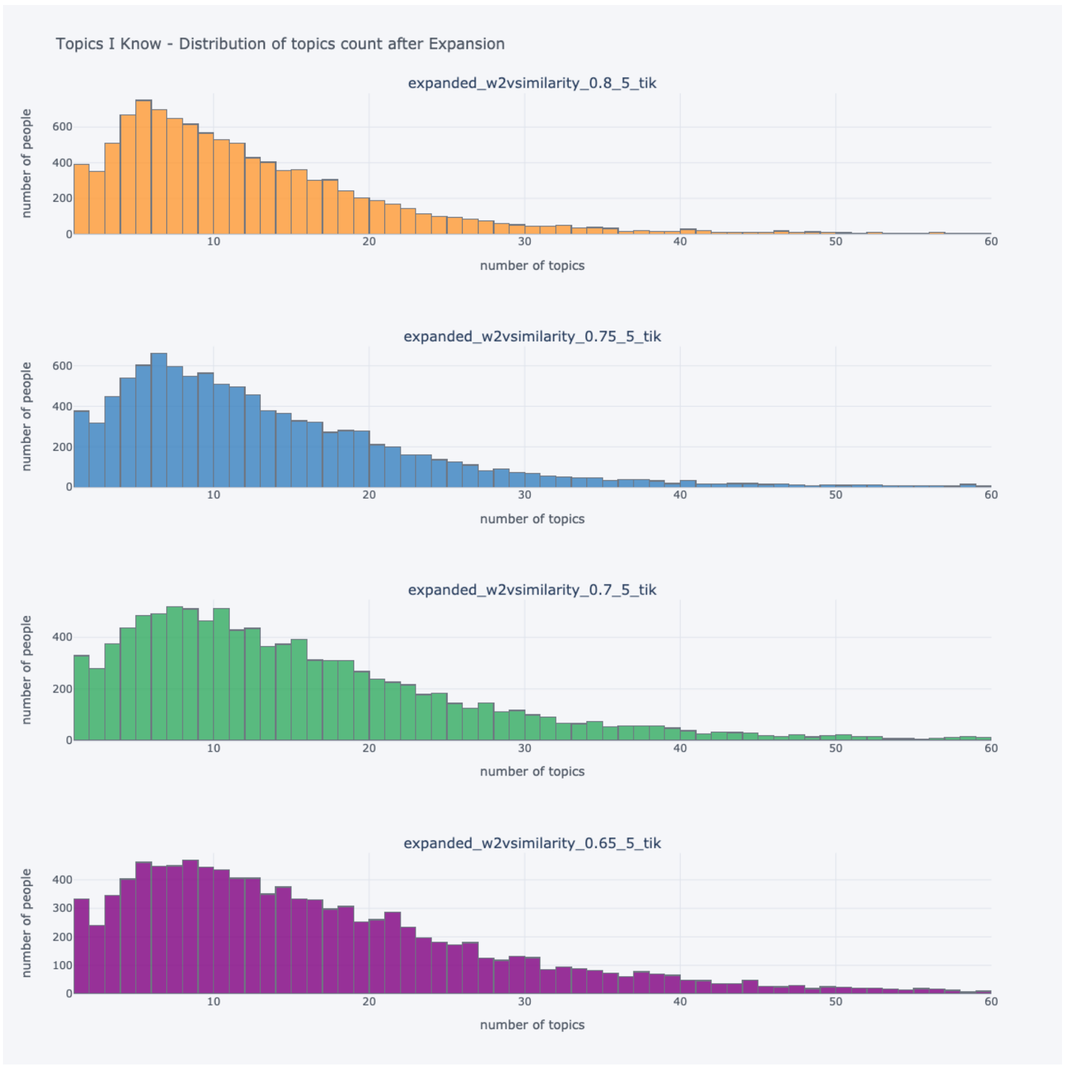


Figure 4.17: Comparison of distribution of topics from people's profiles after the expansion with different parameters. Bin size = 1. First bin = 1. Last bin for all profiles with more than 59 topics

Now, let us compare quantitatively the differences in labels assigned for the different parameters, based on standard method. Table 4.9 summarizes the most important values.

Finally, Figures 4.18, 4.19 and 4.20 show results for different parameter settings using Flair method. Table 4.10 summarizes the main values observed.

Parameters	**Mean**	**Median**	**std**	**max**	**min**
w2vSim 0.80, DL 0.15, depth1	3	2	4	61	0
w2vSim 0.75, DL 0.15, depth1	4	3	4	62	0
w2vSim 0.70, DL 0.15, depth1	4	3	4	64	0
w2vSim 0.65, DL 0.15, depth1	4	3	5	70	0
w2vSim 0.80, DL 0.15, depth2	3	2	4	57	0
w2vSim 0.75, DL 0.15, depth2	**3**	**2**	**4**	**58**	**0**
w2vSim 0.70, DL 0.15, depth2	3	2	4	60	0
w2vSim 0.65, DL 0.15, depth2	4	3	4	66	0

Table 4.9: Median and rounded mean and std, maximum and minimum values for number of labels assigned per user, related to their ”Topics I Know” session in profile, through standard method.

Parameters	**Mean**	**Median**	**std**	**max**	**min**
flair: max sim. 0.95, min sim. 0.85, limit 5, pool	4	3	4	60	0
flair: max sim. 0.95, min sim. 0.85, limit 7, pool	4	3	4	60	0
flair: max sim. 0.90, min sim. 0.80, limit 5, pool	**6**	**5**	**6**	**101**	**0**
flair: max sim. 0.90, min sim. 0.75, limit 5, pool	6	5	6	92	0
flair: max sim. 0.90, min sim. 0.70, limit 5, pool	8	7	7	101	0
flair: max sim. 0.90, min sim. 0.65, limit 5, pool	10	8	9	145	0
flair: max sim. 0.90, min sim. 0.60, limit 5, pool	10	8	9	146	0
flair: max sim. 0.90, min sim. 0.75, limit 5, RNN	5	4	5	60	0
flair: max sim. 0.90, min sim. 0.70, limit 5, RNN	9	7	8	107	0
flair: max sim. 0.90, min sim. 0.65, limit 5, RNN	10	8	9	124	0
flair: max sim. 0.90, min sim. 0.60, limit 5, RNN	10	8	9	129	0

Table 4.10: Median and rounded mean and std, maximum and minimum values for number of labels assigned per user for ”Topics I Know” session in profiles, using Flair method.

