Overview of the SIGIR 2018 eCom Rakuten Data Challenge

Yiu-Chang Lin
Rakuten Institute of Technology
Boston, MA, 02110 - USA
yiuchang.lin@rakuten.com

Pradipto Das
Rakuten Institute of Technology
Boston, MA, 02110 - USA
pradipto.das@rakuten.com

Ankur Datta
Rakuten Institute of Technology
Boston, MA, 02110 - USA
ankur.datta@rakuten.com

ABSTRACT

This paper presents an overview of the SIGIR 2018 eCom Rakuten Data Challenge. In this data challenge, `Rakuten.com` has released a sampling of one million product titles and the corresponding anonymized category paths from their entire product catalog. Of these, 0.8 million of product titles and their corresponding category paths are released as training data and 0.2 million of product titles are released as test data. The task is to predict the category, defined as a full path in the taxonomy tree as provided in the training set, of the product titles in the test set. The evaluation is divided into two stages to measure system performance on a part of the test data and the entire test data. The different systems are evaluated using weighted precision, recall and F1. In total, 26 teams have submitted 28 systems with a top performance of 0.8513 weighted F1 score.

1 INTRODUCTION

The SIGIR 2018 eCom Rakuten Data Challenge [1] is organized by `Rakuten Institute of Technology, Boston`[2] (RIT-Boston), a dedicated R&D organization for the Rakuten group[3] of companies. The data challenge focuses on the task of large-scale taxonomy classification of product titles, where the goal is to predict each product's category, defined as a full path from root node to a leaf node in the taxonomy tree provided in the training set. The cataloging of product listings through taxonomy categorization is a fundamental problem for any e-commerce platform, with applications ranging from basic data organization, personalized search recommendations to query understanding and targeted campaigning. For instance, in the `Rakuten.com` catalog, "Dr. Martens Air Wair 1460 Mens Leather Ankle Boots" is categorized under the "Clothing, Shoes & Accessories > Shoes > Men > Boots" leaf. However, manual and rule based approaches to categorization are not scalable since commercial product taxonomies are organized in tree structures with three to ten levels of depth and thousands of leaf nodes. Advances in this area of research have been limited due to the lack of real data from actual commercial catalogs.

The challenge presents several interesting research aspects due to the intrinsic noisy nature of the product labels, the size of modern e-commerce catalogs, and the typical imbalanced data distribution.

---

[1] https://sigir-ecom.github.io/data-task.html
[2] https://rit.rakuten.co.jp/
[3] https://www.rakuten.com/

We hope that by making the data available to the participants, the task will attract more research institutions and industry practitioners, who do not have the opportunity to contribute their ideas due to the lack of an actual commercial e-commerce catalog data. In a typical e-commerce setting, merchants are responsible for matching their products with an existing category, which is a leaf node in the taxonomy tree. The problem of large scale taxonomy classification is thus immensely useful to help merchants upload their products in the right places of an e-commerce platform catalog.

2 DATASET

Rakuten has released a sample of one million product listings, including the training (0.8 million) and test (0.2 million) set, consisting of product titles and their corresponding category paths that belong to a taxonomy tree to describe the varying degrees of generality of the items in the catalog. A product taxonomy is a tree-based hierarchical representation of labels of the listings in a catalog.

Rakuten's catalog of products is much larger than a million listings. The larger catalog of product listings is first de-duplicated using leaf node label and product title tuples as keys and then a random sampling of one million listings is performed. These set of listings have been allowed to be published for this year's data challenge. The set of one million product listings may not cover the entire set of leaf nodes in `Rakuten.com`'s taxonomy, and so, any taxonomy generated from the category labels provided with the training set may yield a truncated taxonomy.

Further, to preserve anonymity of the taxonomy, the nodes in the actual taxonomy have been masked with random integers. The class or the category label of a product listing is thus a path from the root of the taxonomy tree to a leaf node where the listing resides. In the left to right ordering of the nodes in such a path, the path becomes more specific in describing the listing as it approaches the leaf level. The string representation of a path from root to leaf is henceforth dubbed as a "category-id-path".

The training data file is in a tab-separated values (TSV) format where each line contains a product title and its corresponding category-id-path. In the test set, only the product titles are provided and the objective of this data challenge is to predict the full category-id-path for each such title. Table 1 and table 2 show some examples of product titles from the training set and test set, respectively. The partitioning of the training and the test sets are obtained using category-wise stratified sampling of the one million listing dataset.

