COMTRAVO-DS team at GermEval 2019 Task 1 on Hierarchical Classification of Blurbs

David S. Batista
Comtravo GmbH
david.batista@comtravo.com

Matti Lyra
Comtravo GmbH
matti.lyra@comtravo.com

Abstract

We present two systems developed by the Comtravo Data Science team for the GermEval'19 Task 1 on hierarchical classification of blurbs. The challenge is a document classification task where the hierarchical structure of each document needs to be captured. Our systems achieved the 13th place out of 19 submissions for Sub-Task A and the 11th place out of 19 submissions for Sub-Task B. We describe in detail these two systems pointing out the advantages and disadvantages of each as well as laying out future research directions.

1 Introduction

This paper describes the approach taken by the Comtravo Data Science team for the GermEval'19 Task 1 (Remus et al., 2019). The task aimed at developing systems to tackle the task of multi-label hierarchical classification of text.

Several real-world classification problems are naturally cast within a hierarchy, where the labels to be predicted are organized in an hierarchy. Typically the hierarchies form a tree, several trees (a forest) or a directed acyclic graph.

Examples of hierarchical document categorization are for instance categorizing news articles into an hierarchy of categories (Lewis et al., 2004), a web page into a web directory structure, Wikipedia articles into the Wikipedia taxonomy (Partalas et al., 2015), or in biomedical literature, for instance, the assignment of Medical Subject Headings to PubMed abstracts (Lipscomb, 2000).

We developed two distinct approaches, one based on a local classifier strategy, where different classifiers are trained according to the hierarchical structure of the label space, another approach uses a single classifier which tries to naively predict the entire label hierarchy for each sample.

This paper is organized as follows, in Section 2 we describe the task in detail and give a statistical description of the provided dataset. In Section 3 we briefly describe some of the proposed approaches in the literature for hierarchical document classification. In Section 4 we describe our approaches to tackle both sub-tasks. Section 5 details the our experimental setup and results. Finally in Section 6 we outline some ideas for future work.

2 Task

The GermEval 2019 Task 1 on hierarchical classification of blurbs involved the classification of german language books into genres given a book's blurb i.e., a short textual description of the book and related meta-data. The competition contained two sub-tasks:

- Sub-Task A: classify German books into one or multiple general genres, a non-hierarchical multi-label classification task with a total of 8 classes.
- Sub-Task B: targets hierarchical multi-label classification into multiple writing genres. In addition to the general genres from Sub-Task A, any number of sub-genres of increasing specificity can be assigned to a book.

2.1 Dataset

The dataset made available for these tasks contains 3 label levels organized in a hierarchy; any book can be can associated with more than one label at any given level of the hierarchy. In the hierarchy every child-label has exactly one parent-label.

The hierarchy contains a total of 343 distinct classes, 3 datasets were provided: 14 548 samples available for training, 2 079 for development and 4 157 for testing. Tables 1, 2, 3 contain a detailed

Training set	
Avg. length of blurb (tokens)	96.78
Std. deviation σ (tokens)	39.63
Avg. length of blurb (sentences)	6.55
Std. deviation σ (sentences)	2.76
Nr. unique tokens original	114 903
Nr. unique tokens lowercase	107 998
Total number of genres	343
Possible genres per level (1;2;3)	8; 93; 242
Avg. genres per blurb	3.1
Std. deviation σ	1.36
Avg. genres per blurb at level (1;2;3)	1.06; 1.34; 0.69
Std. deviation σ	0.27; 0.76; 0.79
Avg. blurb per co-occurrence	6.48
Co-Occurrence std. deviation	35.90
Nr. samples with leaf nodes at:	
- Level 1	1.9% (311)
- Level 2	44,6% (7.422)
- Level 3	53,5% (8.894)
Total number of samples	14 548

Table 1: Quantitative analysis of the training dataset.