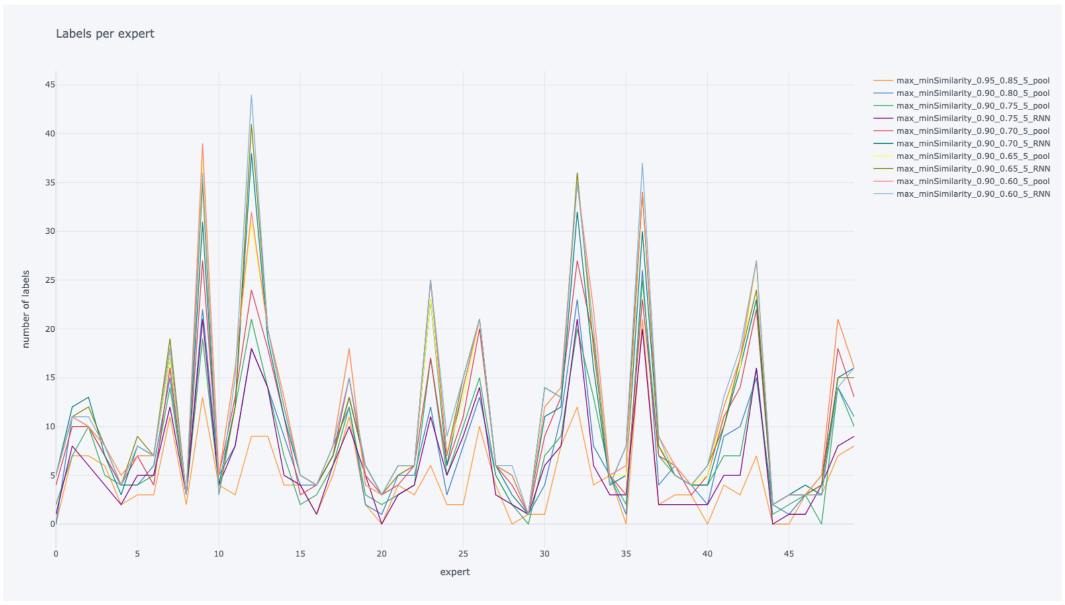


Figure 4.18: Comparison of labels assigned per user on 50 samples. Labels assigned to the topics using different values for $max_{similarity}$ and $min_{similarity}$

.

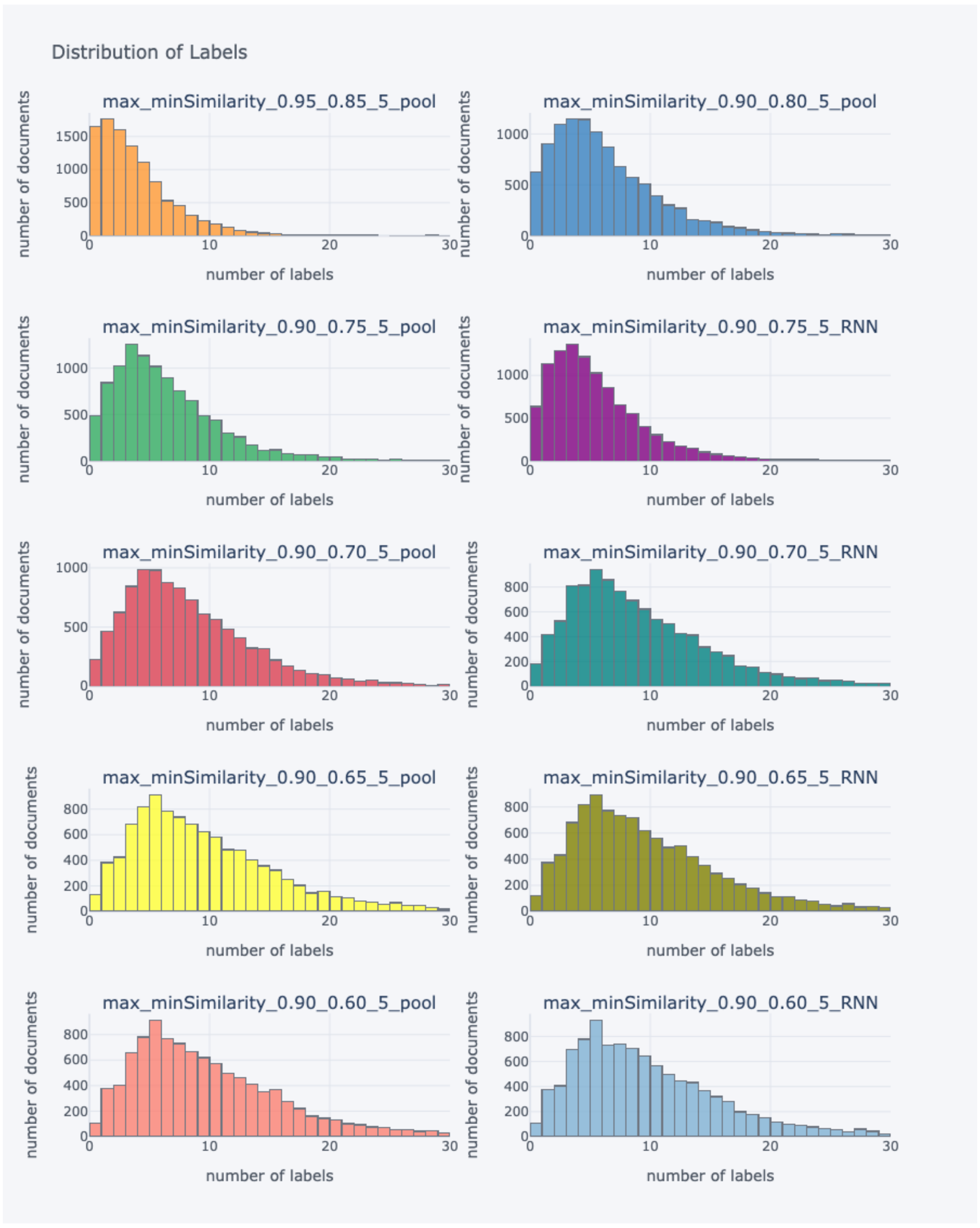


Figure 4.19: Distribution of labels assigned for documents, for different values for $max_{similarity}$ and $min_{similarity}$

