LARGE SCALE HIERARCHICAL TEXT CLASSIFICATION

Aris Kosmopoulos

PH.D. THESIS

DEPARTMENT OF INFORMATICS
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

2015

Abstract

Hierarchies are becoming increasingly popular for the organization of documents, particularly on the Web. Web directories, such as the *Yahoo! Directory* and the *Dmoz Directory*, are typical examples. Along with their widespread use, comes the need for automated classification of new documents to the classes of the hierarchy. In this thesis, we call this problem *Large Scale Hierarchical Text Classification*. It is a large scale classification problem, since the classes are thousands and the documents can be hundreds of thousands or even millions. It is also hierarchical, since the classes are connected by parent-child relations.

An important issue in hierarchical classification is the evaluation of different classification algorithms, an issue which is complicated by the hierarchical relations among the classes. Several evaluation measures have been proposed for hierarchical classification using the hierarchy in different ways without however providing a unified view of the problem. In this thesis, we study the problem of evaluation in hierarchical classification by analysing and abstracting the key components of the existing performance measures. We also propose two alternative generic views of hierarchical evaluation and introduce two corresponding novel measures. The proposed measures, along with the state-of-the-art ones, are empirically tested on three large datasets from the domain of text classification. The empirical results illustrate the limitations of existing approaches and how the proposed methods overcome most of them across a range of cases.

We then focus on the simplest case of large scale hierarchical text classification,

where the hierarchy is a tree and each document belongs in a single leaf class of the hierarchy. A popular method of hierarchical classification is cascade classification, which greedily traverses the hierarchy from the root to the predicted leaf. In order to perform cascade classification, a classifier must be trained for each node of the hierarchy, but in the upper levels the number of features can be prohibitively large. It is therefore desirable to reduce the dimensionality of the feature space at these levels. We examine the computational feasibility of the most common dimensionality reduction method (Principal Component Analysis) for this problem, as well as the computational benefits that it provides for cascade classification and its effect on classification accuracy. Furthermore, we propose a probabilistic cascading approach, which outperforms the traditional greedy cascade, by making better use of the probabilities estimated by the classifiers.

Finally, we consider a more complex domain, known as *biomedical semantic indexing*, where biomedical documents have to be classified to the classes of a large biomedical taxonomy. This domain is more complex in that the taxonomy is a directed acyclic graph, rather than simply a tree, the same document may belong in several classes, and the correct classes are not necessarily leaves of the taxonomy. We examine the use of dense word vectors, also known as *word embeddings*, as a method of dimensionality reduction. We consider several efficient approaches for the transition from dense word vectors to vectors that represent entire texts, proposing a simple weighted centroid approach that is suitable for this domain. We show that by adopting this approach, hierarchical text classification algorithms become sufficiently scalable for large scale semantic indexing, without being less effective than then usual bag of words representation. We experiment with flat and hierarchically expanded k-nearest neighbor classifiers that employ our centroid representations of article abstracts, examining the effect of various parameters. We also present a high precision system that can be combined with the widely used Medical Text Indexer (MTI) system of the National Library of Medicine to improve its performance.

Acknowledgements

I would like to thank my supervisors, Georgios Paliouras and Ion Androutsopoulos, for their guidance and support throughout my studies. I would also like to thank professor Eric Gaussier and Ioannis Partalas for our cooperation during the LSHTC challenges and our work regarding the hierarchical evaluation measures. Special thanks to George Giannakopoulos and Anastasios Skarlatidis for the long discussions and help during these years. I would like to express my gratitude to NCSR "Demokritos", where I conducted most of my research, for providing financial support and the necessary infrastructure for all these demanding experiments. Last but not least, I wish to thank all the members of the Natural Language Processing Group of AUEB's Department of Informatics and of course my friends and family!

Contents

Chapter 1

Introduction

1.1 Large Scale Hierarchical Text Classification

Text classification deals with the problem of assigning documents to a predefined set of classes. In the simplest case, there is just one class and each document either belongs to it or not. Spam filtering is such an example, where emails must be classified as desirable or not. In machine learning a classifier can be trained using positive and negative instances in order to perform this procedure automatically, although even in this simplest case the predictions of the classifier are rarely 100% correct.

In this thesis, we study a much more complex case. In large scale text classification the number of classes can run into thousands. In some cases each document may only belong to a single class (*single-label* classification), while in others to more than one (*multi-label* classification). The number of documents is also very large (hundreds of thousands or even millions), leading to a very large vocabulary (unique different words in the documents, also known as *types*)

Another aspect of the problem is that the classes are connected with each other through parent-child relations composing a *hierarchy* (also called *taxonomy*). A class taxonomy is important for two reasons. First it offers extra information to a classifica-

tion system, which in theory can be exploited either to improve scalability or to improve accuracy or even both. Second it can affect the evaluation of a classification system. For example in Figure 1.1 the true class is *Pop* and we wish to compare two systems predicting the classes *Rock* and *Theater* respectively. According to any flat evaluation measure both predictions would be simply wrong, since flat evaluation measures ignore the hierarchy.

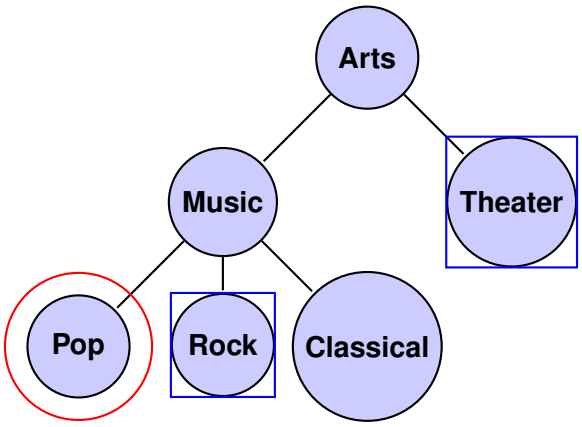


Figure 1.1: Two examples of missclassification. The true class is marked with a red circle, while the two predictions are marked with blue squares.

A hierarchical evaluation measure should consider the hierarchy and assign a smaller penalty to the system predicting *Rock*, compared to the one predicting *Theater*. This is based on the intuition that confusing *Pop* with *Theater* is worse than confusing it with another type of music, such as *Rock*.

1.2 Contribution of this Thesis

In this thesis, we start by describing the problems and limitations that existed in this domain at the outset of this work. Initially there were no suitable large scale hierarchical datasets publicly available. Therefore, one of our main considerations was to create some and offer them to the research community. A series of challenges and workshops

were organized in this direction (the LSHTC [1] and BioASQ [2] challenges). By studying the participating systems and through the discussions that took place during the workshops of these challenges, we made three main observations regarding the gaps that existed in state-of-the-art research:

- Flat evaluation measures were insufficient for the evaluation of hierarchical classification systems.
- The scalability of the problem led many participants to focus on simpler and more efficient classifiers.
- Many systems ignored the hierarchy by applying flat classification techniques.

Taking into account the above observations, we first focused on evaluation measures for hierarchical classification. Although some hierarchical evaluation measures already existed, they were rarely used in practice. They suffered from a number of limitations regarding different aspects of the problem, such as the type of the hierarchy. The main contribution of this thesis in this direction was a study all the existing hierarchical evaluation measures, by analysing and abstracting their key components under a unified view that helps to understand them and realise their limitations. We proposed two alternative generic views of hierarchical evaluation and introduced two corresponding novel measures. The proposed measures, along with the state-of-the-art ones, were empirically tested on three large datasets from the domain of text classification. The empirical results illustrated the limitations of existing approaches and how the proposed methods overcome most of these problems across a range of cases.

Given the scalability problems faced by many participants, we then studied the effect of dimensionality reduction in the simplest case where the hierarchy is a tree and each document belongs to a single leaf class. In order to perform cascade classification,

[1] http://lshtc.iit.demokritos.gr

[2] http://bioasq.org/

a common top-down approach discussed in the following chapters, a classifier must be trained for each node of the hierarchy, but in the upper levels the number of features can be prohibitively large. It is therefore desirable to reduce the dimensionality of the feature space at these levels. We examined the computational feasibility of the most common dimensionality reduction method (Principal Component Analysis) for this problem and measured the computational benefits that it provides for cascade classification and its effect on classification accuracy. Furthermore, we proposed a probabilistic cascading methodology, which outperforms the traditional greedy cascade, by making better use of the probabilities estimated by the classifiers.

We then dealt with a more complex problem, by experimenting with the datasets of the BioASQ challenge on large scale biomedical semantic indexing. This BioASQ task is more complex, since it is multi-labeled, the hierarchy is not a tree (each class can have more than one parent) and it is larger compared to those used in our previous work. Given these considerations, we examined the use of dense word vectors (also known as *word embeddings*) as an efficient method of dimensionality reduction that makes hierarchical text classification algorithms more scalable in biomedical semantic indexing, without being less effective than the usual bag-of-words representation. We considered several approaches for the transition from dense word vectors to dense vectors that represent entire texts, proposing an approach that is suitable to this domain and also computationally efficient. We experimented with flat and hierarchically expanded K-nearest neighbor classifiers that employ dense vector representations of article abstracts, examining the effect of various parameters. We also presented a high precision system that can be combined with the Medical Text Indexer (MTI) system of NLM to improve its performance.

1.3 Outline of the Remainder of this Thesis

The remainder of this thesis is organized as follows: Chapter 2 describes our work on the creation of large-scale text classification benchmarks from the prespective of the LSHTC challenges; Chapter 3 presents our research on evaluation measures for hierarchical classification; Chapter 4 discusses our approaches to single-label classification on tree hierarchies; Chapter 5 investigates the use of dense word vectors for classification in the biomedical domain and Chapter 6 concludes.

Chapter 2

Creating Large-Scale Text Classification Benchmarks

2.1 Introduction

One of the main problems that we faced in the early stages of the thesis was the unavailability of open-access large-scale hierarchical text datasets. This fact prompted us to initiate a series of challenges on Large Scale Hierarchical Text Classification (LSHTC).[1] The LSHTC challenges ran from December 2009 until 2014 in four editions, and attracted more than 150 teams from around the world (USA, Europe and Asia). The results of the challenges were presented in subsequent workshops at the conferences ECIR 2010, ECML 2011, ECML 2012 (where the challenge was the discovery challenge of the conference) and WSDM 2014.

The LSHTC initiative aimed to assess the performance of classification systems in large-scale classification using a large number of classes. It included tracks of various scales in terms of classes, from thousands to hundreds of thousands as well as many flavors of the classification problems, standard classification problem, multi-task clas-

[1]The LSHTC challenges are presented in (Kosmopoulos et al., 2010; Partalas et al., 2015).

sification and unsupervised classification. Most tracks were based on two corpora from Wikipedia[2] and from the ODP Web directory data[3]. The LSHTC training datasets may be downloaded from the permanent LSHTC website, where one may still submit a run on the test datasets and get the performance of a system ranked among the systems that participated in the task.[4]

As mentioned above the data used in LSHTC comes from two popular sources: DBpedia[5] and the ODP (Open Directory Project) directory, also known as DMOZ[6]. DBpedia instances were selected from the English, non-regional Extended Abstracts provided by the DBpedia site. The DMOZ instances consist of either *Content* vectors, *Description* vectors or both. A *Content* vector is obtained by directly indexing the web page using a standard indexing pipeline (pre-processing, stemming/lemmatization, stop-word removal). A *Description* vector is created by indexing the ODP descriptions of the web pages, which are manually created by the ODP editors.

In the next section we will provide some basic information regarding the datasets that we created and offered to the community. Sections 2.3, 2.4, 2.5 and 2.6 describe the tasks, evaluation measures and the main approaches in each iteration of the LSHTC challenges. Finally, Section 2.7 concludes the chapter.

2.2 Data Creation

Each dataset is provided in a sparse vector format file, where each line corresponds to an instance. Here is an example of an instance in sparse vector format:

```
5 0:10 8:1 18:2 54:1 442:2 3784:1 5640:1 43501:1
```

[2] www.wikipedia.org

[3] www.dmoz.org

[4] http://lshtc.iit.demokritos.gr

[5] http://dbpedia.org/About

[6] http://www.dmoz.org/

The first number (5 in the example) corresponds to the category of the instance. In the case of multi-label classification comma-separated numbers are used instead in order to define the categories of and instance.

Each set of numbers separated by ':' correspond to a (feature,value) pair of the vector, where the first number is the feature's id and the second number its frequency (for example the feature with id 18 appears twice in the instance). The special feature id 0 is used for internal indexing (it corresponds to a document id) and should by ignored during classification.

Each token and category of a dataset are mapped to unique numbers. In each track of the challenge a different mapping was used for tokens and categories so that no information could be carried over between tracks. Instances of a track are either split into training, validation and test data or just training and test data. If a validation file is not provided in a track, participants could create one, using a subset of the training file. All test instances belong to category 0, meaning that their true label is hidden from the participants.

For each track dataset a hierarchy file is also provided. This file contains the parent-child relations between categories. In the first challenge (LSHTC1) this file provided category paths from root to leaf of the category tree. In the following challenges each line of the file defined a relation between a parent and a child category.

The hierarchy of the DMOZ datasets is a tree (each node has a single parent) with a maximum depth of 5. All instances deeper than 5 nodes from the root are assigned to their ancestor in depth 5.

The hierarchy of the DBpedia datasets is a graph (a node can have more than one parent), containing also cycles. In some datasets these cycles were removed by ignoring nodes that have been visited already in paths from root to leaf (i.e. any parent-child relation that would lead to cycle was omitted). In most datasets classification was only allowed to leaf nodes. In case some inner node of the hierarchy contained any instance,

an artificial new node was added as a child of this node and all instances belonging to the initial node were reassigned to the new one.

2.3 LSHTC1

2.3.1 Tracks and Evaluation

The tracks of the first year of the challenge were based on the DMOZ dataset (tree hierarchy) using only single-label instances. The challenge was split into 4 tracks based on different combinations of *Content* and *Description* vectors. Since both types of vector were used in this challenge only the intersection of the two sets of instances could be used (i.e. we used only instances which had both a *Content* and *Description* vector). In Table 2.1 we present which type of vector was given in each track for training and test.

Track Name	Content		Description	
	Train	Test	Train	Test
Track 1: Basic	✓	✓	-	-
Track 2: Cheap	-	✓	✓	-
Track 3: Expensive	✓	✓	✓	-
Track 4: Full	✓	✓	✓	✓

Table 2.1: *Content* and *Description* vectors per track of the first LSHTC. Track 1 is called Basic because it used only the major type of vector (*Content* vector). Track 2 is called Cheap because it uses only *Decription* vectors for training, which are generally smaller than the *Content* vectors. The third one is called Expensive because it requires both types of vector for training and the final one is called Full since it uses all the data for both training and test.

For the tracks of the first challenge a smaller (dry-run) dataset was also provided to facilitate the development of new systems. This dataset was a subset of the respective main track dataset, using different mappings in order to create the category ids and

tokens ids.

In Table 2.2 we present some statistics regarding the main track and dry-run datasets of the first LSHTC. In all the datasets of this iteration, classification was only allowed at leaf nodes.

Dataset type	Main	Dry-run
Number of categories	12,294	1,139
Number of training instances	93,805	4,463
Number of validation instances	34.905	1.860
Number of test instances	34.880	1,858

Table 2.2: Statistics regarding each dataset of the first LSHTC.

In the first challenge we used accuracy, precision, recall and F_1 (harmonic mean of precision and recall) as flat evaluation measures. Flat evaluation measures ignore the hierarchy and treat all misclassification errors in the same way, regardless of the distance between the predicted and correct classes. In single-label classification accuracy, is defined as:

$$accuracy = \frac{\sum_{i=1}^{|C|} TP_i}{|D|}$$

where TP_i is the number of documents correctly classified (true positives) and $|D|$ is the number of test documents.

Precision (P_i) of a class C_i (node of the hierarchy) is the number of test documents correctly classified in C_i (true positives, TP_i), divided by the number of test documents the classifier placed in C_i (true positives and false positives, $TP_i + FP_i$). Recall (R_i) of a class C_i is the number of test documents correctly classified in C_i (TP_i), divided by the number of test documents that truly belong in C_i (true positives and false negatives, $TP_i + FN_i$). The F_1 score of C_i combines P_i and R_i.

$$P_i = \frac{TP_i}{TP_i + FP_i}, \quad R_i = \frac{TP_i}{TP_i + FN_i}, \quad F_{1,i} = \frac{2 \cdot P_i \cdot R_i}{P_i + R_i}$$

Macro precision and macro recall average P_i and R_i over all the $|C|$ classes.

$$\text{Macro Precision} = \frac{1}{|C|} \sum_{i=1}^{|C|} P_i, \quad \text{Macro Recall} = \frac{1}{|C|} \sum_{i=1}^{|C|} R_i$$

Macro F_1 combines macro precision and macro recall.

$$\text{Macro}\,F_1 = \frac{2 \cdot \text{Macro Precision} \cdot \text{Macro Recall}}{\text{Macro Precision} + \text{Macro Recall}}$$

Micro versions of these measures provide always the same results with the macro versions in single-label classification, so there is no point using them here.

On the other hand, hierarchical evaluation measures take into account the positions of the predicted and correct classes in the hierarchy. In the first LSHTC we used the most basic hierarchical evaluation measure, which is tree-induced error (usable only for single-label classification with a **tree** hierarchy). In Figure 2.1 for example, where the correct class is *Pop* and the predicted one is *Theater*, in order to compute tree-induced error we must measure the distance between nodes *Pop* and *Theater*.

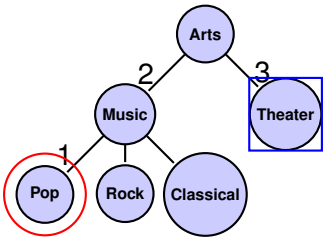


Figure 2.1: Tree-induced error example. The correct class is *Pop* and the predicted one is *Theater*.

The total error is measured as the average error over all test documents:

$$\text{Tree-induced error} = \frac{\sum_{i=1}^{D} \text{Shortest-path}(P_i, T_i)}{|D|}$$

where $|D|$ is the number of test documents, P_i the category in which the document was classified and T_i the true category of the document. Since tree-induced error is an error the lower it is the better the system performs (a system that predicts all classes correctly would have zero error).

System	Description	Hierarchical
alpaca	SVMs	✓
(Madani and Huang, 2010)	Online training	-
(Miao and Qiu, 2009)	Centroid based	✓

Table 2.3: Best systems of LSHTC challenge.

2.3.2 Participation

In the first iteration of the LSHTC challenge, 19 systems participanted in the 4 tracks. All of them participated in the first track, while less than half participated in the other 3 tracks. In Table 2.3 we present the best systems, with some basic information about them. The most interesting result was that one of the two best state of the art systems was hierarchical and the other one flat. The first (alpaca) used binary SVMs per class and expoited the hierarchy by computing a weighted combination of the leaf classifier score and all of its ancestors for each leaf. The second (Madani and Huang, 2010) used online training techniques and ignored the hierarchical relations of the classes. Another approach that performed very well was by Miao and Qiu (2009), which used centroid classification and mildly exploited the hierarchy.

2.4 LSHTC2

2.4.1 Tracks and Evaluation

In LSHTC2 we used multi-labeled instances and non-tree hierarchies. For DMOZ, instead of using both types of vectors (*Content* and *Description* vectors), we decided to keep one of them. We kept the *Content* vectors, since they did not require human annotation. Since we decided to move to multi-label classification, we used all the *Content* vectors that we had.

LSHTC2 consisted of three tracks, with the first one being the multi-label DMOZ dataset, which as we explained above has a tree type hierarchy. The other two tracks

were based on DBpedia datasets. Track 3 consisted of all the extended abstracts, while Track 2 was a subset of the Track 3 dataset containing fewer instances and categories. While the hierarchy of Track 3 contained cycles, the one of Track 2 was cleaned in order to be a DAG. Table 2.4 presents basic statistics regarding the datasets of LSHTC2.

Dataset	DMOZ	Medium DBpedia	Large DBpedia
Number of categories	27,875	36,504	325,056
Number of training instances	394,754	456,886	2,365,436
Number of test instances	104,263	81,262	452,167
Number of stems	594,158	346,299	1,617,899
Average categories per inst	1.02	1.86	3.26
Deepest leaf in graph	5	10	14

Table 2.4: Statistics regarding each dataset of LSHTC2.

In LSHTC2 we required evaluation measures appropriate for multi-label classification. In multi-label classification, as explained in (Tsoumakas and Vlahavas, 2007), the micro versions of precision and recall differ from the macro ones. Macro versions are defined as in single-label classification, while, micro precision and micro recall are defined as follows:

$$\text{Micro Precision} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} TP_i + FP_i}, \quad \text{Micro Recall} = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} TP_i + FN_i}$$

Micro F_1 combines micro precision and micro recall.

$$\text{Macro}\,F_1 = \frac{2 \cdot \text{Micro Precision} \cdot \text{Micro Recall}}{\text{Micro Precision} + \text{Micro Recall}}$$

The micro-averaged measures assign greater importance to larger classes that consist of many test instances; consequently, rare classes do not influence much the results. By contrast, the macro-averaged measures assign equal importance to all classes. We also report accuracy scores, i.e., the number of correctly classified test instances divided by the total number of test instances. Accuracy, in multi-label classification is also defined differently:

$$\text{Accuracy} = \frac{\sum_{i=1}^{|D|} \frac{|P_i \cap T_i|}{|P_i \cup T_i|}}{|D|}$$

where P_i and T_i the predicted and true label sets of document i. Precision and recall may also be defined per document and then averaged (as done for accuracy). This is called in (Tsoumakas and Vlahavas, 2007) example based precision (EBP) and recall EBR:

$$\text{EBP} = \frac{\sum_{i=1}^{|D|} \frac{|P_i \cap T_i|}{|P_i|}}{|D|}$$

$$\text{EBR} = \frac{\sum_{i=1}^{|D|} \frac{|P_i \cap T_i|}{|T_i|}}{|D|}$$

As a hierarchical evaluation measure we designed a multi-label version of tree-induced error which we named graph-induced error. We will describe it in detail in Chapter 3, together with other hierarchical evaluation measures.

2.4.2 Participation

In LSHTC2, there were 3 tracks with 16 participants. All of them participated in the first track and half of them participated in the other two tracks. In Table 2.5 we present the best systems, with some basic information regarding them. Interestingly, the winning systems in the different tasks are flat: the first one (Brouard, 2011) uses an associative network coupled with a post processing of the scores; the second one (Wang et al., 2011) uses a k-NN approach based on a BM25 similarity and a thresholding strategy. Nevertheless, many systems obtained very good and very similar results using either hierarchical strategies or flat ones.

System	Description	Hierarchical
(Brouard, 2011)	associative network	-
(Wang et al., 2011)	K-nn with BM25 similarity	-

Table 2.5: Best two systems of the LSHTC2 challenge.

2.5 LSHTC3

2.5.1 Tracks and Evaluation

The two DBpedia datasets were also used in the first track of LSHTC3. In particular for the Medium DBpedia dataset, we provided also the original text of the instances, without pre-processing. LSHTC3 contained also tracks about multi-task learning and refinement learning, but these two tracks are not related to this thesis.

The evaluation of LSHTC3 was done with the same measures used in LSHTC2. The only addition was the implementation of hierarchical versions of precision and recall, which are described in Chapter 3.

2.5.2 Participation

In LSHTC3, there were 16 participants for the first track and very few for the other tracks. Flat and hierarchical approaches performed competitively in track 1. In Table 2.6 we present the best systems, with some basic information about them. The winning system in the first track is based on the approach of Wang et al. (2014), a hierarchical approach which has the particularity of considering the multi-class classification as a meta-learning problem. It starts by constructing a classifier for each node of the hierarchy and then extracts meta-features for a sample from the accuracy scores of each classifier in the tree. Once the meta-estimator is learned, thresholding strategies are used to classify a sample in the multi-class setting. Another hierarchical approach in the top tier is from Sasaki and Weissenbacher (2012). They consider a usual hierarchical framework, learning a classifier for each edge of the hierarchy and combining it with a threshold pruning strategy to improve multi-class classification results.

The flat classification approaches in the top-tier were competitive with the hierarchical ones. Puurural and Bifet (2012a) use an ensemble of multinomial Naive Bayes classifiers with optimization strategies. Han et al. (2012) use a k-NN based approach,

System	Description	Hierarchical
(Wang et al., 2014)	Top-down and meta-features	✓
(Sasaki and Weissenbacher, 2012)	Pruning strategy for multi-class	✓
(Puurula and Bifet, 2012a)	Ensemble of multinomial naive bayes	-
(Han et al., 2012)	k-NN based approach and ranking	-
(Lee, 2012)	Centroid similarity Rocchio classification	-

Table 2.6: Some of the best systems of the LSHTC 3 challenge.

by retrieving the most similar training examples and exploit also the hierarchical information to provide a ranking for each possible class. Lee (2012) uses a modification of the Rocchio classifier, based on the centroid similarity between an example and the classes. In order to extend the approach to multi-label classification treats each original subset of labels as a separate new label.

2.6 LSHTC4

During LSHTC 4, only the Large DBpedia dataset was used for the first track called "Very Large Supervised Learning", and the challenge was performed through Kaggle.[7] LSHTC4 aimed at attracting more participants to the existing tracks, rather than introducing new challenges. Indeed participation increased to more than a hundred teams, but no description of their systems were made available. Performance according to the only measure of the challenge (macro-F_1) was also increased.

2.7 Conclusions

By studying the participating systems and through the discussions that took place during the workshops of these challenges, we made some main observations regarding the gaps that existed in state-of-the-art research:

- Flat evaluation measures were insufficient for the evaluation of hierarchical clas-

[7] http://www.kaggle.com/

sification systems.

- The existing hierarchical evaluation measures were rarely used and were not good enough.
- The scalability of the problem led many participants to focus on simpler and more efficient classifiers.
- Many systems ignored the hierarchy and used flat classification techniques.

Another very interesting and unexpected observation was related also to the use of flat classification techniques. Since many particpants already had systems for flat classification, they were expected to use them also on the hierarchical problem, ignoring the hierarchy. What we did not expect was that the best flat systems would perform very close to the best hierarchical ones. Even today many researchers in this domain tend to ignore the hierarchy in their models.

In Chapter 3 we examine the existing hierarchical evaluation measures and we propose two new ones which deal with the main limitations of existing ones.

Given the scalability problems faced by many participants, we studied the effect of dimensionality reduction in the simplest case where the hierarchy is a tree and each document belongs to exactly one leaf class. We present this work in Chapter 4.

Chapter 3

Evaluation Measures for Hierarchical Classification: a Unified View and Novel Approaches

3.1 Introduction

Hierarchical classification addresses the problem of classifying items into a hierarchy of classes.[1] In past years mainstream classification research did not place enough emphasis on the presence of relations between the classes, in our cases hierarchical relations. This is gradually changing and more effort is put into hierarchical classification in particular, partly because many real-world knowledge systems and services use a hierarchical scheme to organize their data (e.g. Yahoo, Wikipedia). Research in hierarchical classification has become important, because flat classification algorithms are ill-equipped to address large scale problems with hundreds of thousands of hierarchically related classes.

Several evaluation measures have been proposed for hierarchical classification (HC)

[1] This work has already been published (Kosmopoulos et al., 2014b).

(Costa et al., 2007; Sokolova and G., 2009) Many research questions in HC remain open. An important issue is how to properly evaluate HC algorithms. While standard flat classification problems have benefited from established measures such as precision and recall, there are no established evaluation measures for HC tasks, where the assessment of an algorithm becomes more complicated due to the relations among the classes. For example, classification errors in the upper levels of the hierarchy (e.g. when wrongly classifying a document of the class `music` into the class `food`) are more severe than those in deeper levels (e.g. when classifying a document from `progressive rock` as `alternative rock`). using the hierarchy in different ways. Nevertheless, none of them is widely adopted, making it very difficult to compare the performance of different HC algorithms.

A number of comparative studies of HC performance measures have been published in the literature. The early study of Sun et al. (2003) is limited to a particular type of graph-distance measures. Costa et al. (2007) present a review of HC measures, focusing on single-label tasks and without providing any empirical results; in multi-label tasks each object can be assigned to more than one classes, e.g. a newspaper article may belong to both `politics` and `economics`. Nowak et al. (2010) compare many multi-label evaluation measures, but the role of the hierarchy is not emphasized. Finally, Brucker et al. (2011) provide a comprehensive empirical analysis of HC performance measures, but they focus on the evaluation of clustering methods rather than classification ones. While these studies provide interesting insights, they all miss important aspects of the problem of evaluating HC algorithms. In particular, they do not abstract the problem in order to describe existing evaluation measures within a common framework.

The work presented here addresses these issues by analyzing and abstracting the key components of existing HC performance measures. Specifically:

1. It groups existing HC evaluation measures under two main types and provides a

generic framework for each type, based on flow networks and set theory.

2. It provides a critical overview of the existing HC performance measures using the proposed framework.

3. It introduces two new HC evaluation measures that address important deficiencies of state-of-the-art measures.

4. It provides comparative empirical results on large HC datasets from text classification with a variety of HC algorithms.

The remainder of this chapter is organized as follows. Section 3.2 introduces the problem of HC, presents general requirements for HC measures and the proposed frameworks. Furthermore, it presents existing HC evaluation measures using the proposed frameworks and introduces two new measures that address problems the state-of-the-art measures have. Section 3.3 presents a case study comparison and analysis of the proposed measures and the existing ones. Section 3.4 describes the setting and data of the empirical analysis of the measures and Section 3.5 presents and discusses the empirical results. Finally, Section 3.6 concludes and summarizes remaining open issues.

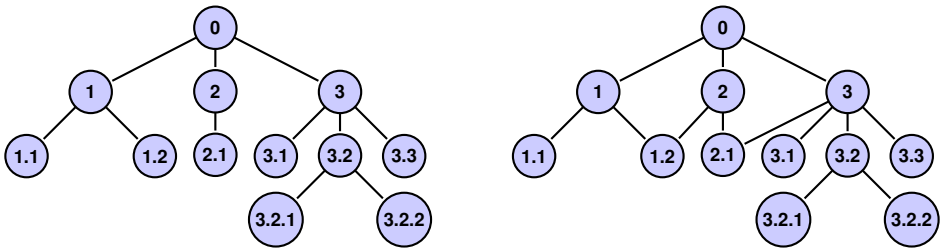
3.2 A Framework of Hierarchical Classification Performance Measures

In classification tasks the training set is typically denoted as $S = \left\{(\mathbf{x}^i, \mathbf{y}^i)\right\}_{i=1}^{n}$, where $\mathbf{x}^i \in \mathscr{X}$ is the feature vector of instance i in the input space $\mathscr{X}$ and $\mathbf{y}^i \in \mathscr{Y}$ is the set of classes in which the instance belongs, where $\mathscr{Y} = \{y_1, \ldots, y_K\}$ is the set of the available classes.

We define, as the *first dimension* (**D1**) of the HC problem, whether it is single-label or multi-label. In the single-label case, which is the simplest one, each instance i

belongs to a single category (the size of the vector $\mathbf{y}^i$ is 1), while in the multi-label case an instance i might belong to more than one category (the size of the vector $\mathbf{y}^i$ is ≥ 1).

