Proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019)

Label Frequency Transformation for Multi-Label Multi-Class Text Classification

Raghavan A K
Global AI Accelerator, Ericsson / Chennai
k.raghavan.a@ericsson.com

Venkatesh Umaashankar
Ericsson Research / Chennai
venkatesh.u@ericsson.com

Gautham Krishna Gudur
Global AI Accelerator, Ericsson / Chennai
gautham.krishna.gudur@ericsson.com

Abstract

In this paper, we (***Team Raghavan***) describe the system of our submission for GermEval 2019 Task 1 - *Subtask (a)* and *Subtask (b)*, which are multi-label multi-class classification tasks. The goal is to classify short texts describing German books into one or multiple classes, 8 generic categories for *Subtask (a)* and 343 specific categories for *Subtask (b)*. Our system comprises of three stages. **(a)** Transform multi-label multi-class problem into single-label multi-class problem. Build a category model. **(b)** Build a class count model to predict the number of classes a given input belongs to. **(c)** Transform single-label problem into multi-label problem back again by selecting the top-k predictions from the category model, with the optimal k value predicted from the class count model. Our approach utilizes a *Support Vector Classification* model on the extracted vectorized *tf-idf* features by leveraging the *Byte-Pair Token Encoding (BPE)*, and reaches f1-micro scores of **0.857** in the test evaluation phase and **0.878** in post evaluation phase for *Subtask (a)*, while **0.395** in post evaluation phase for *Subtask (b)* of the competition. We have provided our solution code in the following link: https://github.com/oneraghavan/germeval-2019.

1 Introduction

Multi-label Multi-class Hierarchical Classification tasks typically encompass multiple possible (one or more) labels for each instance (not mutually exclusive) across multiple possible classes (two or more) with many levels of hierarchies, and are widely used in domains like text classification (Rousu et al., 2006), image classification (Hsu et al., 2009) and bioinformatics (Barutcuoglu et al., 2006), (Feng et al., 2017). In this paper, we present our submission approach for *Subtask (a)* and *Subtask (b)* in *GermEval 2019 Task 1* (Remus et al., 2019). Here, we convert our task into two sub-problems: first, to predict the category; second, to predict the class frequency corresponding to the multi-label setting. Our approach can be broadly split into three stages.

- Transform the multi-label multi-class problem into a single-label multi-class problem, and build a category model.
- Build a class count predictor model to predict the number of classes that a given input could be categorized.
- Transform single-label problem back into a multi-label problem by selecting the top k predictions from the category model, with the optimal k value predicted from the class count model.

Conventionally, Natural Language Processing (NLP), particularly text classification tasks have been modeled using variants of Support Vector Machines (Joachims, 1998) and Naive Bayes Classifiers (McCallum et al., 1998). With the widespread adoption of deep learning models, there has been considerable increase in efficiencies for such tasks, however they are still considered black box models, and it is extremely hard to interpret them, in contrast to conventional Machine Learning algorithms. Moreover, the time and computational resources required for training deep neural networks are extremely higher than conventional Machine Learning models.

Hence, in our approach, we utilize the traditional ***Support Vector Classification*** modeled using *tf-idf* (term frequency - inverse document frequency) feature vectors combined with class count predictor, and we leverage the *Byte-Pair Encoding (BPE)* compression tokenization mechanism. Experimental results from exploiting

such simple model fusion approaches show that in the post-evaluation phase, f1-micro scores of 0.878 on *Subtask (a)* and 0.395 on textitSubtask (b) could be achieved. The overall modeling pipeline/architecture of our approach can be found in Figure 1.

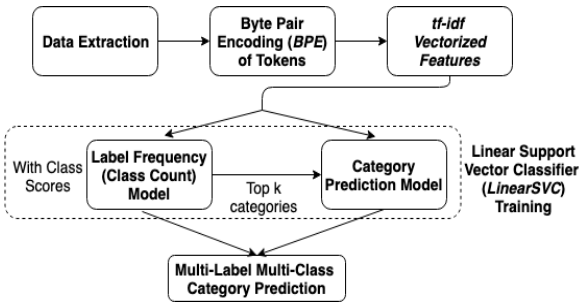

Figure 1: Modeling Pipeline/Architecture for both *Subtask (a)* and *Subtask (b)*

The rest of the paper is organized as follows. The characteristics of the dataset are described in Section 2. In Section 3, the data extraction and pre-processing techniques to obtain our feature vectors are discussed. Section 4 presents our model architecture along with training and aggregation phases. This is followed by systematic evaluation of our model's results and submission in Sections 5 and 6, and we finally conclude the paper.

