Team Waterloo at the SIGIR E-Commerce Data Challenge

Angshuman Ghosh
University of Waterloo
a25ghosh@uwaterloo.ca

Vineet John
University of Waterloo
email@institution.domain

Rahul Iyer
University of Waterloo
rahul.iyer@uwaterloo.ca

ABSTRACT

This paper describes the methods we use to solve a taxonomy classification problem specific to categorization of products in an e-commerce catalog. We describe the use of learning algorithms like multinomial naive-bayes, logistic regressors, batched gradient-descent classifiers and neural softmax classifiers retro-fitted for the hierarchical classification objective. We explore methods that exploit the hierarchical nature of the data, and compare it against flat classification approaches. We use simple models with minimal pre-processing to build and compare classifiers by reporting their respective weighted precision, recall and F_1 score metrics on the test data-set.

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**; **Information extraction**;

KEYWORDS

taxonomy,classification

1 INTRODUCTION

Taxonomy classification is a multi-label classification problem where the classes form a hierarchy, as in a tree or a directed acyclic graph (DAG). Although a significant amount of research in natural language processing (NLP) pertains to flat-classification problems, class hierarchies are fairly common in the real-world, and it follows that are there are tasks that require classification algorithms that address such hierarchical classification problems.

Hierarchies emerge as a natural result of the organization of extensive data-sets into fine-grained categories [6]. In this task we work with a product catalog taxonomy, with each individual product being a part of an increasingly finer-grained category. The objective of the task is to be able to accurately predict the entire class hierarchy under which the product is categorized.

The contributions of this paper are the evaluation of different strategies including flat and hierarchical classification, and the implementation of a general framework to address the task of taxonomy classification.

2 RELATED WORK

Taxonomy classification is a topic well-explored within the NLP community, with several novel approaches over the past two decades. It is also known in the literature as 'hierarchical' or 'structured' classification. The nature of the task implies that binary and multi-class classifiers cannot be used as is to solve this problem. However, these existing algorithms can be re-purposed for taxonomy classification, as will be subsequently discussed in Section 3.

The seminal work on taxonomy classification was published by KOLLER. The authors propose learning a small set of features to solve each classification task in the tree structure independently. This method involves training a classifier for each non-leaf node in the tree. This model comprises a top-down approach under the assumption that the instance is already classified with respect to all the labels that precede the current label in the hierarchy. This also drastically reduces the number of target classes' probabilities to predict. However, a drawback of this approach is the large number of models that need to be trained to facilitate it. It also suffers from the cascading effect of classification failure i.e. all subsequent classifications after an incorrect classification will also be incorrect.

Since the most common schemes for multi-class classification in general are the One-vs-One and One-vs-All schemes, support vector machines, being max-margin classifiers, have been used frequently to address this task in previous works [5] [4] [2]. These works have used the SVM in the standard flat multi-classification setting and also proposed changes to adapt the classifier to the hierarchical setting, namely the Binary Hierarchical Classifier (BHC) [4] and the Hierarchical SVM (H-SVM) [2].

Another work, published by McCallum et al. uses shrinkage to improve classification of a hierarchy of text articles. They use a maximum log-likelihood estimate (MLE) based objective to shrink the MLE of each node towards the MLEs of all its ancestors.

Our approaches, in contrast, use simpler estimators, and rely on on minimal post-processing for a few among them to collate the predictions in the event of multiple classifiers being used for a single learning algorithm.

3 APPROACH

In Section 3.1, we describe the data cleaning and simple intuitions we use to guide our learning strategies. We subsequently describe each of the approaches that we utilized to obtain the requisite predictions.

3.1 Preprocessing

Each data instance to be classified in this task is a product name, which is usually comprised of the product's canonical name, a brand name and any high-level specifications (like dimensions, color etc.) if applicable.

There are 3008 unique category paths known from the training data, with a maximum category depth of 8.

As opposed to full sentences, there is intuitively a lesser advantage in implementing a forget mechanism for past contexts in a product name, disincentivizing the use of recurrent networks as feature extractors.

Also, most of the numbers in the data-set, for instance, in the examples presented in Table 1 are used to quantify specifications

SIGIR'18, July 2018, Ann Arbor Michigan, USA
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Replacement Viewsonic VG**710** LCD Monitor **48**Watt AC Adapter **12**V **4**A
4-Pack Replacement Engine Air Filter for **2009** Sterling Truck Bullet **55 L6 6.7** Car/Automotive
Luscious Pink Perfume **3.4** oz Eau De Parfum Spray For Women By MARIAH CAREY

Table 1: Examples of products with numbers

of the product, and are not useful unless they're part of the brand or product name itself.