In contrast to flat classification, where the classes are considered unrelated, in HC the classes are organized in taxonomies. The type of the taxonomy is the *second dimension* (**D2**) of HC. It can be either a tree, in which case nodes (classes) have a single parent each, or directed acyclic graphs (DAGs), in which case nodes can have multiple parents; see Figures 3.1(a) and 3.1(b) respectively. In some cases, hierarchies may also be cyclic graphs. In all cases the hierarchy imposes a parent-child relation among the classes, which implies that an instance belonging in a specific class, *also belongs in all its ancestor classes*. A taxonomy is thus usually defined as a pair $(\mathcal{V}, \prec)$, where $\mathcal{V}$ is the set of all classes (Silla et al., 2011) and $\prec$ is the `subclass-of` relationship with the following properties:[2]



(a) Tree (b) DAG

Figure 3.1: A tree and a DAG class hierarchy.

- Asymmetry: if $v_i \prec v_j$ then $v_j \not\prec v_i$ for every $v_i, v_j \in \mathcal{V}$.
- Anti-reflexivity: $v_i \not\prec v_i$ for every $v_i \in \mathcal{V}$.
- Transitivity: if $v_i \prec v_j$ and $v_j \prec v_k$, then $v_i \prec v_k$ for every $v_i, v_j, v_k \in \mathcal{V}$.

[2]Without loss of generality, we assume a `subclass-of` relationship among the classes, but in some cases a different relationship may hold, for example `part-of`. We assume however, that the three properties always hold for the relationship.

In graphs with cycles, only the transitivity property holds. In this thesis we consider only hierarchies without cycles and we denote the descendants and ancestors of a class $v \in \mathscr{V}$ as $De(v)$ and $An(v)$, respectively. The parents of a class v are denoted as $Pa(v)$.

The *third dimension* (**D3**) examines whether an instance can be classified only to a leaf or to any class of the hierarchy. The former problem is also known as "mandatory leaf node prediction" (Silla et al., 2011). In all cases considered in this work an instance must always be classified to at least one class (it cannot be rejected by the root class).

Table 3.1 presents the descriptions of the most frequent symbols used throughout this chapter.

Symbol	Description
$\mathbf{x}$	a feature vector
y	a class label
$De(v), An(v), Pa(v)$	descendants, ancestors and parents of class v
$\hat{Y}, Y$	sets of predicted and true classes
$\hat{Y}_{aug}, Y_{aug}$	augmented sets of predicted and true classes
$[b_u, c_u]$	capacity intervals of an edge u of a flow network
G	a graph
LCA	lowest common ancestor

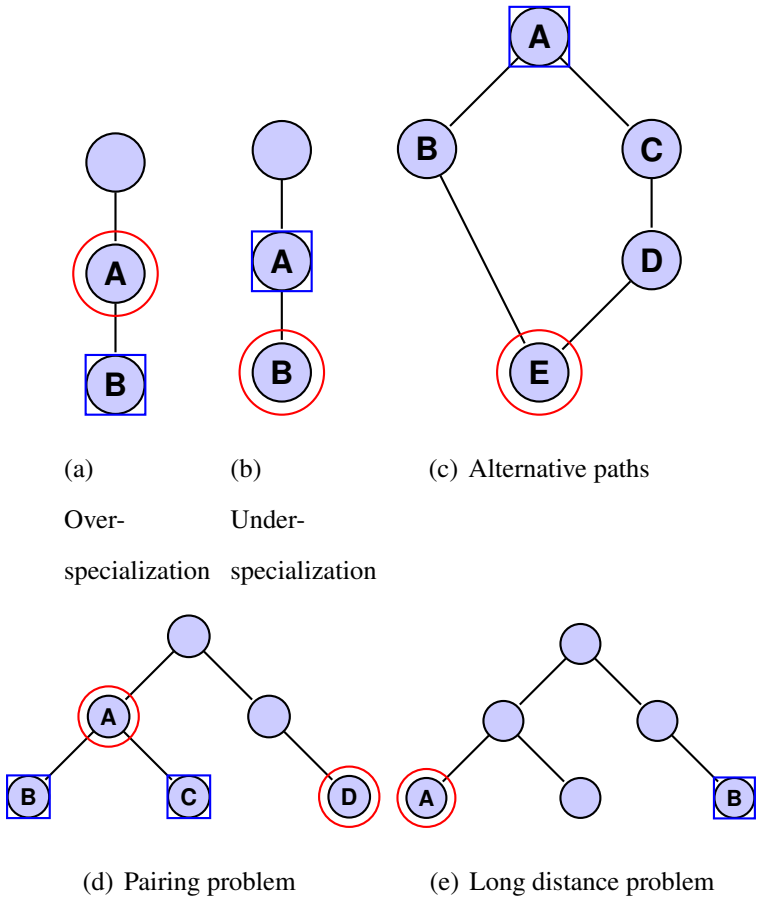
Table 3.1: Description of symbols used throughout the chapter.

3.2.1 General Problems in Hierarchical Classification Evaluation

The commonly used measures of precision, recall, F-measure, accuracy etc. are not appropriate for HC, due to the relations that exist among the classes. A hierarchical performance measure should use the class hierarchy in order to evaluate properly HC algorithms. In particular, one must account for several different types of error according to the hierarchy. For example, consider the tree hierarchy in Figure 3.1. Assume that the true class for a test instance is `3.1` and that two different classification systems output `3` and `1` as the predicted classes. Using flat evaluation measures, both systems are punished equally, but the error of the second system is more severe as it makes a

prediction in a different and unrelated sub-tree.

In order to measure the severity of an error in HC, there are several interesting issues that need to be addressed. Figure 3.2 presents five cases that require special handling. In all cases, the nodes surrounded by circles are the true classes, while the nodes surrounded by rectangles are the predicted ones. These cases can be sub-grouped into a) *pairing* problems (Figures 3.2(d) and 3.2(e)) where one must select which pairs of predicted and true classes to take into account for the calculation of the error, and b) *distance-measuring* problems (Figures 3.2(c), 3.2(a) and 3.2(b)) which concern the way that the error will be calculated for a pair of predicted and true classes.



(a) Over-specialization (b) Under-specialization (c) Alternative paths

(d) Pairing problem (e) Long distance problem

Figure 3.2: Interesting cases when evaluating hierarchical classifiers. Nodes in circles are the true classes while nodes in rectangles are the predicted classes.

The first two *distance-measuring* cases are linked with dimension **D3** of the HC

problem. They appear only when classification to inner nodes is allowed. Figure 3.2(a) presents an *over-specialization* error where the predicted class is a descendant of the true class. Figure 3.2(b) depicts an *under-specialization* error, where an ancestor of the true class is selected. In both these cases the desired behavior of the measure would be to reduce the penalty to the classification system, according to the distance between the true class and the predicted one.

The third case (Figure 3.2(c)), called *alternative paths*, presents a scenario where there are two different ways to reach the true class starting from a predicted class. In this case, a measure could use one of the two paths or both in order to evaluate the performance of the classification system. Selecting the path that minimizes the distance between the two classes and using that as a measure of error seems reasonable. In Figure 3.2(c) the predicted class is an ancestor of the true class, but an alternative paths case may also involve multiple paths from an ancestor to a descendant predicted class. This case appears only in DAG taxonomies and in that way it is related to the **D2** dimension of the HC problem.

Figure 3.2(d) presents a scenario linked with the **D1** dimension of the HC problem, which is only common in multi-label data. In this case one must decide, before even measuring the error, which pairs of true and predicted classes should be compared. For example, node `A` (true class) could be compared to `B` (predicted) and `D` to `C`; or node `A` could be compared to both `B` and `C`, and node `D` to none; other pairings are also possible. Depending on the pairings, the score assigned to the classifier will be different. It seems reasonable to use the pairings that minimize the classification error. For example, in Figure 3.2(d) it could be argued that the prediction of `B` and `C` is based on evidence about `A` and thus both `B` and `C` should be compared to A.

Finally, Figure 3.2(e) presents a case where the predicted class should probably not be matched to any true class. This is typically the case when the predicted class and the true class are too distant, which is why we call this case the *long distance problem*.

The simplest case scenario is when we have a single-label classification problem, with a tree hierarchy and where instances are only allowed to be classified to leaves. The first four cases, do not appear in this scenario and in this way all the existing hierarchical evaluation measures have a similar behavior. If we alter any dimension of the problem we will have to deal with some of the above cases and this is where the behavior of the existing measures differs. Only the last case (3.2(e)) appears in any combination of the dimensions (D1-D3) of the problem and it always distinguishes the hierarchical evaluation measures regarding the way they deal with it.

3.2.2 Pair-based Measures

Pair-based measures assign costs to pairs of predicted and true classes. For example, in Figure 3.2(d) class `B` could be paired with `A` and class `C` with `D`, and then the sum of the corresponding costs would give the total misclassification error.

Let $\hat{Y} = \{\hat{y}_i | i = 1 \dots M\}$ and $Y = \{y_j | j = 1 \dots N\}$ be the sets of the predicted and true classes respectively, for a single test instance (the index of the instance is omitted for simplicity). The sets Y and $\hat{Y}$ are augmented with a default predicted and a default true class, respectively, corresponding to $\hat{y}_{M+1}$ and y_{N+1}. These classes are used when a predicted class cannot or should not be paired to any true class and vice-versa. For example, when the distances between a predicted class $\hat{y}_i$ and all the true classes y_i exceed a predefined threshold (see the long distance problem in Figure 3.2(e)), the predicted class $\hat{y}_i$ may be paired with the default true class.

Additionally, let κ_{ij} be the cost of predicting class $\hat{y}_i$ instead of the true class y_j. The matrix $\mathbf{K} = [\kappa_{ij}]_{i=1\dots M+1, j=1\dots N+1}$, $\kappa_{ij} \geq 0, \forall i, j$ contains the costs of all possible pairs of predicted and true classes, including the default classes.

Pair-based measures typically calculate the cost κ_{ij} of a pair of a predicted class $\hat{y}_i$ and a true class y_i as the minimum distance of $\hat{y}_i$ and y_i in the hierarchy, e.g. as the number of edges between the classes along the shortest path that connects them. The

intuition is that the closer the two classes are in the hierarchy, the more similar they are, and therefore the less severe the error. More elaborate cost measures may assign weights to the edges of the hierarchy and the weights may decrease when moving from the top to the bottom (Blockeel et al., 2002; Holden and Freitas, 2006). The distance to the default classes is usually set to a fixed large value.

In a spirit of fairness (minimum penalty), the aim of an evaluation measure is to pair the classes returned by a system and the true classes in a way that minimizes the overall classification error. This can be formulated as the following optimization problem:

Problem 1.

$$\begin{cases} \min\limits_{\substack{x_{ij},\\ 1\leq i\leq M+1,\\ 1\leq j\leq N+1}} \sum\limits_{\substack{i=1\ldots(N+1),\\ j=1\ldots(M+1)}} \kappa_{ij}x_{ij} \\ \textit{subject to:} \\ \quad \textit{(i)}\ \forall i = 1\ldots M, \forall j = 1\ldots N, x_{ij} \in \{0;1\};\ x_{(M+1)(N+1)} = 0 \\ \quad \textit{(ii)}\ \alpha_p \leq \sum_{j=1}^{N+1} x_{ij} \leq \beta_p, \forall i = 1\ldots M \\ \quad \textit{(iii)}\ \alpha_t \leq \sum_{i=1}^{M+1} x_{ij} \leq \beta_t, \forall j = 1\ldots N \end{cases}$$

The intuition behind this sum is that each predicted node must be matched with a true one (x_{ij}) or with a default one, with a certain cost (κ_{ij}) and vice versa. We want to find the matches ($\kappa_{ij}x_{ij}$) with the minimum sum of costs given certain constraints. Constraint (i) states that x_{ij}, which denotes the alignment between classes, is either 0 (classes $\hat{y}_i$ and y_j are not paired) or 1 (classes $\hat{y}_i$ and y_j are paired); it furthermore states that the default predicted and true classes cannot be aligned (these default classes are solely used to "collect" those predicted and true classes with no counterpart). The parameters α_p, $\beta_p \in \mathbb{N}$ (constraint (ii)) are the lower and upper bounds of the allowed number of true classes that a predicted class can be paired with. For example, setting $\alpha_p = \beta_p = 1$ requires each predicted class to be paired with exactly one true class. Similarly, the parameters α_t, $\beta_t \in \mathbb{N}$ (constraint (iii)) limit the number of predicted labels that a true class can be paired with. The above constraints directly imply[3] that

[3]Indeed, in the worst case, i.e. when all x_{ij} but x_{iN+1} are 0, constraint (ii) yields $x_{iN+1} = \beta_p$; the

$\forall i = 1 \ldots M, x_{iN+1} \leq \beta_p$ and $\forall j = 1 \ldots N, x_{M+1j} \leq \beta_t$, meaning that the default true class can be aligned to at most $\beta_p M$ predicted classes and the default predicted class to at most $\beta_t N$ true classes.

The above problem corresponds to a best pairing problem in a bipartite graph, the nodes of which being respectively the predicted and true classes. It is important to note here that the pairing we are looking for is not a 1-1 matching, since the same node can be paired with several nodes. We opt to approach this problem through graph pairing rather than linear optimization, for two reasons: first because there exist polynomial solutions to pairing problems in graphs, and second because the graph framework allows one to easily illustrate how the different cost-based measures proposed so far relate to each other. In particular, we model it as a cost flow minimization problem (Ahuja et al., 1993).

3.2.2.1 A Flow Network Model for Class Pairing

A flow network is a directed graph $G = (V, E)$ with m edges, where each edge $u \in E$ is associated with a lower and an upper capacity denoted b_u and c_u respectively. The flow along an edge u is denoted as ϕ_u and $b_u \leq \phi_u \leq c_u$. The flow of the network is a vector $\phi = (\phi_1, \phi_2, \ldots, \phi_m)^T \in \mathbb{R}^m$. For each vertex $i \in V$, the flow conservation property holds:

$$\sum_{u \in \omega^+(i)} \phi_u = \sum_{u \in \omega^-(i)} \phi_u$$

where $\omega^+(i)$ and $\omega^-(i)$ denote the set of edges entering and leaving vertex i respectively. Each edge u is also associated with a cost γ_u which represents the cost of using this edge. The total cost of a flow ϕ is:

$$\gamma^T \phi = \sum_{u \in E} \gamma_u \phi_u,$$

where $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_m)^T \in \mathbb{R}^m$. The minimum cost flow is the one that minimizes $\gamma^T \phi$ while satisfying the capacity and flow conservation constraints. The quantity to be

reasoning is similar for constraint (iii).

minimized in flow networks is the same as in Problem 1, the constraints in this problem corresponding to capacity constraints, as explained below. Furthermore, the following integrality theorem (Ahuja et al., 1993) states that when the bounds of capacity intervals are integers, there exists a minimal cost flow such that the quantity of flow on each edge is also an integer:

Integrality Theorem. *If a flow network has capacities which are all integer valued and there exists some feasible flow in the network, then there is a minimum cost feasible flow with an integer valued flow on every arc.*

All standard algorithms for finding minimal cost flows guarantee finding this particular flow (Ahuja et al., 1993).

Pairing problems in bipartite graphs are represented with flow networks by adding two nodes, a source and a sink, and edges from the source to the first set of nodes, from the second set of nodes to the sink, and from the sink to the source. These extra nodes and edges ensure that the flow conservation constraints are satisfied. For pair-based measures, one thus obtains the following flow network framework $G(V,E)$ (see also Figure 3.3):

- V includes a source, a sink, the predicted classes, the true classes, a default true class and a default predicted class;
- E includes edges from the source to all the predicted classes (including the default predicted class), from every predicted class to every true class (including the default true class), from every true class to the sink and from the sink to the source.

No edges exist between the default predicted and default true class, as required by constraint (i) above.

In our setting, the capacity interval $[b_u; c_u]$ of an edge u expresses the possible number of pairs that each predicted or true class can participate in. The interval between

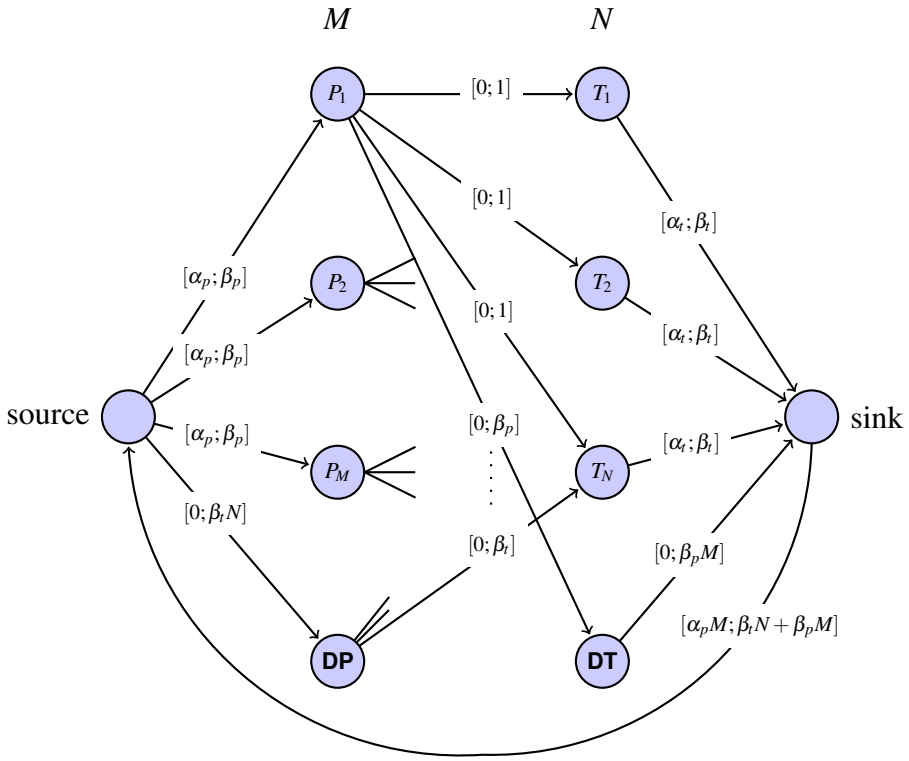


Figure 3.3: The proposed flow network.

each pair of predicted and true classes restricts the flow on that edge, which indicates whether this pair will be considered in the calculation of the evaluation measure. Put it differently, in the solved flow network the flow values will reflect the specific evaluation measure as they show the pairs that make up the solution with the minimum cost. The intervals between the source and the predicted classes, as well as between the true classes and the sink also affect the way that the pairing will be performed.

Due to the constraints in Problem 1, the capacity intervals are defined as follows:

- From each predicted class $\hat{y_j}$ to each true class y_i, excluding the default class, the capacity interval is [0;1]; the integrality theorem here implies that the flow value between predicted and true classes will be either 0 or 1, i.e. a predicted and a true class either be paired (1) or not paired (0). The capacity bounds here correspond

to the x_{ij} values of Problem 1 (constraint (i)).

- From the source to a (non-default) predicted class, the capacity interval is $[\alpha_p;\beta_p]$ meaning that a predicted class is aligned with at least α_p (and at most β_p) true classes.

- Similarly, from a (non-default) true class to the sink, the capacity intervals is $[\alpha_t;\beta_t]$ meaning that a true class is aligned with at least α_t (and at most β_t) predicted classes.

- From each predicted class $\hat{y_j}$ to the default true class the capacity interval is $[0;\beta_p]$ and from the default predicted class to each true class the capacity interval is $[0;\beta_t]$; from the source (resp. sink) to the default predicted (resp. true) class, the capacity interval is $[0;\beta_t N]$ (resp. $[0;\beta_p M]$), as mentioned in footnote 3.

- Lastly, from the sink to the source, the capacity interval is $[\alpha_p M;\beta_t N+\beta_p M]$, which corresponds to a loose setting compatible with the intervals given above (this last capacity interval does not impose any constraint but is necessary to ensure flow conservation).

3.2.2.2 Existing Pair-based Measures

The majority of the existing pair-based measures deals only with tree hierarchies and single-label problems. Under these conditions the pairing problem becomes simple, because a single path exists between the predicted and the true classes. The complexity of the problem increases when the hierarchy is a DAG or when the problem is multi-labeled; current measures cannot handle the majority of the phenomena presented in Section 3.2.1.

In the simplest case of pair-based measures (Dekel et al., 2004; Holden and Freitas, 2006), the measure trivially pairs the single prediction with the single true label ($M =$

$N = 1$), so that $\alpha_p = \beta_p = \alpha_t = \beta_t = 1$. Note that no default classes exist in this measure, or equivalently the corresponding costs are equal to infinity, $\gamma(DP,T) = \gamma(DT,P) = +\infty$

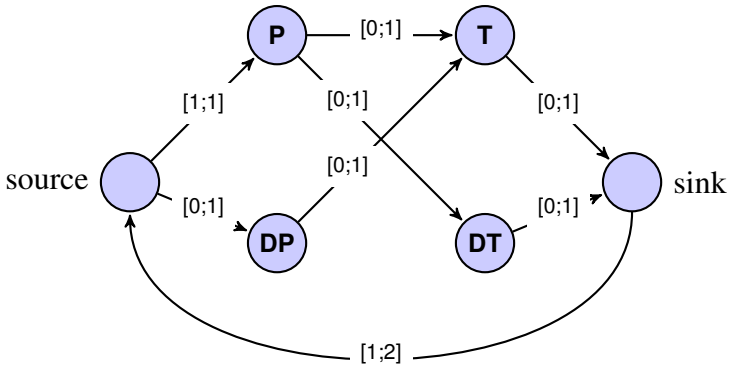


Figure 3.4: Tree Induced Error flow network

For a pair $(\hat{y}_j, y_i)$, of a predicted and a true class, depicted as P and T respectively in Figure 3.4, $\gamma(\hat{y}_j, y_i) = \gamma(P,T) = \kappa_{ij}$ is taken to be the distance between y_i and $\hat{y}_j$:

$$\kappa_{ij} = \sum_{e \in E(i,j)} w_e, \tag{3.1}$$

where $E(i,j)$ is the set of edges along the path from y_i to $\hat{y}_j$ in the hierarchy and w_e is the weight of edge e. For $w_e = 1$, we get what Dekel et al. (2004) call *tree induced error*.

Sun and Lim (2001) propose two cost measures for multi-label problems in tree hierarchies, where all possible pairs of the predicted and true classes are used in the calculation. In this case, $\alpha_p = \beta_p = N$ and $\alpha_t = \beta_t = M$. Again, no default classes are used and so the corresponding costs are: $\gamma(DP, y_i) = \gamma(DT, \hat{y}_j) = +\infty,\ i = 1 \ldots N, j = 1 \ldots M$. Note that this is an extreme case, where all pairs of predicted and true labels are used. The weights w_e are calculated in two alternative ways: a) as the similarity (e.g., cosine similarity) between the classes of the predicted and true labels, and b) using the distances of the hierarchy as in Equation 3.1.

A measure dubbed Graph Induced Error (GIE) was proposed and used during the second Large Scale Hierarchical Text classification challenge (LSHTC)[4]. GIE is based on the best matching pairs of predicted and true classes and can handle multi-label (and single-labeled) classification with both tree and DAG class hierarchies. For a particular instance being classified, each predicted class is paired either with one true class or with the default true class; multiple predicted classes can be paired with the default true class (Figure 3.5). Similarly, each true class is paired with exactly one predicted class or with the default predicted class, and several true classes can be paired with the default predicted class. Hence, $\alpha_p = \beta_p = \alpha_t = \beta_t = 1$. The cost κ_{ij} is computed as in Equation 3.1, with $w_e = 1, \forall e$. If the hierarchy is a DAG, multiple hierarchy paths may link each predicted class $\hat{y_j}$ to its paired true class y_i; then $E(i, j)$ is taken to be the shortest of these paths. The cost of pairing a class (predicted or true) with a default one is set to a positive value $D_{\max}$. Figure 3.5 presents the corresponding flow network.

In multi-label classification, GIE's concept of "best" matching fails to address the pairing problem of Section 3.2.1. For example, if a predicted class has two true classes as children, as in Figure 3.2(d), then only one of them would be paired with its parent. The other one would either be penalized with $D_{\max}$ or would be paired with another distant class.

3.2.2.3 Multi-label Graph Induced Accuracy

We propose here a straightforward extension of GIE called Multi-label Graph Induced Accuracy (MGIA), in which each class is allowed to participate in more than one pair. This extension makes the method more suitable to the pairing problem. Figure 3.6 presents the MGIA flow network, in which $\alpha_p = \alpha_t = 1$, $\beta_p = N$, $\beta_t = M$. The cost of pairing a class (predicted or true) with a default one is set as in GIE. Solving the flow network optimization problem is easy since the only constraints are that the default

[4] http://lshtc.iit.demokritos.gr/

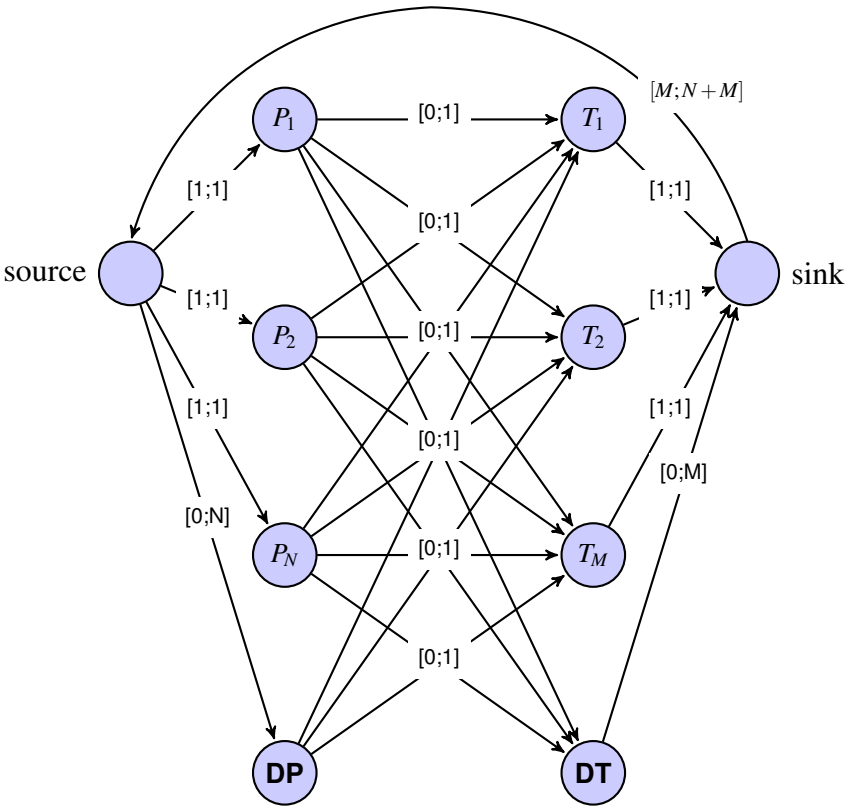


Figure 3.5: Graph Induced Error (GIE) flow network.

predicted class cannot be paired with the default true class and that categories of the same set (predicted or true) cannot be paired to each other. Thus each pairing can be solved separately from the others by pairing a class with either the default class of the other set, or the nearest class of the other set.

As in the previous pair-based measures, once the optimization problem is solved, an error is calculated on the solved network. Instead of using directly this error for evaluation we define an accuracy-based measure as follows:

$$1 - \frac{fnerror}{|(P \cup T) \setminus (P \cap T)| \cdot D_{max}}$$

where $fnerror$ is the value provided by the solved flow network.

The above measure is bounded in [0,1] and the better a system is the closer it will be to 1. Note that in the case where all predicted classes and all true classes are paired

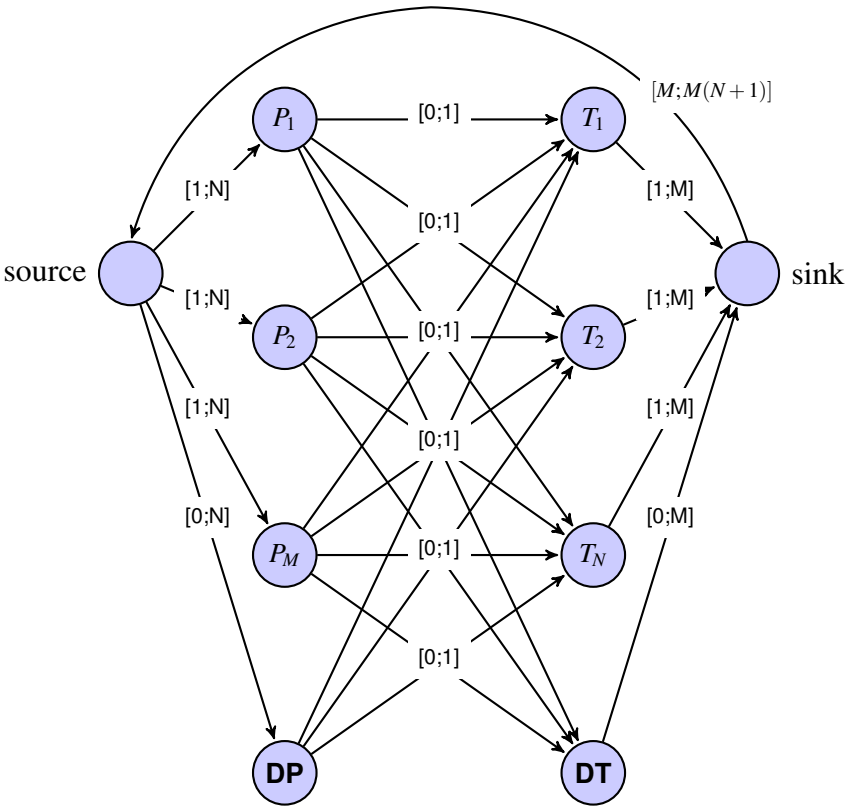


Figure 3.6: Multi-label Graph Induced Accuracy flow network.

with the respective default classes, $fnerror$ will reach its maximum value $|P \cup T| \cdot D_{max}$ and will be equal to the denominator as $P \cap T = \emptyset$ resulting in an accuracy value of 0. Essentially, the advantage of the proposed measure over other pair-based measures is that it takes into account the correct predictions of the classification system (that is the true positives, $P \cap T$). This is important because given two systems that make exactly the same errors, while the first makes an extra correct prediction, the first is better than the second. But by measuring only the error these two systems would be evaluated as performing equally well.

3.2.3 Set-based Measures

The performance measures of this category are based on operations on the entire sets of predicted and true classes, possibly including also their ancestors and descendants, as opposed to pair-based measures, which consider only pairs of predicted and true classes.

Set-based measures have two distinct phases:

1. The augmentation of Y and $\hat{Y}$ with information about the hierarchy.

2. The calculation of a cost measure based on the augmented sets.

The augmentation of Y and $\hat{Y}$ is a crucial step, attempting to capture the hierarchical relations of the classes. For example, the sets may be augmented with the ancestors of the true and predicted classes as follows:

$$Y_{aug} = Y \cup An(y_1) \cup \ldots \cup An(y_N) \tag{3.2}$$

$$\hat{Y}_{aug} = \hat{Y} \cup An(\hat{y}_1) \cup \ldots \cup An(\hat{y}_M) \tag{3.3}$$

Using the augmented sets of predicted and true classes, two approaches have mainly been adopted to calculate the misclassification cost: a) symmetric difference loss and b) hierarchical precision and recall.

