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 these products, we would like to predict the category path of a given product, only based on its title. The provided paths are structured as a forest of 14 trees. The learning process is split into two steps: we first retrieve the tree the input belongs to and then handle the category path. We describe data embedding which represents an important task in this challenge and then introduced the so-called two step architecture. The *original idea* is based on deep neural network model. We also introduce an *actual method* as second step modification since the former second step is not efficient enough. This last technique makes usage of multiple sets of random forest classifiers to navigate inside each tree.

KEYWORDS

Machine Learning, Natural Language Processing, Deep Neural Network, Random Forest

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 products 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, reaches 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 lead to high computation complexity and could simply fail because of the not adequate input and output data representation.

Up to now, datasets which are representative of practical usage of e-commerce websites have not 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 and we put emphasis on that point in section 5. Some categories like "4015" foster a lot of product whereas some others do not. 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 labels in the most efficient way.

In this paper, we introduce two methods to tackle the challenge. The *original method* refers to our first idea which results are not convincing. Hence, we also introduce the *actual method* whose differences with the original one will be discussed in section 4. This last method has been used to get results of Rakuten Challenge shown in section 5.

As for data preprocessing, we 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

*Corresponding Author

In: J. Degenhardt, G. Di Fabbrizio, S. Kallumadi, M. Kumar, Y.-C. Lin, A. Trotman, H. Zhao (eds.): Proceedings of the SIGIR 2018 eCom workshop, 12 July, 2018, Ann Arbor, Michigan, USA, published at http://ceur-ws.org

right way, we define the architecture of the models we built to perform the prediction. Both methods are introduced. Based on this architecture, we outline the experiment strategy. Finally, we compare differences between the two strategies and provide critical explanations of our work.

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. 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 do not 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 us 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

3.1 Input Features

Let's first take a look at the input feature transformation process. We describe two steps corresponding to *bag of words* and *word2vec* processes. Both representations are further used in section 4 and so we introduce them once at a time as follows.

3.1.1 Bag of words. 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 number of words in the title x_i. We define a binary vector 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 all the trees – such words are not discriminative across different categories. When dealing with an instance x_i, any word which does not belong to the dictionary is 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 do not proceed to any lemmatization of the words. We refer to the bag of word vector of the title x_i using $\mathbf{x_i}$.

3.1.2 word2vec. The next step is dimensionality reduction of each words in the title $\mathbf{x_i}$. To do so, we use the well known *word2vec* technique[7]. 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 does not 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 $\mathbf{x_i}$ 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 using sum 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.

3.2 Output labels

This section is specifically designed for the *original method*. Labels are originally formated like "c_1>c_2>. . .>c_{D_k}" as mentioned in section 2. However, such format cannot 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 do not take any advantage of the tree structure hypothesis. Also, there exist some paths containing a very small number

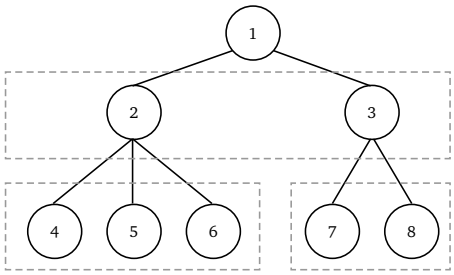

Figure 1: Example of a category tree. Valid labels are paths starting from the root node 1 and reaching one of the leaves 4 to 8. One can also notice any valid label share the same length. Each box represents the local information one need to take into account when walking from the root node.

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 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 [6]. 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:

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

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

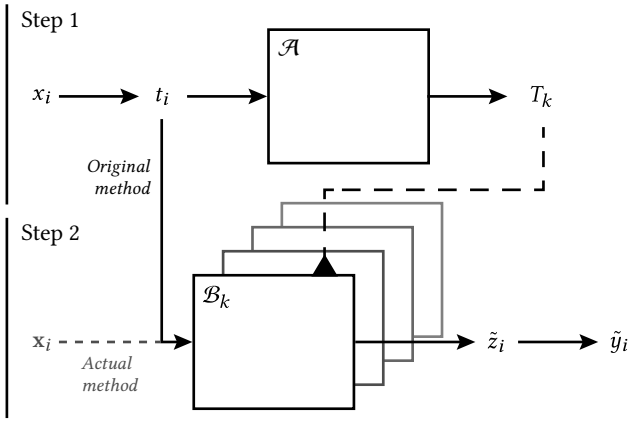

Figure 2: (*Both methods*) Overview of the two step architecture for prediction. It includes model $\mathcal{A}$ as first step and depending on its result, the corresponding model among $\mathcal{B}_1, \ldots, \mathcal{B}_{13}$ handles the input as second step.

