Enrichment of Taxonomy

Master Thesis

Mili Biswas

University of Fribourg

October 2020

Supervisor

Dr. Mourad Khayati

eXascale Infolab, Department of Informatics, University of Fribourg

Co-supervisor

Prof. Dr. Philippe Cudré-Mauroux

eXascale Infolab, Department of Informatics, University of Fribourg

Abstract

Building a taxonomy from textual data is an essential task in various applications and systems. A taxonomy created from textual data helps to structure information into categories that further enables search and reuse of those information effectively. This is the case of Vogue retailer, where taxonomies are used to improve the online search and recommending products. Taxonomies allow also to define rules and relationships among different information categories in an abstract way that facilitates further development and refinement of a knowledge-base system. Many applications such as Information Retrieval, Text Clustering and Classification or Text Mining heavily relies on taxonomies built on textual data.

Similarly to the automatic creation of taxonomy, enriching an existing taxonomy based on new text data is also gaining popularity now-a-days. For example, in fashion technology, new product concepts or categories are needed to be identified from new textual data and then these can be added in an existing product category taxonomy. This helps to create new product and better product recommendations. However, both creating a taxonomy and enriching an existing one are challenging tasks because these require domain knowledge of underlying data which is not available most of the time. Moreover, enriching an existing taxonomy based on little new data is almost impossible as most of the algorithms require large amount of data to train the associated model.

In this thesis, we study different state-of-the art algorithms used for building taxonomy automatically from textual data. We also propose a novel technique TaxoTL for enriching existing taxonomy from new data. Our empirical results on several real-world datasets show that TaxoTL is capable of enriching taxonomy correctly. Our measured scores of TaxoTL has comparable accuracy with the other state-of-the-art algorithms. We also show that TaxoTL is on average more than $20\times$ faster compared to the other state-of-the-art algorithms (2min vs. 40mins). In addition, TaxoTL is more memory efficient and consumes on average $10\times$ less memory than other algorithms.

Acknowledgement

First of all, I would like to express my deep and sincere gratitude to my research supervisor, Dr. Mourad Khayati, for his constant and invaluable support, feedback, and advice throughout my thesis work. I would also like to extend my gratitude to Prof. Dr. Philippe Cudre-Mauroux for his co-supervision of the thesis. I would also like to thank Ines Arous for her help in this research work. My sincere gratitude also goes to all who helped me during this research work by providing valuable information and data as and when needed.

Last but not least, I would like to thank my family for their patience and encouragement during the tenure I spent working on the thesis.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

The notion of taxonomy generally defines the process that classifies things or concepts. It has various applications in different branches of sciences and economics. For instance, in the biology domain, a taxonomy includes the classification of different kinds of organisms. In this process, things are grouped from the content of information that is otherwise not known. The information that it describes may contain the description, nomenclature, or identification in the classification. Another real-world example related to the importance of taxonomy is in search and recommending products online in vogue fashion. Earlier, categorising fashion items was done manually which is not perfect. Now, taxonomy has transformed the way of describing and classifying items in vogue fashion such that more comprehensive descriptions of products are being assigned resulting in more lucrative product recommendations[6]. In this thesis, we study different algorithms that leverage natural language processing techniques to find taxonomy in textual data. We also propose a novel technique that relies on transfer learning to enrich existing taxonomy with new information.

1.1 Context of work and motivation

The automatic construction of taxonomy from textual data has been studied in the literature[27],[11],[22],[3]. There are different types of algorithms that can be used for finding the taxonomies out of textual data. However, these techniques are heavily dependent on learning word embeddings based on complex mathematical models. There are additional challenges that lie with text data. Varied texts contain hidden information which does not follow proper grammatical rules, but colloquial senses. Therefore, applying rules that depend on the grammar of the written text corpus remains a great challenge[18].

In this thesis, we study different algorithms that can be effectively used to extract taxonomy hidden in any text corpus. We survey two main categories of algorithms where one uses the skip-gram[15] model to learn word embedding and the other one leverages the Nested Chinese Restaurant Process[11] to learn the representation. All these techniques however work for the entire set of text data that is given as input. These are not capable of enriching existing taxonomy without training the model from the beginning whenever new text data is available from the same domain of knowledge. Motivated by this shortcoming, we develop a new approach where we can process the new data from the same domain of knowledge without training the model from the beginning with a full set of data.

1.2 Contributions

The main contribution of this thesis consists of two major parts. First, we study different state-of-the-art algorithms to build a taxonomy. We study approaches that are based on the neural network where algorithms use a skip-gram model and for others, the probabilistic nested chinese restaurant process (nCRP) is used. We extensively study these algorithms and perform the evaluation for each of these algorithms. With our analysis, we show that one algorithm performs consistently better than others for a wide range of dataset, but sometimes, other algorithms also over-perform than the rest due to variation in the input data that possess specific pattern.

Second, we propose a novel approach to enrich existing taxonomy based on new text data within the same domain of knowledge. Here, we use the transfer learning technique. We retrain the already trained model on new additional text data. This process shows significant benefits. First, we don't have to train the full model repetitively on the full dataset as soon as there are new additional text data available. It reduces the overhead of the training time of the model on the full dataset. Additionally, we observe that using Transfer Learning, we are efficiently using the existing knowledge from the trained model to get the learned representation of words from new data. This significantly improves the performance to enrich existing taxonomy by adding new words when we have a very less amount of data which is not enough for training the neural network-based model and algorithm e.g.word2vec or nCRP based model. As soon as we successfully find the new word embeddings, we use cosine similarity measures to find the closest match between the embeddings of each word from the new text data and the existing taxonomy nodes (concept). We evaluate the whole process justified with test results quantitatively (NMI and F1-Scores) as well as qualitatively (Hypertree of taxonomy).

1.3 Outline

This thesis is structured into 5 chapters. Chapter 1 provides the introductory notes along with the contribution of the used algorithms and processes to build a new taxonomy. It also highlights the novel approach to enrich an existing taxonomy when there is a need for the availability of a new set of data from the same domain of knowledge. Chapter 2 introduces the notations that are used throughout the thesis. Next, it provides the background to the thesis by covering three main different embedding techniques. Topics related to word2vec, Nested Chinese Restaurant Process (nCRP), and Transfer Learning are discussed in detail. Here, along with the definition, theoretical derivations of important results are included. Moreover, examples are also given to clarify the concepts. Chapter 3 covers the techniques to build a new taxonomy using word2vec as a building block as well as the nCRP process. Here, each related algorithm is discussed and explained in detail. Running examples are also given related to each process of building a taxonomy. Finally, the evaluation based on Normalised Mutual Information (NMI) and F1-Scores (Micro & Macro) is used to evaluate the performances of these algorithms. In Chapter 4, the novel Transfer Learning technique to enrich an existing taxonomy is introduced together with algorithm description and running example. This chapter also includes an evaluation of performance comparisons among our novel technique based on two models of taxogen and one of its variation (nole), and a full run of the taxogen algorithm. Here, critical analysis based on Normalised Mutual Information (NMI) and F1-Scores (Micro & Macro) is given for our novel technique and other available algorithms (e.g. taxogen). Chapter 5 concludes the thesis and highlights possible future work that could be conducted on the topic by employing other state-of-the-art algorithms.

2 Background

This chapter serves as an introduction to the Word2Vector, Nested Chinese Restaurant Process, and Transfer Learning. It covers the relevant background of the techniques and algorithms we use on our propositions, implementations, and improvements.

2.1 Notation

The following notations are used throughout this thesis. Variables in bold upper-case letters refer to matrices, and bold lower case letters refer to vectors. Normal lower case letters to individual elements of matrix or vector. Double subscript indices denote rows and columns (in that order) of a matrix, e.g., w_{ij} is the i^{th} row and j^{th} column of matrix $\mathbf{X}$. Similarly, v_i denotes i^{th} element of the vector $\mathbf{v}$. We denote the transpose of a matrix $\mathbf{X}$ as $\mathbf{X}^T$ and the norm of a vector $\mathbf{v}$ is denoted as $||\mathbf{v}||$.

We denote the probability of a random variable a as $p(a)$. Also, we denote the conditional probability of a random variable a given the random variable θ as $p(a|\theta)$. The probabilistic model parameter is denoted by θ. We denote a probability distribution by the full name of the distribution with parameters enclosed in pairs, e.g., normal distribution with parameter μ and σ^2 as $Normal(\mu, \sigma^2)$. We use $a \sim Normal(\mu, \sigma^2)$ to say that random variable a has normal distribution as its probability distribution. The expectation of a random variable a is denoted as $E(a)$, the variance of a random variable a is denoted as $Var(a)$. We denote set with normal capital letter and its elements are denoted comma-separated under curly braces e.g. a set S is denoted as $S = \{a_1, a_2, \ldots, a_n\}$. Finally, the corpus text data is denoted by D and the topic or term set is defined as C.

2.2 word2vec model

word2vec[15] or w2v, in short, is a technique to learn word embedding from a text corpus. In this process, a skip-gram model is used which is a linear neural network by the implementation. Every word here is being fed into the neural network as a one-hot encoding vector representation which makes the input layer of the network. The hidden layer consists of a different number of nodes each act as a linear function. This function has weight vectors initialized randomly. Finally, all the outputs from hidden layers nodes

are passed into output layers nodes. Output layers consist of the same number of nodes as the number of words in the vocabulary. Each of the output layer's node has activation function softmax that essentially provides a probability for the respective word defined for an individual node.

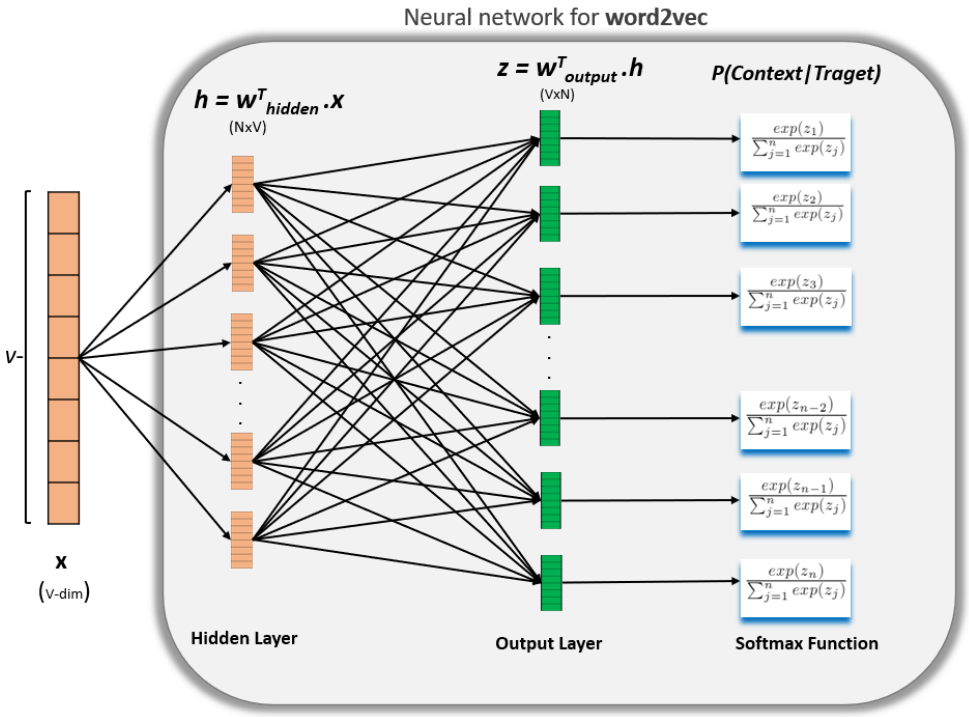


Figure 2.1: Schematic diagram of word2vec model. The softmax activation function is shown separate to each output node, but in reality they belong to each perceptron of the output layer.

Figure 2.1 depicts the word2vec neural network of an input one-hot encoding vector $\mathbf{x}$. Notice that, how the hidden layer weight matrix $\mathbf{W}_{hidden}$ acts as a projection that gives the embedding of a word for associated one-hot encoding input vector $\mathbf{x}$. The output of the hidden layer finally reaches the output layer that each perceptron uses a soft-max activation function to generate conditional probability $p(context|target)$. This model learns the word embedding by optimizing the objective function that is based on the conditional probability and can be defined as :

$$J(\theta) = \prod_{w \epsilon Text} \left[\prod_{c \epsilon C(w)} p(c|w;\theta) \right] \tag{2.1}$$

Here, $J(\theta)$ is the objective function, θ is the parameter, *Text* is the corpus data, and $p(c|w;\theta)$ denotes the conditional probability of a context word c given that the target word w with model parameter θ. Word2Vec is essentially a neural network model having linear functions that learn the weight of the hidden layer's nodes. These weights correspond to the learned embedding of each word in the vocabulary defined from the textual input. As per the work of Goldberg et. al.[8], we can write the objective function as below.

$$\log_e J(\theta) = \sum_{(w,c)\epsilon D} (\boldsymbol{v_c}.\boldsymbol{v_w}) - \sum_{(w,c)\epsilon D} \log_e \sum_{c' \epsilon C} e^{\boldsymbol{v_{c'}}.\boldsymbol{v_w}} \tag{2.2}$$

Here $\boldsymbol{v_w}$ and $\boldsymbol{v_c}$ represent embedding of w and c respectively, C refers to the set of context words for w and D is the set of word and context pairs.