Symmetric difference loss is calculated as:

$$l_{\Delta}(Y_{aug}, \hat{Y}_{aug}) = |(\hat{Y}_{aug} \setminus Y_{aug}) \cup (Y_{aug} \setminus \hat{Y}_{aug})|$$

where $|S|$ the cardinality of a set S. If we use the initial $\hat{Y}$ and Y sets instead of $\hat{Y}_{aug}$ and Y_{aug}, the measure becomes the standard symmetric difference for flat multi-label classification. Also, note that the two quantities combined in the formula of the symmetric difference loss express the false positive and false negative rates respectively.

On the other hand, hierarchical precision and recall are defined as follows:

$$P_H = \frac{|\hat{Y}_{aug} \cap Y_{aug}|}{|\hat{Y}_{aug}|}$$

$$R_H = \frac{|\hat{Y}_{aug} \cap Y_{aug}|}{|Y_{aug}|}$$

The nominator of these measures expresses the true positive rate and can be written as follows:

$$|\hat{Y}_{aug} \cap Y_{aug}| = |\hat{Y}_{aug} \cup Y_{aug}| - l_{\Delta}(Y_{aug}, \hat{Y}_{aug})$$

where we note that the symmetric loss is a subtractive term.

Set-based measures are not affected by the pairing problem of Figure 3.2(d) and the long distance problem of Figure 3.2(e), as they do not rely on pairing of true and predicted classes.

3.2.3.1 Existing Set-based Measures

Different measures differ mainly in the way the sets of predicted and true classes are augmented. In (Kiritchenko et al., 2005; Struyf et al., 2005; Cai and Hofmann, 2007) the ancestors of the predicted and true classes are added to Y_{aug} and $\hat{Y}_{aug}$, as in Equations 3.2 and 3.3 above. Alternatively, in (Ipeirotis et al., 2001) the descendants of the true and predicted classes are added:

$$Y_{aug} = Y \cup De(y_1) \cup \ldots \cup De(y_N)$$

$$\hat{Y}_{aug} = \hat{Y} \cup De(\hat{y}_1) \cup \ldots \cup De(\hat{y}_M)$$

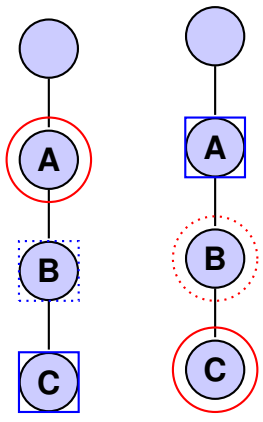
In the latter approach, when the true and predicted classes are in different sub-graphs of the hierarchy (different sub-trees, if the hierarchy is a tree), a maximum penalty will be given, even when several ancestors have been correctly predicted.

In (Cesa-Bianchi et al., 2006), the approach that adds the ancestors is adopted (Equations 3.2 and 3.3) but the augmented sets are then altered as follows:

$$Y_{aug} \leftarrow Y_{aug} \setminus \left\{ y_k \;\middle|\; y_k \in Y_{aug} \wedge y_k \notin \hat{Y}_{aug} \wedge Pa(y_k) \notin \hat{Y}_{aug} \right\} \tag{3.4}$$

$$\hat{Y}_{aug} \leftarrow \hat{Y}_{aug} \setminus \left\{ \hat{y_k} \;\middle|\; \hat{y_k} \in \hat{Y}_{aug} \wedge \hat{y_k} \notin Y_{aug} \wedge Pa(\hat{y_k}) \notin Y_{aug} \right\} \tag{3.5}$$

Equation 3.5 introduces some tolerance to over-specialization. Consider, for example, Figure 3.7(a) where we assume that the only true class is A and the only predicted class is C. According to Equation 3.3 we add class B (and class A) to $\hat{Y}_{aug}$. Based on Equation 3.5 we then remove C from $\hat{Y}_{aug}$ to avoid penalizing the classification method for B and C. Similarly, with Equation 3.4 we tolerate under-specialization. In Figure 3.7(b) the only true class is C and the only predicted class is A. According to Equation 3.2 class B (and A) are added to Y_{aug}. Based on Equation 3.4 then we remove C from $\hat{Y}_{aug}$ to avoid penalizing the classification method for both B and C. The drawback of this measure is that it tends to favor systems that stop their predictions early in the hierarchy.



(a) (b)

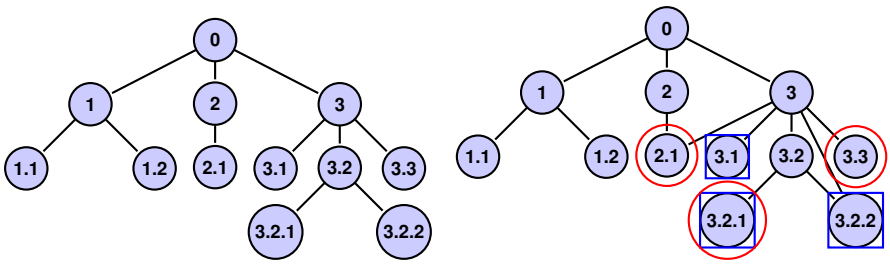
Figure 3.7: Tolerating over-specialization and under-specialization.

3.2.3.2 Lowest Common Ancestor Precision, Recall and F_1 Measures

The approach proposed in this thesis is based on the hierarchical versions of precision, recall and F_1, which add all the ancestors of the predicted and true classes to Y_{aug} and $\hat{Y}_{aug}$. Adding all the ancestors has the undesirable effect of over-penalizing errors that happen to nodes with many ancestors.

In an attempt to address this issue, we propose the Lowest Common Ancestor Precision (P_{LCA}), Recall (R_{LCA}) and F_1 (F_{LCA}) measures. These measures use the concept of the lowest common ancestor (LCA) as defined in graph theory (Aho et al., 1973).

Definition 1. *The lowest common ancestor $LCA(n_1, n_2)$ of two nodes n_1 and n_2 of a tree T is defined as the lowest node in T (furthest from the root) that is an ancestor of both n_1 and n_2.*



(a) Tree (b) DAG

Figure 3.8: Tree and DAG hierarchies. Nodes surrounded by circles are true classes. Nodes surrounded by rectangles are predicted classes.

For example, in Figure 3.8(a) $LCA(3.1, 3.2.2) = 3$. In the case of a DAG the definition of LCA changes. $LCA(n_1, n_2)$ is a set of nodes (instead of a single node), since it is possible for two nodes to have more than one LCA. Furthermore, the LCA may not necessarily be the node that is furthest from the root. In order to define the LCA between two DAG nodes, we use the concept of the shortest path between them.

Definition 2. *Given a set $P_{all}(n_1, n_2)$ containing all paths that connect nodes n_1 and n_2,*

we define $paths_{min}(n_1, n_2) \subseteq P_{all}(n_1, n_2)$ *as the set for which:*
$\forall p \in paths_{min}(n_1, n_2); \nexists p' \in P_{all}(n_1, n_2) \setminus paths_{min}(n_1, n_2) : cost(p) \leq cost(p')$

where the cost of a path corresponds to its length, when the edges of the hierarchy are unweighted.

For example in Figure 3.8(b):

- $path_{min}(2.1, 3.1) = \{2.1, 3, 3.1\}$
- $path_{min}(2.1, 3.2.2) = \{2.1, 3, 3.2.2\}$
- $path_{min}(3.2.2, 3.2.1) = \{3.2.2, 3.2, 3.2.1\}$

It is worth noting that in the general case $paths_{min}(n_1, n_2)$ is a set of paths; not a single one.

In multi-label classification, we would like to extend the definition of *LCA* to compare a node n (e.g. a true class) against a set of nodes S (e.g. the predicted classes).

Definition 3. *The* $LCA(n, S)$ *of a node n and a set of nodes S is the set of all the lowest common ancestors* $LCA(n, i)$ *for each* $i \in S_{best}(n, S) \subseteq S$*, where* $S_{best}(n, S)$ *is a subset of all the lowest common ancestors, which are the closest to node n (*$S_{best}(n, S) = \{i \in S : \nexists j \in S,\ j \neq i \wedge cost(path_{min}(n, i)) > cost(path_{min}(n, j))\}$*)*

For example, in Figure 3.8(b) $S_{best}(3.1, \{2.1, 3.3, 3.2.1\}) = \{2.1, 3.3\}$ and $LCA(3.1, \{2.1, 3.3, 3.2.1\})$ is $\{3\}$.

Given this definition and sets, Y being the true and $\hat{Y}$ the predicted classes of an instance, we compute the $LCA(y, \hat{Y})$ of each element y of Y. Similarly for each element $\hat{y}$ of $\hat{Y}$ we compute $LCA(\hat{y}, Y)$. Using Figure 3.8(b), let Y = {2.1, 3.2.1, 3.3} and $\hat{Y}$ = {3.1, 3.2.1, 3.2.2}. Then:

- $LCA(2.1, \hat{Y}) = \{3\}$, connecting 2.1 with either 3.1 using $path_{min}(2.1, 3.1)$ or 3.2.2 using $path_{min}(2.1, 3.2.2)$.

- $LCA(3.3, \hat{Y}) = \{3\}$, connecting 3.3 with either 3.1 using $path_{min}(3.3, 3.1)$ or 3.2.2 using $path_{min}(3.3, 3.2.2)$.
- $LCA(3.2.1, \hat{Y}) = \{3.2.1\}$, connecting 3.2.1 with itself.
- $LCA(3.2.1, Y) = \{3.2.1\}$, connecting 3.2.1 with itself.
- $LCA(3.1, Y) = \{3\}$, connecting 3.1 with either 2.1 using $path_{min}(3.1, 2.1)$ or 3.3 using $path_{min}(3.1, 3.3)$.
- $LCA(3.2.2, Y) = \{3.2, 3\}$, the first connecting 3.2.2 with 3.2.1 using $path_{min}(3.2.2, 3.2.1)$ and the second connecting 3.2.2 with either 2.1 using $path_{min}(3.2.2, 2.1)$ or 3.3 using $path_{min}(3.2.2, 3.3)$.

Additionally, we are interested in the sets containing all the *LCAs* of each of the two sets.

Definition 4. *Given a set of true classes (nodes) Y and a set of predicted classes (nodes) $\hat{Y}$, we define $LCA_{all}(Y, \hat{Y})$ as the set containing all LCA(y,$\hat{Y}$) for all $y \in Y$. Similarly we define $LCA_{all}(\hat{Y}, Y)$ as the set containing all LCA($\hat{y}$,Y) for all $\hat{y} \in \hat{Y}$.*

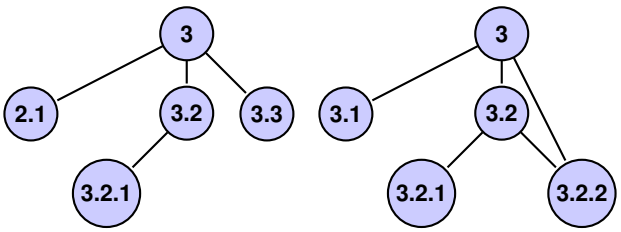
In the above example $LCA_{all}(Y, \hat{Y})$ ={3, 3.2.1}, $LCA_{all}(\hat{Y}, Y)$ ={3, 3.2, 3.2.1}.

Definition 5. *Given a set of true classes (nodes) Y, a set of predicted classes (nodes) $\hat{Y}$ and a set of LCA_{all}, we define $G_t^{ex}(Y, \hat{Y})$ as the graph that contains:*

- *all $paths_{min}(y, a)$: $y \in Y \wedge a \in LCA(y, \hat{Y})$*
- *all $paths_{min}(y, a)$ subpaths of $paths_{min}(\hat{y}, y) : \hat{y} \in \hat{Y} \wedge y \in S_{best}(\hat{y}, Y) \wedge a \in LCA(\hat{y}, Y)$*

Similarly $G_p^{ex}(Y, \hat{Y})$ is the graph that contains:

- *all $paths_{min}(\hat{y}, b)$: $\hat{y} \in \hat{Y} \wedge b \in LCA(\hat{y}, Y)$*
- *all $paths_{min}(\hat{y}, b)$ subpaths of $paths_{min}(y, \hat{y}) : y \in Y \wedge \hat{y} \in S_{best}(y, \hat{Y}) \wedge b \in LCA(y, \hat{Y})$*



(a) $G_t^{ex}(Y,\hat{Y})$ (b) $G_p^{ex}(Y,\hat{Y})$

Figure 3.9: $G_t^{ex}(Y,\hat{Y})$ and $G_p^{ex}(Y,\hat{Y})$ augment the sets of the true and the predicted classes with *LCAs*.

For example, for the Y and $\hat{Y}$ of Figure 3.8(b) we get the $G_t^{ex}(Y,\hat{Y})$ and $G_p^{ex}(Y,\hat{Y})$ graphs of Figure 3.9(a) and 3.9(b), respectively.

Based on these graphs the true and predicted sets of classes are augmented, in order for the set-based measures to be calculated. In the case of Figure 3.9, $Y_{aug} = \{3, 2.1, 3.2, 3.3, 3.2.1\}$ and $\hat{Y}_{aug} = \{3, 3.1, 3.2, 3.2.1, 3.2.2\}$. The next step is to calculate cost measures based on these two sets which in our case are the following:

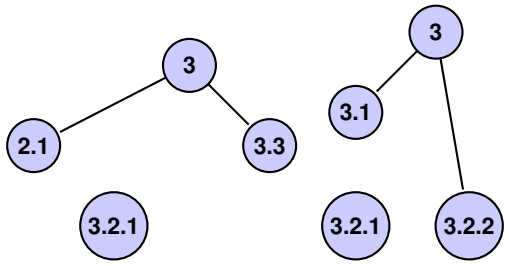
$$P_{LCA} = \frac{|\hat{Y}_{aug} \cap Y_{aug}|}{|\hat{Y}_{aug}|}$$

$$R_{LCA} = \frac{|\hat{Y}_{aug} \cap Y_{aug}|}{|Y_{aug}|}$$

$$F_{LCA} = \frac{2P_{LCA}R_{LCA}}{P_{LCA} + R_{LCA}}$$

In the example of Figure 3.9 all three measures, P_{LCA}, R_{LCA} and F_{LCA}, between sets Y_{aug} and $\hat{Y}_{aug}$, are 0.6. We prefer this approach over the symmetric difference loss, since it takes into account the True Positives (TP) in addition to False Positives (FP) and False Negatives (FN). Ignoring TP leads systems to prefer predicting fewer categories, since missing a single FP usually costs more than the gain of finding an extra TP. This behavior is also observed in the results of real systems, (see Section 4) and is considered undesirable.

The two graphs $G_t^{ex}(Y,\hat{Y})$ and $G_p^{ex}(Y,\hat{Y})$ were created using all nodes of $LCA_{all}(Y,\hat{Y})$ and $LCA_{all}(\hat{Y},Y)$ and all corresponding paths. However, subgraphs of the two graphs $G_t(Y,\hat{Y}) \subseteq G_t^{ex}(Y,\hat{Y})$ and $G_p(Y,\hat{Y}) \subseteq G_p^{ex}(Y,\hat{Y})$ could be selected that would connect each node of $Y \cup \hat{Y}$ with an *LCA* and vice versa. For example, in Figure 3.8(b) node 3.2.2 has two *LCAs*, nodes 3.2 and 3. Node 3.2 could be removed from $G_t^{ex}(Y,\hat{Y})$ and $G_p^{ex}(Y,\hat{Y})$, without leaving any node in $Y \cup \hat{Y}$ without an *LCA*. We would then get the reduced graphs $G_t(Y,\hat{Y})$ and $G_p(Y,\hat{Y})$ of Figure 3.10. P_{LCA}, R_{LCA} and F_{LCA}, between the reduced sets Y_{aug} and $\hat{Y}_{aug}$ of Figure 3.10, are 0.5 instead of 0.6 (Figure 3.9).



(a) $G_t(Y,\hat{Y})$ (b) $G_p(Y,\hat{Y})$

Figure 3.10: $G_t(Y,\hat{Y})$ and $G_p(Y,\hat{Y})$ are the minimal graphs augmenting the sets of the true and the predicted classes with *LCAs*.

In other words graphs $G_t(Y,\hat{Y})$ and $G_p(Y,\hat{Y})$ should comprise the nodes necessary for connecting the nodes of the two sets, through their *LCAs*. Redundant nodes can lead to fluctuations in P_{LCA}, R_{LCA} and F_{LCA}, and should be removed. In order to obtain the minimal *LCA* graphs, we have to solve the following maximization problem:

Problem 2. *Minimal LCA graph extension.*

$$
\begin{cases}
\underset{\substack{(G_t(Y,\hat{Y})\subseteq G_t^{ex}(Y,\hat{Y}),\\ G_p(Y,\hat{Y})\subseteq G_p^{ex}(Y,\hat{Y}))}}{\text{argmax}} \quad F_{LCA}(G_t(Y,\hat{Y}),G_p(Y,\hat{Y})) \\
\textit{subject to:} \\
\quad \textit{(i)}\ \forall y \in Y; y \in G_t(Y,\hat{Y})\ \textit{and}\ \forall \hat{y} \in \hat{Y}; \hat{y} \in G_p(Y,\hat{Y}) \\
\quad \textit{(ii)}\ \forall y \in Y; \exists a : a \in G_t(Y,\hat{Y}) \wedge a \in G_p(Y,\hat{Y}) \wedge a \in LCA(y,\hat{Y}) \\
\qquad \forall \hat{y} \in \hat{Y}; \exists b : b \in G_t(Y,\hat{Y}) \wedge b \in G_p(Y,\hat{Y}) \wedge b \in LCA(\hat{y},Y) \\
\quad \textit{(iii)}\ \nexists G'_t(Y,\hat{Y})\ \textit{subject to constraints (i) and (ii)}: \\
\qquad\qquad |\{a : a \in G_t(Y,\hat{Y}) \wedge a \in LCA_{all}(Y,\hat{Y})\}| > \\
\qquad\qquad |\{a' : a' \in G'_t(Y,\hat{Y}) \wedge a' \in LCA_{all}(\hat{Y},Y)\}| \\
\qquad \nexists G'_p(Y,\hat{Y})\ \textit{subject to constraints (i) and (ii)}: \\
\qquad\qquad |\{b : b \in G_p(Y,\hat{Y}) \wedge b \in LCA_{all}(Y,\hat{Y})\}| > \\
\qquad\qquad |\{b' : b' \in G'_p(Y,\hat{Y}) \wedge b' \in LCA_{all}(\hat{Y},Y)\}| \\
\quad \textit{(iv)}\ \forall y \in Y; \exists p_y \in G_t(Y,\hat{Y}) : p_y \in paths_{min}(y,a) \wedge a \in LCA_{all}(Y,\hat{Y}) \\
\qquad \forall \hat{y} \in \hat{Y}; \exists p_{\hat{y}} \in G_p(Y,\hat{Y}) : p_{\hat{y}} \in paths_{min}(\hat{y},b) \wedge b \in LCA_{all}(\hat{Y},Y) \\
\quad \textit{(v)}\ \forall a \in G_t(Y,\hat{Y}) \cap LCA_{all}(\hat{Y},Y); \exists p_y \in G_t(Y,\hat{Y}) : \\
\qquad\qquad p_y \in paths_{min}(y,a) \wedge y \in Y \\
\qquad \forall b \in G_p(Y,\hat{Y}) \cap LCA_{all}(Y,\hat{Y}); \exists p_{\hat{y}} \in G_p(Y,\hat{Y}) : \\
\qquad\qquad p_{\hat{y}} \in path_{min}(\hat{y},b) \wedge \hat{y} \in \hat{Y}
\end{cases}
$$

The maximization of $F_{LCA}(G_t(Y,\hat{Y}),G_p(Y,\hat{Y}))$ is subject to a set of constraints: Constraint (i) requires all class nodes of an initial set (Y or $\hat{Y}$) to be included in the final subgraphs. Constraint (ii) enforces the existence of at least one *LCA* for each node of $Y \cup \hat{Y}$ in the subgraphs. Constraint (iii) limits the total number of *LCAs* used to the minimum required in order to be able to satisfy constraints (i) and (ii). Constraint (iv) implies the existence of at least one path connecting each class node of each subgraph to one of its LCAs, while constraint (v) implies the inverse, i.e. that each *LCA* of the subgraphs is connected with at least one class node of each subgraph.

One way to solve this maximization problem would be to create all possible $G_t(Y,\hat{Y})$ and $G_p(Y,\hat{Y})$ graphs that respect the above constraints and choose the ones leading to the highest F_{LCA}. This procedure is very computationally expensive and for this reason we devised the approximation presented in Algorithm 1.

The main procedure is decomposed into three subprocedures. The first procedure, GetBestLCAs, returns an approximation of the minimum number of *LCAs* needed in or-

der to satisfy constraints (i), (ii) and (iii) of the maximization problem. This is achieved by initially sorting, in descending order, all *LCAs* by the number of Y and $\hat{Y}$ that they connect. On this list, we perform two passes, first top-down and then bottom-up removing all redundant *LCAs*, i.e., *LCAs* of nodes for which other *LCAs* are already included in the list. In the final step of the algorithm, GetBestPaths selects the minimum paths that satisfy constraints (iv) and (v). In case two or more paths exist that connect the same node with an *LCA*, we choose the one which leads to the smallest possible subgraphs.

An interesting issue arises when a class and one of its ancestors co-exist in the predicted or the true class sets. Assume for example a system A predicting that an instance belongs to node X, while another system B assigns it also to one of the ancestors of X. Each extra ancestor of X would lead to higher F_1 score since it would increase the size of the $Y_{aug} \cap \hat{Y}_{aug}$ set. This happens because all the ancestors of an $LCA(n_1, x_2)$ are also ancestors of nodes n_1 and n_2. We address this issue by removing from set Y any node y for which $\exists y' \in Y : y'$ is a descendant of y. We then do the same for set $\hat{Y}$ by removing each node $\hat{y}$ for which $\exists \hat{y'} \in \hat{Y} : \hat{y'}$ is a descendant of $\hat{y}$.

LCA precision, recall and F_1 are not purely set-based measures, but are actually in between pair-based and set-based measures. They are based on augmented sets of predicted and true nodes and calulate scores, based on the relation of those two sets. However the use of the $LCA(n, S)$ leads to a pairing of predicted and true nodes. So these measures could also be characterized as a hybrid, combining the advantages of both types of measure.

Algorithm 1 Approximation algorithm for computing $G_t(Y,\hat{Y})$ and $G_p(Y,\hat{Y})$, subgraphs of $G_t^{ex}(Y,\hat{Y})$ and $G_p^{ex}(Y,\hat{Y})$

procedure GETSUBGRAPHS($G_t^{ex}(Y,\hat{Y})$, $G_p^{ex}(Y,\hat{Y})$)
  $LCA_{all} \leftarrow getLCAsFrom(G_t^{ex}(Y,\hat{Y}), G_p^{ex}(Y,\hat{Y}))$
  $bestLCAs \leftarrow$ GETBESTLCAS($LCA_{all}, G_t^{ex}(Y,\hat{Y}), G_p^{ex}(Y,\hat{Y})$)
  $finalPaths \leftarrow$ GETBESTPATHS($bestLCAs, G_t^{ex}(Y,\hat{Y}), G_p^{ex}(Y,\hat{Y})$)
  $G_t(Y,\hat{Y}) \leftarrow$ all paths from $finalPaths$ containing a node $\in Y$
  $G_p(Y,\hat{Y}) \leftarrow$ all paths from $finalPaths$ containing a node $\in \hat{Y}$
end procedure
procedure GETBESTLCAS($LCAs$, G_t, G_p)
  $sortedLCAs \leftarrow sort(LCAs)$ ▷ sort LCAs, by the number of Y and $\hat{Y}$ that they connect, in descending order
  $bestLCAs \leftarrow \{\}$, $i \leftarrow 1$
  repeat
    $bestLCAs \leftarrow bestLCAs \cup sortedLCAs_i$
    $i \leftarrow i+1$
  until SATISFIED($bestLCAs$, $G_t(Y,\hat{Y})$, $G_p(Y,\hat{Y})$) ▷ procedure SATISFIED checks whether constraints (i), (ii) and (iii) of the optimization problem are satisfied
  for $i \leftarrow 1$ to $sizeof(bestLCAs)$ **do** ▷ top-down redundancy removal
    if SATISFIED($bestLCAs \setminus bestLCAs_i$, $G_t(Y,\hat{Y})$, $G_p(Y,\hat{Y})$) **then**
      $bestLCAs \leftarrow bestLCAs \setminus bestLCAs_i$
    end if
  end for
  for $i \leftarrow sizeof(bestLCAs)$ to 1 **do** ▷ bottom-up redundancy removal
    if SATISFIED($bestLCAs \setminus bestLCAs_i$, $G_t(Y,\hat{Y})$, $G_p(Y,\hat{Y})$) **then**
      $bestLCAs \leftarrow bestLCAs \setminus bestLCAs_i$
    end if
  end for
  return $bestLCAs$
end procedure
procedure GETBESTPATHS($bestLCAs$, $G_t(Y,\hat{Y})$, $G_p(Y,\hat{Y})$)
  $bestY \leftarrow \{\}$, $best\hat{Y} \leftarrow \{\}$
  for $i \leftarrow 1$ to $sizeof(bestLCAs)$ **do**
    for $j \leftarrow 1, sizeof(Y)$ **do** ▷ find nodes Y_j, for which $bestLCA_i$ is an LCA
      if $bestLCAs_i \in LCAs(Y_j,\hat{Y})$ **then**
        $bestY \leftarrow bestY \cap$ BESTPATH($bestLCAs_i, Y_j$) ▷ BestPath selects the path of $path_{min}(bestLCA_i, Y_j)$ that shares most common nodes with all other selected paths
      end if
    end for
    for $j \leftarrow 1$ to $sizeof(\hat{Y})$ **do**
      if $bestLCAs_i \in LCAs(\hat{Y}_j, Y)$ **then**
        $best\hat{Y} \leftarrow best\hat{Y} \cap$ BESTPATH($bestLCAs_i, \hat{Y}_j$)
      end if
    end for
  end for
  return $bestY \cup best\hat{Y}$
end procedure

3.2.4 Summary of Approaches

Table 3.2 summarizes the approaches described in Section 3.2 along three dimensions:

- Dimension 1: the problem can be either single-label (SL), where each instance is assigned to a single class label or multi-label (ML), where instances can be assigned to multiple class labels.
- Dimension 2: the graph structure can be either a tree (T) or a directed acyclic graph (DAG).
- Dimension 3: each instance is classified only to leaf class labels of the hierarchy (mandatory leaf node prediction (MLNP)) or to any class in the hierarhcy (non-MNLP).

Reference	Dimension 1	Dimension 2	Dimension 3
(Dekel et al., 2004)	SL	T	non-MLNP
(Holden and Freitas, 2006)	SL	T	non-MLNP
(Sun and Lim, 2001)	ML	T	non-MLNP
GIE	ML	DAG	non-MLNP
MGIA	ML	DAG	non-MLNP
(Kiritchenko et al., 2005)	ML	DAG	non-MLNP
(Cai and Hofmann, 2007)	ML	DAG	non-MLNP
(Struyf et al., 2005)	ML	DAG	non-MLNP
(Cai and Hofmann, 2007)	ML	DAG	non-MLNP
(Cesa-Bianchi et al., 2006)	ML	DAG	non-MLNP
LCA	ML	DAG	non-MLNP

Table 3.2: Summary of the different evaluation measures according to the three dimensions: a) single or multi-label problem, b) structure of the hierarchy and c) mandatory leaf node prediction problem or not. Alhtough a good evaluation measure should be able to deal with all three dimensions (ML, DAG and non-MLNP) and many measures are able to do that, we will show in the next section that this is not enough.

3.3 Case Studies

In this section we apply various measures to selected cases in order to demonstrate their pros and cons. As a representative of the pair-based measures we chose the Graph Induced Error (GIE), while for set-based ones we selected the hierarchical versions of precision (P_H), recall (R_H), F_1-measure ($F_H = \frac{2 \cdot P_H \cdot R_H}{P_H + R_H}$) and Symmetric Difference Loss ($l_\Delta(Y_{aug}, \hat{Y}_{aug})$), using all the ancestors of the predicted ($\hat{Y}$) and true (Y) labels in order to augment the sets of classes. We also use our proposed pair-based measure MGIA and the set-based *LCA* versions of precision (P_{LCA}), recall (R_{LCA}), F_1-measure (F_{LCA}) in order to illustrate their advantages and limitations, as well as the differences between the two types of measure.

Since GIE is an error and MGIA is an accuracy, for MGIA we also provide the $fnerror$ in order to facilitate their comparison.[5] All the above measures are implemented in a fast and easy to use tool written in C++ that is open source and available for download.[6] Like all pair-based methods, GIE and MGIA require a maximum distance threshold, above which nodes are paired with a default one. In the cases that we study here, this threshold is set to 5.

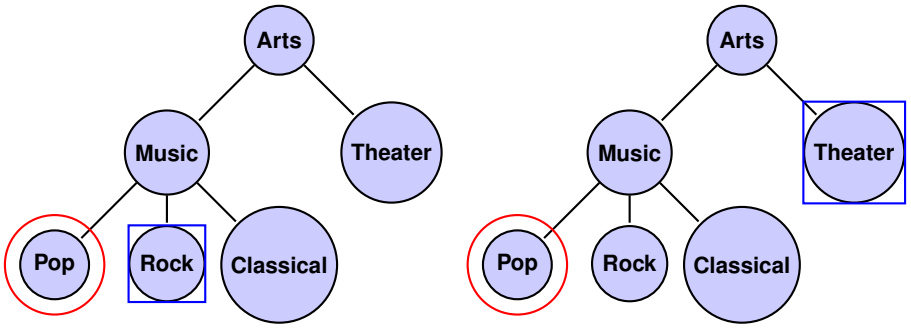
Based on the three dimensions of the HC problem presented in Section 2.1 and their subproblems presented in Section 2.2, which highlight important challenges in hierarchical evaluation, we present cases of specific problem settings. The list of cases here is not exhaustive, but it is sufficient to motivate the use of the proposed measures. Additionally, in Section 3.3.4 we present results on real datasets and classification systems.

[5]The reader is reminded that in pair-based measures each predicted node is matched with a true one (including the default ones) and vice versa. The length of the shortest path between these nodes is the error of that match and the sum of all these errors is $fnerror$.

[6]The tool is available from http://nlp.cs.aueb.gr/software_and_datasets/HEMKit.zip

3.3.1 Simplest problem setting

The simplest problem setting in HC is the one where each instance belongs to a single category (single-label classification), the hierarchy is a tree and classification is allowed only at the leaves. Figure 3.11(a) presents such a case, where the true category of an instance is *Pop*, while the predicted one is *Rock*. Figure 3.11(b) presents a similar case, where the predicted class is *Theater*, which is a more serious mistake. Table 3.3 presents the results of each measure for each of these two cases of the same setting.



