Ecommerce Product Title Classification

Sylvain Goumy
CEO, Uplab SAS
Lyon 69007
France
sylvain@uplab.fr

Mohamed-Amine Mejri
Data Scientist, Uplab SAS
Lyon 69007
France
Mohamed-amine@uplab.fr

ABSTRACT

E-commerce catalogs include a continuously growing number of products that are constantly updated. Each item in a catalog is characterized by several attributes and identified by a taxonomy label. Categorizing products with their taxonomy labels is fundamental to effectively search and organize listings in a catalog. However, manual and/or rule based approaches to categorization are not scalable.

In this paper, we explain our work for the SIGIR eCom'18 Rakuten Data Challenge [1] which focuses on the Topic of large-scale taxonomy classification. We first start with data processing. Secondly we investigate a number of feature extraction techniques and observe that TF-IDF with both bigram and unigram work best for categorization than CNN and word embedding. Finally, we evaluate several models and find than Support Vector Machines yield the highest result.

CCS CONCEPTS

Computing methodologies → Machine learning

KEYWORDS

Product Catalog; Taxonomy Classification; Machine Learning

1 INTRODUCTION

Product taxonomy categorization is a key factor in exposing products of merchants to potential online buyers. Most catalog search engines use taxonomy labels to optimize query results and match relevant listings to users' preferences. In addition to improving the quality of product search, good categorization also plays a critical role in targeted advertising, personalized recommendations and product clustering. However, there are multiple challenges in achieving good product categorization:

- The class set is very large: Machine learning applications typically only have to predict between a few selected classes (e.g. classifying an email as spam or no-spam), but in e-commerce there are often hundreds or thousands of categories that need to be classified. To train robust models for these cases, you need a particularly large amount of training data.
- Product data is diverse and unbalanced.

This naturally raises the question of whether machine learning and natural language processing can successfully handle these large-scale classification taxonomies. In this paper, we will try to explain our method to solve this problem through the rakuten data challenge.

2 DATA PREPROCESSING

After doing some data exploration of the training dataset provided by Rakuten, we took product titles and we run them through a preprocessing pipeline, mainly using the libraries 're' (Regular Expressions) and 'nltk' (Natural Language ToolKit), inspired by the methodology proposed by Shubham Jain in his blog post [2].

1. Text Case:
   Lowercasing all letters.
2. Special characters:
   Removing punctuation and special characters.
3. Stop words:
   We removed the stop words (the, and, in, etc.) and then decided to preserve them because we didn't get any improvement. Indeed, product titles are not phrases, and each words seems to have its importance.
4. Digits:
   We figured out that product titles have sometimes a lot of numerics in their text so we started by removing them because we didn't expect them to have much predictive value but after that we tried another approach which consists of replacing words that contain digits with 0 so that the algorithm deal with words like 12V and 9V as the same word: 0V.
5. Rare Words:
   Because they are so rare, the association between them and other words is dominated by noise. We started by removing words that occurs less than 3 times in text and after that we tried different values (less than 2 times, 1 time, percentage approach: terms that have less than 10% of document frequency) but finally we found that best performance was with less than 3 times.

6. Tokenization:
   Tokenization refers to dividing the text into a sequence of words or sentences (We used word_tokenize from nltk to do that).
7. Stemming & Lemmatizing:
   Finding word stems to remove variance from word inflection (i.e we want our model to know that laptops and laptop refer to the same thing). We tried different Stemmers (Snowball, Regexp, Porter) from the nltk library as well as WordNet Lemmatizer, and we decided to keep Lemmatizer has it gave better performance on this dataset.

3 FEATURE EXTRACTION

After we preprocessed text samples we want to convert them into vectors of numbers because this is the only input that machine learning algorithms can work with. We tested several methods to accomplish this:

3.1 N-grams

N-grams are the combination of multiple words used together. The basic principle behind n-grams is that they capture group of words and not only words. For example this allow us to treat 'tee shirt' as a single entity instead of two entities 'tee' and 'shirt'.

N-grams with N=1 are called unigrams. Similarly, bigrams (N=2), trigrams (N=3) and so on can also be used.

Unigrams do not usually contain as much information as compared to bigrams and trigrams. That's why we used both unigrams and bigrams to improve the model performance.

3.2 TF-IDF (Term Frequency – Inverse Document Frequency)

Similar to bag-of-words, but weighs word occurrences in a text sample higher when the words are rare in the rest of the dataset, since these words are likely to be more descriptive of the sample. Further, words with a high overall frequency in the dataset can be excluded from the lexicon. As a result, both the impact of non-informative words as well as the dimensionality of the vector space can be reduced.

3.3 Word Embedding

Word Embedding is the representation of text in the form of vectors. The underlying idea here is that similar words will have a minimum distance between their vectors.

Word embeddings models require a lot of text, so either we can train it on our training data or we can use the pre-trained word vectors developed by Google, Wiki, etc.

We used Glove embeddings [3] and specifically the 100-dimensional GloVe embeddings of 400k words computed on a 2014 dump of English Wikipedia. This was more complex to compute, but manages to create a low-dimensional text representation that encodes subtle semantic similarities between words and is easier for classifiers to train on. The implementation was inspired from this keras blog about solving a text classification problem using pre-trained word embeddings and a convolutional neural network. [4]

We achieved the best results with TF-IDF combined with both unigrams and bigrams. Even though Word embeddings definitely outperforms TF-IDF in tasks that include complex semantic relationships between text samples, it is an overkill for our use case, since product names are rather simplistic and have barely any syntax in them.

4 MODEL SELECTION

After preprocessing and vectorization, we built our classifier. We tested both accuracy and weighted-{precision, recall, F1} for a range of machine learning models in the library scikit learn such as Logistic Regression, K-nearest Neighbors, Support Vector Machines etc. To do that we used scikit learn pipeline function [5] to combine our vectorizer and classifier.

