UNIVERSITY OF AMSTERDAM

MSc ARTIFICIAL INTELLIGENCE
MASTER THESIS

Large Scale Hierarchical Text Classification

by
THARANGNI HARSHA SIVAJI
11611065

June 25, 2019

36 EC
January 2019 - July 2019

Supervisor:
Dr. E. KANOULAS (UvA)
Dr. P. TILLMAN (Elsevier)

Assessor:
Dr. I. MARKOV (UvA)

Abstract

One of the main challenges in large-scale hierarchical classification is the inability to predict classes which have few training samples. In this paper we use two methods to boost classification performance by leveraging the information encoded in an hierarchy. The first approach alleviates the rare category problem by incorporating dense text representations along with a regularization framework dependent on the relationships between classes. The other approach uses neural embedding models to learn the hierarchy first and classify documents based on the learnt hierarchical relations. Additionally, we also evaluate all our experiments with hierarchical metrics to see how the information is encoded in the designed models.

Acknowledgements

I would like to thank Philip Tillman for introducing this interesting topic to me. His guidance and supervision in the office helped me finish this thesis and kept me motivated. I would also like to thank Evangelos Kanoulas for guiding me and for all the interesting discussions we had throughout this period. It wouldnt have been possible for me to finish this thesis without their help. I am also very grateful to Ehsan Khoddam for his excellent practical advice on the different methods implemented in this paper. I thank Ilya Markov for agreeing to be a part of my defense committee. I thank my parents for their endless motivation and support. Finally, I thank my friends for proof-reading my report, giving me countless advice and helping me get through this period.

Contents

Chapter 1

Introduction

Text classification is the process of automatically assigning single or multiple labels to documents. Classical text classification approaches include binary, multiclass and multilabel classification. In binary classification a single (class) label is assigned to a document from two labels, whereas in multiclass classification a single label is assigned to a document from multiple labels. Similarly, tagging documents as spam/ham and sentiment classification are examples of binary and multiclass classification respectively. On the other hand, in multilabel classification documents are assigned to more than one label at a time from a set of labels. For instance, tagging a movie or book with multiple genres is an example of multilabel classification. These classification approaches ignore any form of inherent relationship between the labels and classify it *flatly* i.e., they consider that all labels are independent of each other.

Essentially, both binary classification and multiclass classification follow a one-to-one mapping between the documents and labels whereas multilabel classification follows a one-to-many mapping between documents and labels. The most common approach for multilabel classification is the *one-vs-rest classification* or *flat classification* approach, where n binary classifiers [1] are trained with one label at a time leaving the rest out. After training, these classifiers predict the most likely label from n labels.

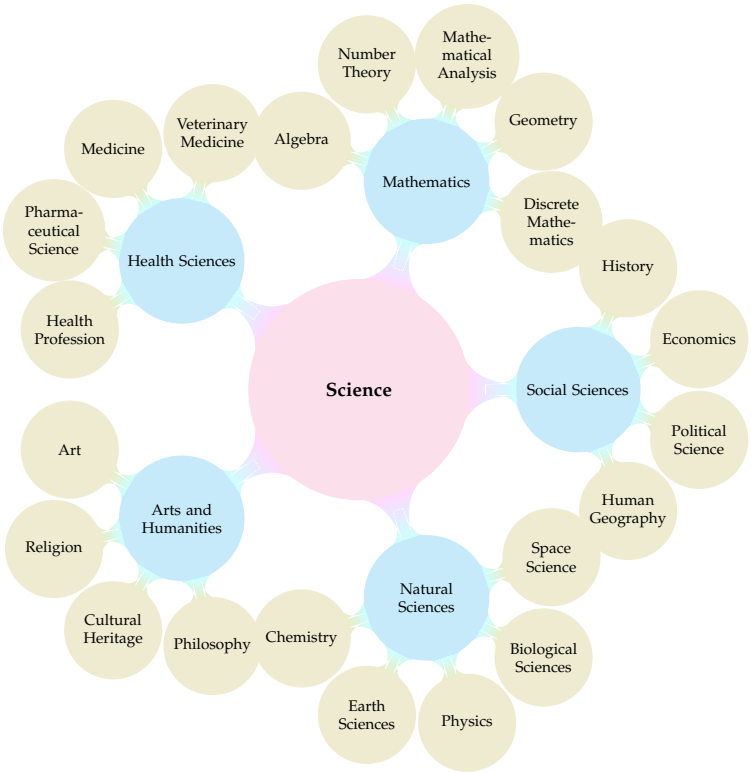


Figure 1.1: *Snippet of OmniScience hierarchy.* ***Science*** *is the main class from which other classes are derived. Different coloured classes represent different levels of the hierarchy. Refer to Chapter 3 for detailed information*

[1] n is the number of classes/labels

In multilabel classification, the inherent relationships between classes can be represented using hierarchies. Formally, a hierarchy refers to a partially ordered set where the partial ordering is due to the different relationships between elements of the set such as parent-child or sibling relationship.

There are two main relationships that exist within a hierarchy – 'parent-child' and 'siblings' [2]. In such a hierarchy, a root node refers to a node which does not have any incoming edges and therefore is the node from which the tree grows. Leaf nodes do not have any outgoing edges. Internal nodes are all the nodes in between the root and the leaf, which have both incoming and outgoing edges. Simple hierarchies have only one root node with each node having only incoming edges from one parent node. Such hierarchies are known as *tree-based hierarchies* (Figure 3.2b). Complex hierarchies on the other hand, can have more than one root or can have nodes with more than one parent (or both). These hierarchical structures form a *directed acyclic graph* (DAG - Figure 3.2a). Additionally, taxonomies such as NCBI[3], OmniScience (Figure 1.1) and online web databases such as the shopping product hierarchy of Amazon[4] are examples of large scale hierarchical datasets.

As the amount of data increases, the number of labels associated with it also increases. However, the performance of flat classifiers in dealing with large number of labels and multi-label classification tends to be poor (Babbar et al., 2013) as these classifiers fail to account for categories with few data samples. Another major drawback of classifying labels *flatly* is that it ignores any inherent hierarchy between categories and trains an independent one-vs-rest binary or multi-class classifiers corresponding to each of the 'leaf node/category'. Often, these labels have some inherent relationship with each other i.e., it could be a part of a knowledge base or a taxonomy. One possible way of improving the performance of the large multi-label classifiers is to leverage the information encoded in these relationships.

Large Scale Hierarchical Classification

Manual classification is a non-trivial task as it involves categorizing unlabelled instances and identifying relationships between concepts by domain experts based on semantic relationships between classes. Although this gives users (domain experts) more control over classification, it is both expensive and time consuming. Moreover, the complexity of classifying is directly proportional to the amount of data thereby, making such methods of classifying infeasible for large-scale datasets.

Large scale datasets not only have millions of instances but also have an increasing number of labels/classes to classify. Moreover, these classes have inherent relationships within that can be viewed as a hierarchy in the form of trees or graphs. Thus, there is a need to develop efficient machine learning approaches to categorize millions of unknown instances into hierarchical classes. This problem is referred to as *Large Scale Hierarchical Classification* (LSHC) task.

The most common challenges in LSHC are:

1. **Single label vs. Multi label classification**: Unlike the ease of classifying one-to-one relationships in single label classification, mapping one-to-many relationships for multilabel classification is quite difficult. Moreover, if these labels belong to a hierarchy, then it becomes difficult to assign multiple labels, since these labels can belong to different branches in the hierarchy.

2. **Leaf node prediction vs. Internal node prediction**: In hierarchical classification each instance can either be associated to labels that lie on a path from root to leaf node (leaf node prediction) or it can stop at an internal node (internal node prediction). Assigning instances to internal nodes is difficult compared to internal nodes (Naik and Rangwala, 2017).

3. **Rare classes**: Most large-scale datasets have heavily unbalanced class distributions. For example, 75% of categories in DMOZ[5] and Yahoo![6] directory have five or fewer positive instances. This heavily skewed distribution of classes makes it difficult to generalize classification models as it is prone to over-fitting (Babbar et al., 2013).

4. **Feature selection**: Identifying discriminative features is important in text classification since it affects classification and runtime performance along with memory consumption. However, feature selection is non-trivial with large-scale datasets since they are high dimensional and have thousands to millions of features.

[2] More information in Chapter 3
[3] https://www.ncbi.nlm.nih.gov/taxonomy
[4] https://www.amazon.com/gp/site-directory
[5] Archived, shutdown in 2017. https://web.archive.org/web/20131030121541/http://www.dmoz.org/
[6] Archived, shutdown in 2015. https://web.archive.org/web/20131231053351/http://dir.yahoo.com/

5. **Learning hierarchies**: Information pertaining to hierarchical relationships need to be incorporated while training models for overall model generalizability. Often, parent-child and sibling relationships are widely used in literature to improve classification performance (Babbar et al., 2013).

6. **Scalability**: Dealing with large scale datasets requires algorithms that can be scalable either in a distributed fashion or a parallel framework. However, due to hierarchical dependencies between different classes and due to varying depth of hierarchies, it is difficult to parallelize model learning.

7. **Error Propagation**: One of the most common way to learn hierarchies is through path distances from root to predicted node. However, these decisions can become quite invalid at times when predictions are incorrect. Such erroneous calculations are often propagated to lower levels in the hierarchy affecting overall prediction performance. On the other hand, flat classifiers do not suffer from any error propagation problems since they rely on only a single decision.

Due to these challenges, it is not easy to solve the large-scale classification problem. Most of the existing techniques solve a specific formulation of the problem (Charuvaka and Rangwala, 2015; Naik and Rangwala, 2016).

Assumptions & Research Question

In this work, we focus only on *tree-based hierarchies* and convert any directed acyclic graphs to trees. This is to see how the parent-child relationship within trees affects classification. We try to address the following questions:

- How can we best *leverage the information encoded in the hierarchy* of a taxonomy of labels to boost classification performance?
  - Does improving the *text representation* lead to better classification performance wherein hierarchical information is used.
  - Does *learning hierarchy explicitly* using a neural embedding model capture class information better than the above method which in turn improves classification performance.
- How can the performance of these models be evaluated using hierarchical metrics apart from standard metrics?

This thesis is structured as follows. Chapter 2 discusses about the related work and research done in the field of LSHC. Chapter 3 describes the dataset used for experimentation. Chapter 4 gives details about the evaluation metrics used for assessing performance of models. Chapter 5 gives a detailed overview of the model used for leveraging class information. Chapter 6 suggests a general neural embedding model which tries to learn hierarchy to see if it improves classification performance. Chapter 7 discusses the results from the different experiment gives an overall analysis of the methods discussed. Finally, Chapter 8 concludes this thesis and also gives directions for future work.

Chapter 2

Literature Review

Over the past few years, challenges in LSHC has been extensively researched and many authors have provided an array of solutions to these challenges.

(Babbar et al., 2013) devised a theorem that explains why flat classification fails to perform well for large scale taxonomies when data is highly unbalanced. Flat classifiers have to make a *single decision* for many categories - this can be a problem if a category is under represented i.e. it has few training samples (unbalanced dataset). However, hierarchical classifiers have to make *several intermediary decisions* before reaching the final category. Due to this they are not affected by the imbalance problem as severely as flat classifiers. They prove this theoretically by introducing a bound on the generalization error of a top-down multiclass hierarchical classifier. Conversely, they also prove that flat classification performs better than hierarchical classification for well-balanced datasets.