Development set	
Avg. length of blurb (tokens)	98.71
Std. deviation σ (tokens)	46.29
Avg. length of blurb (sentences)	6.68
Std. deviation σ (sentences)	3.80
Nr. unique tokens original	33 599
Nr. unique tokens lowercase	31 818
Total number of genres	343
Possible genres per level (1;2;3)	8; 93; 242
Avg. genres per blurb	3.1
Std. deviation σ	1.39
Avg. genres per blurb at level (1;2;3)	1.07;1.35;0.69
Std. deviation σ	0.27;0.80;0.79
Avg. blurb per co-occurrence	3.08
Co-Occurrence std. deviation	8.19
Nr. samples with leaf nodes at:	
- Level 1	1.6% (34)
- Level 2	44.8% (932)
- Level 3	53.6% (1113)
Total number of samples	2 079

Table 2: Quantitative analysis of the development dataset.

Test set	
Avg. length of blurb (tokens)	96.91
Std. deviation σ (tokens)	39.83
Avg. length of blurb (sentences)	6.55
Std. deviation σ (sentences)	2.62
Total number of samples	4 157

Table 3: Quantitative analysis of the test dataset.

description of the datasets provided i.e.: training, development and test, respectively. One can see that in terms of tokens and sentences the 3 datasets are aligned, and also between training and development in terms of labels per blurb, and labels per blurb per level.

3 Hierarchical Document Classification

There are have been different strategies to approach the problem of hierarchical classifying a document (Silla and Freitas, 2011; Wehrmann et al., 2017; Kowsari et al., 2017). Within the context of GermEval'19 Task 1 we explored two main strategies: a local classifier and a global classifier.

3.1 Local Classifier

The local classifier strategy is one way to approach the hierarchical document classification task and it was first proposed, to the best of our knowledge, in the seminal work of Koller and Sahami (1997), it is also sometimes referred to as top down approach in the literature.

There are different approaches, based on the idea of a local classifier, depending on how they use the local information and devise a strategy to build several classifiers.

3.1.1 A classifier per node

The *local classifier per node approach* consists of training one binary classifier in a one-versus-rest scenario for each node in the hierarchy, where each label in the hierarchy is a node. Normally, the negative training data is taken from the same level in the label hierarchy as the positive data.

During prediction a top-down strategy is applied, the output of each binary classifier is a prediction of whether or not a given test sample belongs to the classifier's predicted class. This approach is naturally multi-label since it is possible to predict multiple labels at each level of the hierarchy.

This approach, however, is prone to label inconsistency. Consider a document that has, for the first level, labels 1, 2 and 3, and, for the second level, labels 1.1, 1.2. Since the classifiers for nodes 1 and 1.1 are independently trained, it is possible to classify a sample as having labels 1.2 and 1.1 but not the parent label 1. This approach should, therefore, be complemented by a post-processing method that tries to correct the label inconsistency.

3.1.2 A classifier per parent node

In *a classifier per parent node* approach, a multi-class classifier, possibly also multi-label, is trained for each parent node in the label hierarchy. The classifier is trained to classify the probability of

a given sample belonging to each of the parent's child nodes. In this case, a parent node is every label in the hierarchy-tree which has one or more child labels.

Given a test sample, first the top-level classifier is applied, then for every top-level predicted label (e.g., class 2 and 3) its child classifiers, e.g.: a classifier trained to predict the 2.x labels and another for 3.x labels, and so on until the last level is reached.

Note that the sub-classifiers are only trained with the children of each respective parent label, therefore this approach avoids the label inconsistency problem and respects the constrains of class-membership defined by the label hierarchy.

3.1.3 A classifier per level

The *local classifier per level* approach consists of training one multi-class, and possibly also multi-label, classifier for each level of the label hierarchy. When a new test sample is presented the output of the classifiers from each level is used as the final classification.

This approach, however, is prone to label inconsistency, as different classifiers are trained for each level of the hierarchy and should, therefore, also be complemented by a post-processing method to correct the prediction inconsistency.

One common problem for all local classifier strategies the utilize the top-down class-prediction approach is the propagation of errors down the hierarchy.

3.2 Global Classifier

Another type of strategy is to learn a classifier than can globally learn to output the predictions for each level in the hierarchical structure. This is done by flattening the whole hierarchical structure.