Hence, we trim the numbers from the product names, in addition to trimming the default english stopwords from curated dictionaries maintained by NLTK[1], spaCy[2] and scikit-learn[3]. This eliminates most of the conjunctions and prepositions used within a product name that are not relevant to its class. Additionally, we also remove any word less that 3 characters in length, to eliminate most of the uninformative measure of quantities (e.g. V, mm, oz etc.).

3.2 One vs. All Classification

The class taxonomy provided is a Tree and each leaf node uniquely identifies a class. Thus, for our initial trials, we decided to train models using only leaf nodes. We used two classifiers: Multinomial Naive Bayes (MNB) and XGBoost [1]. While there were no hyper-parameters for MNB, XGBoost has a considerable number of tunable parameters. XGBoost is an ensemble method where a large number of decision trees are combined using gradient boosting and it has performed really well in multiple Kaggle competitions. The two most important hyper parameters are the depth of the decision trees that are grown, and the number of trees that are trained. We performed a random search for parameters and obtained the best results with a max-depth of 7 and 200 estimators.

3.3 Multi-level Classification

This classifier attempts to exploit the hierarchical nature of the categories. This requires using multiple classifiers (one for each level of the taxonomy tree), and combining their eventual prediction results to form the entire category path prediction. Based on the training set, the count of unique classes to predict at each level is shown in Table 2.

This is analogous to the Local Classifier Per Level approach discussed in a comprehensive survey of hierarchical classification methods [8]. This is not a hierarchical classification algorithm in itself, but the method serves to correct the hierarchical inconsistencies that result from predicting the class at each level in isolation. To contextualize this approach with the current problem, we train 8 separate classifiers, each the input for each classifier being the features extracted from the product name and the output being the probability distribution across all the possible classes at the specific level for which the classifier was trained, as depicted in Figure 1. The feature extractor for this model is a bag-of-words tf-idf vectorizer.

[1]http://www.nltk.org/nltk_data
[2]https://github.com/explosion/spaCy/blob/master/spacy/lang/en/stop_words.py
[3]https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/feature_extraction/stop_words.py

Level	Unique Categories
0	14
1	108
2	865
3	1573
4	752
5	232
6	148
7	3

Table 2: Level-wise category counts

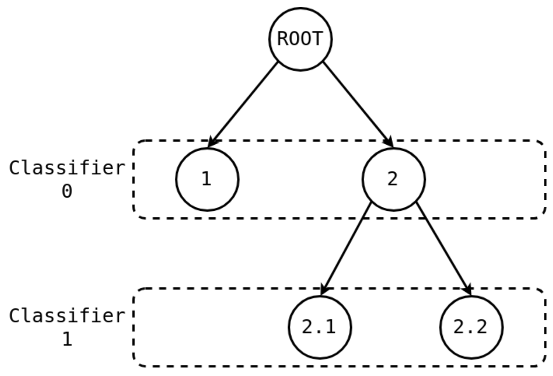

Figure 1: Classifier per level architecture

However, this method suffers from the shortcoming of disjoint predictions from level-to-level. For example, from the classifier 1 may predict a class that is not a descendant of the prediction of class 0. In this event, some additional processing is required such that the final category path is valid.

We use a greedy search within the empirically computed log-probabilities of classifiers for level ≥ 1. Instead of predicting the child class with the maximum log-likelihood predicted by the classifier, we simply chose the next level's prediction to be the valid child of the previous level's prediction with the maximum log-likelihood. An alternative to this approach would be to evaluate multiple possibilities at each level in a beam-search with the objective of maximizing sum of their log-likelihood. An obvious drawback of using this method is that an incorrect prediction at a level will result in incorrect predictions at all of the levels below it.

If a the class predicted at any level happens to be a leaf node, the search is terminated. This is a safe operation to do, as the tree nodes in the product data-set are exclusively either leaf or non-leaf nodes.

3.4 Neural Classification

We build multiple models using artificial neural networks as the function approximators with non-linear activation functions (ReLU).

3.4.1 Simple Multi-Layer Perceptron. The feature extractor for the first neural classifier uses the tf-idf weighting scheme for the words in the corpus, and a single hidden layer of neurons to predict the entire category path of each product. In this formulation, the entire category path, like '2296>3597>2989', for instance, is treated as a distinct class.

Another neural network model we tried was using the ULMFiT [7] pre-trained weights trained on Wikipedia-103 data-set and fine-tuning the language model to predict the taxonomy. This method didn't produce any better results as the data-set contained a lot of rare words and the number of examples to fine-tune the embeddings layer was simply not sufficient. We report a precision of 0.64 with this model.