The objective function given in equation 2.2 is very expensive to optimize due to the calculation of $\sum_{c' \epsilon C} e^{\boldsymbol{v_{c'}} \cdot \boldsymbol{v_w}}$ overall context words c'. To circumvent this issue, the authors of word2vecto proposed to use a negative sampling technique where instead of updating all the vectors of incorrect contexts of a word per iteration, only a sample of contexts vectors get updated. As a result, get the following objective function:

$$\log_e J(\theta) = \sum_{(w,c) \epsilon D} \log_e \sigma(-\boldsymbol{v_c} . \boldsymbol{v_w}) + \sum_{(w,c) \epsilon D'} \log_e \sigma(\boldsymbol{v_c} . \boldsymbol{v_w}) \tag{2.3}$$

Optimizing the objective function as given in Equation 2.3 has the benefit of using negative sampling which essentially has helped to reduce the processing burden which wouldn't otherwise be possible.

2.3 Nested Chinese Restaurant Process model

Nested Chinese Restaurant Process (nCRP) is a nonparametric Bayesian probabilistic model for learning hierarchical relationships available in data. Many state-of-the-art algorithms that deal with hierarchical data (e.g. Taxonomy) have used nCRP probabilistic model as a prior distribution of tree-like structure which is otherwise unknown. We first define Chinese Restaurant Process[2] before introducing nCRP.

Let us assume a restaurant with countably infinitely many tables. Customers walk in and sit down at some table. Let's denote z_i as the indicator of i^{th} customer's sitting table. Thus if we have N number of customers, then we'll have a vector of "table assignments", $\mathbf{z} = (z_1, z_2, ..., z_N)$. Next, let n_k denote the number of people sitting at the k^{th} table, and let K denote the total number of non-empty tables. Then the vector $\mathbf{n} = (n_1, ..., n_K)$ tells us how many people are at each table. Note that $\sum_{k=1}^{K} n_k = N$. With these arrangements, the tables are chosen by the customer according to the following random process.

1. The first customer always chooses the first table,
2. The $n + 1^{th}$ customer sitting at k^{th} table with the the probability:

$$p\Big(z_{n+1} = k | \mathbf{n}, \alpha\Big) = \begin{cases} \frac{\alpha}{n+\alpha} & \text{if k is a new table} \\ \frac{n_k}{n+\alpha} & \text{if k is already occupied by } n_k \text{ customers} \end{cases} \tag{2.4}$$

Here, α is a scalar hyper-parameter of the process. This whole process is named as Chinese Restaurant Process (CRP). It is typically used as a prior probability distribution for partitioning a sample. The probability of a particular set of assignments $\mathbf{z}$ (with corresponding count vector $\mathbf{n}$) for a CRP with hyper-parameter α is as follows:

$$P(z|\alpha, N) = \frac{\Gamma(\alpha) \prod_{k=1}^{K} \Gamma(n_k)}{\Gamma(N + \alpha)} \alpha^K \tag{2.5}$$

nCRP is a distribution over hierarchical partitions. It generalizes the Chinese restaurant process (CRP), which is a distribution over partitions. Let's now imagine that we have tables that are organized in a hierarchy. There is one table at the first level, and it is associated with an infinite number of tables at the second level. Each of the second-level tables is again associated with an infinite number of tables at the third level and so on until the L^{th} level. Each customer starts choosing a table at the first level and comes out at the L^{th} level that creates a path with L tables as the customer sits each table. Here, the customer chooses a table to move from level l to level $l + 1$ following the CRP defined by Equation 2.4.

2.4 Transfer Learning

Let's assume a domain D that consists of two components, a feature space $\mathcal{X}$ and a marginal probability distribution $p(X)$, in which $X = \{x_1, \ldots, x_n\} \in \mathcal{X}$. Given a specific domain $D = \{\mathcal{X}, p(X)\}$, a task can be defined that consists of two components, a label space $\mathcal{Y}$ and an objective predictive function $f(.)$, denoted by $T = \{\mathcal{Y}, f(.)\}$, which can be learned from the training data pairs $\{x_i, y_i\}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. The function $f(.)$ can be used to predict the label of a new instance x, which can be rewritten by the conditional probability distribution $p(Y|X)$. A task can then be defined as $T = \{\mathcal{Y}, p(Y|X)\}$. With these definitions of domain and task, the transfer learning is defined as: given a source domain D_S with a corresponding source task T_S and a target domain D_T with a corresponding task T_T , transfer learning is the process of improving the target predictive function $f_T(.)$ by using the related information from D_S and T_S, where $D_S \neq D_T$ or $T_S \neq T_T$. Transfer learning is used to improve a learner from one domain by transferring information from a related domain[25].

In the above definition, the condition $D_S \neq D_T$ refers to either $\mathcal{X}_S \neq \mathcal{X}_T$ or $p_S(X) \neq p_T(X)$, i.e., the source and target domains have different feature spaces or marginal probability distributions, whereas the condition $T_S \neq T_T$ means either $\mathcal{Y}_S \neq \mathcal{Y}_T$ or $p(Y_S|X_S) \neq p(Y_T|X_T)$, i.e., the source and target domains have different label spaces or conditional probability distributions. Note that when the target and source domains are the same, i.e., $D_S = D_T$, and their learning tasks are the same, i.e., $T_S = T_T$, the learning problem becomes a traditional machine-learning problem. If the transfer learning improves the performance upon using only D_T and T_T, the outcome is referred to as a positive transfer. Otherwise, transfer learning deterioration leads to a negative transfer.[13][16]

Consider an example of retraining a word2vec model based on new textual data. Here, an already trained word2vec model is given. Also, the new textual data is from the same domain of the data that had trained the existing model. Now, the retraining of the word2vec model can be done efficiently by transfer learning where the retraining process starts with the existing trained model's word embeddings. Here, essentially, the previously trained model has transferred the embeddings of words as 'knowledge' to the new retraining process. Therefore, the retraining process does not have to start with embeddings that are initialized with random values. This 'transfer of knowledge' is helping the algorithm to reach the optimal state correctly and efficiently.

3
Building Taxonomy

In this chapter, we discuss different algorithms and processes that are being used to build hierarchical taxonomy. We begin with a brief idea about how to build taxonomies using different processes. Then we define and explain different algorithms in detail.

3.1 Idea

In this section, we study algorithms that use word2vec or nCRP while building taxonomy based on textual data. First, we extract the important words a.k.a keywords, that convey the importance in the text by employing different methods e.g. TF-IDF scoring to remove unimportant words or by using a chunker (e.g. noun phrase chunker) to do the same. Once the list of important words (keyword) is available, we learn the embedding of these words based on word2vec or nCRP based process. Finally, we use the clustering algorithm recursively to build the taxonomy based on those embeddings of keywords.

Using the word2vec method is straightforward. Once we find all the learned embeddings of keywords, we run the Spherical Clustering method to build the cluster of categories for the first level after the root node of the hierarchy tree. Then, for each cluster, we either cluster further taking the words from that respective cluster or we can relearn the words of that cluster using word2vec. In another approach, where we use nCRP, first, we build the word co-occurrence network based on TF-IDF scoring from the text corpus. Then with the word co-occurrence network, we run the nethiex[11] algorithm to learn the embedding of the keywords that we need. Each of these word embedding consists of a subsection that learned the category of each level. We then use these categories' embeddings and employ clustering to build the nodes of categories in each level of the hierarchy that eventually gives the taxonomy as a tree structure in the JSON data format. There is a visualization program that makes use of the JSON file and creates hypertree based on javascript functionality. This enables viewing taxonomy more user friendly. Like any other framework & algorithm building process, we have an evaluation method that takes the output of the algorithm and calculates different evaluation metrics e.g. NMI, F1-Score (Micro & Macro).

In this process flow, first, we are doing pre-processing of the textual data. This includes the removal of unwanted words (e.g. stop words, common English words, etc.). Also, we are doing lemmatization and stemming to make the text corpus invariant to change in the grammatical form of an individual word. We also split the text corpus into tokens of words which are also known as tokenization. After the

data pre-processing is successfully done, we enter into taxonomy creation stage where the pre-processed data goes into individual algorithm of choice (e.g. Taxogen or Nethiex or Hierarchical Clustering) that eventually creates the taxonomy.

Now, we describe in detail three algorithms that we study as generating hierarchical taxonomy algorithms: *Taxogen, Hierarchical Clustering (HCLUS), and Nethiex*.

3.2 Taxogen

As explained in the original paper Taxogen[27], there are two key challenges while building high-quality taxonomies. First, it is difficult to determine the levels of different concept terms in the hierarchy. When splitting a coarser node into fine-grained ones, not all the concept terms should be included to the child level. Second, global embeddings have limited discriminative power at lower levels. Term embeddings are typically learned based on the context words within the corpus, such that terms sharing similar contexts tend to have close embeddings. However, as we move down in the hierarchy, the learned term embeddings on corpus data have limited power in capturing subtle semantics. To solve these challenges, Taxogen[27] has proposed the notions of Adaptive Clustering and local embedding.

Algorithm 1: Taxogen

Input: : A parent corpus D ; A parent topic C ; Number of clusters in each level N ; Depth of the Hierarchy H_d

Output: A dictionary representing the hierarchy

Initialize: $LVL \leftarrow 0$, $Center \leftarrow None$

function taxogen$(D, C, LVL, N, H_d, Center)$
  $L \leftarrow \{$name $:= None$; children $:= []\}$
  if $(LVL = 0)$ **then**
    $W_l \leftarrow word2vec(D)$
    $\{(S_1, \textbf{Center}_1)\} \leftarrow SphericalKmeans(W_l, 1)$
    $L[name] \leftarrow$ 'Root'
  else
    $\{(S_1, \textbf{Center}_1), (S_2, \textbf{Center}_2), ..., (S_N, \textbf{Center}_N)\} \leftarrow AdaptiveClustering(C, N, \delta)$
    $W_l \leftarrow LocalEmbeddings(D)$
    for $w \in C$ **do**
      $\mathbf{w}_{vector} \leftarrow VectorLookup(\mathbf{W}_l, w)$
      $r_i \leftarrow EuclideanDistance(\textbf{Center}_i, \mathbf{w}_{vector})$
      $R_{list} \leftarrow (r_i, w)$
    $w_{min} \leftarrow Min(R_{list})$ [$Min(.)$ returns w_i whose r_i is minimum]
    $L[name] \leftarrow w_{min}$
  if $(LVL < H_d)$ **then**
    for *each* S_i **do**
      $L_{ret} \leftarrow$ taxogen$(D_i, S_i, LVL + 1, N, H_d, Center_i)$
      $L[children].add(L_{ret})$
  return L

With this information on adaptive clustering and local embedding, now we are ready to describe the Taxogen[27] algorithm. Formally it is defined below in Algorithm 1. In this algorithm, it takes corpus D, topic C to process it to build a taxonomy. It also takes a number that it clusters the topics in each level along with the depth of the hierarchy. First, the level value LVL is initialized to value 0 that indicates the root level. Now, the function $taxogen(.)$ is used recursively to achieve the hierarchical taxonomy. At

the beginning of the function $taxogen(.)$, it initializes an empty dictionary L that eventually become the taxonomy tree. When the LVL is 0, the dictionary L stores the name as root and children array as empty. In other cases, L is completely blank.

Now as a next step, LVL is compared with the value of H_d, the depth of the hierarchy. If LVL is less than H_d value, then adaptive clustering is applied on the topic C coming as input. This process produces N number of clusters as provided as input. Each cluster S_i along with its center vector $\textbf{Center}_i$ are then used in an iterative process where for each S_i, first an empty dictionary L_{ret} is initialized. Next, the local embedding is applied on the corpus D. Now for each word w in S_i, the lookup function $VectorLookup(.)$ is done on the output of local embedding $\mathbf{W}_l$ and word w as its input. This lookup returns the embedding of word w denoted as $\mathbf{w}_{vector}$. Next, euclidean distance is calculated between the center vector $\textbf{Center}_i$ of the respective cluster using function $EuclideanDistance(.)$. The algorithm then stores the distance value and the corresponding word in a list R_{list}. Next, the algorithm finds the word from R_{list} by calculating the minimum distance value using $Min(.)$ function. This word becomes the node name that represents the respective cluster in the hierarchy. The algorithm calls the function $taxogen(.)$ recursively to build the hierarchical structure of the taxonomy.

One variation of Taxogen is no local embedding a.k.a nole. This particular variation has been proposed to compare the performance of the Taxogen algorithm. The important point here is that nole uses global embeddings throughout the process and does not use local embedding. Another variation of the Taxogen is no adaptive clustering a.k.a noac. This particular variation has also been proposed to compare the performance of Taxogen algorithm. Essentially, here the noac uses global embeddings throughout the process but does not use adaptive clustering.

