Large-Scale Taxonomy Problem: a Mixed Machine Learning Approach

Quentin Labernia
Tohoku University, GSIS
quentin@dais.is.tohoku.ac.jp

Yashio Kabashima
Tohoku University, GSIS
kabashima@dais.is.tohoku.ac.jp

Michimasa Irie
Tohoku University, GSIS
irie@dais.is.tohoku.ac.jp

Toshiyuki Oike
Tohoku University, GSIS
oike@dais.is.tohoku.ac.jp

Kohei Asano
Tohoku University, GSIS
asano@dais.is.tohoku.ac.jp

Jinhee Chun
Tohoku University, GSIS
jinhee@dais.is.tohoku.ac.jp

Takeshi Tokuyama
Tohoku University, GSIS
tokuyama@dais.is.tohoku.ac.jp

ABSTRACT

Rakuten Data Challenge suggests tackling the Large-Scale Taxonomy Challenge. Given a large amount of product titles and category paths leading to the product, we would like to predict the category path of a new product, based on its title. We suggest the description of a method whose scope start from data preprocessing, which represents the main task of this challenge. After embedding titles and encoding the labels using a fixed scheme, we build a two step architecture based on different machine learning models. We rely on gradient boosting method, shallow fully connected neural networks and ResNet deep neural networks models since the former is powerful enough to handle small example category trees, the second offers good results on average tree and the latter helps capturing expressive features as for complex category trees.

KEYWORDS

Machine Learning, Natural Language Processing, Deep Convolutional Neural Networks, Category Tree

1 INTRODUCTION

As part of the SIGIR 2018 workshop, we tackle in this paper the Rakuten Data Challenge. E-commerce websites handle a large variety of data and their product are categorized in some way. When a new product is given to the system, one has to compute its category. For instance, an inkjet printer is an office product, an electronic office good, and of course kind of printer. Rakuten Data Challenge addresses this problem in the following terms. Given only the title of a product, try to predict its path through some categories – as usually shown to the customer in most e-commerce website. Each path starts out from one of the root categories, reach to the actual product's category, going through some intermediate levels. For example, the product 'Replacement Viewsonic VG710 LCD Monitor 48Watt AC Adapter 12V 4A' is associated with such path `3292>114>1231`. We refer to this problem as large-scale taxonomy classification.

Rakuten Data Challenge focuses on machine learning techniques as the amount of data is large, about one million instances. Also, labels correspond to paths in tree structure. Considering these characteristics, it might seem more natural to tackle this problem using machine learning techniques rather than typical data mining methodologies like FCA – Formal Concept Analysis – or logical rules based approaches. Such approaches indeed leads to high computation complexity and could simply fail because it doesn't fit as it the actual data features and labels.

Up to now, datasets which are representative of practical usage haven't been made public yet and Rakuten Data Challenge provides such dataset. However, it brings with it serious challenges. Among those, the large size of the dataset represents one of the main issue. This justifies the large-scale classification appellation. One indeed needs to handle around 0.8 million of instances each of them consisting of a textual feature – title of the product - and categories' path description. The output label consists of an ordered list of categories/. We are dealing with a classification problem and each distinct list is a different label. Considering the test dataset, we come up with more than 3000 distinct list configurations. Labels are also unbalanced with regard to the input instances. Some categories like "4015" foster a lot of product whereas some others don't. Handling very small classes along with over-representative ones represents another big issue. The last considered point is related to the *noisy* semantic of each instance's title. We are referring here to the fact that the title of a product can include the name of the product or its description which is more or less precise. Although two products share some words, it might be the case they are actually very different and so are not categorized under the same label. For instance, a laptop whose name is given after a given fruit, a book which talks about this fruit: both share common words. To get rid of such mistakes is a fundamental and necessary condition for our model to predict the most correct labels.