3.4.2 Per-Level Multi-Layer Perceptron. We also built a neural network with the number of layers equivalent to the number of the maximum depth of the taxonomy. We employed layer-wise training and used the softmax predictions of a layer along with the word embeddings together as the input for the next layer. The model showed promising results for the initial two levels but the performance fell as the number of classes increased as we go down the tree.

3.5 Ensembles

We also build two ensemble models constructed using all the previously described classifiers. We utilize a simple voting-based scheme for combining the classifier predictions, with the majority label being selected as the final label. We built two different ensembles: (i) Combining predictions made by all the models implemented, (ii) Combining predictions for which the weighted F_1 score was at least 0.70.

4 EVALUATION

The training data is comprised of 800,000 products and their respective category path labels, a subset of which we hold out for validation. The test set is comprised of 200,000 products, on which the experiment results are reported.

The metrics used to evaluate model performance, are weighted precision, recall and a F_1 scores.

Table 3 shows the performance of each of the models against the test set.

Approach	P	R	F1
Local-Per-Level Stochastic Gradient Descent	0.50	0.49	0.46
Flat One-vs-All Multinomial Naive Bayes	0.55	0.54	0.47
Flat Stochastic Gradient Descent	0.65	0.66	0.63
Flat One-vs-All Logistic Regression	0.76	0.74	0.72
Flat One-vs-All XGBoost	0.77	0.77	0.75
Flat Multi-Layer Perceptron	0.78	0.79	0.78
Ensemble - All	0.77	0.76	0.75
Ensemble - XGBoost and Neural Networks	0.78	0.77	0.75

Table 3: Experiment Results

We observe that even simple classification approaches that ignore the hierarchical nature of the data, like the Flat MLP classifier, obtains the best scores among the different models used, with the ensembling methods achieving comparable performance.

5 DISCUSSION

Based on our experimentation and reported results, there seems to be little to no advantage in using methods that predict each level of the category path independently. This could be attributed to the fact that the evaluation metric is equally punitive for both partially correct and completely incorrect category paths, thereby, providing no extra benefit for paths that are partially correct, typically seen in piece-wise constructed methods like the per-level local classifier approach.

We evaluated a per-level local classifier, using a batched gradient-descent model and a neural model. These were post-processed to include only valid category-paths. However, these turned out to be our worst performing models. The baseline against the flat classification model using the same classification algorithm and hyper-parameters indicates that leaf-node only flat classification performs better and our subsequent evaluations were restricted to flat classifiers.

6 CONCLUSION

In this paper, we describe our approach to address the taxonomy classification task for product categorization. We build a variety of models include flat and per-level classifiers, implemented using multinomial naive-bayes, logistic regressor, batched SGD classifier and multi-layer perceptron models. Our best performing model uses a minimal amount of data-preprocessing, and uses a single hidden layer in an MLP model with non-linear activations with the features extracted using a tf-idf weighting scheme vectorizer.

In future work, we would like to explore the possibility of training a distinct model per non-leaf node to address offset the problem of having to perform a greedy search for postprocessing in the per-level classifier approach, as well as to implement additional boosting techniques to ensemble our best performing models.

REFERENCES

[1] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 785–794.

[2] Yangchi Chen, Melba M Crawford, and Joydeep Ghosh. 2004. Integrating support vector machines in a hierarchical output space decomposition framework. In *Geoscience and Remote Sensing Symposium, 2004. IGARSS'04. Proceedings. 2004 IEEE International*, Vol. 2. IEEE, 949–952.

[3] D KOLLER. 1997. Hierarchically classifying documents using very few words. In *Proc. 14th International Conference on Machine Learning*. 170–178.

[4] Shailesh Kumar, Joydeep Ghosh, and Melba M Crawford. 2002. Hierarchical fusion of multiple classifiers for hyperspectral data analysis. *Pattern Analysis & Applications* 5, 2 (2002), 210–220.

[5] Ana Carolina Lorena and André Carlos PLF de Carvalho. 2004. Comparing techniques for multiclass classification using binary SVM predictors. In *Mexican International Conference on Artificial Intelligence*. Springer, 272–281.

[6] Andrew McCallum, Ronald Rosenfeld, Tom M Mitchell, and Andrew Y Ng. 1998. Improving Text Classification by Shrinkage in a Hierarchy of Classes.. In *ICML*, Vol. 98. 359–367.

[7] Sebastian Ruder and Barbara Plank. 2018. Strong Baselines for Neural Semi-supervised Learning under Domain Shift. *arXiv preprint arXiv:1804.09530* (2018).

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