(Weinberger and Chapelle, 2009) introduce a framework for datasets with large taxonomies which models the *latent semantic space* underlying within a hierarchy. In this method, documents are represented as high dimensional sparse vectors using tf-idf scores. Documents are classified after being mapped to the same low dimensional semantic space as class labels using nearest neighbour rule . Furthermore, they use a light weight convex optimization which incorporates large margin constraints to ensure closeness of correct class prototypes. However, this convex optimization problem fails for datasets with substantially large amount of labels (> 1000).

(Bengio et al., 2010) proposed a new way to tackle the amount of growing data and classes in multi class classification since most methods use one-vs-rest classifier which is very slow. They introduce novel methods to learn the label tree structure and thereby generate label embeddings from the data in the absence of a taxonomy. They learn the label tree structure by learning sets of labels at each node and the predict by optimizing the overall misclassification at each node of the tree. The main contribution of this method was the reduced *memory consumption* for generating label embeddings as they were projected onto a lower dimensional space leading to faster computation.

Some categories have different meanings when they lie in different places in the same hierarchy. For example, **Car** $\rightarrow$ *Jaguar* is different from **Cat** $\rightarrow$ *Jaguar*. Even though they both have the same category names, the path to reach these two sematically different nodes from the root is different. (Gao et al., 2009) introduced path semantic vectors (path-based semantic representation) to capture the exact semantic information of categories in a hierarchy. Using this, they mitigate the *error propagation* problem which was found in the earlier works of (D'Alessio et al., 2000; Mladenic, 1998; Sasaki and Kita, 1998; Wang et al., 2001; Weigend et al., 1999) by introducing a prior information based framework to deal with classification errors at different levels of the hierarchy.

(Gopal and Yang, 2013) introduced a model that encoded class interdependency into the classification model by *regularizing* the model parameters of child nodes to the common parent node. Unlike using fully Bayesian models where the dependencies are modelled using means and covariances of Gaussian models (Gopal et al., 2012), their proposed method was much simpler. Due to the simplicity, their model was scalable to larger datasets. Following this work, (Charuvaka and Rangwala, 2015) addressed problems of class imbalance and training efficiency. They improve training efficiency by redefining the *regularization* approach to one where they minimize the misclassification between classes with respect to the hierarchy. By doing so, they also countered the effects of imbalanced datasets by giving more importance to *rare categories*.

More recently, (Peng et al., 2018) introduced Deep Graph-CNN which significantly improved classification performance using *graph-of-words* embeddings to represent documents. *Graph-of-words* are gen-

erated by using graph convolution operations to convolve the entire word graph. They achieved state of the art results on two large-scale hierarchical news datasets[1] by benefiting from graph representation of text along with class-interdependency between labels (Gopal and Yang, 2013).

Many have shown that the use of hierarchical taxonomies improves text classification performance. (Sinha et al., 2018) used a global neural attention model for hierarchical classification which uses knowledge from an *external taxonomy* to enable document representations to depend on the parent class predicted by the model. Their model was more robust and also performed better than the state of the art flat classifier (Joulin et al., 2016). Furthermore, (Meng et al., 2018) used weak supervision to generate document representations and designed a hierarchical neural model that mirrored *class taxonomy*. Using these two, they automatically determine the appropriate level and thereby the correct category in the hierarchy for each document.

One of the most important factors that triggered this extensive research in large scale hierarchical classification is Kaggle's ***Large Scale Hierarchical Text Classification*** [2] challenge (or LSHTC in short) which ran for 4 years from 2010−2014 (Partalas et al., 2015). The objective of this challenge was to analyze the performance of different classifiers for large scale datasets which had a large number of classes (up to hundreds of thousands).

The first challenge in (Kosmopoulos et al., 2010) provided a simple multi class classification problem with a *tree* based hierarchy assigned with labels only from the leaf node. The winning solutions in this challenge used hierarchical polynomial SVM[3] and another *flat* approach which disregarded the taxonomy information and used online training techniques (Madani and Huang, 2010).

Multilabelled datasets were introduced the next edition (PKDD2011, 2011) which had both *DAG* and *tree* based hierarchies. The winning solution that year used an enhanced kNN approach based on a BM25 similarity and DSS-cut thresholding to predict multiple labels (Wang et al., 2011a).

The third edition of the challenge evaluated performance of the models using hierarchical metrics along with standard metrics. The winning model of this challenge (Wang et al., 2011b) considered the classification problem as a meta learning problem based on hierarchical top down classification followed by extraction of meta features from the scores of each classifier. The last edition of the challenge was run as a Kaggle competition using a large-scale dataset with undirected graph hierarchy.

[1] Reuters and New York Times

[2] https://www.kaggle.com/c/lshtc

[3] Participants did not provide a description

Chapter 3

Dataset

Owing to the popularity of the LSHTC Kaggle challenge, many large-scale hierarchical datasets were created. The most commonly used public large-scale hierarchical datasets (Partalas et al., 2015) include the RCV1 dataset, Wikipedia (Small and Large) datasets, DMOZ dataset, MeSH[1] dataset etc. These datasets have up to hundreds and thousands of samples which can be used for solving LSHC. The nature of hierarchy is also different for each dataset. The hierarchy in RCV1 follows an *undirected graph*, Small Wikipedia follows a *DAG* while the Large Wikipedia follows an *undirected graph*. DMOZ has a *tree-based* hierarchy and taxonomies in MeSH are defined using a *directed graph*. Apart from these public datasets, there exists proprietary large-scale datasets that can also be used for LSHC. One such dataset is the **OmniScience** dataset - an inhouse dataset of Elsevier which is used to perform our experiments.

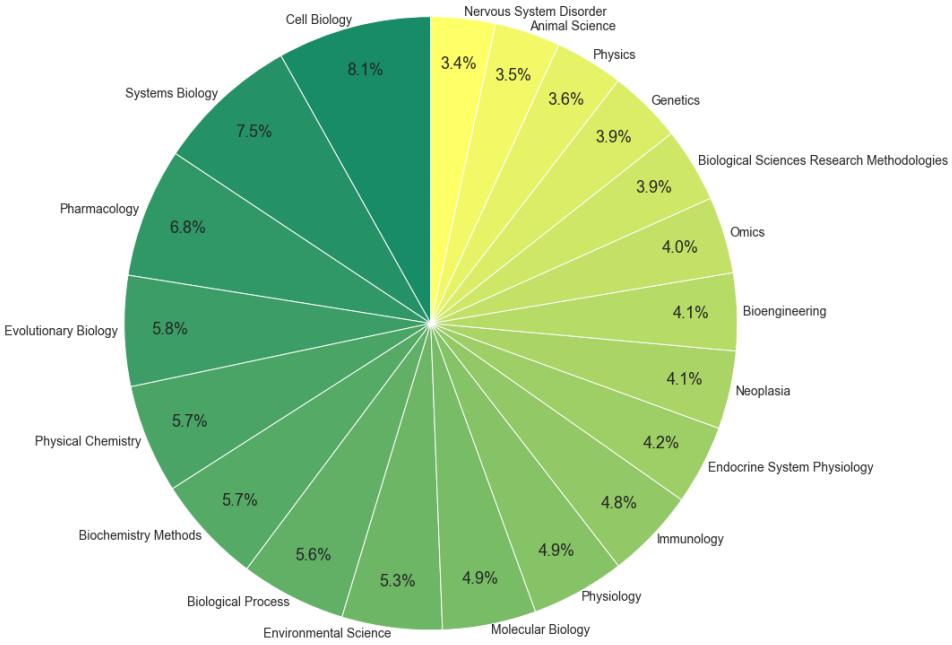


Figure 3.1: *OmniScience Dataset: Top 20 classes distribution*

The **OmniScience** taxonomy is a global vocabulary of topics in science maintained by Elsevier. It was created to bridge the vocabularies within Elsevier owned classification systems such as ScienceDirect, EES/EVISE and between the inhouse vocabulary and international standards. Omniscience has a broad structure inspired from WordNet domains with data represented in SKOS format. It is mainly used for downstream tasks like automatic document classification, user recommendation, content analytics etc.

The OmniScience dataset [2] has abstracts and titles of documents in English from ArXiv (78.78%), BMED (12.75%) and EVISE (8.45%) manually tagged with labels (label text as well label id) from the

[1] https://www.nlm.nih.gov/databases/download/mesh.html
[2] OmniScience version 01.19.04

OmniScience taxonomy (Figure 1.1). BMED has articles belonging to the medical domain whereas ArXiv and EVISE have articles which belong to technology, science, mathematics etc. Figure 3.1 gives an insight of the class distribution of the top 20 labels in the dataset.

3.1 Text Preprocessing

We consider the OmniScience abstract text as the documents. However these documents were unclean. Hence, the dataset was subjected to a list of standard preprocessing steps which include:

- Transforming text to lowercase
- Removal of punctuation and numerals
- Stopword removal
- Stemming

Since the nature of the topics covered by OmniScience is not general and rather more scientific, we used a corpus specific stopword removal scheme to remove functional words. Moreover, the dataset contained a lot of inconsistent instances. Therefore, we removed instances which had the following characteristics:

1. Empty abstracts
2. Duplicates i.e., instances with same abstract and labels.
3. Instances with uninformative abstracts: These instances had abstracts that just contained information like: "acknowledgement", "reference", "publisher's note", "graphical abstract", "paper withdrawn".
4. Instances with short repetitive abstracts: Along with the criterion described above, there were some abstracts which had really short abstracts, i.e. less than 5 words that was not unique to just one instance. Examples of such instances are:

   ```
   this paper is withdrawn
   short philosophical essay
   the paper is obsolete
   graphical abstract unlabelled image
   ```

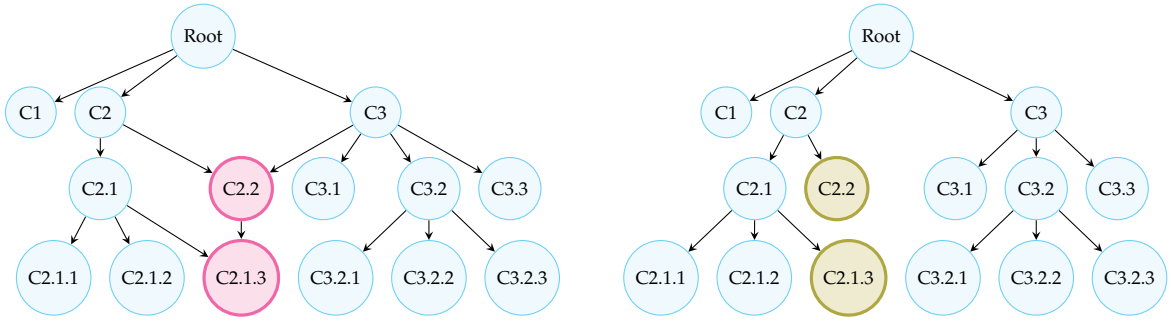
Abstract	Label	Parent Label	Path
Consider laying out a fixed-topology tree ... optimal number of memory transfers is ...	Algorithms	Metabolic Pathway Analysis	/Science/Natural Sciences/Biological Sciences/ Omics/Metabolomics/Metabolic Pathway Analysis/Algorithms/
			/Science/Applied Sciences/Engineering/Bioengineering/ Metabolic Engineering/Metabolic Pathway Analysis/Algorithms/
			/Science/Natural Sciences/Biological Sciences/ Bioengineering/Metabolic Engineering/ Metabolic Pathway Analysis/Algorithms/

Table 3.1: *Example of an instance having miscellaneous paths to reach* ***Label*** *from root* ***Science****. We disregard this path information while preprocessing data.*

Due to scientific nature of the dataset, we created a corpus specific stopwords for OmniScience. These stopwords included the standard stopwords in English (Bird et al., 2009) and subsampling of frequent words (such as - `aim`, `analyze`, `drug`, `theories`, `variable`, `volume`) in the corpus (McCormick, 2011). Moreover, many instances have the same abstract but their labels are associated with a different path to root in the hierarchy. Table 3.1 illustrates an example. We run experiments disregarding these miscellaneous path information. Table 3.2 lists the overall dataset statistics after performing the preprocessing steps mentioned above.