(a) (b)

Figure 3.11: Simplest case of misclassification. Nodes surrounded by circles are the true classes while the nodes surrounded by rectangles are the predicted classes.

	GIE	MGIA	P_H	R_H	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	P_{LCA}	R_{LCA}	F_{LCA}
a	2	0.8(2)	0.66	0.66	0.66	2	0.5	0.5	0.5
b	3	0.7(3)	0.5	0.33	0.4	3	0.5	0.33	0.4

Table 3.3: Results per measure for Figure 3.11.

The first observation is that GIE behaves in the same desirable way as MGIA in such simple settings, since they both penalize the second case more than the first. This is also true for all the set-based measures (they all penalize the second case more than the first). The only interesting observation here is between the existing hierarchical measures of precision, recall, F_1 and the proposed *LCA* versions of them. In the first

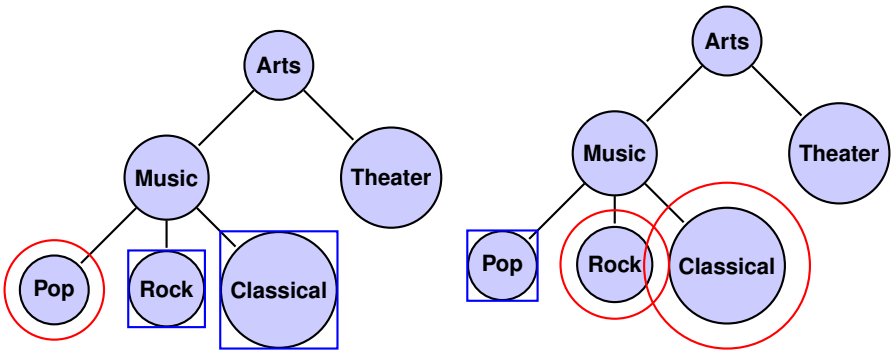
case, the *LCA* versions use in their calculations only node *Music*, which is the *LCA* of *Pop* and *Rock*, while the existing hierarchical versions also take into account node *Art*. In this way the *LCA* versions are stricter than the existing set-based measures. We believe that such a behavior is desirable, because the *LCA* versions are not affected by the number of ancestors that *Art* has, while the existing set-based measures would increasingly underestimate the error, as the path grows.

3.3.2 Switching from Single-label to Multi-label, Handling the Pairing Problem

The cases presented here alter **D1** dimension of the HC problem, by allowing an instance to belong to more than one category. The problems in this setting arise when the number of true and predicted labels (classes) differ, which was defined in Section 2.2 as the Pairing Problem. In this elementary case, where the true and predicted classes are at the same level of the hierarchy, Figures 3.12(a) and 3.12(b) are two symmetric variants leading to different, but symmetric hierarchical precision (P_H) and recall (R_H) scores, as shown in Table 3.4. The same is true for our proposed *LCA* versions of precision and recall (P_{LCA} and R_{LCA}). The results of the simple versions differ from the *LCA* ones, because the *LCA* versions ignore the graph above the node *Music*, which is the lowest common ancestor of nodes *Pop*, *Rock* and *Classical*. The simple hierarchical versions on the other hand also take into account node *Arts* and in that way they give higher results. This behavior is undesirable, as explained in Section 3.3.1.

All other measures give the same result in both cases. $l_\Delta(Y_{aug}, \hat{Y}_{aug})$ always computes the symmetric difference between the two augmented sets. The symmetric difference takes into account only the sum of false positives and false negatives which are the same in both cases. Among the pair-based measures, GIE seems inappropriate for this problem. It matches the true node (*Pop*) with one of the two predicted (*Rock* and *Classical*) in Fig 3.12(a) and penalizes the unmatched predicted class with the maxi-

mum cost, ignoring the fact that the misclassification is in the proximity of the correct category. Similarly in Figure 3.12(b) MGIA provides a more suitable evaluation, as it allows multiple categories to match to the nearest one.



(a) (b)

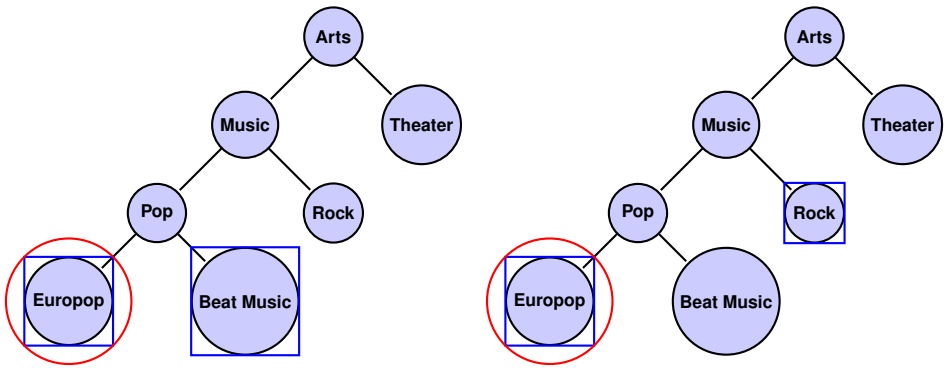
Figure 3.12: Different numbers of true and predicted labels. Nodes surrounded by circles are the true classes while the nodes surrounded by rectangles are the predicted classes.

	GIE	MGIA	P_H	R_H	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	P_{LCA}	R_{LCA}	F_{LCA}
a	7	0.73(4)	0.5	0.66	0.57	3	0.33	0.5	0.4
b	7	0.73(4)	0.66	0.5	0.57	3	0.5	0,33	0.4

Table 3.4: Results per measure for Figure 3.12.

The more complex case of Figure 3.13 is an example showing that taking into account all the ancestors is undesirable compared to our proposed *LCA* approach for set-based measures. The hierarchy is still a tree and the classification is multi-label. *Europop* is predicted correctly, while an extra false category is predicted. Although the mistake in Figure 3.13(b) is worse than that of Figure 3.13(a), since it is further from the true class *Europop*, all set-based measures except our proposed *LCA* measures continue to give the same results. This is because $l_\Delta(Y_{aug}, \hat{Y}_{aug})$ and the hierarchical versions of precision, recall and F_1 take into account all the ancestors of the predicted

and true labels, while the *LCA* versions, which are hybrid measures, uses the augmented graphs $G_t(Y,\hat{Y})$ and $G_p(Y,\hat{Y})$ which were created using only the least common ancestors (*LCAs*). *LCA* measures implicitly pair each node with the closest node of the other set and in that way take into account the distance between predicted and true nodes, which the other set-based measures ignore.



(a) (b)

Figure 3.13: Different numbers of true and predicted labels. Distance from closest true class differs. Nodes surrounded by circles are the true classes while the nodes surrounded by rectangles are the predicted classes.

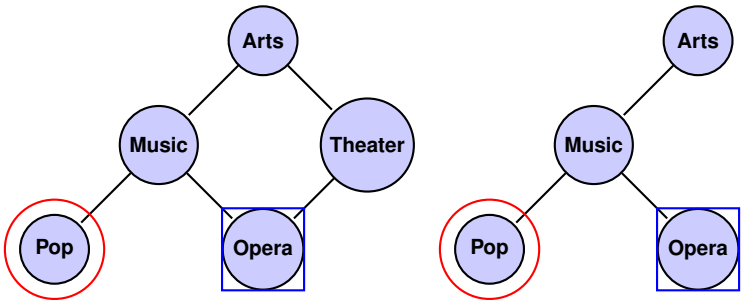
	GIE	MGIA	P_H	R_H	F_H	$l_\Delta(Y_{aug},\hat{Y}_{aug})$	P_{LCA}	R_{LCA}	F_{LCA}
a	2	0.8(2)	0.8	1	0.89	1	0.66	1	0.8
b	3	0.7(3)	0.8	1	0.89	1	0.66	0.66	0.66

Table 3.5: Results per measure for Figure 3.13.

Thus in Figure 3.13(a) the augmented sets of the *LCA* are {*Europop*, *Pop*} and {*Europop*, *Pop*, *Beat Music*}, while for all other set-based measures are {*Europop*, *Pop*, *Music*, *Arts*} and {*Europop*, *Pop*, *Beat Music*, *Music*, *Arts*}. In Figure 3.13(b) the augmented sets of *LCA* become {*Europop*, *Pop*, *Music*} and {*Europop*, *Rock*, *Music*} while for all other set-based measures they remain the same. The differentiation between such cases is an advantage of the *LCA* measures over existing set-based measures.

3.3.3 Single Label Classification with a DAG Hierarchy, Handling Alternative Paths

In Figure 3.14(a) we assume a single-label classification task, where the hierarchy is a *DAG*. In this *DAG* there are two paths from the root (node *Arts*) to node *Opera*. It is worth noting that this is the simplest case, where both paths {*Arts*, *Music*, *Opera*} and {*Arts*, *Theater*, *Opera*} have the same length. In this case, pair-based measures (Table 3.6) provide the same results as for the tree case of Figure 3.14(b), a desirable behavior that is due to the use of the shortest path from *Pop* to *Opera*. On the other hand, existing set-based measures are affected, as they calculate a misclassification error that takes into account both of the two alternative paths to *Opera*. In contrast, *LCA* measures like the pair-based measures, due to the use of the lowest common ancestor.



(a) (b)

Figure 3.14: Many paths of the same length leading node *Arts* to the same node *Opera*. Nodes surrounded by circles are the true classes while the nodes surrounded by rectangles are the predicted classes.

	GIE	MGIA	P_H	R_H	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	P_{LCA}	R_{LCA}	F_{LCA}
a	2	0.8(2)	0.5	0.66	0.57	3	0.5	0.5	0.5
b	2	0.8(2)	0.66	0.66	0.66	2	0.5	0.5	0.5

Table 3.6: Results per measure for Figure 3.14.

3.3.4 Multi-label Classification with a DAG Hierarchy

In this subsection we study cases where the multi-label classification combined with the DAG hierarchy (dimensions **D1** and **D2**) leads to variable behavior of the set-based evaluation measures. Figure 3.15 presents such a case, where although GIE is affected by the matching problem discussed in Section 2.2 , all the set-based methods provide similar results (Table 3.7). The multiple paths phenomenon does not affect the hierarchical versions of precision, recall and F_1 because all nodes above the lowest common ancestors (*Theater* and *Music*) of *Drama*, *Opera* and *Rock* are also shared by all classes (true and predicted). With this example we wish to show that there are certain cases in which existing set-based methods give the same results as the *LCA* ones, but we have not identified cases in which the behaviour of an existing set-based measure would be more desirable than that of the *LCA* versions.

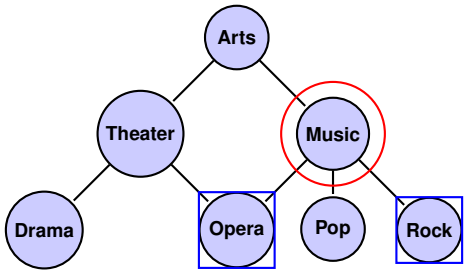


Figure 3.15: Combining the pairing problem with alternative paths in a single example, where none of the set-based methods is affected. Nodes surrounded by circles are the true classes while the nodes surrounded by rectangles are the predicted classes.

GIE	MGIA	P_H	R_H	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	P_{LCA}	R_{LCA}	F_{LCA}
7	0.6(6)	0.4	0.66	0.5	4	0.4	0.66	0.5

Table 3.7: Results per measure for Figure 3.15.

In the case shown in Figure 3.16 the right sub-graph now connects to the left one

only through the *Arts* node and also *Beat Music* connects to *Music* through two different nodes *Electro* and *Pop*. This is the reason why simple hierarchical versions of precision, recall and F_1 differ from the *LCA* versions, as shown in table 3.8. *LCA* versions, due to their nearest common ancestor approach, ignore node *Electro* while all other set-based measures count both *Electro* and *Pop* and thus over-penalize the error of *Beat Music*.

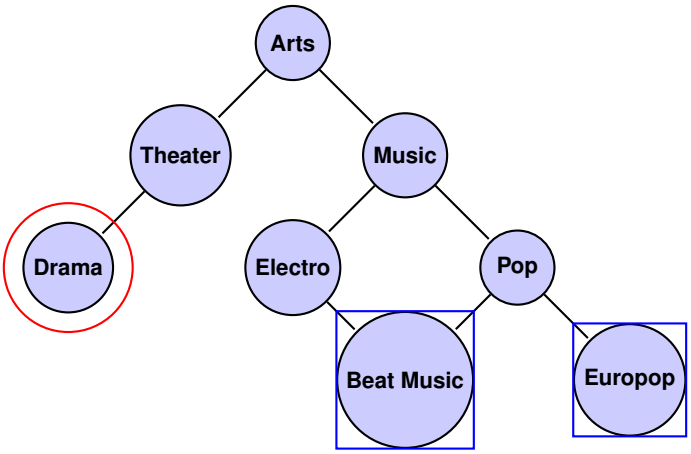


Figure 3.16: Combining the pairing problem with alternative paths in a single example where all previous set-based measures behave undesirably, while our proposed *LCA* measures do not. Nodes surrounded by circles are the true classes while the nodes surrounded by rectangles are the predicted classes.

GIE	MGIA	P_H	R_H	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	P_{LCA}	R_{LCA}	F_{LCA}
10	0(10)	0.166	0.33	0.22	7	0.2	0.33	0.25

Table 3.8: Results per measure for Figure 3.16.

3.3.5 Classification to Inner Nodes, Over and Under-specialization

In all of the previous cases the predicted and the true categories were leaves of the hierarchies, but as we discussed in Section 2.1 (dimension **D3**) this is not always the

case. Figure 3.17 presents simple examples of an inner node being either a true (Figure 3.17(a)) or a predicted (Figure 3.17(b)) category. As shown in Table 3.9 the two cases receive the same score.

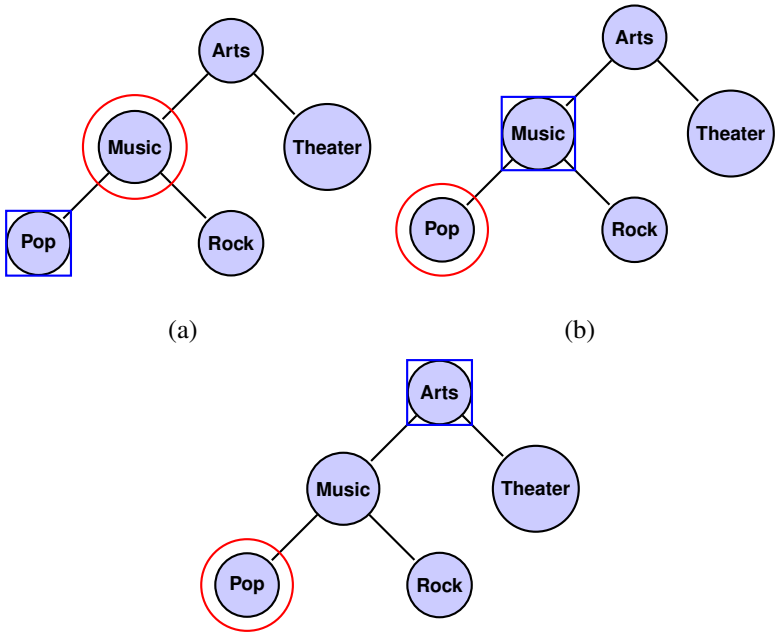


Figure 3.17: Simplest over and under-specialization cases. Nodes surrounded by circles are the true classes while the nodes surrounded by rectangles are the predicted classes.

	GIE	MGIA	P_H	R_H	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	P_{LCA}	R_{LCA}	F_{LCA}
a	1	0.9(1)	0.66	1	0.8	1	0.5	1	0.66
b	1	0.9(1)	1	0.66	0.8	1	1	0.5	0.66
c	2	0.8(2)	1	0.33	0.49	2	1	0.33	0.49

Table 3.9: Results per measure for Figure 3.17.

Figure 3.17(a) shows a case of over-specialization. As described in Section 2.2, different evaluation measures treat this type of error differently. One could even argue that since *Pop* is predicted, *Music* is also predicted as a direct ancestor of it. This is not the case here and as shown in Table 3.9 all measures treat it as a misclassification error.

Regarding under-specialization, the simplest example is shown in Figure 3.17(b). It is also considered an error that is more severe the further the true category is from the predicted. For example in Figure 3.17(c) the predicted node is an ancestor of the predicted node of Figure 3.17(b). All measures lead to a higher error estimate in this case than in 3.17(b). A similar example for over-specialization would lead to the same observations. Our proposed measures provide similar results to the existing ones in this case.

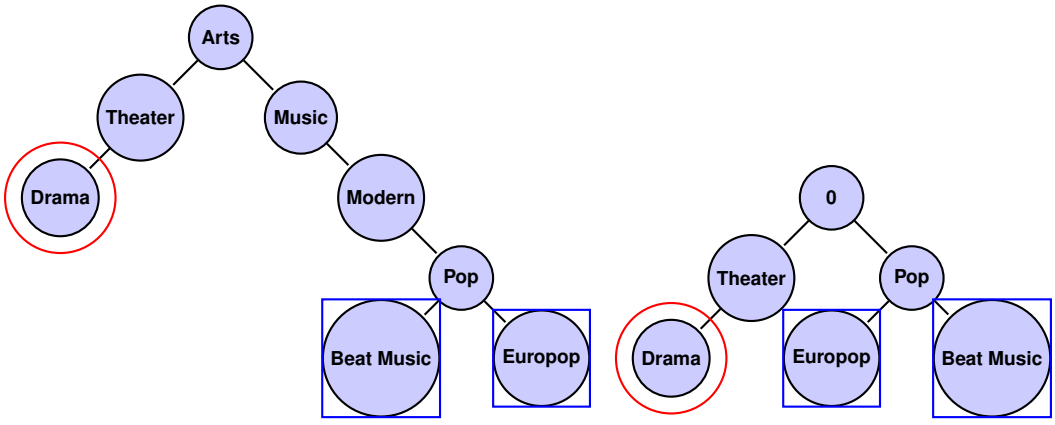
3.3.6 Very distant predictions

The aim of this case (Figure 3.18) is to show how each of the two types of measure handle very large distances between predicted and true labels. As we discussed in Section 2.2, the long distance problem is not affected by the dimensions of the HC problem. Pair-based measures compute the distance between each pair of predicted and true nodes and if this distance is above a certain threshold, a standard maximum distance is assigned. Set-based measures can use a threshold on the number of ancestors of the predicted and true nodes that will be used in the augmented sets. Using this threshold, they impose a common ancestor to be used in order to connect at least one predicted with one true node, at a distance equal to the threshold.

For example, in the case shown in Figure 3.18(a), if a maximum distance of 4 is used for pair-based measures, then for set-based measures it would be reasonable to request a lowest common ancestor at distance 2. By adding the distance of the lowest common ancestor to both the true and the predicted labels, a distance of 4 between them is reached.

This operation on the hierarchy of Figure 3.18(a) leads to the hierarchy of Figure 3.18(b), where a dummy node 0 is used to directly connect nodes *Theater* and *Pop*. In case multiple distant predictions exist, we add a single node 0, which is used to connect them all. The results of each measure are presented in Table 3.10. In this example

we see that all the measures can incorporate a maximum distance threshold that might be necessary either for computational reasons or due to these long distance problem discussed in section 2.2. Pair-based measures are affected more by the threshold than set-based ones, as shown in the example. $l_\Delta(Y_{aug}, \hat{Y}_{aug})$ decreased by 2 points, while MGIA decreased by 4 points. This is due to multiple counting of paths, which most of the times is undesirable as we will discuss in the next section.



(a) (b)

Figure 3.18: Distant prediction problem. Nodes surrounded by circles are the true classes while the nodes surrounded by rectangles are the predicted classes.

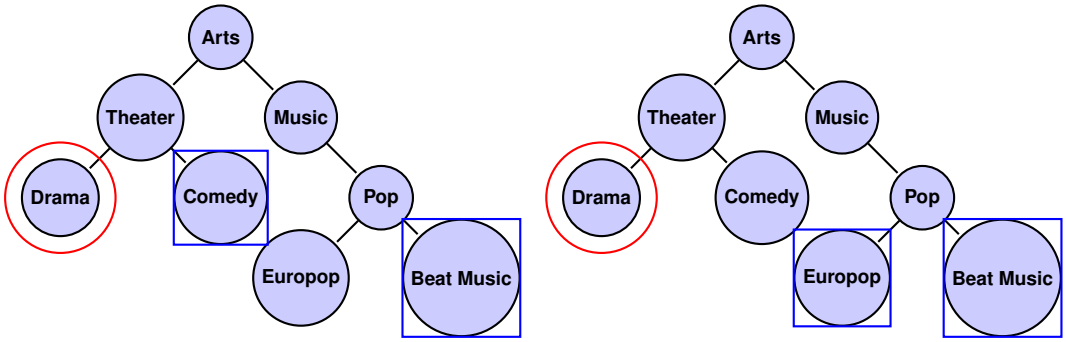
	GIE	MGIA	P_H	R_H	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	P_{LCA}	R_{LCA}	F_{LCA}
a	6 + Max	0.2(12)	0.16	0.33	0.22	7	0.16	0.33	0.22
b	4 + Max	0.466(8)	0.25	0.33	0.28	5	0.25	0.33	0.28

Table 3.10: Results per measure for Figure 3.18.

3.3.7 Multiple path counting

Due to comparison of pairs of true and predicted classes, pair-based methods will often count the same path more than once. This multiple counting increases the error estimated by these methods. Figure 3.19 illustrates such a case. In Figure 3.19(a) the

edge between *Drama* and *Theater* will be counted both in the path {*Drama*, *Theater*, *Comedy*} and in the path {*Drama*, *Theater*, *Arts*, *Music*, *Pop*, *Beat Music*}. As a result, pair-based measures in this case tend to overestimate the errors, in comparison to set-based ones. For each extra predicted node that we add as a descendant of *Theater*, pair-based error estimates increase by at least 2 while the size of Y_{aug} in set-based measures increases by 1 which seems more reasonable for such a change in the hierarchy.



(a) (b)

Figure 3.19: Comparison between pair-based and set-based measures, counting paths more than once. Nodes surrounded by circles are the true classes while the nodes surrounded by rectangles are the predicted classes.

	GIE	MGIA	P_H	R_H	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	P_{LCA}	R_{LCA}	F_{LCA}
a	7	0.533(7)	0.33	0.66	0.44	5	0.33	0.66	0.44
b	10	0.33(10)	0.2	0.33	0.25	6	0.2	0.33	0.25

Table 3.11: Results per measure for Figure 3.19.

Figure 3.19(b) presents a similar example. According to Table 3.11, the error of MGIA before the proposed transformation would increase from 7 to 10, while $l_\Delta(Y_{aug}, \hat{Y}_{aug})$ increases from 5 to 6 and F_{LCA} decreases from 0.44 to 0.25. This is because the whole path form *Drama* to *Pop* is counted twice by the pair-based method. However double counting seems desirable in this case, as the error in 3.19(b) is more severe than that in 3.19(a). While both measures penalize the error in 3.19(b) more than in 3.19(a), the

extra penalization of MGIA is roughly proportional to the distance between the true and the predicted category node, while that of the set-based measures is not. All set-based measures would give the same result even if *Europop* was a child of *Music*, which is a less severe error than when it is a child of *Pop*. Therefore, counting more than once the common paths may be an advantage of the pair-based measures in some cases.

3.3.8 Summary

Table 3.12 presents the advantages and disadvantages of each measure across all the cases presented in Section 3.2.2. The pair-based measures can handle alternative paths, while, from the set-based measures, only the proposed *LCA* measures are able to deal with them efficiently. All measures can handle over-specialization and under-specialization in some way. All set-based measures can deal with the pairing problem since they produce augmented sets, while GIE cannot handle it efficiently and this is why MGIA was proposed. Considering the long distance problem both pair-based measures and set-based measures can handle it using the appropriate thresholding (see Section 3.3.6). Finally, multi-path counting is a special feature of the pair-based measures which although most of the times is undesirable, it could make them behave better than the set-based measures in certain cases.

	GIE	MGIA	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	F_{LCA}
Alternative Paths	+	+	-	-	+
Over-specialization	+	+	+	+	+
Under-specialization	+	+	+	+	+
Pairing Problem	-	+	+	+	+
Long Distance Problem	+	+	+	+	+
Multiple Path Count	-	-	+	+	+

Table 3.12: Summary table regarding evaluation measures in certain situations.

Table 3.13 presents the best measures that researchers could use in any given prob-

lem setting. The first dimension defines whether the problem is multi-label or single-label, while the second is about the type of the hierarchy. We have not added the third dimension here (classification to any node or only to leaves) since we have not found any advantages of the existing or proposed measures in that dimension.

	Single-label	Multi-label
Tree hierarchy	Any pair-based or LCA	LCA or MGIA
DAG hierarchy	LCA or MGIA	LCA or MGIA

Table 3.13: Summary table regarding evaluation measures in certain problem settings.

As a general conclusion the proposed measures always behave better or at least as well as the existing measures of their category. Therefore, if one wishes to use a pair-based or a set-based measure we suggest using the ones proposed in this thesis, instead of the existing ones. Furthermore, in most cases one should choose the *LCA* measures over MGIA, due to the multiple counting of paths discussed in Section 3.7. Multiple counting of paths is most of the times undesirable since it leads to over-penalization. Additionally, these cases could also serve as benchmarks, in order to observe the behaviour of newly proposed hierarchical evaluation measures. In this way, we conclude the discussion regarding the behavior of the measures in benchmark cases and we move on to observe their behaviour when using real data and systems in the following section.

3.4 Empirical Study

In this section we apply various evaluation measures to the predictions of the systems that participated in the Large Scale Hierarchical Text Classification Pascal Challenges of 2011 (LSHTC2) and 2012 (LSHTC3). We also present ranking correlations between measures from the first task of the 2013 BioASQ challenge. The goal of this section is to study using real data and systems, the extent to which the performance ranking of systems is affected by the choice between flat and hierarchical evaluation measures and

also by the type of hierarchical measure used.

In the first subsection we present the datasets that we used, in the second subsection we discuss the evaluation measures included in the comparison and in the final subsection we discuss the results of the study. In Section 3.3 we demonstrated that, in certain cases, some measures behave more desirably than others. In this section we show that the differences among the methods also affect the rankings of real systems in practice.

3.4.1 Datasets

In LSHTC2 three different datasets were provided as three separate tasks. Each participant could participate in any or all of them with the same system or with different ones. The first dataset (DMOZ) was based on pages crawled from the Open Directory Project (ODP), a human-edited hierarchical directory of the Web.[7] The hierarchy of this dataset was transformed into a tree and all instances deeper than level five of the hierarchy were transferred to the fifth level, thus leading to a hierarchy with a maximum depth of 5. This dataset was the smallest of the three, regarding the number of categories and instances.

The other two datasets of LSHTC2, also used in LSHTC3, are based on DBpedia.[8] They are called DBpedia Large and DBPedia Small, respectively. The largest of the two datasets, DBPedia Large, contains almost all abstracts of DBpedia, as instances to be used for training and classification, with the exception of some non-English abstracts. Therefore this dataset comprises many more categories than the DMOZ one and goes to a larger depth. DBpedia Small is a subset of DBpedia Large, selected in a way that led to a dataset of similar size to DMOZ, while maximizing the ratio of instances per node. This process has resulted in a much easier classification task. The hierarchy of the DBpedia Small dataset has been transformed into a *DAG*, by removing cycles, while

[7] http://www.dmoz.org/

[8] http://dbpedia.org/About

cycles still appear in DBpedia Large.

All three datasets were pre-processed in the same way. All the words of the abstracts were stemmed and each stem was mapped to a feature id. The categories (classes) were also mapped to category ids. Each instance was represented in sparse vector format as a collection of category ids and a collection of feature ids accompanied by their frequencies in the instance. The mapping between ids, categories and stems was different for each dataset. Only leaves of each hierarchy were used as valid classification nodes for LSHTC2, while in LSHTC3 participants were also allowed to classify instances in inner nodes. For each inner node of the hierarchy that was assigned instances, however, a dummy leaf was created for evaluation purposes as a direct child and all the instances were transferred to the child.

Table 3.14 presents basic statistics of the three datasets. The first two datasets are almost of the same size, but DBpedia Small is more multi-labeled and has a deeper, less ballanced hierarchy than DMOZ. However the ratio of training instances to categories is comparable in the two datasets (14.16 for DMOZ and 12.5 for DBpedia Small). DBpedia Large is very different in this respect, having a ratio of training instances to categories equal to 7.2. The ratio of multi-labeled training instances per category is almost the same across the two DBpedia datasets (23.27 for Small and 23.73 for Large) and much smaller for DMOZ (14.5). DBpedia Large is also much larger than the other two datasets in terms of training and testing instances.

We also present rankings from the results of the first task of the 2013 BioASQ Challenge.[9] The training data of this task was composed of 800,000 *PubMed* biomedical journal abstracts which belonged to 25,000 classes, that come from the *MeSH* hierarchy. The participants were asked to classify new *PubMed* documents, as they appeared online, before they were manually annotated by *PubMed* curators. The whole task was composed of three batches and each batch lasted six weeks. Our proposed F_{LCA} was the

[9] http://www.bioasq.org/

	DMOZ	DBpedia Small	DBpedia Large
#cats	27,875	36,504	325,056
#train inst	394,756	456,886	2,365,436
#test inst	104,263	81,262	452,167
multi-label factor	1.0239	1.8596	3.2614
train inst per cat	14.16	12.5	7.2
multi-label train inst per cat	14.5	23.27	23.73
max depth	5	10	14

Table 3.14: Basic statistics of datasets showing the number of categories, the number of training and test instances, the average number of true categories per instance (multi-label factor), the ratio of training instances to categories, the ratio of multi-labeled training instances to categories and the maximum depth of the hierarchy.

hierarchical measure used in this task in order to decide the winning systems.

3.4.2 Evaluation Measures and Statistical tests

The evaluation measures that were used in this study were the ones presented in Section 3.3. Accuracy and GIE are reproduced here as reported during the challenge. Using these evaluation measures, different rankings of the participating systems are created. In order to measure the correlation between these rankings, we used Kendall's rank correlation (Kendall, 1938).

In the LSHTC2 challenge, statistical significance tests were only used for the flat evaluation measures. To the best of our knowledge, the literature does not provide special statistical significance tests for hierarchical measures. In this work, as well as in LSHTC3, we performed a micro sign test (s-test) similar to that used in (Yang and Liu, 1999). Each of the hierarchical measures provides a score for each instance and this score is always averaged over the number of instances. Assuming that:

- a_i is the performance of system a for instance i, according to an evaluation mea-

sure,

- b_i is the performance of system b for instance i, according to the same evaluation measure,
- n is the number of times that a_i and b_i differ over all i,
- k is the number of times that a_i performs better than b_i over all i,

the null hypothesis (H_0) is that k has a bionomial distribution Bin(n, p), where $p = 0.5$. H_1 is that $p > 0.5$, meaning that system a is better than system b. According to (Yang and Liu, 1999), if n is greater than 12, which is always the case in these large scale problems, then the p-value can be approximately computed using the standard normal distribution for:

$$Z = \frac{k - 0.5n}{0.5\sqrt{n}} \tag{3.6}$$

It is worth stressing that the s-test only takes into account which system performs better at each instance, ignoring how much better it performs. Alternatively, the Wilcoxon signed-rank test (Wilcoxon, 1945) could take into account the difference in performance at each instance. For reasons of simplicity however, in these experiments we used the s-test.