Having only a single classifier, although easier to tune, it can turn the hierarchical classification into a much harder problem, specially having a sparse label space with an order of magnitude of 10^2, i.e. there are 343 possible classes, but the labels co-occurrence can be a good guiding heuristic for a statistical model to infer the hierarchical label structure associated with a given sample.

4 Systems Developed

We developed two systems implementing the following approaches:

Local Classifier: a classifier per parent node using different types of classifiers;

Global Classifier: a single classifier relying on the hypothesis to explore the labels co-occurrence;

4.1 Local Classifier

We employed a classifier per parent node approach, which has the advantage of not being prone to label inconsistency errors. We need to train classifiers for each parent node. For Level 3 we don't need to train any classifier since it contains only leaf nodes, plus some nodes on Level 2 are already leaf nodes.

According to Table 1 the Level 1 has 8 possible labels, which means that the parent node of the first level (i.e, the Root Node) needs to be trained in a multi-label fashion and predict over 8 classes. Each of these 8 classes represents a parent node of some child classes on the next level of the hierarchy. So for Level 1 we need to train eight multi-label classifiers where the labels are the child's of each parent in the root level. Finally, for Level 2, we train only 42 classifiers, since according to the hierarchy-tree some labels in this level are already leaf nodes, and only 42 labels have then child labels. So, in total we trained 51 classifiers distributed by different levels as described in Table 4.

Level	Nr. Parent Nodes
Root Node	1
Level 1	8
Level 2	42
Total Classifiers	51

Table 4: Number of parent nodes per level in the hierarchy.

As stated before, one of the advantages of this approach is that it always produces a label structure that is enforced by the hierarchy-tree, but it is prone to error propagation from the top levels further down the tree.

4.2 Global Classifier

The global classifier needs to be a multi-label classifier targeting a label space with a total of 343 classes. One of the advantages is that there is only one single multi-label classifier to tune and explore, but on the other and it has a high and sparse label space. Plus, one needs to employ

some post-processing cleaning to enforce the hierarchical structure, since there is an hierarchical dependency between the some of the 343 possible classes. For instance, the classifier can predict the labels 4.3 and 4.4 - corresponding to labels in the Level 2 in the hierarchy - but not the label 4, corresponding to Level 1 in the hierarchy.

5 Experiments and Results

In this section we describe the experimental setup and results for the two devised strategies for tackling both sub-tasks.

5.1 Results on the Development Set

During the development phase we only had the labels for training, so we randomly split the training dataset into two-subsets of 70% and 30% for training and parameter tuning, respectively. Then, train on the whole dataset with the best parameters and using the model to generate predictions on the development dataset. This approach was mainly to have a working framework for experiments and submit valid results.

During the test phase the labels for the development set were made available and we could use them to tune the classifiers. We used 3-fold cross-validation to perform parameter search using the training dataset. The parameter configuration which yielded the best results was then used to train the classifiers over the complete training dataset, and the classifier is then evaluated against the development set. Results reported in this section are all in regard to the development set.

5.1.1 Pre-processing

For representing a book we concatenated the book's title with book's textual description. In some cases only the title is present, for this cases we simply use the title.

We explored two tokenization schemas, one tokenizes the blurbs into sentences, and then from sentences into tokens, considering the title of the book as a sentence, this tokenization strategy was based on the german sentence tokenizer, and the *word_punkt_tokenizer* from NLTLK 3.4.1 (Bird et al., 2009), and considered alphanumeric tokens only. The other approach was based on a simple regular expression: `(?u)\b\w\w+\b`. We also experimented lower casing and removing stop-words. After running a few experiments and comparing some initial results we opted for the regular expression for tokenization, lower case token representations and removal of stop-words. For the neural networks the padding was done to match the size of the longest document in the dataset.