In this paper, we describe our methodology to deal with such problem. We first underline the given data structure and put emphasize on some important hypothesis we infer from the provided dataset. Based on these hypothesis, we preprocess the provided data and give a detailed explanation of how do we address most of the previously mentioned problems in a logical way. Once the input and output data are formatted in the right way, we define the architecture of the models we built to perform the prediction. Based on this architecture, we outline the experiment strategy. The conclusion recaps the theoretical advantages and drawbacks of our modelization.

2 DATA OVERVIEW

The provided data is composed of one million instances, split between the training dataset (80%) and test dataset (20%). A pair (x_i, y_i) denotes the instance of index i. A sample of data picked up from the training dataset is shown in table 1.

The input data – title x_i – is composed of one feature formated as a string of UTF-8 characters. It corresponds to the title of the product. A title x_i has a specific length and potentially contains special characters such as the trademark character.

Title x_i	**Path** y_i
Circuit Breaker Standard BR-330	2199>4592>12
Nike Sun Sport Visor 343278 Black	1608>2227>574>522
Style & Co. Jacquard Cowl-Neck Top	1608>4269>3031>62

Table 1: Data sampled from the training dataset. Categories are given as integers.

The taxonomy is a tree as stated in the Rakuten Data Challenge rules. After gathering all the nodes, each actually corresponds to one category, we end up with a forest of 14 trees. One tree is sort of very general category of product. It is obviously possible to link the root node of all the trees together so we form a unique tree. However, we chose to not perform such operation since it would make more sense to build specialized models managing one of these general categories at a time. Indeed, the classification logic between electronic products and make-up goods – for instance – could be potentially totally different. As a result, we consider the existence of $T_1, \ldots, T_{14}$, the trees we build from the given learning dataset only. A node of a tree T_i is a category $c \in \mathbb{N}$. This category corresponds to the node in the forest, which means that such category c is unique across all the trees. Notice that tree T_{14} is degenerated in the sense that it contains only one node.

The output data, or labels, are tuples of categories. One tuple $y_i = (c_1, \ldots, c_{D_k})$ describes a path of categories – associated with the instance x_i–, that is: c_1 is a root node of one tree T_k, c_2 is one of the c_1 child nodes, and so on until we reach c_{D_k}, the end of the path. After running analysis on the provided datasets, we found that c_{D_k} is actually always a leaf, thus we consider the following hypothesis to always be true: for any instance (x_i, y_i), the last component of y_i, c_{D_k}, is a leaf of a tree. Another analysis result is that given a tree, paths may have different length. However, since it is easier to consider the same length across every paths of a same tree k, we pad paths with dummy nodes in such a way every path have the same length D_k with D_k denoting the height of the tree k. Labels are strings initially formated like "c_1>c_2>...>c_{D_k}". A predicted tuple $\tilde{y}_i$ matches its ground truth counterpart y_i if and only if both tuples are equal – perfect matching between the two. Put another words, our goal is to find a tuple with the right amount of components and each of these components have to match the provided ground truth.

We didn't proceed to any data augmentation, either for data processing or model usage. This paper focuses on getting the best results while just relying on the title as input feature. Here, we find more interests to look at how we could obtain consistent results with semantically *noisy* title features as recalled in the introduction.

Let's fix an important notation hereafter used: all the variables written with a tilde is a predicted value from our models. Next section introduces the data preprocessing based on the previously stated hypothesis.