Example 1 (Taxogen)**.** *Let's assume that we have a text corpus from a fashion dataset. Let's also assume that this text corpus contains customers' feedback about the fashion products that include shoes, clothes, and other accessories. There is a latent taxonomy of the fashion products in the text corpus which the algorithm extracts upon successfully analyzing the related customer reviews using varied taxonomy generation algorithms i.e. Taxogen, HCLUS, and Nethiex. In the latent taxonomy, each node represents a concept term associated with the product categories. More generic terms represent nodes near to the root. As we move down in the taxonomy, finer terms start to appear.*

Step 1 *Text corpus file has 5 sentences as below. Each sentence is considered as separate document within the text corpus.*

Natural Language Processing is a subdomain of Artificial Intelligence under computer science.
More often computer vision is now part of Artificial Intelligence in computer science.
Technology are changing where computer system is also a core part of computer science.
Database is playing an important role in system where many business applications depend.
Security is a prime concept in computer networking that work under computer system.

Figure 3.1: The corpus data for Taxogen. Each sentence in the corpus is individual document.

Step 2 *In this step, we use NLP chunker to extract important keywords from the corpus.*

natural_language_processing
artificial_intelliegnece
computer_science
computer_vision
computer_system
database
technology
security
computer_networking

Step 3 *In this step, sentence tokenization happens where each sentence becomes list of words*

```
[
[natural language processing, subdomain, artificial intelligence, computer science],
[computer vision, artificial intelligence, computer science],
[technology, computer system, computer science],
[database, system, business application],
[security, computer networking, computer system]
]
```

Figure 3.2: Tokenized corpus data for creating the initial embedding using word2vec (skip-gram) model.

Step 4 *From this step, Taxogen creates the word embeddings a.k.a initial embedding using word2vec skip-gram model.*

```
natural language processing : [0.9, -2.1, 1.6, 2.3, 1.9, 1.7, 1.1, 0.6]
subdomain : [9.9, -3.1, -2.8, 7.9, 6.3, -2.9, 3.7, 10.5]
artificial intelligence : [1.0, -2.0, 1.5, 2.5, 1.7, 1.5, 1.0, 0.5]
computer vision : [0.8, -2.4, 1.7, 2.1, 1.7, 1.3, 1.2, 0.9]
artificial intelligence : [1.0, -2.0, 1.5, 2.5, 1.7, 1.5, 1.0, 0.5]
computer science : [5.0, -5.1, 2.6, -4.7, 2.3, -2.7, 1.9, 7.5]
technology : [5.0, -5.0, 2.5, -5.1, 2.7, -2.7, 1.9, 7.5]
database : [-8.0, 3.5, 2.3, -5.6, 8.7, -2.5, 2.9, 4.0]
computer system : [-9.1, 3.0, 2.5, -5.6, 7.7, -3.5, 1.9, 4.2]
business application : [9.0, -3.1, -2.6, 9.9, 8.3, -2.3, 3.9, 9.5]
security : [-9.3, 3.4, 2.7, -4.5, 8.8, -2.6, 2.9, 3.1]
computer networking : [-9.4, 3.3, 2.7, -4.5, 8.7, -2.5, 2.9, 3.2]
computer system : [-9.0, 3.0, 2.5, -5.5, 7.7, -3.5, 1.9, 4.2]
```

Figure 3.3: Initial word embedding created for Taxogen using word2vec (skip-gram) model.

Step 5 *In this step, the initial embeddings from Step 4 are used in $SphericalKmeans(.)$ with number of cluster 1 for root node. This $SphericalKmeans(.)$ gives the word 'computer_science' as the closest word to the center of the cluster. This word 'computer_science' becomes the root node of the taxonomy.*

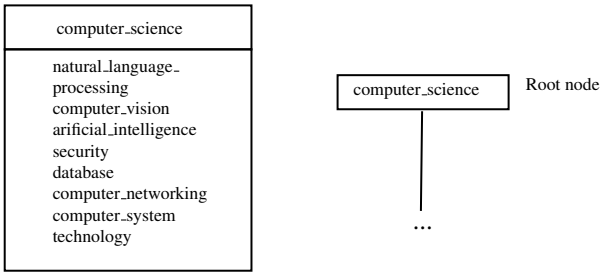


Figure 3.4: Taxonomy created by Taxogen with root node highlighted.

Step 6 *In this step, Taxogen runs recursively to build the rest of the nodes. Taxogen uses local embeddings and adaptive clustering. First, it runs the adaptive clustering where it excludes the keywords that can not be pushed further down in the hierarchy. For example, the term technology is not available whist clustering in level 1 as 'technology' is more generic name and can be placed in the root level.*

Step 7 *Now, Taxogen runs the local embedding that generates new word embedding vectors.*

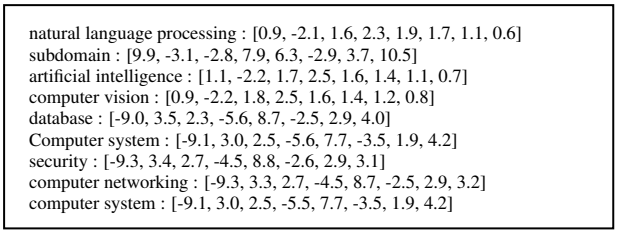


Figure 3.5: The local embedding created by Taxogen before the level 1 formation in the taxonomy.

Step 8 *Now, Taxogen runs the same SphericalKmeans() on the generated new word embeddings and cluster the data as specified. This clustering creates the 1st level nodes.*

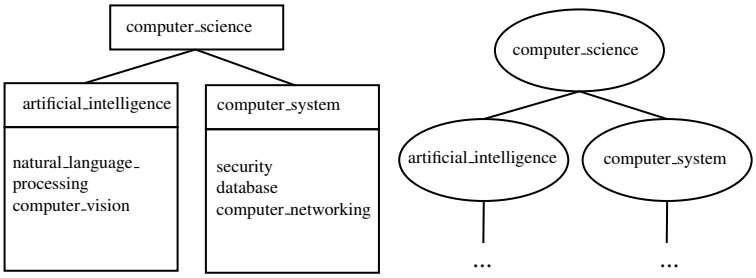


Figure 3.6: 1st level nodes created by Taxogen where root node and its child nodes are shown at the left. On the right, the taxonomy with topic nodes are shown.

Step 9 *The step 6-8 runs recursively until the depth of taxonomy exceeds the level creation value in the process. In this example, the Taxogen finally creates another level and then we get the final taxonmy.*

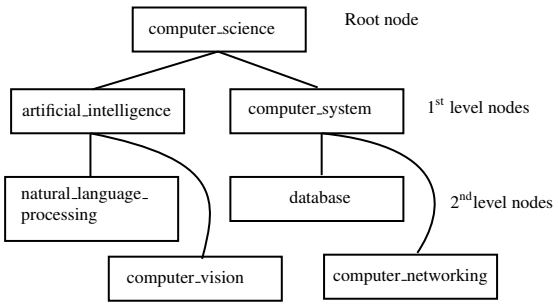


Figure 3.7: Taxonomy example created by Taxogen. Each node represents a concept topic.

3.3 Hierarchical Clustering

In this section, we explain Hierarchical Clustering a.k.a HCLUS which is formally defined in Algorithm 2.

Algorithm 2: Hierarchical Clustering (HCLUS)

Input: : A parent corpus D ; A parent topic C ; Number of clusters in each level N ; Depth of the Hierarchy H_d

Output: A dictionary representing the hierarchy

Initialize: $LVL \leftarrow 0$; $W \leftarrow word2vec(D)$; $Center \leftarrow None$

function hclus$(D, C, LVL, N, H_d, W, Center)$
- $L \leftarrow$ {name := $None$; children := []}
- $W_l \leftarrow W$
- **if** $(LVL = 0)$ **then**
  - $\{(S_1, \textbf{Center}_1)\} \leftarrow SphericalKmeans(W_l, 1)$
  - $L[name] \leftarrow$ 'Root'
- **else**
  - $\{(S_1, \textbf{Center}_1), (S_2, \textbf{Center}_2), ..., (S_N, \textbf{Center}_N)\} \leftarrow Kmeans(W_l, N)$
  - **for** $w \in C$ **do**
    - $\mathbf{w}_{vector} \leftarrow VectorLookup(\mathbf{W}_l, w)$
    - $r_i \leftarrow EuclideanDistance(\textbf{Center}_i, \mathbf{w}_{vector})$
    - $R_{list} \leftarrow (r_i, w)$
  - $w_{min} \leftarrow Min(R_{list})$ [$Min(.)$ returns w_i whose r_i is minimum]
  - $L[name] \leftarrow w_{min}$
- **if** $(LVL < H_d)$ **then**
  - **for** *each* S_i **do**
    - $L_{ret} \leftarrow$ hclus$(D_i, S_i, LVL + 1, N, H_d, W_l, Center_i)$
    - $L[children].add(L_{ret})$
- **return** L

The Hierarchical Clustering (HCLUS) algorithm starts with the initial value of LVL as 0 which indicates the root level. It also initializes the global embedding of corpus D to W. The HLCUS algorithm once completed returns hierarchy or taxonomy as a dictionary data structure. The function $hclus(.)$ runs recursively to build the hierarchy. Upon starting the algorithm, $hclus(.)$ function, it first initializes an empty dictionary L. Then it checks if the current level of hierarchy LVL is less than the depth of the hierarchy H_d. Based on the logical conclusion of this fact, the process starts. First, it clusters the data using the $Kmeans$ algorithm into N clusters (provided as input). Next, for each cluster or group, a loop iterates for every word in that group that calculates the euclidean distance between every word vector in that group with the centroid $\textbf{Center}_i$ of that group and stores the word and distance pairs in an array R_{list}. After this, it identifies the word that is closest to the centroid by finding the minimum distance from the array R_{list}. This word becomes the node of the hierarchy for that corresponding group or cluster at that level. This process continues until the algorithm exhausts when the current level LVL is greater than or equal to H_d and the algorithm stops by returning the result of recursions. This whole method is based on a depth-first search. In HCLUS Algorithm 2, we can see that we are only learning the word embedding at the beginning before starting the algorithm. Therefore, HCLUS fully depends on the initial mechanism to learn word representations.

Example 2 (HCLUS)**.** *In this example, we now show step by step how the data is being processed by HCLUS algorithm. We use the same dataset as given in Example1.*

Step 1-5 *Here Step 1 to Step 5 are the same as corresponding Step 1 to Step 5 from Example1.*

Step6 *In this step, HLCUS runs recursively to add the rest of the nodes in the taxonomy.*

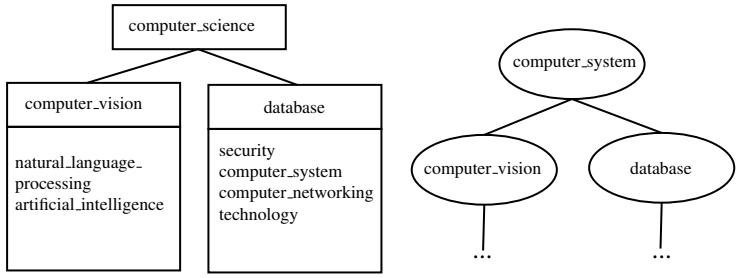


Figure 3.8: Taxonomy with 1st level nodes created by HCLUS along with different child concept terms.

Step 7 *HCLUS runs in similar way recursively as stated in previous Step6 and generates the taxonomy as below.*

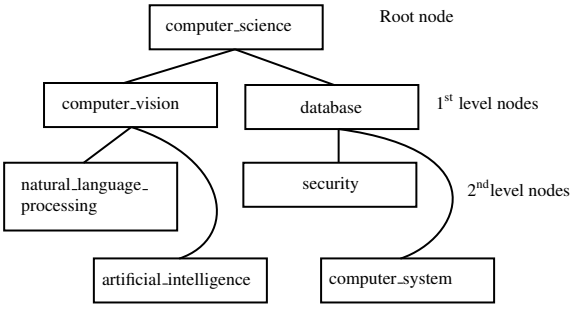


Figure 3.9: Taxonomy created by HCLUS. Each node represents a topic or concept term.

3.4 Nethiex

In this section, we explain the nCRP based algorithm nethiex[11] which is formally given in Algorithm 3. Here, the algorithm first initializes the variable LVL to 0 which indicates the process of the root. The algorithm then creates the word co-occurrence matrix using the TF-IDF scoring mechanism. Next, from the word co-occurrence matrix, the algorithm builds network data where each node of the network represents concept words or topics. This network data is then processed by nCRP based probabilistic method as proposed in the original paper of nethiex[11]. This produces learned embeddings of the concept words of topics. Each of the embeddings contains a number of compartments equal to the size of the dimension. Each compartment represents the embedding of a corresponding level of the latent hierarchy. The top compartment represents the root level and the lowest compartment represents the leaf node or the topics in the network data. Now the algorithm triggers the function $nethiex(.)$ function that runs recursively to extract the hierarchical information for the taxonomy. At first, the function $nethiex(.)$ initializes the empty dictionary data type variable L. Next, for the root level process when $LVL = 0$, the function $nethiex(.)$ extracts the top-level compartment from every embedding from matrix W by using the expression given below.

$$W\left[LVL * \lfloor \frac{d}{D_p} \rfloor : (LVL + 1) * \lfloor \frac{d}{D_p} \rfloor\right] \tag{3.1}$$

The value of each of these compartments is stored in matrix W_l. Next, the function clusters the W_l using $Kmeans$ where the number of clusters is 1 for the root. The output of $Kmeans$ is then used for the next level process recursively. In the root level process, the function also assigns the value of the dictionary L's key 'name' as 'Root'.