2.1 Data Characteristics

In the training set, there are 800, 000 product titles from 3, 008 leaf-level nodes. Product titles are unevenly distributed among these 3, 008 categories. The top ten categories compose around 30% of the data set and the top forty categories compose around 50% of
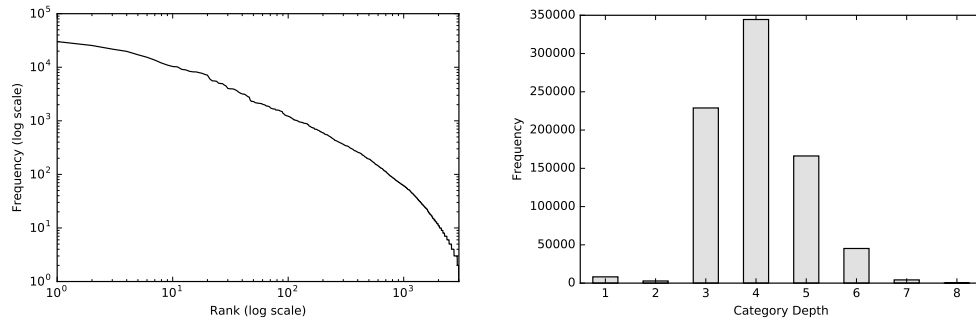
Figure 1: *Left*: **Product title frequencies for 3,008 leaf nodes in the taxonomy tree. Both X and Y axis are in log scale. Rank ranges from 1 to 3,008.** *Right*: **Product title frequencies for different lengths of category paths.**
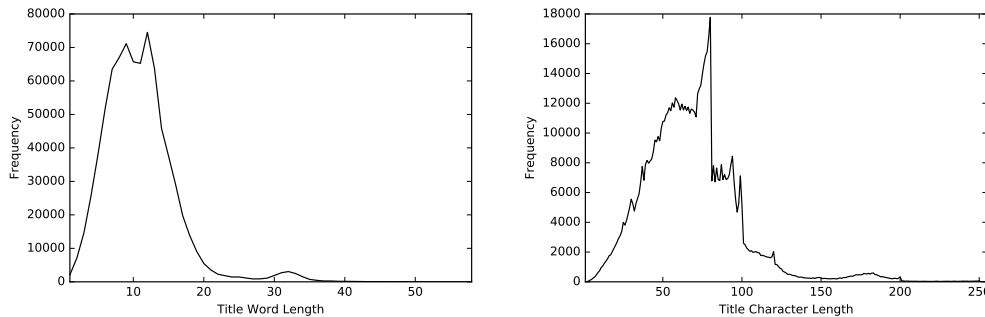

Figure 2: *Left*: **Word length distribution over all product titles in the training set.** *Right*: **Character length distribution over all product titles in the training set.**

Product Titles	category-id-paths
Replacement Viewsonic VG710 LCD Monitor 48Watt AC Adapter 12V 4A	3292>114>1231
Ka-Bar Desert MULE Serrated Folding Knife	4238>321>753>3121
5.11 TACTICAL 74280 Taclite TDU Pants, R/M, Dark Navy	4015>3285>1443>20
Skechers 4lb S Grip Jogging Weight set of 2- Black	2075>945>2183>3863
Generations Small Side Table White	4015>3636>1319>1409>3606

Table 1: Examples of product titles from the training set.

Product Titles
Disc Brake Rotor-Advanced Technology Rear Raybestos 980368
Coquette Neon Pink Ruffle Babydoll 7035 Neon Pink One Size Fits All
12V 7Ah (SPS Brand) APC NS3000RMT3U Replacement Battery (4 Pack)
Honda Ridgeline 2006-08 Black Radio AM FM 6 Disc CD PN 39100-SJC-A100 Face 3TS1
Frankford Arsenal Platinum Series Case Prep Essentials Kit

Table 2: Examples or product titles from the test set.

the data set. The left hand side of Figure 1 shows the distribution of product title frequency across all the leaf-level categories (sorted by frequency) where the **X** and **Y** axes are both in log scale. Across the $800,000$ product titles, the maximum depth of category level is 8 and the average depth is 4.

The right hand side of Figure 1 shows the distribution of product title frequency across different depths of category-id-paths. The average word-level title length over all categories, after the surface form of the title is tokenized by white space, is 10.93, with a maximum length of 58 words. Similarly, the average character-level title

length is 68.44, with a maximum length of 255. Figure 2 shows the title length distributions at world-level and character-level, respectively.

3 EVALUATION

In this section, we briefly mention the evaluation criteria used to judge the different systems. Although we do not resort to a stricter statistical significance testing of different systems using confidence interval estimation from bootstrapped samples of the test set, it will be useful to do so once this pilot task has been completed. One motivation to not introduce such testing in this pilot task is that in an e-commerce setting, even a 0.01% improvement means thousands of listings being assigned to their right places in the catalog without human intervention.

3.1 Metrics

The metrics that we have used to evaluate the different classification systems are based on weighted-precision, weighted-recall and weighted-F1 for the test set of **exact** "category-id-path" match. In other words, partial path match does not count as a correct prediction. We assume that there are a total of K classes, $\{c_i | i = 1, 2, ..., K\}$ in the training set. The number of true instances for each class (support) is n_i, and the total number of training instances is $N =$

Rank	Team	P	R	F1	Submission Time(UTC)
1	mcskinner	0.8734	0.8425	0.8510	2018-06-23 08:56:23
2	MKANEMAS	0.8509	0.8445	0.8421	2018-06-23 22:49:38
3	tiger	0.8552	0.8389	0.8404	2018-06-23 16:00:01
4	Uplab	0.8435	0.8427	0.8375	2018-06-23 10:22:33
5	JCWRY	0.8545	0.8172	0.8278	2018-06-23 00:56:34
6	neko	0.8311	0.8296	0.8245	2018-06-23 11:18:16
7	Ravenclaw	0.8394	0.8118	0.8197	2018-06-01 15:37:05
8	ssdragon	0.8310	0.8173	0.8185	2018-06-23 23:55:15
9	RITB-Baseline	0.8389	0.8097	0.8172	2018-06-21 10:06:25
10	inception	0.8364	0.8087	0.8166	2018-05-10 05:11:22
11	Uplab-2	0.8196	0.8228	0.8149	2018-05-30 08:50:59
12	minimono	0.8119	0.8042	0.8020	2018-05-18 00:25:06
13	Tyche	0.8536	0.7655	0.7976	2018-06-20 08:12:12
14	Topsig	0.8009	0.8042	0.7967	2018-06-07 00:19:27
15	VanGuard	0.7950	0.7915	0.7871	2018-05-10 18:21:06
16	Waterloo	0.7819	0.7853	0.7767	2018-06-17 15:20:03
17	CorUmBc	0.7822	0.7722	0.7702	2018-06-08 21:21:02
18	Sam-chan	0.7695	0.7704	0.7617	2018-05-10 06:38:18
19	Tyken2018	0.7545	0.7561	0.7431	2018-06-22 14:13:30
20	Or	0.7446	0.7232	0.7226	2018-05-25 11:52:26
21	HSJX-ITEC-YU	0.7274	0.7241	0.7176	2018-06-23 02:43:20
22	Coumodo	0.7304	0.7116	0.7078	2018-06-01 23:37:30
23	Uplab-3	0.6700	0.6571	0.6480	2018-05-29 15:34:15
24	the1owl	0.5664	0.6220	0.5603	2018-04-23 23:46:54
25	sherlock	0.5862	0.5124	0.5065	2018-04-24 01:40:50
26	B4_toku	0.3843	0.3630	0.3467	2018-06-23 10:09:41
27	Hawk	0.2129	0.0544	0.0626	2018-05-16 03:12:29
28	Fractal AIML	0.0149	0.0152	0.0150	2018-05-17 13:24:18