3 DATA PREPROCESSING

Let's first take a look at the input feature transformation process. A title x_i is a string of characters whose encoding is UTF-8. We split the title into words using space characters. The integer q_i denotes the amount of words in the title x_i. We define a binary representation of each word and so create a dictionary using the previously split words. The size of the vocabulary corresponds to the number of distinct words we meet in the training dataset because we build it using the training dataset. We drop the least and "most" frequent words if respectively: a word appears less than ten times ; a word's frequency for each tree is high and about the same across the trees – such words are not discriminative across different categories. When dealing with an instance x_i, all the words which doesn't belong to the dictionary are simply dropped. Any word $w_{i,j}$ – word j of the title x_i – can be encoded into a binary vector of size ω. One can notice we don't proceed to any lemmatization of the words. We refer to the bag of word vector of the title x_i using $\mathbf{x_i}$. The next step is the reducing the dimensionality of each words of the title. To do so, we use the well known *word2vec* technique[6]. We set each resulting word to be an element of $\mathbb{R}^{2000}$. Since titles are not of fixed size, we need to either build a model which can handle such variable length inputs, or transform the data into a predefined fixed format. Since our words are embedded in sort of semantic space using *word2vec*, we state that it makes sense to perform the addition of all the words of one title to get a significant meaning. Considering titles of Internet marketplaces, words order usually doesn't matter. For instance, "`red shoes`" and "`shoes / red`" should be meaningful speaking quite close. Moreover it gives us an elegant way to take into account all the words in the title while having a final fixed size. To recap, given x_i a title, we first embed its binary vector words $w_{i,1}, \ldots, w_{i,q_i}$ in a semantic space and obtain $u_{i,1}, \ldots, u_{i,q_i}$. We then aggregate all the words of as a sum in a title to form the final title $t_i = \sum_{1 \leqslant j \leqslant q_i} u_{i,j}$. We feed the machine learning model with these t_i.

We now deal with the labels. Labels are originally formated like "c_1>c_2>...>c_{D_k}" as mentioned in section 2. However, such format can't be efficiently used as output label because it would have to be parsed. Each leaf is reached by only one path so it is theoretically possible to proceed to the prediction task considering multiclass – an instance is associated with one and only one leaf, which also corresponds to only one path. By doing so, we don't take any advantage of the tree structure hypothesis. Also, there exist some paths containing a very small number of instances. Therefore, we consider the tree structure and suggest another way of performing prediction. We choose to transform these raw labels y_i into binary vectors z_i. Our goal is to describe the path from the root node to one leaf by giving to which direction we go at each level. An easy way to proceed is to first sort all the nodes by level, then we encode which node we go through using a binary vector. For instance, the

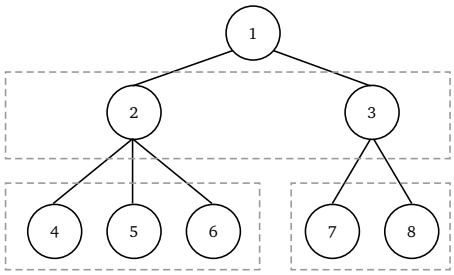

Figure 1: Example of a category tree. Valid labels are paths starting from the root node 1 and reaching one of the leaves 4 to 8. One can also notice any valid label share the same length. Each box represents the local information one need to take into account when walking from the root node.

path 1>3>8 shown in figure 1 could be written as:

$$(\underbrace{0,1}_{\text{level 1}},\underbrace{0,0,0,\ 0,1}_{\text{level 2}})$$

The first level group tells us to keep going through the node 3 – within a level, we use category integer representation to order nodes. The first subgroup of the second level is filled out with zeros for the reason that it corresponds to the nodes reachable from 2. Since it is mandatory to go through the root node, we ignore it and start out from the second level of the tree. If the tree is big, it implies very large labels. In order to reduce the size of such label, we use a simple trick as a reworked idea of anchor representation, used for instance in [5]. This idea is used in the case of image recognition where the number of contained labels can differ between images.

A label y_i belongs to exactly one of the 14 trees, thus we only consider the k^{th} tree. The height of this tree is D_k as defined in section 2. We also recall that the depth of a node is its distance to the root node. We define $L_{k,d}$ to be the set of nodes of depth d and the set $M_{k,d} = \{|C| \mid C \subseteq L_{k,d} \wedge \text{every nodes in C have the same parent}\}$ with $d > 0$. The set $M_{k,d}$ gathers the number of children of the nodes in the level $d-1$. The compressed vector z_i is of length $r_i = \sum_{1 \leqslant d \leqslant D_k} \max M_{k,d}$ and can be represented as follows:

$$z_i = (\underbrace{\dots}_{\text{level 1}:\ G_{k,1} := \max M_{k,1}} \quad \dots \quad \underbrace{\dots}_{\text{level } D_k:\ G_{k,D_k} := \max M_{k,D_k}})$$