In the next phase, the root process triggers the function $nethiex(.)$ from within as recursive calls where LVL value is 1. In this process, the function $nethiex(.)$ first initializes the empty dictionary L. Next, it extracts the respective compartment for the embeddings using the same expression 3.1 and assigned that to matrix W_l. Next, the function triggers the $Kmeans(.)$ with W_l as input along with N which is the number of clusters. This creates the N clusters each having its corresponding topics represented by S_i and centroid vector $Center_i$. Next, the function iterates over each word in topic C and calculates the euclidean distance between the input center $Center$ and stores that in array R_{list}. The function then finds the minimum value of the distance and get the corresponding word in w_{min} which is added to the dictionary L as value for the key $name$.

Algorithm 3: nCRP based Nethiex

Input: A parent corpus D; A parent topic C ; Number of clusters in each level N ; Depth of the Hierarchy H_d
Output: A dictionary representing the hierarchy
Initialize: $LVL \leftarrow 0$; $Center \leftarrow None$

for *each document* $d \in D$ **do**
  for *for each* $w \in d$ **do**
    $tf\_idf[d][w] \leftarrow TF - IDF(w)$
$G_{net} \leftarrow Graph(tf\_idf)$
$W \leftarrow Nethiex(G_{net})$
function nethiex$(D, C, LVL, N, H_d, W, Center)$
  $L \leftarrow$ {name := $None$; children := []}
  if $(LVL = 0)$ **then**
    $W_l \leftarrow W\left[LVL * \left\lfloor \frac{d}{D_p} \right\rfloor : (LVL+1) * \left\lfloor \frac{d}{D_p} \right\rfloor\right]$
    $\{(S_1, \textbf{Center}_1)\} \leftarrow Kmeans(W_l, 1)$
    $L[name] \leftarrow$ 'Root'
  else
    $W_l \leftarrow W\left[LVL * \left\lfloor \frac{d}{D_p} \right\rfloor : (LVL+1) * \left\lfloor \frac{d}{D_p} \right\rfloor\right]$
    $\{(S_1, \textbf{Center}_1), (S_2, \textbf{Center}_2), ..., (S_N, \textbf{Center}_N)\} \leftarrow Kmeans(W_l, N)$
    for $w \in C$ **do**
      $\mathbf{w}_{vector} \leftarrow VectorLookup(\mathbf{W}_l, w)$
      $r_i \leftarrow EuclideanDistance(\textbf{Center}_i, \mathbf{w}_{vector})$
      $R_{list} \leftarrow (r_i, w)$
    $w_{min} \leftarrow Min(R_{list})$ [$Min(.)$ returns w_i whose r_i is minimum]
    $L[name] \leftarrow w_{min}$
  if $(LVL < H_d)$ **then**
    for *each* S_i **do**
      $L_{ret} \leftarrow$ nethiex$(D_i, S_i, LVL+1, N, H_d, W_l, Center_i)$
      $L[children].add(L_{ret})$
  return L

The algorithm finally calls the $nethiex(.)$ recursively where LVL is incremented by 1. This way the

Algorithm3 builds the taxonomy.

Example 3 (Nethiex)**.** *In this example, we now show step by step how the data is being processed by nethiex algorithm.*

Step 1 *Text corpus file has 5 sentences as below. Each sentence is considered as separate document within the text corpus.*

Natural Language Processing is a subdomain of Artificial Intelligence under computer science.
More often computer vision is now part of Artificial Intelligence in computer science.
Technology are changing where computer system is also a core part of computer science.
Database is playing an important role in system where many business applications depend.
Security is a prime concept in computer networking that work under computer system.

Figure 3.10: Corpus data for Nethiex. Each sentence within the corpus is considered as individual document. The TF-IDF score is calculated based on the documents.

Step 2 *In this step, based on TF-IDF scores, word co-occurrence is determined that creates the word co-occurrence network as below.*

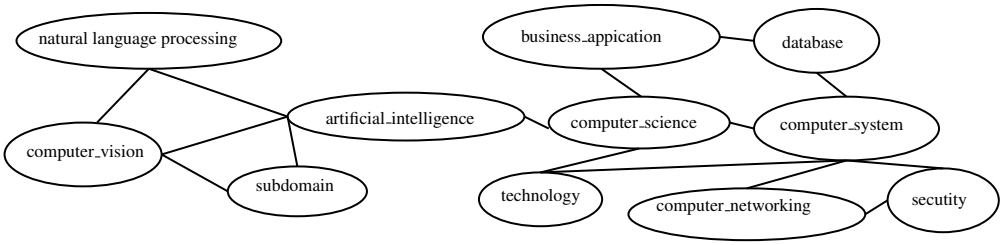


Figure 3.11: Word co-occurrence network based on TF-IDF score. The words are connected in the network where the TF-IDF score is more than a threshold.

Step 3 *In this step, the word co-occurrence network data is used by the nethiex algorithm and using nCRP probabilistic framework model, it generates word embeddings for each word from the network.*

natural_language_processing : [0.9, -2.1, 1.6, 2.3, 1.9, 1.7, 1.1, 0.6]
subdomain : [3.1, -3.2, 1.9, 2.5, 1.6, 1.4, 1.6, 0.7]
artificial_intelligence : [1.1, -2.2, 1.7, 2.5, 1.6, 1.4, 1.1, 0.7]
computer_vision : [0.9, -2.2, 1.8, 2.5, 1.6, 1.4, 1.2, 0.8]
database : [-9.0, 3.5, 2.3, -5.6, 8.7, -2.5, 2.9, 4.0]
security : [-9.3, 3.4, 2.7, -4.5, 8.8, -2.6, 2.9, 3.1]
computer_networking : [-9.3, 3.3, 2.7, -4.5, 8.7, -2.5, 2.9, 3.2]
computer_system : [-9.1, 3.0, 2.5, -5.5, 7.7, -3.5, 1.9, 4.2]
technology : [-8.9, 3.7, 4.3, -3.5, 0.7, -3.5, 1.9, 4.2]
business_application : [-0.1, 2.5, 2.9, -3.5, 5.2, -3.6, 2.9, 3.2]
computer_science : [3.6, -0.2, 5.9, 6.5, 1.6, 2.4, 1.3, 2.7]

Figure 3.12: The word embeddings created by Nethiex. Each embedding consists of embeddings of each level's concept nodes or topics.

Step 4 *In this step, nethiex algorithm takes the top level part of each word embeddings that are for level 0 or root of the taxonomy and sends that to SphericalKmean() to cluster. Here cluster number is 1*

as this is the root node. Then nethiex algorithm finds the nearest word from the cluster center that becomes the name of the root node of the taxonomy.

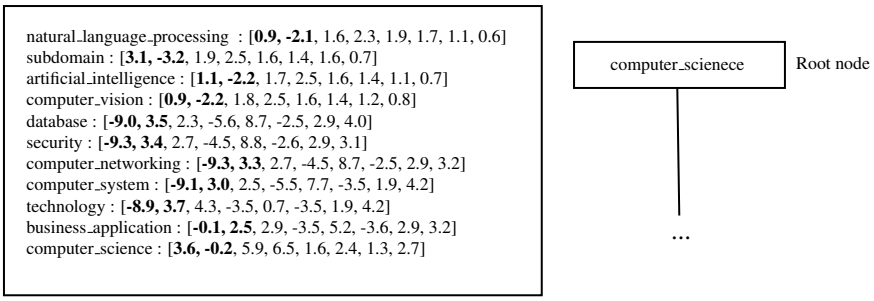


Figure 3.13: The 0^{th} level word embeddings created by Nethiex on the left. The root node is shown on the right.

Step 5 *Now, nethiex takes the next section of each word embeddings from the top after the root level section and using the SphericalKmeans() cluster the words. In this example, we are using number of cluster as 2, so it cluster the words into two groups. This creates level 1 nodes in the taxonomy.*

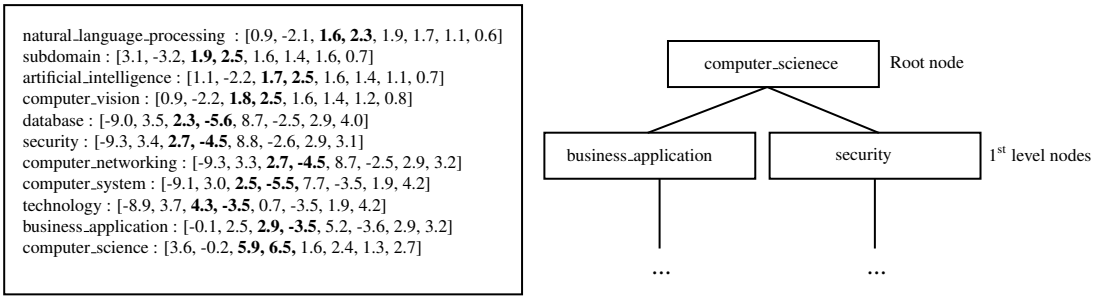


Figure 3.14: 1^{st} level embedding created by Nethiex. On the left, the embeddings are shown. Highlighted part are the dimensions of each topic word that are used to cluster the data. On the right, the 1^{st} level nodes under the root node are shown.

Step 6 *Nethiex runs Step5 recursively until the number of levels are less than the taxonomy depth defined from outside. In this example we use depth as 3. So, Nethiex is creating the taxonomy as below.*

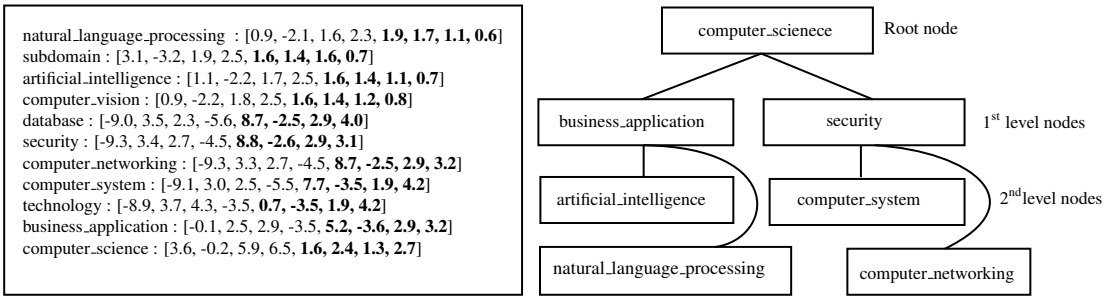


Figure 3.15: Second level node created by Nethiex. The topic embeddings shown on the left where the dimensions (highlighted in bold font) are used to create the taxonomy that is shown on the right.

3.5 Evaluation and Result

In this section, we describe our experiments and results. We begin by describing each dataset that we use in our algorithms. Next, we provide information about different algorithms and their comparison. After that detail test results of running each algorithm using all the datasets are given. We discuss the effectiveness of different algorithms based on our experimental results.

3.5.1 Experimental Setup

Datasets and Environment For all our experiments, we use three real world datasets. The reason for using these datasets is to evaluate all the algorithms' unbiased performance as well as to determine how much the models are generalized.

1. Amazon Fashion Review[10]: This dataset contains product reviews and metadata from Amazon.This contains approximately 1000000 customer reviews on clothes, shoes and accessories product categories.

2. DBLB Network: This contains approximately 48000 citations. In this dataset, titles of computer science papers from the areas of information retrieval, computer vision, robotics, security & network, and machine learning.

3. BBC News Article: This dataset contains around 2000 news articles. It has five categories of news articles as business, entertainment, politics, sport or tech.

We use Python version 3.x as our main programming language to implement all the algorithms. We built the software application in mac and tested in Ubuntu machine as well as mac.

Compared Algorithms We evaluate the below mentioned algorithms. Each of these algorithm is capable of generating the taxonomy using different methodologies.

1. HCLUS (Hierarchical Clustering) uses hierarchical clustering to create the taxonomy. Here, initial word embedding is created using word2vec model (a.k.a skip-gram). Then it uses clustering algorithm (spherical kmeans) recursively to build the taxonomy.

2. Nethiex is based on nested chinese restaurant process probabilistic framework. It first creates word co-occurrence network from the data corpus using TF-IDF scores. Then using the built word

co-occurrence network, nethiex finds representations of path from root to leaf of the latent taxonomy. The algorithm uses these representations and apply spherical clustering recursively to build the taxonomy.

3. Taxogen uses adaptive clustering and local embedding techniques to first create the word embeddings. Then recursively build the taxonomy using the embeddings from each level.

4. noac is a variant of Taxogen where adaptive clustering is not used.

5. nole is a variant of Taxogen where local embedding is not used.