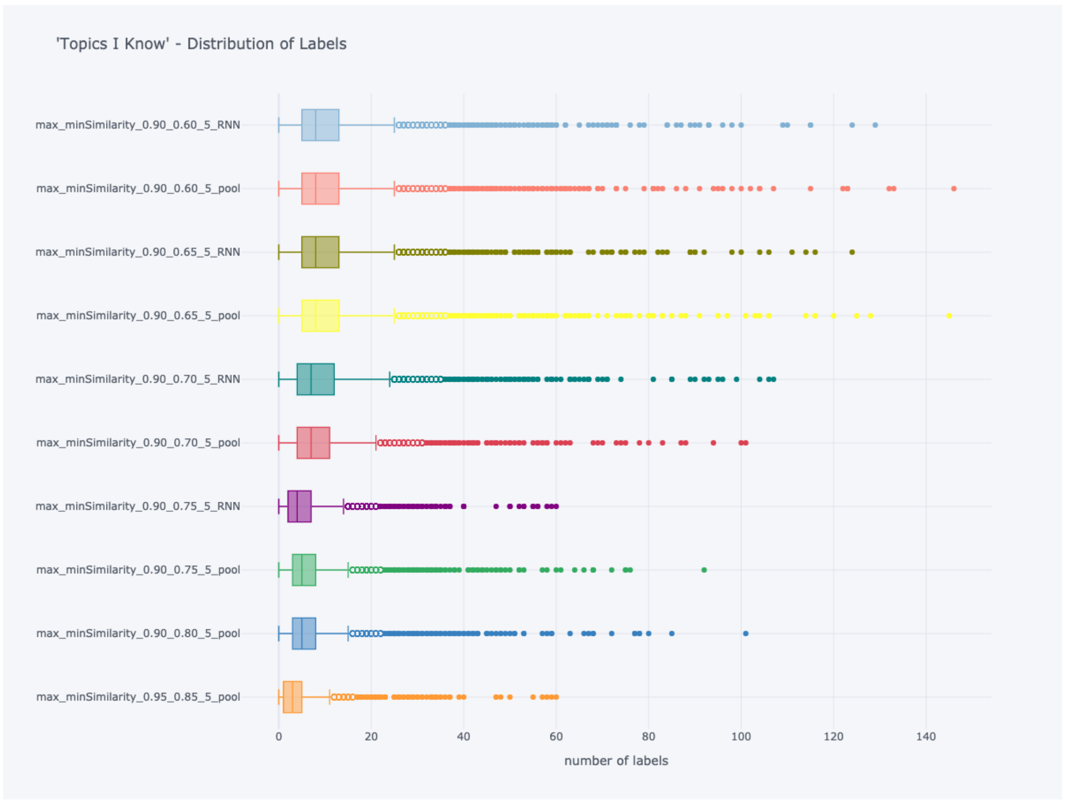


Figure 4.20: Comparison of labels assigned for documents, for different values for $max_{similarity}$ and $min_{similarity}$

Based on the quantitative analysis by distributions and manual evaluation of the quality of topics assigned to sampled users, it was showed to be reasonable to use labels assigned by standard method having parameters w2v $max_{similarity} = 0.75$, $DL_{tolerance} = 0.15$ and $max_{similar} = 5$. Also, Flair method with parameters $max_{similarity} = 0.90$, $min_{similarity} = 0.80$, $max_{similar} = 5$ and $phrase\_encoder = Pool$.

4.6 Expert Candidate Representation

After the documents are classified, the following step is to build the expert candidate representation. This consists in building the table associating each expert to each topic, giving an association score.

Expert Candidate Representation - Internal Publications

For Internal Publications, the table associating each author to each topic is built assigning the scores following Equation 4.2:

$$score(topic, c) = \sum_{d \epsilon D_c} tf(topic, d) \times e^{\frac{-t_d}{T}} \quad (4.2)$$

where *topic* is a taxonomy term, c is an candidate, $tf(topic, d)$ is the frequency of label *topic* in document d, D_c is all documents authored by candidate c , t_d is the age of the document in years, T is the retention period, so $e^{\frac{-t_d}{T}}$ defines the knowledge decay factor, similarly to baseline model.

Expert Candidate Representation - Previous Work Assignments

Now, for Previous Work Assignments, the table associating each employee to each topic is built assigning the scores following Equation 4.3:

$$score(topic, c) = \sum_{a \epsilon A_c} H(topic, a) \times e^{\frac{-t_a}{T}} \quad (4.3)$$

where *topic* is a taxonomy term, c is an candidate, $H(topic, a)$ is the duration in hours of assignment a, multiplied by 1, if *topic* is a label in assignment a, or multiplied by 0 otherwise, A_c is the set of all assignments candidate c worked at, t_a is the age of the assignment in years, T is the retention period, so $e^{\frac{-t_a}{T}}$ defines the knowledge decay factor.

4.7 Topics I Know as Target Data

From Internal Publications and Previous Work Assignments the expert candidate representations are constructed. Those documents carried the information about previous experiences of the employees. To learn in supervised fashion how to classify an expert, given an sample input, a set of true labels is needed. The Classification of *Topics I Know* transformed the free filled text into only terms known by Taxonomy set. Though it was generated unsupervisedly, it is assumed that they represent the true expertise of a person.

For each user, it is assigned a 1, if he/she has this label assigned by classification step, or 0 otherwise. The result is a binary matrix associating users to topics they know about. Nevertheless, a 0 not necessarily represent a user doesn't know about the topic, considering the topic was just not filled previously, whereas a 1 is considered to be truly known from this point on.

4.8 Data Preprocessing

Models in supervised machine learning such as Neural Networks require numerical input vectors. The expert candidate representations, built from all the text collections, provide inputs in such format. Lets call these vectors *raw inputs*. Now, the problem of recommending expertise can be solved as a multi-class multi-label classification task.