5.1.2 Prediction threshold gltment

We set the prediction threshold to 0.5, so any label with a predicted probability above 0.5 is selected. We noticed that for some samples no labels were being selected, as all labels had predicted probability scores lower than 0.5. To tackle this problem, for a given sample, if no predictions were done we lowered the threshold to 0.4, if still no label predictions are done, we lowered again the threshold to 0.3. This was done in a simple ad-hoc way, and no proper strategy was employed, and is done in a per label fashion.

5.1.3 Models implementation

For all the neural network models we used pre-trained embeddings, specifically the public available German fastText embeddings trained on Wikipedia, of dimension 300 and obtained using the skip-gram model as described in Bojanowski et al. (2017), the embeddings are fine-tuned during learning and out of vocabulary words are randomly initialized.

The training was done with the Adam optimizer (Kingma and Ba, 2014) using binary cross-entropy as a loss function and 30% of the training dataset for validation. Unless stated otherwise training was performed with mini-batch sizes of 16 for 10 epochs. All the neural network models were implemented in Keras 2.2.4 with Tensorflow 1.13.1 backend. The Logistic Regression classifier was based on the scikit-learn 0.21.1 (Pedregosa et al., 2011).

All the code used for this experiments is available on-line [1].

5.1.4 Local classifier per node: Root Node

We explored different classifiers for the Root Node and for Level 1 and Level 2 we selected a Convolutional Neural Network (CNN).

We briefly describe the models used for the Root Node and the parameters explored, in bold we have the parameters that yielded the best scores:

Logit (TF-IDF): a logistic regression classifier with TF-IDF weighted vectors in a one-

[1] https://github.com/davidsbatista/GermEval-2019-Task_1

versus-rest scenario varying the following parameters:

- n-grams: 1, **2**, 3;
- class weight: **balanced**, not-balanced;
- norm: l1, **l2**;
- regularization C: 0.1, 10, 100, **300**;

Training was performed with Stochastic Average Gradient for a maximum of 5 000 iterations.

CNN (Kim, 2014): for sentence classification with rectified linear units in the activation functions of the 1D convolutions, and with a fully connected layer of size 600 and varying:

- filter windows: (1), **(1, 2)**, (1, 2, 3);
- feature maps: 256, **300**;

LSTM (Hochreiter and Schmidhuber, 1997): recursively reads each token in the text updating it's internal state and using the last state to represent the document, with a dropout layer of rate 0.1 between the LSTM's last state and the final sigmoid layer.

- single LSTM vs **bi-directional LSTM**;
- hidden units: 32, 64, **128**;

Bag-of-Tricks (Joulin et al., 2017): token embeddings representations are averaged into a single vector representation, which is fed to a sigmoid classifier, we varied the following parameters:

- n-grams: **2**, 3, 4, 5;
- top-k most frequent tokens: **100k**, 90k, 80k;

Table 5 shows the results for different classifiers when trained with the best parameters on the training set and evaluated against the development set.

Method	Precision	Recall	F_1
Logit (TF-IDF)	0.8211	**0.8359**	**0.8284**
CNN	**0.8542**	0.7879	0.8197
bi-LSTM	0.8062	0.7987	0.8024
Bag-of-Tricks	0.3787	0.6717	0.4843

Table 5: Results for different classifiers on the Sub-Task A on the development set.

The Logistic Regression classifier achieved the best results although the CNN classifier had a similar F_1 score, essentially by trading recall for precision.

5.1.5 Local classifier per node: Level 1 and 2

For Level 1 and 2 in the hierarchy tree we trained a total of 50 classifiers for each parent node, 8 for Level 1 and 42 for Level 2. All these classifiers were based on the CNN model.

We explored some parameters configurations by varying the filter windows size and the filter maps size, Table 6 shows a subset of the different configuration parameters tried which yielded some of the best results. All these classifiers were training with mini-batch size of 16 for 5 epochs.

	Filter Windows	Filter Maps
$Conf_1$	**1,2**	**300**
$Conf_2$	1,2,3	200
$Conf_3$	1,2,3,5,7	300
$Conf_4$	3,5,6,10	256

Table 6: Different configuration parameters for the CNN classifiers for Level 1 and 2.