Dataset	#Training	#Testing	#Class-labels	#Leaf-labels	Depth	#Features	Avg #labels per instance
OmniScience	484,129	99,804	55,258	44,353	19	300	3.26

Table 3.2: *Dataset Statistics*



(a) *DAG (Directed Acyclic Graph)* ***(b)*** *Tree*

Figure 3.2: *Example of DAG to Tree conversion using Algorithm 1. Nodes depicted in red represent nodes having more than one parent which will be 'unlinked' using the respective algorithm to give nodes with only one parent (green nodes) which is representative of a tree.*

3.2 Conversion of Directed Acyclic Graph to Tree

Since we focus on only tree based hierarchies, it is essential to convert graphs ($\mathcal{G}$: directed acyclic graph) to trees ($\mathcal{H}$). The resulting tree $\mathcal{H}$ is a special case of (un)directed graph, where any two nodes are connected by exactly one path as opposed to any two nodes having more than one path ($\mathcal{G}$).

Any undirected graph can be converted to its optimal equivalent tree representation using *minimum spanning trees* (Kruskal, 1956; Prim, 1957). The equivalent for directed graphs can be achieved using *minimum spanning arborescence*.

An *arborescence* is defined as a 'rooted digraph' (directed graph) where there exists only one unique path between the root and any other vertex (Gordon and McMahon, 1989). It is also called as 'directed rooted tree' or 'out-tree'. Additionally, it is also possible to reverse *all* the direction of the arcs of an arborescence and make it point towards the root rather than away from it. This results in *anti-arborescence* or 'in-tree' representation of the digraph (Korte et al., 2012). Thereby, an essential condition while converting DAGs to trees using arborescence is that, *all* nodes should be either pointed towards (inward) or away (outward) from the root (single root).

Algorithm 1 Conversion of DAG to Trees

Input: $\mathcal{G}$, $\mathfrak{r}$ (root)
Return: $\mathcal{H}$
initialize new_edges, queue, visited $\leftarrow$ Counter()
push queue $\leftarrow \mathfrak{r}$
update visited[$\mathfrak{r}$]+=1
while queue **do**
  pop $s \leftarrow$ queue[0]
  for $i \in \mathcal{G}$.neighbours(s) **do**
    push queue $\leftarrow i$
    update visited$[i]+=1$
    if visited$[i]! = 1$ **then**
      pass
    else
      add new_edges $\leftarrow (s, i)$
    end if
  end for
end while
create $\mathcal{H}$ from new_edges

We use *arborescence* to convert DAG to trees if it is possible otherwise we devise an alternate algorithm (Algorithm 1) for conversion to trees which considers both inward and outward edges of a digraph (Figure 3.2a).

Algorithm 1 converts DAG to trees if *minimum spanning arborescence* fails (Figure 3.2). It performs a breadth-first-search at every node, starting from the root and ensures that every node is visited only once. It creates a new (di)graph by appending only unvisited edges of a node. Although this brute force conversion results in severe loss of hierarchical information, it is ignored since we want to emphasize on the hierarchical structure.

An analysis of the **OmniScience** taxonomy showed that it contained nodes that possessed both inward and outward edges. Therefore, it did not satisfy the condition for *arborescence* and the hierarchy was converted to tree (Figure 3.2b) using Algorithm 1.

Chapter 4

Evaluation Metrics

The most common metric used for calculating the efficiency of a classifier is accuracy. Accuracy is defined as the ratio of correct to total predictions, and is widely used as the metric for evaluating the performance of a classifier. However, accuracy is unreliable if the dataset is imbalanced since the classifier can achieve high accuracy by predicting all examples as the majority class. Real-world large-scale datasets are often imbalanced and hence accuracy cannot be used. However, other measures such as precision and recall can be used since they account for class imbalance.

4.1 Standard Metrics

Standard set-based measures (Yang, 1999) such as Micro-F1 and Macro-F1 are generally used for evaluating the performance of a classifier under imbalanced datasets.

Micro-F1 is defined as the harmonic mean of class-specific Precision (P) and class-specific Recall (R). Let TP_c, FP_c, FN_c denote the true-positives, false-positives and false-negatives for class $c \in \mathcal{N}$, where $\mathcal{N}$ is set of all classes. Precision identifies the proportion at which the model was correct (TP_c) when identifying the positive classes ($TP_c + FP_c$). Recall measures the proportion of positive classes (TP_c) from all classes that were possibly correct ($TP_c + FN_c$). Thus, micro-averaged F1 is can be defined as:

$$P = \frac{\sum_{c \in \mathcal{N}} TP_c}{\sum_{c \in \mathcal{N}} (TP_c + FP_c)}$$
$$R = \frac{\sum_{c \in \mathcal{N}} TP_c}{\sum_{c \in \mathcal{N}} (TP_c + FN_c)}$$
$$Micro - F1 = \frac{2PR}{P + R}$$

Unlike Micro-F1, Macro-F1 on the other hand, gives equal weightage to all classes such that the average score is not skewed in favour of majority classes. It is defined as:

$$P_c = \frac{TP_c}{TP_c + FP_c}$$
$$R_c = \frac{TP_c}{TP_c + FN_c}$$
$$Macro - F1 = \frac{1}{|\mathcal{N}|} \sum_{c \in \mathcal{N}} \frac{2P_c R_c}{P_c + R_c}$$

4.2 Hierarchical Metrics

These standard metrics, although useful, suffer from a major drawback. Standard metrics penalize all misclassified examples equally while evaluating the performance of the classifier. Intuitively, the degree of misclassification is an important factor for hierarchical classification since the penalty of misclassifying into a neighbourhood class in less than misclassifying to farther classes.

(a) *Small misclassification penalty* ***(b)*** *Large misclassification penalty*

Figure 4.1: *Hierarchical distance to mitigation misclassification penalty. The nodes in green represent the true class whereas the nodes in red represent the predicted class*

Due to the drawbacks mentioned above, we consider hierarchical measures that consider the hierarchical distance between true and predicted label for evaluating classifier performance (Kosmopoulos et al., 2015). Figure 4.1 depicts an example of penalty for misclassification. For instance, **C1.2** misclassified as **C1.1** is less severe that being misclassified as **C2.2** since **C1.1** and **C1.2** belong to a common parent **C1**, whereas **C2.2** is a child of **C2** which belongs to a completely different branch of **Root**.

The hierarchical metrics include hierarchical F1 ($hF1$), hierarchical precision (hP) and hierarchical recall (hR) (Kiritchenko et al., 2005; Struyf et al., 2005).

Let $\hat{Y} = \{\hat{y}_i | i = 1 \dots M\}$ and $Y = \{y_j | j = 1 \dots N\}$ be the sets of predicted and true labels respectively for a single test instance (index of instance is omitted for simplicity). Then, $\hat{Y}$ and Y are augmented with all ancestors (An) except root of the predicted and true labels as follows:

$$\hat{Y}_{aug} = \hat{Y} \cup An(\hat{y}_1) \cup \dots \cup An(\hat{y}_M) \tag{4.1}$$

$$Y_{aug} = Y \cup An(y_1) \cup \dots \cup An(y_N) \tag{4.2}$$

Thus hP, hR and $hF1$ can be defined respectively as:

$$hP = \frac{\left|\hat{Y}_{aug} \cap Y_{aug}\right|}{\left|\hat{Y}_{aug}\right|} \tag{4.3}$$

$$hR = \frac{\left|\hat{Y}_{aug} \cap Y_{aug}\right|}{|Y_{aug}|} \tag{4.4}$$

$$hF1 = \frac{2 \times hP \times hR}{hP + hR} \tag{4.5}$$

Consider Figure 4.2 which explains hierarchical metric. From Eq. 4.2 and 4.1 we get the augmented true label set as { T2, 1, T1, 3, **o**} and augmented predicted label set as { P1, 4, **o** }. Using the definitions in Eq. 4.3 - 4.5 we get $hP = \frac{1}{3}$, $hR = \frac{1}{5}$ and $hF1 = \frac{1}{4}$.

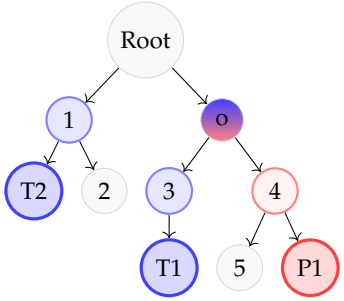


Figure 4.2: *Example of Hierarchical Metric. Node P1 is predicted label. Nodes T1 and T2 are true labels.*

Chapter 5

Parent - Child Regularization

The *recursive regularization* framework (Gopal and Yang, 2013) is an approach that leverages hierarchical dependencies as well as graph-based dependencies among class labels for large scale classification. It is applicable to both large-margin classifiers (SVM) as well as probabilistic classifiers (logistic regression). We experiment using the probabilistic classifier and also focus only on the hierarchical dependencies between class labels.

In general, the dependencies between parent labels in a hierarchy and its children are used to define a joint objective for regularization of model parameters. This is based on the assumption that neighbouring classes in a hierarchy are semantically close to each other and hence share similar model parameters. The objective is formulated in such a way that the parameters at each node are only dependent on itself and its parent and not the rest of the hierarchy; therefore by fixing the parameters of the parent and children, the node can be optimized locally within that neighbourhood.

Joint objective optimization of parent and children dependencies aids in achieving a large degree of parallelization and gets a speedup close to linear in the number of processors used. Due to the simplicity of this approach, it is easier to scale using such approaches rather than fully Bayesian (Shahbaba et al., 2007; Gopal et al., 2012) approaches where the dependencies are modelled in richer forms by controlling both the means and covariences in Gaussian models.

5.1 Problem Definition

Let hierarchy be defined over a set of nodes $\mathcal{N} \in \{1, 2 \ldots\}$ where $\pi(n)$ and $C(n)$ are the parent and child of node $n \in \mathcal{N}$. The respective parent-child relationship is given by $\pi : \mathcal{N} \rightarrow \mathcal{N}$. Let $\mathcal{T} \subset \mathcal{N}$ denote the set of leaves in the hierarchy labelled from 1 to $|\mathcal{T}|$.