Rank	Team	P	R	F1	Submission Time(UTC)
1	mcskinner	0.8697	0.8418	0.8513	2018-06-23 08:56:23
2	MKANEMAS	0.8425	0.8427	0.8399	2018-06-23 22:49:38
3	tiger	0.8397	0.8428	0.8379	2018-06-23 23:59:59
4	Uplab	0.8368	0.8419	0.8366	2018-06-23 10:22:33
5	JCWRY	0.8528	0.8172	0.8295	2018-06-23 00:56:34
6	neko	0.8267	0.8305	0.8256	2018-06-23 11:18:16
7	Ravenclaw	0.8289	0.8114	0.8175	2018-06-01 15:37:05
8	Uplab-2	0.8186	0.8243	0.8173	2018-05-30 08:50:59
9	ssdragon	0.8226	0.8163	0.8172	2018-06-23 23:59:59
10	RITB-Baseline	0.8276	0.8077	0.8142	2018-06-21 10:06:25
11	inception	0.8259	0.8077	0.8139	2018-05-10 05:11:22
12	Tyche	0.8599	0.7644	0.8004	2018-06-20 08:12:12
13	minimono	0.8019	0.8023	0.7994	2018-05-18 00:25:06
14	Topsig	0.7921	0.8014	0.7941	2018-06-07 00:19:27
15	VanGuard	0.7899	0.7917	0.7884	2018-05-10 18:21:06
16	HSJX-ITEC-YU	0.7809	0.7821	0.7790	2018-06-23 23:59:59
17	Waterloo	0.7802	0.7857	0.7781	2018-06-17 15:20:03
18	CorUmBc	0.7745	0.7712	0.7690	2018-06-08 21:21:02
19	Sam-chan	0.7718	0.7745	0.7666	2018-05-10 06:38:18
20	Tyken2018	0.7654	0.7603	0.7509	2018-06-22 14:13:30
21	Or	0.7419	0.7250	0.7245	2018-05-25 11:52:26
22	Coumodo	0.7275	0.7140	0.7107	2018-06-01 23:37:30
23	Uplab-3	0.6698	0.6588	0.6509	2018-05-29 15:34:15
24	the1owl	0.5947	0.6277	0.5682	2018-04-23 23:46:54
25	sherlock	0.5855	0.5091	0.5025	2018-04-24 01:40:50
26	B4_toku	0.4340	0.4751	0.4144	2018-06-23 10:09:41
27	Hawk	0.2679	0.0561	0.0642	2018-05-16 03:12:29
28	Fractal AIML	0.0148	0.0152	0.0150	2018-05-17 13:24:18

Figure 3: *Left*: **Stage 1 leaderboard (evaluated on the first 20,000 test titles).** *Right*: **Stage 2 leaderboard (evaluated on all 200,000 test titles).**

$\sum_{i=1}^{K} n_i$. After calculating the precision (P_i), recall (R_i) and F1 ($F1_i$) scores for each class c_i, the weighted-precision, weighted-recall and weighted-F1 are defined as follows:

$$weighted\text{-}P = \sum_{i=1}^{K} \frac{n_i}{N} P_i \qquad (1)$$

$$weighted\text{-}R = \sum_{i=1}^{K} \frac{n_i}{N} R_i \qquad (2)$$

$$weighted\text{-}F1 = \sum_{i=1}^{K} \frac{n_i}{N} F1_i \qquad (3)$$

It can be shown that weighted-recall is actually equal to absolute accuracy. Since the product distribution over the taxonomy tree is highly imbalanced, weighted version of precision, recall, and F1 make much more sense than macro or micro version of precision, recall, and F1 do. The evaluation script is also provided during the data challenge[4].

___
[4]https://github.com/sigir-ecom/dataChallenge

3.2 Leaderboard

The leaderboard shows the weighted precision, recall and F1 scores, upto four decimal digits of precision, for the latest submissions from each participating team. The corresponding file submission time is shown as well so participants can refer the scores to which submission it belongs to. The leaderboard is sorted by the weighted F1 score. We choose not to use an off-the-shelf classifier, such as a logistic regression model or some other standard classifier, as a minimal baseline (lowerbound) in the leaderboard.