Internal Publications and Previous Work Assignments Preprocessing

The raw inputs (resulted from the previous steps using the chosen parameter values already described) have the shapes:

- Internal Publications: 11824 users, 2868 features (topics).
- Previous Work Assignments: 15740 users, 880 features (topics).

One important step to perform, is the dimensionality reduction. PCA was not considered for this task, because it results in orthogonal features that do not contain the original meaning anymore (how strong is the association of a user to the topic). The method chosen to eliminate some features was *drop by correlation*. This method makes great sense particularly considering how the input data was generated. If two topics on taxonomy would have very similar embeddings, for example, both would always be added together as labels in a document, resulting in columns with correlation 1. Following, the steps for this process:

- Normalize to mean=0 and std = 1, column-wise. Here there are two options: Normalize over all users, or separately by category. The category indicates the seniority of the employee, and might be desirable to compare only employees of same category when normalizing.
- Compute Pearson Correlation between all features. Check pairwise correlation and drop one column if the correlation is above threshold. 0.9 was the value used for the threshold.

An important observation is that any threshold with value $|v| < 1$ results in instability, when using this method; meaning the column chosen to be dropped in the pairwise comparison impacts all next pairwise comparisons for remaining columns. Regardless, it is known that only columns with correlation below the threshold would remain. This step, with threshold = 0.9 results in:

- 1781 features for Internal Publications.
- 800 features for Previous Work Assignments.

Some features appears in both inputs. It is expected as they represent degrees of association between users and taxonomy topics. A simple join of the tables would treat a repeated feature from both inputs as independent, renaming one of them. However, a user associated to a topic both by publishing documents about it, or working on assignments related to it, has in fact a stronger association. Therefore, for the common features, their values are summed during merge of the inputs. It is possible to apply this approach in both inner or outer joins. For outer join, a user not present in one of the inputs should be assigned the minimum value of each feature column. For inner join, as only users present in both inputs are kept, they can just be merged summing the values from common columns and keeping the value for unique ones.

The result of this merge, with inner join, is a input, is an input with 11528 users, and 2217 features. Still considerably high number of features, but lately handled by a convolutional layer in a neural network. Motivated by the usage of a convolutional layer, one last step is taken: ordering of input.

Input Ordering

Topics are many times not independent features. A field of expertise can actually be composed of many topics. *"Machine Learning"* could be seen as a topic closer to *"Artificial Intelligence"* than to *"Corporate Governance"*, for example. The Taxonomy graph is actually built keeping terms related closer in same lineage. Figure 4.21 shows an small example of how the taxonomy looks like.

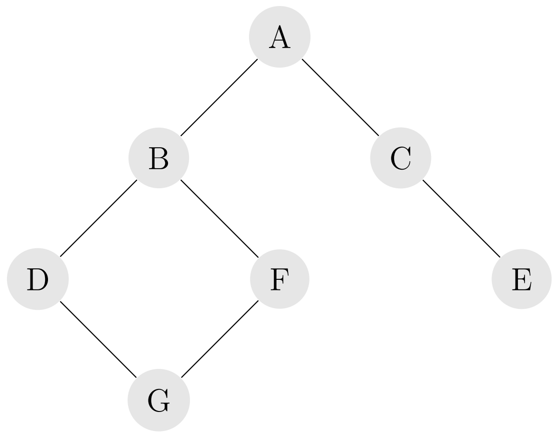


Figure 4.21: Example of how the taxonomy graph can is structured. For the ordering, consider each edge with distance 1. Distance from topic F to topic C would be 3 ($F \to B \to A \to C$)

To order the input, firstly the taxonomy graph is represented in an adjacency matrix. Then, the minimum distances between pairs of nodes (topics) are computed applying Floyd-Warshall algorithm - *a dynamic-programming formulation to solve the all-pairs shortest-paths problem on a directed graph* (Cormen et al. [2009]), generating Table 4.11. Having the distances between each topic-pairs, a fully connected graph is directly obtained from the adjacency matrix given by the the topic distances.

	A	**B**	**C**	**D**	**E**	**F**	**G**
A	0	1	1	2	2	2	3
B	1	0	2	1	3	1	2
C	1	2	0	3	1	3	4
D	2	1	3	0	4	2	1
E	2	3	1	4	0	4	5
F	2	1	3	2	4	0	1
G	3	2	4	1	5	1	0

Table 4.11: Distances between all topics according to topics computed for the graph in Figure 4.21

Now, to obtain the ordered taxonomy topics, they need to be placed so that the total distance is minimal:

$$Total_d = \sum_{i=1}^{n-1} D(topic_i, topic_{i+1}) \tag{4.4}$$

This problem can be reduced to the TSP (Traveling Salesman Problem), being the main difference the fact that there is no need to return to start topic.

Therefore, to use traditional algorithms to solve the TSP, it suffices to add a *dummy* topic in the adjacency matrix with distance 0 to all other topics and to itself.

Given the taxonomy has 4600 topics, and TSP is an np-hard problem, approximation algorithms have to be used. 2-OPT (Nearest Neighbors as initial trip, and 100 2-OPT iterations) and Greedy were tested for this task. The best distances found were 9216 and 9142 respectively. Greedy is over one hundred times faster than 2-OPT and also produces better results. Finally, the input features are ordered according to the result obtained by TSP solver.

”Topics I Know” Preprocessing

Last step on data preparation, is the preprocessing of the Target Data. Although the original terms in profiles were filled by users themselves, now they are already replaced by the labels obtained at the classification step. The criteria to assign the original topics are subjective - each person might have different standards to feel knowledgeable in some topic - and, also, the classification was made by unsupervised method, adding more uncertainty regarding the reliability of those topics. Yet, it is assumed that a topic assigned with a 1 in the raw target data represents something a user knows about, whereas a 0 possibly means the user does not, or simply this information was missing in their profile.

Considering the raw Target Data have 2656 different topics, 384 of them having just 1 input sample, and 1927 with less than 20, it is definitely needed to reduce the number of classes (topics) to use for training. The drop by correlation method is applied again in this case, with correlation threshold set to 0.9. However, instead of dropping a topic, all 1’s from the dropped topic are actually added to the persisting correlated topic. The maximum value is clipped at 1 to keep a binary table - in the cases the topic was present in both columns.