2 Data Description

The *GermEval 2019 Task 1* dataset (Remus et al., 2019) consists of German books crawled from `randomhouse.de`, with the following attributes – title, description, author name, ISBN and book release date. Apart from date, most of the other features available are in the form of text, where title and description are very short texts. A total of 343 categories are present across three levels of hierarchy with 8, 93 and 242 categories in each level, and multiple labels can be assigned to each book.

The corpus has a very imbalanced label distribution. Figure 2 shows the top-level label distributions, while Figure 3 shows the top 30 label distributions, and it can be vividly inferred that the label categories are highly skewed.

3 Data Extraction and Prepossessing

The corpus was presented as an XML file, with XML tags for each feature. The XML files were parsed using the Python library - *BeautifulSoup*,

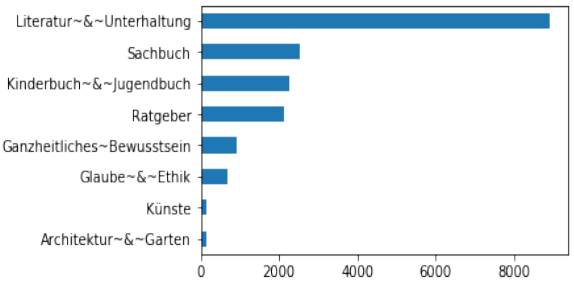

Figure 2: Top-level Label Distribution

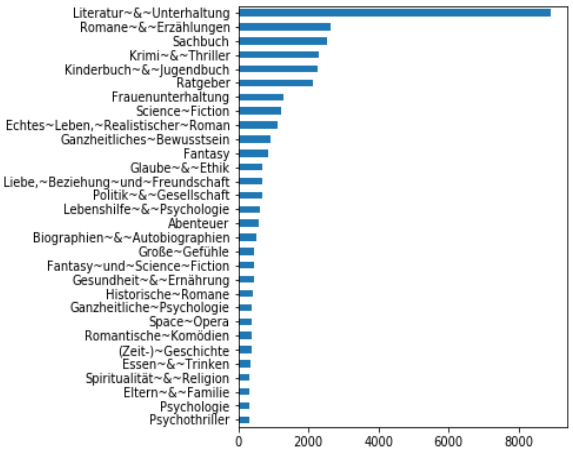

Figure 3: Label Distribution for top 30 classes with hierarchies

and the columns given in the corpus such as title, description, list of authors, date of publication, ISBN were extracted from the XML format into a CSV format. The corpus has hierarchy information in the *hierarchy.txt* file, which was used to validate and prepare the corpus for modelling.

From the extracted data, we initially filter out the stop words and numbers from title and description for every book using the *Natural Language Toolkit (NLTK)* library available in Python. We then utilize ***Byte-Pair Encoding (BPE)*** (Heinzerling and Strube, 2018) based tokenization to create tokens from the titles and descriptions for various texts. BPE tokenization, in this context, identifies the most common consecutive bytes of German words and replaces with a byte that does not occur within the data. For instance, *'Ein Blick hinter die'* might be converted into *'__ein __blick __hinter __die'* after BPE with a vocabulary size of 25,000. We utilize the BPEmb library for the same, which utilizes pre-trained byte-pair embeddings, and require no tokenization, also being much shorter.

The German words are split into multiple smaller sub-words, which would inherently en-

able better representations for each book's Title and Description, and provide more context for the feature vectors which are learned. Subwords in BPEmb also enable effective guessing of unknown/out-of-vocabulary words' meaning based on context. For instance, the suffix -shire in Melfordshire indicates a location.

After tokenizing each title and description of books using *BPE*, we create ***term frequency - inverse document frequency (tf-idf)*** vectors from the tokens. *tf-idf* is a widely used statistical measure in domains of NLP, text mining and information retrieval, which signifies the importance of a word to a document in the whole corpus (Ramos, 2003).

We experimented with various contiguous sequences of n-grams – unigrams, bigrams and trigrams for creating *tf-idf* vectors, and observed that bigram models yielded better results than the rest. Each book can have multiple authors, so we treat the authors' data as a binomial attribute (creating a *Label Binarizer* which returns 1 if the book was authored by that author, else 0) across all authors. We extract the ***year*** from the publication date, and utilized it as a categorical feature. We also extracted the ***Group ID*** and ***Publisher ID*** from the 13-digit ISBN and represented them as categorical features again.

We created two sets of target variables from the extracted data – one with the categories as target labels, and other with the count of categories. Hence, we also created and pipelined two different models – *class count model* and *category score predictor model*. For instance, for a given book instance, if there are k categories, then that training sample has been duplicated k times with each sample having one category. Adding all features, a sparse feature matrix was created. The final feature matrix is of size (17783 x 594931) for both class count model and category score predictor model.