3.4.3 Results

In this subsection we present the results for each dataset and discuss the behavior of each measure. Table 3.15 presents the results on the DMOZ dataset for all the systems that participated in the LSHTC2 challenge. Recall that DMOZ has a tree hierarchy and is the least multi-labeled dataset of the three. Systems are evaluated by each measure and are ranked in descending order. The number in brackets indicates the system's rank using the corresponding measure. If two systems have the same rank, it means that there is no statistically significant difference between their results according to our statistical

significance tests. Table 3.16 presents the Kendall rank correlation between each pair of rankings.

System	Acc	GIE	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	MGIA	F_{LCA}
A	0.388 (1)	2.829 (2)	0.653 (3)	3.962 (2)	0.541 (2)	0.557 (2)
B	0.387 (2)	2.823 (1)	0.660 (1)	3.910 (1)	0.550 (1)	0.559 (1)
C	0.386 (3)	2.831 (3)	0.654 (2)	3.987 (3)	0.542 (1)	0.555 (3)
D	0.380 (4)	3.322 (6)	0.642 (5)	4.257 (4)	0.515 (4)	0.544 (5)
E	0.378 (5)	**3.832 (10)**	0.640 (6)	4.458 (7)	0.501 (5)	0.538 (6)
F	0.371 (6)	2.891 (4)	0.652 (4)	3.996 (4)	**0.538 (3)**	**0.547 (4)**
G	0.347 (7)	3.027 (5)	0.622 (7)	4.335 (6)	0.497 (6)	0.522 (7)
H	0.284 (8)	3.456 (7)	**0.497 (11)**	**5.878 (11)**	0.364 (8)	0.440 (9)
I	0.269 (9)	3.503 (9)	0.571 (8)	4.987 (9)	0.421 (7)	0.460 (8)
J	0.262 (10)	3.476 (8)	0.570 (8)	4.966 (8)	0.428 (7)	0.458 (8)
K	0.172 (11)	3.898 (11)	0.469 (12)	6.165 (12)	0.318 (10)	0.373 (11)
L	0.155 (12)	4.010 (12)	0.446 (13)	6.430 (13)	0.282 (11)	0.353 (12)
M	0.153 (13)	4.024 (13)	0.497 (10)	5.803 (10)	0.333 (9)	0.374 (10)
N	0.107 (14)	4.289 (14)	0.384 (14)	7.080 (14)	0.202 (12)	0.306 (13)
O	0.087 (15)	4.419 (15)	0.340 (15)	7.744 (15)	0.175 (13)	0.280 (14)

Table 3.15: DMOZ results in LSHTC2. Interesting rank changes are marked with bold.

	Acc	GIE	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	MGIA	F_{LCA}
Acc	1					
GIE	0.829	1				
F_H	0.842	0.785	1			
$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	0.790	0.810	0.919	1		
MGIA	0.829	0.810	0.976	0.924	1	
F_{LCA}	0.867	0.810	0.976	0.924	0.962	1

Table 3.16: Kendall's rank correlation on the evaluation measure rankings of DMOZ in LSHTC2.

The first observation is that the ranking of the flat accuracy is different from that of the hierarchical measures. This shows that flat and hierarchical measures treat the problem differently. Another interesting observation is that the rankings also differ between hierarchical measures.

The handling of multiple labels per instance is an important aspect of the classification methods. Table 3.17 presents the average number of predictions per instance for each system. Since most instances of the dataset are single-labeled, most of the participants treated the task as a single-label one. As discussed in previous sections, the treatment of multi-labeling by different measures, greatly affects their behavior, but since multi-labeling is rare in this dataset, this decision did not affect much the hierarchical measures. However there are some examples of systems, such as M and J, which assign multiple labels and perform better according to hierarchical measures than according to accuracy. D and E, on the other hand, perform worse using some hierarchical measures than with accuracy. The more multi-labeled a result is, the greater the opportunity is for a hierarchical measure to reward or penalize the systems for their decision.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	1	1	1.11	1.22	1	1	1	1.02	1.01	1	1	1.02	1	1

Table 3.17: Average number of predictions per instance of DMOZ systems in LSHTC2.

As discussed in previous sections, hierarchical measures vary in the way they handle multi-labeling and *DAG* hierarchies. Being a tree hierarchy and almost single-labeled, the DMOZ dataset does not reveal a lot of these differences. The tree hierarchy is the main reason why, according to Table 3.16, $l_\Delta(Y_{aug}, \hat{Y}_{aug})$ and F_H are very highly correlated with F_{LCA}, since their main difference is in the way they treat multiple ancestors, something possible only in DAGs and not in trees. F_{LCA} and F_H are more correlated with each other than with $l_\Delta(Y_{aug}, \hat{Y}_{aug})$, as expected, since they differ in the calculations they perform on the augmented sets. We also observe a correlation between GIE and MGIA, due to the limited multi-labeling that provides fewer opportunities for dealing with the pairing problem and the proposed tranformation of the error that MGIA performs.

Tables 3.18 and 3.19 present the LSHTC2 results on DBpedia Small, which is a more multi-labeled dataset with a *DAG* hierarchy. As expected, these characteristics

greatly affect the behavior of the measures. The most important observation is that the two hierarchical measures (GIE and $l_\Delta(Y_{aug}, \hat{Y}_{aug})$) that measure only the error, without applying a transformation to it, have very low correlation with accuracy and the hierarchical measures.

System	Acc	GIE	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	MGIA	F_{LCA}
A2	0.374 (1)	**4.171 (5)**	0.647 (1)	**12.114 (6)**	0.356 (1)	0.481 (1)
B2	0.362 (2)	**4.364 (6)**	0.641 (3)	**12.000 (4)**	0.337 (2)	0.470 (2)
C2	0.354 (3)	4.076 (4)	0.646 (2)	11.651(3)	0.323 (4)	0.463 (3)
D2	0.351 (4)	3.858 (2)	0.629 (4)	11.332 (2)	0.329 (3)	0.462 (4)
E2	0.279 (5)	3.726 (1)	0.600 (5)	11.326 (1)	0.286 (5)	0.414 (5)
F2	0.252 (6)	3.859 (3)	0.579 (6)	11.996 (5)	0.280 (6)	0.399 (6)
G2	0.249 (7)	5.701 (7)	0.561 (7)	16.915 (7)	0.245 (7)	0.381 (7)

Table 3.18: DBpedia Small results in LSHTC2. Significant deviations of rankings are highlighted in bold.

	Acc	GIE	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	MGIA	F_{LCA}
Acc	1					
GIE	-0.143	1				
F_H	0.905	-0.048	1			
$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	-0.143	0.810	-0.048	1		
MGIA	0.810	-0.143	0.714	-0.143	1	
F_{LCA}	0.905	-0.238	0.810	-0.238	0.905	1

Table 3.19: Kendall's rank correlation on the evaluation measure rankings of DBpedia Small in LSHTC2.

Furthermore, taking into consideration the average predictions per instance of each system (Table 3.20), we observe a relation between the rankings of these two measures. By computing only the error, these two measures take into account only FP and FN, without counting the TP. For this reason they tend to penalize systems with higher average predictions per instance, since they are more likely to make more mistakes. The way the rest of the set-based measures handle the augmented true and predicted

sets of classes and the transformation that MGIA performs to the error is much closer to the idea of doing calculations with TP, FP, TN, FN, as accuracy does, and for this reason these measures are more correlated with it. These measures also reward the systems that make more predictions per instance, if they manage to have some extra TPs by doing so.

A2	B2	C2	D2	E2	F2	G2
2.04	1.94	1.82	1.51	1.11	1.14	2.84

Table 3.20: Average predictions per instance of DBpedia Small systems in LSHTC2.

Tables 3.21 and 3.22 present the results on the same dataset (DBpedia Small), but with the systems of LSHTC3. The number of systems participating in LSHTC3 is much larger than LSHTC2 (17 instead of 7). This is not only important for statistical reasons (more experiments lead to safer conclusions), but also because according to Table 3.23 we now have more systems that make many predictions per instance, something which affects the behavior of the measures. Another important difference is that in LSHTC3 systems were allowed to classify to inner nodes, even if these nodes did not have any training instances directly belonging to them.

F_H and F_{LCA} are the hierarchical measures that are most correlated with flat accuracy, although the correlation is much lower in this case where we have many more systems and inner node classification is treated as a mistake by accuracy. The correlation between GIE and MGIA is much higher than that of LSHTC2, but they are not fully correlated. A high correlation also continues to be observed between GIE and $l_\Delta(Y_{aug}, \hat{Y}_{aug})$ for the reason explained above for LSHTC2.

A very interesting case is that of system $X2$, which predicts many categories (labels) per instance, 10.649 labels per instance, when the average true labels per instance is 1.8550. This means that a large number of predicted labels is wrong, while the predicted labels could still be in the vicinity of the correct ones. As expected, flat ac-

System	Acc	GIE	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	MGIA	F_{LCA}
H2	0.438 (1)	3.060 (1)	0.709 (1)	9.096 (1)	0.421 (2)	0.543 (1)
I2	0.429 (2)	3.155 (2)	0.689 (3)	9.310 (2)	**0.398 (4)**	**0.525 (4)**
J2	0.42 (3)	3.530 (5)	0.692 (2)	10.143 (6)	0.403 (3)	0.529 (2)
K2	0.417 (4)	**4.428 (11)**	0.677 (5)	11.385 (11)	0.378 (6)	0.509 (5)
L2	0.408 (5)	3.187 (3)	0.680 (4)	9.561 (3)	**0.443 (1)**	**0.527 (3)**
M2	0.385 (6)	3.319 (4)	0.666 (7)	10.122 (5)	0.390 (4)	0.500 (6)
N2	0.371 (7)	4.991 (13)	0.645 (8)	13.117 (13)	0.342 (8)	0.476 (8)
O2	0.357 (8)	4.302 (10)	0.643 (8)	12.185 (12)	0.323 (9)	0.462 (10)
P2	0.354 (9)	3.550 (6)	0.633 (9)	**11.146 (8)**	0.381 (5)	0.478 (7)
Q2	0.327 (10)	3.600 (8)	0.639 (8)	10.944 (7)	0.312 (11)	0.450 (12)
R2	0.32 (11)	3.552 (7)	0.603 (10)	11.365 (10)	**0.361 (7)**	0.453 (11)
S2	0.298 (12)	5.693 (14)	0.549 (13)	16.873 (15)	0.243 (13)	0.407 (13)
T2	0.25 (13)	3.741 (9)	0.592 (11)	11.304 (9)	**0.089 (14)**	0.397 (14)
U2	0.249 (14)	5.701 (15)	0.561 (12)	16.915 (16)	0.245 (12)	0.381 (15)
V2	0.245 (15)	4.780 (12)	0.537 (14)	14.351 (14)	0.234 (13)	0.374 (16)
W2	0.063 (16)	9.139 (16)	0.345 (15)	24.009 (17)	0.045 (15)	0.208 (17)
X2	**0.047 (17)**	**25.775 (17)**	**0.668 (6)**	**9.607 (4)**	**0.321 (10)**	**0.471 (9)**

Table 3.21: DBpedia Small results in LSHTC3. With bold, interesting differences in rankings.

curacy penalizes this behavior giving the lowest rank to this system. GIE and MGIA also penalize this system, although MGIA less severely. On the other hand, the set-based measures do not punish X2 that much (6 for F_H, 4 for $l_\Delta(Y_{aug}, \hat{Y}_{aug})$ and 9 for F_{LCA}. A closer look at the system shows that it is in fact not that bad. However, it returns all the nodes of a path from the root to leaf as predicted labels, instead of just the leaf. Set-based measures still penalize it when the leaf and its ancestors are wrong predictions, but they do not over-penalize it, unlike pair-based measures. This is a nice example, of the difference between set-based and pair-based measures. The treatment of the X2 system by MGIA and F_{LCA} is very similar and falls in between the set-based measures that treat it rather favorably and the flat and pair-based methods that punish it more severely. This is because these measures can be seen as hybrid measures since the former conducts a set operation (although it is a pair-based measure) and the later incorporates matching between true and predicted nodes in order to create the augmented

	Acc	GIE	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	MGIA	F_{LCA}
Acc	1					
GIE	0.662	1				
F_H	0.765	0.485	1			
$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	0.485	0.735	0.662	1		
MGIA	0.691	0.618	0.721	0.588	1	
F_{LCA}	0.794	0.574	0.853	0.603	0.838	1

Table 3.22: Kendall's rank correlation on the evaluation measure rankings of DBpedia Small in LSHTC3.

H2	I2	J2	K2	L2	M2	N2	O2	P2
1.506	1.482	1.909	2.208	1.415	1.490	2.427	1.889	1.414
Q2	R2	S2	T2	U2	V2	W2	X2	
1.529	1.184	2.423	2.000	2.841	1.712	4.334	10.649	

Table 3.23: Average predictions per instance on DBpedia Small in LSHTC3.

sets. These characteristics help them to overcome the weaknesses of the measures of their respective categories and in that way their behavior is more desirable.

On the third dataset (DBpedia Large) we faced some computational issues with the hierarchical measures. The problem originated from the very large scale of the dataset's hierarchy, which is a *DAG* (in reality it contains circles but we removed them). To avoid the computational problems, we ran the evaluation measures with a maximum path threshold of 2 and 4. This means that all nodes are forced to have a lowest common ancestor at a depth of 1 and 2 respectively (if they do not have one we create a dummy one). The approach is similar to the idea of the Long Distance problem of Figure 3.2(e) discussed in Section 3.2.1. In the Long Distance problem we used dummy nodes in order to link nodes that were further than a threshold from each other, in order to avoid overpenalization. The same dummy nodes are used here for computational reasons.

Tables 3.24 and 3.25 present the results for a distance threshold of 4 for the systems of LSHTC2. Since this dataset has the most complex hierarchy and it is the most multi-labeled one, it should be treated very differently by each measure. Interestingly the

rankings of F_H, MGIA and F_{LCA} remain highly correlated with accuracy compared to the other measures, although the number of systems is not high enough (only 5 systems) to make safe conclusions.

System	Acc	GIE	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	MGIA	F_{LCA}
A3	0.347 (1)	4.647 (4)	0.538 (1)	42.470 (3)	0.319 (1)	0.44 (1)
B3	0.337 (2)	4.392 (2)	0.511 (2)	40.811 (1)	0.315 (2)	0.437 (2)
C3	0.283 (3)	**6.178 (5)**	0.440 (4)	50.709 (5)	**0.253 (4)**	0.39 (4)
D3	0.272 (4)	4.288 (1)	0.483 (3)	42.430 (2)	0.294 (3)	0.388 (3)
E3	0.177 (5)	**4.535 (3)**	0.314 (5)	47.957 (4)	**0.212 (5)**	0.331 (5)

Table 3.24: DBpedia Large results with a maximum path threshold of 4 in LSHTC2. With bold, interesting differences in rankings.

	Acc	GIE	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	MGIA	F_{LCA}
Acc	1					
GIE	0	1				
F_H	0.8	0.2	1			
$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	0.2	0.8	0.4	1		
MGIA	0.8	0.2	1	0.4	1	
F_{LCA}	0.8	0.2	1	0.4	1	1

Table 3.25: Kendall's rank correlation on the evaluation measure rankings with a maximum path threshold of 4 on DBpedia Large in LSHTC2.

Another interesting observation is the disagreement of GIE and MGIA about systems C3 and E3. As shown in Table 3.26, E3 predicts fewer categories per instance than C3. Since most of the times the predicted categories (labels) are fewer than the true ones and GIE over-penalizes all the unmatched true categories, it is natural for GIE to penalize system C3 more than E3. This problem is fixed by MGIA, which allows multi-pairing and this is why it instead ranks C3 as a better system than E3.

We can also see that this difficult hierarchy affects the performance of $l_\Delta(Y_{aug}, \hat{Y}_{aug})$ and its ranks become less correlated with F_{LCA}. Another interesting observation is that

A3	B3	C3	D3	E3
3.15	2.69	3.62	2.81	1.27

Table 3.26: Average predictions per instance on DBpedia Large systems in LSHTC2.

F_H, MGIA and F_{LCA} are completely correlated with each other and not correlated with the average number of predictions per instance (Table 3.26).

Tables 3.27 and 3.28 present the results for a maximum path threshold of 2. The main purpose of this experiment is to show that the measures remain largely unaffected by this parameter. Indeed most rankings do not seem to be affected compared to Tables 3.24 and 3.25. Nevertheless, general advice is to keep the maximum paths parameter as large as possible.

	Acc	GIE	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	MGIA	F_{LCA}
A3	0.347 (1)	4.647 (4)	0.503 (1)	22.006 (3)	0.206 (2)	0.460 (2)
B3	0.337 (2)	4.392 (2)	0.475 (2)	21.028 (1)	0.207 (1)	0.461 (1)
C3	0.283 (3)	6.178 (5)	0.412 (4)	25.465 (5)	0.163 (3)	0.416 (3)
D3	0.272 (4)	4.288 (1)	0.439 (3)	21.702 (2)	0.155 (4)	0.405 (4)
E3	0.177 (5)	4.535 (3)	0.282 (5)	24.146 (4)	0.134 (5)	0.360 (5)

Table 3.27: DBpedia Large results with a maximum path threshold of 2 in LSHTC2.

	Acc	GIE	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	MGIA	F_{LCA}
Acc	1					
GIE	0	1				
F_H	0.8	0.2	1			
$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	0.2	0.8	0.4	1		
MGIA	0.8	0.2	0.6	0.4	1	
F_{LCA}	0.8	0.2	0.6	0.4	1	1

Table 3.28: Kendall's rank correlation on the evaluation measure rankings with a maximum path threshold of 2 on DBpedia Large in LSHTC2.

Tables 3.29 and 3.30 present the results on the same dataset (DBpedia Large), but with the systems of LSHTC3. The main difference is that in LSHTC3, systems were

allowed to classify to inner nodes. Table 3.31 shows that the average number of predictions per instance is similar to that of LSHTC2. The most interesting observation is that system F3 is ranked once again high according to Accuracy and all the other hierarchical measures except GIE and $l_\Delta(Y_{aug}, \hat{Y}_{aug})$ which rank it as one of the worst systems. It is even more interesting that, according to Table 3.31, system F3 provides the largest number of labels per instance. The assignement of many labels is penalized heavily by measures based only on FP and FN as we mentioned above.

System	Acc	GIE	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	MGIA	F_{LCA}
F3	**0.381 (1)**	**6.104 (6)**	0.557 (1)	**42.790 (5)**	0.374 (2)	0.465 (1)
G3	0.346 (2)	3.756 (1)	0.513 (3)	33.630 (1)	**0.309 (5)**	0.456 (2)
H3	0.340 (3)	3.763 (2)	0.508 (4)	38.137 (2)	0.35 (3)	0.45 (3)
I3	0.333 (4)	3.763 (3)	0.507 (5)	44.435 (6)	0.31 (4)	**0.43 (5)**
J3	0.332 (5)	4.216 (4)	0.517 (2)	40.560 (3)	0.381 (1)	0.449 (4)
K3	0.272 (6)	4.288 (5)	0.483 (6)	42.430 (4)	0.294 (6)	0.388 (6)

Table 3.29: DBpedia Large results with a maximum path threshold of 4 in LSHTC3. With bold, interesting differences in rankings.

	Acc	GIE	F_H	$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	MGIA	F_{LCA}
Acc	1					
GIE	0.276	1				
F_H	0.6	0.138	1			
$l_\Delta(Y_{aug}, \hat{Y}_{aug})$	0.2	0.552	0.067	1		
MGIA	0.2	-0.276	0.6	-0.067	1	
F_{LCA}	0.828	0.071	0.828	0.276	0.414	1

Table 3.30: Kendall's rank correlation on the evaluation measure rankings with a maximum path threshold of 4 on DBpedia Large in LSHTC3.

F3	G3	H3	I3	J3	K3
3.949	1.482	2.315	2.903	2.902	2.810

Table 3.31: Average predictions per instance of DBpedia Large systems in LSHTC3.

Another interesting observation is that MGIA and F_{LCA} are not fully correlated anymore. In fact MGIA is more correlated with F_H than with F_{LCA}. As the hierarchy becomes more complicated and the results more multi-labeled our two proposed measures behave more differently. Finally system G3, which predicts the smallest number of instances per document, is one of the best systems according to all measures except MGIA which ranks it as one of worst. This is because although G3 has the highest P_H and P_{LCA}, it also has a very low R_H and R_{LCA} compared to other systems. The computation of F_1 seems more suitable in this case compared to the transformation that we proposed for MGIA.

Similar experiments with a maximum path threshold of 2 were also conducted with systems of LHSTC3, but the results were similar to the ones of LSHTC2 and thus we omit them here, in the interest of space.

Tables 3.32, 3.33 and 3.34 present the Kendall's rank correlations of the three task 1 batches of the 2013 BioASQ challenge. These tables contain results for two flat and two hierarchical evaluation measures. For the flat measures we present accuracy and Micro-F_1 measure, while regarding the hierarchical measures we present our proposed F_{LCA} and F_H. F_{LCA} and Micro-F_1 were used in the challenge to select the winning systems.

The first observation is that according to all three tables the two flat measures are more correlated with each other, than with the rest of the measures. This is a confirmation indication that the flat and the hierarchical measures lead to different rankings. Interestingly the same is not always true for the two hierarchical ones, since in the second batch F_H is more correlated with the two flat measures than with F_{LCA}. This observation supports the argument that F_{LCA} has sufficiently different behavior in order to differentiate from the behavior of F_H. Finally, F_{LCA} is the less correlated with all the other measures.

The experiments presented in this section illustrated, with the use of real systems and datasets, that hierarchical measures treat the competing systems differently than flat

	Acc	F_{LCA}	Micro-F_1	F_H
Acc	1			
F_{LCA}	0.91	1		
Micro-F_1	0.95	0.88	1	
F_H	0.89	0.90	0.87	1

Table 3.32: Kendall's rank correlation on the evaluation measure rankings with a maximum path threshold of 8 on batch 1 of the BioASQ challenge (Task 1).

	Acc	F_{LCA}	Micro-F_1	F_H
Acc	1			
F_{LCA}	0.89	1		
Micro-F_1	0.98	0.91	1	
F_H	0.94	0.92	0.94	1

Table 3.33: Kendall's rank correlation on the evaluation measure rankings with a maximum path threshold of 8 on batch 2 of the BioASQ challenge (Task 1).

measures. This was shown by presenting the differences in the rankings of the systems across four datasets. Flat evaluation measures, which are commonly used, often provide a false impression of which system performs better by ignoring hierarchical dependencies of classes and treating all errors equally. As a result, their use guides research away from the methods that incorporate the hierarchy in the classification process. We also showed that different variants of hierarchical measures give different rankings under different conditions. The goal was not to choose the best measure, but to show that different hierarchical evaluation measures give different results, not only in absolute values but also in the ranking of the systems. Finally we showed that the scale of the task is also an issue which requires attention.

	Acc	F_{LCA}	Micro-F_1	F_H
Acc	1			
F_{LCA}	0.90	1		
Micro-F_1	0.99	0.90	1	
F_H	0.92	0.92	0.90	1

Table 3.34: Kendall's rank correlation on the evaluation measure rankings with a maximum path threshold of 8 on batch 3 of the BioASQ challenge (Task 1).

3.5 Conclusions

In this chapter we studied the problem of evaluating the performance of HC methods. Specifically, this work abstracted and presented the key points of existing performance measures. We proposed a grouping of the methods into a) *pair-based* and b) *set-based*. Measures in the former group attempt to match each prediction to a true class and measure the distance between classes. In contrast set-based measures use the hierarchical relations in order to augment the sets of predicted and true labels, and then use set operations, like symmetric difference and intersection, on the augmented label sets.

In order to model pair-based measures, we introduced a novel generic framework based on flow networks, while for set-based measures we provided a framework based on set operations. Thus, salient features of these measures are stressed and presented under a common formalism.

Another contribution of this work was the proposal of two measures (one for each group) that address several deficiencies of existing measures. The proposed measures, along with existing ones, were assessed in two ways. First, we applied them to selected cases, in order to demonstrate their pros and cons. Then we also studied their behaviour empirically on four large datasets based on DMOZ, Wikipedia and biomedical data (BioASQ) with different characteristics (single-label, multi-label tree and DAG hierarchies). The analysis of the results showed that the hierarchical measures behave differently, especially in cases of multi-label data and DAG hierarchies. Also, the two

proposed measures have shown a more robust behavior compared to their counterparts. Finally, the results supported our initial premise that flat measures are not adequate for evaluating hierarchical categorization systems.

Our analysis showed that although in certain rare cases pair-based measures may behave more desirably, in most cases the set-based measure proposed in this thesis (F_{LCA}) exhibits the most desirable behavior, since it is actually a hybrid measure that uses pairing for the selection of the lowest common ancestors. This why we propose the use of F_{LCA} instead of all other hierarchical measures. Nevertheless there seem to be certain cases where the pair-based MGIA methods should be preffered over F_{LCA}.

In then next chapter we will focus on the simplest case of single-label classification on the leaf nodes of a tree hierarchy. In that case, tree-induced error is sufficient as a hierarchical evaluation measure for the experiments that are performed.

Chapter 4

Single-label Leaf Classification on Tree Hierarchies

4.1 Introduction

In most classification problems the predefined categories are assumed to be independent. In hierarchical classification problems, a hierarchy is also given, which contains the relations between the categories. In the simplest case, which we study in this chapter, these relations are of is-a type and the hierarchy is a tree. We also assume that each instance belongs to a single category (single-label classification) and that this category is always a leaf of the tree hierarchy.

Many researchers ignore the hierarchy and treat hierarchical classification of this type using flat classifiers, while others use mildly hierarchical approaches (Kosmopoulos et al., 2010). In most flat approaches, a binary classifier is trained for each category, using all the instances belonging to that category as positive examples and all or some of the other instances as negative examples (one-versus-all). The simplest form of hierarchical classification is that of cascade classification, where a binary classifier is trained for each node of the hierarchy, in order to separate it from its siblings. Then each test

instance is guided through the hierarchy from the root to a leaf, choosing each time the most probable child class (node).

In large scale hierarchical classification problems, the number of training instances and features can be very high (thousands or even millions). A flat classification approach can deal with the high dimensionality by performing instance and/or feature selection for each one-versus-all classifier. For a hierarchical cascade classifier, however, feature selection is more complicated. Classifiers at the upper levels of the hierarchy need to deal with instances of all of their numerous descendant classes and any kind of intense feature selection could lead to a situation where many test instances cannot be represented adequately by the selected features. For example, in text classification with binary bag-of-word features (each indicating if a particular word is present in a text or not), there may be many test texts (belonging to very different descendant categories) that do not contain any of the words corresponding to the selected features of the upper level classifiers. These texts will have identical (all-zero) feature vectors and, hence, the upper-level classifiers will be unable to distinguish them. Since intense feature selection is impossible in the upper levels of the hierarchy, classifier training can be very computationally expensive, because both the number of training instances and the number of features is high.

In order to facilitate hierarchical classification, we examine the use of a principal component (PCA) transformation, reducing the dimensionality at the top levels of the hierarchy. In this way, hierarchical classifiers can be trained on much fewer dimensions, leading to faster training and testing and to lower memory demands. At the same time, the use of a linear transformation of the initial feature set, instead of a discrete feature selection, reduces the risk that test instances will not be represented in the new space.

However, the use of PCA is also not without difficulties. First, one needs to use a version of PCA that can handle the scale of the data. In this chapter we select one

such method and show that it can be used for large-scale hierarchical classification.[1] Additionally, we study the effect of dimensionality reduction on the computational performance of the classifier and its classification accuracy. In particular, we show that the combination of classifiers trained on a reduced feature space at the top levels of the hierarchy with classifiers trained on the original space at the lower levels provides the highest benefit for the lowest cost. We experiment with two popular hierarchical text classification datasets, but our approach should be useful in any hierarchical classification problem with many dimensions and sparse feature vectors.

Regardless of dimensionality reduction, cascade classification has an important limitation, any mistake done during the descent leads to the wrong final decision. Therefore the cascade is very sensitive to the quality of the inner node classifiers. In this chapter we also propose a new approach, called probabilistic cascading, which is as fast as cascade regarding training but leads to better results compared to cascade and flat classification, using the same classification algorithms. [2]

In Section 4.2, we present the proposed approaches and the related work. Section 4.3 shows experimentally the effect of our approach on the computational cost and the accuracy of the hierarchical classifiers. Finally Section 4.4 concludes and points to future work.

[1] Our work regarding the effect of dimensionality reduction in large scale hierarchical classification was presented in the main conference of CLEF 2014 (Kosmopoulos et al., 2014a).

[2] Our work regarding probabilistic cascading is alos available as a technical report (Kosmopoulos et al., 2015b).

4.2 Methods

4.2.1 Cascade Classification with PCA

In cascade classification a classifier must be trained for each node of the hierarchy. In this chapter we focus on tree hierarchies, where instances belong only to the leaves of the hierarchy. An example of such a hierarchy is presented in Figure 4.1. In this example, a text classifier must be trained for each of the following nodes: *Arts*, *Health*, *Music*, *Dance*, *Fitness* and *Medicine*. A classifier of a node U is trained with all instances belonging to the leaf descendants of U as positive examples and all instances belonging to the leaf descendants of the siblings of U as negative ones. The classifier of node *Arts*, for example, would use instances of *Music* and *Dance* as positive examples and instances of *Fitness* and *Medicine* as negative ones.

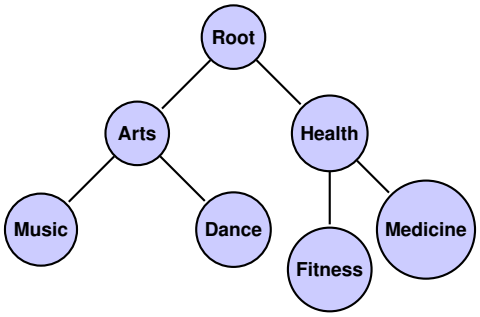


Figure 4.1: Tree hierarchy example.

Assuming again bag-of-word features, if we do not perform any feature selection the classifier of node Arts would use features for all the words of its positive and negative instances. In large datasets, where the hierarchy is composed of thousands of categories, the number of initial features can be hundreds of thousands (or even millions in text classification, if stemming or other similar preprocessing techniques are not used). Training a classifier for each node of the hierarchy using all these features can be very computationally demanding. On the other hand, an intense feature selection

at the upper levels of the hierarchy would lead to inaccurate classifiers, since the few selected features would be unlikely to represent the test instances adequately, as already discussed.