After this merge of correlated topics, a maximum number of topics to be covered is defined through parameter setting. The selection of the parameter value should consider how many samples would be available for each topic label. For the following model, it was selected 200 most popular topics, being 51 the minimum number of samples for the least popular in this set, and 786 the maximum.

4.9 Multi Label Classifier

With the input data and the target data, it is possible to build a model to perform a multi label classification. The output of such model is a table with estimated probabilities a expert candidate knows about a topic, similarly to Table 2.16, except that differently than bm25 scores, the estimated probabilities have values in [0,1] range. The final input is a combination of Internal Publications and Previous Work Assignments data, and only users that is part of both sets are considered. Looking into the target data, only users that have at least 1 topic listed in their profiles are considered. Satisfying both constraints in input and target, the set for training a model is reduced to 5815 samples.

Bootstrapping and Bagging

Ensemble methods are suitable for unstable models such as neural networks and nearly always outperform a single model (Maclin and Opitz [2011]). Neural networks are naturally unstable due to random weights initialization. In addition, bootstrapping can be used to split training and validation sets, adding another source of instability. In this work, the target data was obtained by unsupervised labeling method, applied over incomplete data (profiles), resulting in noisy labels for training - it is not known that an absence of a label for a sample represents missing data or the sample does not belong to the class.

As mentioned before, there are 11528 input samples, but after joining with target data for users that have at least one topic assigned, the number of samples that can be used for training goes down to 5815. The remaining samples are used also for predictions and can be considered a test set, that are evaluated by the number of accepted topics from the recommender system later on.

Now, for each model in the ensemble, the bootstrapping (sample with repetition) is performed over the trainable samples. The probability of picking a sample s in a set of n samples is $P(s) = \frac{1}{n}$. The probability of not picking a sample during bootstrapping process is given by:

$$1 - P(s) = (1 - \frac{1}{n})^n \approx e^{-1} = 0.368 \tag{4.5}$$

Therefore, for each model, around 63% of the samples are used for training. Some of the samples are repeated, biasing the model towards them. The remaining 37% of the samples are using as validation set. All the samples for which there was no target at all, are used as a true test set, that obviously can not be evaluated automatically, but from the users feedback. For each model, the parameters that provide the highest $F_{0.5}$ score on validation set are saved.

Neural Network - Convolutional Layer

Bootstrapping and bagging are very popular on Decision Trees, but a particular characteristic of the data set already mentioned motivates the use of Neural Networks instead: the input features are not fully independent. An example of the Neural Network used in practice for this task is described as following:

- Conv 1D, 50 filters, Kernel size: 5, Stride: 1, Padding: valid, Activation: ReLU, Batch Normalization
- Max Pooling 1D, Pool size: 3, Stride: 3, Dropout: 0.5
- Dense Layer size: 500, Activation: ReLU, Batch Normalization, Dropout: 0,5
- Dense Layer size: 50, Activation: ReLU, Batch Normalization, Dropout: 0,5
- Output Layer: Dense Layer size: 150, Activation: Sigmoid

Training performed in batches of size 20, in 50 epochs. More model hyperparameters will be compared further on.

Loss Function - Weighted $F_{0.5}$ score

In classification tasks two important metrics very often are taken into consideration: Precision and Recall. Accuracy many times is not a meaningful metric, specially having unbalanced classes, like in this case: for each topic, much more users do not know about it than users know it, therefore, classifying every user knowing nothing would lead to high accuracy despite being an useless classifier.

Precision reflects how well the model predicts an expertise of a user:

$$Precision = \frac{TP}{TP + FP} \tag{4.6}$$

being TP True Positives and FP False Positives.

Recall measures how well the model retrieves expertise, regardless of making some mistakes:

$$Recall = \frac{TP}{TP + FN} \tag{4.7}$$

being TP True Positives and FN False Negatives.

It is easy to note that for Recall, predicting all 1's, make 100% recall. For precision, predicting just one 1, but correctly, makes 100% precision. These two metrics have opposite forces, therefore a metric that takes both measurements into consideration should be used. Traditionally, F_1 score can be applied in such scenario:

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.8}$$

This metric forces both Precision and Recall to be as high as possible. A tweak for that is weighting Precision or Recall to be more important. In case of a recommender system, Precision is more important in the sense it is needed to build trust in users. The negative impact of wrong recommendations are stronger than the impact of no recommendations at all. To privilege Precision, the metric $F_{0.5}$ can be used:

$$F_{0.5} = 1.25 \times \frac{precision \times recall}{0.25 \times precision + recall} \tag{4.9}$$

Neural Networks are basically an optimization process and need a Objective Function, also called Loss Function in this context. The weights are updated according to the gradient of the Loss Function, minimizing it. Using $F_{0.5}$ as a Loss Function in a minimization problem is directly done by optimizing $-F_{0.5}$ instead.

Calculation of Precision and Recall during training

Precision and Recall are usually calculated taking the sample classification as integer: 1 for Positive and 0 for Negative. However, the output of the last layer of the neural network, with sigmoid activation, is a vector of real number in the interval [0,1] containing the estimated probabilities for the classes for a given sample. One can set a threshold and turn all values above it to 1, and all values below it to 0, but this method does not help the network to learn properly. For example, if a threshold of 0.5 is used during training, a prediction of 0.51 would make no error for a positive class, resulting in a gradient that does not push it closer to 1 during the back propagation. Same applying to negative samples. Table 4.12 shows an example of how would be the values for TP, FP and FN given Y_{pred} from target data and Y_{pred} as the rounded predictions, assuming 6 samples (from S1 to S6) for a single class.

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	SUM
Y_{true}	1	1	0	1	1	0	
Y_{pred}	0	0	1	1	1	0	
TP	0	0	0	1	1	0	2
FP	0	0	1	0	0	0	1
FN	1	1	0	0	0	0	2

Table 4.12: TP, FP and FN calculations from vectors of true values and predictions. $FP = clip(Y_{pred} - Y_{true}, 0, 1)$. $TP = (Y_{pred} - FP)$, $FN = clip(Y_{true} - Y_{pred}, 0, 1)$. Taking the sum of TP, FP and FN and following the Equations 4.6 and 4.7: Precision = 2/3 and Recall = 2/4.