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 $G_{k,1}, \ldots, G_{k,D_k}$. 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

This section describes two different methods as recalled in section 1: our *original* idea along with the *practical method*. Both are based on a two step model architecture since we deal with the 14 trees separately and do not gather them into a unique one. The process is to first predict the tree T_i the current input belongs to, and then find the entire path in that selected tree as the second step. First step model is denoted by $\mathcal{A}$, while the second step ones are written $\mathcal{B}_1, \ldots, \mathcal{B}_{13}$. Step one is common to both methodologies and only the latter part is what changes between the two. We put in section 4.1 and section 4.2 the respective explanations for step one and step two.

4.1 First step: classification over the trees

Let us consider an instance x_i. We feed the model $\mathcal{A}$ with t_i – recall that t_i is the preprocessed title coming from x_i. Our goal is to first

determine to which tree this instance belongs to. For that purpose, we define the model $\mathcal{A}$ to be a neural network, whose output is a class between 1 and 14. If the instance x_i is classified as $k \in [\![1; 14]\!]$, it means the instance is part of the tree T_k and thus further consider the model $\mathcal{B}_k$ as second stage process. We use a five layers fully connected neural network. Since we classify over 14 trees, model's output size is also 14. Loss function is softmax cross entropy as loss function since any input title is associated with exactly one tree – the use of softmax operation transforms the raw output into a probability vector.

4.2 Second step: prediction of the path

4.2.1 Original Method. Each of the model $\mathcal{B}_1, \ldots, \mathcal{B}_{13}$ is either a shallow neural network or a deep neural network. All the models are independent each other. Depending on the characteristics of a tree, we rely on different machine learning methods.

First, when trees have simple structures, we use shallow feed-forward fully connected neural networks, each of these composed of three layers. The model outputs z_i. A label z_i contains multiple 1, thus they do not 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:

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

Next, 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 we deal with fixed inputs, we do not consider recurrent networks and instead focuses on deep convolutional neural networks. Let us recall that inputs t_i are embedded in a 2000 dimensional semantic space created using *word2vec*[7] 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 3. 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[8] or AlexNet[4]. The loss function is same as the former shallow fully connected neural networks' one.

4.2.2 Actual Method. As second step, we construct hierarchical models. Let us recall that step one provides us the tree k the current input belongs to. Put another words, we know the first category c_1 of the path $y_i = (c_1, \ldots, c_{D_k})$. In order to predict the full path, we construct 13 hierarchical classifiers $\mathcal{B}_1, \ldots, \mathcal{B}_{13}$ whose task is to predict the rest of the path c_2 to c_{D_k} once at a time. Notice that as for tree 14, the answer becomes obvious since T_{14} has only one node.

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: (*Original method*) Gathered specifications as for neural networks models. We use the following notation $\mathcal{B}\mathbf{s} \coloneqq \mathcal{B}_1, \ldots, \mathcal{B}_{13}$.

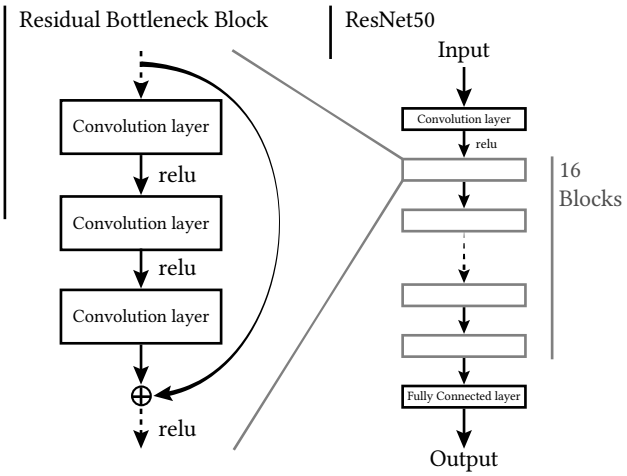

Figure 3: (*Original method*) Broad representation of the ResNet50 architecture. The number of stacked layers facilitates relevant features extraction and shortcuts help to train. Each block contains a bottleneck convolutional structure.

We build the second model $\mathcal{B}_k$ of the tree k as follows. It is based on a set of models $\mathcal{B}_{k,n}$ where n is a parent node label of the tree k. The goal of each classifier $\mathcal{B}_{k,n}$ is to output the child node corresponding to the input title. Starting from the root node of the tree k, we can navigate in the tree by predicting the nodes of the path y_i, once at a time. Theoretically, this strategy seems to be computationally expensive since the number of parent node increases exponentially with the layers. However, the maximum number of node is less than 1000, which means that we can afford such strategy.

Each classifier $\mathcal{B}_{k,n}$ is fed with bag of words titles $\mathbf{x}_i$ and outputs the child node of n of the tree k which corresponds to our input. We use random forest[1, 5] algorithm which has got 150 trees – this hyperparameter is fixed for all $\mathcal{B}_{k,n}$. As for the implementation, we rely on the open source library *scikit-learn* (http://scikit-learn.org/). Algorithm 1 shows how to build the model $\mathcal{B}_k$. When classifying a title x_i in tree k, we call PredictionProcedure($k, \mathbf{x}_i, \top_k, \varnothing$) as described in algorithm 2. The symbol $\top_k$ refers to the root node of the tree k. This simple algorithm consists in starting from the root node

and going deeper once at a time by choosing the corresponding classifier inside the set $\mathcal{B}_k$.