Let $\mathcal{D} = \{x_i, t_i\}_{i=1}^{M}$ denote the training dataset of M instances where each $x_i \in \mathbb{R}^d$ and $t_i \in \mathcal{T}$. d refers to the number of features (dimension) in the dataset. It is assumed that each instance is labelled to one or more leaf nodes in the hierarchy. If it is not and the instances are assigned to an internal node, then spawn a surrogate leaf node under it and re-assign all the instances from the internal node to this new surrogate leaf node.

Let y_{in} be a binary indicator function defined for instance i w.r.t nodes n as:

$$y_{in} = \begin{cases} +1, \text{ if } x_{in} \in \mathcal{T} \\ -1, \text{ otherwise} \end{cases}$$

The aim of the prediction function f in this scenario is to learn a mapping from instances ($x_i \in \mathbf{X}$) to target-classes such that it predicts the target class-label for a given instance with minimal error i.e., $f : \mathbf{X} \rightarrow \mathcal{T}$. This function is parameterized in the context of hierarchies by $\mathbf{W} = \{w_n : n \in \mathcal{N}\}$ [1]. Thereby, the model parameters $\mathbf{W}$ are estimated by minimizing,

$$\arg\min_{\mathbf{W}} \lambda(\mathbf{W}) + C \sum_{n \in \mathcal{T}} \sum_{i=1}^{M} L(y_{in}, x_i, w_n) \tag{5.1}$$

[1] $w_n \in \mathbb{R}^d$

where L is the logistic loss function, $\lambda(\mathbf{W})$ is the regularization term and C is a parameter that controls trade-off between the loss and regularizer. The regularizer $\lambda(\mathbf{W})$ can be further elaborated as,

$$\lambda(\mathbf{W}) = \arg\min_{\mathbf{W}} \sum_{n \in \mathcal{N}} \frac{1}{2} \left\| w_n - w_{\pi(n)} \right\|_2^2 \tag{5.2}$$

The above equation incorporates a recursive structure into the regularizer term by modelling hierarchical dependencies which enforces the parameters of node n to be similar to its parent under euclidean norm. Not only does this helps nodes to leverage information from its neighbours in the hierarchy while estimating $\mathbf{W}$, but it also helps share statistical strength across the hierarchy.

The primary optimization problem in RR-LR is to estimate $\mathbf{W}$. Solving for non-leaf nodes becomes trivial since they only serve as virtual labels in the hierarchy due to the initial assumption. The closed form update of w_n for non-leaf node $n \notin \mathcal{T}$ is given by,

$$w_n = \frac{1}{|C_n| + 1} \left(w_{\pi(n)} + \sum_{c \in C_n} w_c \right) \tag{5.3}$$

For each leaf node n, isolating the terms that depend on w_n, the objective (f_{w_n}) and corresponding gradient G_{w_n} is given by,

$$f_{w_n} = \frac{1}{2} \left\| w_n - w_{\pi(n)} \right\|_2^2 + C \sum_{i=1}^{M} \log(1 + \exp(-y_{in} w_n^\top x_i)) \tag{5.4}$$

$$G_{w_n} = w_n - w_{\pi(n)} - C \sum_{i=1}^{M} \frac{y_{in} x_i}{1 + \exp(y_{in} w_n^\top x_i)} \tag{5.5}$$

The above objective function for RR-LR is convex and differentiable. Hence second order methods such as exact netwon's methods or quasi-newton methods are applicable. LBFGS (Liu and Nocedal, 1989), a limited variant of quasi-newton method was used to solve the optimization problem since it is not feasible to store inverse of the Hessian for large-scale problems in the memory. The pseudo-code for optimization of $\mathbf{W}$ is given in Algorithm 2.

Algorithm 2 Optimization for RR-LR

```
Input: 𝒟, C, π, 𝒯, 𝒩
Return: Weight vectors W*
while Not converged do
    for n ∈ 𝒩 do
        if n ∉ 𝒯 then
            Update w_n using Eq. 5.3
        else
            Solve Eq. 5.4 using LBFGS
        end if
    end for
end while
```

The main advantage of this method over other models such as (Bengio et al., 2010; Zhou et al., 2011) is the absence of a constraint that minimized the error margin of predictions. This reduces dependencies between parameters and allows a parallel-iterative method to optimize the objective (5.3 & 5.4) thereby scaling to large hierarchical classification problems.

5.2 Text Representation

Most of the crucial challenges in large scale hierarchical classification include identifying important features in real world datasets to improve classification performance and runtime performance followed by efficient memory usage to store learned models. Traditional methods perform hierarchical classification with sparse representation of the datasets generated by normalized tf-idf weighting.

TF-IDF (Term Frequency Inverse Document Frequency) is one of the most common approaches to transform textual information to vector representation. It involves a simple transformation of text based on the frequency of a word in a corpus (*tf*) and the inverse frequency of its occurrence in the overall set of documents (*idf*) (Salton and Buckley, 1988). The entire transformation results in a sparse term-document matrix where each row is a vector representation of each document and each column represents the words (terms) in the corpus.

Although tf-idf features are easy to compute and capture the most descriptive terms of a document, it is useful only as a lexical level feature. It does not capture semantics or co-occurrences in different documents or position in text which is possible distributed representations of text such as word embeddings. Word2Vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014) and FastText (Joulin et al., 2016) are examples of different word embeddings.

Word Embeddings are a mapping of $V \to \mathbb{R}^D : w \mapsto \vec{w}$ i.e., it maps a word w from a vocabulary V to a real valued vector $\vec{w}$ in an embedding space of D dimensionality. Word2Vec was the first model which could train word representations in an unsupervised manner. It was introduced by (Mikolov et al., 2013) where vector representations for each word is learnt using a shallow neural network. These vectors are generated by training to maximize the log probability of neighbouring words in a corpus. GloVe (Global word Vectors) (Pennington et al., 2014) vectors are generated by combining global co-occurrence matrix factorization and local context window methods. FastText (Joulin et al., 2016) generates word embeddings like Word2Vec but considers bag of *n*-grams instead of bag of words enabling it to capture information regarding word order.

5.3 Experiment Details

Document representation

For each dataset we follow the pre-process using the steps mentioned in Chapter 3. All experiments are run in Python 3.6 on a CPU with 12 cores and 96GB RAM [2].

TF-IDF vectors are generated using the top 300 features and ignoring terms that have a document frequency greater than 70% of the corpus size [3].

Word embeddings are generated using the FastText library[4] in unsupervised mode for each dataset. These embeddings have a dimensionality of 300 and are generated after training for 5 epochs using CBoW with window size = 3 (neighbourhood size) and no negative sampling or hierarchical softmax. From these embeddings, document representations ($\vec{u}$) are generated by adding the word vectors (w) of each word in each document and dividing it with the euclidean norm of the summed word vectors i.e.,

$$\vec{u} = \frac{\sum_{i \in \text{words}} w_i}{\left\| \sum_{i \in \text{words}} w_i \right\|_2}$$

We use the above implementation instead of simply averaging the word vectors due to reasons mentioned by (Kenter et al., 2016; Wu et al., 2017), even though FastText argues to form good sentence representations by simply averaging (Joulin et al., 2016).

Hierarchy Assumption

The premise of this model is that every document in each dataset is tagged with atleast one leaf node from the hierarchy. However, the **OmniScience** dataset (Table 3.2) had only 1023 out of 44,353 ($\approx$ 2.3%) of its leaf nodes tagged to documents. Therefore, we selected a subgraph of the entire taxonomy which had these 1023 leaf nodes and converted this subgraph to a tree using Algorithm 1. Table 5.1 lists the modified statistics after converting hierarchies to trees and introducing surrogate leaf nodes.

Since these are multilabeled datasets we need to select a thresholding strategy to perform classification. A trivial strategy would be to threshold at 0 or 0.5. But these do not give optimal results when handling *large* amount of multilabel data. Instead, we optimize a threshold θ_n for each class n using a validation set, generated by stratified k-fold cross validation, such that class n is assigned to test instance x_i if $w_n^\top x_i > \theta_n$. This is called *SCut thresholding* strategy (Yang, 2001). It tunes the threshold

[2] Databricks

[3] `max_features` and `max_df` in sklearn TfIdfVectorizer

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

Dataset	#Training	#Testing	#Class-labels	#Leaf-labels	Depth	#Features	Avg #labels per instance
OmniScience Subgraph	484,129	99,804	1148	1010	6	300	2.18

Table 5.1: *Dataset statistics after modifying. Overall number of leaf nodes in OmniScience taxonomy decreased to 1010 from 1023 after modification because the new subgraph had fewer leaves i.e., 886 which increased to 1010 after augmenting the hierarchy with 124 surrogate leaf nodes.*

independently of all other classes and is inexpensive when compared to other strategies used in (Gopal and Yang, 2010).

We run a series of 4 experiments on the dataset to test the hierarchical classification performance of sparse and dense text representations. We run two experiments using flat (**FLAT-LR**) and hierarchical (**RR-LR**) methods to see if introducing inter-dependency amongst nodes, each for sparse text features (**TF-IDF**) and dense features (**FastText**), improves classification performance. Results and analysis of these experiments are given in Chapter 7.1.

Chapter 6

Learning Hierarchical Structures

In this chapter we use neural embedding model to learn hierarchical structures explicitly and see if that improves classification performance.

6.1 StarSpace: Embed All Things!

StarSpace (Wu et al., 2017) is a general purpose neural embedding model that can be used to solve a wide variety of problems such as - labelling tasks for document classification, ranking tasks for information retrieval/web search, learning embeddings for documents, sentences, words etc. It creates embeddings for features in each of these tasks which lie in the same vector space. These embeddings are learnt by minimizing a pair of entities (embeddings) on a metric (e.g., similarity) of choice. Entity pairs are defined on the basis of the task. In a document classification task, document-label is considered an entity pair.

The model starts by assigning a d-dimensional vector to each of the discrete features that we want to embed directly (referred to as dictionary - it can contain features like words, etc). Entities such as documents are comprised of these features (bag of words) from the dictionary and their embeddings are initialized from a normal distribution in the beginning upon which they are learnt implicitly.

Consider a dictionary of D features as F which is a $D \times d$ matrix, where F_i indexes the i^{th} feature (row) and yields its d-dimensional embedding. Thereby an entity u (e.g., document) can be embedded as $\sum_{i \in u} F_i$.

Models are trained by comparing pairs of entities (document(u) - label(b)) and this is done by minimizing the following loss function:

$$\sum_{\substack{(u,b) \in E^+ \\ b^- \in E^-}} L^{batch}(sim(u, b^+), sim(u, b_1^-), \ldots, sim(u, b_k^-)) \tag{6.1}$$

E^+ generates positive entity pair (u, b^+) i.e. it generates a document (u) and its correct label (b^+).

E^- samples negative entities for b_i^- using k-negative sampling (Mikolov et al., 2013) to select k negative pairs during each batch update. In document classification negative entities are sampled from the set of labels.

Based on the task and optimal results, similarity of these embeddings is computed using either cosine similarity or inner product. (Wu et al., 2017) suggest that due to fewer number of label features required for computing relational embeddings, inner product performed better whereas cosine similarity worked better for generating document embeddings.

L^{batch} compares positive entity pairs with negative entity pairs using either margin ranking loss $\max(0, \mu - sim(u, b))$ where μ is the margin parameter or negative log loss of softmax. Again, the choice of function depends on the task and for document classification, margin ranking loss performed better than negative log of softmax [1].