Table 1. **Experimental Results**

Model	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.80	0.79	0.80	0.79
CNN + Glove pre trained word embedding	0.75	0.67	0.66	0.65
Support Vector Machines (LinearSVC)	0.82	0.81	0.82	0.81

Support Vector Machines (LinearSVC) with *TF-IDF* vectorization performed best on both validation set and test set.

In addition to that we tested two approaches:

4.1 Single Classifier

This is the classic classification approach: train a classifier that predict the whole taxonomy label.

4.2 Top Down Classifier

This approach tries to leverage the hierarchical nature of the product taxonomy by building a classifier that predict only the 1st level category (the TopClassifier) and then a sub classifier for each branch of the categories tree (the SubClassifiers).

One benefits of this approach is that it lowers a lot the numbers of distinct labels for each classifier, thus they need less memory for training and we've been able to train them with a higher number of features, by preserving the words that appears only once or twice in the dataset, instead of limiting the features to the words that appears at least 3 times for the single classifier, which improves the global performances.

ALGORITHM 1: Top Down Classifier

```
top_category ← TopClassifier.predict(title)
if top_category = "92" then
    full_category ← "92"
else
    full_category ← SubClassifiers[top_ category].predict(title)
end
return full_category
```

Top level category "92" is a special case because it has no sub categories, thus it does not need a SubClassifier.

Here are the final performances of each classifier that we trained for realizing the Top Down Classifier Algorithm:

Table 2. **Top Classifier Performance**

Model	Train Set Size	Test Set Size	Acc.	Prec.	Rec.	F1.
TopClassifier	600 000	200 000	0.94	0.94	0.94	0.94

Table 3. **Sub Classifiers Performance**

Model	Train Docs	Test Docs	Acc.	Prec.	Rec.	F1.
SubClassifier 4015	201 221	67 074	0.85	0.84	0.85	0.84
SubClassifier 3292	150 708	50 237	0.87	0.87	0.87	0.87
SubClassifier 2199	72 535	24 179	0.95	0.95	0.95	0.95
SubClassifier 1608	64 165	21 389	0.9	0.9	0.9	0.9
SubClassifier 3625	22 167	7 390	0.84	0.82	0.84	0.82
SubClassifier 2296	21 309	7 103	0.39	0.37	0.39	0.37
SubClassifier 4238	17 646	5 883	0.79	0.78	0.79	0.78
SubClassifier 2075	15 064	5 022	0.75	0.74	0.75	0.74
SubClassifier 1395	14 135	4 712	0.76	0.75	0.76	0.74
SubClassifier 3730	6 084	2 029	0.74	0.71	0.74	0.71
SubClassifier 4564	4 236	1 412	0.76	0.74	0.76	0.74
SubClassifier 3093	3 823	1 275	0.84	0.84	0.84	0.84
SubClassifier 1208	772	258	0.66	0.66	0.66	0.63

We can see that SubClassifier performances vary a lot: from 0.39 to 0.95 accuracy.

This is due to the unbalanced nature of the dataset and the semantic nature of each category. For example, category 2296, which has the lowest accuracy contains DVD & Blu-Rays. Those products have titles that identifies their movies (eg: James Bond, Star Wars, ...), and not their type (eg: Thriller, Science-Fiction, ...), thus making it very difficult for a text classifier to perform correctly.

5 RESULTS AND DISCUSSION

5.1 Internal Evaluation

We've split the training dataset to keep 75% only for our training (600 000 products), and the remaining 25% (200 000 products) for our evaluation and performance measurement.

Table 4. **Internal Evaluation**

Model	Accuracy	Precision	Recall	F1-score
Top Down Classifier	0.90	0.90	0.90	0.90
Single Classifier	0.82	0.81	0.82	0.81

The Top Down Classifier outperforms the Single classifier on this comparison.

5.2 Rakuten Challenge Leaderboard

The Rakuten Challenge website shows a leaderboard that calculates competitor's submissions on a subset of the final test file:

Table 5. **Rakuten Leaderboard**

Submission Name	Model	Precision	Recall	F1-score
Uplab-1	Single Classifier	0.82	0.82	0.82
Uplab-2	Top Down Classifier	0.82	0.81	0.81
Uplab-3	CNN + Glove pre trained word embedding	0.67	0.66	0.65

The Single Classifier performs slightly better on Rakuten Leaderboard Evaluation, unlike our Internal Evaluation.

We can imagine from the nature of the Top Down Classifier that it may be very impacted by the balance and the categories of the test file, and this may explain why it performs better in our internal evaluation and worse in the rakuten evaluation.

6 CONCLUSIONS

For the Rakuten Challenge, we've submitted three very distinct participations:

- Single Classifier
- Top Down Classifier
- CNN + Glove pre trained word embedding

and thus tested three different classification approaches for e-commerce products.

We were surprised to see that the most complex ones were not the most performants.

Actually, a state of the art Support Vectors Machines, combined with TF-IDF vectorizer and efficient data preprocessing has proved to be the most powerful tool for this text classification challenge.

REFERENCES

[1] Rakuten Institute of Technology. 2018. *Rakuten Data Challenge:* https://sigir-ecom.github.io/data-task.html

[2] Jeffrey Pennington, Richard Socher, Christopher D. Manning. 2014. *GloVe: Global Vectors for Word Representation:* https://nlp.stanford.edu/projects/glove/

[3] François Chollet. 2016. *Using pre-trained word embeddings in a Keras model:* https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html

[4] Scikit-learn developers. 2017. *Pipeline:* http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline

[5] Shubham Jin. 2018. *Ultimate guide to deal with Text Data (using Python) – for Data Scientists & Engineers:* https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/