Parameter Settings Algorithm Taxogen has two key hyper-parameters: number of cluster N to split a topic and the representativeness threshold δ. The variations of Taxogen also use the same parameters as applicable. For example, noac does not use adaptive clustering. Therefore, it does not use δ. For all the three datasets as mentioned earlier, we use δ as 0.25. We use different value for N for each clustering point during the entire algorithm process for each dataset. It is fully dependent on the dataset that we use because each data is unique and different. Therefore, to get the taxonomy from the data, we set the parameter N as pre-known value as supervised way. For example, we have three levels (including leaves) for dataset BBC news article. For 0^{th} level, we used $N = 1$ as trivial case. For 1^{st} level, we used $N = 2$. This non-uniformity produces our desired taxonomy more accurately. Nethiex algorithm has also two key hyper-parameters, first the representation size d which we set to 60 for all the dataset. This gives effective results while comparing the algorithm with others. The second hyper-parameter is number of cluster N to split a topic. Now it is getting the value exactly same way like before that we explained for Taxogen. We use different values for sample hyper parameter when training word2vec model for different datasets. In our case, we use '1e-4' for 'amazon_fashion' and 'bbc' and '1e-3' for 'dblp'. These yield best results for our experiments.

Metrics We consider the following metrics to evaluate the topic level taxonomy:

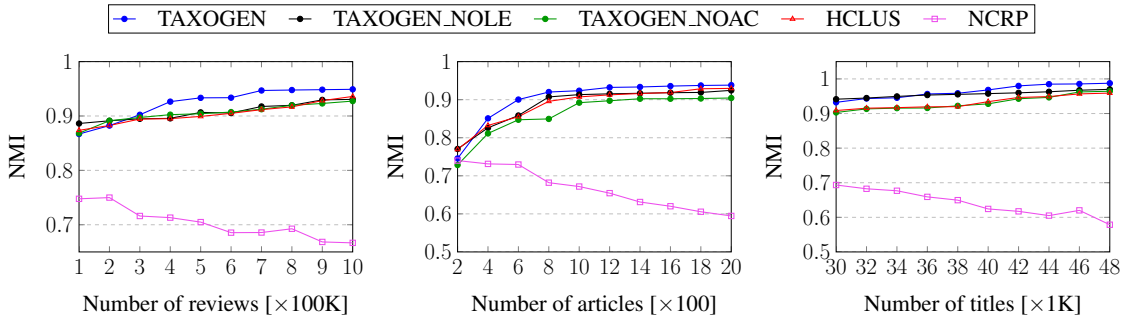
- **NMI Score** is a normalization of the Mutual Information (MI) score to scale the results between 0 (no mutual information) and 1 (perfect correlation).

- **F1 Score** is a measure of a test's accuracy where both the precision and the recall of the test are used to compute the score. The F1 score is the harmonic mean of the precision and recall, where its best value is at 1 (perfect precision and recall).

We also analyse the algorithms from a different perspective where prior knowledge of categories of unseen data is known. This supervised way to providing data into different algorithms has significant impact on the results.

3.5.2 Accuracy Results

NMI Score analysis We conduct our test for all the five algorithms on the three datasets and calculated average level-wise Normalized Mutual Information (NMI) score. Figure 3.16 depicts the results.



(a) NMI on Amazon Fashion Review data

(b) NMI on BBC news article data

(c) NMI on DBLP citation network data

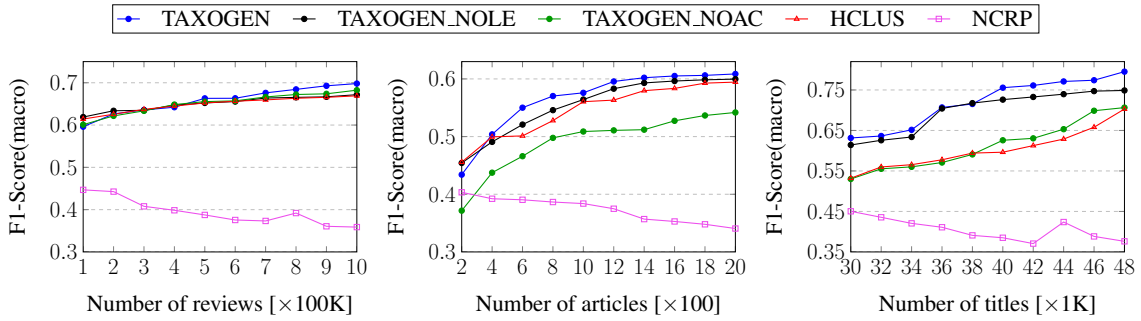
Figure 3.16: NMI scores for different algorithms with varying data volume of corpus dataset

For the amazon fashion review dataset, we observe that the NMI score for all the algorithms except ncrp increases with the number of reviews. This is because more accurate contexts of target words are available that enables better learning. Moreover, Taxogen outperforms the other algorithms because of its adaptive clustering and local embedding features. Since the volume of data in amazon review dataset is significantly high, it helps training the neural network based local embedding process very well within Taxogen. The local embedding what is used by Taxogen helps to discriminate the clusters while moving down to more granular level. Otherwise, more we go down to the hierarchy of taxonomy, the other algorithms that do not use local embedding and do clustering based on fixed global values, could not identify proper clustering boundary leading to sub-optimal results. We also observe that the ncrp based algorithm is performing sub-optimally. We believe that, ncrp based algorithm heavily depends on the probability distribution of different words within the corpus. It lacks the capability to learn semantic meaning of the target word which word2vec can do very well incorporating the context based learning. In addition to these, we notice that, sometime, the NMI score might suddenly reduce as we increase the data volume. We believe that, due to randomness of the selected data, contexts of words within the data become poor that overall reduces the NMI score.

For the BBC news article dataset, we observe similar trends like NMI scores of amazon fashion reviews data. Here, Taxogen outperforms the other algorithms. The performance is increasing as we are increasing the number of news articles. Taxogen's superior ability to learn latent taxonomy in this case is due to local embedding and adaptive clustering features. Also we believe that the BBC news article contains well written contents. It has rich set of context words for each target word which enable the algorithms to learn the semantic meaning easily. As expected, ncrp based algorithm does not perform well. This explains our justification that ncrp based probabilistic model is not able to learn the semantic meaning only based on the probability distribution of words within the corpus.

For the DBLP dataset as well, we see performance increase of algorithms along with the increase of data volume. Here, also, the Taxogen outperforms other algorithms when running on full volume of data. Here, in DBLP, we use the titles of papers from different research groups. Therefore, the context words are heavily coherent for every target word we are interested in. The ncrp based algorithm performs sub-optimally here due to poor semantic understanding of the data. The probability distribution of words within corpus is not able to encode meaning of the data properly.

F1 Score analysis In this section we show the F1-scores (Micro & Macro) averaged over all the levels based on all three datasets. The F1-macro score is plotted in Figure 3.17 and the F1-micro score is in Figure 3.18.



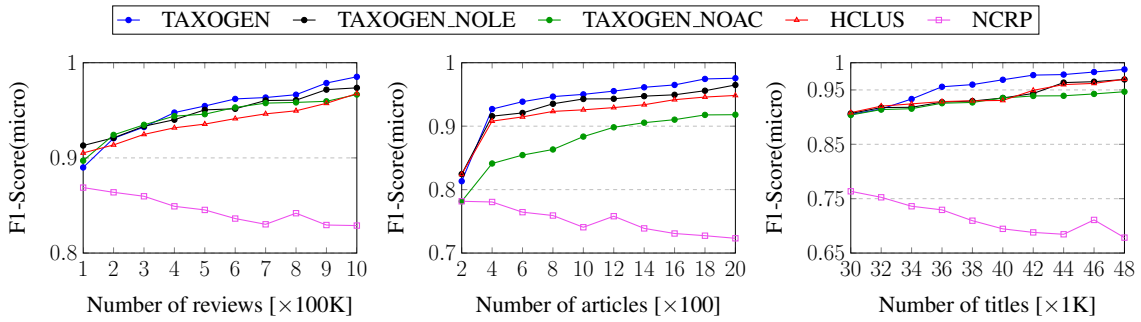
(a) F1-Score(macro) on Amazon Fashion Review data

(b) F1-Score(macro) on BBC news article data

(c) F1-Score(macro) on DBLP citation network data

Figure 3.17: F1-scores(macro) for different algorithms on varying data volume of different dataset

We observe in Figure 3.17 that the F1-macro scores of all the five algorithms calculated on the three datasets are consistent with our averaged NMI scores. Here also, as we increase the volume of data, the scores are increasing slowly. For all the datasets Taxogen outperforms the other algorithms. The reason that Taxogen is performing significantly well for all the datasets is due to better semantic learning by local embedding and adaptive clustering. The ncrp based algorithm performs sub-optimally. Our justification for this is same as before, where we argue that, ncrp based probabilistic model lacks to capture the semantic meaning based on the probability distribution of words within the corpus.



(a) F1-Score(micro) on Amazon Fashion Review data

(b) F1-Score(micro) on BBC news article data

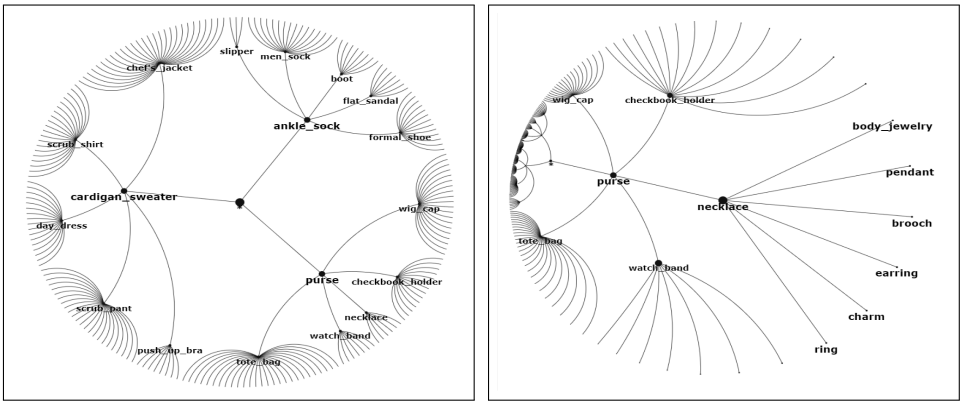
(c) F1-Score(micro) on DBLP citation network data

Figure 3.18: F1-scores(micro) for different algorithms on varying data volume of different dataset

We observe similar trend in Figure 3.18 for F1 micro scores of all the five algorithms as we have seen earlier in Figure 3.17 and in Figure 3.16. Here also, as we increase the volume of data, the scores are increasing slowly. Taxogen outperforms among other algorithms for all the datasets. The ncrp based algorithm performs sub-optimally here due to the same reason that we have argued for NMI score.

3.5.3 Qualitative Results

In this Section, we show the taxonomies created by Taxogen algorithm for the three datasets. Each node is split uniformly here. The left part of Figure 3.19 shows the full taxonomy created by Taxogen for amazon fashion review dataset where the number of clusters for each 1^{st} level as 3 and for 2^{nd} level as 4. Taxogen splits the root topic into 3 subtopics: 'cardigan_sweater', 'ankle_sock' & 'purse'. These subtopics represents the major product categories that are found in the dataset.

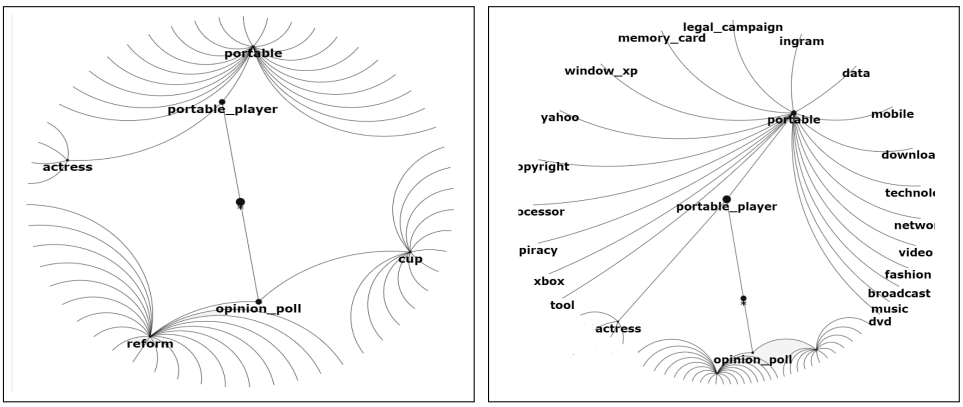


(a) Taxonomy on amazon fashion review data

(b) One of the second level nodes expanded

Figure 3.19: Taxonomy created by Taxogen on Amazon Fashion Review dataset with one of the second level nodes expanded.

On the right part of Figure 3.19, the second level node 'purse' is expanded where the fine grained topics consist of semantically coherent data. For example, the node 'purse' contains 'tote_bag', 'watch_band', 'necklace', 'checkbook_holder', 'wig_cap'. This strongly agrees with our fashion product category relationship.