3.3 Timeline

- Stage 1 (April 9 - June 23): Participants build and test models on the training data. Each team can make at most three submissions per day in this stage. The leaderboard (Fig. 3 left) only shows the model performance on a **subset** of the test set, i.e., the first 20, 000 test product titles. Note that this subset is randomized since the entire test set is randomized using stratified sampling on the one million titles.
- Stage 2 (June 24): The leaderboard switches to Stage 2 on June 24 (Fig. 3 right) and shows the model performance on the **entire** test set, i.e., all 200, 000 test product titles. No submission is allowed once the leaderboard has been switched to this stage.

4 RIT-BOSTON BASELINE METHOD

The RITB-Baseline method uses the method of Joulin et al. [4], which has been very popular recently for large-scale multi-label text classification. The method, dubbed, fastText, uses lock free asynchronous gradient descent optimization [8] of a log linear loss function – it is actually a single layer perceptron with **no** non-linearities in the activation functions. For details on the derivations for the fastText model parameters and using a softmax loss function, see Section 7.

Generally, in a production setting, we have seen gradient boosted decision trees (GBDTs) to perform just as well as state-of-the-art Convolutional Neural Networks (CNNs), which has been previously reported in [1]. However, although, GBDTs have the added advantage of incorporating intuitive feature engineering as well as deciding feature importance, their training time on large datasets can be prohibitively slow. Recent trends in e-commerce industry has thus shifted to using fastText or other deep learning approaches for generic text classification problems, of which taxonomy classification of product titles is a specific instance. Deep learning techniques are a continuously evolving family of models with varying degrees of architecture engineering, optimization algorithms and parameter fine tuning methods and as such it is difficult to assign a baseline model using this family of methods.

We thus choose to use fastText as a quick way to solve the taxonomy classification problem. Further fastText *apparently* has less parameters to tune than comparable deep learning methods. The reason for the choice of the word *apparent* will become clear once we explain the graphs in Fig. 5. The fastText method in [4] employs many tricks, including hashing to a fixed vocabulary size of n-grams, lock free asynchronous gradient descent optimization [8], averaging of word representations as document representation and others. Further, given a training set containing N_V words from a vocabulary of V words and a number of passes over the data, i.e. epochs, equal to E, the learning rate, α, at time t is updated according to

$$\alpha^{(t)} = \gamma(1 - \frac{t}{N_V E}) \qquad (4)$$

where γ is a fixed parameter. Hence, unlike optimizing for the learning rate or step size found in more principled convex optimization techniques, fastText uses an heuristic estimate and a linear decrement of the learning rate that allows it to calculate an estimated time of completion for the training phase to end.

The high degree of class imbalance becomes apparent upon observing the Kullback Leibler (KL) divergences of the empirical distribution over data points in the training set for each of the top level categories, from their respective uniform distributions (see Fig. 4). The RDC Challenge dataset may have exhibited this phenomenon due to the dataset selection process, however, in general, depending on common choices of e-commerce taxonomies for mid-size organizations, KL divergences usually vary between 1.0 and 2.0 [1].

We initially split the 0.8M training dataset into a 10% Dev2 set, a 10% Dev1 set after excluding the Dev2 set, and the rest for training and cross-validation. The splitting is done using stratified sampling. Following a bi-level classification scheme in [1], we initially built
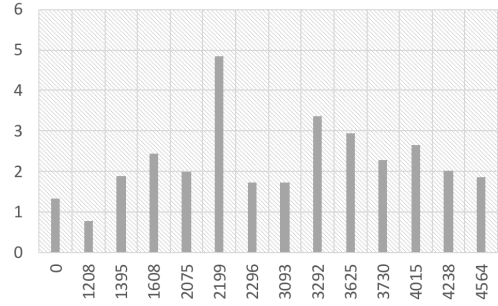

Figure 4: Class imbalance measured as KL divergences of empirical distributions over data proportions in top level categories from their respective uniform distributions.

a baseline hierarchical cascading classifier using fastText, however, for this dataset, we found out that a "flat" fastText classifier performed better by at least one percentage point absolute. Additionally, the "negative sampling" loss function has also shown better performance than the typical "softmax" loss. However, fine tuning of parameters based on the negative sampling loss function can show signs of overfitting.

For data preprocessing, we use the token normalizer mentioned in [1]. We first tokenize using whitespace and lowercase the tokens. Then for each token, we separate any adjoining punctuations from numbers (decimal or otherwise). We replace all numbers with a @NUMBER@ literal token and remove all punctuation tokens. We preserve alphanumeric tokens in the hope that sometimes longer alphanumeric strings encode model numbers that are unique to each category. For instance, the surface form of the title, "*"Ana Silver Co Rainbow Moonstone Earrings 2 1/4"" (925 Sterling Silver) - Handmade Jewelry EARR346812""* gets converted to "*ana silver co rainbow moonstone earrings @NUMBER@ @NUMBER@ @NUMBER@ @NUMBER@ sterling silver handmade jewelry earr34681*" using our normalization scheme. For the baseline, we also do not remove any stop words or perform any morphological operations on the resulting tokens. The entire training dataset is de-duplicated using the (category label, title normal form) tuples.