Looking at the figure 1 as an illustration, each group of components has the size of the biggest box at the considered level. After choosing a node g in the path, we can only reach its children child(g). Therefore the way we compress is by considering child(g) – one box in figure 1 – instead of $L_{k,d}$ – the whole layer. However, depending on g, the number of elements in child(g) might differ, which is why we encode a level d using $G_{k,d}$ components – the biggest box in one level. It ensures we can always encode child(g) while setting up a fixed size. Let's fix an ordering for each child(g). Since we only pick exactly one node per level, we set up the n^{th} component to be 1 if and only if we select the n^{th} node of child(g). Otherwise, we put 0. That is, the previous 1>3>8 path of figure 1 can be rewritten like so:

$$(\underbrace{0,1}_{\text{level 1}:\ G_{k,1}=2},\ \underbrace{0,1,0}_{\text{level 2}:\ G_{k,2}=3})$$

By doing so, we drastically reduce the size of the labels and give a fixed representation given the tree k: indeed, recall from section 2 that given any tree k, all its labels share the same length. Albeit this representation is dependent on the tree we choose, it is not a problem here since we build a specific model for each tree.

We obviously need to first compute for each tree its level representation, that is $G_{k,1}, \dots, G_{k,D_k}$. Then it is possible to transform any y_i into its corresponding z_i. Conversely, it allows us to get the raw format label $\tilde{y}_i$ back from the predicted $\tilde{z}_i$.

4 MODELS DESCRIPTION

The present system focuses on a combination between boosting, shallow neural networks and deep learning techniques. Since we deal with the 14 trees and don't gather them into a unique one, we have to build a two stage model architecture. In order to predict the label of x_i, we feed t_i to a first model $\mathcal{A}$ whose task is to predict in which tree $T_1, \dots, T_{14}$ this instance belongs to. We also build 13 models, one per not degenerated tree. Put another words, the prediction process can be summed up in the following way. Let's consider an instance x_i. We feed the first stage model $\mathcal{A}$ with t_i – recall that t_i is the preprocessed title coming from x_i. If the instance is classified as k – it means the instance belongs to T_k –, we feed the k^{th} second stage model $\mathcal{B}_k$ corresponding to the tree T_k with the same t_i than before and the output is the corresponding vector describing the label – category path as introduced in section 3. We finally operate the transformation which gives us $\tilde{y}_i$ back from $\tilde{z}_i$. If the predicted tree is the 14^{th} one, the answer becomes obvious since T_{14} have only one node.

Each of the model $\mathcal{A}$ and $\mathcal{B}_1, \dots, \mathcal{B}_{13}$ is either a tree based gradient tree boosting, shallow neural network or a deep neural network. Note that each models are independent. Depending on the characteristics of a tree, we rely on one of the following three machine learning methods.

First, when the amount of examples is low we apply gradient tree boosting method which is a well known ensemble learning method[1]. Such boosting method is often used in data analyzing competition and produce good results. Since the amount of examples for the considered trees is small, we also take into account a specific word embedding. Formally, given a tree k, we regard the value of each leaf node c_{D_k} of a path $y_i = (c_1, \dots, c_{D_k})$ as a target label and apply gradient tree boosting method on multiclass classification task. As for the input features, we simply make usage of the corresponding bag of words $\mathbf{x}_i$ of the title x_i. We feed to the gradient tree boosting algorithm such example pairs of the considered tree. As for the implementation, we rely on gradient tree boosting provided by scikit-learn library (www.scikit-learn.org). Parameters are not fixed yet.

Second, when the trees hold enough training data, we use simple shallow feed-forward fully connected neural networks, composed of three layers.