Labels	Counts
Label 1	15549
Label 2	1004
Label 3	70
Label 4	4

Table 1: Class Count Frequencies

The class count model is a classification problem with labels 1, 2, 3 as class frequencies. While there are book instances up-to 4 top-level observed class frequencies, there are only 4 training samples for category 4 which is extremely less (negligible), hence we consider only 3 categories. The class count distribution can observed in Table 1. The categories are predicted based on an 8-class classification approach for top-level categories (*Subtask (a)*), and a 343-class classification mechanism for multi-level (*Subtask (b)*) categories.

4 Model Pipeline

Given the corpus has a heavy class imbalance across all levels, we choose a Support Vector Machine (SVM) based model for the task. SVMs are known to exhibit robustness and perform effectively under class imbalance (Tang et al., 2009).

The ***Linear Support Vector Classifier (LinearSVC)*** in a multi-class classification problem utilizes a one vs rest scheme, wherein the objective is to return the best-fit hyperplane that categorizes the data in an n-dimensional space. Identifying the right hyperplane is primarily dependent on the *margin* (maximized distance between hyperplane and nearest data point), and the loss function governing the margin. We utilize the *squared-hinge loss* for the same, as it is widely used for maximum-margin classification in SVMs that penalizes the violated margins more strongly (quadratically). The *squared-hinge loss*, l for Linear SVM is given by,

$$l(y) = max(0, 1 - t \cdot y)^2$$

where y is the classifier score $y = w.x + b$, w, b are the parameters of the hyperplane, x is the input variable(s) and the intended output t = ± 1.

We created two Linear SVC models, one for predicting the count of classes a book could belong to, and the other for category score predictor. *Linear SVC* was chosen as it uses the *liblinear* framework and scales well with the increase in the number of features (Fan et al., 2008). Moreover, it is computationally faster than most other Linear SVM implementations, and utilizes many other advanced optimization techniques. We utilize the *scikit-learn* Python framework (Pedregosa et al., 2011) to train and test the *LinearSVC* model, which uses liblinear as its default implementation. Also, this implementation offers flexibility in the choice of penalties and loss function parameters, and would inherently scale well to larger samples of data.

In our case, use of *tf-idf* vectors to represent words created a large number of feature vectors. The *tf-idf* vector representation were very sparse owing to the short text representations for each book. Compressed sparse representation (*CSR*) of the feature matrix was further utilized reduce the size of the training set.

4.1 Model Parameters

After extensive parametric optimization for both models – count classifier and category classification using grid search, we arrived the following set of final parameters to yield best efficiencies. The optimal parameters for the LinearSVC model are elucidated in Table 2.

Parameter	Optimal Value
C	1.0
Tolerance for stopping	0.0001
Loss	Squared Hinge Loss
Penalty	L2
Optimization algorithm	Dual
Max Iterations	3000

Table 2: Optimal Parameters for *LinearSVC*

For the top-level classifier, the following class weights are also used to handle class imbalance. We utilized grid search to fine tune the class weights again, which can be observed in Table 3.

Category	Class Weight
Kinderbuch & Jugendbuch	1.8
Ratgeber	3
Sachbuch	2
Glaube & Ethik	2
Künste	6
Architektur & Garten	6
Literatur & Unterhaltung	1
Ganzheitliches Bewusstsein	1

Table 3: Class Weights for Top-level Categories to handle Class Imbalance in *LinearSVC*

The class weights might be conventionally perceived that the categories with lower cardinality might have higher class weights. However, the model prioritizes and takes into account the confusion between various classes than the imbalance in classes alone. For instance, we could observe that a lot of Sachbuch and Glaube get classified as Literatur & Unterhaltung, hence giving more weightage to the latter aids in higher efficiencies.

4.2 Model Fusion

We pipeline the class score predictor and class count predictor models together for effective classification of categories. Since the input features for both models remain the same, we train our model with categories as target variables for the class score predictor model, and with class counts as target variables for class count predictor model.

In the class count model, we also add the scores of output class predictions to the input feature space. This is motivated by the inherent fact that there is a high correlation between the number of categories a book belongs to, and its corresponding class with the highest score. While predicting a book's category, we first get the class scores for all categories from the class score predictor model, and then append those predictions with input data to the class count classifier model. Once the class count predicts number of possible categories k, we find the top k category predictions from the class category predictor (likelihood) model.