The loss is optimized using stochastic gradient descent (SGD). In each SGD step, one sample is taken from the set of positive generator (E^+) in the outer sum using Adagrad (Duchi et al., 2010) and Hogwild! (Recht et al., 2011) over multiple CPUs. Moreover, the embeddings are subjected to a max norm to ensure that the learnt vectors lie in a ball of radius r in space R^d (Weston et al., 2011).

[1] This was suggested by (Wu et al., 2017)

At test time, the learnt similarity function is used to measure similarity between entities. In document classification, a label is predicted for a given document u using $\max_{\hat{b}} sim(u, \hat{b})$ over a set of possible labels $\hat{b}$.

With such a general objective, StarSpace can be used to learn embeddings for a wide range of tasks. StarSpace defined different training modes which required inputs in different format based on the task. The subsequent section elaborates on the different training modes.

6.1.1 Task Dependent Training Modes

The aim of StarSpace is to learn entity embeddings such that it can generalize over diverse tasks. Due to this diverse nature the authors (Wu et al., 2017) devised a set of pre-defined training modes dependent on the nature of the task.

Input files in StarSpace are given in the following format. Each line of the file is one instance, that is an input of that instance has `k` words paired with labels from `1..r`. Labels are distinguished from input text by assigning a prefix such as `__label__`. The most commonly used file format is the same as FastText (Joulin et al., 2016).

`__label__1` ... `__label__r word_1 word_2` ... `word_k`

Entity pairs are defined on the basis of the training mode. In the following paragraphs, these entity pairs are defined as LHS (label) and RHS (input).

trainMode 0 - Classification task

This mode is often used for classification tasks. For example, TagSpace (Weston et al., 2014) can be trained using this mode by assigning the label prefix as '#'. Each instance contains input and labels. It is defined the same way as above. Here, the feature dictionary F is composed of all words in the input file. Entity pairs are label (LHS) and document (bag-of-words, RHS) in this mode. For single label classification each input has one label whereas for multi-label classification each input has multiple labels and the algorithm selects one label as LHS during training.

trainMode 1 - Content Based Recommendation

This mode is often used for recommendation tasks. For instance, new movies can be recommended to users based on the movies watched in the past by the user. Here, movies will be used as features for generating embeddings (F). Users are represented as a 'bag-of-movies' based on the movies they have watched i.e.,

$$\text{user} = \texttt{avg}(\text{embedding of movies watched in the past})$$

This system works better when number of users is greater than number of movies watched & the number of movies watched by a user is small on average. Format of input file in this setting is defined as follows (label prefix is movie):

`movie_1, movie_2, ..., movie_M`

During training, at each step, for each instance (user), one random movie is selected as label (LHS) and the remaining bag-of-movies are selected as input (RHS).

trainMode 2 - Information Retrieval

This setting is used to learn a mapping from an object to a set of objects which it belongs to. For example, it can be used to learn the mapping between sentences and articles. Given the embedding of one sentence (query), one can find which document that sentence originally belonged to. Each instance is an article which contains multiple sentences. At training, one sentence is picked at random as the input (LHS) and the remaining sentences in the article become the label (RHS) while other articles are sampled as random negatives. Instantiations of words from all the documents are defined as unique features in the dictionary F. Input file format is as follows:

`sentence1 <tab> sentence2 <tab> ... <tab> sentenceN`

trainMode 3 - Learn Sentence Similarity

This option is used to learn pairwise similarity from collections of similar objects. An example would be learning sentence similarity from a set of documents. Given a training set of unlabeled documents, each consisting of sentences, two sentences are selected at random from the same document; one sentence is considered as the label (LHS) while the other is considered as the input (RHS). Other labels (sentences) are drawn from other documents as negative samples. This ensures that semantic similarity between sentences is shared within a document (Wu et al., 2017). Dictionary F and file input is of the same format as trainMode 2.

trainMode 4 - Learning from Multi-Relational Knowledge Graphs

Apart from comparing embeddings to classify documents or learning similarity, Starspace can also be extended to learn embeddings for multi-relational graphs, similar to learning embeddings for relations described in TransE (Bordes et al., 2013). Relations in such graphs are defined by triplets (h, r, t) consisting of a head concept h, tail concept t and relation r.

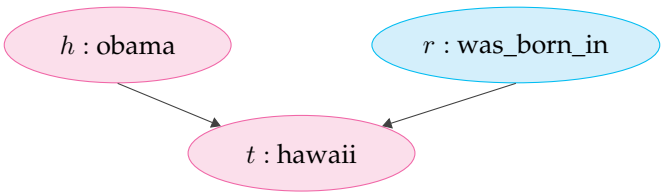


Figure 6.1: *Relations in a graph*

All entities (h, t) and relations (r) are initialized as unique features in the dictionary F. In general, trainMode 4 is used to learn mappings between entities and relations. Therefore, performing node prediction can be formalized as completing missing entities in triplets as:

`(head_entity, relation_type, ?) or (?, relation_type, tail_entity)`

Either (h r - t) or (h - r t) are selected in random uniformly as the LHS-RHS pair respectively while training. Learning embeddings of these entities can thereby be used to learn the respective graph. Input file format in this mode is as follows:

```
relation __label__head __label__tail
```

trainMode 5 - Learn Word Embeddings

StarSpace can also be used to learn unsupervised word embeddings using training data consisting of raw text. These words are initialized as unique features in the dictionary F. Following the principles of word2vec (Mikolov et al., 2013) and fastText (Joulin et al., 2016), the neighbouring words around a word is considered as LHS while the word itself is considered as RHS during training.

6.1.2 Multi Task Learning

Due to the diverse nature, different tasks can be combined and trained at the same time if they share the same base dictionary F. For example, generating unsupervised word embeddings which can be used passed on to another StarSpace model such as TagSpace, a supervised classification task, to achieve semi-supervised learning (Wijffels, 2019).

6.1.3 Parameters

One of the beneficial things about StarSpace is the ability to tune a wide range of hyperparameters. StarSpace offers a wide range of hyperparamaters to tune such as learning rate, training with AdaGrad, dropout for LHS and RHS entities, controlling the negative sampling amount, alternating between hinge loss and softmax loss, loading a pretrained model, adjusting batch size etc.

6.2 Experiment details

A major problem in hierarchical classification is the data sparsity for majority of class labels. Majority of data belong to only a subset of the classes and this results (Liu et al., 2005; Yang et al., 2003) in heavily biased predictions. One way to mitigate this problem is by enabling the model to learn hierarchy. We run experiments to see if inclusion of hierarchy improves classification performance.

The text dataset is subjected to standard pre-processing steps mentioned in Chapter 3. Additionally, it is also converted to the format required by StarSpace dependent on the training mode. After this, a fixed dictionary (F) was initialized randomly from the vocabulary to ensure uniformity across tasks. We run experiments using the modified subgraph of OmniScience. All experiments are run on a CPU with 64GB RAM and use Python 3.6. Results and details of these experiments are given in Chapter 7.2.

Chapter 7

Results & Analysis

In this chapter we present results for two models. The first model enlists experiment results and observations from *Parent-Child regularization* in Chapter 5. The second model depicts results from experiments using *StarSpace* models in Chapter 6.

7.1 Parent-Child Regularization

We run two experiments using flat (**FLAT-LR**) and hierarchical (**RR-LR**) methods to see if introducing inter-dependency amongst nodes, each for sparse (**TF-IDF**) and dense text features (**FastText**), improves classification performance. Table 7.1 depicts the results of the experimental runs. From these results the following conclusions can be drawn:

Dataset	Metric	Sparse Features		Dense Features	
		FLAT-LR	**RR-LR**	**FLAT-LR**	**RR-LR**
OmniScience	*Micro-F1*	19.69	22.43	35.34	39.22
	Macro-F1	5.04	5.67	11.37	12.83
	h-F1	32.55	32.65	40.08	40.3

Table 7.1: *Results of Recursive Regularization*

1. Overall, dense representation of text gives better classification performance when compared to sparse representations from TF-IDF weight vectors. It performs almost twice as better as TF-IDF weight vectors (on standard metrics).

2. For both formats of text representations, the classifier performs better when trained with an hierarchical objective (RR-LR > FLAT-LR). The higher *Micro-F1* scores of RR-LR compared to FLAT-LR suggest that hierarchical models predicted more correct classes than flat models.

3. Additionally, due to the hierarchical models having higher *Macro-F1* scores compared to the flat model, they capture more rare categories than flat models. This is because *Macro-F1* gives equal importance to all classes unlike *Micro-F1* which favours the majority class. Hence, higher *Macro-F1* scores correspond to capturing more rare categories.

4. Dense representation also captures class information in *general* better than sparse representation. This is evident from the vast difference in *Macro-F1* scores between the two.

5. Due to the higher *h-F1* score, it can be inferred that dense representations capture dependency between parent-child nodes of a hierarchy better than sparse representation. This is because, higher *h-F1* means the classifier is able to predict more classes correctly and thereby has more overlapping *ancestors* between the true and predicted labels. More overlapping ancestors in the hierarchy imply low misclassification penalty. Hence it can be inferred that dense representations capture inter-dependency between nodes better than sparse representations.

Example Prediction

We present a sample result from a multilabel test cases in Table 7.2. It can be seen that the sparse representation methods are not able to predict multiple labels as effectively as dense representation methods.

Document id	True Label	Parent Label	Child Label	FLAT-LR-TFIDF	RR-LR-TFIDF	FLAT-LR-FASTTEXT	RR-LR-FASTTEXT
BMED.PUI:52134430	**252578107** **250232503**	248811479 253230491	-	253033933, 249133392 253230925, **250232503**	253033933, 249133392 253230925, **250232503** 253230766	**250232503**, 253033933 **252578107**,210628467, 249133392,248811484, 249672056,253230766, 209895092,189721148,	**250232503**, 253033933 **252578107**,189721083, 210628467,249133392, 248811484,249672056, 189721148,253230766, 209895092

Table 7.2: *Example of text prediction using different text representation and different model for sample test instance. Bold face text means that the predicted label is correct.*

Table 7.3 displays the amount of time taken to train these models. It can be seen dense features take more time to train than sparse features which is expected from such features. Moreover, the recursive model takes longer time to train than flat model since it has to build classifiers for non-leaf nodes as well.

Model	Time to train (hrs)
Sparse Flat-LR	4.3
Sparse RR-LR	4.8
Dense Flat-LR	8
Dense RR-LR	10.5

Table 7.3: *Training time (hrs) for all models.*

Thus, from this study we can conclude that dense features are a more suitable textual representation format for dealing with hierarchical classification.

7.2 StarSpace

We have already seen in the previous section that using dense text representations improves classification performance. However, one drawback with the recursive regularization model is the amount of time it takes to train. Although, training can be made faster by parallelizing it still takes a lot of time (from 4hrs - 10 hrs). Also, it requires training one classifier per node along with a mandatory leaf node for each instance and scalability was a severe issue in the previous model. One way to mitigate problems pertaining to scalability and mandatory leaf node requirement is to see and capture whether there is any similarity with the hierarchy/taxonomy and the documents associated with it.