Algorithm 1 HierarchicalRandomForest

Input Current node n
Training data $S_n = \{(\mathbf{x}, c_n) \mid c_n \in \text{child}(n)\}$
Output Trained model

if n is not a leaf node **then**
  $\mathcal{B}_{k,n} \leftarrow \text{RandomForest}(S)$
  $\mathcal{B}_k \leftarrow \{\mathcal{B}_{k,n}\}$
  for $m \in \text{child}(n)$ **do**
    $S_m \leftarrow \{(\mathbf{x}, c_m) \mid c_m \in \text{child}(c)\}$
    $\mathcal{B}_k \leftarrow \mathcal{B}_k \cup \text{HierarchicalRandomForest}(m, S_m)$
  end for
  return $\mathcal{B}_k$
else
  return $\emptyset$ # It reaches leaf node.
end if

Algorithm 2 PredictionProcedure

Input Tree index k
Title as bag of words embedding $\mathbf{x}$
Current node n
Current path p
Output Augmented path

while n is not a leaf node **do**
  $p \leftarrow p$ with c_n appended
  $c_n \leftarrow \mathcal{B}_{k,n}(\mathbf{x})$
  PredictionProcedure(k, $\mathbf{x}$, c_n, p)
end while
$p \leftarrow p$ with c_n appended
return p

5 EXPERIMENTAL RESULTS

In this section, we report results of the Sigir Rakuten workshop official evaluation. As stated in section 2, 0.8 million train data and 0.2 million test data are provided. As for test data, ground truth is not given. A leaderboard system shows up the results at each stage of the evaluation process. The computed evaluation metrics are weighted-{precision, recall, f1} on the test set of *exact* CategoryId-Path match and a ranking is set over the f1-score. Visible in table 3 results we showed the best result in each original and actual model. The Rakuten Data Challenge is split into two evaluation stages. The first stage corresponds to the evaluation over a subset of the test dataset and multiple submissions are possible. On the other hand, the second stage consists of only one evaluation of the method over the whole test dataset. Both original and actual methods have been tested during the first stage while only the actual one is used for the second stage. Table 3 summarizes these evaluations along with their corresponding evaluation measures. The final ranking of the method is 20^{th} over 28 teams.

It is clear that our original method does not provide satisfactory results. We suppose the main reason relies on how complex category trees structure is and the number of data available for training considering all distinct paths. It has to be said that this original method's issue is overfitting. The first consideration is that given a tree, it might happen that a small number of training data is only available. Even considering tree with large amount of data, the *original method* fails to provide good generalization property. Indeed, going deeper in the tree means having less data available for training. Since deep neural networks are used in this context, deep levels cannot be learned properly. This leads to get good results on the training set and a very poor generalization ability. On the other hand, the *actual method* provides good performance because it relies on decision trees – more precisely, random forest algorithm – combined with a simple bag of words embedding. Predicting independently children of each node of the tree with such technique gives us insurance that the learning process will lead to good and comprehensible results even if near the leaves.

Method	Precision	Recall	F1
Original (Stage 1)	≈ 0.10	≈ 0.08	≈ 0.07
Actual (Stage 1)	0.75	0.76	0.74
Actual (Stage 2)	0.77	0.76	0.75

Table 3: Results of the two methods for each stage.

6 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 create bag of words then embed each words of the vocabulary in a semantic space and aggregate words of title using summation. Bag of words are used for the *actual method*'s second step while the further embedding is used for the rest. We build a two step architecture to predict the final path associated with a newly seen instance. The first step model $\mathcal{A}$ chooses which tree we have to consider – for instance the k^{th} –, we then feed the right second step model – $\mathcal{B}_k$ – to output the encoded path. This second step is split into two different methods: *original* and *actual*. Our original idea relies on a one shot fixed scheme encoding of the whole category path. We rely on several machine learning techniques: shallow feed-forward fully connected neural networks and ResNet architecture. As for the *actual method*, we make usage of hierarchical classification model composed of random forests models. This method provide nice performances compared to the former one as shown in section 5.

Because of the imbalanced classes and complex data structures, our original strategy does not offer good performances. However, we think that such method becomes very efficient when having enough samples and a uniform distribution over the classes. Since this situation appears in the shallow levels of most of the trees, we think it could be worthy of combining the *original method* with the *actual* one in order to improve overall performances.

ACKNOWLEDGEMENT

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

REFERENCES

[1] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (01 Oct 2001), 5–32. DOI: http://dx.doi.org/10.1023/A:1010933404324

[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] Andy Liaw, Matthew Wiener, and others. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.

[6] 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

[7] 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

[8] 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