The plots in Fig. 5 confirms the non-deterministic nature of fastText due to the use of asynchronous SGD. In fastText, each thread reads its own view of the data, creates a mini-batch of size one, and updates the parameter matrices without any locking mechanism. This type of lock free parameter updates have been shown to be just as effective as their synchronous counterparts under the assumptions of feature sparsity in the data [8]. The plots in Fig. 5 show absolute accuracy results on our 10% Dev2 set for two different scenarios, after training a bi-level fastText classifier on the training data that results after excluding the Dev1 and Dev2 sets. For the first scenario, we run fastText for forty runs with the same parameter settings of dim=min(120,K/2), epoch=100, wordNgrams=3, loss=softmax, thread=45, with K being the number of classes for a particular subtree rooted at a classification node in the taxonomy tree. For this scenario, we obtain the mean absolute accuracy to be 74.0908 with a standard deviation of 0.0741. For the second scenario, we run fastText for six different runs with the same settings, except that the number of threads are set to 1, 2, 5,
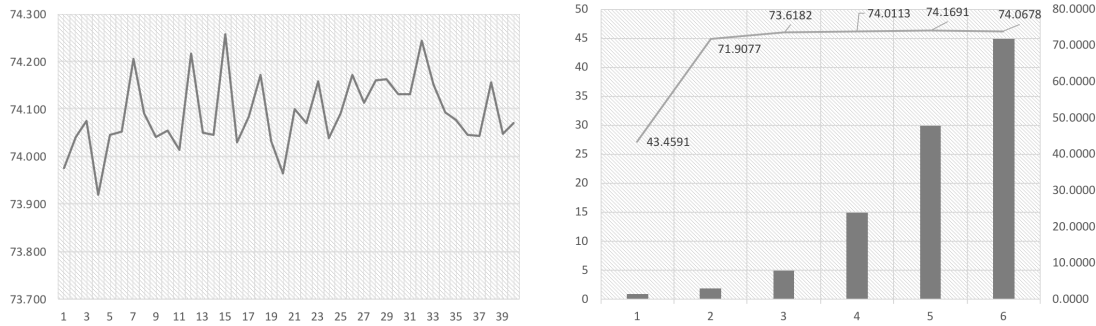
Figure 5: Absolute accuracy results on our 10% Dev2 set for the following two scenarios. *Left*: **Forty different runs of `fastText` using the exact same settings.** *Right*: **Six different runs of `fastText` using the same settings except for the number of threads.**

15, 30 and 45 respectively. The solid bars in the right plot of Fig. 5 shows the number of threads and the corresponding accuracy numbers are shown on the line graph above the solid bars.

We can observe from the graph in the right of Fig. 5, that the trivially synchronous version of `fastText`, i.e. using just one thread, is the weakest of all models. This makes sense since `fastText` is just a singe layer perceptron with no non-linearities. Accuracy improves by almost 70% when the number of threads is set to two, but, plateaus off for the other settings of thread counts after that. For our Dev2 set, we obtain the best accuracy using thirty threads and that is the number of threads we set for all of our `fastText` baseline runs. The asynchronous SGD of `fastText` with a mini-batch size of one, raises a legitimate concern – the number of threads *apparently* behaves as a parameter that regulates generalization performance and is machine architecture dependent. Needless to mention that this behavior is independent of the loss function used. In our experiments, the default setting of twelve threads have shown lower performance than thirty threads for this task.

After some minimal experiments on our Dev2 set, we finalized the settings of our `fastText` baseline, **RITB-Baseline** (see Table 3 and Fig. 3), to be the following: `-dim 300 -minn 4 -maxn 10 -wordNgrams 3 -neg 10 -loss ns -epoch 3000 -thread 30`. We have set a high number of epochs due to the availability of free computational cycles in our servers.

In the next section, we briefly highlight the methods used for the data challenge task by the teams who have submitted their system description papers.

5 SYSTEM DESCRIPTIONS

Below is a list of each team's systems. Their team names, affiliation and system description paper title available on the workshop website can be seen in Table 3.

- **Team CorUmBc** submitted one system (0.7690 F1 score) based on a Bidirectional Long Short Term Memory Network (BiLSTM) to capture the context information for each word, followed by a multi-head attention model to aggregate useful information from these words as the final representation of the product title. Their model adopts an end-to-end architecture without any hand-crafted features and regulated by various dropout techniques.

- **Team HSJX-ITEC-YU** submitted one system (0.7790 F1 score) that uses K Nearest Neighbors (KNN) classification model and the Best Match (BM) 25 probabilistic information retrieval model. The top k product title matches of the BM25 model are then classified by the KNN model.

- **Team JCWRY** submitted one system (0.8295 F1 score) that uses deep convolutional neural networks with oversampling, threshold moving and error correct output coding to predict product taxonomies. Their best accuracy is obtained through an esemble of multiple networks, such as Kim-CNN [5] and Zhang-CNN [9], trained on different extracted features inputs, inlcuding doc2vec [6], NER and POS features.

- **Team MKANEMAS** submitted one system (0.8399 F1 score) that formulates the task as a simple classification problem of all leaf categories in the given dataset. The key feature of their system is the combination of a Convolutional Neural Network and Bidirectional LSTM using ad-hoc features generated from an external dataset (Amazon Product Data) [3][6].

- **Team mcskinner** submitted one system (0.8513 F1 score) using a straightforward network architecture and ensemble LSTM strategy to achieve competitive results. The positive impact of tightening the connections between recurrent and output layers through the use of pooling layers is also demonstrated. The author similarly provides practical details on their training methodology and algorithms for probability calibration. The final solution is produced by a bidirectional ensemble of 6 LSTMs with Balanced Pooling View architecture.

- **Team minimono** submitted one system (0.7994 F1 score) based on word-level sequence-to-sequence neural networks widely used in machine translation and automatic document summarization. By treating taxonomy classification as a translation problem from a description of a product to a category path. The text of the product name is viewed as the encoder input and the sequence of category name as the decoder output.