In these experiments we used the Root Node classifier which yielded the best results, i.e. the logistic regression, to predict the labels for the root level, and experiment with different parameters for the Levels 1 and 2. Table 7 shows the results for Sub-Task B for configurations of parameters in Table 6.

	Precision	Recall	F_1
$Conf_1$	0.7151	**0.5330**	**0.6108**
$Conf_2$	0.7144	0.5303	0.6087
$Conf_3$	0.7219	0.5235	0.6069
$Conf_4$	**0.7274**	0.5085	0.5986

Table 7: Results for Sub-Task B for different configurations of the CNN-based classifier for Levels 1 and 2 of the hierarchy, using the best classifier for the Root Level from Table 6.

We opted not to use a logistic regression classifier for Level 1 and 2 since this type of classifier needed to be trained in a one-versus-rest fashion. This would mean that for Level 2 we would need to train 93 classifiers and 242 for Level 3.

5.1.6 Global classifier

The global classifier uses a flattened hierarchy and learns how to predict a vector of 343 dimensions.

One advantage of this approach, in contrast to the local one, is the we need only to tune a single classifier, and both sub-tasks can also be tackled with this single classifier.

Our initial idea was to use a neural network architecture and leverage on the labels co-occurrence by initializing a weight matrix - we describe this idea in Section 6 - but due to time constrains this was not possible to explore this idea to the end. Also, due to time constrains we did not employ a post-processing step.

Nevertheless, we still applied a CNN architecture and explored different configuration parameters, varying the filter windows and the filter maps.

We used the same tokenization schema as with the Local classifier, as well as the same pre-trained word embeddings, which are fine-tuned during training.

The training was done with the Adam optimizer (Kingma and Ba, 2014) using binary cross-entropy as a loss function, with mini-batch sizes of 128 for 250 epochs and using 30% of the training dataset for validation.

We also used the same threshold filtering strategy as described in Section 5.1.2.

Table 8 presents some of the configurations of parameters used in the experiments and Table 9 the corresponding results for the same configurations for both sub-tasks.

	Filter Windows	**Filter Maps**
Conf_1	1, 2, 3	300
Conf_2	3, 5, 7, 10	256
Conf_3	**1, 2, 3, 5, 7, 10**	**256**

Table 8: Different configuration parameters for the CNN classifiers for the global classifier.

	Precision	**Recall**	**F_1**
Conf_1			
Sub-Task A	0.7163	0.7484	0.7320
Sub-Task B	0.5257	0.4603	0.4909
Conf_2			
Sub-Task A	0.7353	**0.7686**	0.7516
Sub-Task B	0.5470	0.4717	0.5066
Conf_3			
Sub-Task A	**0.8389**	0.7659	**0.8008**
Sub-Task B	**0.6733**	**0.5032**	**0.5760**

Table 9: Results for both sub-tasks using the configuration parameters from Table 8.

5.2 Test Results

We applied the classifiers described in the Section 5 with the parameters that yielded the best results on the development dataset, by training on all available data (i.e., training + development sets) and applied them on the test dataset, therefore generating two submissions for each sub-task.

5.2.1 Parent Per Node

With the one parent per node strategy classifier, we achieved the 13th best place on Sub-Task A, and the 11th best place on Sub-Task B, out of a total of 19 submissions. Results for the detailed evaluation metrics are described in Table 10.

This classifier achieved the 9th best recall and the 15th best precision for Sub-Task A, and the 8th best recall and the 14th best precision for Sub-Task B.

For roughly 5% of the test samples the classifier did not produce any predictions, due to the class probabilities scores being lower 0.3., the lowest possibly threshold selected in the adjustment.

Task	**Precision**	**Recall**	**F_1**
Sub-Task A	0.8144	0.8255	0.8199
Sub-Task B	0.7042	0.5274	0.6031

Table 10: Best achieved results on the development set for both Sub-Tasks A and B with the parent per node classifier.

5.2.2 Global

With the global classifier strategy we achieved the 17th best place on Sub-Task A, and the 13th best place on Sub-Task B. Detailed results are presented in Table 11.