Finally, we consider hard tree structures, that is trees whose labels z_i have a high dimension. For this purpose, we rely on deep neural networks, inspired by image processing technique. Since

Hyperparameter	Value	Models
Model topology	ResNet50[2] or Shallow NN	All
Regularization	$\ell 2$-regularization	All
Loss function	Softmax cross entropy	$\mathcal{A}$
	$\forall k \in [\![1, 13]\!], \mathcal{L}_k$	$\mathcal{B}$s
Optimizer	Adam[3]	All
Batch scheme	Mini-batches ($\leqslant 32$)	All

Table 2: Gathered specifications as for neural networks models. Parameters of gradient tree boosting is not fixed yet. We use the following notation $\mathcal{B}$s $\coloneqq \mathcal{B}_1, \ldots, \mathcal{B}_{13}$.

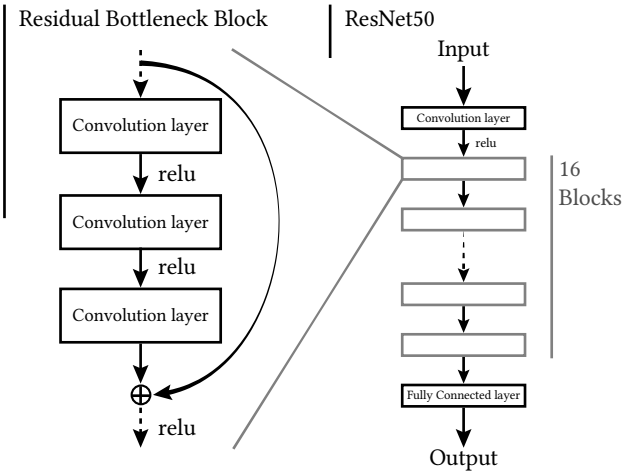

Figure 2: Broad representation of the ResNet50 architecture. The number of stacked layers facilitates relevant features extraction and shortcuts help to train. Each block contains a bottleneck convolutional structure.

we deal with fixed inputs, we don't consider recurrent networks and instead focuses on deep convolutional neural networks. Let's recall that inputs t_i are embedded in a 2000 dimensional semantic space created using *word2vec*[6] and then aggregating all the words in the title using summation. We want to take advantage of convolution in order to retrieve expressive features before applying classical fully connected layers at the end of the network. To that purpose we suggest the use of ResNet[2] in its 50 layer flavor. Such architecture is based on residual blocks composed of three convolutional layers and a shortcut link between the input and output of the layer. A more detailed view of the architecture is shown figure 2. It allows us to enjoy the expressiveness of a deep architecture and is known to be easier to train than other deep convolutional models like VGG[7] or AlexNet[4]. All models $\mathcal{A}$ and $\mathcal{B}_1, \ldots, \mathcal{B}_{13}$ are built on this ResNet50 architecture. What changes is only the last output fully connected layer. The output size is obviously 14 for the model $\mathcal{A}$ because we need to choose among 14 trees to which instances belong. For each $\mathcal{B}_k$, the corresponding output size is $\sum_{1 \leqslant d \leqslant D_k} G_{k,d}$ as defined in section 3.

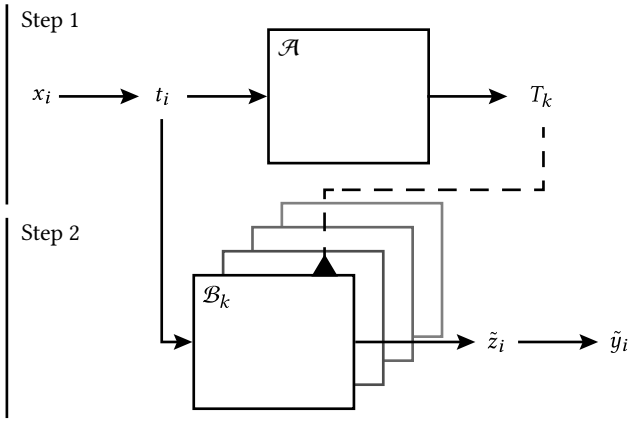

Figure 3: Overview of the two stage architecture for prediction. It includes model $\mathcal{A}$ as first step and depending on its result, the corresponding model among $\mathcal{B}_1, \ldots, \mathcal{B}_{13}$ handles the input as second step. If the output of the first step is T_{14}, the second step returns a hardcoded answer.