Instead of feature selection, we suggest dimensionality reduction with PCA applied to each set of siblings of the hierarchy. In Figure 4.1, for example, we would need to perform PCA three times:

- for the nodes *Arts* and *Health*
- for the nodes *Music* and *Dance*
- for the nodes *Fitness* and *Medicine*.

The two classifiers of nodes *Arts* and *Health* would use the same feature space, and similarly for the siblings *Music* and *Dance* and *Fitness* and *Medicine*. Hence, PCA needs to be performed only once for each set of siblings. We note that performing PCA on all the leaves (as if we had a flat classification problem) would require a very large number of principal components, drastically reducing the benefit of applying PCA.

Even applying PCA to sets of siblings, however, is not trivial at the scale that we are considering. In PCA an eigen-decmposition of the covariance matrix YY^T $(p \times p)$ must be performed, where Y is the $p \times n$ matrix of the observed data, p is the number of features, and n the number of instances. A common approach is to perform the eigen-decomposition of YY^T via Singular Value Decomposition (SVD):

$$Y = U\Sigma V^T \tag{4.1}$$

where U is the square $(n \times n)$ matrix whose columns contain the left singular vectors of Y, Σ is a $n \times p$ rectangular diagonal matrix containing the singular values and V is the square $(p \times p)$ matrix whose columns contain the right singular vectors of Y. The left singular vectors (U) of Y are the eigenvectors of YY^T, the right singular vectors (V) of

Y are the eigenvectors of Y^TY, and the non-zero singular values are the square roots of YY^T (and Y^TY).

The number of features (p) and instances (n) can by too large to perform PCA. For this reason, in (Roweis, 1998) an Expectation Maximization (EM) algorithm for PCA is proposed, where the number k of principal components must be set from the beginning. The steps of the EM are the following:

$$\begin{aligned} &\text{E-step: } X = (C^TC)^{-1}C^TY \\ &\text{M-step: } C^{new} = YX^T(XX^T)^{-1} \end{aligned} \tag{4.2}$$

where X is a $k \times n$ matrix and C is $p \times k$ matrix. These quantities are much easier to compute than those of the regular PCA, since k can be set to a much smaller value than p. In order to compute the final eigenvectors and eigenvalues, we only need to project the observed data Y to the orthonormal basis for the range of matrix C ($orth(C)^TY$) and perform a regular PCA in this k-dimensional subspace.

Adopting this approach, we can perform PCA in very large datasets, where normal PCA would be very computationally demanding, especially in terms of memory. Even with this approach, PCA remains computationally expensive, but in practice it only needs to be performed once per dataset, greatly reducing the time needed to perform subsequent experiments with many different classifiers.

The only disadvantage is that we need to choose the value of k (number of principal components) prior to performing PCA. In Section 4.3 we present results which show that by using only a few hundreds of principal components one can achieve similar results as when thousands of initial features are used.

4.2.2 Related Approaches

Various other linear and nonlinear dimensionality reduction approaches exist (van der Maaten et al., 2009). In this work we focus on linear approaches, because of the large scale factor. Another linear approach that we could use is that of Simple PCA (Partridge

and Calvo, 1997), but we chose EM since the principal components in Simple PCA are calculated approximately.

The most similar approach is the extension of the Stochastic Gradient Ascent (SGA) neural network, proposed in (Oja, 1992). The disadvantage of this method compared to EM PCA is that it requires a rate parameter to be set and also converges less quickly. Another approach would be to use the Arnoldi method to compute the SVD of the data matrix (Lehoucq et al., 1998). However the EM approach seems more straightforward. We also examined the idea of using Sparse Principal Component Analysis (Grbovic et al., 2012), which was not suitable, since it was more computationally expensive than the chosen method and our focus here was on solving computational issues. Another suitable approach could be the Fisher vector, which was used for large-scale image search by Perronnin et al. (2010), but we did not examine it further since it is mostly used for images in the bibliography.

4.2.3 Probabilistic Cascading

Although hierarchical classification has many advantages, typically researchers resort to mildly hierarchical or even flat approaches (Kosmopoulos et al., 2010). One reason for this is that flat classification is well studied, so it is easier to transfer methods from this field. On the other hand in large-scale problems, the flat use of traditional classifiers, such as SVMs, is often prohibitively expensive computationally (Liu et al., 2005).

Early work in hierarchical classification focused on approaches such as shrinkage (McCallum et al., 1998) and hierarchical mixture models (Toutanova et al., 2001). Unfortunately most of these approaches cannot be applied to large-scale problems, at least in the form described in the original papers. New methods based on similar ideas, such as that of latent concepts (Qiu et al., 2011), continue to appear in the literature, taking also into account scalability issues. But still most of the proposed methods are tested on rather small datasets with small hierarchies.

Mildly hierarchical approaches, typically make limited use of the hierarchy. Methods such at that of Xue et al. (2008) use only some levels of the hierarchy, flattening the rest. Other approaches such at that of Babbar et al. (2013), alter the initial hierarchy before performing cascading in order to minimize errors at the upper levels of the hierarchy.

In our method, following the cascading approach, we train one binary classifier for each node of the hierarchy, as already discussed. These binary classifiers require fewer resources to be trained compared to training one for each leaf versus all. They can also be more accurate, since they aim to distinguish between fewer categories. For example, if we have 10,000 leaves, each binary classifier would need to separate one class from 9,999 others. In the case of cascading, it would only need to separate between the sibling categories. Such classifiers would also require fewer features to train on, an important characteristic if we consider large datasets.

The main disadvantage of cascading is that any mistake is carried over. For example if an instance belonging to the category Music, gets a higher probability by the classifier of Health than that of Arts, it is classified wrongly, without taking into consideration the classifiers of Music and Dance. In contrast, our method computes the probability of each root-to-leaf path for a testing instance and classifies the instance to the most probable path, which we call P_{path}. Consider, for example, the hierarchy of Figure 4.1. The joint probability that an instance d belongs in both Music and Arts can be written in two ways, using the chain rule of conditional probabilities, as:

$$P(Music, Arts|d) = P(Arts|d) \cdot P(Music|Arts, d) = P(Music|d) \cdot P(Arts|Music, d)$$

Solving for $P(Music|d)$, we obtain:

$$P(Music|d) = \frac{P(Arts|d) \cdot P(Music|Arts, d)}{P(Arts|Music, d)} \tag{4.3}$$

but since $P(Arts|Music, d) = 1$:

$$P(Music|d) = P(Arts|d) \cdot P(Music|Arts, d) \tag{4.4}$$

Similarly:

$$P(Arts|d) = \frac{P(Root|d) \cdot P(Arts|Root,d)}{P(Root|Arts,d)} \tag{4.5}$$

but since $P(|Root|Arts,d) = 1$ and $P(Root|d) = 1$:

$$P(Arts|d) = P(Arts|Root,d) \tag{4.6}$$

By combining (4.4) and (4.6) we get:

$$P(Music|d) = P(Arts|Root,d) \cdot P(Music|Arts,d) \tag{4.7}$$

These conditional probabilities are in fact the ones computed by the binary classifiers of each node. So given a document d, a leaf C and a set S of all the ancestors of C:

$$P(C|d) = \prod_{i=1}^{|S|} P(S_i|Ancestor(S_i),d) \tag{4.8}$$

and we define P_{path} as the:

$$P_{path}(d) = \underset{C}{\operatorname{argmax}}\, P(C|d) \tag{4.9}$$

Let's get back to our initial example where document d belonged to Music. Lets assume that we have the following probabilities:

- $P(Arts|Root,d) = 0.2$
- $P(Health|Root,d) = 0.21$
- $P(Music|Arts,d) = 0.9$
- $P(Dance|Arts,d) = 0.6$
- $P(Fitness|Health,d) = 0.1$

- $P(Medicine|Health, d) = 0.2$

If we used standard cascading, document d would be classified to category Medicine. Using P_{path} we get:

- $P(Music|d) = 0.18$
- $P(Dance|d) = 0.12$
- $P(Fitness|d) = 0.021$
- $P(Medicine|d) = 0.042$

and P_{path} would assign d to class Music. The cost that we have to pay in order to select the path with the highest probability is proportional to the number of classes in the hierarchy, while the cost of standard cascading is proportional to the logarithm of the number of classes.

4.3 Experiments

4.3.1 Experimental Set-up

In order to assess the effect of combining PCA with cascade classification, we used both the dry-run and the large datasets form Task 1 of the first Large Scale Hierarchical Text Classification Challenge (LSHTC1).[3]

The dry-run dataset contains 6,323 instances (split into train and validation files), composed of 55,765 distinct features and belonging to 1,139 categories. An extra set of 1,858 test instances is also provided for evaluation. The large dataset contains 93,505 instances (split into train and validation files), composed of 381,581 distinct features and belonging to 12,294 categories. The test instances in this dataset are 34,880.

[3]http://lshtc.iit.demokritos.gr/node/1

In both datasets, every instance has to be classified in a single leaf of the hierarchy, and the hierarchy is a tree. The systems are evaluated using the evaluation measures of the challenge, which are: Accuracy, Macro F-measure, Macro Precision, Macro Recall and Tree Induced Error.

As statistical significance tests, we used p-test ($p < 0.01$) for accuracy and S-test ($p < 0.01$) for macro F-measure. More information regarding these tests can be found in (Yang and Liu, 1999).

In the experiments we report, we used an L2 Regularized Logistic Regression (Fan et al., 2008), with the regularization parameter C set to 1 (usually the default value). We also conducted experiments with other regularization methods and other values of C, but the results were similar. We experimented with TF and TF-IDF bag-of-word features, but we report mostly experimental results with TF-IDF features, which led to better performance.

For each node of the hierarchy we trained two binary classifiers. One using all the initial features and one using k principal components. R-analysis requires that $k < n$ (Venables and Ripley, 2002), but for computational reasons we set a much smaller value for k. In practice, we observed that in both datasets for values of k greater than a certain point the computational cost increased a lot, without significant gains in terms of classification accuracy. For the dry-run dataset, we set k equal to 390, i.e. 390 components. In cases where $390 > n$ we set k equal to $n-1$ in order to satisfy $k < n$. In Figure 4.2 we present accuracy at the top level of the hierarchy (children classes of the Root node) of the dry-run dataset, for various numbers of principal components. The figure illustrates the decreasing gains in accuracy as the number of components increases. Similarly for the large dataset we set k equal to 490.

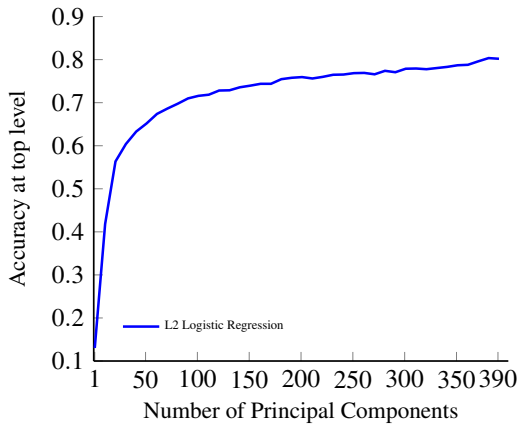


Figure 4.2: Accuracy at top level of the hierarchy of the dry-run dataset, using k Principal Components for various values of k.

4.3.2 Feature Selection Results

In this section we present results showing that feature selection at the top levels of the hierarchy can heavily decrease the accuracy of the classifiers. In Tables 4.1 and 4.2 we present results in terms of accuracy and training time at the top level of the hierarchy of the dry-run and the large dataset, using feature selection. For each category the best features were selected according to the Chi-square statistic χ^2 (Liu and Setiono, 1995). These experiments were conducted using an i7 3.2 GHz CPU (single thread).

Number of Features	Accuracy	Training Time (sec)
55,765 (100%)	0.82	7
27,882 (50%)	0.76	5
5,576 (10%)	0.56	0.84
557 (1%)	0.29	0.18

Table 4.1: Accuracy and training time at the top level of the hierarchy of the dry-run dataset using χ^2 feature selection.

Although the best features are selected for each binary classifier of the top level, many instances cannot be represented adequately by the selected features (empty feature vectors). As a result, in both datasets the accuracy falls significantly with the

Number of Features	Accuracy	Training Time (sec)
381,580 (100%)	0.83	328
190,790 (50%)	0.78	182
38,158 (10%)	0.63	57
3,815 (1%)	0.33	7

Table 4.2: Accuracy and training time at the top level of the hierarchy of the large dataset using χ^2 feature selection.

reduced feature sets. Since these errors at the top level will be carried to the leaves of the hierarchy, any form of intense feature selection will lead to inferior final results compared to keeping all the features. On the other hand, the training times seem to be almost proportional to the number of features. Therefore, we gain in terms of training times, as well as memory requirments, since the size of the training models is correlated to the number of features.

4.3.3 Results on the Dry-run Dataset

Since feature selection is ineffective at the top levels of the hierarchy, we reduce the number of features using PCA. In Table 4.3, we present the results of four different systems on the dry-run dataset, using the five evaluation measures of Section 4.3.1. The first system (*Cascade*) uses all features (TF-IDF) in order to train each binary classifier. The second system (*PCA Cascade*) uses only classifiers trained with PCA features in all levels of the hierarchy (with PCA applied to sibling classes). In the system *Combo Cascade*, the classifiers at the top two levels of the hierarchy are trained using PCA features, while the ones at the lower levels are trained using the initial features. Finally we also provide results of flat classifiers (*Flat*) trained using all the initial features.

The first observation is that the *Cascade* system performs better than all the other systems, including the popular flat classifier. Flat classifiers perform well enough according to the four flat evaluation measures, but they have the worst performance according to the Tree Induced Error (lower number indicates better performance), which

Evaluation Measure	Cascade	PCA Cascade	Combo Cascade	Flat
Accuracy	**0.444***	0.419*	0.436*	0.438*
Macro F-measure	**0.312***	0.276	0.302*	0.304*
Macro Precision	**0.284**	0.250	0.275	**0.284**
Macro Recall	**0.346**	0.307	0.336	0.326
Tree Induced Error	**3.588**	3.754	3.673	3.976

Table 4.3: Results using the dry-run dataset for each approach per evaluation measure, using TF-IDF features. The best performing approach per evaluation measure appears in bold. Since Tree Induced Error is an error rate, lower values are better. Results with no statistically significant difference are marked with a $\star$ symbol.

is the only hierarchical evaluation measure. This means that when the flat classifier fails to predict the exact category, its mistake is further from the correct category compared to the hierarchical systems. This is particularly important in hierarchical classification problems. Another disadvantage of flat classification is that on large-scale problems, the use of traditional classifiers, such as SVMs or Logistic Regression, can be prohibitively expensive computationally (Liu et al., 2005), since a binary classifier must be trained for each leaf using all instances.

Comparing *PCA Cascade* and *Cascade* we see that the latter is more accurate according to all evaluation measures. However, the difference is relatively small and in *PCA Cascade* the classifier is trained with a few hundreds of features instead of a few tens of thousands. Furthermore, we can improve the performance of *PCA Cascade* by using PCA only at the top levels of the hierarchy, where training is most expensive. As can be seen in Table 4.3, *Combo Cascade* achieves classification performance that is less than one precedence point smaller than that of *Cascade*.

Therefore, the combination of PCA at the top levels with training on the original feature space at the lower levels, provides high classification accuracy, while making cascading scalable to large datasets. The choice of the level at which the method should stop reducing the dimensionality of the feature space is largely an issue of computational cost. However, it is important to assess the effect of dimensionality reduction at

each level. The results of this experiment are presented in Table 4.4. At each level of the hierarchy, we assume that the preceding (higher-level) classifiers have predicted the correct category and we measure only the accuracy at the corresponding level. According to the results in Table 4.4, it seems safe to assume that PCA affects classification accuracy similarly in all levels of the hierarchy.

Level of the Hierarchy	Original Features	Reduced Dimensions (PCA)
1	0.820*	0.814*
2	0.819*	0.812*
3	0.820*	0.807*
4	0.856*	0.849*
5	0.840*	0.834*

Table 4.4: Accuracy for cascade classification on the dry-run dataset, using the original and the reduced dimensions (PCA) per level of the hierarchy. Results with no statistically significant difference are marked with a $\star$ symbol.

In section 4.3.1, we mentioned that TF-IDF features provided better results than TF features. In Table 4.5 we present the results for *Cascade*, *PCA Cascade* and *Flat* using TF features. Not only *Cascade* and *Flat* systems performed worse with TF features, but also the *PCA Cascade* was greatly affected. Therefore we advise those who may use the proposed hierarchical PCA approach to perform a TF-IDF transformation.

Evaluation Measure	Cascade	PCA Cascade	Flat
Accuracy	**0.388***	0.361*	0.385*
Macro F-measure	**0.254***	0.221	0.248*
Macro Precision	0.232	0.201	**0.235**
Macro Recall	**0.281**	0.246	0.262
Tree Induced Error	**4.065**	4.356	4.625

Table 4.5: Results for each approach per evaluation measure, using TF features. The best performing approach per evaluation measure appears in bold. Since Tree Induced Error is an error rate, lower values are better. Results with no statistically significant difference are marked with a $\star$ symbol.

4.3.4 Results on the Large Dataset

In order to examine the scalability of our approaches, in Table 4.6 we present results for the large dataset, using TF-IDF features. As in the dry-run data set, *Cascade* uses all features in order to train each binary classifier. In *Combo Cascade* the classifiers at the top level of the hierarchy are trained using PCA features, while the ones at the lower levels using the initial features. Since the initial features are much more than the dry-run dataset (381,581 compared to 55,765), we used 100 more principal components (490) in the *Combo Cascade* system. We also provide results of flat classifiers (*Flat*) trained using all the initial features. Finally, we present the training times of the classifiers in each case.

Evaluation Measure	Cascade	Combo Cascade	Flat
Accuracy	0.404*	0.385	**0.405***
Macro F-measure	**0.278***	0.259	0.256*
Macro Precision	**0.269**	0.249	0.254
Macro Recall	0.289	0.268	**0.302**
Tree Induced Error	**3.609**	3.845	3.874
Training Time	8.66 min	6.2 min	1017.63 min

Table 4.6: Results on the large dataset for each approach per evaluation measure and training time, using TF-IDF features. The best performing approach per evaluation measure appears in bold. Since Tree Induced Error is an error rate, lower values are better. Results with no statistically significant difference are marked with a $\star$ symbol.

According to Accuracy and Macro Recall, *Flat* is somewhat more accurate than *Cascade*, although the p-test detected no statistically significant difference between them in Macro Recall. On the other hand, as in the dry-run dataset, according to the hierarchical evaluation measure (tree induced error) *Cascade* performs better. *Cascade* also performs better in terms of Macro Precision and Macro F-measure, although the S-test for Macro F-measure detected no significant difference. Finally *Combo Cascade*, with PCA applied at the top level of the hierarchy, performs slightly worse. In terms of

training times, *Flat* is very slow compared to *Cascade*. Between *Cascade* and *Combo Cascade*, we observe a speed up of about 30%. Performing PCA at lower levels would only mildly affect speed, since the initial features are much fewer compared to those of the higher levels.

Furthermore, computational cost could also affect the choice of the classifier used. For example, an RBF SVM (Liu et al., 2005) is very expensive at the top level of the hierarchy, if the original feature set is used. However, with PCA the use of such a costly classifier is made possible, as the number of features is reduced from 381,581 to a few hundreds. In Table 4.7 we present the time in minutes required to train L2 logistic regression and SVM with an RBF kernel using the original and the reduced (PCA) features at the top level of the hierarchy of the large dataset.

	Original Features	Reduced Dimensions (PCA)	Gain
L2 Logistic Regression	5.46 min	2.84 min	48%
SVM with RBF kernel	691.2 min	124.1 min	82%

Table 4.7: Training time for L2 logistic regression and RBF SVM using the original and the reduced dimensions (PCA) at the top level of the hierarchy of the large dataset.

Although in both cases the training times are reduced, the gain is much larger for the RBF SVM. In addition to the training time, complex classifiers require more parameter tuning, which is only made possible with the *Combo Cascade* method. In Table 4.8 we present results, using an RBF SVM classifier at the top level of the hierarchy for the *Combo Cascade* method. In one case the SVM is trained using the default parameters, while in the other the parameters have been (non-exhaustively) tuned (c=100, g=0.01). As we can observe the tuned SVM performs much better than the default one. The p-test and S-test detected no statistically significant difference between *Cascade* and *Combo Cascade* with a tuned SVM. Given the training time cost in the initial feature space, this tuning would be much harder without the use of PCA.

Evaluation Measure	Cascade	Combo Cascade with Tuned SVM	Combo Cascade with Default SVM
Accuracy	**0.404***	0.402*	0.377
Macro F-measure	**0.278***	0.274*	0.252
Macro Precision	**0.269**	0.264	0.243
Macro Recall	**0.289**	0.283	0.262
Tree Induced Error	**3.609**	3.616	3.952

Table 4.8: Results using the large dataset for Combo Cascade with an SVM classifier at the top level, using TF-IDF features. Cascade results are repeated for ease of reference. The best performing approach, given each evaluation measure, appears in bold. Since Tree Induced Error is an error rate, lower values are better. Results with no statistically significant difference are marked with a $\star$ symbol.

4.3.5 Probabilistic Cascading Experiments

The goal of this experiment was to illustrate that the probabilistic cascading method can improve the results of flat and cascade classification, using the same algorithm, L2 Regularized Logistic Regression in this case.

In Table 4.9, we present the results of each approach, for each evaluation measure. The main observation is that P_{path} outperforms both Flat and Cascade. Another interesting result is that Flat performs the worst, according to Tree Induced Error. This is an indication that by ignoring the hierarchy (flat classification), the mistakes tend to be located further from the correct category in the hierarchy. This is very important in hierarchical classification, since different mistakes carry different weight. Misclassifying an instance to a sibling of the correct category is a smaller error than if it was classified to a category 5 nodes away. Flat evaluation measures, generally fail to capture this, so tree induced error, being the only hierarchical measure of the five that we use is more suitable for comparing the three approaches.

Given that our hierarchy is a tree and each instance belongs to only a single class, there is no need to take into account more complex hierarchical evaluation measures

and tree induced error is sufficient for safe conclusions.

Evaluation Measure	Flat	Cascade	P_{path}
Accuracy	0.405	0.404	**0.431**
Macro F-measure	0.256	0.278	**0.294**
Macro Precision	0.254	0.269	**0.287**
Macro Recall	**0.302**	0.289	**0.302**
Tree Induced Error	3.874	3.609	**3.437**

Table 4.9: Results for each approach per evaluation measure, using TF-IDF features. With bold we mark the best performing approach, given each evaluation measure.

Both P_{path} and Flat classification produce a probability for each leaf and the highest one is returned as the predicted category. But what if we evaluated the list of categories, ranked according to their probability? In order to obtain such an assessment, in Figure 4.3 we calculate the recall for the K most-probable categories, with K ranging from 1 to 10. As expected the probability of success increases rapidly with K. This is very important, for a realistic semi-automated classification scenario, where a human annotator selects the correct label between thousands of categories. Such a system would allow the annotator to select only between five or ten suggestions. The second observation is that for all values of K, P_{path} performs better than the Flat classifier.

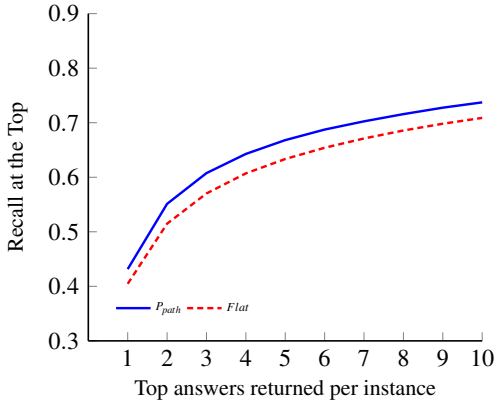


Figure 4.3: Recall at the top K answers of P_{path} and Flat classification for various values of K.

Regarding the scalability of the approaches, the two cascading approaches (standard and P_{path}) require fewer resources than the flat classifiers. During classification, P_{path} is slower than Cascade, since it takes into account all the root-to-leaf paths, and is similar to the cost of Flat classification. Further experimentation and engineering could make the method competitive to the best-performing systems, in the challenge. However, we consider this exercise beyond the scope of this thesis.

4.4 Conclusions

In large-scale hierarchical classification problems the number of features and instances used for training classifiers can be very large at the upper levels of the hierarchy. This can discourage the use of more complex, but probably more accurate classifiers and cause computational issues. In this chapter we examined the use of dimensionality reduction (PCA) for hierarchical classification. This approach is independent of the classifier used and can be applied to all or some nodes of the hierarchy.

Even though performing PCA itself on such large-scale datasets is not trivial, there are methods that can handle the complexity. We also showed experimentally that, although applying PCA to all levels of the hierarchy can decrease accuracy to some extent, this effect can be drastically limited, if we apply PCA only to the upper levels. It would also be interesting to adaptively select the nodes were PCA should be used, instead of just applying it to the upper levels. We plan to examine such an adaptive selection method in the future.

The dimensionality reduction of the PCA procedure allows the use of more complex and possibly more accurate classifiers, such as non-linear SVMs. As future work, we also plan to compare the presented results with that of better tuned SVM classifiers. These classifiers are easier to train, using the reduced feature space provided by the PCA approach.

Furthermore, we presented our probabilistic cascading method (P_{path}) for hierarchical classification. P_{path} addresses the disadvantages of traditional flat and cascade classification. Flat classification can be very computational demanding in large scale problems and also ignores completely the hierarchy information which can be exploited for better results. Standard cascading on the other hand is much more efficient computationally, but suffers from the problem of early misclassification at the top levels of the hierarchy. Our approach has the same training computational complexity as standard cascading, while achieving better scores according to all the tested evaluation measures. However, it is slower during classification, having a complexity similar to that of flat classification.

Although these approaches are currently designed for tree hierarchies, in the future we plan to extend them to DAG hierarchies. Furthermore, we focused on single-label classification and while the ideas of P_{path} and EM PCA seem compatible with multi-label approaches, further experiments need to be conducted in this direction.

In the next chapter, we deal with a more complex problem, by experimenting with the biomedical datasets of the BioASQ project. These datasets are larger in terms of documents and classes and their hierarchy is more complex (can be viewed as a DAG).

Chapter 5

Biomedical Semantic Indexing using Word Embeddings in BioASQ

5.1 Introduction

Curating biomedical articles is an overwhelming task, because of the very large number of articles being published and the specialized knowledge required to understand and organize them. The online biomedical bibliographic database PubMed currently comprises approximately 24 million references and was growing at a rate of approximately 21,500 new articles *per week* in 2013. Figure 5.1 shows the number of biomedical articles indexed by PubMed per year.[1] A common curation task is to manually tag each newly published article with concepts from a controlled biomedical taxonomy, comprising tens of thousands of concepts. The tags and the taxonomy can then be used, for example, in search engines to retrieve articles whose concepts correspond to terms of the query (or their synonyms, hypernyms, etc.), or to organize hierarchically the retrieved articles.[2]

[1] All statistics obtained from Medline trend (`http://dan.corlan.net/medline-trend.html`) in March 2015.

[2] See, for example, GoPubMed (`http://www.gopubmed.com/`).

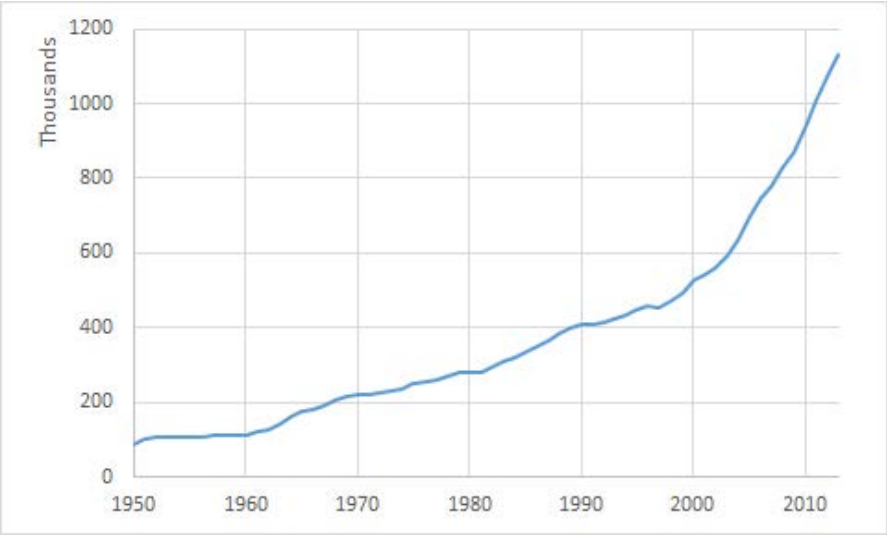


Figure 5.1: Biomedical articles indexed by PubMed per year.

The US National Library of Medicine (NLM), the world's largest biomedical library, employs biomedical experts as curators to index biomedical journal articles by tagging them with *subject headings* (concepts) of the MeSH hierarchy.[3] Figure 5.2 shows a tiny fraction of the MeSH hierarchy, which contains approximately 27,150 subject headings (shown as nodes). Tagging each article with one or more MeSH headings or, more generally, concepts from a taxonomy can be viewed as a problem of hierarchical classification (Silla et al., 2011); in particular a case of multi-label hierarchical text classification (Kosmopoulos et al., 2010). The concepts are viewed as classes and the articles as instances to be classified; the term 'multi-label' indicates that there may be more than one correct concepts (classes) per instance (article) (Tsoumakas and Katakis, 2007). Specialized machine learning algorithms and evaluation measures have been proposed for hierarchical classification and they were discussed in Chapter 3. In Fig. 5.2, for example, mistakenly tagging an article with the heading *Symbolism* instead of *History* is less severe an error than tagging it with *Virus Diseases* and evaluation

[3]See `http://www.nlm.nih.gov/mesh/`. We consider only MeSH 'subject headings' (also known as 'descriptors'), not 'qualifiers' ('subheadings'), as in BioASQ.

measures need to take differences of this kind into account.

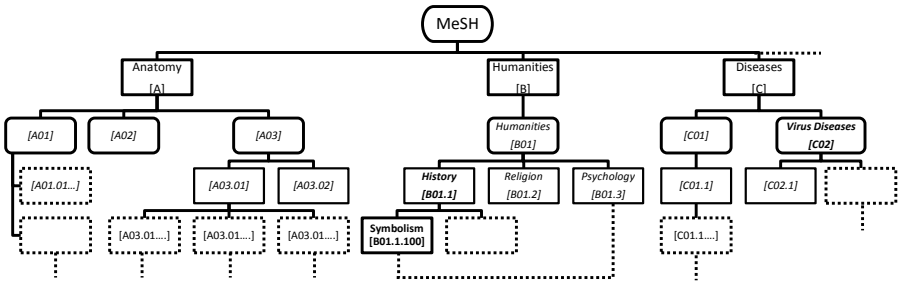


Figure 5.2: A tiny fraction of the MeSH hierarchy. Assigning MeSH subject headings to each article is a case of multi-label hierarchical text classification. Tagging an article with *Symbolism* instead of *History* is less severe an error than tagging it with *Virus Diseases*.