- **Team neko** submitted one system (0.8256 F1 score) that treats category prediction as a sequence generation task. The authors built a word-level sequence-to-sequence model

Team Name	Affiliation	System Description Paper
CorUmBc	University of Maryland Baltimore County	Large Scale Taxonomy Classification using BiLSTM with Self-Attention
HSJX-ITEC-YU	York University & Trent University	A Best Match KNN-based Approach for Large-scale Product Categorization
JCWRY	Individual Researchers	An Empirical Study of Using An Ensemble Model in E-commerce Taxonomy Classification Challenge
MKANEMAS	Yahoo Japan Corporation	Convolutional Neural Network and Bidirectional LSTM Based Taxonomy Classification Using External Dataset at SIGIR eCom Data Challenge
mcskinner	Individual Researcher	Product Categorization with LSTMs and Balanced Pooling Views
minimono	University of Tsukuba	Encoder-Decoder neural networks for taxonomy classification
neko	Rakuten Institute of Technology Singapore	Unconstrained Production Categorization with Sequence-to-Sequence Models
RITB-Baseline	Rakuten Institute of Technology Boston	Overview of the SIGIR 2018 eCom Rakuten Data Challenge
Tyken2018	Tohoku University	Large-Scale Taxonomy Problem: a Mixed Machine Learning Approach
Topsig	Queensland University of Technology	TopSig at the SIGIR'eCom 2018 Rakuten Data Challenge
tiger	JD.com	Multi-level Deep Learning based E-commerce Product Categorization
Uplab	Uplab SAS	Ecommerce Product Title Classification
Waterloo	University of Waterloo	Team Waterloo at the SIGIR E-Commerce Data Challenge

Table 3: List of participants who submitted system description papers. All submitted papers are available at: https://sigir-ecom. github.io/accepted-papers.html.

to generate non-constrained product categories that are not limited to the labels from the training set. The final model is an ensemble of several attentional sequence-to-sequence models.

- **Team Tyken2018** submitted one system (0.7509 F1 score) that builds upon a two-step architecture where the first step classifies the input product title to the genre tree it belongs to and the second step predicts its category path. For the path prediction, two methods are proposed, *original method* and *actual method*. They originally proposed a shallow feed-forward fully connected or deep neural network, i.e., 50-layer ResNet architecture [2] depending on how complex the tree structure is, but constructed a hierarchical style model in the end.
- **Team Topsig** submitted one system (0.7941 F1 score) that utilizes random indexing to create topological signatures (TopSig) to categorize the product names in the provided data sets. The authors make use of their open-source Top-Sig tool to generate dimensionality-reduced signatures and search for these signatures. The signature generation approach used by TopSig has been shown to be effective at different tasks. The approach can be regarded as a variation on LSH to approximate nearest neighbour search.
- **Team tiger** submitted one system (0.8379 F1 score) that combines machine learning, deep learning, and natural language processing to create a multi-level and multi-class deep learning tree method. This includes multiple models based on single-label and multi-level label predictions, as well as characteristics of the product tree structure. The training dataset and the test dataset are merged to pre-train word vectors to calculate semantic similarity. To address the unbalanced

category issue, sampling and data enhancement techniques are used. They build eight sample datasets according to the category hierarchy and develop two classification algorithms to build models for different levels and search paths using category trees.

- **Team Uplab** submitted three systems based on different classifier types, including single flat linear Support Vector Machines classifier (0.8366 F1 score), a top down esemble which combines top-level and sub-level classifiers (0.8173 F1 score) and a CNN with pre-trained word embeddings (0.6509 F1 score). The authors found that TF-IDF with both bi-gram and uni-gram features work best for categorization.
- **Team Waterloo** submitted one system (0.7781 F1 score) that uses multinomial naive-bayes, logistic regression, batched gradient-descent and neural softmax classifiers retro-fitted for the hierarchical classification objective. The authors evaluate different classification methods, including flat and hierarchical classification, and implement a general framework to tackle the task of taxonomy classification.

6 RESULTS

Twenty six research groups/individuals participated in the data challenge. Each team submitted their predicted category-id-path on the test set in the same TSV format as the training set (Table 1). We score the system submissions against the gold standard using weighted precision, recall and F1 as defined in Section 3. The final results are summarized in Figure 3. The left hand side of Figure 3 shows the leaderboard in Stage 1 (evaluated on the first 20,000 test titles) and the right hand side shows stage 2 results (evaluated on the entire set of test titles).

In Stage 1, the top five teams are **team mcskinner**, **team MKANE-MAS**, **team tiger**, **team Uplab** and **team JCWRY** with 0.8510, 0.8421, 0.8404, 0.8375 and 0.8278 weighted-F1 scores, respectively. In Stage 2, the same five teams rank at top five with 0.8513, 0.8399, 0.8379, 0.8366 and 0.8295 weighted-F1 scores. The rank does not change much between Stage 1 and Stage 2, except for **team Uplab-2**, **team Tyche** and **team HSJX-ITEC-YU**. Table 4 shows the list of teams whose ranks is higher in Stage 2 than in Stage 1.

Team Name	Δ Rank	Stage 1 Rank	Stage 2 Rank
Uplab-2	+3	11	8
Tyche	+1	13	12
HSJX-ITEC-YU	+5	16	21

Table 4: List of teams that rank higher in Stage 1 than in Stage 2.