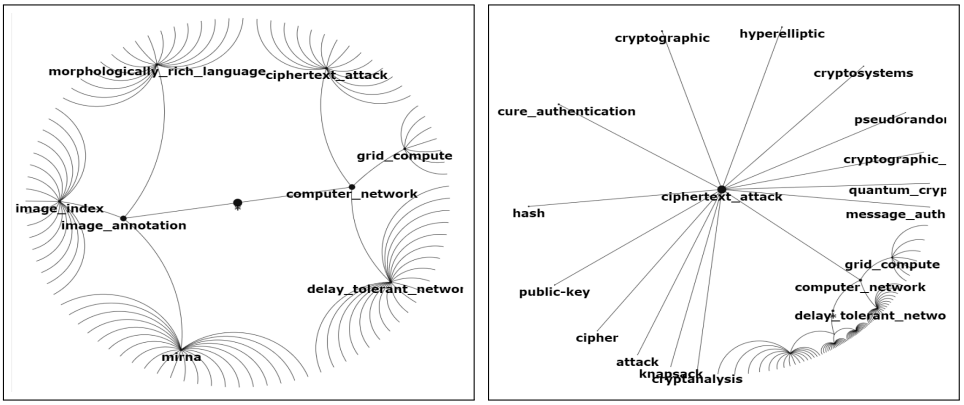
(a) Taxonomy created by Taxogen on BBC dataset

(b) One of the second level nodes expanded

Figure 3.20: Taxonomy created by Taxogen on BBC news article dataset with second level node expanded.

Similarly, in Figure 3.20 (a), we show the taxonomy created by Taxogen for BBC news article dataset. Here each parent topic is split uniformly. The cluster value for each level is used as 2. The root level is split into two subtopics: 'opinon_poll', 'portable_player'. This splits the subtopics into two broader categories, where one of them represents business, politics & sports related articles, and other one is related to technology & entertainment. These clustering truly represents the nature of the BBC news article dataset where we take articles from business, entertainment, technology, politics, sports. Furthermore, as we can see in Figure 3.20 (b), where 'portable' node under 'portable_player' node is expanded. We can see here that, Taxogen makes the taxonomy of news article categories from BBC news article data where categories

are identified at appropriate levels.



(a) Taxonomy created by Taxogen on DBLP dataset

(b) One of the second level nodes expanded

Figure 3.21: Taxonomy created by Taxogen on DBLP dataset with second level node expanded.

Finally, in Figure 3.21(a), we show the taxonomy created by Taxogen for DBLP citation titles dataset. Here each topic is split uniformly as well. The cluster value 2 is used for splitting the 1^{st} level node. Each node in 2^{nd} level is split using cluster value 3. The root level is split into two subtopics: 'image_annotation' and 'computer_network'. This clustering of words from different titles broadly segregate the data where in one part, it clubs words from computer vision, bio-informatics and natural language processing. Whereas in other part, all the words from architecture, network and security. This greatly represents the true relationship among different titles which were hidden in the citation network data. Finally, in Figure 3.21(b), the further split of one of the 2^{nd} level node 'cyphertext_attack' which is under the node 'computer_network' is highlighted. As we can see, the algorithm successfully collected all the relevant words which are solely related to computer security. We can conclude from this qualitative analysis that, Taxogen algorithm can successfully extract taxonomy that is hidden in textual data.

4
Enriching Taxonomy

In this thesis, we only study the insertion of a new concept node in a taxonomy whenever new data from the same domain of knowledge is available. We use the transfer learning[1] technique to get word embeddings from new data. In addition, we use similarity scores to identify the correct parent node in the existing taxonomy in order to add the new word. In the following section, we describe in detail our novel approach 'TaxoTL' algorithm.

4.1 TaxoTL

To implement the logic to modify a taxonomy, first we save the trained model from existing algorithm (e.g. Taxogen). Next, we instantiate a new blank word2vec model based on the new data. After this, we load the word embeddings from the trained model to the new blank model. Then, we train this new model. Once the training is done, we use cosine similarity between the each new word's embedding and each embedding of the nodes from existing taxonomy. Based on similarity scores, we add the new word in the existing taxonomy where score is best by traversing the taxonomy tree. The basic algorithm uses word2vec[15] to find the word embedding from the new data. TaxoTL is defined in Algorithm 4.

[1] See also `https://en.wikipedia.org/wiki/Transfer_learning`

Algorithm 4: TaxoTL

Input: New corpus D_{new} ; new topic C_{new}; existing taxonomy T ; existing trained model M
Output: New taxonomy T_{new}

$M_l \leftarrow LoadModel(M)$
$M_l.Retrain(D_{new})$
for *(each word* $w_i \in C_{new}$*)* **do**
  $\mathbf{w}_{vec} \leftarrow GetVector(M_l, w_i)$
  $similarity\_score \leftarrow []$
  for *(each node* $n \in T$*)* **do**
    $\mathbf{w}_n \leftarrow VectorLookup(T, n)$
    $similarity\_score[n] \leftarrow CosineSimilarity(\mathbf{w}_n, \mathbf{w}_{vec})$
  $similarity\_score\_sorted \leftarrow Sorted(similarity\_score)$
  if *(*w_n *is first index in* $similarity\_score\_sorted$*)* **then**
    $T_{new} \leftarrow T.addnode(w_n, w_i)$
Return T_{new}

Algorithm 4 takes as inputs new corpus data D_{new}, new topic names C_{new}, existing trained model M and existing taxonomy T. At the beginning of the algorithm, it first loads the existing model M into M_l using $LoadModel(.)$ function. Next, the algorithm retrains the loaded model M_l using new corpus data D_{new}. Now, the algorithm iterates for each word w available in C_{new} and get the corresponding word embedding $\mathbf{w}_{vec}$ from M_l using the $GetVector(.)$ function. After this, for each word, it iterates again for all nodes in the existing taxonomy T and gets the vector of each node using $VectorLookup(.)$. Then the algorithm calculates the cosine similarity using $CosineSimilarity(.)$ between the word embeddings of each word from C_{new} and every node embeddings from T. Then add the new word to that node of T for which the cosine similarity value is highest i.e. 1. These two iterative processes eventually modify the existing taxonomy T by adding new words from C_{new} and return T_{new}.

Example 4 (TaxoTL Algorithm)**.** *Let's assume that we have an existing taxonomy given in Figure 4.2. In addition, we have a new data set from the same domain of knowledge, i.e. from "Database Management System". After the prepossessing, we retrain the already saved trained model with this new data. Then we apply the concept of "Transfer Learning" to extract the learned representation of new words or topics as available from the new data set. Eventually, these embeddings are used to enrich the existing taxonomy by means of similarity measures.*

Step 1 *At first, we load the already trained model that was saved. Then, we retrain this loaded model on the new dataset. The process is depicted in Figure 4.1. The existing taxonomy that needs to be enriched is also available at the beginning. In this example the same is seen in the Figure 4.2.*

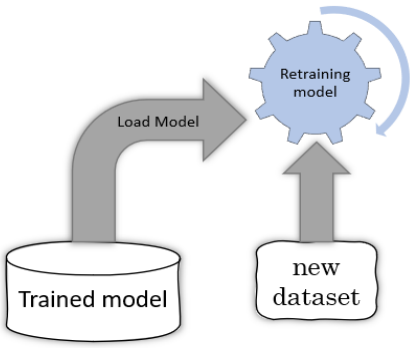


Figure 4.1: Retraining the existing model. First the model is loaded from disk into memory. Then the loaded model is retrained based on the new dataset.

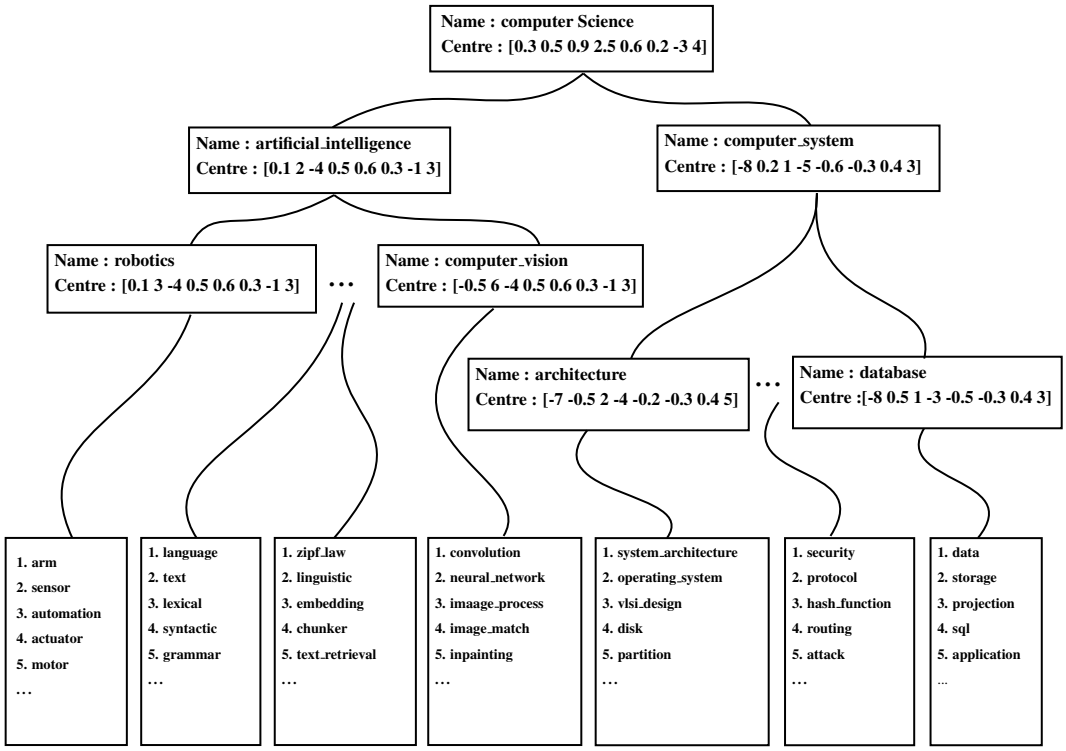


Figure 4.2: Existing Taxonomy to be enriched by TaxoTL

Step 2 *Let's say that we have a new term in new dataset as 'query'. We extract the embedding of this term from the retrained model. The value is [**-9.0,-0.2, 1, -3, 0.4, -0.15, 1, 2**].*

Step 3 *Now, we extract each of every node of the existing taxonomy by visiting the taxonomy tree using tree traversal algorithm.*

Step 4 *Now, in this step, we measure the cosine similarity between the embedding of 'query' and each of*

the node's embedding that we collected from the existing taxonomy. We use this similarity score to find the node of the taxonomy to which the term 'query' is most similar i.e. where the score is maximum. Here, the similarity scores w.r.t every the branch nodes (assuming existing leaves do not have any children) is calculated and available in Table 4.1.

Node Name	Cosine Similarity
Artificial Intelligence	-0.029525469369307444
Robotics	-0.0307924300118451
Speech	0.121751157196299
NLP	-0.20548227674242
Computer Vision	0.0384822663647254
System	0.9616400471365544
Architecture	0.917039797156302
Network	0.9332197571660438
Database	**0.9818993128302635**

Table 4.1: Cosine similarity scores between node of taxonomy and new term embedding

Step 5 *Finally, we've added the term 'query' as a child node under the node 'Database' of the taxonomy found in previous step based on similarity score. This is shown in Figure 4.3.*

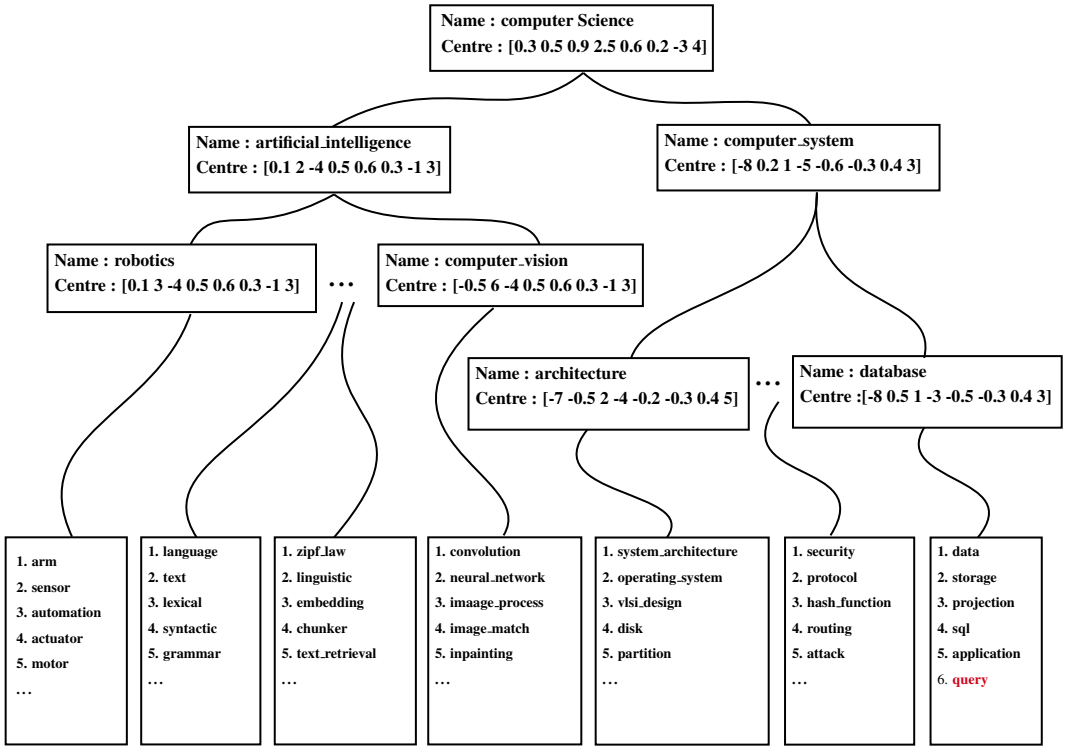


Figure 4.3: Existing Taxonomy enriched with new word. The new keyword added in the taxonomy is highlighted in red colour.