As stated above, the output size of the model $\mathcal{A}$ is a 14 dimensional vector. Since we only pick one tree among the 14 it makes sense to represent the output by a probability distribution. That is why we use the softmax cross entropy – a softmax operation is applied just before the cross entropy operation since it transform the raw output into a probability vector – as loss function. By doing so, we can treat the output in a relevant way. On the other hands, neural network models among $\mathcal{B}_1, \ldots, \mathcal{B}_{13}$ output z_i. A label z_i contains multiple 1, thus they don't represent a probability distribution. Actually, each group $G_1, \ldots, G_{D_k}$ of z_i is a probability distribution itself. Then we choose to decompose the loss function in such a way we apply the softmax cross entropy over each group, and we finally aggregate the resulting values using summation. If we denote each group of z_i by $g_{i,1}, \ldots, g_{i,D_k}$ – which are of respective length $G_{k,1}, \ldots, G_{k,D_k}$ – then the loss function $\mathcal{L}_k$ for the model $\mathcal{B}_k$ of tree T_k is given by:

$$\mathcal{L}_k(\tilde{z}_i, z_i) = \sum_{1 \leqslant d \leqslant D_k} \text{cross entropy}(\text{softmax}(\tilde{g}_{i,d}), g_{i,d})$$

This formulation allows us to consider each group independently as probability distributions and thus captures the way we handle our walk in the tree k.

All neural network models $\mathcal{A}$ and among $\mathcal{B}_1, \ldots, \mathcal{B}_{13}$ are trained using mini-batches of reasonably small size[9] – typically less than 32 mini-batch size. The main specifications of these neural network architectures are grouped in the table 2. We use chainer[8] as for the neural networks implementation.

5 CONCLUSION

In this work, we suggest a two layer architecture which tackles the large-scale taxonomy challenge as part of Rakuten Data Challenge in SIGIR 2018 workshop. While putting the accent on machine learning methodology, data preprocessing represents a major point

of attention. We embed each words of the vocabulary in a semantic space and aggregate words of title using summation. Labels describe the path from the root node of the considered category tree and leads to a leaf. We provide a fixed encoding scheme for all the paths of the same tree. We rely on three different machine learning techniques: gradient tree boosting, shallow feed-forward fully connected neural networks and ResNet architecture. We build a two step architecture to predict the final path associated with a newly seen instance. The first step model $\mathcal{A}$ choose which tree we have to consider – k –, we then feed the right second step model – $\mathcal{B}_k$ – and it outputs the encoded path.

The two step architecture splits the prediction process into multiple specialized models as they are independently trained. Gradient tree boosting handles trees whose number of example is not enough to perform relevant neural networks training. Shallow network seems to be surprisingly effective to classify in simple tree configurations. The deep convolutional approach using ResNet finally allows us to process large and complex trees, while being easier to train compared to other deep convolutional models.

ACKNOWLEDGEMENT

This work was funded by ImPACT Program of Council for Science, Technology and Innovation.

REFERENCES

[1] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). http://arxiv.org/abs/1512.03385

[3] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). http://arxiv.org/abs/1412.6980

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. 2015. SSD: Single Shot MultiBox Detector. *CoRR* abs/1512.02325 (2015). http://arxiv.org/abs/1512.02325

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *CoRR* abs/1310.4546 (2013). http://arxiv.org/abs/1310.4546

[7] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). http://arxiv.org/abs/1409.1556

[8] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. 2015. Chainer: a Next-Generation Open Source Framework for Deep Learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*. http://learningsys.org/papers/LearningSys_2015_paper_33.pdf

[9] D. Randall Wilson and Tony R. Martinez. 2003. The General Inefficiency of Batch Training for Gradient Descent Learning. *Neural Netw.* 16, 10 (Dec. 2003), 1429–1451. DOI: http://dx.doi.org/10.1016/S0893-6080(03)00138-2