With regards to F1 scores, the difference between Stage 1 and Stage 2 is less than 0.01 except for **team HSJX-ITEC-YU**, whose score difference is more than 0.06. Table 5 shows the the list of top twenty teams whose F1 score is higher in Stage 2 than in Stage 1.

Team Name	Stage 1 F1	Stage 2 F1	Stage 2 Rank
mcskinner	0.8510	0.8513	1
JCWRY	0.8278	0.8295	5
neko	0.8245	0.8256	6
Uplab-2	0.8149	0.8173	8
Tyche	0.7976	0.8004	12
VanGuard	0.7871	0.7884	15
HSJX-ITEC-YU	0.7176	0.7790	16
Waterloo	0.7767	0.7781	17
Sam-chan	0.7617	0.7666	19
Tyken2018	0.7431	0.7509	20

Table 5: List of the top twenty teams that have higher F1 score in Stage 1 than in Stage 2.

REFERENCES

[1] Pradipto Das, Yandi Xia, Aaron Levine, Giuseppe Di Fabbrizio, and Ankur Datta. 2017. Web-scale language-independent cataloging of noisy product listings for e-commerce. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, Vol. 1. 969–979.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[3] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. International World Wide Web Conferences Steering Committee, 507–517.

[4] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, 427–431.

[5] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882* (2014).

[6] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 43–52.

[7] Lam Nguyen, Phuong Ha Nguyen, Marten van Dijk, Peter Richtarik, Katya Scheinberg, and Martin Takac. 2018. SGD and Hogwild! Convergence Without the Bounded Gradients Assumption. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Jennifer Dy and Andreas Krause (Eds.), Vol. 80. PMLR, Stockholmsmässan, Stockholm Sweden, 3747–3755.

[8] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 693–701.

[9] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*. 649–657.

7 APPENDIX

The loss function for `fastText`, mentioned in [4], is the following log likelihood function:

$$L = \log\left(\prod_{n=1}^{N} f(BA\mathbf{x}_n)^{t_n}\right) = \sum_{n=1}^{N} t_n \log(f(BA\mathbf{x}_n)) = \sum_{n=1}^{N} L_n \quad (5)$$

where t_n is the label of the nth training instance. The optimization problem on the training set is to find the optimal parameters $\mathbf{A}^*$ and $\mathbf{B}^*$ by minimizing the negative log likelihood in Equ. 5.

$$\operatorname*{arg\,min}_{\mathbf{A}^*, \mathbf{B}^*} -\frac{1}{N} \times \sum_{i=1}^{N} L_n \quad (6)$$

The parameter matrix A connects the input features to the units in the hidden layer of dimension H. This matrix is a look-up table over the words [4] and H is a parameter of the model. The weight parameter matrix, B, connects the hidden units to the output units. The kth output unit of the nth instance contributes to the conditional probability of the output $y_{n,k}$, of the kth label being 0 or 1. That is, for a particular n^{th} instance $(\mathbf{x}_n, t_n)$, we have, $L_n = \prod_{k=1}^{K} y_{n,k}^{t_{n,k}}$ where $t_{n,k} \in \{0, 1\}$.

From now on, let $\mathcal{L} = -L_n$, so that we can clear up the notation. Also, the data is assumed to be i.i.d so that $p(\mathbf{X}) = \prod_{n=1}^{N} p(\mathbf{x}_n)$. Let the input nodes be labeled as $x_i, i \in \{1, ..., D\}$; the hidden layer nodes be labeled as $z_j, j \in \{1, ..., H\}$ and the output layer nodes be labeled as $v_k, k \in \{1, ..., K\}$. The target for the input $\mathbf{x} = \{x_d\}$ is a one-hot K vector with the kth entry to be 1 if $\mathbf{x}$ belongs to class k. Let v_k be the input to the kth output unit i.e. $v_k = \mathbf{B}_k^T \mathbf{z}$ with $z_h = \left(\sum_{d=1}^{D} A_{d,h} x_d\right)$.

Using the method of maximum likelihood estimation on the loss function L, where $t_k \in \{0, 1\}$ and $y_k = p(v_k) \in [0, 1]$, or, equivalently, using gradient descent minimization on $\mathcal{L}$, we can derive the updates for model parameter estimators.

7.1 Parameter updates for the output layer

We first begin by finding the updates to the **B** parameter from Equ. 6. For brevity, we drop the subscript n henceforth. Taking derivatives of the loss function at the output layer w.r.t. $B_{k,h}$, we obtain,

$$\frac{\partial \mathcal{L}}{\partial B_{k,h}} = -(t_k - y_k)z_h \quad (7)$$

where, $y_k = p(v_k) = softmax(v_k) = e^{\mathbf{B}_k^T \mathbf{z}} / \sum_{k'=1}^{K} e^{\mathbf{B}_{k'}^T \mathbf{z}}$

Derivation: For a particular n^{th} instance, we have

$$
\begin{aligned}
-\mathcal{L} &= \sum_{k=1}^{K} t_k \log y_k \\
&= \sum_{k=1}^{K} t_k \left(v_k - \log \sum_{k'=1}^{K} \exp(v_{k'}) \right) \quad (8) \\
&= \sum_{k=1}^{K} t_k \left(\mathbf{B}_k^T \mathbf{z} - \log \sum_{k'=1}^{K} \exp(\mathbf{B}_k^T \mathbf{z}) \right) \\
&= \sum_{k=1}^{K} t_k \left(\sum_{h=1}^{H} B_{k,h} z_h - \log \sum_{k'=1}^{K} \exp\left(\sum_{h=1}^{H} B_{k,h} z_h \right) \right) \quad (9)
\end{aligned}
$$