With simple element wise subtractions and clipping the results to the interval [0,1], TP, FP and FN can be calculated and consequently, Precision and Recall. The exact same operations can be performed keeping a real value vector for predictions, as in Table 4.13

	S1	**S2**	**S3**	**S4**	**S5**	**S6**	SUM
Y_{true}	1	1	0	1	1	0	
Y_{pred}	0.2	0.1	0.6	0.7	0.9	0.3	
TP	0.2	0.1	0	0.7	0.9	0	1.9
FP	0	0	0.6	0	0	0.3	0.9
FN	0.8	0.9	0	0.3	0.1	0	2.1

Table 4.13: TP, FP and FN calculations from vectors of true values and predictions. $FP = clip(Y_{pred} - Y_{true}, 0, 1)$. $TP = (Y_{pred} - FP)$, $FN = clip(Y_{true} - Y_{pred}, 0, 1)$. Taking the sum of TP, FP and FN and following the Equations 4.6 and 4.7: Precision = 1.9/2.8 and Recall = 1.9/4

Table 4.13 shows an example for one single class, in a vector Sx1 (for S samples). It is trivial to see that the same operations can be performed in a matrix SxC (S samples and C classes), which is the case for this work: to build multi class classifier.

Weighted $\mathbf{F_{0.5}}$ score

Having the Precision and Recall calculated for each class, $F_{0.5}score$ can be obtained directly . Thus, for each training batch of size S, $F_{0.5}score$ is computed for each class, following Equation 4.10. Within the batches, the $F_{0.5}score$ is weighted according to the number of samples of each class. Table 4.14 illustrates an example of a batch. This is done to compensate for classes with few samples in the training set. The weight should produce a stronger signal in the back propagation for under represented classes.The weighted negative $F_{0.5}score$ is then fed to the optimizer.

$$w_i = \frac{M}{\sum_{s \epsilon C_i} s + k}$$

$$M = \max_j \left(\sum_{s \epsilon C_j} s\right) \quad (4.10)$$

$$j \epsilon [0..n]$$

being (for each batch) w_i the weight for class i, M the maximum sum of positive samples of the same class, C_i the number of positive samples of class i, k the a positive constant to smooth the weights. In this experiments, the value chosen is $k = 1$.

	$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_3}$	$\mathbf{C_4}$	$\mathbf{C_5}$
S1	0	1	0	1	0
S2	0	1	0	1	0
S3	0	1	0	1	0
S4	0	0	1	1	1
S5	0	0	1	1	0
SUM	0	3	2	5	1

Table 4.14: Example of a batch of size S=5, for classes C_i

For classes C_2 and C_4 in the table, for example, the weights would be given by:

$$\begin{aligned} w_2 &= \frac{5}{3+1} = 1.25 \\ w_4 &= \frac{5}{5+1} = 0.83 \end{aligned} \quad (4.11)$$

The weights for the classes are proportional to the inverse of the number of positive samples in the classes, making the effects in the back propagation being evenly distributed among all classes.

Now, all details about the classifier are already described. Following in Chapter 5 different model hyperparameters are tested and evaluated.

5. Experiments

In this Chapter, different setups of the network are tried and their results are compared. The classified documents used are the ones obtained through the steps described in Chapter 4, using the recommended parameter values. The details to obtain the expert candidate representation from classified document combinations and the model hyperparameters for each experiment are described in next session.

5.1 Experiment Setup

The evaluation consists in the Averaged Precision (AP) and Averaged Precision at 5 (AP_5) . The metrics are computed considering a set of topic queries Q_{150} of 150 topics from which the recommendations are taken, and the four sets of users obtained according to Section 1.2. Q_{150} is taken from the resulting topics in target data after *”Topics I Know” Preprocessing.*

For each model, the steps for evaluation are:

- Build a table like Table 1.1 having the Q_{150} topics, for each set of users.
- Take top 5 scored topics per user.
- Evaluate the relevance of the topics selected: If topic already present in user profile or if recommendation was accepted
- compute AP and AP_5 for each set of users

The next subsections present the summary of inputs and model details for each experiment, specially what might differ from the implementation details presented in Chapter 4. All neural network models were implemented in Keras[1].

5.1.1 Baseline

Two versions of the baseline model are evaluated: Baseline 1 without forgetting factor ($e^{\frac{-t_d}{T}}$), and Baseline 2 with forgetting factor. The association table is constructed with scores given by following equations for each model respectively:

[1]https://keras.io/getting-started/sequential-model-guide/

$$score(q,a) = \sum_{d \epsilon D_a} \frac{S(q,d)}{max(S(q))}$$

$$score(q,a) = \sum_{d \epsilon D_a} \frac{S(q,d)}{max(S(q))} \times e^{\frac{-t_d}{T}}$$

BM25+ model is trained on Internal Publications only.

5.1.2 Base Neural Network

Inputs

- Internal Publications, Previous Work Assignments and ”Topics I Know” classified only by standard method
- Expert candidate representation is built without forget factor.

Model

```
# CONVOLUTIONAL LAYERS
model.add(Conv1D(filters=50, kernel_size=5, strides=1, padding="valid", input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling1D(pool_size=3))
model.add(Dropout(0.50))

# FULLY CONECTED LAYERS
model.add(Flatten())
model.add(Dense(500))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(50))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# OUTPUT LAYER
model.add(Dense(classes))
model.add(Activation("sigmoid"))
```

5.1.3 Model A

Inputs

- Internal Publications, Previous Work Assignments and ”Topics I Know” classified only by standard method.
- Expert candidate representation including forget factor.

Model

This model has the the same hyper parameters as the neural network defined in 5.1.2.

5.1.4 Model B

Inputs

- Internal Publications, Previous Work Assignments and ”Topics I Know” classified only by standard method
- Expert candidate representation including forget factor.

Model

```
# CONVOLUTIONAL LAYERS
model.add(Conv1D(filters=50, kernel_size=10, strides=1, padding="valid", input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling1D(pool_size=3, strides=2))

model.add(Conv1D(filters=50, kernel_size=5, strides=1, padding="valid", input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling1D(pool_size=3, strides=2))

# FULLY CONECTED LAYERS
model.add(Flatten())
model.add(Dense(1000))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(100))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# OUTPUT LAYER
model.add(Dense(classes))
model.add(Activation("sigmoid"))
```

5.1.5 Model C

Inputs

- Internal Publications classified by standard method.
- Previous Work Assignments and ”Topics I Know” classified by both standard and Flair methods.
- Expert candidate representation including forget factor.