In this way, the class imbalanced multi-label problem is split into two simple prediction problems. Also, by splitting the problem into smaller chunks, we are able to train multiple models in parallel, thus reducing the total training time of the system. With the above setup, retraining the entire set of models takes just under 2 minutes.

5 Evaluation

For building an end-to-end classifier system, we build a Classifier Class extending the *scikit-learn* Base estimator API, with the respective fit and predict functions. The constructor parameters passed, are the hyperparameters for the class count predictor and class likelihood predictor models. Inside the constructor, we create two LinearSVC models. In the fit (training) phase, both the predictors are trained with their respective target variables. We utilize a ***K-Fold Cross-validation*** strategy ($K = 4$) and get the class scores to be used in training for class count predictor. Once we have the class scores, we retrain the class predictor with the whole training data again. The class count predictor is then utilized for training along with the class scores appended.

Similarly, in the predict phase, the class prediction model is first used to obtain class prediction scores, and further utilized by class count predictor to get the class count distribution. We select take k highest category predictions from the class

scores.

The micro-f1 scores for *Subtask (a)* and *Subtask (b)* with CV=4 folds can be found in Table 4.

CV Fold	*Subtask (a)*	*Subtask (b)*
Fold 1	0.833	0.384
Fold 2	0.943	0.471
Fold 3	0.950	0.484
Fold 4	0.900	0.397

Table 4: k-fold Cross-Validation (k=4) on *Subtask a* and *Subtask b*

The experimental setup (HW/SW configurations) utilized for our solution are as follows:

(1) Intel® Xeon® Processor E5-2650 v4 30M Cache, 2.20 GHz, 12 Cores, 24 Threads **(2)** 250 GB RAM **(3)** CentOS 7.

6 Submission and Results

With the above setup, the model was able to achieve the results showcased in Table 5 in the test phase.

Phase	*Subtask (a)*	*Subtask (b)*
Validation Phase	0.851	0.4098
Test Phase	0.857	-
Post Evaluation Phase	0.878	0.3947

Table 5: Evaluation Metrics (f1-micro scores) for Test Data on *Subtask a* and *Subtask b*

Team Raghavan achieves rank 4 in *Subtask (a)* during the test phase (f1-score of $\sim$0.86). However, in the post-evaluation phase, we achieve an f1-score of 0.878, which secures us the first position in *Subtask (a)*. The additional 0.02 gain in micro-f1 score during post-evaluation phase finally was achieved by adding ISBN based features – *Group ID* and *Publisher ID*.

7 Conclusion

In this paper, we have successfully demonstrated that traditional approaches like Linear Support Vector Machine Classifier, with a class count predictor model can effectively model Multi-label Multi-class Hierarchical Text Classification of German blurbs – *GermEval Task 1*. The model designed by us was aimed for top-level categories, i.e., *Subtask (a)*, which implements flat classification and doesn't make use of much hierarchical dependencies. That is one primary reason for *Subtask (b)* achieving relatively less efficiencies.

The authors would like to emphasize that conventional machine learning solutions would help in better interpretability, and when pipelined/fused with the right set of techniques, can effectively save a lot computational resources and time.

8 Credits

The authors would like to thank their colleagues at Ericsson Research and Global AI Accelerator (GAIA) at Ericsson during this competition. Also, we would like to thank the *scikit-learn* developers for actively developing and maintaining this awesome tool. Finally, we thank the *GermEval* competition organizers for fostering a friendly and collaborative environment around this dataset and for answering our questions throughout the competition.

References

Zafer Barutcuoglu, Robert E Schapire, and Olga G Troyanskaya. 2006. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9(Aug):1871–1874.

Shou Feng, Ping Fu, and Wenbin Zheng. 2017. A hierarchical multi-label classification algorithm for gene function prediction. *Algorithms*, 10(4):138.

Benjamin Heinzerling and Michael Strube. 2018. BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. European Language Resources Association (ELRA).

Daniel J Hsu, Sham M Kakade, John Langford, and Tong Zhang. 2009. Multi-label prediction via compressed sensing. In *Advances in neural information processing systems*, pages 772–780.

Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer.

Andrew McCallum, Kamal Nigam, et al. 1998. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, pages 41–48. Citeseer.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Juan Ramos. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the First Instructional Conference on Machine Learning*, volume 242, pages 133–142.

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. Erlangen, Germany*.

Juho Rousu, Craig Saunders, Sandor Szedmak, and John Shawe-Taylor. 2006. Kernel-based learning of hierarchical multilabel classification models. *Journal of Machine Learning Research*, 7(Jul):1601–1626.

Y. Tang, Y. Zhang, N. V. Chawla, and S. Krasser. 2009. Svms modeling for highly imbalanced classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(1):281–288.