Automatically assigning MeSH subject headings to biomedical articles is also one of the tasks of the annual Biomedical Semantic Indexing and Question Answering challenge (BioASQ, Task A) (Tsatsaronis et al., 2012). In this task, a large corpus of abstracts (and titles) of biomedical journal articles (from PubMed) is provided to the challenge participants as training data, along with the MeSH subject headings the NLM curators assigned to the articles. During the challenge, the participating systems are given the abstracts (and titles) of newly published articles before the NLM curators have assigned them MeSH headings (for most journals, there is a curation backlog of a few weeks).[4] The systems report their decisions, and when the correct MeSH headings (of the curators) are available, the responses of the systems are evaluated. In practice, helping the curators assign subject headings in a semi-automated manner (e.g., by proposing possibly relevant headings) is a more realistic goal. The NLM has developed

[4]The PubMed identifiers of the articles are also provided. Hence, the participants may also consider the full text of the articles when it is available (e.g., via PubMed Central), but most (if not all) of the participating systems use only the abstracts.

a system, the Medical Text Indexer (MTI), which is used for this purpose.[5] MTI and its results are available to BioASQ participants as a baseline and a system to build upon.

Most hierarchical classification systems employ either *flat* classifiers or *cascades* of classifiers, as already discussed in Chapter 2. Flat classifiers (Madani and Huang, 2010; Cissé et al., 2011; Mao et al., 2014) ignore the hierarchical relations between the classes (concepts), i.e., treat the classification problem as a non-hierarchical one, with as many (unrelated) classes as the number of nodes in the hierarchy. Cascades train a separate classifier for each node and usually operate in a top-down manner (Xue et al., 2008; Liu et al., 2005). Starting from the root, the classifier of each node decides if the instance (article) being classified should be passed on to any of the daughter nodes (and which ones), or if the instance should be classified as belonging in the current node. When classifying texts, both flat classifiers and cascades usually represent each text as a bag-of-words (BOW), i.e., a vector whose components (viewed as *features*) show which words are present in the text (for Boolean features), or the TF-IDF (or other similar) scores of the words in the text, in both cases ignoring word order.[6]

Unfortunately, in large-scale hierarchical text classification BOW representations lead to millions of features, one for each vocabulary word (e.g., a word with at least a minimum number of occurrences in the training data) (Yuan et al., 2012). Standard feature selection techniques (e.g., Information Gain, χ^2 (Forman, 2003)) cannot be used to reduce significantly the number of BOW features, because in practice at least a few features (for characteristic vocabulary words) are needed per class, and there are thousands of classes (approximately 27,150 in our case). Indeed, the best performing flat classifiers of BioASQ use BOW features without feature selection (Tsoumakas et al., 2013). In top-down cascades with BOW representations, feature selection at the top nodes of the hierarchy quickly degrades performance, as already discussed in Chapter 4, because

[5] Consult `http://ii.nlm.nih.gov/MTI/`.

[6] Features corresponding to n-grams (Tsoumakas et al., 2013) are rarely used.

large numbers of features (vocabulary words) are needed to capture the very large topic variety of the texts those classifiers handle, and reducing the number of features quickly leads to indistinguishable instances (e.g., all-zero feature vectors) belonging in different daughter nodes. Methods like Principal Components Analysis (PCA) (Jolliffe, 2002), which can be used to map the original vector space to a space of lower dimensionality (intuitively constructing new features for combinations of the original ones), can also not be applied to large scale hierarchical text classification, because of their computational complexity (e.g., PCA requires a costly eigendecomposition) (Kosmopoulos et al., 2014a). In Chapter 4, we proposed a PCA version efficient enough to be applied to the upper levels of a top-down cascade, but it decreased the cascade's accuracy. Furthermore, that PCA version is only applicable to tree concept hierarchies, but the MeSH hierarchy is not a tree (a node can have more then one parents).

Another problem that is also related to the representation of text documents is that similar words (e.g., singular and plural forms, synonyms) are treated as completely different (they are represented by different features). Stemmers are often used to group morphologically related words (e.g., convert 'proteins' to 'protein'), but standard stemmers perform poorly in biomedical texts (Han et al., 2006). Biomedical thesauri (e.g., UMLS[7]) and tools that identify and normalize biomedical terms (e.g., MetaMap[8]) can help group (e.g., represent by the same feature) closely related terms (e.g., synonyms), possibly increasing classification accuracy (Zhu et al., 2013), but they do not decrease significantly the vocabulary size (number of BOW features). Very large numbers of features often lead to training and classification times that are prohibitively large for practical applications, even if implementations of machine learning algorithms optimized for (typically very sparse) BOW feature vectors are used.[9]

[7]Consult `http://www.nlm.nih.gov/research/umls/`.

[8]Consult `http://metamap.nlm.nih.gov/`.

[9]In our experiments, we use implementations optimized for each type of features. Large feature sets may also lead to over-fitting, but we do not study this issue here.

In this chapter we examine the use of dense word vectors known as *word embeddings*, as an efficient method of dimensionality reduction that makes hierarchical text classification algorithms more scalable in biomedical semantic indexing, without being less effective than the usual BOW representation. We consider several approaches for the transition from dense word vectors to dense vectors that represent entire texts, proposing the approach that we believe fits better this domain.[10]

The remainder of this chapter is organized as follows. Section 5.2 presents the methods of our work, while Section 5.3 describes our experiments. Finally, Section 5.4 concludes and summarizes remaining open issues.

5.2 Methods

5.2.1 Dense Word Vectors

In BOW representations, each word of the vocabulary is in effect represented as a 'one-hot' vector with as many components (features) as the size of the vocabulary (millions of components in our case), and only one non-zero component (corresponding to the particular word). When Boolean features are used, the representation of a text is the sum of the vectors of its words, typically a very sparse vector (mostly zero components). In recent years, several methods have been proposed (Bengio et al., 2003; Mikolov et al., 2013b; Pennington et al., 2014; Levy and Goldberg, 2014a; Levy and Goldberg, 2014b) that map each vocabulary word to a *dense vector* (mostly non-zero components) of a space with a much lower dimensionality (often 100–300 dimensions in practical applications), so that words that occur in similar contexts in a large corpus are mapped to similar vectors (e.g., in terms of cosine similarity). Vectors of this kind, also known as *continuous space word vectors* and *word embeddings* have been found to capture morpho-syntactic and semantic properties of the corresponding words (Mikolov et al.,

[10]This work has been accepted for publication (Kosmopoulos et al., 2015a).

2013c).

We show experimentally in this chapter that dense word vectors with as few as 200 features (components) allow performing large-scale biomedical semantic indexing as effectively as (and in some cases better than) with BOW vectors of millions of features, significantly reducing training and classification times. Previous work has already used dense vectors in flat text classification tasks (e.g., sentiment classification (Maas and Ng, 2010; Mikolov et al., 2013b)) and other biomedical text processing tasks (e.g., biomedical named entity recognition (Tang et al., 2014)), but not in hierarchical text classification and biomedical semantic indexing.

To obtain dense word vectors, we applied *word2vec* (Mikolov et al., 2013b) to the 10,876,004 English abstracts of biomedical articles that were provided as training data in the first year of the BioASQ semantic indexing challenge. We used the 'skip-gram' model of word2vec, which can be efficiently applied to very large corpora (Mikolov et al., 2013b; Mikolov et al., 2013a).[11] Roughly speaking, the model maps each vocabulary word w to a dense vector $\vec{w}$ that can be used to predict the (dense vectors of) the other vocabulary words w' that are likely to occur near w.[12] We set the dimensionality of the dense vectors to 200, since previous work shows that 100–200 dimensions lead to reasonably good vectors (Dal et al., 2014). Punctuation symbols, brackets etc. were removed and all letters were converted to lower case. All words appearing in fewer than five abstracts were ignored. The resulting dense vectors of 1,701,632 distinct words (the vocabulary) are available.[13] Constructing them took approximately 3 hours.[14]

As a preliminary investigation of the value of the constructed dense vectors, we identified the 100 most frequent words (excluding stop-words) of the 310 biomedical

[11] Available at `https://code.google.com/p/word2vec/`.

[12] We use the skip-gram model with negative sampling and default parameters.

[13] Available at `http://participants-area.bioasq.org/info/BioASQword2vec/`.

[14] A server with an Intel Xeon 2.7 GHz CPU and 64 GB RAM was used. The word2vec implementation we used was single-threaded.

questions that were provided in the first year of the BioASQ question answering task (Task B).[15] For each of these 100 words, we selected its three closest words (among the 1,701,632), using the cosine similarity of the corresponding dense word vectors as a measure of word proximity. We then showed the 100 words and the closest three words of each one to two biomedical experts (members of the BioASQ biomedical experts group). For each of the 100 words, the two experts were asked to jointly mark the three closest words as relevant (closely related), possibly relevant, or irrelevant. Table 5.1 shows the 30 most frequent of the 100 words and the verdicts of the experts.

protein	**proteins**	**a-anchoring**	**pka-anchoring**
thyroid	**thyroidal**	**nonthyroid**	hyperfunctioning
associated	**correlated**	**related**	**correlates**
hormone	**gh**	**luetinizing**	**fshluteinizing**
human	murine	mouse	~~immortalized~~
used	**utilized**	**employed**	**applied**
genes	**gene**	**paralogs**	**operons**
treatment	**therapy**	**treatments**	**treating**
disease	**diseases**	disease-like	~~mmrn1rs6532197~~
gene	**genes**	**pseudogene**	**gene-encoding**
heart	**cardiac**	**chf**	**congestive**
role	**roles**	~~plays~~	~~play~~
affect	alter	modify	impair
dna	**dnas**	bisulfite-treated	polymerase-mediated
histone	**histones**	**h4k16**	**h4**
involved	**implicated**	**participates**	**regulating**
list	**lists**	**listing**	to-do
proteins	**protein**	**polypeptides**	**hsp70s**
known	~~yet~~	~~presently~~	well-known
patients	**outpatients**	**subjects**	whom
present	this	~~aimed~~	~~our~~
cancer	**cancers**	**crc**	~~caner~~
receptor	**receptors**	**hmc5**	5-nonyloxytryptamine
regulate	**modulate**	**regulates**	**orchestrate**
cell	**cells**	**cancer-cell**	**sw1710**
coding	**5-noncoding**	**5-untranslated**	**3-noncoding**
inhibitors	**inhibitor**	**small-molecule**	**atp-competing**
many	**several**	**some**	**numerous**
related	**linked**	**associated**	**relate**
cardiomyopathy	**cardiomyopathies**	**myocardiopathy**	**dcm**

Table 5.1: Closest words to the 30 most frequent words of the BioASQ question answering task, using the cosine similarity of the dense vectors to measure proximity. Relevant (closely related) words are shown in bold, possibly relevant in normal font, and irrelevant (or misspelled) words in strikeout.

Most of the closest words were found to be closely relevant. Notice that the closest words include several synonyms, semantically, and morphologically related terms (e.g., 'treatment'–'therapy', 'heart'–'cardiac', 'thyroid'–'thyroidal').

[15] Task B uses benchmark datasets containing development and test questions, constructed by a team of biomedical experts, along with gold standard (reference) answers.

5.2.2 Dense Vectors for Abstracts

The simplest approach to obtain a dense vector for an entire text is to compute the centroid of the dense vectors of its words. More elaborate methods to obtain dense vectors for entire texts have also been proposed (Le and Mikolov, 2014; Dal et al., 2014) and are discussed below. Having computed the dense vectors of all the vocabulary words, the simplest method to obtain a dense vector $\vec{t}$ (of the same dimensionality) for a text $t = \langle w_1, w_2, \ldots, w_n \rangle$ of n consecutive word occurrences is to simply compute the *centroid* of the dense vectors $\vec{w}_i$ of the word occurrences:

$$\vec{t} = \frac{1}{n} \sum_{i=1}^{n} \vec{w}_i = \frac{\sum_{j=1}^{|V|} \vec{w}_j \cdot \mathrm{TF}(w_j, t)}{\sum_{j=1}^{|V|} \mathrm{TF}(w_j, t)} \tag{5.1}$$

where $|V|$ is the number of (distinct) words in the vocabulary, and $\mathrm{TF}(w_j, t)$ is the term frequency (number of occurrences) of the j-th vocabulary word in the text t.

Preliminary experiments that we performed indicated improved classification results by including the inverse document frequencies $\mathrm{IDF}(w_j)$ in the centroids of the texts (in our case, abstracts), as follows:

$$\vec{t} = \frac{\sum_{j=1}^{|V|} \vec{w}_j \cdot \mathrm{TF}(w_j, t) \cdot \mathrm{IDF}(w_j)}{\sum_{j=1}^{|V|} \mathrm{TF}(w_j, t) \cdot \mathrm{IDF}(w_j)} \tag{5.2}$$

where $\mathrm{IDF}(w_j) = \log \frac{|D|}{|D(w_j)|}$, $|D|$ is the total number of abstracts in the dataset we obtained dense word vectors from, and $|D(w_j)|$ is the number of those abstracts that contain the word w_j. In the remainder of this Chapter, we use Eq. 5.2 to compute the centroids of abstracts. Having computed (off-line) the dense vectors of the vocabulary words and the IDF scores, computing the centroids of the approximately 4,000 (unseen) abstracts of a new weekly test set of the BioASQ semantic indexing task takes less than a minute.[16]

[16] On an Intel i7 3.2 GHz CPU with 32 GB RAM, using 8 threads.

More elaborate methods to produce dense vectors for entire texts have also been proposed. Le and Mikolov (Le and Mikolov, 2014) map not only vocabulary words, but also entire paragraphs (or sentences, or texts) to dense *paragraph vectors*, so that the dense vector of each paragraph can be used to predict the (dense vectors of) the paragraph's words; alternatively, the dense vector of each paragraph and the (dense vectors of) sequences of words from the paragraph are required to predict the (dense vector of) the words that follow the sequences. In both models, each paragraph vector can be thought of as a representation of the topics of the corresponding paragraph. Le and Mikolov (Le and Mikolov, 2014) reported that the best paragraph vectors were the concatenations of the paragraph vectors of their two models. We used an implementation of their methods, called *sentence2vec*, to produce a paragraph vector for each abstract in our experiments, as an alternative to using abstract centroids.[17] The main limitation of sentence2vec is that it is very slow, given the scale of our datasets. Furthermore, it requires reprocessing the entire training dataset jointly with each new weekly test set of abstracts to produce paragraph vectors for the new abstracts. Dal et al. (2014) have performed experiments to compare paragraph vectors to LDA-based representations (representing each text as a distribution of latent topics) (Blei et al., 2003), BOW vectors, and centroids of dense vectors (without IDF scores). Their experiments showed that Paragraph Vectors in certain cases (i.e. measuring semantic similarity on Wikipedia articles) performed better than the compared approaches, while in other cases they performed less well.

[17] Available at `https://github.com/klb3713/sentence2vec`.

We note, however, that the documentation of sentence2vec is currently very limited, and it is unclear if it follows closely the methods of Le and Mikolov (2014), and exactly which ones. An alternative implementation, called doc2vec (`http://radimrehurek.com/2014/12/doc2vec-tutorial/`), has recently become available.

5.2.3 Large Scale Hierarchical Text Classification

As already noted, several hierarchical text classification systems use flat classifiers, in effect ignoring the hierarchy. In the Large Scale Hierarchical Text Classification (LSHTC) challenges, for example, three of the best performing systems did not use the hierarchy at all (Madani and Huang, 2010; Cissé et al., 2011; Puurula and Bifet, 2012b).[18] Other methods use the hierarchy only to create features for flat classifiers (Han et al., 2011), or use simplified forms of the hierarchy (Xue et al., 2008). Among the systems that use the hierarchy (or a simplified form of it), top-down cascades are the most popular approach (Xue et al., 2008; Liu et al., 2005). They scale well to very large training datasets, compared to other hierarchy-aware classifiers (McCallum et al., 1998; Cai and Hofmann, 2004), but the higher-level classifiers propagate errors to the lower-level ones. Applying top-down cascades is also not straightforward when the hierarchy is not a tree (when a node can have multiple parents) and when texts belong in multiple classes (nodes).

In the BioASQ semantic indexing challenge, the hierarchy of MeSH headings is not a tree and each abstract belongs in 12 classes (nodes) on average. These factors make the successful use of the hierarchy more difficult, and most of the participants so far ignore it. The best system in the first year of the challenge, for example, used a flat Support Vector Machine (SVM) (Tsoumakas et al., 2013). The most popular machine learning algorithms (used as flat classifiers or in cascades) are SVMs (Cortes and Vapnik, 1995; Joachims, 1998), K-nearest neighbors (K-nn) (Jiang et al., 2007), and (less frequently) Naive Bayes (McCallum and Nigam, 1998; Metsis et al., 2006). BOW features are the most popular representation of texts, usually with TF-IDF values.

The Medical Text Indexer (MTI) (Mork et al., 2013) of NLM plays a central role in the BioASQ semantic indexing challenge, both as a baseline and as a component in several of the participating systems. The Default MTI is a high recall rule-based system

[18] Consult `http://lshtc.iit.demokritos.gr`.

that performs very well and is available to all participants. MTI First-Line Indexer (MTIFL), on the other hand, is a high precision system, used to assign preliminary MeSH headings to journal abstracts; its decisions are then reviewed by expert curators. In BioASQ, the performance of MTIFL was lower than that of MTI.

5.2.4 Flat K-nn Classifiers

In most of our experiments, we used flat K-nn classifiers. During training, K-nn classifiers simply store the vector representations of the training instances (in our case, the BOW, dense centroid, or paragraph vectors of the training abstracts), along with their correct classes (the correct MeSH headings). In single-label classification, i.e., when each instance belongs in a single class, K-nn places each test instance (new abstract) into the class that is most frequent in the K training instances (neighbors) that are closest to the test instance, K being a parameter to be tuned. In BioASQ, however, classifying each test instance into the single, most frequent class of its K neighbors would miss many of the correct classes of the test instance. A simple alternative that we adopted, was to consider each class separately, and classify the test instance in the class being considered if at least p (e.g., $p = 50\%$) of the K neighbors belong in the class. As the number of classes increases to thousands, using the same p for all the classes tends to hurt recall, especially for rare classes. Since the main goal of our work was to demonstrate that dense vectors are a good option for large scale biomedical semantic indexing, we decided to ignore the problem of rare classes, but our experimental results confirm that rare classes need to be considered further.

The value of K also greatly affects the behavior of the classifier. As K increases, precision improves, but recall decreases. Rare classes are particularly affected by the value of K. In Fig. 5.3, for example, D is a relatively rare class, since it appears in only 5 of the 21 nearest training instances. Thus, for $K = 21$ it would not be selected if $p = 50\%$, but for $K = 5$ it would be selected (again if $p = 50\%$), since it would

appear in 3 of the 5 nearest training instances. An interesting approach for future work would be to combine K-nn classifiers with different K values in a meta-classifier. The meta-classifier might learn to trust classifiers with small K values for rare classes, and classifiers with large K values for more frequent classes.

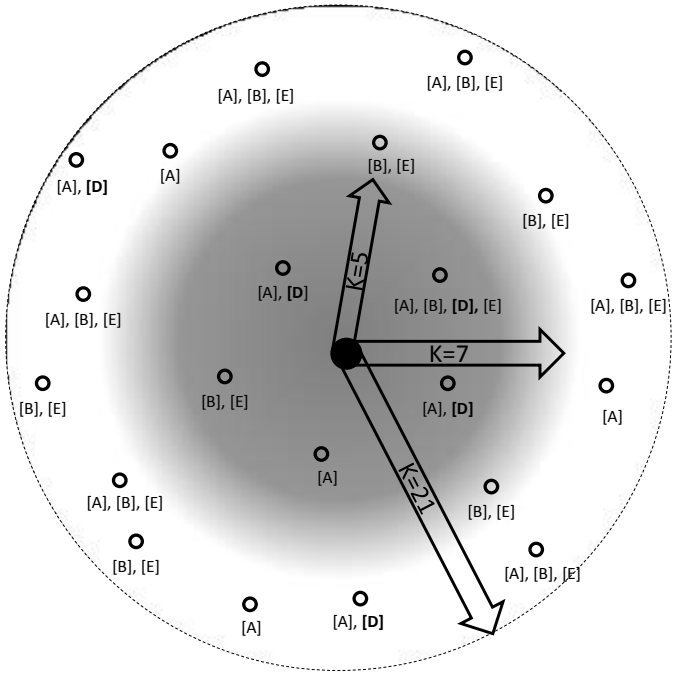


Figure 5.3: Illustration of a K-nn classifier. A test instance is shown at the center, along with its K nearest training instances, for different values of K. The classes of the training instances are shown in square brackets.

In large-scale classification, where there are tens of thousands of classes and millions of instances, K-nn is particularly popular, because its training time is minimal (at the expense of storing all the training instances), whereas other classifiers (e.g., SVMs) require much longer training times. On the other hand, K-nn is slower during prediction (when classifying test instances), since it has to compute the distance between the test instance and every (stored) training instance to obtain the K neighbors. Efficient implementations for sparse vectors (e.g., storing and comparing only non-zero features) can

significantly reduce the prediction time of k-NN when BOW features are used. Dense vectors with small numbers of features also significantly reduce the prediction time. We used implementations optimized for each type of features in our experiments.[19] To compute the distances between training and test instances, we experimented with both Euclidean distance and cosine similarity.

5.2.5 Hierarchically Expanded K-nn Classifiers

We also experimented with a version of K-nn that exploits (still to a small extent) the class hierarchy. This version, which we call *hierarchically expanded K-nn* (HE-K-nn), treats the parents and children of a class C as being identical to C, when examining the K neighbors. Let us assume, for example, that $K = 5$, $p = 50\%$, that only 2 of the nearest 5 training instances belong in class C, and that another training instance among the 5 nearest ones belongs in a parent or child class of C. The simple K-nn would not classify the test instance in C, but HE-K-nn would. A second version of HE-K-nn, treats only the children (not the parents) of the class C as being identical to C.

5.2.6 High Precision Classification

Our experiments show that K-nn can provide precise predictions, especially for frequent classes. On the other hand, Default MTI (hereafter MTI) is a high recall system that performs well for rare classes. Therefore, we decided to combine the two systems. Since MTI already performs well, we aimed to increase its performance by adding to its predictions some classes for which K-nn is very confident, thus increasing MTI's recall further, without losing too much in terms of precision.

The first step in this hybrid MTI/K-nn approach is to select a relatively large value for K. A large K will cause K-nn to miss some of the rare classes, but we hope that MTI

[19] Further improvements are possible by indexing the training instances (Muja and Lowe, 2009).

will compensate for this. The second step is to filter the predicted classes of K-nn using a separate logistic regression classifier for each class (node of the hierarchy).[20] For each predicted class C of K-nn, the corresponding logistic regression classifier decides if the test instance will be classified in C or not. Filtering can be useful, in order to detect classes that were erroneously predicted by the K-nn, because of their high frequency in the datasets. The final step is to add the filtered K-nn predictions to the predictions of MTI.

More precisely, the logistic regression classifier of each class (node) C returns a probability estimate that the test instance belongs in C. If the estimate exceeds a threshold t, we classify the test instance in C. The logistic regression classifier of each class C was trained using the training instances of C as positive examples, and the training instances of the parents, children, and siblings of C as negative examples. When the negative examples were too few (less than 50), training instances from the parents and children of the siblings of C were added as negative examples.

5.3 Experimets

5.3.1 Datasets

All of our experiments were performed on datasets of the BioASQ semantic indexing task. To obtain dense word vectors, we applied word2vec to the 10,876,004 training abstracts of the first year of the task (Table 5.2). This dataset contains 18,560,735 distinct words, but we produced dense vectors only for words that occured in at least five abstracts of this dataset, which led to dense vectors for a vocabulary of 1,701,632 distinct words. The classifiers (K-nn, HE-K-nn, logistic regression) of our experiments

[20] We used the Liblinear implementation (Fan et al., 2008) of logistic regression, with ridge regularization (L2), which performs better than lasso (Tibshirani, 1996). We used logistic regression (to filter the decisions of K-nn), because it is scalable to the size of the data and insensitive to its single parameter.

were trained on the 4,499,338 abstracts of the training dataset of the second year of the BioASQ task; this dataset is smaller than the training dataset of the first year, because it comprises abstracts only from the particular journals the test abstracts of BioASQ come from. In experiments with BOW features, we used features corresponding to the 1,701,632 distinct words we had dense vectors for; we also experimented with BOW features corresponding to the 4,406,024 distinct words of the dataset we used to train the classifiers, but there was hardly any difference in the results. Recall, also, that in experiments with dense vectors, only 200 features are used. For testing, we used the test abstracts of the first five weeks of the second year of BioASQ (approx. 4,500 abstracts per week).[21]

	Number of Abstracts	Distinct Words
Word2vec training dataset	10,876,004	18,560,735
Classifiers training dataset	4,499,338	4,406,024
Week1 test dataset	4,440	98,555
Week2 test dataset	4,721	100,994
Week3 test dataset	4,802	105,654
Week4 test dataset	3,579	84,623
Week5 test dataset	5,299	113,658

Table 5.2: The datasets of our experiments. We applied word2Vec to the training abstracts of the first year of the BioASQ semantic indexing task. The classifiers were trained on the training abstracts of the second year. For testing, we used the test abstracts of the first five weeks of the second year.

In experiments with sentence2vec, we used the 10,876,004 training abstracts of the first year to obtain dense word vectors (as with word2vec), and we then produced paragraph vectors for the 4,499,338 training abstracts of the second year and the 4,440 test abstracts of the first week. We did not experiment with test abstracts of the other four weeks using sentence2vec, mostly because it was very slow.

[21] These five test datasets are collectively known as 'Batch 1' of the second year.

5.3.2 Experiments with Flat K-nn Classifiers

In a first set of experiments, we compared three flat K-nn classifiers that represent the abstracts as centroids (with TF-IDF weights) of dense word vectors, paragraph vectors, and BOW vectors (with TF-IDF values), respectively. We set $p = 50\%$ and $K = 5$ in all three classifiers. In all three classifiers, we used cosine similarity. We tested the classifiers only on the test dataset of the first week (Week 1 of Table 5.2), because some of these experiments were particularly time consuming.[22]

Table 5.3 shows the results of this first set of experiments. The centroids of dense

	Dense TF-IDF Centroids	Paragraph Vectors	TF-IDF Bag-of-words
Micro F_1	**40.8%**	39.9%	38.1%
Micro Precision	62.2%	64.6%	61.9%
Micro Recall	30.4%	28.9%	27.6%
Macro F_1	17.1%	13.3%	**20.1%**
Macro Precision	51.7%	52.8%	59.7%
Macro Recall	15.2%	11.8%	17.9%
Accuracy	**26.3%**	25.3%	24.6%
LCA-F_1	**34.5%**	32.5%	31.8%
LCA Precision	53.7%	54.4%	53.1%
LCA Recall	27.5%	25.3%	24.8%
Classification Time	**50 min**	**50 min**	470 min
Preprocessing Time	**1 min**	9,960 min	**1 min**

Table 5.3: Comparing K-nn classifiers ($K = 5$, $p = 50\%$, cosine similarity) that use centroids of dense word vectors, paragraph vectors, and BOW vectors.

word vectors led to better performance than BOW vectors in micro F_1 and LCA F_1 (BioASQ uses these two measures to decide the winners of the challenge) and also in terms of accuracy. In contrast, BOW vectors were better in macro F_1. It seems that BOW vectors represent rare classes better; hence, they win in macro F_1, which assigns equal importance to rare and frequent classes. Paragraph vectors also performed better than BOW vectors in terms of micro F_1, LCA F_1, and accuracy, and worse than

[22] This and all of the following experiments were performed using an Intel i7 3.2 GHz CPU (8 threads) with 32 GB RAM.

BOW vectors in terms of macro F_1. Compared to dense vector centroids, paragraph vectors performed worse in all four measures (micro F_1, macro F_1, accuracy, LCA F_1). Also, the preprocessing time to construct the paragraph vectors of the test abstracts was prohibitively high (9,960 min), because sentence2vec preprocesses the training abstracts anew, along with the test abstracts, to produce paragraph vectors for the test abstracts. In contrast, it took only one minute to produce the dense centroids or BOW representations of all the test abstracts. Most importantly, classifying all the abstracts of the first test week took only 50 minutes using the dense centroid or paragraph vectors (both use 200 features), but 470 minutes using the BOW representations (1,701,632 features).

In further experiments, we measured the effectiveness of the K-nn classifiers that use dense vector centroids and paragraph vectors, now adopting Euclidean distance, rather than cosine similarity; we used the same test data (Week 1), as in the previous experiments. Cosine similarity is usually considered more appropriate than Euclidean distance for BOW vectors.[23] It is less clear if cosine similarity is also better for dense vectors (centroids of dense vectors and paragraph vectors). Hence we wanted to investigate in these experiments if the effectiveness of the two classifiers that use dense vector representations could be improved further with the use of the Euclidean distance. Table 5.4 shows that Euclidean distance does not improve the overall performance of the classifier that uses dense vector centroids. The micro F_1, macro F_1, accuracy, and LCA-F_1 scores of this classifier were higher with cosine similarity, but the difference was small, and the precision (all three variants) of the classifier was slightly higher with Euclidean distance. In the following experiments, we used the Euclidean distance in the classifier with dense vector centroids, because we aimed to eventually combine a high precision classifier with Default MTI. The overall performance of the classifier

[23]Cosine similarity compares the directions of the vectors, intuitively their topical similarity, whereas Euclidean distance also considers their magnitutes.

	Dense TF-IDF Centroids		Paragraph Vectors	
	Cosine sim.	Euclidean dist.	Cosine sim.	Euclidean dist.
Micro F_1	**40.8%**	40.3%	39.9%	28.6%
Micro Precision	62.2%	*62.6%*	64.6%	66.6%
Micro Recall	30.4%	29.7%	28.9%	18.2%
Macro F_1	**17.1%**	15.9%	13.3%	5.7%
Macro Precision	51.7%	*52.2%*	52.8%	49.7%
Macro Recall	15.2%	13.9%	11.4%	4.5%
Accuracy	**26.3%**	25.8%	25.3%	16.9%
LCA-F_1	**34.5%**	33.6%	32.5%	23.3%
LCA Precision	53.7%	*53.9%*	54.4%	52.5%
LCA Recall	27.5%	26.6%	25.3%	16.4%

Table 5.4: Comparing K-nn classifiers ($K = 5$, $p = 50\%$) that use dense vector representations, with cosine similarity or Euclidean distance.

that uses paragraph vectors was also worse with Euclidean distance, but in this case the deterioration (in micro F_1, macro F_1, accuracy, and LCA-F_1) was surprisingly larger. We did not experiment any further with paragraph vectors and BOW features, given that the dense centroid vectors performed overall better and were computationally more efficient.

Table 5.5 shows the effect of K on the performance of the K-nn classifier that uses dense vector centroids (the best representation of our previous experiments) with Euclidean distance; we used the same test data (Week 1), as in the previous experiments. As the value of K increases, precision (all three variants) also increases, while recall (the corresponding variants) decreases. This is expected due to the requirement that at least half ($p = 50\%$) of the K nearest neighbors must belong in a class C_i, in order to classify the test instance into C_i.