Model

The model has the structure presented in 5.1.4

5.1.6 Model D

Inputs

- Internal Publications, Previous Work Assignments and ”Topics I Know” classified only by standard method
- Expert candidate representation is built without forget factor.
- Shuffled features of final input to test the effects of unordered inputs.

Model

The same model described in 5.1.2 is used here.

5.1.7 Model E

Inputs

- Internal Publications, Previous Work Assignments and ”Topics I Know” classified only by standard method
- Expert candidate representation is built without forget factor.

Model

The convolutional layers are removed from this model, keeping only the fully connected layers as follows:

```
# FULLY CONECTED LAYERS
model.add(Flatten())
model.add(Dense(500))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Dense(50))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# OUTPUT LAYER
model.add(Dense(classes))
model.add(Activation("sigmoid"))
```

5.1.8 Model F

Inputs

- Internal Publications classified by standard method.
- Previous Work Assignments and ”Topics I Know” classified by both standard and Flair methods.
- Expert candidate representation including forget factor.

Model

The model used here is has the same structure described in 5.1.2

5.2 Experiment Results

5.2.1 Expert Profiling

Now the AP and AP_5 are calculated for all models described above, for each of the four different sets of users. The results for AP are shown in Table 5.1. Table 5.2 shows the AP_5 results.

Average Precision				
Model	**S1**	**S2**	**S3**	**S4**
Baseline 1	0.1257	0.2387	0.0707	0.4016
Baseline 2	0.1246	0.2418	0.0685	0.4176
Base Neural Net	0.4618	0.5684	**0.3612**	**0.7535**
Model A	0.4648	**0.587**	0.2283	0.726
Model B	0.4019	0.5382	0.1934	0.7167
Model C	0.1993	0.2697	0.0054	0.5388
Model D	**0.4869**	0.5841	0.3505	0.7487
Model E	0.3774	0.5184	0.2383	0.7377
Model F	0.1676	0.2336	0.0079	0.4163

Table 5.1: Average Precision over all 150 topics, computed for each set of users

For expert profiling, Base Neural Net model produced th best results for most of the sets, including S4 and set S3, which is the most challenging set given its users are not seeing during training and had no topics filled in their profiles. It is surprising that models D and E have high performances despite having unordered inputs and no convolution operations respectively. Generally, the addition of flair matches in Previous Work Assignments and Topics I Know classification makes

Average Precision at 5				
Model	**S1**	**S2**	**S3**	**S4**
Baseline 1	0.0679	0.1273	0.034	0.2141
Baseline 2	0.0688	0.1298	0.034	0.2254
Base Neural Net	**0.3007**	**0.5149**	**0.1791**	0.7465
Model A	0.1618	0.3235	0.0515	0.6563
Model B	0.1623	0.3209	0.054	0.6535
Model C	0.1572	0.263	0.0044	0.4886
Model D	0.2543	0.4658	0.1343	0.738
Model E	0.2667	0.4832	0.1439	**0.7493**
Model F	0.1424	0.2378	0.0047	0.4371

Table 5.2: Average Precision at 5 over users, for each set of users

the performance worse. Topic-wise, the forget factor improves the performance on training set, but reduces it on unseen data, as seen in models Baseline 2 compared to Baseline 1, and Model A compared to Base Neural Net, for sets S2 and S3 in table 5.1. For user-centric measurements, the forget factor reduces drastically the performance on neural network models, but has no big impact for baseline. Adding second convolutional layer did not improve the results, which can be seem comparing Model B with Model A in both tables.

5.2.2 Expert Finding

Now the AP_{10} is calculated for each of the four different sets of users, and AP_{100} except for $S4$, for all models described above. The results are shown in Table 5.3 and Table 5.4.

Average Precision at 10				
Model	**S1**	**S2**	**S3**	**S4**
Baseline 1	0.0822	0.0905	0.0373	0.0765
Baseline 2	0.0844	0.0902	0.044	0.0744
Base Neural Net	0.5446	0.5835	0.3227	0.2695
Model A	0.5812	0.6266	0.2404	0.2822
Model B	**0.6171**	0.6507	0.2418	**0.2875**
Model C	0.4154	0.4187	0.017	0.1753
Model D	0.5776	0.6079	0.3027	0.2699
Model E	0.6116	**0.6555**	**0.3253**	0.2836
Model F	0.3384	0.344	0.02	0.1695

Table 5.3: Average Precision at 5 over users, for each set of users

In TREC Enterprise 2007 (Bailey et al. [2007]) the task was also to retrieve

Average Precision at 100			
Model	**S1**	**S2**	**S3**
Baseline 1	0.0697	0.0719	0.0333
Baseline 2	0.0696	0.0717	0.0347
Base Neural Net	0.3744	0.3644	**0.1852**
Model A	0.3553	0.3606	0.1071
Model B	0.3713	0.3712	0.1148
Model C	0.2676	0.278	0.0042
Model D	0.3757	0.361	0.1655
Model E	**0.4139**	**0.3968**	0.1785
Model F	0.2206	0.2336	0.0046

Table 5.4: Average Precision at 5 over users, for each set of users

experts given topic queries. The best results achieved for expert finding task had Precision at 5 around 0.25 and Precision at 20 of 0.09, having 50 topic queries used. The same task was studied in TREC Enterprise 2008 - on different data and now 77 different topics and 3678 expert candidates (Balog et al. [2008]), achieving the best result of 0.371 Precision at 10. Besides the data being different, these values serve as some reference to understand the performance observed in the proposed methodology on this work. Moreover, 150 topics were covered in these experiments, almost doubling the topic coverage, and the test set S3 had 4163 expert candidates.