4.2 Evaluation & Results

In this section, we describe experiment results of the TaxoTL algorithm. First, we describe the datasets we used, the evaluated algorithms and the parameter settings for the experiment. Then, we analyse in detail the experimental results.[2]

4.2.1 Experimental Setup

Datasets For all our experiments, we use three real world datasets. We use these different datasets in order measure the generalization performance of our TaxoTL algorithm. These different datasets also help to measure the performance in unbiased way leading to accurate insight.

1. Women's E-Commerce Clothing Reviews[10]: Women's Clothing E-Commerce dataset revolving around the reviews written by customers. It has 26797 customer reviews for women clothing. We take 20000 reviews from it for our experiment.

2. DBLB Network: This contains approximately 4800 citations. In this dataset, titles of computer science papers from the areas of information retrieval, computer vision, robotics, security & network, and machine learning.

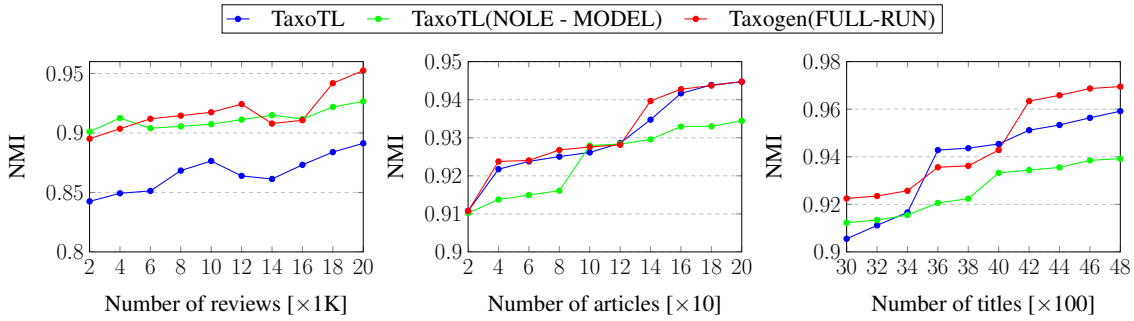
3. News Article: This is a reusable publicly-available dataset for "media bias" studies. The content of this dataset is publish date, title, subtitle and text for 3824 news articles. We take 200 news articles from these dataset.

Evaluated Algorithms In this experiment, we compare the performance of TaxoTL against Taxogen and Taxogen nole. The choice of Taxogen is motivated by the fact that this algorithm outperforms the rest of algorithms for the generation of taxonomy.

4.2.2 Accuracy Results

We conduct our test on the three datasets where we use Taxogen to run fully on each dataset (existing and new data). We use TaxoTL algorithm on the Taxogen's model as well as Taxogen nole model. We calculate average Normalized Mutual Information (NMI) score for each of these cases and plot the same in Figure 4.4. In each dataset, we consistently increase the volume of new data and then we run each of the above cases where training is happening from scratch for Taxogen full run. Whereas, TaxoTL first loads the already trained model (from Taxogen or Taxogen nole) that is initially saved on the disk. After that, it retrains the model using only the new dataset. Our novel approach TaxoTL makes use of 'Transfer Learning'. Due to the lack of new data volume, the algorithms (e.g. Taxogen) can not train the model efficiently only based on the new data. Hence, we always have to run Taxogen on full dataset that includes original old dataset as well as new data. On the other hand, our transfer learning based TaxoTL helps in such scenarios where it uses only new data for training the model in order to get the word embeddings. Moreover, for each dataset, the data belongs to same domain of knowledge from where the already trained model is being used. This homogeneity in data is helping our TaxoTL algorithm to train the model easily and efficiently.

[2]The code is available at: `https://github.com/milibiswas/taxonomy-enrichment`



(a) NMI on Fashion Review data (b) NMI on news article data (c) NMI on DBLP citation network data

Figure 4.4: NMI scores for different algorithm on varying data volume of corpus data

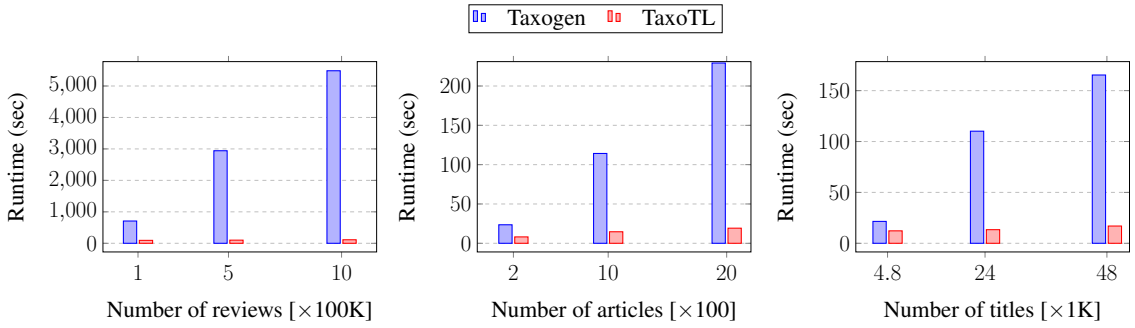
In Figure 4.4, we observe that TaxoTL and and Taxogen full run have similar performance in all datasets except fashion review. In Figure 4.4(a), we see the lower performance of TaxoTL on Taxogen model. Since, Taxogen uses local embedding process, therefore, it generates multiple models each associated with local embedding. These multiple models are saved on disk. Our TaxoTL sequentially loads these multiple models that contain overlapping words each having different embeddings. In this loading process, some of the correct embeddings of these overlapping words are overwritten by embeddings from wrong model. This leads to poor performance by TaxoTL. However, we also see that, TaxoTL on Taxogen nole model performs similar to Taxogen full run. We believe that, TaxoTL on Taxogen nole has no overlapping models because Taxogen nole does not have any local embeddings. It only uses one global embedding.

For news article and DBLP datasets, we see that TaxoTL on Taxogen model performs better compared to what we have seen for fashion data. This is due to the fact that, the number of overlapping words in different local embedding models saved on disk are less or none. Therefore, we see better scores for these datasets.

4.2.3 Efficiency Results

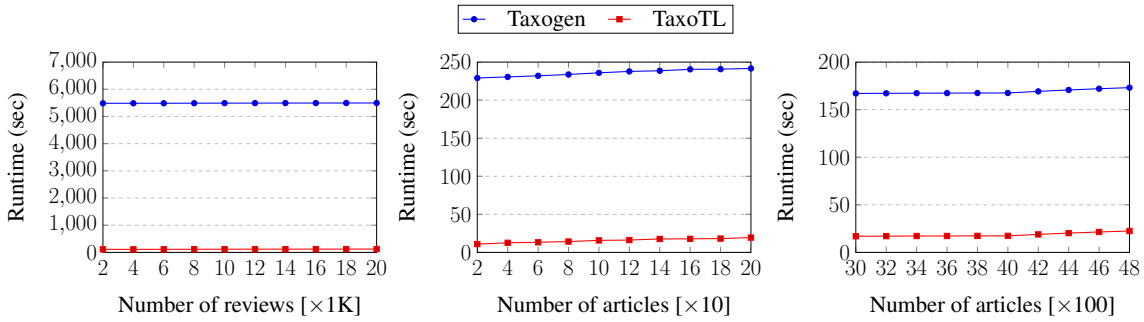
In this section, we compare the runtime and memory consumption of TaxoTL against Taxogen. Here, first we save Taxogen models setting input data volume as 10%, 50% and 100% for all the three datasets. Then we use these models while training TaxoTL and capture the running time and memory consumption as 10%, 50% and 100% input data volume. We also capture the runtime and memory consumption of Taxogen by running the algorithm with previously saved 10%, 50% and 100% data and adding each with new additional data. We give details of the results below. In addition, we also measure the runtime of TaxoTL and Taxogen algorithms while fixing the full original data volume of each datasets to 100% and then we increase the new additional data volume. Figure 4.6 depicts the results.

Runtime analysis We measure the runtime of TaxoTL and Taxogen algorithms on all the three datasets in order to compare the efficiency of the algorithms. We plot the results for varying input data volume in Figure 4.5.



(a) Fashion Review Dataset (b) News article Dataset (c) DBLP dataset

Figure 4.5: Runtime of different algorithms with varying volume of input data



(a) Fashion Review Dataset (b) News article Dataset (c) DBLP Dataset

Figure 4.6: Runtime of different algorithms with varying volume of input data

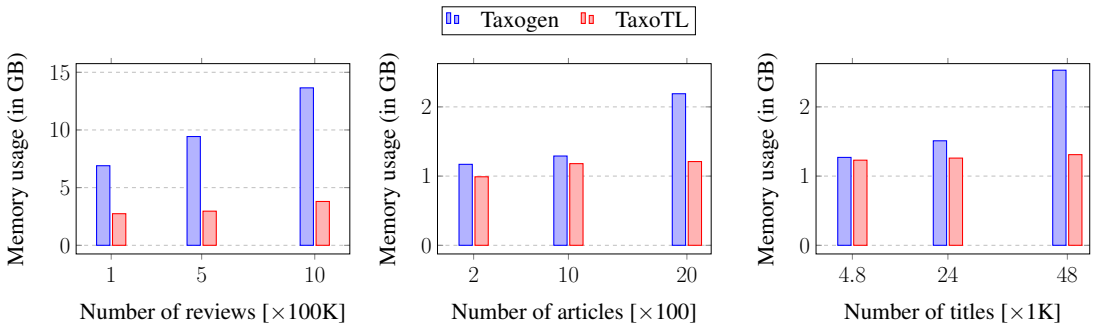
Figure 4.5 shows that the runtime of TaxoTL increases almost linearly with the data size while the runtime of Taxogen increases almost exponentially. We notice that the runtime difference between TaxoTL and Taxogen increases significantly with the the data size. We see in fashion review for 1M reviews data, TaxoTL is $50\times$ faster than Taxogen where TaxoTL takes 112 seconds whereas Taxogen takes 5484 seconds. We also notice that in DBLP for 48K titles as well as in BBC news articles for 2K articles, TaxoTL is approximately $10\times$ faster than Taxogen. In news article for 2K articles, TaxoTL takes 20 seconds and Taxogen takes 229 seconds. In DBLP for 48K titles, TaxoTL takes 17 seconds and Taxogen takes 165 seconds. Since the data volume of fashion review dataset is significantly higher than news article and DBLP dataset volume, therefore the run time of Taxogen is much higher for fashion review data.

Figure 4.6 shows that, if we use the full original dataset, the runtime increases almost linearly for both Taxogen and TaxoTL. We also notice that, the difference in runtime between the Taxogen and TaxoTL remains significantly high. We argue that, in Figure 4.6, as the increase of new data volume is not very significant compared to full data volume, therefore, we see linear pattern in runtime increase along with data volume increments. For fashion review data, we use 1 million reviews and then increase the new data volume by 10%. We see that TaxoTL is almost $50\times$ faster than Taxogen. Since the new dataset size is very less compared to the original data volume, therefore, increasing in new data size does not impact the runtime of Taxogen as well as TaxoTL and we get constant runtime gap between the algorithms. We also see similar pattern for DBLP and news article where TaxoTL is almost $10\times$ faster than Taxogen. We see in fashion review for 20k reviews, TaxoTL takes 123 seconds whereas Taxogen takes 5495 seconds. We also notice that in DBLP for 4.8K titles, TaxoTL takes 22 seconds and Taxogen takes 173 seconds. In news

article for 200 articles, TaxoTL takes 20 seconds and Taxogen takes 241 seconds.

We can justify that, our algorithm TaxoTL is taking less time than the Taxogen because TaxoTL uses only new incremental data while training the model.

Memory consumption analysis We also measure the memory consumption of TaxoTL and Taxogen algorithms on all the three datasets. We run every algorithm with relevant input data volume and capture the memory usage of each algorithm. We then average the memory consumption over the time of the algorithms and plot the data in Figure4.7. We measure physical memory, shared memory and virtual memory consumption.