Task	**Precision**	**Recall**	**F_1**
Sub-Task A	0.7761	0.7839	0.7839
Sub-Task B	0.5672	0.5185	0.5418

Table 11: Best achieved results on the development set for both Sub-Tasks A and B with the global classifier.

This classifier achieved the 15th best recall and the 17th best precision for Sub-Task A, and the 9th best recall and the 18th best precision for Sub-Task B. For roughly 1% of the samples the classifier did not produced any predictions. The hierarchy consistency is of 0.9363, reflecting the lack of a post-processing step to enforce the hierarchical structure, which would be 1.0 if employed.

Considering only the best submissions from all teams, we ranked 8th for Sub-Task A and 6th for Sub-Task B.

6 Future Work

We had planned to explore different features and carry more experiments but time constraints for the submission of the test results did not allowed us to experiment all that was planned.

One crucial aspect in the global classifier is a post-processing step to make sure that the labels output is in-line with the hierarchy-tree constrains. The global classifier could also be improved by initializing a weight matrix based on label co-occurrence. Kurata et al. (2016) proposed a neural network initialization method to treat some of the neurons in the final hidden layer as dedicated neurons for each pattern of label co-occurrence. These dedicated neurons are initialized to connect to the corresponding co-occurring labels with stronger weights than to others representing non co-occurring labels. Baker and Korhonen (2017) applied this idea in the biomedical domain and to a much more compact hierarchy than the one presented in this paper.

In the local classifier strategy for Level 1 and 2 we use the same architecture for all classifiers and tuned them in the same way, the type of architecture and tuning process could be made dependent on the numbers of samples available to train and level in the hierarchical tree.

A few more features could have been explored, for instance the author's name and the release date of the book. The padding of the documents representation for the neural network could be set to the average since of the documents, instead of the longest one.

The values for the prediction threshold were selected in an ad-hoc fashion, these could also be properly set, through a set of experiments.

References

Simon Baker and Anna Korhonen. 2017. Initializing neural networks for hierarchical multi-label text classification. In *BioNLP 2017*, pages 307–315, Vancouver, Canada,. Association for Computational Linguistics.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*, 1st edition. O'Reilly Media, Inc.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, Valencia, Spain. Association for Computational Linguistics.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Daphne Koller and Mehran Sahami. 1997. Hierarchically classifying documents using very few words. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 170–178, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Kamran Kowsari, Donald E. Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S. Gerber, and Laura E. Barnes. 2017. Hdltex: Hierarchical deep learning for text classification. In *ICMLA*, pages 364–371. IEEE.

Gakuto Kurata, Bing Xiang, and Bowen Zhou. 2016. Improved neural network-based multi-label classification with better initialization leveraging label co-occurrence. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 521–526, San Diego, California. Association for Computational Linguistics.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397.

Carolyn E Lipscomb. 2000. Medical subject headings (mesh). *Bulletin of the Medical Library Association*, 88(3):265.

Ioannis Partalas, Aris Kosmopoulos, Nicolas Baskiotis, Thierry Artières, George Paliouras, Éric Gaussier, Ion Androutsopoulos, Massih-Reza Amini, and Patrick Gallinari. 2015. LSHTC: A benchmark for large-scale text classification. *CoRR*, abs/1503.08581.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830.

Steffen Remus, Rami Aly, and Chris Biemann. 2019. Germeval-2019 task 1: Shared task on hierarchical classification of blurbs. In *Proceedings of the GermEval 2019 Workshop*, KOVENS '19, Erlangen, Germany.

Carlos N. Silla, Jr. and Alex A. Freitas. 2011. A survey of hierarchical classification across different application domains. *Data Min. Knowl. Discov.*, 22(1-2):31–72.

Jônatas Wehrmann, Rodrigo C. Barros, Silvia N. das Dôres, and Ricardo Cerri. 2017. Hierarchical multi-label classification with chained neural networks. In *Proceedings of the Symposium on Applied Computing*, SAC '17, pages 790–795, New York, NY, USA. ACM.