The Base Neural Net has a consistent good performance in all evaluations. It is interesting to observe, that for expert finding the additional convolutional layer improves the results (Model B vs Model A, and Model C vs Model F). Again, the forget factor reduces the performance in S3, indicating a possible overfit. The use of Flair matching when generating the inputs also produces worse results. Again, Model E has unexpected high performance, specially in training data, but still worse than Base Neural Net for unseen samples. The Baseline 1 and 2 have similar results and both are comparable to average performances observed in TREC Enterprise 2007 and 2008.

Overall, despite all the unsupervised methods applied to classify the documents and preprocess the data for training a classifier, and also the target data being incomplete, all the variants of the neural network model perform considerably better than the baseline models. The results achieved by the proposed neural networks in the sets unseen during training (test sets $S3$ and $S4$) are comparable to the state of the art results observed in TREC Enterprise 2008.

6. Conclusion

The goal of this work was to propose a solution for expertise retrieval task based on a neural network classifier. Despite the mapping between people and topics being made through *expert profiling* perspective, the results showed that the method can also perform well in *expert finding.*

One of the early steps to solve the task is the document classification, consisting in representing the documents in the collection exclusively by topics from a taxonomy set. The impact of this step was clearly observed in the final results, especially comparing the models that had inputs generated only through standard method and the models that had combined input and target data generated from standard and Flair methods. It was observed that, despite the flexibility of Flair embeddings, classifying the documents using direct semantic matching reduces the performance of the system as a whole.

Ordering the input features was expected to promote better results based on the intuition behind the max pooling operation. However, the results showed it has minimal impact. The hypothesis is that groups of similar topics have different lengths and therefore a fixed size kernel can not produce a satisfactory feature extraction for the next layers. Yet, some future work may confirm it.

Since ordering the inputs did not improve the results significantly, it was expected the convolutional layer would not impact much the results. The fully connected layers treat the features as independent and even with a high dimensional input vectors - 2381 dimensions - and 3909 samples, the neural networks classifier performs well. However, for the models having inputs generated from combination of standard and flair methods during document classification, the additional convolutional layer improves the results reasonably.

The impact of the forget factor applied while building the expert candidate representations was unclear. In *expert profiling*, it generally reduces the precision. However, in *expert finding* the results are improved. For baseline models the difference is irrelevant.

Finally, it was shown that a supervised method based on neural networks trained on incomplete target data and high dimension inputs is robust, resulting in precision considerably higher than the baseline. Moreover, even covering more topics, the results are comparable to the state of the art obtained in similar benchmark tasks, like TREC Enterprise.

Bibliography

Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. 08 2018.

Samuel Arbesman. Truth decay: The half-life of facts. *New Scientist*, 215:36–39, 09 2012. doi: 10.1016/S0262-4079(12)62454-3.

Peter Bailey, Arjen P. de Vries, Nick Craswell, and Ian Soboroff. Overview of the TREC 2007 enterprise track. In *Proceedings of The Sixteenth Text REtrieval Conference, TREC 2007, Gaithersburg, Maryland, USA, November 5-9, 2007*, 2007. URL `http://trec.nist.gov/pubs/trec16/papers/ENT.OVERVIEW16.pdf`.

K. Balog, Y. Fang, M. de Rijke, and P. Serdyukov. Expertise retrieval. *Foundations and Trends in Information Retrieval*, 6:127–256, 2012. doi: 10.1561/1500000024.

Krisztian Balog, Leif Azzopardi, and Maarten Rijke. Formal models for expert finding in enterprise corpora. pages 43–50, 01 2006. doi: 10.1145/1148170.1148181.

Krisztian Balog, Ian Soborof, Paul Thomas, Peter Bailey, Nick Craswell, and Arjen P. de Vries. Overview of the trec 2008 enterprise track. In *Proceedings of The Seventeenth Text REtrieval Conference, TREC 2008*, 2008. URL `https://trec.nist.gov/pubs/trec17/papers/ENTERPRISE.OVERVIEW.pdf`.

L. Charlin and R.S. Zemel. The toronto paper matching system: An automated paper-reviewer assignment system. *Proceedings of the 30 th International Conference on Machine Learning, Atlanta, Georgia, USA*, 28, 2013.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017. doi: 10.18653/v1/d17-1070. URL `http://dx.doi.org/10.18653/v1/d17-1070`.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009. ISBN 0262033844.

Thomas H. Davenport and Laurence. Prusak. *Working Knowledge : How Organizations Manage What They Know*. Harvard Business School Press, 1998. ISBN 9780875846552.

Hui Fang and ChengXiang Zhai. Probabilistic models for expert finding. In Giambattista Amati, Claudio Carpineto, and Giovanni Romano, editors, *Advances in Information Retrieval*, pages 418–430, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-71496-5.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

S.S. Haykin and B. Van Veen. *Signals and systems*. Wiley, 1999. ISBN 9780471138204. URL `https://books.google.de/books?id=QA9LAQAAIAAJ`.

Yuanhua Lv and ChengXiang Zhai. Lower-bounding term frequency normalization. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 7–16, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0717-8. doi: 10.1145/2063576.2063584. URL `http://doi.acm.org/10.1145/2063576.2063584`.

Richard Maclin and David W. Opitz. Popular ensemble methods: An empirical study. *CoRR*, abs/1106.0257, 2011. URL `http://arxiv.org/abs/1106.0257`.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008. ISBN 0521865719.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013b. URL `http://arxiv.org/abs/1310.4546`.

Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072.

Juan Enrique Ramos. Using tf-idf to determine word relevance in document queries. 2003.

David C. Rubin, Sean Hinton, and Amy Wenzel. The precise time course of retention. *Journal of Experimental Psychology*, 25, 1999.

Iskandar Setiadi. Damerau-levenshtein algorithm and bayes theorem for spell checker optimization. 12 2013. doi: 10.13140/2.1.2706.4008.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015. URL `http://arxiv.org/abs/1506.04214`.

List of Figures

List of Tables

List of Abbreviations

AP	Average Precision
CNN	Convolutional Neural Networks
DAG	Direct Acyclic Graph
LSTM	Long Short-Term Memory
MLP	Multi-Layered Perceptron
NLP	Natural Language Processing
PCA	Principal Component Analysis
RNN	Recurrent Neural Networks
TFIDF	Term Frequency-Inverse of Document Frequency
TREC	Text Retrieval Conference
TSP	Travelling Salesman Problem