Therefore, we run experiments to see if general neural embedding models (such as Starspace) can learn hierarchical relationships of classes such as 'parent-child' and 'sibling' relationships. Once these relationships are mapped to the vector space we also want to check if they can be used to improve classification performance by enabling the documents associated with these relations to lie close to their respective hierarchies in the vector space. Hence, we run experiments to see if there is any performance difference in classification when the model is trained with and without inclusion of hierarchy.

7.2.1 Learning Hierarchy

The objective of this experiment is to see StarSpace can learn and model hierarchical relations of a taxonomy. More precisely, we see if child nodes lie closer to their parent or not. Based on this, the input to Starspace (trainmode 4) is formatted in the following way: `'child parent'` i.e. child node is the LHS entity while the parent node is the RHS entity.

Once the model is trained on this objective we test it by giving some child labels as input and to check if it predicts the parent node. Not only do we check if it predicts the parent node, but also the position of the parent node when the top 10 queries are predicted. The intuition behind this is that if hierarchy was learnt, then ideally the parent node should be in the first position. Table 7.4 displays results of this analysis.

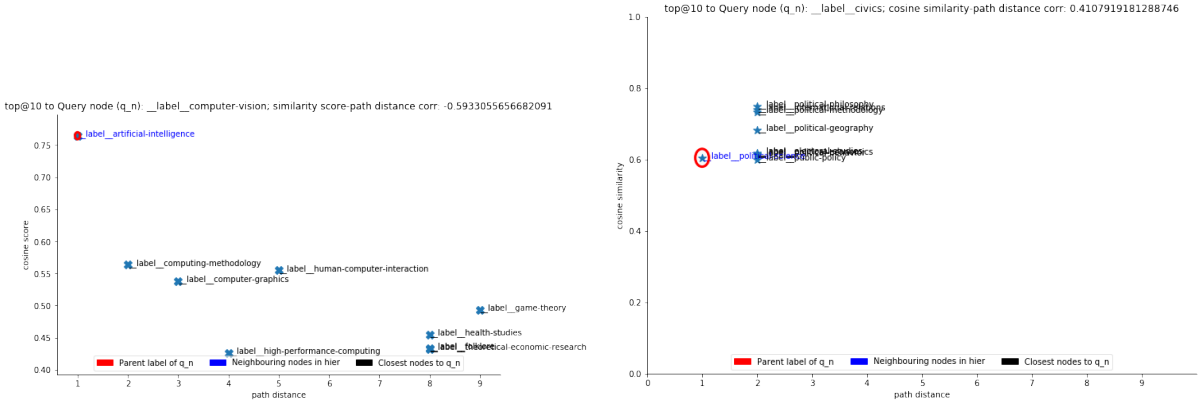
Level in tree	Number of nodes per level	Branching Factor per level	Mean position of parent node in top@10 queries		
			dim = 300	dim = 400	dim = 500
1	45	0.156	1.49	1.71	0.96
2	7	**7.143**	**2.43**	**1.57**	**2.86**
3	50	**9.06**	**2.52**	**2.58**	**2.48**
4	453	0.86	1.92	2.10	1.92
5	391	0.19	1.79	1.72	1.39
6	77	0	1.53	1.86	1.44

Table 7.4: *Analysis of mean position of parent node when queried with child node (from each level) when model is subjected to learn only the hierarchy. From this it can be seen that nodes having a high branch factor inversely affects the position of parent node (bold faced results).*

From Table 7.4 it can be seen that dimensionality of embeddings and branching factor affect the position of parent node retrieval. Branching factor per node is basically the average of the number of outgoing edges of a node. When this is averaged over all the nodes over a level of the tree we get the *branching factor per level*. Additionally, we can also see that nodes which have a high branching factor (level 2 and 3), directly affects the position of the parent node while querying the results. This is because if a parent node has many branches, then the respective parent node embedding will have to compared with *many* other *children* nodes. And in hierarchies with many branches it becomes quiet difficult to compute vector similarities in the Eucledian space (Nickel and Kiela, 2017; Anonymous, 2019). In order to compensate for this, one possible solution is to increase the dimensionality of the embeddings.

However, it is observed that increasing dimensions does not improve parent node position for nodes with severely high branching factor (level 2, 3). Once again this is a drawback of modelling hierarchical representations in Eucledian space (Nickel and Kiela, 2017). Although, it can be observed that the position of parent node slowly approaches 1 for nodes with low branching factor as we increase the dimensionality.

In order to further assess the performance of these hierarchical embeddings we see if there is any correlation between the similarity score of embeddings and the path distance between the child node and its top 10 predicted nodes. Figure 7.1 and depicts this. We used Spearman's rank correlation co-



(a) *Query node: __label__computer-vision* ***(b)*** *Query node: __label__civics*

Figure 7.1: *Vector Similarity vs. Path Distance Correlation: For both the images y axis represents cosine similarity measure of the query node with the retrieved nodes and the x axis represents the path distance in the graph between the same. Points encircled in red denote the parent node of the queried (child) node. The remaining text in black are the retrieved nodes. The text in blue indicate nodes that are 1-hop away from the query node in the graph. These visualizations are of embeddings with dim=500*

efficient (Zar, 2005) to see if there exists any relationship between similarity score and path distance of child node and its retrieved nodes. From both the images it can be seen that there exists a correlation between cosine score and path distance. A negative correlation is observed in the case of Figure 7.1a. This is because parent node of 'computer-vision' is 'artificial-intelligence' which is validated by its unit path length and high cosine similarity score. The remaining predictions have decreasing similarity scores with 'computer-vision' while having an increasing path length. For instance, 'computing-methodology'

is 2 hops away from 'computer-vision'. This could imply that 'computing-methodology' is either a *sibling* of 'computer-vision' or is its *grandparent* (parent of 'artificial-intelligence'). It is in fact the parent of 'artificial-intelligence' and is hence more similar to 'artificial-intelligence'. Similarly with the next prediction, 'computer-graphics' it is actually the *child* of 'artificial-intelligence' and is hence closer to 'computer-vision'. This trend of nodes lying close in the hierarchy also lie close in the vector space, continues for the remaining predicted nodes and they all can be accounted for with similar reasons.

However, we observe a positive correlation with the next example in Figure 7.1b. The parent node of 'civics' is 'political-science' which has a lower similarity score compared to the other predicted nodes. This is because, in the hierarchy the parent node 'political-science' has *many children* (high branching factor). As a result of this, the similarity of that parent node to one child will decrease since it will have to be similar to rest of the children as well. Adversely, in such cases the child node becomes similar to all the other children of that parent node. Due to this we observe a positive correlation for nodes which have high branching factor as observed in Figure 7.1b.

From these results we can infer that StarSpace can capture hierarchical relationships although it has difficulty in capturing complex hierarchies i.e., nodes with high branching factor in the Eucledian space.

7.2.2 Learning Hierarchy to Classify

Since StarSpace can learn embeddings for hierarchical relations we extend this to classification to see if these learnt relations can improve the classification performance of large-scale hierarchical documents. We compare these results with document classification without learning hierarchy.

In order to see if classification benefits from learning hierarchy We first train the hierarchy in trainMode 4 and then use that model as the initialization dictionary (F) for the document classification task using trainMode 0. Similarly, we classify documents without learning any hierarchy by directly using trainMode 0. Table 7.5 displays results of the classification.

Classification type	Model parameters	Micro-F1	Macro-F1	h-F1
Without Hierarchy	dim=300	26.28	11.09	52.21
	dim=400	26.53	11.06	52.41
	dim=500	26.37	11.23	52.28
With Hierarchy	dim=300	26.70	11.09	52.07
	dim=400	26.76	11.16	52.12
	dim=500	26.63	11.13	52.22

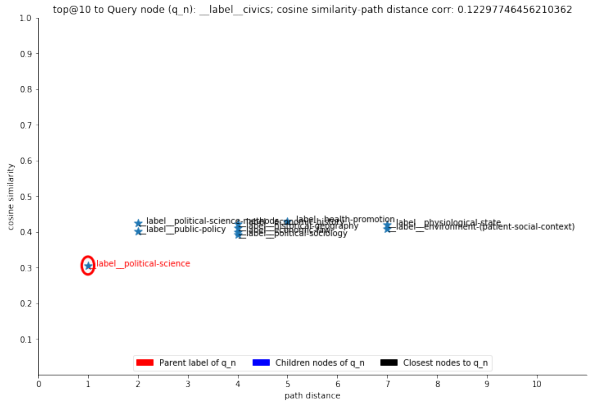
Table 7.5: *StarSpace Text Classification Results*

Looking at the results from the table above, it can be seen that classification performance remains unaffected when classified after learning hierarchy *and* increasing dimensions does not improve classification performance in StarSpace model. However, we are not able to infer why by just looking at these results. Therefore we performed the same analysis as we did in the section before i.e. to learn hierarchical relations. The results of these experiments are given below.

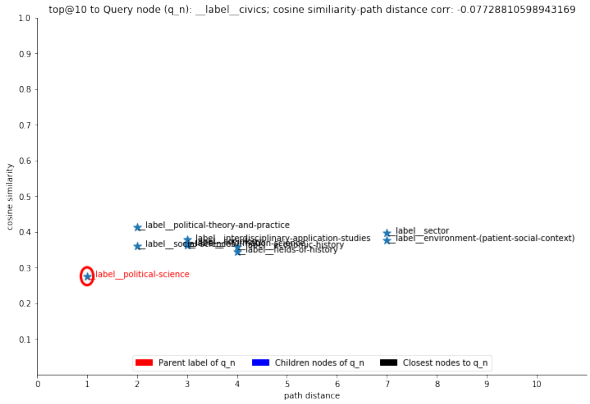
First, we check if there was any correlation between vector similarity and path distance followed by position of parent node when queried with child node. These results are illustrated in Figure 7.2 and Table 7.6.

Looking at the results from Table 7.5, 7.6 and Figure 7.2 it can be inferred that **text classification using StarSpace was not able to benefit from learning hierarchy since it performed almost equally as classifying without learning hierarchy**. This is due to the following reasons -

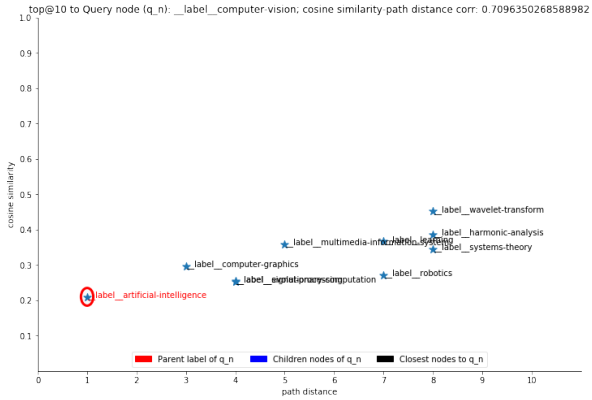
- When checking for correlation between vector similarity and path distance between query node and retrieved node, although the parent node was never retrieved when trained to classify with (Figure 7.2b & 7.2d) and without hierarchy (Figure 7.2a & 7.2c), we still plot the parent to show how similar it is w.r.t its child.
  - Although there is a negative correlation between the two when trained with hierarchy, it is quite low i.e., it is not close to -1. Moreover, the labels which are retrieved and are most similar to the query node (example: 'political theory and practice' and 'social science' are the closest to 'civics') in the vector space, are in fact atleast 2 hops away from the query node in the hierarchy. This means that instead of retaining the learnt the hierarchical relations, the classification objective actually *override* some of them with the text classification objective.