5.3.3 Experiments with Hierarchically Expanded K-nn Classifiers

We also experimented with our two hierarchically expanded versions of K-nn (HE K-nn), using Euclidean distance. Recall that the first expanded version (HE K-nn v1) treats the children and parents of a class C as being identical to C. The second version

	$K=5$	$K=7$	$K=13$	$K=21$
Micro F_1	**40.3%**	39.7%	38.3%	37.3%
Micro Precision	62.6%	66.0%	70.3%	**72.4%**
Micro Recall	**29.7%**	28.4%	26.4%	25.1%
Macro F_1	**15.9%**	14.3%	11.6%	9.9%
Macro Precision	52.2%	56.7%	62.8%	**66.4%**
Macro Recall	**13.9%**	12.2%	9.8%	8.2%
Accuracy	**25.8%**	25.3%	24.2%	23.4%
LCA-F_1	**33.6%**	32.3%	30.4%	29.0%
LCA Precision	53.9%	55.1%	56.6%	**57.2%**
LCA Recall	**26.6%**	24.8%	22.4%	20.9%

Table 5.5: Performance of the K-nn classifier ($p = 50\%$) with dense vector centroids and Euclidean distance, for different values of K.

(HE K-nn v2) treats only the children (not the parents) of C as being identical to C.

Table 5.6 compares the performance of the two hierarchically expanded versions against a non-expanded K-nn classifier, for $K = 13$ (a mid-range value of Table 5.5), using dense vector centroids and Euclidean distance in all cases; we used the same test data as above (Week 1). As expected, the hierarchically expanded versions achieve higher recall at the cost of lower precision. HE K-nn v1 increases recall more than HE K-nn v2, since it expands each class being considered with more classes (both parents and children), at the cost of lower precision. HE K-nn v2 (the more conservative expanded version) had the best overall performance in hierarchical evaluation (LCA F_1), while HE K-nn v1 (the less conservative expanded version) had the best overall performace in flat evaluation (micro F_1, macro F_1, accuracy). The non-expanded K-nn had the best precision scores. Hence, in the following section, where we aimed to combine Default MTI with a high precision classifier, we used the non-expanded K-nn version.

	K-nn 13	HE K-nn 13 v1	HE K-nn 13 v2
Micro F_1	38.3%	**39.4%**	38.7%
Micro Precision	**70.3%**	44.1%	54.4%
Micro Recall	26.4%	**35.6%**	30.1%
Macro F_1	11.6%	**17.6%**	13.4%
Macro Precision	**62.8%**	34.3%	44.4%
Macro Recall	9.8%	**17.6%**	12.1%
Accuracy	24.2%	**24.5%**	24.3%
LCA-F_1	30.4%	33.7%	**34.3%**
LCA Precision	**56.6%**	48.0%	55.2%
LCA Recall	22.4%	**28.5%**	26.8%

Table 5.6: Comparing a flat K-nn classifier to two hierarchically expanded K-nn classifiers, using dense vector centroids, $K = 13$, $p = 50\%$, and Euclidean distance in all cases. HE K-nn v1 treats the children and parents of a class C as being identical to C. HE K-nn v2 treats only the children (not the parents) as being identical to C.

5.3.4 Improving Default MTI

The reader is reminded that we aimed to improve the performance of Default MTI by adding high precision predictions of a K-nn classifier to the predictions of Default MTI, thus increasing its recall without losing much in terms of precision. Table 5.7 shows how the performance of Default MTI changes, when the predicted classes of our flat K-nn ($K = 5, 13, 21$, $p = 50\%$) with dense vector centroids and Euclidean distance are added to the predictions of Default MTI; again, we used the same test data (Week 1). We call MTI/K-nn the combined system; this system does not yet include the logistic regression classifiers that filter the predictions of K-nn. As expected, the best overall results were obtained for large K values ($K = 13$ and $K = 21$). MTI/21-nn was the best performer in terms of micro F_1 (the main non-hierarchical measure of BioASQ) and also in terms of accuracy, whereas MTI/13-nn was the best performer in terms of LCA-F_1 (the main hierarchical measure of BioASQ) and macro F_1, but the differences between MTI/13-nn and MTI/21-nn were very small. The overall improvements of MTI/13-nn and MTI/21-nn (compared to Default MTI) were due to their increased recall, which

	Default MTI	MTI/5-nn	MTI/13-nn	MTI/21-nn
Micro F_1	57.0%	56.7%	57.4%	**57.5%**
Micro precision	**58.3%**	53.8%	56.1%	56.5%
Micro recall	55.9%	**59.9%**	58.8%	58.5%
Macro F_1	48.1%	48.4%	**48.5%**	48.4%
Macro precision	**54.9%**	51.5%	53.9%	54.3%
Macro recall	51.7%	**53.2%**	52.4%	52.2%
Accuracy	40.8%	40.4%	41.1%	**41.2%**
LCA-F_1	48.4%	48.6%	**48.9%**	48.8%
LCA precision	51.5%	48.9%	50.3%	**50.5%**
LCA recall	48.6%	**51.5%**	50.7%	50.5%

Table 5.7: Adding the predictions of a flat K-nn classifier ($K = 5, 13, 21$, $p = 50\%$) that uses dense centroid vectors and Euclidean distance to the predictions of Default MTI.

was obtained at the expense of a smaller decrease in precision. The larger the value of K, the smaller the decrease in precision, but also the smaller the increase in recall.

Table 5.8 shows the effect of adding the logistic regression classifiers (one per hierarchy node) to MTI/K-nn; we call the resulting system MTI/K-nn/LR. We considered only $K = 21$ in these experiments, since the overall performance of MTI/13-nn was very similar to that of MTI/21-nn in Table 5.7; we used the same test data as above (Week 1). Recall that in MTI/K-nn/LR, for each predicted class C of the K-nn classifier, the logistic regression classifier of C returns a probability estimate that the test instance belongs in C. C is added to the predictions of Default MTI if the probability estimate exceeds a threshold t. We experimented with several thresholds, but in the end we set $t = 0.5$ (in all classes), since the results were not particularly sensitive to the choice of t. Unfortunately, the overall effect of the logistic regression classifiers was minimal.

Table 5.9 compares the performance of Default MTI and MTI/21-nn/LR.[24] The scores are now averaged over the five test datasets of Table 5.2 (Weeks 1–5), but the same conclusions hold for each individual week. The scores of the best performing system that participated in the BioASQ semantic indexing task in the same weeks (Weeks

[24]The differences between MTI/21-nn/LR and MTI/21-nn were again minimal.

	MTI/21-nn	MTI/21-nn/LR
Micro F_1	57.5%	**57.6%**
Micro Precision	56.5%	57.1%
Micro Recall	58.5%	58.1%
Macro F_1	**48.4%**	**48.4%**
Macro Precision	54.3%	54.6%
Macro Recall	52.2%	52.1%
Accuracy	41.2%	**41.3%**
LCA-F_1	**48.8%**	**48.8%**
LCA Precision	50.5%	50.7%
LCA Recall	50.5%	50.2%

Table 5.8: The effect of adding logistic regression classifiers (LR, one per node of the hierarchy) to filter the predictions of the K-nn classifier ($K = 21$, $p = 50\%$, dense centroid vectors, Euclidean distance) in the combined MTI/K-nn system.

1–5) are also shown, averaged over the five weeks.[25] The overall scores of MTI/21-nn/LR (micro F_1, macro F_1, accuracy, LCA-F_1) are better than those of Default MTI. In terms of LCA-F_1 (the main hierarchical measure of BioASQ), the score of MTI/21-nn/LR is also close to the score of the best system(s). As shown in Table 5.8, they

	Default MTI	MTI/21-nn/LR	Best System
Micro F_1	57.1%	57.7%	**59.0%**
Micro precision	58.5%	57.4%	59.4%
Micro recall	55.7%	58.1%	58.6%
Macro F_1	48.5%	**48.8%**	43.7%
Macro precision	54.7%	54.4%	59.8%
Macro recall	52.2%	52.4%	44.8%
Accuracy	40.8%	41.3%	**42.3%**
LCA-F_1	48.6%	49.0%	**49.5%**
LCA precision	51.8%	51.0%	51.5%
LCA recall	48.8%	50.3%	50.5%

Table 5.9: Comparing Default MTI to our combined MTI/21-nn/LR system and the winner of the BioASQ semantic indexing task (results averaged over Weeks 1–5 of Batch 1).

[25] The best performer was not the same system in all the weeks.

increased the precision of the combined system, but at the expense of lower recall, and there was a very small or no difference in the overall scores (micro F_1, macro F_1, accuracy, LCA-F_1). Given the complexity of training a separate logistic regression classifier per class, MTI/K-nn seems a better choice than MTI/K-nn/LR for practical purposes.

Interestingly, the best systems seem to also focus on frequent classes. Thus, although the 'Best System' has the highest scores in terms of micro F_1 and LCA-F_1 (the main BioASQ mesures), its macro F_1 is the lowest of the three systems. Hence, the appropriate handling of rare classes remains an open issue and the BioASQ evaluation measures could be extended to highlight it. Given the positive results we obtained with dense word vectors, it would be interesting to study the effect they would have on the performance of the systems that perform well in the BioASQ challenge.

5.4 Conclusions

Using datasets from the BioASQ challenge, we demonstrated that dense word vectors (word embeddings) can lead to very large dimensionality reduction, compared to the usual bag-of-word vectors, making hierarchical text classification algorithms more scalable to biomedical semantic indexing. We experimented with flat K-nn classifiers and hierarchically expanded variants, representing article abstracts either as (TF-IDF weighted) centroids of dense word vectors, or directly as dense paragraph vectors. Our experimental results indicate that despite the very large dimensionality reduction (from millions of features to only 200), the centroids of dense word vectors lead to similar or even better performance, compared to bag-of-word vectors. Paragraph vectors did not perform better than the centroids of dense word vectors, and constructing paragraph vectors was very slow for practical applications (at least using the particular implementation that we had available).

We also showed that adding the predicted classes of a high-precision flat K-nn clas-

sifier with dense vector centroids to the predictions of the Default MTI system can improve the overall performance of the latter, by increasing its recall without losing much in terms of precision. Using additional logistic regression classifiers (one per hierarchy node) to filter the predictions of K-nn in the combined system led to very minor improvements.

To the best of our knowledge, this work is the first to consider dense word (and paragraph) vectors in hierarchical text classification and biomedical semantic indexing. It would be particularly interesting to use dense word vectors in order to improve the results of the best systems participating in the BioASQ semantic indexing task. We believe that most of the participating methods can benefit from the use of dense word vectors; depending on the approach, the benefit could be an increase in performance, dimensionality reduction, or both.

Furthermore, since our K-nn classifiers (and other methods) do not perform well with rare classes, another interesting future direction would be to develop solutions for rare classes in particular. Another direction would be to filter the predictions of Default MTI, instead of only adding classes to its predictions. Finally, more work is needed to exploit the concept hierarchy, since our hierarchically expanded K-nn methods (and most of the BioASQ participants) use the hierarchy to a very limited extent.

Chapter 6

Conclusions

6.1 Thesis Contribution

Large scale hierarchical text classification is the problem of assigning large numbers of documents to a predefined large set of hierarchically organized classes. The documents can be thousands or even millions and each one may belong to one (single-label classification) or more classes (multi-label classification). The taxonomy or hierarchy is composed of thousands of classes, which are connected with each other by parent-child relations. This taxonomy can be used to facilitate the classification, but can also influence the evaluation of the performance of a classification system.

In this thesis we first studied the problem of evaluating the performance of hierarchical classification methods. Specifically, this work abstracted and presented the key points of existing performance measures. We proposed a grouping of the methods into a) *pair-based* and b) *set-based*. In order to model pair-based measures, we introduced a novel generic framework based on flow networks, while for set-based measures we provided a framework based on set operations. Salient features of these measures were stressed and presented under a common formalism.

Another contribution of this work was the proposal of two measures (one for each

group) that address several deficiencies of existing measures. The proposed measures, along with existing ones were assessed in two ways. First, we applied them to selected cases, in order to demonstrate their pros and cons. Second, we studied them empirically on four large datasets based on DMOZ, DBpedia and MEDLINE (BioASQ) with different characteristics (single-label, multi-label tree and DAG hierarchies). The analysis of the results showed that the hierarchical measures behave differently, especially in cases of multi-label data and DAG hierarchies. Also, the two proposed measures have shown a more robust behavior compared to their counterparts. Finally, the results supported our initial premise that flat measures are not adequate for evaluating hierarchical categorization systems.

We then focused on the simplest case of single-label classification at the leaf nodes of a tree hierarchy. In large-scale hierarchical classification problems the number of features and instances to be used for training classifiers can be very large at the upper levels of the hierarchy. This can discourage the use of more complex, but more accurate classifiers and cause computational issues. We examined the use of dimensionality reduction (via PCA) for hierarchical classification. This approach is independent of the classifier used and can be applied to all or some nodes of the hierarchy. We also showed experimentally that, although applying PCA to all levels of the hierarchy can decrease accuracy to some extent, this effect can be drastically limited, if we apply PCA only at the upper levels. Furthermore, we presented our probabilistic cascading method (P_{path}) for hierarchical classification. In contrast to the traditional flat classification method, which is slow on training, P_{path} classifiers are trained in the same fast way as the cascade classifiers, while at the same time, the higher-level classifiers do not propagate errors to the lower-level ones.

We then moved on to a more difficult setting, where a document may belong in multiple classes, not necessarily leaves, and the hierarchy is a DAG. We experimented with biomedical datasets biomedical datasets (BioASQ) and demonstrated that dense

word vectors (word embeddings) can lead to very large dimensionality reduction, compared to the usual bag-of-word vectors, making hierarchical text classification algorithms more scalable to biomedical semantic indexing. We experimented with flat K-nn classifiers and hierarchically expanded K-nn classifiers, representing article abstracts either as (TF-IDF weighted) centroids of dense word vectors, or directly as dense paragraph vectors. Our experimental results indicate that despite the very large dimensionality reduction (from millions of features to only 200), the centroids of dense word vectors lead to similar or even better performance, compared to bag-of-word vectors. We also showed that adding the predicted classes of a high-precision flat K-nn classifier with dense vector centroids to the predictions of the Default MTI system can improve the overall performance of the latter, by improving further its recall without losing much in terms of precision.

In parallel to this thesis, we organized a series of challenges and provided many datasets (LSHTC and BioASQ) to the research community. We also provided useful evaluation infrastructure (oracles), so that different systems can be more easily compared to each other. Finally we developed open-source software for hierarchical evaluation which implements many of the existing hierarchical evaluation measures along with the proposed ones.

6.2 Future Directions

Regarding the problem of evaluating the performance of hierarchical classification methods, two new evaluation measures were proposed, which deal with certain limitations of the existing measures. However, this comes at a computational cost, as already discussed in Section 3.2.3.2, which in some cases may be prohibitive. Thus, optimised, approximate or parallelized versions of the proposed measures would be of particular interest.

The PCA based dimensionality reduction approach for hierarchical classification that we proposed seemed to provide significant benefits for single-label tree classification. Interesting future directions could be to extend it for multi-label classification or DAG hierarchies or even both. The same is true for our probabilistic cascading method, which could also be extended for classification at the inner nodes of the hierarchy (not only leaf classes).

Our work on biomedical semantic indexing focused on demonstrating that dense word vectors (word embeddings) can lead to very large dimensionality reduction, compared to the usual bag-of-word vectors. We believe that many existing biomedical semantic indexing methods can benefit from the use of dense word vectors; depending on the approach, the benefit could be an increase in performance, dimensionality reduction, or both. Furthermore, many approaches do not perform well with rare classes. Thus, another interesting future direction would be to combine these methods with specialised solutions for rare classes. Although the method that we proposed works well with flat classifiers, the exploitation of the concept hierarchy is also a direction that should to be examined.

References

A. Aho, J. Hopcroft, and J. Ullman. 1973. On finding lowest common ancestors in trees. In *Proceedings of the 5th ACM Symposium on Theory of Computing*, pages 253–265, New York, USA.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.

R. Babbar, I. Partalas, É. Gaussier, and M. Amini. 2013. Maximum-margin framework for training data synchronization in large-scale hierarchical classification. In *Proceedings of the 20th International Conference on Neural Information Processing*, pages 336–343, Daegu, Korea.

Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. 2003. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155.

D. M. Blei, A. Y. Ng, and M. I. Jordan. 2003. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022.

H. Blockeel, M. Bruynooghe, S. Dzeroski, J. Ramon, and J. Struyf. 2002. Hierarchical multi-classification. In *Proceedings of the ACM SIGKDD 2002 Workshop on Multi-relational Data Mining*, pages 21–35, Edmonton, Canada.

C. Brouard. 2011. ECHO at the LSHTC Pascal challenge 2. In *Joint ECML/PKDD PASCAL Workshop on Large-Scale Hierarchical Classification (LSHC2)*, pages 49–57, Athens, Greece.

F. Brucker, F. Benites, and E. Sapozhnikova. 2011. An empirical comparison of flat and hierarchical performance measures for multi-label classification with hierarchy extraction. In *Proceedings of the 15th International Conference on Knowledge-based and Intelligent Information and Engineering Systems - Volume Part I*, pages 579–589, Kaiserslautern, Germany.

L. Cai and T. Hofmann. 2004. Hierarchical document categorization with support vector machines. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*, pages 78–87, Washington DC, USA.

L. Cai and T. Hofmann. 2007. Exploiting known taxonomies in learning overlapping concepts. In *International Joint Conferences on Artificial Intelligence*, pages 714–719, Hyderabad, India.

N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. 2006. Incremental algorithms for hierarchical classification. *Journal of Machine Learning Research*, 7:31–54.

M. Cissé, T. Artieres, and P. Gallinari. 2011. Learning efficient error correcting output codes for large hierarchical multi-class problems. In *Proceedings of Joint ECML/PKDD PASCAL Large-Scale Hierarchical Classification Workshop at ECML/PKDD*, pages 37–49, Athens, Greece.

C. Cortes and V. Vapnik. 1995. Support-vector networks. *Machine Learning*, 20(3):273–297.

E. P. Costa, A. C. Lorena, Carvalho, and A. A. Freitas. 2007. A review of performance evaluation measures for hierarchical classifiers. In *Evaluation Methods for Machine Learning II: Papers From the AAAI-2007 Workshop*, pages 1–6, Vancouver, Canada.

A. M. Dal, C. Olah, Q. V. Le, and G. S. Corrado. 2014. Document embedding with paragraph vectors. In *Proceedings of Deep Learning and Representation Learning Workshop at NIPS*, Montreal, Canada.

O. Dekel, J. Keshet, and Y. Singer. 2004. Large margin hierarchical classification. In *Proceedings of the 21st International Conference on Machine Learning*, pages 209–216, Alberta, Canada.

R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. 2008. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874.

G. Forman. 2003. An extensive empirical study of feature selection metrics for text classification. *The Journal of Machine Learning Research*, 3:1289–1305.

M. Grbovic, C. R. Dance, and S. Vucetic. 2012. Sparse principal component analysis with constraints. In *Proceedings of the 26th Conference on Artificial Intelligence*, Toronto, Canada.

B. Han, Z. Obradovic, Z. Hu, C. H. Wu, and S. Vucetic. 2006. Substring selection for biomedical document classification. *Bioinformatics*, 22(17):2136–2142.

X. Han, J. Liu, Z. Shen, and C. Miao. 2011. An optimized k-nearest neighbor algorithm for large scale hierarchical text classification. In *Proceedings of Joint ECML/PKDD PASCAL Large-Scale Hierarchical Classification Workshop at ECML/PKDD*, pages 2–12, Athens, Greece.

X. Han, S. Li, and Z. Shen. 2012. A k-NN method for large scale hierarchical text classification at LSHTC3. In *Discovery Challenge Workshop on Large Scale Hierarchical Classification, ECML/PKDD*, Bristol, UK.

N. Holden and A. A. Freitas. 2006. Hierarchical classification of G-protein-coupled receptors with a PSO/ACO algorithm. In *IEEE Swarm Intelligence Symposium*, pages 77–84, Indianapolis, USA.

P. G. Ipeirotis, L. Gravano, and M. Sahami. 2001. Probe, count, and classify: Categorizing hidden web databases. In *ACM SIGMOD International Conference on Management of Data*, pages 67–78, California, USA.

L. Jiang, Z. Cai, D. Wang, and S. Jiang. 2007. Survey of improving k-nearest-neighbor for classification. In *Proceedings of the 4th Conference on Fuzzy Systems and Knowledge Discovery*, pages 679–683, Haikou, China.

T. Joachims. 1998. *Text categorization with support vector machines: Learning with many relevant features*. Springer.

I. Jolliffe. 2002. *Principal component analysis*. Wiley Online Library.

M. G. Kendall. 1938. A new measure of rank correlation. *Biometrika*, pages 81–93.

S. Kiritchenko, S Matwin, and A. Famili. 2005. Functional annotation of genes using hierarchical text categorization. In *ACL Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*, Michigan, USA.

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

A. Kosmopoulos, G. Paliouras, and I. Androutsopoulos. 2014a. The effect of dimensionality reduction on large scale hierarchical classification. In *Proceedings of the 5th Conference and Labs of the Evaluation Forum*, pages 160–171, Sheffield, UK.

A. Kosmopoulos, I. Partalas, E. Gaussier, G. Paliouras, and I. Androutsopoulos. 2014b. Evaluation measures for hierarchical classification: a unified view and novel approaches. *Data Mining and Knowledge Discovery*, pages 1–46.

A. Kosmopoulos, I. Androutsopoulos, and G. Paliouras. 2015a. Biomedical semantic indexing using dense word vectors in BioASQ. *Journal Of Bio-Medical Semantics, Supplement On Bio-Medical Information Retrieval*.

A. Kosmopoulos, G. Paliouras, and I. Androutsopoulos. 2015b. Probabilistic cascading for large scale hierarchical classification. In *Technical report, arXiv preprint arXiv:1505.02251*.

Q. Le and T. Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning*, pages 1188–1196, Beijing, China.

D. Lee. 2012. Multi-stage rocchio classification for large-scale multi-labeled text data. In *Discovery Challenge Workshop on Large Scale Hierarchical Classification, ECML/PKDD*, Bristol, UK.

R. B. Lehoucq, D. C. Sorensen, and C. Yang. 1998. ARPACK users' guide: Solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods. *Software, Environments, and Tools 6.*

O. Levy and Y. Goldberg. 2014a. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the 18th Conference on Computational Natural Language Learning*, pages 171–180, Ann Arbor, Michigan.

O. Levy and Y. Goldberg. 2014b. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, pages 2177–2185, Montreal, Canada.

H. Liu and R. Setiono. 1995. Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of the 7th International Conference on Tools with Artificial Intelligence*, pages 388–388, Virginia, USA.

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

A. L. Maas and A. Y. Ng. 2010. A probabilistic model for semantic word vectors. In *Proceedings of Deep Learning and Unsupervised Feature Learning Workshop at NIPS*, Whistler, Canada.

O. Madani and J. Huang. 2010. Large-scale many-class prediction via flat techniques. In *Proceedings of Large-Scale Hierarchical Classification Workshop at ECIR*, Milton Keynes, UK.

Y. Mao, C. Wei, and Z. Lu. 2014. Ncbi at the 2014 bioasq challenge task: Large-scale biomedical semantic indexing and question answering. In *Proceedings of Question Answering Lab at CLEF*, Sheffield, UK.

A. McCallum and K. Nigam. 1998. A comparison of event models for naive bayes text classification. In *Proceedings of Workshop on Learning for Text Categorization at AAAI*, volume 752, pages 41–48, Wisconsin, USA.

A. McCallum, R. Rosenfeld, T. M. Mitchell, and A. Y. Ng. 1998. Improving text classification by shrinkage in a hierarchy of classes. In *Proceedings of the 15th International Conference on Machine Learning*, pages 359–367, Madison, USA.

V. Metsis, I. Androutsopoulos, and G. Paliouras. 2006. Spam filtering with naive bayes-which naive bayes? In *Proceedings of 3rd Conference on Email and Anti-Spam*, pages 27–28, Mountain View, USA.

Y. Miao and X. Qiu. 2009. Hierarchical centroid-based classifier for large scale text classification. In *Proceedings of the 1st Large Scale Hierarchical Text classification (LSHTC) Pascal Challenge*, volume 18, Milton Keynes, UK.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. 2013a. Efficient estimation of word representations in vector space. In *Proceedings of Workshop Track at ICLR*, Scottsdale, USA.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Proceedings of 27th Conference on Neural Information Processing Systems*, pages 3111–3119, Harrah's LAke Tahoe, USA.

T. Mikolov, W. Yih, and G. Zweig. 2013c. Linguistic regularities in continuous space word representations. In *Proceedings of 13th Conference of the North American Chapter of Association for Computational Linguistics*, pages 746–751, Atlanta, US.

J. G. Mork, A. Jimeno-Yepes, and A. R. Aronson. 2013. The NLM medical text indexer system for indexing biomedical literature. In *Proceedings of the 1st Workshop on Biomedical Semantic Indexing adn Question Answering at CLEF*, Valencia, Spain.

M. Muja and D. G. Lowe. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proceedings of the 4th Conference on Computer Vision Theory and Applications*, Lisboa, Portugal.

S. Nowak, H. Lukashevich, P. Dunker, and S. Rüger. 2010. Performance measures for multilabel evaluation: a case study in the area of image classification. In *Proceedings of the International Conference on Multimedia Information Retrieval*, pages 35–44.

E. Oja. 1992. Principal components, minor components, and linear neural networks. *Neural Networks*, 5(6):927–935.

I. Partalas, A. Kosmopoulos, N. Baskiotis, T. Artieres, G. Paliouras, E. Gaussier, I. Androutsopoulos, M. Amini, and P. Galinari. 2015. LSHTC: A benchmark for large-scale text classification. In *Technical report, arXiv preprint arXiv:1503.08581*.

M. Partridge and R. Calvo. 1997. Fast dimensionality reduction and simple PCA. *Intelligent data analysis*, 2(3):292–298.

J. Pennington, R. Socher, and C. Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, Doha, Qatar.

F. Perronnin, Y. Liu, J. Sánchez, and H. Poirier. 2010. Large-scale image retrieval with compressed Fisher vectors. In *Proceedings of the 23rd IEEE Conference on Computer Vision and Pattern Recognition*, pages 3384–3391, San Francisco, USA.

A. Puurula and A. Bifet. 2012a. Ensembles of sparse multinomial classifiers for scalable text classification. In *Discovery Challenge Workshop on Large Scale Hierarchical Classification, ECML/PKDD*, Bristol, UK.

A. Puurula and A. Bifet. 2012b. Ensembles of sparse multinomial classifiers for scalable text classification. In *Proceedings of Large-Scale Hierarchical Classification Workshop at ECML/PKDD-PASCAL*, Bristol, UK.

X. Qiu, X. Huang, Z. Liu, and J. Zhou. 2011. Hierarchical text classification with latent concepts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 598–602, Oregon, USA.

S. Roweis. 1998. EM algorithms for PCA and SPCA. *Advances in Neural Information Processing Systems*, pages 626–632.

Y. Sasaki and D. Weissenbacher. 2012. TTI's system for the LSHTC3 challenge. In *Discovery Challenge Workshop on Large Scale Hierarchical Classification, ECML/PKDD*, Bristol, UK.

J. Silla, N. Carlos, and A. A. Freitas. 2011. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72.

M. Sokolova and Lapal G. 2009. A systematic analysis of performance measures for classification tasks. *Information Processing Management*, 45:427–437.

J. Struyf, S. Dzeroski, H. Blockeel, and A. Clare. 2005. Hierarchical multi-classification with predictive clustering trees in functional genomics. In *Progress in Artificial Intelligence*, volume 3808, pages 272–283.

A. Sun and E. Lim. 2001. Hierarchical text classification and evaluation. In *Proceedings of the IEEE International Conference on Data Mining*, pages 521–528, California, USA.

A. Sun, E. Lim, and W. Ng. 2003. Performance measurement framework for hierarchical text classification. *Journal of the American Society for Information Science and Technology*, 54:1014–1028.

B. Tang, H. Cao, X. Wang, Q. Chen, and H. Xu. 2014. Evaluating word representation features in biomedical named entity recognition tasks. *BioMed Research International*, 2014.

R. Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

K. Toutanova, F. Chen, K. Popat, and T. Hofmann. 2001. Text classification in a hierarchical mixture model for small training sets. In *Proceedings of the 10th International Conference on Information and Knowledge Management*, pages 105–112, Atlanta, USA.

G. Tsatsaronis, M. Schroeder, G. Paliouras, Y. Almirantis, I. Androutsopoulos, E. Gaussier, P. Gallinari, T. Artieres, M. Alvers, M. Zschunke, and A. Ngonga. 2012. BioASQ: A challenge on large-scale biomedical semantic indexing and question answering. In *Proceedings of the AAAI Fall Symposium on Information Retrieval and Knowledge Discovery in Biomedical Text*, pages 92–98, Arlington, VA, USA.

G. Tsoumakas and G. Katakis. 2007. Multi-label classification: An overview. *Int J Data Warehousing and Mining*, 2007:1–13.

G. Tsoumakas and I. Vlahavas. 2007. Random k-labelsets: An ensemble method for multilabel classification. In *Machine Learning: ECML 2007*, volume 4701, pages 406–417, Warsaw, Poland.

G. Tsoumakas, M. Laliotis, N. Markantonatos, and I. P. Vlahavas. 2013. Large-scale semantic indexing of biomedical publications. In *Proceedings of the 1st Workshop on Biomedical Semantic Indexing and Question Answering at CLEF*, Valencia, Spain.

L. J. van der Maaten, E. O. Postma, and H. J. van den Herik. 2009. Dimensionality reduction: A comparative review. *Journal of Machine Learning Research*, 10(1-41):66–71.

W. N. Venables and B. D. Ripley. 2002. *Modern applied statistics with S*. Springer.

X. Wang, H. Zhao, and B. Lu. 2011. Enhance K-nearest neighbour algorithm for large-scale multi-labeled hierarchical classification. In *Joint ECML/PKDD PASCAL Workshop on Large-Scale Hierarchical Classification (LSHC2)*, pages 58–67, Athens, Greece.

X. Wang, H. Zhao, and B. Lu. 2014. A meta-top-down method for large-scale hierarchical classification. *Knowledge and Data Engineering, IEEE Transactions on*, 26(3):500–513.

F. Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83.

G. Xue, D. Xing, Q. Yang, and Y. Yu. 2008. Deep classification in large-scale text hierarchies. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 619–626, Singapore.

Y. Yang and X. Liu. 1999. A re-examination of text categorization methods. In *Proceedings of the 22nd Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 42–49, California, USA.

G. Yuan, C. Ho, and C. Lin. 2012. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603.

D. Zhu, D. Li, B. Carterette, and H. Liu. 2013. An incremental approach for MEDLINE MeSH indexing. In *Proceedings of the 1st Workshop on Biomedical Semantic Indexing adn Question Answering at CLEF*, Valencia, Spain.