Using chain rule of derivatives,

$$
\frac{\partial \mathcal{L}}{\partial B_{k,h}} = \frac{\partial \mathcal{L}}{\partial v_k} \frac{\partial v_k}{\partial B_{k,h}}
$$

For the first term, we have:

$$
\begin{aligned}
-\frac{\partial \mathcal{L}}{\partial v_k} &= t_k - \frac{\partial \left(\log \sum_{k'=1}^{K} \exp(v_{k'}) \right)}{\partial v_{k'}} \frac{\partial v_{k'}}{\partial v_k} \\
&= t_k - \frac{\exp(v_{k'})}{\log \sum_{k'=1}^{K} \exp(v_{k'})} \delta_{k'}^k \\
&= t_k - y_k
\end{aligned}
$$

For the second term, we have:

$$
\begin{aligned}
\frac{\partial v_k}{\partial B_{k,h}} &= \frac{\partial \left(\sum_{h=1}^{H} B_{k,h} z_h \right)}{\partial B_{k,h}} \\
&= z_h
\end{aligned}
$$

leading to,

$$
\frac{\partial \mathcal{L}}{\partial B_{k,h}} = -(t_k - y_k) z_h \quad (10)
$$

Thus for the $B_{k,h}$ parameter update, we have:

$$
\begin{aligned}
B_{k,h}^{(t+1)} &= B_{k,h}^{(t)} + \alpha \times -\frac{\partial \mathcal{L}}{\partial B_{k,h}^{(t)}} \\
&= B_{k,h}^{(t)} + \alpha \times \left(t_k - y_k^{(t)} \right) z_h^{(t)} \quad (11)
\end{aligned}
$$

where α is the learning rate that is updated as shown in Equ. 4.

7.2 Parameter updates for the input layer

Taking derivatives of the loss function at the output layer w.r.t. $A_{h,d}$, we obtain,

$$
\frac{\partial \mathcal{L}}{\partial A_{h,d}} = -\left(\sum_{k=1}^{K} (t_k - y_k) B_{k,h} \right) x_d \quad (12)
$$

Derivation: As in the previous section, for a particular n^{th} instance, we have, using the chain rule of derivatives,

$$
\begin{aligned}
-\frac{\partial \mathcal{L}}{\partial A_{h,d}} &= \sum_{k=1}^{K} \left(\frac{\partial \mathcal{L}}{\partial v_k} \frac{\partial v_k}{\partial z_h} \right) \frac{\partial z_h}{\partial A_{h,d}} \\
&= \sum_{k=1}^{K} \left((t_k - y_k) \frac{\partial \left(\sum_{h=1}^{H} B_{k,h} z_h \right)}{\partial z_h} \right) \frac{\partial z_h}{\partial A_{h,d}} \\
&= \sum_{k=1}^{K} \left((t_k - y_k) B_{k,h} \right) \frac{\partial \left(\sum_{d=1}^{D} A_{h,d} x_d \right)}{\partial A_{h,d}}
\end{aligned}
$$

leading to,

$$
\frac{\partial \mathcal{L}}{\partial A_{h,d}} = -\left(\sum_{k=1}^{K} (t_k - y_k) B_{k,h} \right) x_d \quad (13)
$$

Thus for the $A_{h,d}$ parameter estimator update, we have:

$$
\begin{aligned}
A_{h,d}^{(t+1)} &= A_{h,d}^{(t)} + \alpha \times -\frac{\partial \mathcal{L}}{\partial A_{h,d}^{(t)}} \\
&= A_{h,d}^{(t)} + \alpha \times \left(\sum_{k=1}^{K} \left(t_k - y_k^{(t)} \right) B_{k,h}^{(t)} \right) x_d \quad (14)
\end{aligned}
$$

Note that the document or sentence representation in fastText is obtained by an averaging of the vector representations of the constituent n-grams. Thus, the gradients involving $\mathbf{x}$ are normalized by the number of word and/or character n-grams in the input $\mathbf{x}$.

7.3 Asynchronous Stochastic Gradient Descent

The general expression for updating the estimators of a parameter $\mathbf{w}$, in a gradient descent optimization scheme is given by,

$$
\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \alpha \mathbf{u} \quad (15)
$$

where α is the learning rate or step length and $\mathbf{u}$ is a descent direction, which is set to $-\frac{\partial L}{\partial \mathbf{w}^{(t)}}$. The fastText model relies on asynchronous stochastic gradient descent (ASGD) method of optimization following [8], where, several cores can update any component, $\mathbf{w}_j^{(t)}$, several times, but atomically, on the same iteration.

Both ASGD and SGD depend on finding unbiased estimators of gradients. The key difference is that for the former, the gradient computation for parameter estimators is allowed to use stale values of the previously updated parameter iterates. Convergence is shown to happen for certain scenarios, where the loss functions are convex and the stochastic gradients are assumed to be bounded in expectation. In the original ASGD paper [8], this convergence is shown for a fixed step size. The convergence analysis for ASGD in [7] is shown for the non-bounded gradient assumption and a continuously decreasing step size, which is the case for fastText.

It is mentioned in [7], that the permutation of indices for the parameter vector or matrix updates when there are inconsistent reads and writes by several cores is crucial for good performance. In fastText, the compute cores start iterating over the data from fixed positions in the training file and not randomly as in the case of true SGD. This might be a reason why the solution computed by fastText, is not stochastically optimal with regards to the assumptions mentioned in [7].