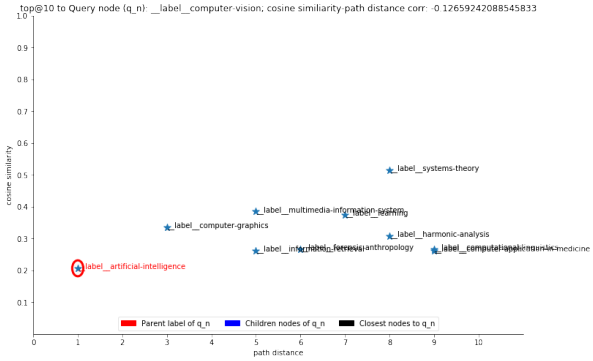
(a) Without Hierarchy - Query node: __label__civics



(b) With Hierarchy - Query node: __label__civics



(c) Without Hierarchy - Query node: __label__computer-vision



(d) With Hierarchy - Query node: __label__computer-vision

Figure 7.2: *Correlation between Vector Similarity and Path Distance while learning to classify with hierarchy:* ***(a)*** *&* ***(c)*** *show document classification embeddings trained without hierarchy and* ***(b)*** *&* ***(d)*** *represent document classification embeddings that have been trained on hierarchy. None of these embeddings are close to their respective parent embeddings in the vector space. Moreover, the vectors which are actually close to them in the embedding space are farther from them in the hierarchy. Dim embedding =500*

Level in tree	Number of nodes per level	Branching Factor per level	Mean position of parent node in top@10 queries with dim = 500	
			With Hierarchy	Without Hierarcy
1	45	0.156	0.22	0.00
2	7	7.143	**0.29**	**0.29**
3	50	9.06	0.68	0.46
4	453	0.86	0.98	0.88
5	391	0.19	**0.49**	**0.49**
6	77	0	0.71	0.55

Table 7.6: *Analysis of mean position of parent node when queried with child node per level, when model is trained to classify document with and without learning hierarchy. Bold face results indicate that there is no change in the retrieval position of parent node in both the cases. Moreover, all the mean position values of the parent node is less than 1.*

- In Figures 7.2c & 7.2a Not only is the parent node not close to the query (child) node, but also all the other retrieved nodes which are closer to the query node in the vector space, are far away from it in the hierarchy. Additionally, this model has more positive correlations than the model trained with hierarchy. This is expected since the vectors have no information regarding hierarchy.
- Even though the examples in Figure 7.2a & 7.2b have correlations close to 0, Figure 7.2b is more negatively correlated than Figure 7.2a. This could be because Figure 7.2b might have retained whatever it learnt initially and is better than Figure 7.2a.

- For Figure 7.2d & 7.2c it is clear that the model was able to retain the hierarchical relations which resulted in the negative correlation for Figure 7.2d while the other Figure 7.2c has positive correlation.

Analysis from Figure 7.2 was able to suggest that the model trained with hierarchy was able to retain some of the relations it learnt initially even after training with the classification object. However, this still does not explain why the results in Table 7.5 are almost the same for all. Upon further analysis with the results from Table 7.6 we can deduce the following:

- All mean positions of the parent node is less than 1. This validates the previous hypothesis that the text classification objective overrides the learnt hierarchical relations.
- The mean positions of parent nodes in levels 2 & 5 are the **same**. This means that for nodes at these levels, training with hierarchy makes no difference at all when compared to training without hierarchy. Even if nodes at this level capture hierarchical relation in the beginning, it is completely overlooked when trained again for classification.
- However, the mean positions of the parent nodes at other levels such as 1, 3, 4 & 6 is greater for model trained with hierarchy than for a model trained without hierarchy. But, it is only greater by **an average factor of 0.12**. A reasonable explanation for models trained without hierarchy to be able to perform on par with models trained with hierarchy is that the dataset might contain similar documents tagged with labels which could be learning the taxonomy structure. For instance, a document tagged with 'particle physics' will be more similar to a document tagged with 'physics'. Also, 'physics' is the *parent* of 'particle physics'. Therefore, the model trained without hierarchy might be leveraging this information implicitly.

Hence, from these reasons we can infer that text classification using StarSpace was not able to benefit from learning hierarchy since it performed almost equally as classifying without learning hierarchy

7.2.3 Further Analysis

Classification type	Fixed parameters	Model parameters	Training Time
Without Hierarchy	lr = 0.1, negative sampling = 40, batch size = 25, adaGrad = True, epochs = 10, thread = 50, similarity = cosine, loss = hinge	dim=300	43min
		dim=400	1h
		dim=500	1h20min
With Hierarchy		dim=300	46min
		dim=400	1h02min
		dim=500	1h30min

Table 7.7: *StarSpace hyperparameters while training models*

A general observation is the time taken to train these models using StarSpace (Table 7.7). Although they have a large number of documents, they can be trained within an hour or two. Essentially, the time taken to train increases linearly with dimension size but that did not affect classification performance as we saw earlier.

We also experimented by increasing number of negative samples (from 10 - 50) and noticed that it did affect classification performance. However, models sampled with 40 negative samples always performed better than the rest, so we stuck to it.

The authors of the main paper (Wu et al., 2017) stated that using 'inner-product' as the similarity measure for performs well for learning relations. However, during our experiments we found that cosine similarity was able to learn relations much better than inner-product. Therefore, we learnt hierarchies using cosine similarity.

In StarSpace, the number of threads affects the time taken to train. This is because *Hogwild!* (Recht et al., 2011) is used during training, *Hogwild!* is a lock-free algorithm that lets each thread modify embeddings without locks [1].

[1] https://github.com/facebookresearch/StarSpace/issues/185

Visualizing Hierarchical Relations

To see how the models projected embeddings we, plotted the embeddings for some examples using t-SNE (Maaten and Hinton, 2008). Figure 7.3 depicts the learnt hierarchical relation embeddings. From

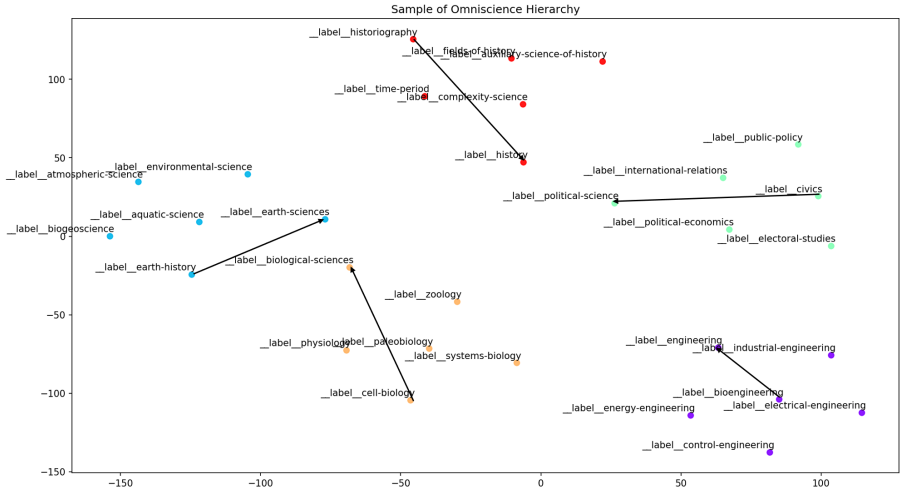
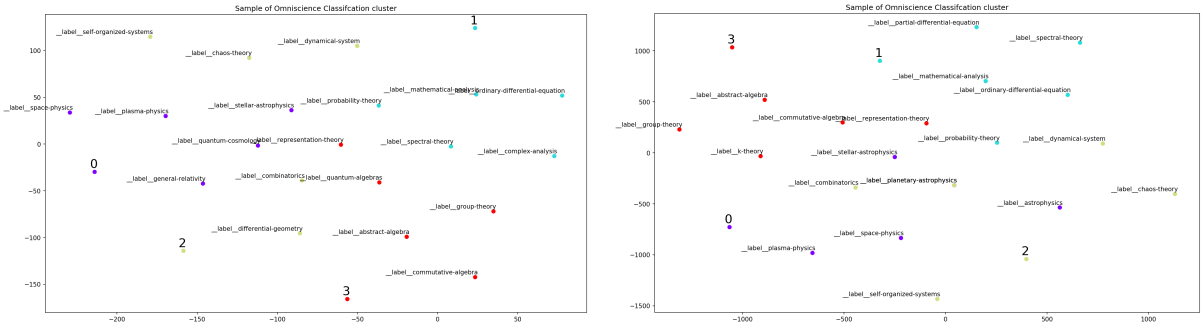


Figure 7.3: *Visualizing Hierarchical Embeddings of OmniScience Taxonomy (dim=500). Different colours indicate different queried child nodes and the retrieved nodes. An arrow indicates the parent-child relationship. Tip of the arrow is the parent node whereas the tail is the child node.*

this figure, we can see that relations which are close in the hierarchy (parent-child and sibling) are reflected in the vector embedding space as well.

Visualizing Classification Embeddings

We also see how the document classification embeddings which were trained with (Figure 7.4a) and without hierarchy (Figure 7.4b) map in the vector space (Maaten and Hinton, 2008).



(a) *Document classification embeddings trained with hierarchy*

(b) *Document Classification Embeddings trained without hierarchy*

Figure 7.4: *Visualization of Document classification examples from Table 7.8. (dim=500)*

Figure 7.4 uses examples from Table 7.8 to visualize embeddings. Although both these models map documents and their most similar labels into the vector space, they do not capture the hierarchy at all.

7.3 Comparison Between Models

In Section 7.1 we concluded that dense features are more suitable for text representation when dealing with hierarchical classification. However, on following up with a model suited for dense representations in Section 7.2.2, we found some surprising results. The StarSpace model from Chapter 6 was not able to capture hierarchy *and* classify documents at the same time. Instead it performed the same as predicting labels when trained without learning hierarchy. One possible way to assess the quality of these dense

Doc id	Document	True Labels	Predicted (with hierarchy)	Predicted (without hierarchy)
0	systemat attempt null convers kinet configur address semi implicit ...	space-physics	__label__plasma-physics **__label__space-physics** __label__stellar-astrophysics __label__quantum-cosmology __label__general-relativity	__label__plasma-physics **__label__space-physics** __label__stellar-astrophysics __label__astrophysics __label__planetary-astrophysics
1	concern hardi inequ euclidean concentr constant inequ	spectral-theory	__label__mathematical-analysis __label__ordinary-differential-equation __label__complex-analysis __label__probability-theory **__label__spectral-theory**	__label__ordinary-differential-equation __label__mathematical-analysis **__label__spectral-theory** __label__probability-theory __label__partial-differential-equation
2	goal connect perron frobenius vertex perron frobenius integ ...	dynamical-system	__label__self-organized-systems __label__chaos-theory **__label__dynamical-system** __label__differential-geometry __label__combinatorics	__label__self-organized-systems __label__combinatorics **__label__dynamical-system** __label__number-theory __label__chaos-theory
3	chevalley algebra subalgebra dist divid power domin highest weyl ...	representation-theory	**__label__representation-theory** __label__commutative-algebra __label__abstract-algebra __label__group-theory __label__quantum-algebras	**__label__representation-theory** __label__abstract-algebra __label__commutative-algebra __label__group-theory __label__function-spaces

Table 7.8: *Prediction results from StarSpace for models trained with and without hierarchy. The predicted labels are ordered in the rank of their prediction. Bold face labels indicate true predicted label.*

representations is if we compare the *FLAT-LR* results of dense features from Chapter 5 with *models trained without hierarchy* from Chapter 6. Table 7.9 illustrates this comparison.