(a) Memory consumption of TaxoTL & Taxogen for Fashion Review data

(b) Memory consumption of TaxoTL & Taxogen for news article data

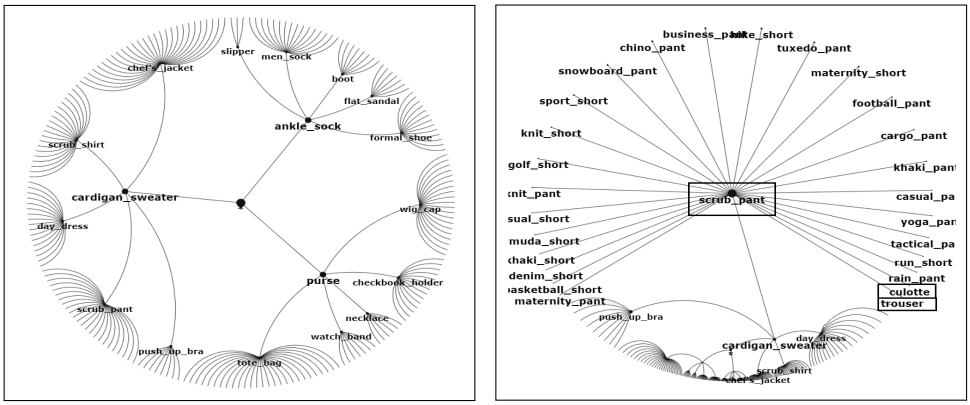
(c) Memory consumption of TaxoTL & Taxogen for DBLP data

Figure 4.7: Memory consumption of different algorithms with varying volume of input data

The resuts show that the memory consumption of TaxoTL is increasing almost linearly along with the data size. However, the memory consumption of Taxogen is increasing almost exponentially as we increase the volume of input data. We also notice that the difference in memory consumption between TaxoTL and Taxogen increases exponentially as the data volume increases. This result shows that our TaxoTL is very memory efficient compared to Taxogen algorithm. Taxogen processes complete data that encompasses original full as well as new additional data which requires sufficient amount of large memory. Whereas, our TaxoTL only needs to load the trained embeddings from saved model to initialize the word2vec model's projection layer and then retrain model using the new data only. This requires significant less memory because the new data size is smaller than the full data size and the train embeddings size also is very less compared to whole corpus data as it only contains embeddings of words and contexts.

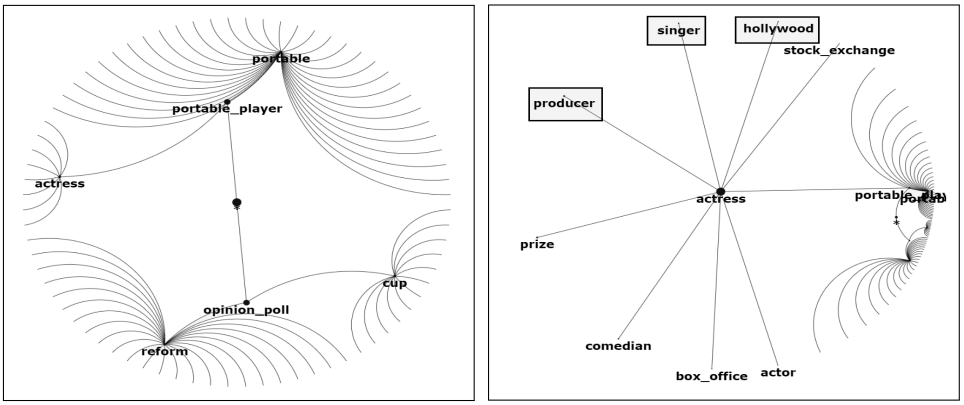
4.2.4 Qualitative Results

In this Section, we show the taxonomies modified by TaxoTL using all three new datasets. In Figure 4.8(a), the full taxonomy for new fashion review data is shown. This is created by running TaxoTL based on the model saved from Taxogen algorithm. Moreover, here, we take taxonomy to be enriched from the Taxogen algrothm. Here, the TaxoTL adds new words e.g. 'culotte', 'trouser', 'tutu','blouse' etc. by learning the corresponding embeddings. Here, this learning is greatly improved due to our 'Transfer Learning' approach. In Figure4.8(b), we expand the third level node 'scrub_pant' which represents the all the pant and short category. We can see that all the new pant related keywords are added under this 'scrub_pant' node. This is highlighted in the Figure 4.8(b)



(a) Taxonomy created by TaxoTL (b) A 2^{nd} level node of TaxoTL

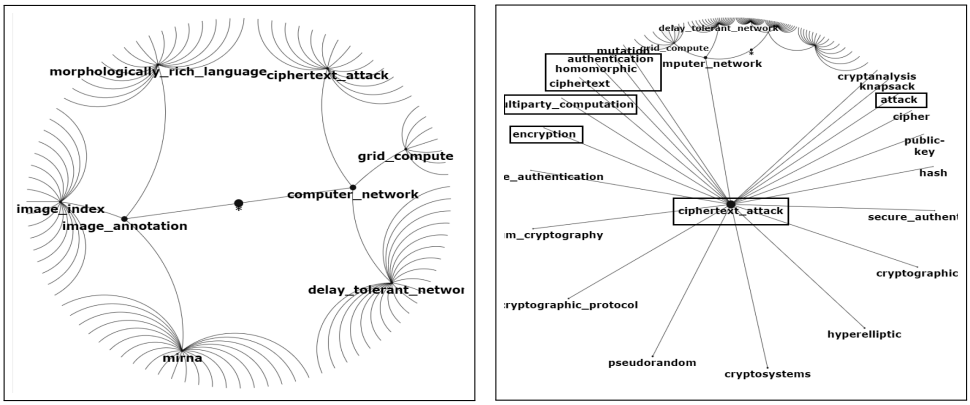
Figure 4.8: Modified taxonomy by TaxoTL algorithm along with one second level node expanded for fashion review data.



(a) Taxonomy modified by TaxoTL (b) A 2^{nd} level node of TaxoTL

Figure 4.9: Modified taxonomy by TaxoTL algorithm along with one second level node expanded for news article data.

In Figure 4.9, we show the modified taxonomy created by TaxoTL for new news article dataset. Here, the algorithm adds new keywords e.g. 'hollywood', 'singer', 'producer', 'television' etc. under 'actress' node because these new words are are related to technology and entertainment group. On similar note, TaxoTL adds new words from new DBLP tiltes in the corresponding existing taxonomy. In Figure 4.10 the newly added words e.g. 'attack', 'encryption', 'authentication', 'homomorphic', 'multiparty_computation', 'ciphertext' etc. under the node 'ciphertext_attack'. These new words are from computer security group that the algorithm correctly adds in the taxonomy.



(a) Taxonomy modified by TaxoTL (b) A 2^{nd} level node of TaxoTL is shown.

Figure 4.10: Modified taxonomy by TaxoTL along with one second level node expanded for DBLP data.

5
Conclusion and Future Work

In this thesis, we study algorithms capable of learning taxonomy from a text corpus. We see two main categories of algorithms where one makes use of word2vec (skip-gram) that is essentially a neural network model, and the others are using the nCRP probabilistic framework as a prior distribution while building the tree-like taxonomy from the corpus. Our experimental results show that word2vec based algorithm generally performs well because it heavily relies on the context of the target words, whereas, the nCRP based model performs sub-optimal as it is based on a probabilistic model that seeks the probability distribution of words in textual data. In most of the cases, nCRP based algorithm is not able to encode the context while learning the embeddings of the target words from the text corpus data. Additionally, we propose a novel approach TaxoTL that uses transfer learning in order to learn embeddings of new words for enriching the existing taxonomy when a new textual dataset is available from the same domain of knowledge. Our measured NMI scores show that our novel approach based on the transfer learning technique can enrich the taxonomy by identifying the correct concept nodes using the similarity score among different learned word embeddings. This score also shows that our TaxoTL algorithm's accuracy is almost similar to Taxogen. Moreover, our TaxoTL has superior efficiency in terms of running time as well as memory consumption. We observe that TaxoTL is more than $50\times$ faster than Taxogen when running on Fashion review data and $10\times$ faster on DBLP & news article datasets.

Finally, we observe that there are areas where we still need to do further research. For example, it would be of interest to work on an automatic cluster detection algorithm. This will enable many taxonomy generation algorithms that can generate the hidden taxonomy from corpus automatically without any need of manually given information about the number of clusters in each level from outside. Secondly, it would be useful to work on cases where we can extract a sub-taxonomy from a new dataset and then merge that sub-taxonomy with the original taxonomy. There are different proposed algorithms available which claim to work for streaming data instead of batch mode for the skip-gram model.

Bibliography

[1] Abien Fred Agarap. *Statistical Analysis on E-Commerce Reviews, with Sentiment Classification using Bidirectional Recurrent Neural Network*. Mar. 2018. DOI: 10.13140/RG.2.2.16988.08321.

[2] David J. Aldous. "Exchangeability and related topics". In: *École d'Été de Probabilités de Saint-Flour XIII — 1983*. Ed. by P. L. Hennequin. Berlin, Heidelberg: Springer Berlin Heidelberg, 1985, pp. 1–198. ISBN: 978-3-540-39316-0.

[3] Philipp Cimiano, Andreas Hotho, and Steffen Staab. "Learning Concept Hierarchies from Text Corpora using Formal Concept Analysis". In: *CoRR* abs/1109.2140 (2011). arXiv: 1109.2140. URL: http://arxiv.org/abs/1109.2140.

[4] tianru dai. *News Articles*. Version V1. 2017. DOI: 10.7910/DVN/GMFCTR. URL: https://doi.org/10.7910/DVN/GMFCTR.

[5] *DBLP Dataset*. URL: https://dblp.uni-trier.de/xml/.

[6] *Fashion Taxonomy*. URL: https://www.voguebusiness.com/technology/taxonomy-is-the-new-fashion-tech-essential-the-yes.

[7] *Github Code repository*. URL: https://github.com/milibiswas/taxonomy-enrichment.

[8] Yoav Goldberg and Omer Levy. "word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method". In: *CoRR* abs/1402.3722 (2014). arXiv: 1402.3722. URL: http://arxiv.org/abs/1402.3722.

[9] Derek Greene and Pádraig Cunningham. "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering". In: *Proc. 23rd International Conference on Machine learning (ICML'06)*. ACM Press, 2006, pp. 377–384.

[10] R. He and J. McAuley. "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering". In: *WWW* (2016).

[11] *Hierarchical Taxonomy Aware Network Embedding KDD '18: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining Pages 1920–1929*. London, United Kingdom: Association for Computing Machinery, 2018. ISBN: 9781450355520.

[12] Nobuhiro Kaji and Hayato Kobayashi. "Incremental Skip-gram Model with Negative Sampling". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 363–371. DOI: 10.18653/v1/D17-1037. URL: https://www.aclweb.org/anthology/D17-1037.

[13] Yuan-Pin Lin and Tzyy-Ping Jung. "Improving EEG-Based Emotion Classification Using Conditional Transfer Learning". In: *Frontiers in Human Neuroscience* 11 (2017), p. 334. ISSN: 1662-5161. DOI: 10.3389/fnhum.2017.00334. URL: https://www.frontiersin.org/article/10.3389/fnhum.2017.00334.

[14] Chandler May et al. “Streaming Word Embeddings with the Space-Saving Algorithm”. In: *CoRR* abs/1704.07463 (2017). arXiv: 1704.07463. URL: http://arxiv.org/abs/1704.07463.

[15] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *CoRR* abs/1310.4546 (2013). arXiv: 1310.4546. URL: http://arxiv.org/abs/1310.4546.

[16] S. J. Pan and Q. Yang. “A Survey on Transfer Learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359.

[17] Anand Rajaraman and Jeffrey David Ullman. “Data Mining”. In: *Mining of Massive Datasets*. Cambridge University Press, 2011, 1–17. DOI: 10.1017/CBO9781139058452.002.

[18] Juan Enrique Ramos. “Using TF-IDF to Determine Word Relevance in Document Queries”. In: 2003.

[19] Xin Rong. “word2vec Parameter Learning Explained”. In: *CoRR* abs/1411.2738 (2014). arXiv: 1411.2738. URL: http://arxiv.org/abs/1411.2738.

[20] Y. Song et al. “Automatic Taxonomy Construction from Keywords via Scalable Bayesian Rose Trees”. In: *IEEE Transactions on Knowledge and Data Engineering* 27.7 (2015), pp. 1861–1874.

[21] A. K. Sood and S. Zeadally. “A Taxonomy of Domain-Generation Algorithms”. In: *IEEE Security Privacy* 14.4 (2016), pp. 46–53.

[22] Pucktada Treeratpituk, Madian Khabsa, and C. Lee Giles. “Graph-based Approach to Automatic Taxonomy Generation (GraBTax)”. In: *CoRR* abs/1307.1718 (2013). arXiv: 1307.1718. URL: http://arxiv.org/abs/1307.1718.

[23] Arun Varghese, George Agyeman-Badu, and Michelle Cawley. “Deep learning in automated text classification: a case study using toxicological abstracts”. In: *Environment Systems and Decisions* (2020). ISSN: 2194-5411. DOI: 10.1007/s10669-020-09763-2. URL: https://doi.org/10.1007/s10669-020-09763-2.

[24] Kafeng Wang et al. “Pay Attention to Features, Transfer Learn Faster CNNs”. In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=ryxyCeHtPB.

[25] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. “A survey of transfer learning”. In: *Journal of Big Data* 3.1 (2016), p. 9. ISSN: 2196-1115. DOI: 10.1186/s40537-016-0043-6. URL: https://doi.org/10.1186/s40537-016-0043-6.

[26] *Women’s E-Commerce Clothing Reviews*. URL: https://www.kaggle.com/nicapotato/womens-ecommerce-clothing-reviews.

[27] Chao Zhang et al. “TaxoGen: Unsupervised Topic Taxonomy Construction by Adaptive Term Embedding and Clustering”. In: *CoRR* abs/1812.09551 (2018). arXiv: 1812.09551. URL: http://arxiv.org/abs/1812.09551.

[28] F. Zhuang et al. “A Comprehensive Survey on Transfer Learning”. In: *Proceedings of the IEEE* (2020), pp. 1–34.