Model	dim	Micro-F1	Macro-F1	h-F1
Flat-LR (FastText)	300	**35.34**	**11.37**	40.08
StarSpace	300	26.28	11.09	**52.21**

Table 7.9: *Comparison of flat classification (dense representation input) with StarSpace model trained for classification without learning hierarchy. Bold face results indicate best performance in that metric.*

From the table above, it can be seen that FLAT-LR performs better than a StarSpace model as a classifier. This can be inferred from the higher *Micro-F1* and *Macro-F1* scores of FLAT-LR. This could be because FastText vectors are generated using bag-of-*n-grams* thereby enabling it to capture more word-order information and this creates better quality of word representation vectors (Joulin et al., 2016) as opposed to StarSpace that modifies word embeddings based on the *similarity* between two entities (Wu et al., 2017). However, StarSpace has a higher *h-F1* than Flat-LR. This could be because of the reasons mentioned in Section 7.2.2, where models trained without hierarchy could be implicitly learning the taxonomy hierarchy. Due to this, the distance between true labels and predicted labels in the hierarchy is reduced giving a better *h-F1* score.

From this discussion we can conclude that FastText representations are better than StarSpace and hence more suitable for text classification.

Chapter 8

Future Work & Conclusion

There is a lot of scope for future work in this field. We can improve parallelization of the recursive regularization model by introducing lock-free SGD updates (Recht et al., 2011) and improve training time substantially. Currently, documents are represented by merely adding the constituent word vectors and normalizing it by unit length. Although it is quite simple and effective, it loses word order information similar to standard bag of words. Instead, we can create better document embeddings using CNNs or other networks that can model long range semantic structure which is lacking in standard bag-of-words approaches (Gan et al., 2016; Liu et al., 2017; Giel and Diaz, 2016). Additionally, we can also improve hierarchical representations using models that favour complex hierarchies (Nickel and Kiela, 2017; Anonymous, 2019).

Thus, in this thesis we tried to see how we can best leverage the information encoded within the hierarchy of a taxonomy (*tree-based hierarchy*) for large-scale datasets to boost classification performance. We do this using two different approaches towards classification. One approach used dense text representation to leverage parent-child relationships encoded in hierarchy through recursive regularization. This method was able to alleviate the *rare categories* problem which usually persists in large-scale hierarchical classification. Another approach learnt hierarchical relations first using a neural embedding model but was not able to leverage these relations while classifying documents. Moreover, we also used hierarchical metrics to evaluate our results which provided more insights apart from the standard metrics. Hence, from the study detailed in this report, we can conclude that classification performance for large scale datasets can be improved using dense representations *coupled* with a suitable method to incorporate inter-dependencies.

Bibliography

Anonymous (2019). Joint learning of hierarchical word embeddings from a corpus and a taxonomy. In *Submitted to Automated Knowledge Base Construction*. under review.

Babbar, R., Partalas, I., Gaussier, E., and Amini, M.-R. (2013). On Flat versus Hierarchical Classification in Large-Scale Taxonomies. In *27th Annual Conference on Neural Information Processing Systems (NIPS 26)*, pages 1824–1832, Lake Tao, United States.

Bengio, S., Weston, J., and Grangier, D. (2010). Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems 23*, pages 163–171. Curran Associates, Inc.

Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python*. OReilly Media. Chapter 2, https://www.nltk.org/book/ch02.html.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795.

Charuvaka, A. and Rangwala, H. (2015). Hiercost: Improving large scale hierarchical classification with cost sensitive learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 675–690. Springer.

D'Alessio, S., Murray, K., Schiaffino, R., and Kershenbaum, A. (2000). The effect of using hierarchical classifiers in text categorization. In *Content-Based Multimedia Information Access-Volume 1*, pages 302–313. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE.

Duchi, J., Hazan, E., and Singer, Y. (2010). Adaptive subgradient methods for online learning and stochastic optimization. Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley.

Gan, Z., Pu, Y., Henao, R., Li, C., He, X., and Carin, L. (2016). Unsupervised learning of sentence representations using convolutional neural networks. *arXiv preprint arXiv:1611.07897*.

Gao, F., Wu, C., Guo, N., and Zhao, D. (2009). Large-scale hierarchical text classification based on path semantic information. In *2009 International Conference on Business Intelligence and Financial Engineering*, pages 223–227.

Giel, A. and Diaz, R. (2016). Document embeddings via recurrent language models. *University of Standford, Tech. Rep*.

Gopal, S. and Yang, Y. (2010). Multilabel classification with meta-level features. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 315–322. ACM.

Gopal, S. and Yang, Y. (2013). Recursive regularization for large-scale classification with hierarchical and graphical dependencies.

Gopal, S., Yang, Y., Bai, B., and Niculescu-Mizil, A. (2012). Bayesian models for large-scale hierarchical classification. In *Advances in Neural Information Processing Systems*, pages 2411–2419.

Gordon, G. and McMahon, E. (1989). A greedoid polynomial which distinguishes rooted arborescences. *Proceedings of the American Mathematical Society*, 107(2):287–298.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759.

Kenter, T., Borisov, A., and De Rijke, M. (2016). Siamese cbow: Optimizing word embeddings for sentence representations. *arXiv preprint arXiv:1606.04640*.

Kiritchenko, S., Matwin, S., and Famili, F. (2005). Functional annotation of genes using hierarchical text categorization.

Korte, B., Vygen, J., Korte, B., and Vygen, J. (2012). *Combinatorial optimization*, volume 2. Springer.

Kosmopoulos, A., Gaussier, E., Paliouras, G., and Aseervatham, S. (2010). The ecir 2010 large scale hierarchical classification workshop. In *ACM SIGIR Forum*, volume 44, pages 23–32. ACM.

Kosmopoulos, A., Partalas, I., Gaussier, E., Paliouras, G., and Androutsopoulos, I. (2015). Evaluation measures for hierarchical classification: a unified view and novel approaches. *Data Mining and Knowledge Discovery*, 29(3):820–865.

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50.

Liu, C., Zhao, S., and Volkovs, M. (2017). Unsupervised document embedding with cnns. *arXiv preprint arXiv:1711.04168*.

Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(1-3):503–528.

Liu, T.-Y., Yang, Y., Wan, H., Zeng, H.-J., Chen, Z., and Ma, W.-Y. (2005). Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explor. Newsl.*, 7(1):36–43.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Madani, O. and Huang, J. (2010). Large-scale many-class prediction via flat techniques. In *Large-Scale Hierarchical Classification Workshop of ECIR*.

McCormick, C. (2011). Word2vec tutorial part 2 - negative sampling. http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/.

Meng, Y., Shen, J., Zhang, C., and Han, J. (2018). Weakly-supervised hierarchical text classification. *CoRR*, abs/1812.11270.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mladenic, D. (1998). Turning yahoo into an automatic web-page classifier.

Naik, A. and Rangwala, H. (2016). Filter based taxonomy modification for improving hierarchical classification. *arXiv preprint arXiv:1603.00772*.

Naik, A. and Rangwala, H. (2017). Integrated framework for improving large-scale hierarchical classification. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 281–288. IEEE.

Nickel, M. and Kiela, D. (2017). Poincare embeddings for learning hierarchical representations. In *Advances in neural information processing systems*, pages 6338–6347.

Partalas, I., Kosmopoulos, A., Baskiotis, N., Artières, T., Paliouras, G., Gaussier, É., Androutsopoulos, I., Amini, M., and Gallinari, P. (2015). LSHTC: A benchmark for large-scale text classification. *CoRR*, abs/1503.08581.

Peng, H., Li, J., He, Y., Liu, Y., Bao, M., Wang, L., Song, Y., and Yang, Q. (2018). Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 1063–1072.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

PKDD2011, E. (2011). Joint ecml/pkdd pascal workshop on large-scale hierarchical classification.

Prim, R. C. (1957). Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401.

Recht, B., Re, C., Wright, S., and Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701.

Salton, G. and Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523.

Sasaki, M. and Kita, K. (1998). Rule-based text categorization using hierarchical categories. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, volume 3, pages 2827–2830. IEEE.

Shahbaba, B., Neal, R. M., et al. (2007). Improving classification when a class hierarchy is available using a hierarchy-based prior. *Bayesian Analysis*, 2(1):221–237.

Sinha, K., Dong, Y., Cheung, J. C. K., and Ruths, D. (2018). A hierarchical neural attention-based text classifier. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 817–823.

Struyf, J., Džeroski, S., Blockeel, H., and Clare, A. (2005). Hierarchical multi-classification with predictive clustering trees in functional genomics. In *Portuguese Conference on Artificial Intelligence*, pages 272–283. Springer.

Wang, K., Zhou, S., and He, Y. (2001). Hierarchical classification of real life documents. In *Proceedings of the 2001 SIAM International Conference on Data Mining*, pages 1–16. SIAM.

Wang, X., Zhao, H., and Lu, B.-l. (2011a). Enhanced k-nearest neighbour algorithm for large-scale hierarchical multi-label classification. In *Proc Joint ECML/PKDD PASCAL Workshop on Large-Scale Hierarchical Classification*.

Wang, X.-L., Zhao, H., and Lu, B.-L. (2011b). Enhance top-down method with meta-classification for very large-scale hierarchical classification. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 1089–1097.

Weigend, A. S., Wiener, E. D., and Pedersen, J. O. (1999). Exploiting hierarchy in text categorization. *Information Retrieval*, 1(3):193–216.

Weinberger, K. Q. and Chapelle, O. (2009). Large margin taxonomy embedding for document categorization. In *Advances in Neural Information Processing Systems*, pages 1737–1744.

Weston, J., Bengio, S., and Usunier, N. (2011). Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, volume 11, pages 2764–2770.

Weston, J., Chopra, S., and Adams, K. (2014). #tagspace: Semantic embeddings from hashtags. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1822–1827. Association for Computational Linguistics.

Wijffels, J. (2019). *ruimtehol: Learn Text 'Embeddings' with 'Starspace'*. R package version 0.2.

Wu, L., Fisch, A., Chopra, S., Adams, K., Bordes, A., and Weston, J. (2017). Starspace: Embed all the things! *arXiv preprint arXiv:1709.03856*. Git repo: https://github.com/facebookresearch/StarSpace.

Yang, Y. (1999). An evaluation of statistical approaches to text categorization. *Inf. Retr.*, 1(1-2):69–90.

Yang, Y. (2001). A study of thresholding strategies for text categorization. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 137–145. ACM.

Yang, Y., Zhang, J., and Kisiel, B. (2003). A scalability analysis of classifiers in text categorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 96–103. ACM.

Zar, J. H. (2005). Spearman rank correlation. *Encyclopedia of Biostatistics*, 7.

Zhou, D., Xiao, L., and Wu, M. (2011). Hierarchical classification via orthogonal transfer.